Formal Modelling and Verification of Hybrid Systems by Hybrid Relation Approach

Huixing Fang*, Jifeng He*, Qiwen Xu† Huibiao Zhu*, Longfei Zhu*
* East China Normal University, Shanghai, China
†University of Macau, Macau SAR, P.R. China

***Abstract*—Hybrid systems arise in real-time and embedded control systems with the interactions emerged between continuous physical environment and discrete digital controllers. In this paper, we propose an approach for the verification of hybrid systems which are constructed by a hybrid parallel modelling language, where the interaction between the controller and the environment is synchronized by signals. The proof rules with respect to the modelling language are illustrated in the style of Hoare triples. For an application of our approach, a water tank model is produced as a sequential hybrid model, then we verify the model with the satisfiability according to a design requirement. Moreover, for parallel hybrid model, a case of subway control system involving overlapping metro lines is presented and verified with respect to the collision avoidance requirement.**

I. Introduction

Hybrid systems ([1], [2]) consist of discrete control programs and continuous physical plants, which combines logical decision making with the evolution of continuous flows. The continuous behaviors of hybrid systems are usually specified by differential equations. While, the discrete actions are represented by assignments. The interactions between different components of the system are declared by shared variables, and synchronization or asynchronization communications. Hybrid systems play important roles in industry, such as transportation networks, automotive, aviation, manufacturing processes, robotics, even medicine and biology. Nowadays, hybrid systems are ubiquitous in safety critical applications. Therefore, formal modelling and verification techniques are necessary to enhance the safety and reliability of hybrid systems ([3], [4], [5]).

In [6], He proposed a clock-based framework for constructing hybrid systems, in this work, the occurrence of an event was represented as a clock. The clock is an abstract concept with metric spaces for describing temporal order and time latency, which links with synchronous events through the representation of occurrences of events by clocks. The changes of discrete variables and continuous components can also be captured by the recording of time instants whenever the proper variations occur. In addition, He presented the novel hybrid relation calculus [7], of which the alphabet of relations consist of continuous and input/output variables. In hybrid relation calculus, both clocks and signals are introduced to coordinate activities of parallel components of hybrid systems. Moreover, for the modelling of physical world and its interaction with the controller, a hybrid modelling language (abbreviated as HML) was introduced in [7], where the interaction between parallel components of a hybrid system is synchronized by signals. The signal is instantaneous reactive and propagated immediately. The components of a system are consistently aware of the *present* or *absent* of signals.

In this paper, we focus on the modelling and verification of HML models. In traditional modelling notations ([8], [9], [10], [11], [12]), the continuous behaviors of physical plants are modelled in differential equations. In HML, the continuous constraint consists of differential equations, initial values of continuous variables, and exit conditions indicating the situation when the continuous flow shall terminate. The communications between components of a system are represented by signals, the corresponding history is stored in a clock structure including the time instants when the signal is present. Thus, signals are the bridges that construct links among various components of a hybrid system. Moreover, the discrete and continuous variables are not shared between parallel components of the system.

In terms of the formal verification framework for hybrid systems, in [13], He and Xu integrated variants Duration Calculus ([14], [15]) in sequential hybrid programs, then proposed a semantic and proof framework for sequential hybrid programs. Based on this work, we extend the proof rules to suit for HML on the parallel compositions of components in hybrid systems. For the verification of HML models, we present the proof rules in the Hoare triple style. These proof rules are inspired by the hybrid relation calculus which was proposed by He in [7]. In hybrid relation calculus, both clock and signal are introduced to coordinate interactions between parallel components of hybrid systems. The continuous variables in a hybrid relation are used to record the continuous behavior of physical environment. The predicates in hybrid relation calculus are employed to specify the behavior of the HML model. The unprimed (undashed) variables are used to describe the states when the program (model) starts to run, while the primed (dashed) variables for the states when the program terminates or be in progress. Moreover, as the importance of signals and clocks in hybrid relation calculus, the relations of the signals and clocks establish the bases for the proofs of parallel programs.

We present two case studies. The first one is a case study of water tank as an HML program. The water tank program is a sequential HML program, it contains two different continuous sub-programs, one for the rising of water level, another for the falling. The property we are interested in this case is the clock relation between the water level and the signal about the valve of the tank. The second case study concerns a subway control system with two overlapping metro lines. We have a subway train for each metro line. During the running of trains, they have to coordinate their positions through a controller as one segment of the track is overlapped (i.e., shared) by the metro lines. This case depicts a parallel composition of three components, including two trains and the corresponding controller. The requirement in this system is the collision avoidance when the two trains are running on the shared track segment. Firstly, this proof is constructed by the relations of the signals between the trains and the controller, then, proceeded by the proofs for the components separately. The construction of proper relations of signals and clocks is the significant procedure for the proof of parallel programs, and this can be explored from the relations in the components of the system independently.

The contribution of this paper consists of the proof rules for HML programs based on hybrid relation calculus and the application of HML on the subway control systems for formal modelling and verification. The reminder of this paper is organized as follows: in Section II, we describe the syntax for HML[1], and the definitions of hybrid relation calculus. Section III is about the design of HML program. Section IV shows the details for the Hoare logic rules of HML based on hybrid relation calculus. In Section V and VI, we introduce two case studies as the applications of our logic rules. We conclude our work and discuss future works in Section VII.

II. The Language and Hybrid Relation

In this section, we present the syntax for HML. In addition, the basic concepts of hybrid relation calculus are proposed.

A. Hybrid Modelling language

In this section, we present the syntax of the hybrid modelling language (HML) as follows:

$$\begin{aligned}
AP &::= \texttt{skip} \mid x := e \mid !s \mid \texttt{suspend}(e)\\
EQ &::= R(v,\dot v) \mid R(v,\dot v)\ \texttt{init}\ v_0 \mid EQ \,||\, EQ\\
P &::= AP \mid P \sqcap P \mid P;P \mid P \,||\, P \mid P \triangleleft b \triangleright P\\
&\quad \mid EQ\ \texttt{until}\ g \mid \texttt{when}(G) \mid \texttt{while}\ b\ \texttt{do}\ P\ \texttt{od}\\
g &::= \epsilon \mid s \mid b \mid \texttt{out}(e) \mid g \odot g \mid g \oplus g\\
b &::= true \mid false \mid v \sim c \mid \neg b \mid b \wedge b \mid b \vee b\\
G &::= g\&P \mid G \,[\!]\, G
\end{aligned}$$

where, $\sim \in \{\geq, >, =, <, \leq\}$. skip is the empty command, the atomic program '$!s$' emits signal s, then terminates immediately. The assignment statement '$x := e$' has the traditional form for variable x and expression e. The program '$\texttt{suspend}(e)$' suspends the current running program for e time units, where e is an expression that can be evaluated to a nonnegative number.

The relation predicate $R(v,\dot v)$ defines a constraint (e.g., differential equation) for variable v and the first-order derivative of v over time. For instance, $\dot v = 1$ and $\dot v = v^2 + v - 5$ are different relation predicates for v and $\dot v$. Moreover, '$R(v,\dot v)\ \texttt{init}\ v_0$' declares a constraint for v and $\dot v$ with v_0 the initial value of variable v. The parallel composition '$EQ \,||\, EQ$' can be utilized to construct the simultaneous differential equations for hybrid systems with more than two continuous variables.

The symbol '$\sqcap$' is used as an operator for non-deterministic choice. For instance, '$P \sqcap P'$' denotes the program that behaves like either P or P'. The program '$P;P$' denotes the sequential composition of sub-programs. The parallel composition of programs is represented by '$P \,||\, P$'. The conditional choice between programs P and Q with boolean expression b is '$P \triangleleft b \triangleright Q$'. If b is true, it behaves like P, and Q otherwise. The differential equation with exit-condition that denotes the termination condition for the continuous evolving is declared by '$EQ\ \texttt{until}\ EC$'. The exit-condition 'EC' can be conditional guards, signals or their combinations.

For example, if g_1 and g_2 are boolean conditions, '$g_1 \oplus g_2$' can be an exit-condition, $(g_1 \oplus g_2) \overset{\text{def}}{=} (g_1 \vee g_2)$. If g_1 and g_2 are signals, '$g_1 \odot g_2$' can be an exit-condition, $g_1 \odot g_2$ iff both g_1 and g_2 are present only at the last time instant of an observation duration, the detailed definition can be found in Section II-B. The empty guard 'ϵ' is valid (ready) immediately when it is evaluated. The signal guard 's' is valid when s is present. The boolean guard b is used as boolean expression. The timeout guard '$\texttt{out}(e)$' is valid when the current time $t' = t + e$ after the program is started at time t, expression e is evaluated as a non-negative number (for time units).

The when-choice program '$\texttt{when}(G)$' activates program P_i when the corresponding guard g_i is valid. Otherwise when all guards are invalid, the when-choice program will keep waiting until a valid one exits. For instance, the when-choice program '$\texttt{when}(g_1\&P_1 \,[\!]\, g_2\&P_2)$' will wait for one of the guards g_1 and g_2 to be valid. If at one moment g_1 is valid, then the program will behave like P_1. If both g_1 and g_2 are valid at the same time, the program will be performed as $P_1 \sqcap P_2$. The loop program '$\texttt{while}\ b\ \texttt{do}\ P\ \texttt{od}$' is simple, where b is the condition, P is the loop body.

B. Clock and Signal

In this section, we define the clock and signal used in this paper. We use $[t, t']$ to specify the time interval over which the observation of the behavior of an HML program is concerned. The start point t denotes the time instant when the program starts to run, the end point t' is for the time instant when the program terminates or in a waiting status.

Definition 1 (Clock): A clock c is an increasing sequence of non-negative reals. The set of all elements of c is denoted by $\mathbf{Set}(c)$.

$$c : \mathbb{N}^* \to \mathbb{R}_{\geq 0}$$

where, $\mathbb{N}^*$ is $\mathbb{N}$ or $\{n \in \mathbb{N} \mid n < M\}$ for an $M \in \mathbb{N}$. For any natural number $n \in \mathbb{N}^*$, the n-th element of c is represented by $c[n]$. As c is an increasing sequence, we have $c[n] \leq c[n+1]$ for $n \in \mathbb{N}^*, c[n], c[n+1] \in \mathbf{Set}(c)$.

For example, a simple clock c may be the sequence $\langle 0, 10, 21\rangle$, and $c[0] = 0, c[1] = 10, c[2] = 21$. The cardinal number (i.e., size) of c is defined by $|c| \overset{\text{def}}{=} |\mathbf{Set}(c)| = |\mathbb{N}^*|$. In this simple example $|c| = 3$. If $\mathbb{N}^* = \mathbb{N}$ then $|c| = \aleph_0$, otherwise if $\mathbb{N}^* = \{n \in \mathbb{N} \mid n < M\}$ for an $M \in \mathbb{N}$, then $|c| = M$.

A clock c runs faster than clock d if for all $i \in \mathbb{N}$, $c[i] \leq d[i]$, we denote this by $c \preceq d$. For example, clock $c_1 \overset{\text{def}}{=} \langle 0, 1, 1.4\rangle$ is faster than clock $c_2 \overset{\text{def}}{=} \langle 2, 4.3, 18\rangle$. For clocks c and d, we define a relation $\preceq_t$ between two clocks, that $c \preceq_t d$ if $c \preceq d$ and for all $i \in \mathbb{N}, c[i] \leq t, d[i] \leq t$.

Definition 2 (Count-Clock): A count-clock c is an increasing sequence,

$$c : \mathbb{N}^* \to (\mathbb{R}_{\geq 0} \times \mathbb{N}),$$

in which, the element in c is appended with a natural number denoting the value of a variable *count* indicating the receiving

[1]The formal semantics will be presented in another paper.

order when signals are present. The faster between clocks is also defined w.r.t the real numbers ($\mathbb{R}_{\geq 0}$).

In this paper, the clock we are used is the count-clock. And, the clock c is represented by the set of elements in c. Now, we define some special clocks on some conditions as follows, for example, let h, H, L and ε be some variables (c.f., Table I in the water tank case study), Fig. 1 illustrates the following clocks,

$climb(h, H-\varepsilon) \overset{\text{def}}{=} \{(time, count) \mid (h \geq H-\varepsilon)_{time} \wedge (h < H-\varepsilon)_{time-0}\}$;

$drop(h, L+\varepsilon) \overset{\text{def}}{=} \{(time, count) \mid (h \leq L+\varepsilon)_{time} \wedge (h > L+\varepsilon)_{time-0}\}$,

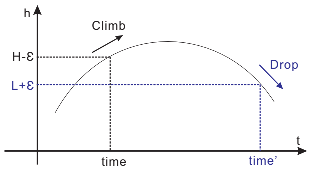

Fig. 1: Clocks of climb and drop, $time \in \pi_1(climb(h, H-\varepsilon))$, $time' \in \pi_1(drop(h, L+\varepsilon))$. π_1 is a function mapping from a clock to the corresponding set of time instants

where, if b is a condition expression, $(b)_{time}$ is the boolean value of b at the time point $time$, $(b)_{time-0} \overset{\text{def}}{=} \lim_{\delta \to 0} b(time-\delta)$, $b(time-\delta)$ denotes the value (*false* or *true*) of condition b at the time instant $(time-\delta)$. Clock $climb(h, H-\varepsilon)$ records the time instants when the condition $(h \geq H-\varepsilon)$ becomes true, while clock $drop(h, L+\varepsilon)$ records the time instants when the condition $(h \leq L+\varepsilon)$ becomes true. The value of $count$ records the index of the latest input signal, it can be used to identify the dependency between that input and the subsequent output signals.

In communication systems, signal processing and electrical engineering, signal is a function that conveys information about the behavior of a system or the environment, or about the property of some phenomenons [16]. In the HML syntax, we introduced signals. Each signal has two statuses, either presence or absence, we use $present(s)$ (abbreviated as $\xi(s)$) and $absent(s)$ (abbreviated as $\theta(s)$) to denote whether the signal s is in the status of presence or absence, respectively. Each signal s is accompanied by a clock variable $s.clock$ recording the time instants whenever s is in the status of presence. The corresponding new clock variable after the execution of a program is $s.clock'$. The predicates $\xi(s)$ and $\theta(s)$ can be defined as follows,

$$\xi(s) \overset{\text{def}}{=} ((\pi_1(s.clock') \cap [t, t']) = \{t'\});$$

$$\theta(s) \overset{\text{def}}{=} ((\pi_1(s.clock') \cap [t, t']) = \emptyset),$$

where, both $\xi(s)$ and $\theta(s)$ take the values of $true$ and $false$. For operators '$\oplus$' and '$\odot$', we have $\xi(s_1 \oplus \ldots \oplus s_n) \overset{\text{def}}{=} \left(\pi_1(\bigcup_{i \in I} s_i.clock') \cap [t, t'] = \{t'\}\right)$, $\xi(s_1 \odot \ldots \odot s_n) \overset{\text{def}}{=} \left(\pi_1(\bigcap_{i \in I} s_i.clock') \cap [t, t'] = \{t'\}\right)$, where, $I \overset{\text{def}}{=} \{i \mid i \in \mathbb{N}, 1 \leq i \leq n, n \in \mathbb{N}_{\geq 1}\}$. $[t, t']$ indicates a time interval, π_1 is a function mapping from a clock to the corresponding set of time instants. Sometimes, π_1 can be used to mapping a clock element to the first member of the element, i.e., $time$, and π_2 for the second member, i.e., $count$. An HML program can emit a signal as '$!s$' or receive (wait for) a signal from the environment as in the differential statement 'EQ `until` s'. In the following, we use $InSignal$ to denote the set of the input signal names of an HML program, and $OutSignal$ for output signal names. For boolean conditions test, let b ranged over boolean conditons, $b(\cdot)$ evaluates the boolean value of b at a time instant, we define the present and absent for tests as follows: $\xi(b) \overset{\text{def}}{=} b(t') \wedge \forall \tau \in [t, t') \bullet \neg b(\tau)$, $\theta(b) \overset{\text{def}}{=} \forall \tau \in [t, t'] \bullet \neg b(\tau)$, for operators '$\odot$' and '$\oplus$': $\xi(b_1 \odot \ldots \odot b_n) \overset{\text{def}}{=} \xi(\bigwedge_{i \in I} b_i), \xi(b_1 \oplus \ldots \oplus b_n) \overset{\text{def}}{=} \xi(\bigvee_{i \in I} b_i)$, for $I \overset{\text{def}}{=} \{i \mid i \in \mathbb{N}, 1 \leq i \leq n, n \in \mathbb{N}_{\geq 1}\}$.

C. Hybrid Relation Calculus

For a program of HML, the relevant observations consist of the values of all variables before the program execution, their values after termination, and the intermediate state before program termination. Such observations therefore constitute a *hybrid relation*. The corresponding predicate in the hybrid relation describes the behavior of an HML program.

For an HML program, two status variables st and st' are employed to describe the program status. Variable st is used before the program starts, and st' after it starts. Status variables st and st' are ranged over $\{term, stable, div\}$.

1) When $st = term$, the predecessor of the program terminates successfully. While, if $st' = term$, the program terminates.
2) When $st = stable$, the predecessor is not terminated and waiting for input signals. While, if $st' = stable$, the program is waiting for input signals.
3) When $st = div$, the predecessor is divergent (or called chaotic). While, if $st' = div$, the program is divergent.

Moreover, $st \in in\alpha P$, $st' \in out\alpha P$ for any HML program P. Where, $in\alpha P$ is the set of input variables for P, and $out\alpha P$ for output variables.

Example 1 (Divergent HML Program): One very simple program which is divergent: $DIV \overset{\text{def}}{=}$ (`while` $true$ `do` $x := x+1$; `od`). The program DIV is an infinite loop consisting of an always valid condition and one assignment statement for variable x that increases the value by 1 in every iteration. As the program could neither wait for a signal nor terminate, the status of the program can only be div indicating divergence.

Definition 3 (Hybrid Relation): For predicate P, a hybrid relation is a binary relation (P, αP). The alphabet of P is αP:

$$\alpha P \overset{\text{def}}{=} in\alpha P \cup con\alpha P \cup out\alpha P$$

where, $in\alpha P$ and $out\alpha P$ are the sets of undashed (i.e., unprimed) and dashed (i.e., primed) variables for indicating initial and final values of variables, respectively. The set $con\alpha P$ consists of continuous variables that are changed continuously according to differential equations (or continuous mathematical functions w.r.t time).

In HML, we have $out\alpha P = in\alpha' P$. The undashed variables are represented by $in\alpha P \overset{\text{def}}{=} \{st, t, count\} \cup PVar \cup ClockVar$. In which, st is a variable indicating the status of the program, t is the variable recording the starting time point of an observation interval. $PVar$ is the set of discrete variables, $ClockVar \overset{\text{def}}{=} \{s.clock \mid s \in InSignal \uplus OutSignal\}$ is the set of clock variables. Let P and Q be hybrid relations, if $\alpha P = \alpha Q$, $out\alpha P = \{v' \mid v \in in\alpha Q\}$, then $(P; Q) \overset{\text{def}}{=} \exists o \bullet P[o/v'] \wedge Q[o/v]$.

For predicate P, we employ $(P, \alpha P)_{pp}$ as a pure-primed hybrid relation, where $\alpha P \overset{\text{def}}{=} con\alpha P \cup out\alpha P$. Likewise, we employ $(P, \alpha P)_{pu}$ as a pure-unprimed hybrid relation, the alphabet $\alpha P \overset{\text{def}}{=} con\alpha P \cup in\alpha P$. In HML programs, let S_{hr}, S_{pp} and S_{pu} be the sets of predicates for hybrid relations, pure-primed hybrid relations, and pure-unprimed hybrid relations, respectively, then we have: $S_{pp} \subset S_{hr}$, $S_{pu} \subset S_{hr}$, and $S_{pp} \cap S_{pu} = \emptyset$.

Example 2 (Hybrid Relation of Differential Equation): A differential equation $EQ \overset{\text{def}}{=} (\dot{v} = 1)$ can be seen as a hybrid relation, where $in\alpha \overset{\text{def}}{=} \{t\}, out\alpha \overset{\text{def}}{=} \{t'\}, con\alpha \overset{\text{def}}{=} \{v\}, P \overset{\text{def}}{=} (t \leq t') \wedge \forall \tau \in [t, t') \bullet \dot{v}(\tau) = 1$. The input variable t represents the starting time instant, the output variable t' denotes the time instant during the continuous flow. The continuous variable is v. The predicate of the relation is referred by P which indicates the first-order derivative of v is always 1 for the time interval $[t, t')$. The right-open interval makes sense when the left- and right- derivatives of continuous variable v are not equal.

III. Design of HML Program

In this section, we introduce the concept called *design* for the behavior of HML programs. A design for an HML program P is about the predicates of assumption for P that we can known when P starts to run, and the predicates of commitment which is guaranteed when P is terminated or during the executing of the program. HML programs have some implied assumptions, such as the time is increasing during the execution of the program, which can be expressed as $t \leq t'$. Another assumption is about the clock of a signal, the clock may be appended with new elements if the signal is present during an observation time interval $[t, t']$. Therefore, an HML program P has to meet the following two healthiness conditions $\mathbf{HC_1}$ and $\mathbf{HC_2}$, where, s represents any signal,

$$\mathbf{HC1}: P = P \wedge t \leq t', \quad \mathbf{H_1}(P) \overset{\text{def}}{=} P \wedge t \leq t';$$

$$\mathbf{HC2}: P = P \wedge inv(s), \quad \mathbf{H_2}(P) \overset{\text{def}}{=} P \wedge inv(s),$$

where, $inv(s) \overset{\text{def}}{=} (s.clock \subseteq s.clock') \wedge (\pi_1(s.clock') \subseteq (\pi_1(s.clock) \cup [t, t']))$. We use the notation $\mathbf{H_{12}}$ to stand for the composition of functions $\mathbf{H_1}$ and $\mathbf{H_2}$, i.e., $\mathbf{H_{12}} \overset{\text{def}}{=} \mathbf{H_1} \circ \mathbf{H_2}$. We define a special predicate as $\mathbf{H_{12}}(\perp) \overset{\text{def}}{=} (t \leq t') \wedge inv(s)$.

The predicate for empty program, $\texttt{skip} \overset{\text{def}}{=} II_A \triangleleft st \neq div \triangleright \mathbf{H_{12}}(\perp)$, where, II_A denotes that the program does not change any input variables in the input alphabet A when the previous program is not in the divergent status. Otherwise, only the healthiness conditions are respected and we can not guarantee any more.

Definition 4 (Healthiness Mapping-$\mathbf{H}$): For $P \in S_{hr}$, we define a mapping,

$$\mathbf{H}(P) \overset{\text{def}}{=} \mathbf{H_{12}}(P; \texttt{skip}) \triangleleft st = term \triangleright \texttt{skip}.$$

The function $\mathbf{H}$ maps a hybrid relation P to a healthiness one that behaves as the sequential composition of P and $\texttt{skip}$ under the healthiness conditions $\mathbf{HC1}$ and $\mathbf{HC2}$ when the previous program is terminated. Otherwise, it maps P to $\texttt{skip}$.

Definition 5 (Design): Let $\beta \in S_{hr}$ be a precondition, $\varsigma \in S_{hr}$ and $\tau \in S_{hr}$ be stable condition and postcondition (termination condition), respectively. In addition, β, ς and τ do not contain st and st'. The design for a program P can be defined as follow,

$$[\beta \vdash_{\mathbf{H}} \langle \varsigma \bowtie \tau \rangle] \overset{\text{def}}{=} \mathbf{H}[(st = term \wedge \beta) \Rightarrow \langle \varsigma \bowtie \tau \rangle],$$

in which, $\langle \varsigma \bowtie \tau \rangle^2 \overset{\text{def}}{=} (\varsigma \wedge st' = stable) \vee (\tau \wedge st' = term)$. The intuitive meaning of the design is that if P's previous program is terminated, then under the condition β, the program P will not diverge and its all intermediate states satisfy ς while all terminated states satisfy τ.

Definition 6 (Invariant-Design): Let $\alpha \in S_{hr}$ be an invariant, Also, α does not contain st and st'. Assume $[\beta \vdash_{\mathbf{H}} \langle \varsigma \bowtie \tau \rangle]$ be a design. The invariant-design for a program P can be defined as follow,

$$[\alpha : (\beta \vdash_{\mathbf{H}} \langle \varsigma \bowtie \tau \rangle)] \overset{\text{def}}{=} [\beta \vdash_{\mathbf{H}} \langle (\alpha \wedge \varsigma) \bowtie (\alpha \wedge \tau) \rangle],$$

the invariant α is respected from the starting to the termination of P. We denote by $[\alpha : \{\beta\} \ P \ \{\langle \varsigma \bowtie \tau \rangle\}]$ that an HML program P implements an invariant-design $[\alpha : (\beta \vdash_{\mathbf{H}} \langle \varsigma \bowtie \tau \rangle)]$.

Example 3 (Invariant-Design of HML Program): Consider a simple assignment program $x := 1$, then we have $[x' \geq 0 : \{x = 0\} \ x := 1 \ \{\langle false \bowtie x' = 1 \rangle\}]$ which indicates that $x := 1$ implements a design $[x' \geq 0 : (x = 0 \vdash_{\mathbf{H}} \langle false \bowtie x' = 1 \rangle)]$. On the contrary, $x := 1$ cannot implement the design $[x' = 0 : (x = 0 \vdash_{\mathbf{H}} \langle false \bowtie x' \geq 0 \rangle)]$, because the invariant $x' = 0$ is unsatisfiable when the assignment is terminated.

Theorem 1 (Design Closure): We have the following laws for design,

$$\begin{aligned}
1)\ & [\beta_1 \vdash_{\mathbf{H}} \langle \varsigma_1 \bowtie \tau_1 \rangle]; [\beta_2 \vdash_{\mathbf{H}} \langle \varsigma_2 \bowtie \tau_2 \rangle] \\
& = [\neg(\neg\beta_1; \mathbf{H_{12}}(\perp)) \wedge \neg(\tau_1; \neg\beta_2) \\
& \quad \vdash_{\mathbf{H}} \langle (\varsigma_1 \vee (\tau_1; \varsigma_2)) \bowtie (\tau_1; \tau_2) \rangle], \\
2)\ & [\beta_1 \vdash_{\mathbf{H}} \langle \varsigma_1 \bowtie \tau_1 \rangle] \triangleleft b \triangleright [\beta_2 \vdash_{\mathbf{H}} \langle \varsigma_2 \bowtie \tau_2 \rangle] \\
& = [(\beta_1 \triangleleft b \triangleright \beta_2) \vdash_{\mathbf{H}} \langle (\varsigma_1 \triangleleft b \triangleright \varsigma_2) \bowtie (\tau_1 \triangleleft b \triangleright \tau_2) \rangle], \\
3)\ & [\beta_1 \vdash_{\mathbf{H}} \langle \varsigma_1 \bowtie \tau_1 \rangle] \vee [\beta_2 \vdash_{\mathbf{H}} \langle \varsigma_2 \bowtie \tau_2 \rangle] \\
& = [(\beta_1 \wedge \beta_2) \vdash_{\mathbf{H}} \langle (\varsigma_1 \vee \varsigma_2) \bowtie (\tau_1 \vee \tau_2) \rangle], \\
4)\ & [\beta_1 \vdash_{\mathbf{H}} \langle \varsigma_1 \bowtie \tau_1 \rangle] \wedge [\beta_2 \vdash_{\mathbf{H}} \langle \varsigma_2 \bowtie \tau_2 \rangle] \\
& = [(\beta_1 \vee \beta_2) \vdash_{\mathbf{H}} \\
& \quad \langle (\beta_1 \Rightarrow \varsigma_1 \wedge \beta_2 \Rightarrow \varsigma_2) \bowtie (\beta_1 \Rightarrow \tau_1 \wedge \beta_2 \Rightarrow \tau_2) \rangle].
\end{aligned}$$

The above theory provides a simple calculus for designs and shows that they are closed under sequential composition, conditional choice, disjunction and conjunction.

For an HML program $(P_1; P_2)$, if the sub-program P_1 implements design $[\beta_1 \vdash_{\mathbf{H}} \langle \varsigma_1 \bowtie \tau_1 \rangle]$, P_2 implements design $[\beta_2 \vdash_{\mathbf{H}} \langle \varsigma_2 \bowtie \tau_2 \rangle]$, then $(P_1; P_2)$ implements design $[\neg(\neg\beta_1; \mathbf{H_{12}}(\perp)) \wedge \neg(\tau_1; \neg\beta_2) \vdash_{\mathbf{H}} \langle (\varsigma_1 \vee (\tau_1; \varsigma_2)) \bowtie (\tau_1; \tau_2) \rangle]$.

The divergent (chaotic) behavior for $(P_1; P_2)$ can be divided into two cases. One for P_1 is $(\neg\beta_1; \mathbf{H_{12}}(\perp))$ which denotes the divergent behavior of P_1 followed by the healthy condition. The condition indicates that the time can only enlarge and also for clocks; another one is $(\tau_1; \neg\beta_2)$ for P_2,

[2]The operator $\bowtie$ has the lowest precedence among traditional logical operators

which means the terminating behavior of P_1 followed by the divergent behavior of P_2.

The intermediate behavior of $(P_1; P_2)$ is the intermediate behavior of P_1 represented by ς_1 or when P_1 terminates then $(\tau_1; \varsigma_2)$, the terminating behavior of P_1 followed by the intermediate behavior of P_2. The terminating behavior of the whole program $(P_1; P_2)$ can be represented by the sequential composition of the terminating behaviors of P_1 and P_2.

For an HML program $(P_1 \lhd b \rhd P_2)$, if sub-program P_1 implements design $[\beta_1 \vdash_{\mathbf{H}} \langle \varsigma_1 \bowtie \tau_1 \rangle]$ when b is evaluated to *true*, P_2 implements design $[\beta_2 \vdash_{\mathbf{H}} \langle \varsigma_2 \bowtie \tau_2 \rangle]$ when b is *false*, then $(P_1 \lhd b \rhd P_2)$ implements the design $[(\beta_1 \lhd b \rhd \beta_2) \vdash_{\mathbf{H}} \langle (\varsigma_1 \lhd b \rhd \varsigma_2) \bowtie (\tau_1 \lhd b \rhd \tau_2) \rangle]$.

For an HML program $(P_1 \sqcap P_2)$, non-deterministic choice between P_1 and P_2. As we can not predict which one will be activated, therefore, both β_1 and β_2 have to be valid. But, when one sub-program is selected, the whole program would behave as the sub-program.

The last law in the theory is for an HML program P which implements both $[\beta_1 \vdash_{\mathbf{H}} \langle \varsigma_1 \bowtie \tau_1 \rangle]$ and $[\beta_2 \vdash_{\mathbf{H}} \langle \varsigma_2 \bowtie \tau_2 \rangle]$. But β_1 and β_2 may conflict at the same time instant when the program starts to run, therefore when β_1 is valid then the behaviors of the program are ς_1 and τ_1, if β_2 is valid then ς_2 and τ_2. If there is no conflict for β_1 and β_2, then the behaviors of P are $\varsigma_1 \wedge \varsigma_1$ and $\tau_1 \wedge \tau_2$ for intermediate and termination of P, respectively.

IV. Hoare Logic for HML

In this section, we present the Hoare rules for discrete, continuous and composition statements for HML programs.

A. Discrete Relation Rules

The rule for assignment program of discrete variable x with expression e is represented as follow:

$$[\mathrm{SA}]\ \frac{}{\mathtt{T} : \{\beta\}\ [x := e]\ \{\langle \mathtt{F} \bowtie II_A[e/x] \rangle\}}$$

where, the name of the rule is SA, the predicates of *true* and *false* are represented by T and F. The set $A \overset{\text{def}}{=} in\alpha(x := e)$, denoting all the input variables of the program. The assertion T is the weakest predicate because *true* can be satisfied all the time. The assertion F is the strongest predicate that no state can satisfy *false*. The intuitive meaning of F is that the intermediate state of a discrete action is unobservable (or designed to be ignored in our logic). The assertion $II_A[e/x]$ represents that all variables are unchanged except for the variable x whose new value referred by the primed variable x' is equal to e.

The emitting of signal s is also a discrete action, the rule for signal-emitting with the rule name SE is represented as follow:

$$[\mathrm{SE}]\ \frac{}{\mathtt{T} : \{\beta\}\ [!s]\ \{\langle \mathtt{F} \bowtie II_A[(s.clock \cup (t, count))/s.clock] \rangle\}}$$

where, the set $A \overset{\text{def}}{=} in\alpha(!s)$. The new value of the clock (i.e., $s.clock$) related to the signal s is appended with the time instant when the signal is emitted and the value of count.

The program skip doses not update any input variable as the assertion II_A represented. This is indicated as follow by the rule SK:

$$[\mathrm{SK}]\ \frac{}{\mathtt{T} : \{\beta\}\ [\mathtt{skip}]\ \{\langle \mathtt{F} \bowtie II_A \rangle\}}$$

where, $II_A \overset{\text{def}}{=} \bigwedge_{x \in A} (x' = x)$.

B. Continuous Relation Rules

The suspend of e units of time is implemented by the program $\mathtt{suspend}(e)$ which does nothing but let the time passing for e units. The rule with name SD for $\mathtt{suspend}(e)$ is represented as follow:

$$[\mathrm{SD}]\ \frac{}{\mathtt{T} : \{\beta\}\ [\mathtt{suspend}(e)]\ \{II_B \wedge \langle t' \in [t, t+e) \bowtie t' = t+e \rangle\}}$$

where, $B \overset{\text{def}}{=} A \backslash (\{t\} \cup \{st\} \cup \{s.clock \mid s \in InSignal\})$ for A the set of all input variables of the program. The assertion of the form $\beta \wedge \langle \varsigma \bowtie \tau \rangle$ is the abbreviation for the assertion $\langle \beta \wedge \varsigma \bowtie \beta \wedge \tau \rangle$. The assertion $t' \in [t, t+e)$ denotes that before the program terminates the new value of time referred by variable x' is evaluated in the left-close and right-open interval $[t, t+e)$. While the new value of time is equal to $(t+e)$ when the program terminates.

For differential equation EQ with exit condition g, we have the following rule with name EQ:

$$[\mathrm{EQ}]\ \frac{}{\begin{array}{l}\mathtt{T} : \{\beta\}[EQ\ \mathtt{until}\ g] \\ \{P \wedge \langle \theta(g) \wedge count' = count \bowtie \xi(g) \wedge count' > \lambda \rangle\}\end{array}}$$

where, $\lambda \overset{\text{def}}{=} \mathtt{max}(\{0, count\} \cup \{\pi_2(last(s.clock')) \mid s \in g\})$. The assertion $\theta(g)$ denotes that g is absent during the continuous evolution of the differential equation. The exit condition g is present when the differential equation terminates. The new value of count referred by $count'$ is greater than the maximum value among all the clocks related to g and the old value of count. The expression $\pi_2(last(s.clock'))$ evaluates the count value in the last element of the new value of $s.clcok$. The assertion $P \overset{\text{def}}{=} (II_B \wedge \forall v \in [t, t') \bullet R(v))$ for $B \overset{\text{def}}{=} PVar \cup \{s.clock \mid s \in OutSignal\}$. For example if the differential equation is $EQ \overset{\text{def}}{=} (\dot{h} = 1)$, then the predicate $\forall v \in [t, t') \bullet (\dot{h}(v) = 1)$ in the postcondition can be explained that the first-order derivative of h is equal to 1 for all the proper time instants after the program started.

C. General Relation Rules

The consequence rule for program S with name CQ is presented as follow:

$$[\mathrm{CQ}]\ \frac{\beta \Rightarrow \iota, \omega : \{\iota\}[S]\{\langle \mu \bowtie \nu \rangle\}, \iota \wedge \omega \wedge \mu \Rightarrow \alpha \wedge \varsigma, \iota \wedge \omega \wedge \nu \Rightarrow \alpha \wedge \tau}{\alpha : \{\beta\}\ [S]\ \{\langle \varsigma \bowtie \tau \rangle\}}$$

where, the precondition (β) is strengthened. The postconditions $(\beta \wedge \alpha \wedge \varsigma)$ and $(\beta \wedge \alpha \wedge \tau)$ are weakened,respectively.

For sequential composition of programs S_1 and S_2 with $out\alpha(S_1) = \{x' \mid x \in in\alpha(S_2)\}$, the rule with name SC is presented as follow:

$$[\mathrm{SC}]\ \frac{\begin{array}{c}\varsigma_1 \vee (\gamma_1; \varsigma_2) \Rightarrow \varsigma,\ (\gamma_1; \tau_2) \Rightarrow \tau,\ \alpha_1 \vee (\alpha_1; \alpha_2) \Rightarrow \alpha \\ \alpha_1 : \{\beta\}\ [S_1]\ \{\langle \varsigma_1 \bowtie \gamma_1 \wedge \varepsilon \rangle\},\ \alpha_2 : \{\varepsilon[x/x']\}\ [S_2]\ \{\langle \varsigma_2 \bowtie \tau_2 \rangle\}\end{array}}{\alpha : \{\beta\}\ [S_1; S_2]\ \{\langle \varsigma \bowtie \tau \rangle\}}$$

where, the assertion $(\gamma_1 \wedge \varepsilon)$ is satisfied when program S_1 terminates. The predicate ε is a pure-primed hybrid relation predicate, all the variables in ε, i.e., $\alpha(\varepsilon)$ is the set $out\alpha(S_1) \cup con\alpha(S_1)$ without containing unprimed input variables. Thus, $\varepsilon[x/x']$ is the predicate after the substitution of x (input variable of S_2) for each free occurrence of variable x' (output variable of S_1) in ε. For instance, if $\varepsilon \overset{\text{def}}{=} (x' \geq 1)$, then $\varepsilon[x/x'] \overset{\text{def}}{=} (x \geq 1)$.

The while-loop program is a program with loop-condition b and loop-body S, the rule for loop is presented as follow:

$$[\mathrm{LP}]\ \frac{\alpha;\alpha \Rightarrow \alpha,\ \ \alpha : \{b \wedge \beta\}\ [S]\ \{\langle \varsigma \bowtie \beta[x'/x]\rangle\}}{\alpha : \{\beta\}\ [\texttt{while}\ \ b\ \ \texttt{do}\ \ S\ \ \texttt{od}]\ \{\langle \varsigma \bowtie \neg b[x'/x] \wedge \beta[x'/x]\rangle\}}$$

where, $b[x'/x]$ is the predicate after the substitution of x' (ranged over output variables of the program) for each free occurrence of variable x (input variables) in b.

The implicit-condition rule is used for recording time instants whenever some implicit conditions (not waited explicitly as guards or exit conditions in the program) are present during the execution of a program.

$$[\mathrm{IC}]\ \frac{\neg E(\tau - 0) \wedge E(\tau)}{(\tau, count) \in E.clock'}$$

where, $E(\tau)$ is true whenever E occurs at the time-point τ. For example, if $E \stackrel{\text{def}}{=} (h \geq H - \varepsilon)$, then, $E.clock \stackrel{\text{def}}{=} climb(h, H - \varepsilon)$ (c.f., Section II-B).

The program $\texttt{when}(g_1\&P_1 [\!] \cdots [\!] g_n\&P_n)$ activates program P_i when the corresponding guard g_i is present.

$$[\mathrm{WP}]\ \frac{\omega : \{\nu[x/x']\}\ [\sqcap_{i\in I'} P_i]\ \{\langle \zeta \bowtie \eta\rangle\},\ \ (\gamma;\omega) \vee \gamma \vee \varsigma \Rightarrow \alpha}{\alpha : \{\beta\}\ [\texttt{when}(g_1\&P_1 [\!] \cdots [\!] g_n\&P_n)]\ \{\langle \varsigma; \zeta \bowtie (\gamma \wedge \nu); \eta\rangle\}}$$

where, $A \stackrel{\text{def}}{=} PVar \cup \{s.clock \mid s \in OutSignal\}$, $I \stackrel{\text{def}}{=} \{1, \cdots, n\}$, and, $I' \subseteq I$, $\lambda \stackrel{\text{def}}{=} \texttt{max}(\{0, count\} \cup \{\pi_2(last(s.clock')) \mid s \in g_i, i \in I'\})$, $\varsigma \stackrel{\text{def}}{=} II_A \wedge count' = count \wedge \forall i \in I \bullet \theta(g_i)$, $\gamma \stackrel{\text{def}}{=} II_A \wedge \forall i \in I' \bullet \xi(g_i) \wedge count' > \lambda$. The variable x ranged over all the input variables for program $\sqcap_{i \in I'} P_i$, the non-deterministic choice between P_is. The rule for non-deterministic choice of programs P_1 and P_2 is presented as follow:

$$[\mathrm{NC}]\ \frac{\alpha : \{\beta\}\ [P_1]\ \{\langle \varsigma \bowtie \tau\rangle\},\ \ \alpha : \{\beta\}\ [P_2]\ \{\langle \varsigma \bowtie \tau\rangle\}}{\alpha : \{\beta\}\ [P_1 \sqcap P_2]\ \{\langle \varsigma \bowtie \tau\rangle\}}$$

where, the proof is reduced to proofs on P_1 and P_2, respectively.

The rule for condition choice program with condition b and two branches S_1 and S_2 is presented as follow:

$$[\mathrm{CC}]\ \frac{\alpha : \{\beta \wedge b\}\ [S_1]\ \{\langle \varsigma \bowtie \tau\rangle\},\ \ \alpha : \{\beta \wedge \neg b\}\ [S_2]\ \{\langle \varsigma \bowtie \tau\rangle\}}{\alpha : \{\beta\}\ [S_1 \triangleleft b \triangleright S_2]\ \{\langle \varsigma \bowtie \tau\rangle\}}$$

where, the left branch S_1 is the program activated when condition b is evaluated to be true, otherwise the right branch S_2 would be activated. Thus the precondition of S_1 contains the condition b, while $\neg b$ is for the right branch S_2.

The rule for parallel composition of programs P_1 and P_2 is present as follow:

$$[\mathrm{PC}]\ \frac{\begin{array}{l}\alpha_1 \wedge \alpha_2 \Rightarrow \alpha,\\ \alpha_1 : \{\beta_1\}\ [P_1]\ \{\langle \varsigma_1 \bowtie \tau_1\rangle\}, \alpha_2 : \{\beta_2\}\ [P_2]\ \{\langle \varsigma_2 \bowtie \tau_2\rangle\}\end{array}}{\alpha : \{\beta_1 \wedge \beta_2\}\ [P_1 \mid\mid P_2]\ \{\langle \mathcal{M}_{stb} \bowtie \mathcal{M}_{ter}\rangle\}}$$

where, $\mathcal{M}_{stb} \stackrel{\text{def}}{=} \mathcal{M}_{\varsigma_1,\varsigma_2} \vee \mathcal{M}_{\varsigma_1,(\tau_2;\delta_2)} \vee \mathcal{M}_{(\tau_1;\delta_1),\varsigma_2}$, $\mathcal{M}_{ter} \stackrel{\text{def}}{=} \mathcal{M}_{(\tau_1;\delta_1),\tau_2} \vee \mathcal{M}_{\tau_1,(\tau_2;\delta_2)}$, for $i = 1, 2$, $\delta_i \stackrel{\text{def}}{=} \mathbf{H_{12}}(II_{A_i})$, $A_i \stackrel{\text{def}}{=} PVar(P_i) \cup \{s.clock \mid s \in OutSignal(P_i)\}$. $\mathcal{M} \stackrel{\text{def}}{=} (count' : \mathbb{N}, \texttt{max})$. The merge mechanism $\mathcal{M}$ is defined as follow. A merge mechanism $\mathcal{M}$ defines an operation on variables.

$$\mathcal{M} \stackrel{\text{def}}{=} (x : \mathcal{T}, \star),$$

where, x is a variable of type $\mathcal{T}$, $\star$ is a binary operator over values of type $\mathcal{T}$, also $\star$ can be considered as a function with two parameters. For instance, if $\star$ is a maximum function $\texttt{max}$ that returns the maximum value of two parameters, let $\mathbb{R}$ be the type for real numbers, we define $\mathcal{M} \stackrel{\text{def}}{=} (x' : \mathbb{R}, \texttt{max})$, and $x' \in out\alpha P \cup out\alpha Q$, then $P \mid\mid_{\mathcal{M}} Q \stackrel{\text{def}}{=} \exists m, n \bullet (P[m/x'] \wedge Q[n/x'] \wedge x' = \texttt{max}(m, n))$. (For simplicity, we employ $\mathcal{M}_{P,Q}$ as an abbreviation of $P \mid\mid_{\mathcal{M}} Q$)

Example 4 (Rule-SC): Let $S_1 \stackrel{\text{def}}{=} (x := x - 1)$, $S_2 \stackrel{\text{def}}{=} (x := x + 1)$, $\alpha \stackrel{\text{def}}{=} (x' < x + 1)$, $\alpha_1 \stackrel{\text{def}}{=} (x' = x - 1)$, $\alpha_2 \stackrel{\text{def}}{=} (x' = x + 1)$, we have the following proof:

$$\frac{\begin{array}{l}\texttt{F} \vee ((x' = x - 1); \texttt{F}) \Rightarrow \varsigma,\\ ((x' = x - 1); (x' = x + 1)) \Rightarrow (x' = x),\\ (x' = x - 1) \vee (x' = x) \Rightarrow \alpha,\\ \alpha_1 : \{x = 0\}\ [S_1]\ \{\langle \texttt{F} \bowtie (x' = x - 1) \wedge (x' = -1)\rangle\},\\ \alpha_2 : \{x = -1\}\ [S_2]\ \{\langle \texttt{F} \bowtie (x' = x + 1)\rangle\}\end{array}}{\alpha : \{x = 0\}\ [S_1; S_2]\ \{\langle \varsigma \bowtie (x' = x)\rangle\}}$$

where, $\alpha_1 \Rightarrow \alpha$ because $(x' = x - 1) \Rightarrow (x' < x + 1)$. And, $\alpha_1; \alpha_2 \Rightarrow \alpha$ as $(x' = x) \equiv (\alpha_1; \alpha_2)$, $(x' = x) \Rightarrow (x' < x + 1)$. The assertion $\gamma_1 \wedge \varepsilon$ is instantiated by $(x' = x - 1) \wedge (x' = -1)$. The precondition $\varepsilon[x/x']$ of S_2 is instantiated by $x = -1$. The postcondition τ_2 of S_2 is instantiated by $(x' = x + 1)$.

Example 5 (Rule-LP): Let $S \stackrel{\text{def}}{=} (\dot{h} = 1\ \texttt{until}\ h \geq 3;\ \dot{h} = -1\ \texttt{until}\ h \leq 1)$, the loop program $P \stackrel{\text{def}}{=} (\texttt{while}\ true\ \texttt{do}\ S\ \texttt{od})$. For clocks c and d, we define a relation $\preceq_t$ between two clocks, that $c \preceq_t d$ if $c \preceq d$ and for all $i \in \mathbb{N}, c[i] \leq t, d[i] \leq t$. Let $b_1 \stackrel{\text{def}}{=} (h \geq 3)$, $b_2 \stackrel{\text{def}}{=} (h \leq 1)$, $\beta \stackrel{\text{def}}{=} t \geq 0 \wedge h(t) \in [1, 3] \wedge b_1.clock \preceq_t b_2.clock$, $\alpha \stackrel{\text{def}}{=} \texttt{T}$, $\varsigma \stackrel{\text{def}}{=} (b_1.clock' \preceq_{t'} b_2.clock')$. Thus the proof is instantiated as follow:

$$\frac{(\texttt{T};\texttt{T}) \Rightarrow \texttt{T},\ \ \texttt{T} : \{\beta\}\ [S]\ \{\langle \varsigma \bowtie \beta[x'/x]\rangle\}}{\texttt{T} : \{\beta\}\ [\texttt{while}\ \ true\ \ \texttt{do}\ \ S\ \ \texttt{od}]\ \{\langle \varsigma \bowtie \texttt{F} \wedge \beta[x'/x]\rangle\}}$$

where, the assertion $\beta[x'/x] \stackrel{\text{def}}{=} t' \geq 0 \wedge h(t') \in [1, 3] \wedge b_1.clock' \preceq_{t'} b_2.clock'$. At this point, we have to proof: $\texttt{T} : \{\beta\}\ [S]\ \{\langle \varsigma \bowtie \beta[x'/x]\rangle\}$. Then, we apply rule-SC:

$$\frac{\begin{array}{l}\varsigma \vee (\gamma_1; \varsigma) \Rightarrow \varsigma,\ \ (\gamma_1; \tau_2) \Rightarrow \beta[x'/x],\ \ \texttt{T} \vee (\texttt{T};\texttt{T}) \Rightarrow \texttt{T},\\ \texttt{T} : \{\beta\}\ [S_1]\ \{\langle \varsigma \bowtie \gamma_1 \wedge \varepsilon\rangle\},\ \ \texttt{T} : \{\varepsilon[x/x']\}\ [S_2]\ \{\langle \varsigma \bowtie \tau_2\rangle\}\end{array}}{\texttt{T} : \{\beta\}\ [S_1; S_2]\ \{\langle \varsigma \bowtie \beta[x'/x]\rangle\}}$$

where, $S_1 \stackrel{\text{def}}{=} (\dot{h} = 1\ \texttt{until}\ h \geq 3)$, $S_2 \stackrel{\text{def}}{=} (\dot{h} = -1\ \texttt{until}\ h \leq 1)$, $\gamma_1 \stackrel{\text{def}}{=} \xi(b_1)$, $\tau_2 \stackrel{\text{def}}{=} \beta[x'/x]$, $\varepsilon \stackrel{\text{def}}{=} |b_1.clock'| > |b_2.clock'| \wedge b_1.clock' \preceq_{t'} b_2.clock'$. Likewise, $\varepsilon[x/x'] \stackrel{\text{def}}{=} |b_1.clock| > |b_2.clock| \wedge b_1.clock \preceq_t b_2.clock$. Then, for left branch S_1,

$$\frac{\begin{array}{l}\beta \Rightarrow \beta,\ \ \beta \wedge P \wedge \theta(b_1) \wedge count' = count \Rightarrow \varsigma,\\ \texttt{T} : \{\beta\}\ [S_1]\ \{P \wedge \langle \theta(b_1) \wedge count' = count \bowtie \xi(b_1) \wedge count' > \lambda\rangle\},\\ \beta \wedge P \wedge \xi(b_1) \wedge count' > \lambda \Rightarrow \gamma_1 \wedge \varepsilon\end{array}}{\texttt{T} : \{\beta\}\ [S_1]\ \{\langle \varsigma \bowtie \gamma_1 \wedge \varepsilon\rangle\}}$$

where, $\lambda \stackrel{\text{def}}{=} \texttt{max}(\{0, count\} \cup \{\pi_2(last(b_1.clock'))\})$, $P \stackrel{\text{def}}{=} (II_B \wedge \forall v \in [t, t') \bullet \dot{h}(v) = 1)$ for $B \stackrel{\text{def}}{=} PVar \cup \{s.clock \mid s \in OutSignal\}$. The proof for the right branch S_2 can be constructed in a similar way.

V. Case Study I: Water Tank

We present the HML program of water tank as a case study of hybrid system with the application of previous proof rules. The model consists of two modes, the water level is rising as a speed of a when the valve is open, and falling as the speed of b when the valve is closed. The detailed values for variables are listed in Table I.

TABLE I: Initial values for variables in the water tank model. The dimension cm represents centimeter(s)

Var	Type	Init	Meaning
H	Discrete	10	The highest water level is 10 cm
L	Discrete	5	The lowest water level is 5 cm
h_{init}	Discrete	8	Initial water level is 8 cm
h	Continue	8	Denotes the water level over time, its initial value equals h_{init}
ε	Discrete	0.5	A small value (0.5 cm) used for the control of water level, i.e., $\varepsilon < \mathtt{min}(H - h_{init}, h_{init} - L)$
a	Discrete	1	The raising velocity ($1\ cm/s$) of the water level when the control valve is open
b	Discrete	1	The falling velocity ($1\ cm/s$) of the water level when the control valve is closed

A. HML Model of Water Tank

First, we define the sub-programs of the system for continuous dynamics as follows:

$$open \overset{\text{def}}{=} (\dot{h} = a)\ \mathtt{until}\ (h \geq H - \varepsilon),$$
$$closed \overset{\text{def}}{=} (\dot{h} = -b)\ \mathtt{until}\ (h \leq L + \varepsilon),$$

where, program $open$ represents the water level is raising according to the first-order differential equation $\dot{h} = a$ with the exit-condition $h \geq H - \varepsilon$. The raising of water level shall terminate when the exit-condition becomes true. Likewise, program $closed$ denotes the scenario where the water level is falling. Utilizing sub-programs that we have defined, we can declared the program $\mathcal{W}$ of water tank to be a loop program: $\mathcal{W} \overset{\text{def}}{=} (\mathtt{while}\ true\ \mathtt{do}\ !on; open; !off; closed;\ \mathtt{od})$. where, two signals on and off are employed for indicating that the valve is open or closed, respectively. Consider a design $\mathcal{D} \overset{\text{def}}{=} [\alpha : (\beta \vdash_{\mathbf{H}} \langle \varsigma \bowtie \tau \rangle)]$, with $\alpha \overset{\text{def}}{=} \mathtt{T}$, $\beta \overset{\text{def}}{=} (h(t) \in [L, H] \wedge c.clock \preceq_t off.clock)$, $\varsigma \overset{\text{def}}{=} c.clock' \preceq_{t'} off.clock'$, and $\tau \overset{\text{def}}{=} \beta[x'/x]$. In which, $c \overset{\text{def}}{=} (h \geq H - \varepsilon)$.

The water tank program $\mathcal{W}$ implements the design $\mathcal{D}$, if we can proof:

$$\alpha : \{\beta\}\ [\mathcal{W}]\ \{\langle \varsigma \bowtie \tau \rangle\},$$

where, the program $\mathcal{W}$ starts under the condition β that indicates the initial water level in the tank is between the lower bound L and the upper bound H, and the clock for boolean condition c is faster than the clock for signal off. The assertion ς denotes that the clock of c is faster than the clock for off when the program is in stable status. The assertion τ represents that the water lever is also positioned between L and H. Meanwhile, the clock relation for c and off is still respected.

B. Verification of Water Tank Program

Fig. 2 illustrates the proof outline for the water tank program $\mathcal{W}$. The assertion $\mathtt{T}$ in the left column represents the invariant α as presented in Sect. V-A. For simplicity we ignore the invariant $\mathtt{T}$ for the inner sub-proofs for the sub-programs in the loop. The program code and the sub-proofs of the loop are encoded in the square brackets '[' and ']'.

The proof outline consists of four sub-proofs one after another for sub-programs $!on$, $open$, $!off$ and $closed$, respectively. The postcondition for one proof is the primed version

$\mathtt{T}$	$\{h(t) \in [5, 10] \wedge c.clock \preceq_t off.clock\}$
	[$\mathtt{while}\ true\ \mathtt{do}$
	$\{\mathtt{T} \wedge h(t) \in [5, 10] \wedge c.clock \preceq_t off.clock\}$
	$[!on]$
	$\{\langle (c.clock' \preceq_{t'} off.clock') \bowtie (\mathbf{h(t')} \in [\mathbf{5, 10}]) \wedge (\mathbf{c.clock'} \preceq_{\mathbf{t'}} \mathbf{off.clock'})\rangle\}$
	$\{(h(t) \in [5, 10]) \wedge (c.clock \preceq_t off.clock)\}$
	$[(\dot{h} = 1)\ \mathtt{until}\ (h \geq 9.5)]$
	$\{\langle (c.clock' \preceq_{t'} off.clock') \bowtie (\mathbf{h(t')} \in [\mathbf{5, 10}]) \wedge (\mathbf{c.clock'} \preceq_{\mathbf{t'}} \mathbf{off.clock'}) \wedge (\lvert\mathbf{c.clock'}\rvert > \lvert\mathbf{off.clock'}\rvert)\rangle\}$
	$\{h(t) \in [5, 10] \wedge c.clock \preceq_t off.clock \wedge \lvert c.clock\rvert > \lvert off.clock\rvert)\}$
	$[!off]$
	$\{\langle (c.clock' \preceq_{t'} off.clock') \bowtie (\mathbf{h(t')} \in [\mathbf{5, 10}]) \wedge (\mathbf{c.clock'} \preceq_{\mathbf{t'}} \mathbf{off.clock'})\rangle\}$
	$\{h(t) \in [5, 10] \wedge c.clock \preceq_t off.clock\}$
	$[(\dot{h} = -1)\ \mathtt{until}\ (h \leq 5.5)]$
	$\{\langle c.clock' \preceq_{t'} off.clock' \bowtie h(t') \in [5, 10] \wedge c.clock' \preceq_{t'} off.clock'\rangle\}$
	$\mathtt{od}$]
	$\{\langle c.clock' \preceq_{t'} off.clock' \bowtie \mathtt{F} \wedge h(t') \in [5, 10] \wedge c.clock' \preceq_{t'} off.clock'\rangle\}$

Fig. 2: The proof outline for the program ($\mathcal{W}$) of water tank, the assertion $\mathtt{T}$ represents *true*, $\mathtt{F}$ for *false*. The values for (constant) discrete variables, for example the lower bound of water level $L = 5$, and the upper bound $H = 10$, are described in Table I

of the precondition for the succeed one. For instance, the postcondition $(h(t') \in [5, 10]) \wedge (c.clock' \preceq_{t'} off.clock')$ for sub-program $!on$ is the primed version of the precondition $(h(t) \in [5, 10]) \wedge (c.clock \preceq_t off.clock)$ for sub-program $open$. In the following, we describe the details for the proof of program $open$,

$$\{(h(t) \in [5, 10]) \wedge (c.clock \preceq_t off.clock)\}$$
$$[(\dot{h} = 1)\ \mathtt{until}\ (h \geq 9.5)]$$
$$\{\langle (c.clock' \preceq_{t'} off.clock') \bowtie (h(t') \in [5, 10]) \wedge (c.clock' \preceq_{t'} off.clock') \wedge (\lvert c.clock'\rvert > \lvert off.clock'\rvert)\rangle\}$$

where, the precondition $(h(t) \in [5, 10]) \wedge (c.clock \preceq_t off.clock)$ denotes that the water level h is between 5 and 10 at the begging, and the clock c is faster than clock off; the stable condition $(c.clock' \preceq_{t'} off.clock')$ indicates the clock c is faster than clock off for all the time instants when the status of the program is stable; the postcondition $(h(t') \in [5, 10]) \wedge (c.clock' \preceq_{t'} off.clock') \wedge (\lvert c.clock'\rvert > \lvert off.clock'\rvert)$ means the water level constraint and clock relation also hold when the program terminates, meanwhile, the size of clock for c is larger than the size of clock for off. Based on Rule-CQ, we have the following proof obligations

1	$(h(t) \in [5, 10]) \wedge (c.clock \preceq_t off.clock)$ $\Rightarrow (h(t) \in [5, 10]) \wedge (c.clock \preceq_t off.clock)$
2	$\{(h(t) \in [5, 10]) \wedge (c.clock \preceq_t off.clock)\}$ $[(\dot{h} = 1)\ \mathtt{until}\ (h \geq 9.5)]$ $\{P \wedge \langle \theta(c) \wedge (count' = count) \bowtie \xi(c) \wedge (count' > \lambda)\rangle\}$

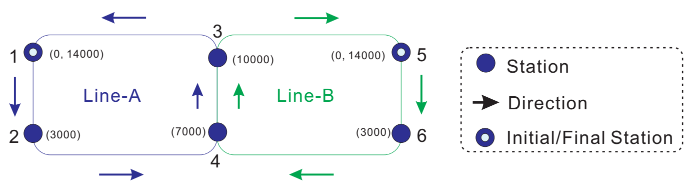

Fig. 4: Overlapping metro lines. The Station-3 and Station-4, together with the track between them, are overlapped by Line-A and Line-B. In the left circle (Line-A with stations $1 \sim 4$) the train (T_1) runs in an anti-clockwise direction, while the train (T_2) runs in a clockwise direction in the right circle (Line-B with stations $3 \sim 6$)

3	$(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock) \wedge P \wedge \theta(c)$ $\Rightarrow (c.clock' \preceq_{t'} off.clock')$
4	$(h(t) \in [5,10]) \wedge (c.clock \preceq_t off.clock) \wedge P \wedge \xi(c) \wedge (count' > \lambda)$ $\Rightarrow (h(t') \in [5,10]) \wedge (c.clock' \preceq_{t'} off.clock') \wedge (\lvert c.clock' \rvert > \lvert off.clock' \rvert)$

in which, $P \overset{\text{def}}{=} (II_B \wedge \forall v \in [t,t') \bullet \dot{h}(v) = 1)$ for $B \overset{\text{def}}{=} PVar \cup \{s.clock \mid s \in OutSignal\}$. The first proof obligation is trivial. The second one is the direct application of Rule-EQ. For the third one, the clock $c.clock'$ is equal to the old value of $c.clock$ as $\theta(c)$ denotes that c is absent during the time interval $[t,t']$. Moreover, because $off.clock$ is in set B, thus $off.clock' = off.clock$ can be implied by II_B directly. As a result, $(c.clock' \preceq_{t'} off.clock')$ is satisfied. In the fourth proof obligate, $\xi(c)$ denotes that c is present at time t', it means that $|c.clock'| = |c.clock| + 1$, while $off.clock' = off.clock$ by II_B, therefore $|c.clock'|$ is larger than $|off.clock'|$.

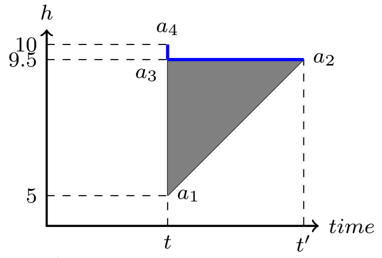

Fig. 3: The state space of *open*

Fig. 3 illustrates the state space (w.r.t water level and time) of the program *open*. All the points in the gray area and two thick line segments $[a_2, a_3]$, $[a_3, a_4]$ constitute the state space between the time interval $[t, t']$, where t' is the maximum time instant the program may terminate. The line segments $[a_2, a_3]$ and $[a_3, a_4]$ denote all possible terminating states. For any point in the line segments $[a_2, a_3]$, the water level is 9.5. If the initial water level $h(t) \in [5, 9.5]$, the program will terminate in $[a_2, a_3]$, the corresponding terminating time t' may not be the maximum time instant if $h(t) \neq 5$. Otherwise, if $h(t) \in [9.5, 10]$, the program will terminate in $[a_3, a_4]$ immediately, and $t' = t$. For both cases, we have $h(t') \in [9.5, 10]$, and $h(t') \in [9.5, 10] \Rightarrow h(t') \in [5, 10]$.

VI. Case Study II: Overlapping Metro Lines

In this section, the case for a subway control system with two overlapping metro lines is proposed as an application of parallel HML programs.

A. Model of Overlapping Metro Lines

In Fig. 4, the distances between a station and its neighbors are 3 (km) in vertical direction or 4 (km) in horizontal direction. For example, the distance between stations 1 and 2 is 3 (km). The distance between stations 2 and 4 is 4 (km). According to the distance, we add a label for each station to denote its position. In Line-A, Station-1 is both the initial and final station, thus we add label $(0, 14000)$ for its position. Likewise, Station-5 is both the initial and final station in Line-B. In this case, we have two trains, one for Line-A, another for Line-B. Let v_i be the continuous variable representing the velocity of train T_i, p_i be the variable for position, where, $i = 1, 2$, T_1 is the train for Line-A, T_2 for Line-B. The HML program for train T_i is illustrated as follow,

```
$Id_i := 0;$
while $true$ do
        $Stop; Id_i := Id_i + 1; Cho; frun_i := true;$
        $(\dot{p}_i = 0$ init $0$ until $true) \triangleleft Id_i = 2 \triangleright$ skip;
        while $frun_i$ do  $Run$  od
 od
```

where, Id_i is a discrete variable as an index used to update the position (S_i) of the station ahead of the train. For example, if the train T_1 is stop at Station-1, then Station-2 is ahead of T_1, the position $S_1 = 3000$. The sub-programs for the program are defined below,

$$Stop \overset{\text{def}}{=} l_i := \lfloor p_i/250 \rfloor; l_i := l_i \ \mathbf{mod}\ 56; !g_{i,l_i}; \texttt{suspend}(50);$$

$$Cho \overset{\text{def}}{=} \texttt{when}(Id_i = 2\&(S_i := 3000) \mathbin{[\!]} Id_i = 3\&(S_i := 7000) \mathbin{[\!]} Id_i = 4\&(S_i := 10000) \mathbin{[\!]} Id_i = 5\&S_i := 14000; Id_i := 1),$$

where, $\lfloor p_i/250 \rfloor$ is the largest integer not greater than $p_i/250$, l_i is a variable for recording the numbers of track segments. In Fig. 5 (see the Appendix), the Line-A (also for Line-B) is partitioned into 56 segments, each has a length of 250 (m). The number $l_i \in \{n \in \mathbb{N} \mid 0 \leq n \leq 55\}$. The emitting of signal g_{i,l_i} represents that the train T_i is in the segment l_i. The sub-program *Run* is presented below.

```
$fcrun_i := true; l_i := \lfloor p_i/250 \rfloor; !g_{i,l_i}; l_i := l_i + 1;$
while $fcrun_i$ do
   $(\dot{v}_i = 0.5 \mid\mid \dot{p}_i = v_i)$ until $(p_i = 250 * l_i) \oplus (S_i = p_i + 500)$
                                        $\oplus (v_i = 20) \oplus urstop_i;$
   when$((p_i = 250 * l_i)\&(!g_{i,l_i}; l_i := (l_i + 1)$ mod $56)$
      $[\!]((v_i = 20) \oplus (S_i = p_i + 500) \oplus urstop_i)\&(fcrun_i := false))$
 od;
when$((S_i = p_i + 500)\&Near \ [\!] \ (v_i = 20)\&Srun \ [\!] \ urstop_i\&Urde)$
```

where, $fcrun$ is a boolean variable, the sub-program *Run* denotes the continuous behavior after the train departed from a station. The acceleration of the velocity is 0.5 (m/s^2). The control of the program will turn to sub-program *Near* when the distance between the train and the station ahead is equal to 500 (m). Moreover, the control will turn to *Srun* whenever the velocity of the train is equal to 20 (m/s). If the signal $urstop_i$ is present, the control will turn to sub-program $Urde$. Meanwhile, if the position of the train is in a segment, then the corresponding signal g_{i,l_i} will be emitted. The following sub-programs can be understood as a similar way. Here, we just list the codes for all the other sub-programs as shown in the Appendix.

The controller for T_1 and T_2, receives signals g_{i,l_i} then decides whether to stop a train urgently by emitting signals $urstop_i$, or starts a train from urgent-stop status by signals $urstart_i$. The HML program of the controller is presented as follow,

```
$f_1 := 0; f_2 := 0; cf_1 := 0; cf_2 := 0;$
while $true$ do
  when$(g_{1,f_1}\&(b := true; cf_1 := f_1) \, [] \, g_{1,f_2}\&(b := false; cf_2 := f_2));$
  $dcond := (26 \leq cf_1 \wedge cf_1 \leq 40 \wedge 26 \leq cf_2 \wedge cf_2 \leq 40);$
  $Decide \lhd \ dcond \ \rhd (!urstart_1; !urstart_2);$
  $(f_1 := (f_1 + 1 \ \mathbf{mod} \ 56)) \lhd b \rhd (f_2 := (f_2 + 1 \ \mathbf{mod} \ 56))$
 od
```

where, the decision is made when the trains are in track segments with numbers ranged over $\{n \in \mathbb{N} \mid 26 \leq n \leq 40\}$, because in this situation, these two trains will share the track segments. The variable f_i and cf_i denote the numbers of expected and current track segments for train T_i, respectively. The sub-program $Decide$ is

```
$ds_1 := cf_1 \leq cf_2 \wedge cf_1 \geq cf_2 - 2;$
$ds_2 := cf_2 \leq cf_1 \wedge cf_2 \geq cf_1 - 2;$
$!urstop_1 \lhd ds_1 \rhd (!urstop_2 \lhd ds_2 \rhd (!urstart_1; !urstart_2));$
```

Let $\mathcal{C}$ be the program of controller, $\mathcal{T}_i$ be the program of train T_i, $i = 1, 2$. Now, we can define the HML program $\mathcal{O}$ for the overlapping metro lines system as follow:

$$\mathcal{O} \overset{\text{def}}{=} \mathcal{C} \mid\mid \mathcal{T}_1 \mid\mid \mathcal{T}_2,$$

where, the program for the system consists of three components, sub-programs of the controller, the trains T_1 and T_2.

B. Verification of Overlapping Metro Lines

We want to make sure the following assertion is true during the execution of the system.

$$(p_1 \in [7000, 10000] \wedge p_2 \in [7000, 10000]) \Rightarrow (p_1 \neq p_2),$$

where, the meaning of the assertion is that if the positions of trains T_1, T_2 are in the shared track segment, then they shall not have a collision. Consider a design for the overlapping metro lines system, $\mathcal{D} \overset{\text{def}}{=} [\alpha : (\beta \vdash_{\mathbf{H}} \langle \varsigma \bowtie \tau \rangle)]$, with $\alpha \overset{\text{def}}{=} \mathtt{T}, \tau \overset{\text{def}}{=} \mathtt{F}$, and,

$$\begin{aligned}
\beta \overset{\text{def}}{=} & (\forall i \in \{1, 2\} \bullet (p_i(t) = 0 \wedge f_i = 0 \wedge cf_i = 0 \wedge Id_i = 0) \\
& \wedge t = 0 \wedge \forall i \in \{1, 2\}, j \in \{0, ..., 55\} \bullet g_{i,j}.clock = \emptyset); \\
\varsigma \overset{\text{def}}{=} & (p_1(t') \notin [7000, 10000] \vee p_2(t') \notin [7000, 10000] \\
& \vee p_1(t') \neq p_2(t')),
\end{aligned}$$

where, β indicates that the initial positions of trains are 0, the trains are stop at the initial stations. The starting time point is 0, and all the clocks are empty. The assertion ς represents that the trains T_1 and T_2 are not at the shared track or they do not have a collision. According to the design, we have the proof obligate for HML program $\mathcal{O}$: $\alpha : \{\beta\} \ [\mathcal{O}] \ \{\langle \varsigma \bowtie \tau \rangle\}$, where, the assertions indicate that the trains T_1 and T_2 will not collision during the execution of the system. For controller $\mathcal{C}$, we have the precondition,

$$\begin{aligned}
\beta_{\mathcal{C}} \overset{\text{def}}{=} & (cf_1 < 27 \vee cf_1 > 40 \vee cf_1 < cf_2 - 1 \vee \\
& cf_2 < 27 \vee cf_2 > 40 \ \vee cf_2 < cf_1 - 1) \\
& \wedge \\
& ((g_{(i,cf_i)}.clock \neq \emptyset \wedge g_{(i,cf_i+1 \ \mathbf{mod} \ 56)}.clock \neq \emptyset) \Rightarrow \\
& \pi_1(last(g_{(i,cf_i)}.clock)) > \pi_1(last(g_{(i,cf_i+1 \ \mathbf{mod} \ 56)}.clock))),
\end{aligned}$$

in which, we have the predicate $\pi_1(last(g_{(i,cf_i)}.clock)) > \pi_1(last(g_{(i,cf_i+1 \ \mathbf{mod} \ 56)}.clock))$ meaning that the signal g_{i,cf_i} is the latest signal that received from train T_i. For the trains $\mathcal{T}_i$, $i = 1, 2$, we define the preconditions,

$$\begin{aligned}
\beta_{\mathcal{T}_i} \overset{\text{def}}{=} & \ p_i(t) \in [250 * in_i, 250 * (in_i + 2)) \\
& \wedge (g_{i,0}.clock \preceq_t ... \preceq_t g_{i,55}.clock),
\end{aligned}$$

where, $in_i \overset{\text{def}}{=} \mathtt{max}(g_{i,0}, ..., g_{i,55})$ implies that g_{i,in_i} is the latest emitted signal from program $\mathcal{T}_i$, thus $cf_i = in_i$. If all the clocks for $g_{i,j}$ $(i \in \{1, 2\}, j \in \{0, ..., 55\})$ are empty, then in_i is 0. The proof outline for HML program $\mathcal{O}$ is illustrated as follows,

$$\begin{array}{l}
\textit{Separate proofs for } \mathcal{C}, \mathcal{T}_1 \textit{ and } \mathcal{T}_2 ... \\
\hline
\mathtt{T} : \{\beta_{\mathcal{C}}\} \quad [\mathcal{C}] \quad \{\langle \beta_{\mathcal{C}}[x'/x] \bowtie \mathtt{F} \rangle\}, \\
\mathtt{T} : \{\beta_{\mathcal{T}_1}\} \ [\mathcal{T}_1] \ \{\langle \beta_{\mathcal{T}_1}[x'/x] \bowtie \mathtt{F} \rangle\}, \\
\mathtt{T} : \{\beta_{\mathcal{T}_2}\} \ [\mathcal{T}_2] \ \{\langle \beta_{\mathcal{T}_2}[x'/x] \bowtie \mathtt{F} \rangle\} \\
\hline
\beta \Rightarrow (\beta_{\mathcal{C}} \wedge \beta_{\mathcal{T}_1} \wedge \beta_{\mathcal{T}_2}), \\
\mathtt{T} : \{\beta_{\mathcal{C}} \wedge \beta_{\mathcal{T}_1} \wedge \beta_{\mathcal{T}_2}\} \\
\qquad\qquad [\mathcal{C} \mid\mid \mathcal{T}_1 \mid\mid \mathcal{T}_2] \\
\qquad \{\langle (\beta_{\mathcal{C}} \wedge \beta_{\mathcal{T}_1} \wedge \beta_{\mathcal{T}_2})[x'/x] \bowtie \mathtt{F} \rangle\}, \\
(\beta_{\mathcal{C}} \wedge \beta_{\mathcal{T}_1} \wedge \beta_{\mathcal{T}_2})[x'/x] \Rightarrow \varsigma, \ \mathtt{F} \Rightarrow \tau \\
\hline
\mathtt{T} : \{\beta\} \ [\mathcal{O}] \ \{\langle \varsigma \bowtie \tau \rangle\}
\end{array}$$

where, $\mathtt{F}$ is used to indicate the non-termination because we have while-true loop in the program. The separate proofs for $\mathcal{C}, \mathcal{T}_1$ and $\mathcal{T}_2$ can be finished respectively as all of them are sequential programs without shared discrete/continuous variables. The merge mechanism is not applied here as we do not have the *count* (and *count*$'$) in the assertions.

VII. Conclusions and Future Works

In this paper, we proposed an approach for verification of hybrid systems which are constructed by HML language. The proof rules with respect to the HML language are illustrated in the style of Hoare logic triples. As an application of our approach, a water tank program is produced as an HML program, also we verified the program with the satisfiability according to a design requirement. For parallel HML programs, we presented the parallel program which consists of overlapping metro lines with two subway trains, and analyzed the collision avoidance scenario. For future work, we will focus on the runtime verification of hybrid systems based on the hybrid relation approach. In addition, the automated tools for formal analysis including simulation, verification and animation are planed to be implemented.

References

[1] O. Maler, Z. Manna, and A. Pnueli, "From timed to hybrid systems," in *REX Workshop*, ser. LNCS, vol. 600. Springer, 1991, pp. 447–484.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3 – 34, 1995.

[3] W. Glover and J. Lygeros, "A Stochastic Hybrid Model for Air Traffic Control Simulation," in *Hybrid Systems*, ser. LNCS, vol. 2993. Springer, 2004, pp. 372–386.

[4] C. Piazza, M. Antoniotti, V. Mysore, A. Policriti, F. Winkler, and B. Mishra, "Algorithmic algebraic model checking i: Challenges from systems biology," in *CAV*, ser. LNCS, vol. 3576. Springer, 2005.

[5] J. H. Gillula, G. M. Hoffmann, H. Huang, M. P. Vitus, and C. J. Tomlin, "Applications of hybrid reachability analysis to robotic aerial vehicles," *International Journal of Robotics Research*, vol. 30, no. 3, pp. 335–354, 2011.

[6] J. He, "A clock-based framework for construction of hybrid systems," in *Proceedings of ICTAC 2013*, ser. LNCS. Springer Berlin Heidelberg, 2013, vol. 8049, pp. 22–41.

[7] ——, "Hybrid relation calculus," in *Proceedings of ICECCS 2013*. IEEE, 2013, p. 2.

[8] T. A. Henzinger, "The theory of hybrid automata," in *LICS*. IEEE Computer Society, 1996, pp. 278–292.

[9] N. A. Lynch, R. Segala, and F. W. Vaandrager, "Hybrid I/O Automata," *Inf. Comput.*, vol. 185, no. 1, pp. 105–157, 2003.

[10] Modelica Association, "Modelica," https://www.modelica.org/.

[11] P. Fritzson, "Modelica - a cyber-physical modeling language and the OpenModelica environment," in *IWCMC*. IEEE, 2011, pp. 1648–1653.

[12] A. Platzer, *Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics*. Springer, 2010.

[13] H. Jifeng and X. Qiwen, "Advanced features of duration calculus and their applications in sequential hybrid programs," *Formal Aspects of Computing*, vol. 15, no. 1, pp. 84–99, 2003.

[14] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn, "A calculus of durations," *Inf. Process. Lett.*, vol. 40, no. 5, pp. 269–276, 1991.

[15] Z. Chaochen, A. P. Ravn, and M. R. Hansen, "An extended duration calculus for hybrid real-time systems," in *Hybrid Systems*, 1992, pp. 36–59.

[16] R. Priemer, *Introductory signal processing*. World Scientific, 1991.

Appendix
Sub-Programs for Overlapping Metro Lines

Sub-program $Near$ represents the behaviors when the train is close to a station:

```
$fcnear_i := true; l_i := \lfloor p_i/250 \rfloor; !g_{i,l_i}; l_i := l_i + 1;$
$acc_i := -0.5 * v^2/(S_i - p_i);$
while $fcnear_i$ do
    $(\dot{v_i} = acc_i \,||\, \dot{p_i} = v_i)$ until $(p_i = 250 * l_i) \oplus (S_i = p_i) \oplus urstop_i;$
    when$((p_i = 250 * l_i)\&(!g_{i,l_i}; l_i := (l_i + 1)$ mod $56)$
          $[\!]((S_i = p_i) \oplus urstop_i)\&(fcnear_i := false))$
 od;
when$((S_i = p_i)\&(frun_i := false) \,[\!]\, urstop_i\&Urde)$
```

Sub-program $Srun$ denotes the stable running of the train which retains the velocity of 20 (m/s):

```
$fcsrun_i := true; l_i := \lfloor p_i/250 \rfloor; !g_{i,l_i}; l_i := l_i + 1;$
while $fcsrun_i$ do
    $(\dot{v_i} = 0 \,||\, \dot{p_i} = v_i)$
     until $(p_i = 250 * l_i) \oplus (S_i = p_i + 500) \oplus urstop_i;$
    when$((p_i = 250 * l_i)\&(!g_{i,l_i}; l_i := (l_i + 1)$ mod $56)$
          $[\!]((S_i = p_i + 500) \oplus urstop_i)\&(fcsrun_i := false))$
 od;
when$((S_i = p_i + 500)\&Near \,[\!]\, urstop_i\&Urde)$
```

Sub-program $Urde$ is used for the urgent decreasing of velocity:

```
$furde_i = true; l_i := \lfloor p_i/250 \rfloor; !g_{i,l_i};$
while $furde_i$ do
     $(\dot{v_i} = -1.5 \,||\, \dot{p_i} = v_i)$ until $v_i = 0;$
     $l_i := \lfloor p_i/250 \rfloor; !g_{i,l_i};$
     $(\dot{v_i} = 0 \,||\, \dot{p_i} = 0)$ until $urstart_i;$
     when$((S_i > p_i + 800)\&(furde_i := false)$
           $[\!](S_i \leq p_i + 800)\&Urin)$
 od;
```

Sub-program $Urin$ is for the urgent increasing of velocity:

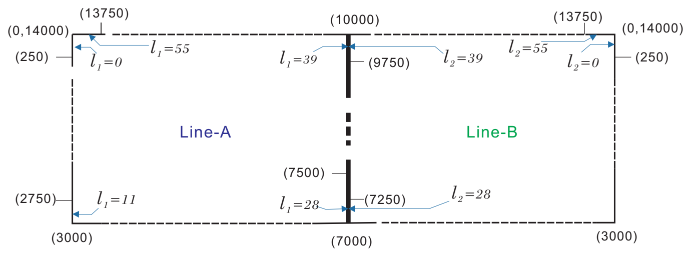

Fig. 5: Track partitions for Line-A and Line-B

```
$fcurin_i := true; l_i := \lfloor p_i/250 \rfloor; !g_{i,l_i}; l_i := l_i + 1;$
while $fcurin_i$ do
  $(\dot{v_i} = 0.5 \,||\, \dot{p_i} = v_i)$
   until $(p_i = 250 * l_i) \oplus urstop_i \oplus (p_i = 0.5 * (S_i + p_i));$
  when$((p_i = 250 * l_i)\&(!g_{i,l_i}; l_i := (l_i + 1)$ mod $56)$
         $[\!](urstop_i \oplus (p_i = 0.5 * (S_i + p_i)))\&(fcurin_i := false))$
 od;
when$(urstop_i\&$skip $[\!]\ p_i = 0.5 * (S_i + p_i)\&Urec)$
```

Sub-program $Urec$ is similar to $Near$ for the control of the velocity when the train is recovered from a urgent stop and close to a station:

```
$fcurec_i := true; l_i := \lfloor p_i/250 \rfloor; !g_{i,l_i}; l_i := l_i + 1;$
while $fcurec_i$ do
  $(\dot{v_i} = -0.5 \,||\, \dot{p_i} = v_i)$ until $(p_i = 250 * l_i) \oplus (S_i = p_i) \oplus urstop_i;$
  when$((p_i = 250 * l_i)\&(!g_{i,l_i}; l_i := (l_i + 1)$ mod $56)$
         $[\!]((S_i = p_i) \oplus urstop_i))\&(fcurec_i := false))$
 od;
when$(S_i = p_i\&(furde_i := false; frun_i := false) \,[\!]\, urstop_i\&$skip$)$
```