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

Daisuke Ishii Kazunori Ueda
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 $p, v : \mathbb{R}_{\geq 0} \to \mathbb{R}$. 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 $dp(\tau)/d\tau$ by $\dot{p}$). A contact of the particle with the ground is detected by the constraint $p_- = 0$ at line 5 (we express the left-hand limit $\lim_{t \uparrow \tau} p(t)$ of $p(\tau)$ by p_-), 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: p > 0,
2: hence {
3:    cont(p),
4:    if p > 0 then {ṗ = v, v̇ = −1},
5:    if p_− = 0 then v = −½v_− }
```

Figure 1: Bouncing particle in Tiny HCC.

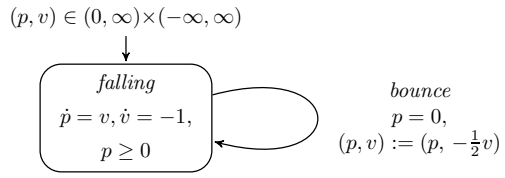

Figure 2: Bouncing particle in HA.

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:

$$\begin{array}{rl} \text{(process)} & P ::= C \mid \texttt{if } C \texttt{ then } P \mid P, P \mid \texttt{hence } P \mid \epsilon \\ \text{(constraint)} & C ::= A \mid \mathit{cont}(V) \\ \text{(expression)} & A ::= \text{an arithmetic expression involving } V \\ \text{(variable)} & V ::= \mathit{id} \mid \dot{\mathit{id}} \mid \mathit{id}_- \end{array}$$

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 $\dot{id}$;
- *continuous constraints* of the form A involving only pure identifiers and $\dot{id}$ as variables; or
- constraints of the form $\mathit{cont}(V)$ 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, $\texttt{if } C \texttt{ then } P$, 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_g). When the entailment relation between the constraint store and the guard constraint c_g held by an ask process is changed, a discrete change is triggered. In this study, we consider the ask processes $\texttt{if } c_g \texttt{ then } P$ formed as follows:

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

A *hence* process $\texttt{hence } P$ 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 $\langle \mathcal{S}, \mathcal{T}, \mathcal{S}_0 \rangle$, where $\mathcal{S}$ is a state space, $\mathcal{T} \subseteq \mathcal{S} \times \mathbb{R}_{\geq 0} \times \mathcal{S}$ is the time transition relation, and $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states. A transition $(s, t, s') \in \mathcal{T}$ is also denoted by $s \xrightarrow{t} s'$. The relation satisfies the following conditions:

$$\langle c, c', H, m \rangle \rightsquigarrow \langle \epsilon, c' \wedge c, H, m \rangle \quad (1)$$

$$\frac{c \vdash c_g}{\langle (\texttt{if } c_g \texttt{ then } P), c, H, m \rangle \rightsquigarrow \langle P, c, H, m \rangle} \quad (2)$$

$$\frac{\langle P_1, c, H, m \rangle \rightsquigarrow \langle P_1', c', H', m \rangle}{\langle (P_1, P_2), c, H, m \rangle \rightsquigarrow \langle (P_1', P_2), c', H', m \rangle} \quad (3)$$

$$\langle (\texttt{hence } P), c, H, \mathbf{c} \rangle \rightsquigarrow \langle P, c, (H, \texttt{hence } P), \mathbf{c} \rangle \quad (4)$$

$$\langle (\texttt{hence } P), c, H, \mathbf{d} \rangle \rightsquigarrow \langle \epsilon, c, (H, \texttt{hence } P), \mathbf{d} \rangle \quad (5)$$

$$\langle \mathit{cont}(x_i), c, H, m \rangle \rightsquigarrow \langle \mathit{cont}(x_i), c \wedge \mathit{cont}(x_i), H, m \rangle \quad (6)$$

$$\frac{\begin{array}{c} \langle H, (x = v), \epsilon, \mathbf{c} \rangle \overset{*}{\rightsquigarrow} \langle P, c, H', \mathbf{c} \rangle, \quad \phi \models c, \\ t \in \mathbb{R}_{>0}, \ \forall t' \in (0, t) \\ (\langle P, (x = \phi(t') \wedge x_- = \phi(t')), \epsilon, \mathbf{d} \rangle \overset{*}{\rightsquigarrow} \langle P, c', \epsilon, \mathbf{d} \rangle) \end{array}}{\langle H, v \rangle \xrightarrow{t} \langle (P, H'), \phi(t) \rangle} \quad (7)$$

$$\frac{\langle P, (x_- = v_-), \epsilon, \mathbf{d} \rangle \overset{*}{\rightsquigarrow} \langle P', c, H, \mathbf{d} \rangle, \quad v \models c}{\langle P, v_- \rangle \xrightarrow{0} \langle H, v \rangle} \quad (8)$$

Figure 3: Translation rules from Tiny HCC into RTTSs.

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

In an RTTS, we describe a discrete change from a state $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ by $s \xrightarrow{0} s'$, 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 $\langle \mathcal{S}, \mathcal{T}, \mathcal{S}_0 \rangle$ 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 $\mathcal{S}$ of the RTTS and states internally used in transitions:

- Pairs $\langle P, v \rangle$ consisting of a Tiny HCC program P and a value v in the continuous state space $\mathbb{R}^n$. $\mathcal{S}$ is a set of these pairs.
- Quadruples $\langle P, c, H, m \rangle$ 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 $\{\mathbf{c}, \mathbf{d}\}$.

Before defining the transition relation $\mathcal{T}$, we define a transition relation $\langle P, c, H, m\rangle \rightsquigarrow \langle P', c', H', m\rangle$ 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_g 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 $cont(x_i)$, 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 $\mathcal{T}$ by Rules (7) and (8).

- Rule (7) describes continuous changes expressed as the transition $\langle H, v\rangle \xrightarrow{t} \langle (P, H'), \phi(t)\rangle$. 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', 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 ϕ 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 $\mathcal{S}_0$ of the RTTS. Assume we have $\langle P, \bullet\rangle$, where P is an input program and $\bullet$ is a special value denoting the undefined value. In the premise, we compute $\langle P, \mathbf{true}, \epsilon, \mathbf{d}\rangle \overset{*}{\rightsquigarrow} \langle P', c, H, \mathbf{d}\rangle$, 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 $\mathcal{S}_0$ is obtained as $\langle H, v\rangle$, 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 $\langle P, (x_- = v_-), \epsilon, \mathbf{d}\rangle \overset{*}{\rightsquigarrow} \langle P', c, H, \mathbf{d}\rangle$ 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 $\langle H, v\rangle$ 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_{in}. We abbreviate sub-processes in P_{in} to P_a and P_h as follows:

- P_a :
  `1:` $cont(p)$`,`
  `2: if` $p_- = 0$ `then` $v = -\frac{1}{2}v_-$
- P_h :
  `1: hence {`
  `2:` P_a`,`
  `3: if` $p > 0$ `then {`$\dot{p} = v$`,` $\dot{v} = -1$`} }`

We also abbreviate the following constraint to c_c:

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

The execution of the program proceeds as follows:

1. We first obtain an initial state at time 0 (i.e., a state $s^0 \in \mathcal{S}_0$ of the associated RTTS). We apply Rule (8) to a state $\langle P_{in}, \bullet\rangle$, where $\bullet$ denotes the undefined value, as follows:

$$\frac{\langle P_{in}, \mathbf{true}, \epsilon, \mathbf{d}\rangle \overset{*}{\rightsquigarrow} \langle \epsilon, c_0, P_h, \mathbf{d}\rangle,\ v_0 \models c_0}{\langle P_{in}, \bullet\rangle \xrightarrow{0} \langle P_h, v_0\rangle} \quad (8)$$

   The first premise is proved by applying the rules $\rightsquigarrow$ in Figure 3 as many times as possible. As a result, we obtain a quadruple $\langle \epsilon, c_0, P_h, \mathbf{d}\rangle$, where

$$c_0 \ \equiv\ p > 0,$$

   and P_h is the hence process in P_{in}. 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 $\xrightarrow{\delta}$, where $\delta \in \mathbb{R}_{>0}$ is a duration for an evolution (we assume a sufficiently small δ that causes no discrete change). The transition is formalized as follows:

$$\frac{\begin{array}{ll} & \forall t' \in (0, \delta) \\ \langle P_h, (x = v_0), \epsilon, \mathbf{c}\rangle & (\langle P_a, (x = \phi_0(t') \\ \overset{*}{\rightsquigarrow} \langle P_a, c_0, P_h, \mathbf{c}\rangle, & \wedge x_- = \phi_0(t')), \epsilon, \mathbf{d}\rangle \\ \phi_0 \models c_0,\ \delta \in \mathbb{R}_{>0}, & \overset{*}{\rightsquigarrow} \langle P_a, c_{t'}, \epsilon, \mathbf{d}\rangle) \end{array}}{\langle P_h, v_0\rangle \xrightarrow{\delta} \langle (P_a, P_h), \phi_0(\delta)\rangle} \quad (7)$$

   The antecedent of $\xrightarrow{\delta}$ is the tuple obtained in Step 1, and the constraint store c_0 in the premise is set as

$$c_0 \ \equiv\ c_c \wedge p = 1 \wedge v = 0.$$

   In the second premise $\phi_0 \models c_0$, a continuous trajectory ϕ_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 $\overset{*}{\rightsquigarrow}$ does not change the program P_a for every $t' \in (0, \delta)$. For example, at time $t' = 1$, we have a state $\langle P_a, \phi_0(1)\rangle$, where $\phi_0(1) = (1/2, 1)$.

As the result of transitions $\overset{*}{\rightsquigarrow}$, we have the quadruple $\langle P_a, c_{t'}, \epsilon, \mathbf{d}\rangle$, where

$$c_{t'} \;\equiv\; cont(p) \wedge x_- = (1/2, 1),$$

and we can confirm that the first element is unchanged from the antecedent.

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

$$\frac{\begin{array}{ll} & \forall t' \in (0, \delta') \\ \langle (P_a \,,P_h), (x = \phi_0(\delta)), \epsilon, \mathbf{c}\rangle & (\langle P_a, (x = \phi_\delta(t') \\ \quad \overset{*}{\rightsquigarrow} \langle P_a, c_\delta, P_h, \mathbf{c}\rangle, & \wedge x_- = \phi_\delta(t')), \epsilon, \mathbf{d}\rangle \\ \phi_\delta \models c_\delta,\ \delta' \in \mathbb{R}_{>0}, & \overset{*}{\rightsquigarrow} \langle P_a, c_{t'}, \epsilon, \mathbf{d}\rangle) \end{array}}{\langle (P_a \,,P_h), \phi_0(\delta)\rangle \overset{\delta'}{\rightarrow} \langle (P_a \,,P_h), \phi_\delta(\delta')\rangle} \quad (7)$$

The reductions are computed as in Step 2. In the first premise, we compute the constraint store c_δ, where δ is the duration evolved in Step 2. When $\delta = 1$,

$$c_\delta \;\equiv\; c_c \wedge p = 1/2 \wedge v = 1.$$

The continuous trajectory ϕ_δ is equivalent to ϕ_0 in Step 2 except that the initial time is shifted for $\delta = 1$.

4. Assume the duration $\sqrt{2}$, and the value $\phi_0(\sqrt{2}) = (0, -\sqrt{2})$ of the trajectory ϕ_0 obtained in Step 2. The guard constraint $p_- = 0$ is satisfied by assigning the first element of $\phi_0(\sqrt{2})$ to p_-. We can consider the continuous transition $\overset{\sqrt{2}}{\rightarrow}$ 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 $(0, \delta'')$, where $\delta'' > \sqrt{2}$.

5. Next, we assume the duration $\delta'' > \sqrt{2}$, and consider again the continuous transition $\overset{\delta''}{\rightarrow}$ from the initial state at time 0. The fourth premise in Rule (7) is checked by computing Rule (8) for $t' \in (0, \delta'')$. For $t' = \sqrt{2}$, the transition is applied as

$$\langle P_a, (x = \phi_0(t') \wedge x_- = \phi_0(t')), \epsilon, \mathbf{d}\rangle \overset{*}{\rightsquigarrow} \langle P'_a, c_{t'}, \epsilon, \mathbf{d}\rangle$$

The process P_a in the antecedent is changed into P'_a:

```
1: cont(p)
```

An ask process in P_a disappears by Rule (2) because the constraint $x_- = \phi_0(t')$ entails the guard constraint $p_- = 0$. Accordingly, the transition $\overset{\delta''}{\rightarrow}$ from the initial state is not possible.

6. We adopt the instantaneous transition at time $t^1 = \sqrt{2}$ as follows:

$$\frac{\langle (P_a \,,P_h), (x_- = \phi_0(t^1)), \epsilon, \mathbf{d}\rangle \overset{*}{\rightsquigarrow} \langle P'_a, c_{t^1}, P_h, \mathbf{d}\rangle, \quad v_{t^1} \models c_{t^1}}{\langle (P_a \,,P_h), \phi_0(t^1)\rangle \overset{0}{\rightarrow} \langle P_h, v_{t^1}\rangle} \quad (8)$$

The resulting constraint store is as follows:

$$c_{t^1} \;\equiv\; cont(p) \wedge v = \sqrt{2}/2,$$

```
1: hence {
2:    ṗ = v, cont(p),
3:    if p ≥ 10 then v̇ = −1,
4:    if p < 10 then v̇ = 1 }
```

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 $cont(p)$ assigns the value in the previous state (i.e., $\phi_0(t^1)$) 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.

$$\langle P_h, v_{t^1}\rangle \overset{\delta}{\rightarrow} \langle (P_a \,,P_h), \phi_{t^1}(\delta)\rangle$$

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 $\rightsquigarrow$ 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

[1] B. Carlson and V. Gupta. The hcc Programmer's Manual. http://www-cs-students.stanford.edu/~vgupta /hcc/papers.html, 1997.

[2] P. J. L. Cuijpers and M. A. Reniers. Lost in Translation: Hybrid-Time Flows vs Real-Time Transitions. *Proc. HSCC'08*, *LNCS* 4981, pages 116–129, 2008.

[3] M. Falaschi, A. Policriti, and A. Villanueva. Time Limited Model Checking. *Proc. SAVE'01*, 2001.

[4] V. Gupta, R. Jagadeesan, and V. A. Saraswat. Hybrid cc, Hybrid Automata and Program Verification. *Hybrid Systems III*, *LNCS* 1066, pages 52–75, 1996.

[5] V. Gupta, R. Jagadeesan, and V. A. Saraswat. Computing with Continuous Change. *Science of Computer Programming*, 30(1-2):3–49, 1998.

[6] T. A. Henzinger. The Theory of Hybrid Automata. *Verification of Digital and Hybrid Systems*, 170:265–292, 2000.

[7] A. J. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*, *LNCIS* 251. Springer-Verlag, 2000.