Translation from Hybrid Concurrent Constraint Programming Language into Real-Time Transition Systems

Daisuke Ishii  
Waseda University  
3-4-1, Okubo, Shinjuku-ku  
Tokyo, Japan  
{ishii, ueda}@ueda.info.waseda.ac.jp

Hiroshi Hosobe  
National Institute of Informatics  
2-1-2, Hitotsubashi, Chiyoda-ku  
Tokyo, Japan  
hosobe@nii.ac.jp

ABSTRACT
We propose a translation method from a constraint-based language into real-time transition systems that are a traditional framework for formalizing HA. We discuss differences between the two modeling languages with respect to the translation method.

Categories and Subject Descriptors
D.3.1 [Programming Languages]: Formal Definitions and Theory

General Terms
Languages

1. INTRODUCTION

Modeling frameworks for hybrid systems [7] include equation-based (or constraint-based) languages [5] and hybrid automata (HA) [6]. This paper provides a translation method (Section 4) from a constraint-based language called Tiny HCC (Section 2) into a real-time transition system (Section 3), which is a traditional framework for describing the semantics of HA [6, 2]. The translation allows developments on the analysis of HA (e.g., nonzeno property [6], finite-set refutability [2], and bisimulation analysis) to be extended for Tiny HCC programs.

Figure 1 illustrates a model of a well-known bouncing particle in the Tiny HCC language. In this model, the one-dimensional position and velocity of the particle are represented by real functions over time $p, v : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$. At line 1, we bound the initial domain of $p$ at time $t = 0$. The `hence $P$ construct at lines 2–5 activates the inner sub-program $P$ over the time line for every $t > 0$. At line 3, we describe a constraint to keep the value of $p$ unchanged even when a discrete change occurred. Constructs of the form `if $C$ then $P$` at lines 4 and 5 check whether the constraint $C$ is entailed by the constraint store, and when $C$ is entailed, interpret the sub-process $P$. The sub-process at line 4 involves a constraint that express the dynamics of the particle (we express the derivative $d(p(t))/dt$ by $\dot{p}$). A contact of the particle with the ground is detected by the constraint $p_\sim = 0$ at line 5 (we express the left-hand limit $\lim_{t\rightarrow t^+} p(t)$ of $p(t)$ by $p_\sim$), and the following sub-process describes an instantaneous change of the velocity. We describe how an execution of the model is formalized in Section 5.

An HA illustrated in Figure 2 also models the bouncing particle. We describe differences between these two frameworks in Section 6.

1.1 Related Work

Gupta et al. [4] proposed a translation method from and to HCC (an extended Tiny HCC) and HA. An example that simulated an HA by HCC (an extended Tiny HCC) was also shown [1]. However, we consider the translation method is not sufficient enough to compare the two languages because the method only handles a variant of HA that is different from HA described in this paper: discrete states involve HCC programs; and a state transition should occur whenever a guard constraint is satisfied.

Falaschi et al. [3] described another translation method from HCC into HA. The method first generates a graph structure that expresses executions of an HCC program, and then the method converts the structure into an HA. The translation method seems to be insufficient because a translation succeeds only when bounded structure can be generated from a program. Moreover, only limited examples were considered. It is not clear whether the method handles generic programs including a program with uncertain parameters that may result in qualitatively different executions.

2. HYBRID CONCURRENT CONSTRAINT PROGRAMMING

Gupta et al. [5] proposed a programming language called hybrid concurrent constraint programming (HCC). HCC describes hybrid systems in a declarative style using constraints (i.e., partial information). Computation in HCC is per-
formed by (concurrent) processes that communicate with each other in terms of constraints intermediated by a constraint store. Computation proceeds by accumulating constraints into the store, and by checking whether constraints are entailed by the constraints in the store.

In this paper, we consider the Tiny HCC language, a small subset of the full HCC constructs.

### 2.1 The Tiny HCC Language

The following is the syntax of Tiny HCC processes:

- **(process)** $P ::= C \mid \text{if } C \text{ then } P \mid P, P \mid \text{hence } P \mid \epsilon$
- **(constraint)** $C ::= A \mid \text{cont}(V)$
- **(expression)** $A ::= \text{an arithmetic expression involving } V$
- **(variable)** $V ::= \text{id} \mid \text{id}_d \mid \text{id}_-$

A *tell* process $C$ adds the constraint $C$ to a constraint store, which is a conjunction of all the constraints given by the tell processes. All variables in an expression $A$ range over real-valued functions over time $\mathbb{R}_{\geq 0} \to \mathbb{R}$. A variable is represented by an identifier $id$ such as $x$ (i.e., an abbreviation for $x(\tau)$). The constraint $C$ held by the tell process is one of the following kinds:

- **instantaneous constraints** of the form $A$ not involving variables of the form $id$;
- **continuous constraints** of the form $A$ involving only pure identifiers and $id$ as variables; or
- constraints of the form $\text{cont}(V)$ that propagate the value of $x(\tau)$ into the variable $x(\tau)$ in an execution (this constraint was introduced in the existing implementation of HCC [1]).

An *ask* process, if $C$ then $P$, contains a constraint $C$ called a guard constraint of the form $A$ involving only pure identifiers, or only $id_-$ as variables (an instance is denoted hereafter by $c_0$). When the entailment relation between the constraint store and the guard constraint $c_0$ held by an ask process is changed, a discrete change is triggered. In this study, we consider the ask processes if $c_0$ then $P$ formed as follows:

- $c_0$ contains only variables of the form $id_-$, and $P$ contains only instantaneous constraints; or
- $c_0$ contains only variables described by pure identifiers, and $P$ contains only tell processes that maintain continuous constraints.

A *hence* process hence $P$ activates a process $P$ along the time line for all $t > t_0$, where $t_0$ is the time at which the hence process is interpreted.

We consider Tiny HCC as a language scheme, and we do not specify a class of expressions that describe constraints.

### 3. REAL-TIME TRANSITION SYSTEMS

**Real-time transition systems** (RTTSs) are transition systems in which transitions are labeled by positive real numbers that represent the duration of the transitions [2].

An RTTS is defined as a triple $(S, T, S_0)$, where $S$ is a state space, $T \subseteq S \times \mathbb{R}_{\geq 0} \times S$ is the time transition relation, and $S_0 \subseteq S$ is a set of initial states. A transition $(s, t, s') \in T$ is also denoted by $s \xrightarrow{t} s'$. The relation satisfies the following conditions:

- Every transition $s \xrightarrow{t} s'$ with $t > 0$ can be split into transitions $s \xrightarrow{t_1} s''$ and $s'' \xrightarrow{t''} s'$, where $t = t_1 + t''$ and $t', t'' > 0$.
- For every two transitions $s \xrightarrow{t_1} s'$ and $s' \xrightarrow{t''} s''$ with $t_1, t'' > 0$, there exists a transition $s \xrightarrow{t_1 + t''} s''$.

In an RTTS, we describe a discrete change from a state $s \in S$ to $s' \in S$ by $s \xrightarrow{t} s'$, and we describe a continuous change from $s$ to $s'$ with the duration of $t \in \mathbb{R}_{>0}$ by $s \xrightarrow{t} s'$.

### 4. TRANSLATION METHOD

We formalize the operational semantics of Tiny HCC by means of translation into RTTSs. Gupta and others described the formal denotational and operational semantics of (default) HCC [5].

A program $P$ in Tiny HCC is translated into an RTTS $(S, T, S_0)$ as follows. We express the tuple of all pure identifiers of variables appearing in a program by $x = (x_1, \ldots, x_n)$, and the tuple of variables of the form $id_-$ corresponding to $x$ by $x_- = (x_1, \ldots, x_n)$. We use the following tuples to express states in $S$ of the RTTS and states internally used in transitions:

- Pairs $(P, v)$ consisting of a Tiny HCC program $P$ and a value $v$ in the continuous state space $\mathbb{R}^n$. $S$ is a set of these pairs.
- Quadruples $(P, c, H, m)$ consisting of a Tiny HCC program $P$, a constraint store $c$, a Tiny HCC program $H$ to be interpreted in the next phase, and a flag for switching between two modes $(c, d)$.

Figure 3: Translation rules from Tiny HCC into RTTSs.

\[
\begin{align*}
(c, c', H, m) & \sim (c', c \land H, m) \quad (1) \\
(c + c_0) & \sim (P, c, H, m) \quad (2) \\
((P_1, c, H, m), c) & \sim (P_1', c', H', m) \quad (3) \\
((P_1, P_2), c, H, m) & \sim ((P_1', P_2'), c', H', m) \\
(hence P, c, H, c) & \sim (P, c, (H, hence P, c)) \quad (4) \\
(hence P, c, H, d) & \sim (c, c, (H, hence P, d)) \quad (5) \\
(\text{cont}(x), c, H, m) & \sim (\text{cont}(x), c \land \text{cont}(x), H, m) \quad (6) \\
(H, (x = v), c, c) & \sim (P, c, H', c) \quad (7) \\
\begin{array}{l}
(\langle P, v \rangle, \text{cont}(P, H), \phi(t)) \\
\end{array} & \sim (P, v) \quad (8)
\end{align*}
\]
Before defining the transition relation $T$, we define a transition relation $(P, c, H, m) \leadsto (P', c', H', m)$ by Rules (1)–(6) in Figure 3 corresponding to each construct of Tiny HCC. When these transition rules are examined in the premises of Rules 7 and 8, the transitions are applied until no further transitions can take place.

- Rule (1) describes the tell process that conjuncts an atomic constraint $c$ with the store $c'$.
- Rule (2) describes the ask processes. The guard constraint $c_0$ should be entailed (denoted by $\vdash$) by the store $c$ to take out the process $P$.
- Rule (3) represents the parallel composition of processes $P_1$ and $P_2$. We also consider the reduction of $P_2$ in the same way.
- Rules (4) and (5) describe the hence $P$ process that expands the process $P$ over the time line. It takes out $P$ to apply the reduction in a continuous change phase, and just disappears in a discrete change phase. In each phase, the process is copied into the third element to continue the process in the next phase.
- Rule (6) describes the tell process of the constraint $\text{cont}(x_i)$, where $x_i$ is an element of $x$. The process is interpreted as in Rule (1), except for that it remains after the transition.

Now, we define the transitions in $T$ by Rules (7) and (8).

- Rule (7) describes continuous changes expressed as the transition $(H, v) \vdash ((P, H'), \phi(t))$. In the first premise, the program $H$ is interpreted with the store $x = v$. In the interpretation a hence process hence $P'$ is reduced to the sub-processes $P_i'$, and all ask processes in $H$ remain in $P$. We then have a store $c$ that can be regarded as an initial value problem for an ordinary differential equation (IVP-ODE), and we can obtain a trajectory $\phi$ satisfying (denoted by $\models$) the constraint store (the second premise). In the fourth premise, we check that the set of ask processes in $P$ does not change over the time interval $(0, t)$. Notice that an ask process disappears when the state (i.e., entailment relation) of the guard constraint on $x$ or $x_-$ changed.

- Rule (8) represents two kinds of transitions. First, it determines the set of initial states $S_0$ of the RTTS. Assume we have $(P, \bullet)$, where $P$ is an input program and • is a special value denoting the undefined value. In the premise, we compute $(P, \text{true}, \epsilon, d) \leadsto (P', c, H, d)$, where $P'$ is a residual process not needed in this case, $c$ stores the constraints added by the tell processes in $P$, and $H$ is a composition of hence processes in $P$. An initial state in $S_0$ is obtained as $(H, v)$, where $v$ is a value of $x$ satisfying the constraint $c$.

- Second, Rule (8) describes the computation of discrete changes. In the premise, the transition $(P, (x_-=v_-), \epsilon, d) \leadsto (P', c, H, d)$ results in a constraint store $c$ and a composed process $H$ of hence processes in $P$. We then compute a value $v$ that satisfies the constraint $c$, and an initial state $(H, v)$ for the next continuous change.

### 5. Example

In the following, we describe an execution of the Tiny HCC program in Figure 1, henceforth referred to as $P_0$. We abbreviate sub-processes in $P_m$ to $P_a$ and $P_b$ as follows:

- $P_a$:
  1. $\text{cont}(p)$,
  2. if $p_0 = 0$ then $v = -\frac{1}{2}v_-$

- $P_b$:
  1. hence {
      2. $P_a$,
      3. if $p > 0$ then $\{ p = v, \dot{v} = -1 \}$
  }

We also abbreviate the following constraint to $c_z$:

$$\text{cont}(p) \land \dot{p} = v \land \dot{v} = -1.$$

The execution of the program proceeds as follows:

1. We first obtain an initial state at time 0 (i.e., a state $s^0 \in S_0$ of the associated RTTS). We apply Rule (8) to a state $(P_0, \bullet)$, where $\bullet$ denotes the undefined value, as follows:

$$\llparenthesis P_0, \text{true}, \epsilon, d \rrparenthesis \leadsto ((\epsilon, c_0, P, d), v_0 = c_0) \quad (8)$$

The first premise is proved by applying the rules $\leadsto$ in Figure 3 as many times as possible. As a result, we obtain a quadruple $(\epsilon, c_0, P, d)$, where $c_0 \equiv p > 0$, and $P$ is the hence process in $P_0$. In the second premise, $v_0$ denotes a value for the variable $(p, v)$ that satisfies the constraint $c_0$. Assume that we take the value $(1, 0)$.

2. The resulting state evolves continuously over time $t > 0$ with respect to Rule (7). Here, we consider the continuous transition $\Delta_{\delta}$, where $\delta \in \mathbb{R}_{>0}$ is a duration for an evolution (we assume a sufficiently small $\delta$ that causes no discrete change). The transition is formalized as follows:

$$\forall t' \in (0, \delta)$$

$$\llparenthesis P_0, (x = v_0), \epsilon, c \rrparenthesis \leadsto ((P_0, (x = \phi_0(t')), \epsilon, d), (P_0, (x = \phi_0(t')), \epsilon, d))$$

$$\llparenthesis P_0, (x = v_0), \epsilon, c \rrparenthesis \leadsto ((P_0, (x = \phi_0(t')), \epsilon, d)) \quad (7)$$

The antecedent of $\Delta_{\delta}$ is the tuple obtained in Step 1, and the constraint store $c_0$ in the premise is set as $c_0 \equiv c_0 \land p = 1 \land v = 0$.

In the second premise $\phi_0 \models c_0$, a continuous trajectory $\phi_0$ is obtained with respect to the IVP-ODE stored in $c_0$. In the last premise of Rule (7), we check that the closure of transitions $\leadsto$ does not change the program $P_0$ for every $t' \in (0, \delta)$. For example, at time $t' = 1$, we have a state $(P_0, \phi_0(1))$, where $\phi_0(1) = (1/2, 1)$. 


As the result of transitions \( \sim \), we have the quadruple 
\( \langle P_a, c_{t'}, \epsilon, d \rangle \), where
\[
c_{t'} \equiv \text{cont}(p) \land x_\epsilon = (1/2, 1),
\]
and we can confirm that the first element is unchanged from the antecedent.

3. We describe another successive continuous transition \( \delta' \) before the transition in Step 2. We again assume a sufficiently small \( \delta' \in \mathbb{R}_{>0} \). The transition is formalized as follows:

\[
\forall t' \in (0, \delta')
\langle (P_a, P_b), (x = h(\delta)), \epsilon, c \rangle 
\rightarrow \left( (P_a, \phi(t')), \land x_\epsilon = \phi(\delta(t')), \epsilon, d \right)
\]

\[
\phi \models c_3, \delta' \in \mathbb{R}_{>0}, \sim \left( \langle P_a, c_{t'}, \epsilon, d \rangle \right)
\]

\[
\langle (P_a, P_b), \phi(\delta) \rangle \xrightarrow{\delta'} \left( (P_a, P_b), \phi(\delta') \right) \tag{7}
\]
The reductions are computed as in Step 2. In the first premise, we compute the constraint store \( c_3 \), where \( \delta \) is the duration evolved in Step 2. When \( \delta = 1 \),
\[
c_3 \equiv c_{e} \land p = 1/2 \land v = 1.
\]
The continuous trajectory \( \phi_0 \) is equivalent to \( \phi_0 \) in Step 2 except that the initial time is shifted for \( \delta = 1 \).

4. Assume the duration \( \sqrt{2} \), and the value \( \phi(\sqrt{2}) = (0, -\sqrt{2}) \) of the trajectory \( \phi_0 \) obtained in Step 2. The guard constraint \( P_0 \), \( \phi_0 = 0 \) is satisfied as the first element of \( \phi(\sqrt{2}) \) to \( P_0 \). We can consider the continuous transition \( \sqrt{2} \) from the initial state at time 0 as in Step 2. The time set \( (0, \sqrt{2}) \) examined in the fourth premise of Rule (7) is the maximal set in this phase of continuous transitions because a discrete change causes within the time set \( (0, \sqrt{2}) \), where \( \delta'' > \sqrt{2} \).

5. Next, we assume the duration \( \sqrt{2} \), and consider again the continuous transition \( \sim \) from the initial state at time 0. The fourth premise in Rule (7) is checked by computing Rule (8) for \( t' \in (0, \sqrt{2}) \). For \( t' = \sqrt{2} \), the transition is applied as
\[
(P_a, (x = \phi(t') \land x_\epsilon = \phi(t')), \epsilon, d) \sim \langle P_a', c_{t'}, \epsilon, d \rangle
\]
The process \( P_a \) in the antecedent is changed into \( P_a' \):

\begin{enumerate}
\item \text{cont}(p)
\end{enumerate}

An ask process in \( P_a \) disappears by Rule (2) because the constraint \( x_\epsilon = \phi(t') \) entails the guard constraint \( p_- = 0 \). Accordingly, the transition \( \sim \) from the initial state is not possible.

6. We adopt the instantaneous transition at time \( t_1 = \sqrt{2} \) as follows:
\[
\langle (P_a, P_b), (x = \phi(t')), \epsilon, d \rangle \sim \langle P_a', c_{t_1}, P_b, d \rangle,
\]

\[
v_{t_1} \models c_{t_1} \tag{8}
\]
The resulting constraint store is as follows:
\[
c_{t_1} \equiv \text{cont}(p) \land v = \sqrt{2}/2,
\]

Figure 4: Thermostat modeled by Tiny HCC.

and the value \( v_{t_1} \) is evaluated by the premise \( v_{t_1} \models c_{t_1} \) as follows:
\[
v_{t_1} = (0, \sqrt{2}/2),
\]
where \( \text{cont}(p) \) assigns the value in the previous state (i.e., \( \phi(1) \)) to the variable \( p \).

7. The resulting state in Step 6 evolves over time \( t_1 + \delta \) (\( \delta \in \mathbb{R}_{>0} \)) with respect to Rule (7) as in Step 2.

\[
(P_0, v_{t_1}) \xrightarrow{\delta'} (P_a, P_b, \phi_1(\delta))
\]
The following transitions are formalized in the same way as described above.

6. DISCUSSION

It is allowed to describe (guarded) constraints separately in a Tiny HCC program. At every instant, the transition relation \( \sim \) described by Rules (1)–(6) computes a constraint store that accumulates activated constraints in the program. In contrast, an HA explicitly associates a set of constraints to a discrete state.

Another aspect of Tiny HCC is that we handle several kinds of constraints as described in Section 2.1, and their combinations. Accordingly, we can consider models that are translated slightly differently from the bouncing particle. For example, a model of thermostat illustrated in Figure 4 has two ask processes involving continuous constraints. A discrete change occurs corresponding to a switching of the two continuous constraints.

Yet another reason is that, in an execution of Tiny HCC, a discrete change occurs at the earliest state satisfying a guard constraint, i.e., an instantaneous transition should take place whenever the premises of Rule (8) hold.

7. REFERENCES


