March 2004

# Sound Code Generation from Communicating Hybrid Models

Yerang Hur
*University of Pennsylvania*

Jesung Kim
*University of Pennsylvania*

Insup Lee
*University of Pennsylvania*, lee@cis.upenn.edu

Jin-Young Choi
*Korea University*



Postprint version. Published in *Lecture Notes in Computer Science*, Volume 2993, Hybrid Systems: Computation and Control: 7th International Workshop, 2004 (HSCC 2004), pages 432-447.
Publisher URL: http://dx.doi.org/10.1007/b96398

# Sound Code Generation from Communicating Hybrid Models

**Abstract**
Precise translation from hybrid models to code is difficult because models are defined in the continuous-time domain whereas code executes on digital computers in a discrete fashion. Traditional approach is to associate the model with a sampling rate before code generation, and rely on an approximate algorithm that computes the next state numerically. Depending on the choice of the sampling rate and the algorithm, the behavior of the code may vary significantly due to numerical errors, but the discrepancy has been addressed informally, making the analysis results at the model level less meaningful for implementation. Formal relationship between the model and the code becomes even more unclear when components of the code execute concurrently. In this paper, we propose a formal framework that addresses the issue of soundness of concurrent programs generated from communicating hybrid models. The motivation is that concurrent programs executing in different rates may cause an erroneous transition when transition conditions are evaluated using values from different time instances. The essence of our technique is to refine the model by tightening transition conditions according to the maximum errors due to different sampling rates. We claim that the generated code has a trace of discrete transitions that is equivalent to one of the traces observable from the model, and that the values of variables are bounded. Our framework demonstrates how hybrid models defined in the continuous time domain are translated into discretized models with or without consideration of errors due to asynchronous sampling, and finally into executable code with real-time scheduling.

**Keywords**
Communicating hybrid systems, automatic code generation, soundness

**Comments**
Postprint version. Published in *Lecture Notes in Computer Science*, Volume 2993, Hybrid Systems: Computation and Control: 7th International Workshop, 2004 (HSCC 2004), pages 432-447.
Publisher URL: http://dx.doi.org/10.1007/b96398

# Sound Code Generation from Communicating Hybrid Models

Yerang Hur, Jesung Kim, and Insup Lee
*Department of Computer and Information Science*
*University of Pennsylvania, Philadelphia, PA, USA*
Jin-Young Choi
*Department of Computer Science and Engineering*
*Korea University, Seoul, Korea*

**Abstract.** Precise translation from hybrid models to code is difficult because models are defined in the continuous-time domain whereas code executes on digital computers in a discrete fashion. Traditional approach is to associate the model with a sampling rate before code generation, and rely on an approximate algorithm that computes the next state numerically. Depending on the choice of the sampling rate and the algorithm, the behavior of the code may vary significantly due to numerical errors, but the discrepancy has been addressed informally, making the analysis results at the model level less meaningful for implementation. Formal relationship between the model and the code becomes even more unclear when components of the code execute concurrently. In this paper, we propose a formal framework that addresses the issue of soundness of concurrent programs generated from communicating hybrid models. The motivation is that concurrent programs executing in different rates may cause an erroneous transition when transition conditions are evaluated using values from different time instances. The essence of our technique is to refine the model by tightening transition conditions according to the maximum errors due to different sampling rates. We claim that the generated code has a trace of discrete transitions that is equivalent to one of the traces observable from the model, and that the values of variables are bounded. Our framework demonstrates how hybrid models defined in the continuous time domain are translated into discretized models with or without consideration of errors due to asynchronous sampling, and finally into executable code with real-time scheduling.

## 1 Introduction

A model-based approach is an emerging paradigm for developing robust software, and has been the focus of increasing research effort. Models are used during the design phase to ensure systems under consideration have desired properties. Benefits of high-level modeling can be significantly improved if the code is generated automatically from the model. However, precise translation from models to code is difficult especially when the model is based on hybrid systems. Hybrid models combine continuous state change specified by differential equations with discrete state transition specified by the finite state machine. Formally, a hybrid model consists of a vector $x = (x_1, x_2, \ldots, x_n)$ of (continuously updated) variables, a finite set of discrete states $P$ that associates $x$ with a differential

equation $\dot{x} = f_p(x)$ for each $p \in P$, a set of transitions $E \subseteq P \times P$, a guard set $G((p,p')) \subseteq \mathbb{R}^n$ for each $(p,p') \in E$ specifying the condition that the transition $(p,p')$ can be taken, and an invariant set $I(p) \subseteq \mathbb{R}^n$ for each $p \in P$ specifying the condition that $x$ follows $\dot{x} = f_p(x)$. Hypothetically, precise implementation of a hybrid model would be possible if there exists a program that computes valuation $x(t)$ of $x$ at time $t \in \mathbb{R}$ according to the dynamics $\dot{x} = f_p(x)$ defined at the current discrete state $p$, and decides the next state $p'$ according to the invariant set $I(p)$ and the guard set $G((p,p'))$, all in infinitesimal time. Of course, such a program is not feasible in digital computers, since it requires indefinite computation power and precision.

A traditional approach to automatic code generation from hybrid models is to associate the model with a sampling rate, and generate code that computes the state of the model at the given rate approximately by using a numerical method (e.g., Runge-Kutta method). Depending on the choice of the sampling rate and the numerical method, the behavior of the synthesized code may vary significantly. However, reasoning on the discrepancy between the model and the code is oftentimes left to the designer's intuition and/or ad hoc experiments. In the case of the code generated from a model consisting of multiple components with different sampling rates, the issue of formal relationship between the model and the code becomes even more unclear. Thus, analysis results obtained at the model level is less useful to the generated code. The desire to bridge this gap motivates our research.

Formally, we can define a discrete-time abstraction of a given hybrid model $A$ over a discrete time domain $T = \{t_i | t_i \in \mathbb{R}, i = 0, 1, 2, \ldots, t_i < t_{i+1}\}$, denoted by $A/T$, as an extended finite state machine with an equivalent set of variables $x_T$ and discrete states $P_T$, and $x_T(t_{i+1}) = x_T(t_i) + \int_{t_i}^{t_{i+1}} f_p(x_T)\, dt$. $A/T$ abstracts $A$, that is, $x_T(t) = x(t)$ for all $t \in T$, provided that $A$ satisfies $x(t') \in I(p_i)$ for all $t' \in [t_i, t_{i+1}]$ for all $i \geq 0$. An implementation $\mathrm{prog}(A/T)$ of $A/T$, then, is a program that computes $x(t_{i+1})$ based on a routine $\mathrm{prog}(f_p)$ that solves the equation $x(t_{i+1}) = x(t_i) + \int_{t_i}^{t_{i+1}} f_p(x)\, dt$ numerically, and determines the next discrete state $p'$, on or before time $t_{i+1}$. $\mathrm{prog}(A/T)$ is a precise implementation of $A/T$ if (1) a precise algorithm $\mathrm{prog}(f_p)$ to solve $\int_{t_i}^{t_{i+1}} f_p(x)\, dt$ is provided, and (2) execution of the algorithm and decision of the next discrete state can be done within the time constraint $(t_{i+1} - t_i)$. The latter requirement is a classic real-time computing problem. In a special case where $t_{i+1} - t_i = h$ for all $i \geq 0$, $\mathrm{prog}(A/T)$ can easily be mapped to a periodic task of the RTOS with a period $h$. On the other hand, the former requirement can be satisfied only for a limited class of differential equations (e.g., zero-order differential equations) and a sampling rate carefully selected to avoid floating-point errors. The effect of numerical errors in hybrid models can be significant since discrete transitions based on erroneous values may lead to entirely different trajectory.

When the model consists of multiple components and the code is generated such that components are mapped to concurrent tasks, in addition to the aforementioned requirements, tasks need to be synchronized with each other. That is, given a composite hybrid model $A = A_1 \| A_2 \| \ldots \| A_n$, $\mathrm{prog}(A/T)$

would consist of concurrent programs $\text{prog}(A_1/T), \text{prog}(A_2/T), \ldots, \text{prog}(A_n/T)$ such that each program $\text{prog}(A_l/T)$ executes (1) a numerical algorithm to solve $x_{A_l}(t_{i+1}) = x_{A_l}(t_i) + \int_{t_i}^{t_{i+1}} f_{p_{A_l}}(x_{A_l})\, dt$ where $x_{A_l}$ is a vector of variables whose dynamics is constrained by $A_l$, (2) wait for $A_m$ for all $m \neq l$ to finish computation of $x_{A_m}(t_{i+1})$, and (3) decides the next discrete state. This requirement may be costly if the programs are executed in a distributed system where communication is an expensive operation. Moreover, formal relationship between the model and the code is not obvious when the concurrent tasks are executing at different rates. Modern modeling tools such as SIMULINK, for example, support code generation with different sampling rates (assuming they are harmonic) for different components of a single model to improve the CPU utilization, but there have been no formalism on the semantic relationship between the implementation and the model. In general, a set of concurrent programs $\text{prog}(A_1/T_1), \text{prog}(A_2/T_2), \ldots, \text{prog}(A_n/T_n)$ for $A = A_1 \| A_2 \| \ldots . \| A_n$, does not implement $A/T$ precisely for any $T$ if $T_l \neq T_m$ for some $m, l$. The reason is that, to evaluate a guard set $G(p, p')$ of $A_l$ at time $t$, valuation $x_m(t)$ should be available for all $1 \leq m \leq n$, which is true only when $T_1 = T_2 = \ldots = T_n$. Evaluation of guard sets based on valuation from different time instances may lead to an erroneous discrete transition that is not allowed in the hybrid model. For example, suppose a hybrid model consisting of two variables $x_1, x_2$ and the code that is generated such that the two variables are updated at different rates. At time $t$, the generated code may have valuation $x_1(t)$ of $x_1$ at time $t$, but, for $x_2$, only valuation $x_2(t')$ at time $t' < t$ may be available. Suppose again, that $(x_1(t), x_2(t')) \in G((p, p'))$ for some $p'$ where $p$ is the discrete state at time $t$. In this case, the generated code may take the transition $(p, p')$ since it *appears* to be enabled. However, if the hybrid model indicates $x(t) = (x_1(t), x_2(t)) \notin G((p, p'))$, the transition should not be taken at time $t$, implying the generated code is not consistent to the model. We call this type of errors *synchronization errors.*

In this paper, we propose a framework aiming at sound code generation from Communicating hybrid models that prevents synchronization errors. Our approach is based on a model refinement technique that we call *instrumentation.* Instrumentation replaces the guard set and the invariant set with their subsets to exclude valuations that may potentially lead to an erroneous discrete transition due to different valuation times of variables. Note that a model obtained by instrumentation is subsumed by the original hybrid model, that is, every possible behavior of the instrumented model is also a valid behavior of the original hybrid model. This comes from the semantics of hybrid automata where transitions *may* be taken when the associated enabling condition (i.e., *guard*) is true (that is, the transition condition is an enabler, rather than a trigger). Therefore, the behavior observed from the code generated from the instrumented model is guaranteed to belong to (a discrete-time abstraction of) the original hybrid model.

Figure 1 shows overall flow of our framework. We start with Communicating Hybrid Automata defined in the continuous time domain. Associating a discrete time domain to Communicating Hybrid Automata defines *Discretized Communicating Hybrid Automata.* States of Discretized Communicating Hybrid

Automata are defined only at the time instances belonging to the given time domain. Discretized Communicating Hybrid Automata generate exact snapshots of Communicating Hybrid Automata in a discrete fashion, under a condition that will be explained in Section 3. *Instrumented Communicating Hybrid Automata* allows heterogeneous discrete time domains between components. States of Instrumented Communicating Hybrid Automata are defined at the time instances belonging to the union of all the discrete time domains of the components. We claim that, for every trace of discrete transitions of Instrumented Communicating Hybrid Automata, there exists an equivalent trace of discrete transitions in Communicating Hybrid Automata. We also claim that the deviation of valuation at every discrete transition is bounded.

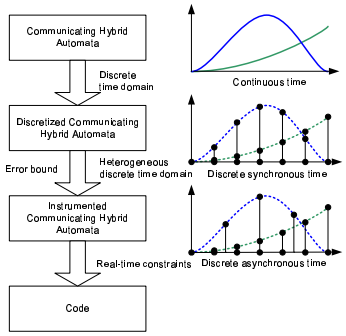


**Fig. 1.** Design flow.

The remainder of the paper is organized as follows. The next section gives a formal description of Communicating Hybrid Automata and the semantics. In the next, we present Discretized Communicating Hybrid Automata and their relationship with Communicating Hybrid Automata. The following is a detailed description of Instrumented Communicating Hybrid Automata with a motivational example. Additional issues that need to be considered when the instrumented model is converted into code are described in the next section. Finally, concluding remarks are given in the last section.

## 2 Communicating Hybrid Automata

This section defines Communicating Hybrid Automaton (*CHA*) as an extension of the timed automaton [1] to include shared variable-based communication. To simplify discussion, we limit our attention to Communicating Hybrid Automata with independent dynamics and guard/invariant sets specified by intervals.

**Definition 1.** *(CHA). A Communicating Hybrid Automaton, CHA, is a tuple* $A = (P, VC, SV, p_0, F, E, I, G, R, INIT)$*, where*

- $P$ *is a finite set of distinct positions,*
- $VC$ *is a finite set of continuous real variables, where* $|VC| = n$*,*
- $SV \subseteq VC$ *is a non-empty finite set of shared variables which are partitioned into input* $SV|_{in}$ *and output* $SV|_{out}$*,*
- $p_0 \in P$ *is the initial position,*

- $F\colon P \to \mathcal{F}$ *assigns to* $p \in P$ *a function* $F_p \in \mathcal{F} : \mathbb{R}^n \to \mathbb{R}^n$, *which defines ordinary differential equations satisfying the assumptions for existence and uniqueness of solutions for all variables in* $VC - SV|_{in}$,
- $E \subseteq P \times P$ *is a finite set of discrete transitions,*
- $I\colon P \to (2^{\mathbb{R}})^n$ *assigns the invariant interval to* $p \in P$ *such that* $I(p) \in (2^{\mathbb{R}})^n$ *and, for all* $x \in VC$, *we denote the invariant interval of* $x$ *at the position* $p$ *by* $I_x(p)$,
- $G\colon E \to (2^{\mathbb{R}})^n$ *assigns to* $(p_1, p_2) \in E$ *the guard interval such that for all* $x \in VC$, $G_x((p_1,p_2)) \cap I_x(p_1) \neq \emptyset$, *where* $G_x((p_1,p_2))$ *denotes the guard interval of* $x$,
- $R\colon E \times VC \to \mathbb{R}$ *assigns a reset value* $R((p_1,p_2),x) \in I_x(p_2)$ *to a pair* $(p_1,p_2) \in E$ *and* $x \in VC - SV|_{in}$, *and*
- $INIT\colon VC \to \mathbb{R}$ *assigns to a variable the initial value satisfying* $INIT(x) \in I_x(p_0)$, *for all* $x \in VC - SV|_{in}$. □

In the rest of the paper, we denote $P$ of $A$ by $P_A$. Likewise, we use $VC_A$, $p_0^A$, $F_A$, $E_A$, $I_A$, $G_A$, $R_A$, and $INIT_A$ to denote $VC$, $p_0$, $F$, $E$, $I$, $G$, $R$, and $INIT$ of $A$, respectively. For all $x \in VC_A$, the invariant and the guard intervals of $x$ are denoted by $I_{A,x}$ and $G_{A,x}$, respectively. When it is clear, we omit $A$.

**Definition 2.** *(State of a CHA). Given a CHA* $A$, *a (time-stamped) state* $s = (p,u,t)$ *is an element of* $P_A \times \mathbb{R}^n \times \mathbb{R}$ *satisfying the following condition: at time* $t$, *for all* $x \in VC$, $u(x) \in I_x(p)$, *where* $u(x)$ *is the valuation of* $x$. □

A state $(p,u,t)$ means that at time $t$ the system is at the position $p$ with the valuation $u$. When a state $s_i = (p_i, u_i, t_i)$ is given, we use $s_i|_p, s_i|_u, s_i|_t$ to denote $p_i, u_i, t_i$, respectively. In addition, we use $u \in I_A(p)$ if $u(x) \in I_{A,x}(p)$ for all $x \in VC$, and $u \in G_A((p_1,p_2))$ if $u(x) \in G_{A,x}((p_1,p_2))$ for all $x \in VC$.

**Definition 3.** *(Discrete transition step of a CHA). Given a CHA* $A$, *a pair of states* $(s_i, s_j)$ *is called a discrete transition step if the following conditions are satisfied:*

- $s_i|_t = s_j|_t$,
- $(s_i|_p, s_j|_p) \in E_A$,
- $s_i|_u \in G_A((s_i|_p, s_j|_p))$, *and*
- $s_j|_u(x) = R_A((s_i|_p, s_j|_p), x)$, *for all* $x \in VC - SV|_{in}$. □

The value of input variable is defined later in Definition 7. Note that a discrete transition is of the form $(p_m, p_n)$, where $p_m$, $p_n \in P$, i.e., a directed edge from the node $p_m$ to the node $p_n$, whereas a discrete transition step is $(s_i, s_j)$, where $s_i$ and $s_j$ are states defined in Definition 2.

**Definition 4.** *(Continuous transition step of a CHA). Given a CHA* $A$, *a pair of states* $(s_i, s_j)$ *is called a continuous transition step if the following conditions are satisfied:*

- $s_i|_t < s_j|_t$,
- $s_i|_p = s_j|_p$,

– *for all* $t \in [s_i|_t, s_j|_t]$, $x(t) \in I_x(s_i|_p)$, *for all* $x \in VC$, *and*
– *for all* $t \in [s_i|_t, s_j|_t]$, $dx(t)/dt$ *is the same as the dynamics defined by* $F_{s_i|_p}$, *for all* $x \in VC - SV|_{in}$. □

The value of input variable is defined later in Definition 8. The continuous transition step corresponds to the continuous flow at the position $p$ with the dynamics specified by $F_p$ from time $t_i$ to time $t_j$.

**Definition 5.** *(System of Communicating Hybrid Automata). Given a finite set of CHAs* $\{(A_0, SV_0), \ldots, (A_i, SV_i), \ldots, (A_n, SV_n)\}$, *a System of Communicating Hybrid Automata denoted by SCHA C is a tuple* $((A_0, SV_0), \ldots, (A_i, SV_i), \ldots, (A_n, SV_n))$, *such that*

– $\cup_i SV_i|_{in} \subseteq \cup_i SV_i|_{out}$ *and*
– $SV_i|_{out} \cap SV_j|_{out} = \emptyset$, *for all* $0 \leq i \neq j \leq n$. □

We will denote $C$ as $(A_0, A_1, \ldots, A_n, SV)$, where $SV = \bigcup_i SV_i|_{out}$. Note that the shared variables in $SV$ are write-exclusive.

**Definition 6.** *(State of an SCHA). Given an SCHA* $C = (A_0, A_1, \ldots, A_n, SV)$, *a state s is defined as* $((p^{A_0}, u^{A_0}), \ldots, (p^{A_n}, u^{A_n}), t)$, *where* $(p^{A_i}, u^{A_i}, t)$ *is a state of CHA A at time t satisfying that* $u^{A_i}(x) = u^{A_j}(x)$ *if* $x \in SV_i|_{in} \cap SV_j|_{out}$. □

Note that $s|_{A_i}$ denotes $(p^{A_i}, u^{A_i}, t)$, and that $s|_t$ denotes the time $t$. We will use $s|_{A_i,p}$, $s|_{A_i,u}$, and $s|_{A_i,t}$ to denote $p^{A_i}$, $u^{A_i}$, and $t$, respectively.

**Definition 7.** *(Discrete transition step of an SCHA). Given an SCHA C, a pair of states* $(s_i, s_j)$ *is called a discrete transition step if there exists* $A_m$ *such that* $(s_i|_{A_m}, s_j|_{A_m})$ *is a discrete transition step in* $A_m$, *and for all* $A_k$ *where* $(s_i|_{A_k}, s_j|_{A_k})$ *is not a discrete transition step in* $A_k$, *if the following is satisfied:*

$s_i|_{A_k,t} = s_j|_{A_k,t}$, $s_i|_{A_k,p} = s_j|_{A_k,p}$, *and*

$$s_j|_{A_k,u}(x) = \begin{cases} s_j|_{A_l,u}(x) \text{ if } x \in SV_k|_{in} \cap SV_l|_{out} \text{ for some } A_l, \\ s_j|_{A_k,u}(x) \text{ otherwise.} \end{cases}$$ □

**Definition 8.** *(Continuous transition step of an SCHA). Given an SCHA C, a pair of states* $(s_i, s_j)$ *is called a continuous transition step if the following is satisfied:*

– $s_i|_t < s_j|_t$,
– $(s_i|_{A_k}, s_j|_{A_k})$ *is a continuous transition step for all* $k \in \{0, 1, \ldots, n\}$, *and*
– $s_j|_{A_k,u}(x) = s_j|_{A_l,u}(x)$, *for all* $x \in SV_k|_{in} \cap SV_l|_{out}$. □

**Definition 9.** *(Run of an SCHA). A run of an SCHA* $C = (A_0, A_1, \ldots, A_n, SV)$ *is a (possibly infinite) sequence of states* $\langle s_0, s_1, \ldots, s_i, s_{i+1}, \ldots \rangle$, *where*

– $s_0 = ((p_0^{A_0}, INIT_{A_0}), (p_0^{A_1}, INIT_{A_1}), \ldots, (p_0^{A_n}, INIT_{A_n}), 0)$, *and*
– $(s_i, s_{i+1})$ *is either a discrete transition step or a continuous transition step for all* $i \geq 0$.

*A run is called an alternating run if discrete transition steps and continuous transition steps occurs alternately, i.e., if* $(s_i, s_{i+1})$ *is a continuous transition step, then* $(s_{i+1}, s_{i+2})$ *is a discrete transition step, and vice versa.* □

## 3 Discretized Communicating Hybrid Automata

We now give a formal definition of Discretized Communicating Hybrid Automata as the first step towards the generated code. A Discretized Communicating Hybrid Automaton is a discrete-time abstraction of a given Communicating Hybrid Automaton over a discrete time domain.

**Definition 10.** *(DCHA) A Discretized Communicating Hybrid Automaton, DCHA, is a tuple $H = (A, T)$, where*

- *$A$ is a Communicating Hybrid Automaton,*
- *$T = \{t_0, t_1, t_2, \ldots\}$ is a discrete time domain, where $t_i \in \mathbb{R}$ and $t_{i+1} > t_i$ for all $i$.* □

**Definition 11.** *(State of DCHA). Given a DCHA $H$, a (time-stamped) state $s = (p, u, t)$ is an element of $P_A \times \mathbb{R}^n \times \mathbb{R}$ satisfying the following condition: $t \in T$, and $s$ is a state of $A$.* □

Note that states of DCHA are defined only at time instances belonging to the given discrete time domain $T$. The following defines a continuous transition step of DCHA over $T$. A discrete transition step is defined similarly.

**Definition 12.** *(Continuous transition step of DCHA). Given a DCHA $H$, a pair of states $(s_i, s_j)$ is called a unit continuous transition step if the following conditions are satisfied:*

- *$s_i|_t = t_m$ and $s_j|_t = t_{m+1}$ for some $t_m, t_{m+1} \in T$,*
- *$s_i|_p = s_j|_p$,*
- *$s_i|_u, s_j|_u \in I_A(s_i|_p)$, and*
- *for all $x \in VC - SV|_{in}$, $s_j|_u(x) = s_i|_u(x) + \int_{t_m}^{t_{m+1}} F_p(x)\, dt$*

*When $\langle (s_i, s_{i+1}), (s_{i+1}, s_{i+2}), \ldots, (s_{j-1}, s_j) \rangle$ is a sequence of unit continuous transition step, we say that $(s_i, s_j)$ is a continuous transition step. When it is clear, a unit continuous transition step is also called a continuous transition step.* □

Note that a pair $(s_i, s_j)$ of states of CHA is a continuous transition step of DCHA if $(s_i, s_j)$ is a continuous transition step of CHA and $s_i|_t = t_m$, $s_j|_t = t_{m+1}$ for some $t_m, t_{m+1} \in T$. However, the contrary is not always true. That is, a pair of states constituting a continuous transition step of DCHA is not necessarily a continuous transition step of CHA. This is because that a continuous transition step of CHA requires that for all $t \in [s_i|_t, s_j|_t]$, $x(t) \in I_x(s_i|_p)$, whereas a continuous transition step of DCHA only requires $s_i|_u, s_j|_u \in I(s_i|_p)$. Thus, DCHA does not represent CHA faithfully when the dynamics changes rapidly during a short time interval such that invariant violation may occur even if the states at the endpoints of the interval satisfy the invariant. The following definition formalizes such a condition. (See [2, 3] for more general discussion on event detection problems of hybrid systems.)

**Definition 13.** *(h-insensitivity). Given a CHA $A$, the invariant $I_x(p)$ is said h-insensitive if, for all $x \in VC - SV|_{in}$, $x(t) \in I_x(p)$ and $x(t+h) = x(t) + \int_t^{t+h} F_p(x)\,dt \in I_x(p)$ implies $x(t+\delta) = x(t) + \int_t^{t+\delta} F_p(x)\,dt \in I_x(p)$ for all $\delta \in [0, h]$, where $x(t)$ denotes valuation of $x$ at time $t$. When all invariants in $A$ are h-insensitive, $A$ is said h-insensitive. We say that SCHA $C$ is h-insensitive when all CHA $A_k$ of $C$ is h-insensitive.* □

Now we give a definition of discretized SCHA.

**Definition 14.** *(discretized SCHA). Given an SCHA $C = (A_0, A_1, \ldots, A_n, SV)$ and a discrete time domain $T$, a discretized SCHA, denoted by DSCHA, is a tuple $(H_0, H_1, \ldots, H_n, SV)$, where $H_i = (A_i, T)$.* □

Note that the components of DSCHA shares the same discrete time domain. That is, we are defining a *discretized* system of CHA, rather than a *system* of discretized CHA. This issue will become clear in Section 4 when we define SICHA that allows heterogeneous time domains. Given that, a state of DSCHA and a run of DSCHA can be defined similar to the case of SCHA. The following lemma states the relation between DSCHA and SCHA.

**Lemma 1.** *Every run of DSCHA has an equivalent run in the originating SCHA, if SCHA is h-insensitive, where $h = \max(t_{m+1} - t_m), t_m, t_{m+1} \in T$.* □

## 4 Instrumented Communicating Hybrid Automata

In this section, we present the effect of accumulated numerical errors and synchronization errors in code generated from an SCHA. To formalize the effect of accumulated numerical errors and synchronization errors of an SCHA, we propose the formalism called a System of Instrumented Communicating Hybrid Automata (SICHA). We show that a run of the SICHA is always included in that of the SCHA. Thus, the SICHA provides correct execution results with regard to runs of the original model. The essence of our idea is to reflect the effect of the errors to the invariants and the guards of each position so that the resulting instrumented hybrid automata will produce a sound trace on discrete transition steps. This paper focuses on instrumenting an SCHA considering the effect of asynchrony due to discretization of a model. For the details of the effect of accumulated numerical errors, refer to the the paper [4].

**Motivating example.** Figure 2 describes the effect of a synchronization error. Let a shared variable $y \in SV|_{in}^{A_0} \cap SV|_{out}^{A_1}$ be read by $A_0$ and written by $A_1$, where the integration stepsizes of $A_0$ and $A_1$ are 0.001 and 0.002, respectively (namely, $h_{A_0} = 0.001$ and $h_{A_1} = 0.002$). In this case, $y$ is updated by $A_1$ at time 0, 0.002, 0.004, ..., and read by $A_0$ at time 0, 0.001, 0.002, 0.003, 0.004, .... In Figure 2, $A_0$ needs to decide whether or not it will take a *discrete transition step* at time $0.001 \times (2n+1)$ with a computation result using the value of $y$ produced at time $0.001 \times 2n$, where $n$ is a non-negative integer. For example, at time 0.003 $A_0$ will evaluate the transition condition with the value of $y$ produced at time 0.002. We need to decide which approximated value of $y$ $A_0$ will use in evaluating the state

of $A_0$. It relies on the effect of discrepancy between the correct value and the approximated value different from the correct one due to numerical errors and synchronization errors.

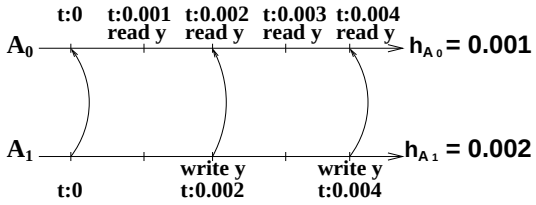


**Fig. 2.** The Effect of Synchronization Errors

Now consider an example of an SCHA. Figure 3 illustrates SCHA `foo` composed of Communicating Hybrid Automata $A_0$ and $A_1$.

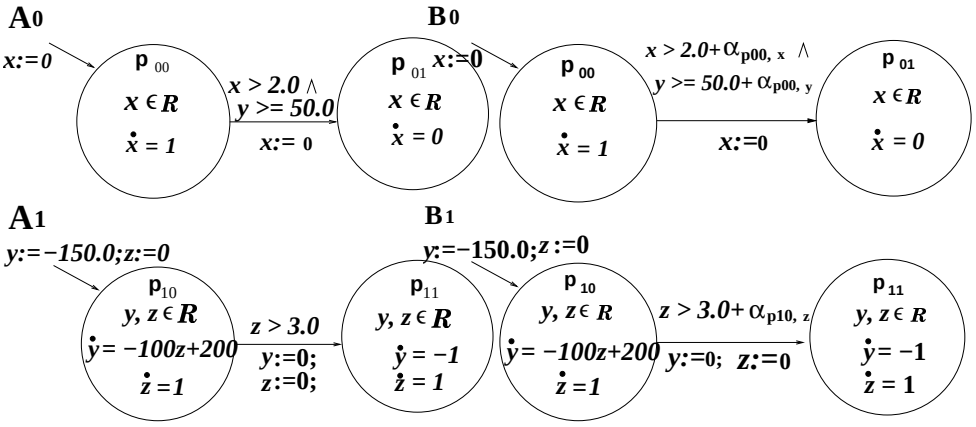


**Fig. 3.** SCHA `foo`

**Fig. 4.** Instrumented `foo`

Table 1 shows that an unexpected trace occurs during simulation of an SCHA `foo` even if we do not consider the effect of numerical errors. If we use the execution stepsize $1.0 \cdot 10^{-3}$ and $2.0 \cdot 10^{-3}$ for $A_0$ and $A_1$, respectively, the value of $y$, $u(y)$ read by $A_0$ at time 2.000 becomes 50.000000. At time 2.001, $A_0$ uses the value produced at time 2.000, thus *discrete transition* from $p_{00}$ to $p_{01}$ is enabled. In `foo`, however, the transition $(p_{00}, p_{01})$ must not occur, as the value of $y$ is always less than 50.000000, where $x \in (2.000, \infty)$.

| t | $y$ | $y_{simulation}$ |
|---|---|---|
| ... | ... | ... |
| 1.998 | 49.999980 | 49.999980 |
| 1.999 | 49.999995 | 49.999980 |
| 2.000 | 50.000000 | 50.000000 |
| 2.001 | 49.999995 | 50.000000 |
| ... | ... | ... |

**Table 1.** The Effect of Synchronization Errors of SCHA `foo`

We denote the bound of that discrepancy at position $p$ by $\gamma_{p_j,y}$ and will compute $\gamma_{p_j,y}$ statically using Equation (1). Note that we allow only *exclusive-write/multiple-read* shared variables.

Given an SCHA $C = ((A_0, SV_0), \ldots, (A_n, SV_n))$ and a shared variable $y \in SV_j|_{in} \cap SV_k|_{out}$, the bound of discrepancy due to synchronization error denoted by $\gamma_{p_j,y}$ at position $p_j$ is computed as follows:

$$\gamma_{p_j,y} = \begin{cases} |f(t,y) \cdot h(p_k)|_{max} & \text{if } h(p_k) \geq h(p_j), \\ |f(t,y) \cdot h(p_j)|_{max} & \text{if } h(p_k) < h(p_j), \\ \text{where } f(t,y) \text{ is the derivative of } y \text{ at time } t, p_k \in P_{A_k}, \text{ and } p_j \in P_{A_j}. \end{cases} \tag{1}$$

To formalize the effect of that asynchronous integration effect in code execution, we propose the formalism called System of Instrumented Communicating Hybrid Automata (SICHA). First, we define ICHA as a correspondent of CHA followed by definition of SICHA and then, we show that the alternating run of the SICHA is always included in that of the SCHA. Thus, the SICHA provides sound execution results with regard to the alternating run of the original model.

In the rest of the paper, given an interval $b$, we use $l(b)$ and $r(b)$ to denote the lower and the upper bounds of $b$, respectively. An interval $b$ can be open on either or both sides; so, $b$ can be $(l(b), r(b))$, $(l(b), r(b)]$, $[l(b), r(b))$, or $[l(b), r(b)]$. For arithmetic operations with $l(b)$ and $r(b)$, we assume that $\infty \pm x = \infty$ for any real $x$.

**Definition 15.** *(ICHA). Given a CHA $A$, an Instrumented Communicating Hybrid Automaton of $A$, called ICHA, is defined as a tuple $B = (A, N, h, \beta, \gamma)$, where*

- $N$: $P_A \rightarrow PROG$ *assigns to $p \in P_A$ a numerical method program with a stepsize $h(p)$,*
- $h$: $P_A \rightarrow \mathbb{R}^+$ *assigns to $p \in P_A$ a stepsize $h(p)$,*
- $\beta$: $P_A \times VC \rightarrow \mathbb{R}$ *assigns to $x \in VC$ at each $p \in P_A$, a maximum accumulated numerical error to calculate $x$ denoted by $\beta_{p,x}$,*
- $\gamma$: $P_A \times SV|_{in} \rightarrow \mathbb{R}$ *assigns to $x \in SV|_{in}$ at each $p \in P_A$, a maximum difference of the value of $x$ from time $t$ to time $t + h(p)$ denoted by $\gamma_{p,x}$,*
- *for each $p \in P_A$ and the invariant interval $I_{A,x}(p)$, $l(I_{B,x}(p)) = l(I_{A,x}(p)) + \alpha_{p,x}$, $r(I_{B,x}(p)) = r(I_{A,x}(p)) - \alpha_{p,x}$, for all $x \in VC_A$, and*
- *for each $e \in E_A$ and the guard interval $G_{A,x}(e)$, $l(G_{B,x}(e)) = l(G_{A,x}(e)) + \alpha_{p,x}$, $r(G_{B,x}(e)) = r(G_{A,x}(e)) - \alpha_{p,x}$, for all $x \in VC_A$,*

*where*

$$\alpha_{p,x} = \begin{cases} \beta_{p,x} & \text{if variable } x \notin SV|_{in} \text{ at position } p \\ \beta_{p,x} + \gamma_{p,x} & \text{otherwise.} \end{cases}$$ □

We will use $\alpha$ or $\alpha_x$ instead of $\alpha_{p,x}$ when clear. The definition of *state* and *discrete transition step* for ICHA are the same as the definitions in Section 2.

**Definition 16.** *(Continuous transition step of an ICHA). Given an ICHA $B$, a pair of states $(s_i, s_j)$ is called a unit continuous transition step if the following conditions are satisfied:*

- $s_j|_t = s_i|_t + h(s_i|_p)$,
- $s_i|_p = s_j|_p$,
- *for all* $i, j$, $s_i|_u, s_j|_u \in I_A(s_i|_p)$, *and*
- *for all* $x \in VC - SV_i|_{in}$, $s_j|_u(x)$ *is computed with* $N(s_j|_p), h(s_j|_p)$, *and* $s_i|_u(x)$. □

The value of input variable is defined later in Definition 19 and Definition 20.

**Definition 17.** *(System of Instrumented Communicating Hybrid Automata). Let* $C = (A_0, A_1, \ldots, A_n, SV)$ *be an SCHA. A System of ICHA (SICHA)* $D$ *of an SCHA* $C$ *is defined by a tuple* $(B_0, B_1, \ldots, B_n, SV)$ *where* $B_i$ *is an ICHA of* $A_i$. □

**Definition 18.** *(State of an SICHA). Given an SICHA* $D = (B_0, B_1, \ldots, B_n, SV)$ *of SCHA* $C$, *a state* $s$ *at time* $t$ *is defined as* $((p^{B_0}, u^{B_0}, t^{B_0}), (p^{B_1}, u^{B_1}, t^{B_1}), \ldots, (p^{B_n}, u^{B_n}, t^{B_n}))$, *letting* $s|_t$, $s|_{B_i}$, $s|_{B_i,p}$, $s|_{B_i,u}$, *and* $s|_{B_i,t}$ *be* $t$, $(p^{B_i}, u^{B_i}, t^{B_i})$, $p^{B_i}$, $u^{B_i}$, *and* $t^{B_i}$, *respectively, where*

- *for every* $i$, $s|_{B_i}$ *is a state of ICHA* $B_i$,
- *for every* $i$, $s|_{B_i,t} \leq s|_t$,
- *for some* $i$, $s|_{B_i,t} = s|_t$,
- *for every* $i$, $s|_t - s|_{B_i,t} < h(s|_{B_i,p})$, *and*
- *if* $x \in SV_{A_i}|_{in} \cap SV_{A_j}|_{out}$ *and* $s|_{B_j,t} < s|_{B_i,t}$ *then* $s|_{B_i,u}(x) = s|_{B_j,u}(x)$.

□

**Definition 19.** *(Discrete transition step of an SICHA). Given an SICHA* $D$, *a pair of states* $(s_i, s_j)$ *is called a discrete transition step at time* $t$, *if the followings are satisfied:*

- $s_i|_{B_k,t} = s_j|_{B_k,t}$ *for every ICHA* $B_k$ *in* $D$,
- *there exists some non-empty set* $K \subseteq \{0, 1, \ldots, n\}$, *satisfying the following:*
  - *if* $k \in K$ *then* $(s_i|_{B_k}, s_j|_{B_k})$ *is a discrete transition step of* $B_k$ *at time* $s_i|_t$, *(that is,* $s_i|_{B_k,t} = s_j|_{B_k,t} = s_i|_t$*), and*

$$s_j|_{B_k,u}(x) = \begin{cases} s_j|_{B_l,u}(x) & \text{if } x \in SV_k|_{in} \cap SV_l|_{out} \\ R((s_i|_{B_k,p}, s_j|_{B_k,p}), x) & \text{otherwise,} \end{cases}$$

  - *if* $k \notin K$ *then* $s_i|_{B_k,t} = s_j|_{B_k,t}$, $s_i|_{B_k,p} = s_j|_{B_k,p}$, *and* $s_i|_{B_k,u}(x) = s_j|_{B_k,u}(x)$ *for all* $x \in VC_{B_k}$. □

**Definition 20.** *(Continuous transition step of an SICHA). Given an SICHA* $D$, *a pair of states* $(s_i, s_j)$ *is called a unit continuous transition step if the following conditions are satisfied:*

- $s_i|_t < s_j|_t$,
- *there exists a unique non-empty set* $K \subseteq \{0, 1, \ldots, n\}$ *such that*
  - *for* $k \in K$, $(s_i|_{B_k}, s_j|_{B_k})$ *is a continuous transition step of* $B_k$ *such that* $s_j|_t = s_i|_{B_k,t} + h(s_i|_{B_k,p})$, *and* $s_j|_{B_k,u}(x) = s_i|_{B_l,u}(x)$, *if* $x \in SV_k|_{in} \cap SV_l|_{out}$, *and*

- *for* $k \notin K$, $s_i|_{B_k,t} + h(s_i|_{B_k,p}) > s_j|_t$ *and* $s_i|_{B_k,t} = s_j|_{B_k,t}$, $s_i|_{B_k,p} = s_j|_{B_k,p}$, *and* $s_i|_{B_k,u}(x) = s_j|_{B_k,u}(x)$ *for all* $x \in VC_{B_k}$. □

**Definition 21.** *(Run of an SICHA). A run of an SICHA* $D = (B_0, B_1, \ldots, B_n, SV)$ *is a sequence of states* $\langle s_0, s_1, \ldots, \rangle$ *such that*

- $s_0 = ((p_0^{B_0}, INIT_{B_0}), (p_0^{B_1}, INIT_{B_1}), \ldots, (p_0^{B_n}, INIT_{B_n}), 0)$,
- *if* $s_i|_t = s_{i+1}|_t$, $(s_i, s_{i+1})$ *is a discrete transition step at time* $s_i|_t$, *and*
- *if* $s_i|_t < s_{i+1}|_t$, $(s_i, s_{i+1})$ *is a unit continuous transition step.*

*The* alternating run *of an* SICHA *is defined similarly to that of an* SCHA. *A run of an* SICHA D *is called an* alternating run *of* D, *if the* discrete transition step *and the* continuous transition step *occurs in* D *alternately.* □

Let $A$ be an *HA* and $B$ be its instrumented correspondent. In this paper, we assume that $A$ is $h_B$-insensitive, where $h_B = h_B(p)_{max}, p \in P_B$. Then, given an *HA* $A$ its *IHA* $B$ always produces safe alternating runs. The claim is stated formally in Theorem 1 and its proof is given in [4]. Also, the proofs of Lemma 2 and Lemma 3 are presented in [5].

**Theorem 1.** *Given an HA* $A$, *let* $B = (A, N, h, \beta, \gamma)$ *be an IHA such that* $A$ *is* $h_B$*-insensitive. Then, for every alternating run* $\langle s_0^B, \ldots, s_i^B, \ldots, \rangle$, *there exists an alternating run* $\langle s_0^A, \ldots, s_i^A, \ldots, \rangle$ *of* $A$ *such that*

- $s_i^A|_p = s_i^B|_p$,
- $s_i^A|_u(x) \in [s_i^B|_u(x) - \beta_x, s_i^B|_u(x) + \beta_x]$ *for all* $x \in VC_A - SV|_{in}$, *and*
- $s_i^A|_t = s_i^B|_t$. □

**Lemma 2.** *Given an SCHA* $C = (A_0, A_1, \ldots, A_n, SV)$, *let* $D = (B_0, B_1, \ldots, B_n, SV)$ *be its SICHA and let* $s^C$ *and* $s^D$ *be states in* $C$ *and* $D$, *respectively, satisfying the following conditions:*

- $s^C|_t = s^D|_t$,
- $s^C|_{A_i,p} = s^D|_{B_i,p}$ *for all* $i, 0 \le i \le n$, *and*
- $s^C|_{A_i,u}(x) \in [s^D|_{B_i,u}(x) - \alpha_x, s^D|_{B_i,u}(x) + \alpha_x]$ *for all* $x \in VC_{A_i}$ *and for all* $i, 0 \le i \le n$.

*Suppose* $A_i$ *is* $h_{B_i}$*-insensitive and, for some state* $s'^D$ *of* $D$ *if* $(s^D, s'^D)$ *is a unit continuous transition step with respect to* $K$, *then there exists a continuous transition step* $(s^C, s'^C)$ *for some state* $s'^C$ *in* $C$ *and it satisfies followings:*

1. $s'^C|_{A_i,p} = s'^D|_{B_i,p}$ *for every* $i$,
2. $s'^C|_{A_i,u}(x) \in \begin{cases} [s'^D|_{B_i,u}(x) - \beta_x, s'^D|_{B_i,u}(x) + \beta_x] & \text{if } i \in K \text{ and } x \notin SV_{A_i}|_{in} \\ [s'^D|_{B_i,u}(x) - \alpha_x, s'^D|_{B_i,u}(x) + \alpha_x] & \text{otherwise,} \end{cases}$
3. $s'^C|_t = s'^D|_t$, *and*
4. $s'^C|_{A_i,u}(x) \in I_{A_i,x}(s'^C|_{A_i,p})$ *for all* $x \in VC_{A_i}$ *and for all* $i$. □

**Lemma 3.** *Given an SCHA* $C = (A_0, A_1, \ldots, A_n, SV)$, *let* $D = (B_0, B_1, \ldots, B_n, SV)$ *be its SICHA and let* $s^C$ *and* $s^D$ *be its states satisfying the conditions for some* $K \subseteq \{0, 1, \ldots, n\}$,

- $s^C|_{A_i,p} = s^D|_{B_i,p}$, *for all* $i$, $0 \leq i \leq n$,
- $s^C|_{A_i,u}(x) \in [s^D|_{B_i,u}(x) - \alpha_x, s^D|_{B_i,u}(x) + \alpha_x]$, *for all* $x \in SV_{A_i}$ *and for all* $i$, $0 \leq i \leq n$,
- $s^C|_t = s^D|_t$, *and*
- $s^D|_{B_j,u}(x) \in G_x(s^D|_{B_j,p})$, *if* $j \in K$.

*Then, for some state* $s'^B$ *of* $B$, *if* $(s^B, s'^B)$ *is a discrete transition step with respect to* $K$ *then there exists a discrete transition step* $(s^C, s'^C)$ *with respect to* $K$ *for some state* $s'^C$ *in* $C$ *satisfying followings:*

1. $s'^C|_{A_i,p} = s'^D|_{B_i,p}$, *for all* $i, 0 \leq i \leq n$,
2. $s'^C|_{A_i,u}(x) = s'^D|_{B_i,u}(x)$, *if* $i \in K$ *and* $x \notin SV_{A_i}|_{in}$,
3. $s'^C|_{A_i,u}(x) \in [s'^D|_{B_i,u}(x) - \alpha_x, s'^D|_{B_i,u}(x) + \alpha_x]$, *if* $i \notin K$ *or* $x \in SV_{A_i}|_{in}$,
4. $s'^C|_t = s'^D|_t$, *and*
5. $s'^C|_{A_i,u}(x) \in I_x(s'^C|_{A_i,p})$, *for all* $x \in SV_{A_i}$ *and for all* $i \in K$. □

**Theorem 2.** *Let* $C$ *and* $D$ *be an SCHA,* $(A_0, \ldots, A_j, \ldots, A_n, SV)$, *and its SICHA,* $(B_0, \ldots, B_j, \ldots, B_n, SV)$, *respectively. Suppose* $A_j$ *is* $h_{B_j}$*-insensitive, then, for every alternating run* $\langle s_0^D, s_1^D \ldots, s_i^D, \ldots, \rangle$ *in* $D$, *there exists an alternating run* $\langle s_0^C, s_1^C \ldots, s_i^C, \ldots, \rangle$ *in* $C$ *such that*

- $s_i^C|_{A_i,p} = s_i^D|_{B_i,p}$,
- $s_i^C|_{A_i,u}(x) \in [s_i^D|_{A_i,u}(x) - \alpha_x, s_i^D|_{B_i,u}(x) + \alpha_x]$, *and*
- $s_i^C|_t = s_i^D|_t$.

*Proof.* Immediately followed by lemma 2 and lemma 3. □

*Example revisited.* Figure 4 depicts the instrumented version of the SCHA `foo` described in Figure 3 to exclude the unexpected execution trace from `foo`. As the stepsize of $A_0$ and $A_1$ are $1.0 \cdot 10^{-3}$ and $2.0 \cdot 10^{-3}$, respectively, $\gamma_{p00,y}$ is $0.002 \cdot 100.0 = 0.2$. If we instrument the guard at position $p_{00}$ of $A_0$ with $\gamma_{p00,y}$, then the unexpected trace disappears.

## 5 Executable Code

We have implemented a code generator that produces `C++` code implementing SICHA. The generated code needs to be associated with a real-time scheduler to satisfy timely computation. Each ICHA $B_i$ of an SICHA can be mapped to a periodic task with the period and the deadline equal to $\min(h_{B_i}(p))$. In addition, a scheduling policy that guarantees the condition of a valid state of an SICHA (Definition 18) should be chosen. That is, for every state $s$ of an SICHA, the condition $s|_t - s|_{B_i,t} < h(s|_{B_i,p})$ should be satisfied for all $B_i$. This implies that the task for $B_i$ should be scheduled earlier than the task for $B_j$ if $s|_{B_i,t} + h(s|_{B_i,p}) < s|_{B_j,t} + h(s|_{B_j,p})$. Note that this requirement is equivalent to the well-known EDF scheduling policy. (The RM scheduling policy can satisfy the requirement only when the periods are harmonic.)

Moreover, it is also required that the task for $B_i$ should be blocked even when the system is idle if the task for $B_k$ such that $s|_{B_k} + h(s|_{B_k,p}) < s|_{B_i} + h(s|_{B_i,p})$ is not ready (i.e., $t < s|_{B_k,t}$, where $t$ is the real time). This wastes the CPU

utilization and may affect schedulability, similarly to the problem of scheduling of tasks with dependency. This problem can be addressed in two ways. First, the execution result of a task can be buffered and emitted later to avoid blocking of such tasks (see [6, 7], for example). Second, the model can be instrumented to tolerate the errors due to scheduling of such tasks, in the same way as we do for models with different rates. Our approach has the advantage of avoiding possible overhead of buffering.

Another important issue is how to resolve non-determinism of discrete transitions. The non-determinism comes from two sources. First, a discrete transition may or may not be taken when the associated guard is enabled. Second, there may be more than one transition whose guard is enabled. Depending on the decision, different behaviors may arise. The set of behaviors are acceptable as long as they do not violate the invariant condition. Note that our framework guarantees that every behavior of SICHA is also found in the hybrid model (within bounded deviation) up to the point before the invariant is violated whatever a decision is made on non-deterministic discrete transitions.

In the case where the code detects violation of the invariant, it means either (1) the hybrid model also has an equivalent run that ends with invariant violation, or (2) the code missed an outgoing transition before the invariant is violated due to discretized guard checking. The former issue is the matter of the validity of the model, rather than code generation. That is, such a case can be prevented by refining the model such that the states outside of the invariant set are unreachable. (See [8–10] for systematic approaches.) On the other hand, the latter case is an artifact of code generation. Instrumentation may additionally cause such an artifact that should not otherwise occur, because it reduces the guard set and the invariant set, and thus gives a better chance of transition misses. Assuming that the hybrid model is valid (i.e., the invariant set is unreachable), transition misses can be prevented if the guard set and the invariant set overlap for a duration of time longer than the step size, and the code employs an *urgent transition* policy (i.e., transition is taken as soon as it is detected enabled). For detailed description, see [11].

## 6 Conclusion

In this paper, we have proposed a code generation framework for hybrid models that focuses on soundness of synthesized code. The idea behind the sound code generation is to refine the hybrid model such that it is robust to erroneous values. In this paper, we have focused on the effect of synchronization errors that occur when components of a hybrid model are synthesized into concurrent programs having different rates. We have proved that every possible behavior of the model instrumented with maximum possible errors is a valid behavior of the original hybrid model. We have also explained the issue of scheduling and non-determinism when the instrumented hybrid automata are finally converted into executable code.

We implemented our idea in the context of the hybrid systems modeling language CHARON [12]. Previous implementation of the code generator [11, 13] has been extended to allow instrumentation of the guard and the invariant. We

have also performed preliminary experiments with Sony's robot dog AIBO to avoid erroneous behavior that is not consistent to the model.

Our work can be extended further in many ways. First, a more systematic way of model instrumentation may be possible if model checking techniques are employed. The predicate abstraction-based model checking tool developed for CHARON [14] can be used for this purpose. We expect that model checking based instrumentation leads to more tightly instrumented code. Second, we can also include controller synthesis techniques [8–10] to our framework to allow automatic refinement of the model such that the generated code is guaranteed to satisfy the invariant condition.

**Acknowledgement** This research was supported in part by NSF CCR-9988409, NSF CCR-0086147, NSF CCR-0209024, ARO DAAD19-01-1-0473, and DARPA ITO MOBIES F33615-00-C-1707.

## References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
2. Esposito, J., Kumar, V., Pappas, G.: Accurate event detection for simulating hybrid systems. In: Proceedings of HSCC. LNCS 2034 (2001) 204–217
3. Park, T., Barton, P.: State event location in differential-algebraic models. ACM Transactions on Modeling and Computer Simulation **6** (1996) 137–165
4. Choi, J.Y., Hur, Y., Lee, I.: IHA: Ensuring sound numerical simulation of hybrid automata. Technical Report MS-CIS-03-06, University of Pennsylvania (2003)
5. Choi, J.Y., Hur, Y., Kim, J., Lee, I.: Sound synchronization of communicating hybrid automata. Technical Report MS-CIS-03-30, University of Pennsylvania. (2003)
6. Henzinger, T., Horowitz, B., Kirsch, C.: Giotto: A time-triggered language for embedded programming. In: Proceedings of EMSOFT. (2001) 166–184
7. Kodase, S., Wang, S., Gu, Z., Shin, K.G.: Improving scalability of task allocation and scheduling in large distributed real-time systems using shared buffers. In: Proceedings of RTAS. (2003) 181–188
8. Wong-Toi, H.: The synthesis of discrete controllers for linear hybrid automata. In: Proceedings of CDC. (1997) 4607–4612
9. Altisen, K., Gößler, G., Pnueli, A., , Sifakis, J., Yovine, Y.: A framework for scheduler synthesis. In: Proceedings of RTSS. (1999) 154–163
10. Altisen, K., Gößler, G., Sifakis, J.: A methodology for the construction of scheduled systems. In: Proceedings of FTRTFT. (2000) 106–120
11. Alur, R., Ivančić, F., Kim, J., Lee, I., Sokolsky, O.: Generating embedded software from hierarchial hybrid models. In: Proceedings of LCTES. (2003) 171–182
12. Alur, R., Dang, T., Esposito, J., Hur, Y., Ivančić, F., Kumar, V., Lee, I., Mishra, P., Pappas, G., Sokolsky, O.: Hierarchical modeling and analysis of embedded systems. Proceedings of the IEEE **91** (2003) 11–28
13. Kim, J., Lee, I.: Modular code generation from hybrid automata based on data dependency. In: Proceedings of RTAS. (2003) 160–168
14. Alur, R., Dang, T., Ivančić, F.: Reachability analysis of hybrid systems via predicate abstraction. In: Proceedings of HSCC. LNCS 2289 (2002) 35–48