Automatically Generating SystemC Code from HCSP Formal Models

GAOGAO YAN, LI JIAO, SHULING WANG, LINGTAI WANG, and NAIJUN ZHAN,
State Key Lab. of Comput. Sci., Institute of Software, Chinese Academy of Sciences, China and Gaogao Yan, Lingtai Wang and Naijun Zhan are also affiliated with University of Chinese Academy of Sciences, China

In model-driven design of embedded systems, how to generate code from high-level control models seamlessly and correctly is challenging. This is because hybrid systems, a normal control model, are involved with continuous evolution, discrete jumps, and the complicated entanglement between them, while code only contains discrete actions. In this paper, we investigate the code generation from Hybrid Communicating Sequential Processes (HCSP), a formal control model, to SystemC. We first introduce the notion of approximate bisimulation as a criterion to check the consistency between two different systems, especially between the original control model and the final generated code. We prove that it is decidable whether two HCSPs are approximately bisimilar in bounded time and unbounded time with some constraints, respectively. For both the cases, we present two sets of rules correspondingly for discretizing HCSPs, and prove that the original HCSP model and the corresponding discretization are approximately bisimilar. Furthermore, based on the discretization, we give a transfer function to map a discretized HCSP model to SystemC code such that they are also approximate bisimilar. We finally implement a tool to automatically realize the translation from HCSPs to SystemC code, and illustrate our approach through some case studies.

CCS Concepts: • **Software and its engineering** → **Model-driven software engineering**; **System modeling languages**; • **Computer systems organization** → **Embedded systems**; • **Theory of computation** → **Timed and hybrid models**;

Additional Key Words and Phrases: Hybrid CSP, approximate bisimulation, code generation

ACM Reference Format:
Gaogao Yan, Li Jiao, Shuling Wang, Lingtai Wang, and Naijun Zhan. 20XX. Automatically Generating SystemC Code from HCSP Formal Models . *ACM Trans. Softw. Eng. Methodol.* 0, 0, Article 00 (20XX), 40 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Embedded Systems (ESs) make use of computer units to control physical processes so that their behaviors could meet expected requirements. They have become ubiquitous in our daily life such as automotive, aerospace, consumer electronics, communication, medical equipment and manufacturing. Many ESs are safety-critical — a tiny fault may lead to a catastrophic result. It is a grand

This work is supported partly by NSFC under grant No. 61625206 and 61732001.
The corresponding author:Naijun Zhan, 4 South Fourth Street, Zhong Guan Cun, Beijing, 100190, China.
Authors' address: Gaogao Yan; Li Jiao; Shuling Wang; Lingtai Wang; Naijun Zhan,
State Key Lab. of Comput. Sci., Institute of Software, Chinese Academy of Sciences, 4 South Fourth Street, Zhong Guan Cun, Beijing, 100190, China, yangg@ios.ac.cn, ljiao@ios.ac.cn, wangsl@ios.ac.cn, wanglt@ios.ac.cn, znj@ios.ac.cn, Gaogao Yan, Lingtai Wang and Naijun Zhan are also affiliated with University of Chinese Academy of Sciences, Beijing, 100190, China.

1049-331X/20XX/0-ART00 $15.00
https://doi.org/0000001.0000001

challenge to design ESs correctly, especially those complex and safety-critical ones. Among a variety of developing methods, Model-Driven Design (MDD) is considered to be effective and has been successfully applied in industry [42, 49]. In the framework of MDD, a model of the system is first constructed and then updated through extensive analysis and verification, so that errors can be detected and corrected at early stages of the design. Afterwards, by applying model transformation techniques, the abstract (formal) model is transformed into a more concrete model and finally into executable code. Particularly, the transformation process should be automatic, since a manual fashion is error prone and costly [16]. In a word, MDD framework is an integration of modeling, analysis, verification and code generation, which is reliable, efficient and has low cost.

Hybrid systems seamlessly integrate traditional discrete models with dynamical models such as differential and algebraic equations, able to represent the feedback dynamics of embedded computers in physical, chemical, or biological environments. Their precise mathematical semantics are appropriate for them to model complex embedded systems, based on which analysis and verification are therefore possible to be performed. Subsequently, a central problem in MDD is then how to transform an abstract hybrid control model to an algorithmic model at code level rigorously and automatically. This is quite challenging, as the controller code determines how to sample data from the continuous plant and the entanglement between sampling data and computing control commands is intricate. An efficient approach is to discretize the continuous plant, and then, the discretized continuous plant together with the controller code constitute a complete implementation for the closed-loop hybrid system. Therefore, how and according to which criterion to sample (discretize) the continuous behaviour is essential to guarantee the correctness and reliability of the implementation. The Simulink Coder [4] of MathWorks, formerly known as Real-Time Workshop, generates code from discrete-continuous Simulink models, and the generated code is used widely for real-time applications including simulation acceleration, rapid prototyping, and hardware-in-the-loop testing. As a Third-Party of Simulink, the dSPACE rapid control prototyping system [1] is integrated seamlessly with Simulink and uses the code generated by Simulink Coder for developing real-time applications. But, the above problem is not well solved and remains challenging.

Except for the above mentioned Simulink, many MDD approaches targeting ESs, support code generation from control models together with ODE models, both in industry and academia, e.g., SCADE [30], Modelica [68], SysML [5], MARTE [64], Ptolemy [31]. However, their correctness from control models to source code is not completely guaranteed. To address this issue, a criterion is absolutely demanded to judge in what sense a transformation from a control model to code is correct, in other words, the criterion should preserve semantics. Clearly, a criterion with exact semantics preservation like that in classical programing theories [63] is impossible.

Fortunately, approximate bisimulation [36] provides an appropriate criterion. Approximate bisimulation allows error tolerance, i.e., the “distance” between the behaviors of two systems is allowed to be within a given threshold rather than exactly identical. The notion of approximate bisimulation has been extended to analysis and verification of different kinds of hybrid systems [41, 46, 48, 54, 60–62, 69]. But all the existing results can only be applied to restricted hybrid systems, as either discrete dynamics is not considered, or atomic actions are required to be independent of continuous variables.

In this paper, we first extend approximate bisimulation to the hybrid modeling language called Hybrid Communicating Sequential Processes (HCSP) [40, 80] which is an extension of Communicating Sequential Processes (CSP) to include differential equations and interrupts. Those interactions model interactions between continuous and discrete dynamics and are very flexible, e.g., discrete actions can be triggered by changes of continuous variables. Compared with other formalisms such as hybrid automata [43] and hybrid programs [59], HCSP provides a compositional way to model complicated ESs due to its flexible constructors. Secondly, we propose a set of rules for discretizing HCSP models, and prove that the discretized HCSP model is approximately bisimilar to the original HCSP model. Besides, we give a method for generating SystemC [45] code from the discretized HCSP model and prove that they are approximately bisimilar with zero tolerance. The reason why we choose SystemC as the target programming language is that SystemC is widely used in industry (it is implemented as a C++ class library) and that its syntax and semantics are very similar to discretized HCSP models (so that the translation from discrete HCSP to SystemC is straightforward). In addition, we define a formal operational semantics for a subset of SystemC by a slight extension of the definition in [74, 75] in order to prove the correctness of the transformation from the discretized HCSP model to SystemC code.

This work is an extension of our previous work [73], where HCSP is used as the modeling language for hybrid systems. In [73], we first extended the notion of approximate bisimulation to HCSP models, then proved that it is decidable whether two HCSP processes are approximately bisimilar if all ordinary differential equations (ODEs) occurring in them satisfy the Globally Asymptotical Stability (GAS) condition [12]. Intuitively speaking, GAS requires an ODE with any initial state can always be arbitrarily close to its equilibrium point as time proceeds. We also presented an algorithm to discretize an HCSP (the control model) into a discrete HCSP (an algorithmic model), and proved that they are approximately bisimilar if GAS is satisfied in the original HCSP.

However, the shortcomings of [12] at least include that GAS is very restrictive and that discrete HCSP is not executable. Therefore, in this paper we consider the bounded-time execution instead of requiring the system satisfying the GAS condition, and furthermore transform the discretized HCSP to executable SystemC code. In detail, we extend and improve [73] in the following aspects:

- We consider hybrid systems executed within a bounded time instead of requiring GAS, and show that whether two such HCSP processes are approximately bisimilar is decidable.
- We develop a set of discretizing rules to transform an HCSP to a discrete one which are proved to be approximately bisimilar.
- We present two algorithms to compute time and value precisions respectively for HCSP processes which are robustly safe. We then consider how to discretize those HCSP processes.
- We present a transformation function mapping a discretized HCSP into SystemC code.
- We introduce a formal operational semantics for a subset of SystemC, based on which the approximate bisimularity between the discretized HCSP and the generated SystemC code is proved.
- We implement a tool to automatically transform an HCSP without (with) GAS condition to SystemC code such that they are approximately bisimilar on bounded (unbounded) time with respect to the given precision.
- We also provide several case studies to illustrate the efficiency of our approach.

Together with the work reported in [77, 85, 86], this paper forms a framework for formal design of safety-critical systems from graphical modeling, simulation and formal verification to executable code with tool supports [27, 71, 84]. We believe it can improve the reliability of such systems.

The rest of this paper is organized as follows. Some preliminary notions on dynamical systems, transition systems, HCSP and SystemC are introduced in Sec. 2. Sec. 3 defines approximate bisimulation between HCSP processes, and proves that it is decidable whether two HCSP processes are approximately bisimilar in time-bounded and time-unbounded cases. In Sec. 4, discretization of HCSP in both cases is presented under certain conditions. Then, the conditions under which the original HCSP and its discretization are approximately bisimilar are discussed in Sec. 5. After that, the translation from discrete HCSP to SystemC code and their bisimulation are presented in Sec. 6.

Symbols	Meaning
$\bar{\mathrm{x}}$	The equilibrium point of a given ODE
TS, TS_i	Transition systems
P, Q, P_i	Sequential HCSP processes
S, S_i	Parallel HCSP processes
B	Boolean expression
$l, l_i \in L$	Actions in the label set of transition system
$X(t, \mathrm{x})$	The solution to a given ODE at time t with initial value x
ε, ξ	The precision thresholds for the discretization of an HCSP process and of an ODE resp.
h	The time precision for the discretization
$D_{h,\varepsilon}(P)$	The discretization function with precisions h and ε
$N(\phi, \varepsilon)$, $N(\phi, -\varepsilon)$	The ε and $-\varepsilon$ neighborhoods of formula ϕ resp.
ϵ, δ	The value and time parameters for robustly safe processes

Table 1. Summary of some frequently used symbols and functions

In Sec. 7, three case studies are provided to illustrate our approach. Sec. 8 discusses related works, and Sec. 9 concludes the paper and discusses the future work.

2 PRELIMINARIES

In this section, we will introduce some preliminary knowledge for understanding the work. In Sec. 2.1, some basic notions about dynamical systems are presented. Labeled transition systems, a semantic model of HCSP, on which the notion of the approximate bisimulation relation between two systems is discussed, is given in Sec. 2.2. Afterwards, the source and target languages, i.e. HCSP and SystemC, are presented in Sec. 2.3 and Sec. 2.4, respectively.

In order to ease the reader to understand the meanings of some frequently used symbols and functions throughout this paper, we give a summary of symbols in Table 1.

2.1 Dynamical Systems

In what follows, $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}^+$, $\mathbb{R}^+_0$ denote the natural, real, positive and nonnegative real numbers, respectively. Vectors are denoted by boldface letters, for instance, $\mathbf{x}$, $\mathbf{y}$. Given a vector $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|$ denotes the infinity norm of $\mathbf{x}$, i.e., $\|\mathbf{x}\| = \max\{|x_1|, |x_2|, ..., |x_n|\}$. A continuous function $\gamma : \mathbb{R}^+_0 \longrightarrow \mathbb{R}^+_0$ is said to be in class $\mathcal{K}$ if it is strictly increasing and $\gamma(0) = 0$, and to be in class $\mathcal{K}_\infty$ if $\gamma \in \mathcal{K}$ and $\gamma(r) \longrightarrow \infty$ as $r \longrightarrow \infty$. A continuous function $\beta : \mathbb{R}^+_0 \times \mathbb{R}^+_0 \longrightarrow \mathbb{R}^+_0$ is said to be in class $\mathcal{KL}$ if for each fixed t, $\beta_t(r) = \beta(r, t) \in \mathcal{K}_\infty$ with respect to r, and for each fixed r, $\beta_r(t) = \beta(r, t)$ is decreasing with respect to t and tends to 0 as $t \longrightarrow \infty$.

A dynamical system is of the following form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}), \quad \mathbf{x}(t_0) = \mathbf{x}_0. \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state and $\mathbf{x}(t_0) = \mathbf{x}_0$ is the *initial condition.*

Suppose $t_0 < b$. A function $X(\cdot) : [t_0, b) \longrightarrow \mathbb{R}^n$ is said to be a *trajectory* (or solution) of (1) on $[t_0, b)$, if $X(t_0) = \mathbf{x}_0$ and $\dot{X}(t) = \mathbf{f}(X(t))$ for all $t_0 \leq t < b$. In order to ensure the existence and uniqueness of trajectories, we assume $\mathbf{f}$ satisfies the local Lipschitz condition, i.e., for every compact set $S \subset \mathbb{R}^n$, there exists a constant $L > 0$ s.t. $\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$ for all $\mathbf{x}, \mathbf{y} \in S$.

Under this condition, $X(t, \mathrm{x}_0)$, which is uniquely determined, denotes the point reached at time $t \in [t_0, b)$ from the initial condition x_0. In addition, we assume (1) is *forward complete* [13], i.e., it is solvable on an open interval $(t_0, +\infty)$. $\bar{\mathrm{x}} \in \mathbb{R}^n$ is an equilibrium point of (1) if $\mathrm{f}(\bar{\mathrm{x}}) = 0$.

Definition 2.1. A dynamical system of form (1) is said to be *globally asymptotically stable* (GAS) if there exists a equilibrium point $\bar{\mathrm{x}}$ and a function β of class $\mathcal{KL}$ s.t.

$$\forall \mathrm{x} \in \mathbb{R}^n\ \forall t \geq 0. \|X(t, \mathrm{x}) - \bar{\mathrm{x}}\| \leq \beta(\|\mathrm{x} - \bar{\mathrm{x}}\|, t), \tag{2}$$

The point $\bar{\mathrm{x}}$ is actually the unique equilibrium point of the system. According to the definition, $X(t, \mathrm{x})$ tends to $\bar{\mathrm{x}}$ as t approaches $+\infty$ no matter what the initial state x is, since the upper bound $\beta(\|\mathrm{x} - \bar{\mathrm{x}}\|, t)$ tends to 0. Methods on judging whether a dynamical system is GAS are proposed in [12, 70].

2.2 Transition Systems

Let the action set $Act = \mathcal{D} \cup \mathbb{R}_0^+ \cup \{\tau\}$, where $\mathcal{D}$ is the set of instantaneous discrete actions, $\mathbb{R}_0^+$ is the set of delay actions, and τ is a special instantaneous internal action. Given $l_1, l_2 \in Act$, the distance $dis(l_1, l_2)$ is defined as follows:

$$dis(l_1, l_2) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } l_1 = l_2 \\ l_1 & \text{if } l_1 \in \mathbb{R}_0^+ \text{ and } l_2 = \tau, \\ l_2 & \text{if } l_2 \in \mathbb{R}_0^+ \text{ and } l_1 = \tau, \\ |l_1 - l_2| & \text{if } l_1, l_2 \in \mathbb{R}_0^+, \\ +\infty & \text{otherwise.} \end{cases} \tag{3}$$

Intuitively, the distance between two actions l_1 and l_2 stands for the remaining execution time of l_2 when l_1 finished earlier, and vice versa. Two distinguished discrete actions cannot match each other, so we define their distance to be infinity.

Definition 2.2 (Labeled transition system). A labeled transition system with observations is a tuple $TS = \langle Q, L, \rightarrow, Q^0, Y, H\rangle$, where Q is a set of states, $L \subseteq Act$ is a set of labels, $\rightarrow \subseteq Q \times L \times Q$ is a transition relation, $Q^0 \subseteq Q$ is a set of initial states, Y is a set of observations and $H : Q \rightarrow Y$ is an observation function. $(\mathrm{q}, l, \mathrm{q}') \in \rightarrow$ is also written as $\mathrm{q} \xrightarrow{l} \mathrm{q}'$, and $\rightarrow$ satisfies

1. **identity:** $\mathrm{q} \xrightarrow{0} \mathrm{q}$ always holds;
2. **delay determinacy:** if $\mathrm{q} \xrightarrow{d} \mathrm{q}'$ and $\mathrm{q} \xrightarrow{d} \mathrm{q}''$, then $\mathrm{q}' = \mathrm{q}''$, where $d \in \mathbb{R}_0^+$; and
3. **delay additivity:** if $\mathrm{q} \xrightarrow{d_1} \mathrm{q}'$ and $\mathrm{q}' \xrightarrow{d_2} \mathrm{q}''$ then $\mathrm{q} \xrightarrow{d_1+d_2} \mathrm{q}''$, where $d_1, d_2 \in \mathbb{R}_0^+$.

A transition system TS is said to be *symbolic* if Q and $L \cap \mathcal{D}$ are finite, and $L \cap \mathbb{R}_0^+$ is bounded, and *metric* if the observation set Y is equipped with a metric $\mathbf{d} : Y \times Y \rightarrow \mathbb{R}_0^+$. Throughout this paper, Y is $\mathbb{R}^n$ with the Euclidean distance $\mathbf{d}(\mathrm{y}_1, \mathrm{y}_2) = \|\mathrm{y}_1 - \mathrm{y}_2\|$.

A *state trajectory* of a transition system TS is a (possibly infinite) sequence of transitions $\mathrm{q}^0 \xrightarrow{l_0} \mathrm{q}^1 \xrightarrow{l_1} \cdots \xrightarrow{l_{i-1}} \mathrm{q}^i \xrightarrow{l_i} \cdots$, denoted by $\{\mathrm{q}^i \xrightarrow{l_i} \mathrm{q}^{i+1}\}_{i\in\mathbb{N}}$, s.t. $\mathrm{q}^0 \in Q^0$ and $\mathrm{q}^i \xrightarrow{l_i} \mathrm{q}^{i+1}$ for any i. An *observation trajectory* is a (possibly infinite) sequence $\mathrm{y}^0 \xrightarrow{l_0} \mathrm{y}^1 \xrightarrow{l_1} \cdots \xrightarrow{l_{i-1}} \mathrm{y}^i \xrightarrow{l_i} \cdots$, denoted by $\{\mathrm{y}^i \xrightarrow{l_i} \mathrm{y}^{i+1}\}_{i\in\mathbb{N}}$, and it is accepted by TS if there exists a state trajectory $\{\mathrm{q}^i \xrightarrow{l_i} \mathrm{q}^{i+1}\}_{i\in\mathbb{N}}$ s.t. $\mathrm{y}^i = H(\mathrm{q}^i)$ for all $i \in \mathbb{N}$. The set of observation trajectories accepted by TS is called the *language* of TS, and is denoted by $L(TS)$. The reachable observation set of TS is a subset of Y defined by

$$Reach(TS) = \{\mathrm{y} \in Y \mid \text{there exists } \mathrm{y}^0 \xrightarrow{l_0} \mathrm{y}^1 \xrightarrow{l_1} \cdots \xrightarrow{l_{i-1}} \mathrm{y}^i \in L(TS) \text{ with } \mathrm{y} = \mathrm{y}^i\}. \tag{4}$$

Let $Y_U \subseteq Y$ be the set of unsafe observations. The safety of TS can be verified by computing $Reach(TS) \cap Y_U$: TS is *safe* if the intersection is empty, and *unsafe* otherwise.

A *maximal sequence of* τ of a state trajectory $\{\mathrm{q}^j \xrightarrow{l_j} \mathrm{q}^{j+1}\}_{j\in\mathbb{N}}$ is a subsequence of the form $\{\mathrm{q}^j \xrightarrow{l_j} \mathrm{q}^{j+1}\}_{i\leq j\leq i+k}$ with $l_i = \ldots = l_{i+k-1} = \tau$, $l_{i-1} \neq \tau$ if $i \geq 1$ and $l_{i+k} \neq \tau$. For a maximal sequence of τ denoted as $\mathrm{q}^i \xrightarrow{\tau} \cdots \xrightarrow{\tau} \mathrm{q}^{i+k}$, we remove the intermediate states and define the *τ-compressed* transition $\mathrm{q}^i \overset{\tau}{\twoheadrightarrow} \mathrm{q}^{i+k}$ instead. We also expand the *τ-compressed* transition to non-τ transitions such that $\mathrm{q}^i \overset{l_i}{\twoheadrightarrow} \mathrm{q}^{i+1}$ if $\mathrm{q}^i \xrightarrow{l_i} \mathrm{q}^{i+1}$ where $l_i \neq \tau$. In what follows, $\langle Q, L, \twoheadrightarrow, Q^0, Y, H\rangle$ denotes the labeled transition system resulted from $\langle Q, L, \longrightarrow, Q^0, Y, H\rangle$ by replacing each transition with its τ-compressed version. Furthermore, following conventions in process algebra, we use $\mathrm{p} \overset{l}{\Longrightarrow} \mathrm{p}'$ to denote the closure of τ transitions, i.e., $\mathrm{p}(\overset{\tau}{\twoheadrightarrow})^{\{0,1\}} \overset{l}{\twoheadrightarrow} (\overset{\tau}{\twoheadrightarrow})^{\{0,1\}}\mathrm{p}'$, for any $l \neq \tau$. Here $(\overset{\tau}{\twoheadrightarrow})^{\{0,1\}}$ means zero or one τ transition.

2.3 Hybrid Communicating Sequential Processes

Hybrid Communicating Sequential Processes (HCSP) is a formal language for describing hybrid systems, which extends CSP by introducing differential equations for modeling continuous evolution and interrupts for modeling the interaction between continuous evolution and discrete jumps. The syntax of HCSP is given below:

$$\begin{array}{lll} P & ::= & \textbf{skip} \mid \textbf{stop} \mid x := e \mid \text{wait } d \mid ch?x \mid ch!e \mid P;Q \mid B \rightarrow P \mid P \sqcap Q \mid P^* \mid \\ & & [\!]_{i\in I}(io_i \rightarrow P_i) \mid \langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0 \& B\rangle \mid \langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(io_i \rightarrow Q_i) \\ S & ::= & P_1 \| P_2 \| \ldots \| P_n \text{ for some } n \geq 1 \end{array}$$

where x and s stand for variables and vectors of variables, respectively, B and e are Boolean and arithmetic expressions, d is a non-negative real constant, ch is the channel name, io_i stands for an input or output communication event (i.e., either $ch_i?x$ or $ch_i!e$), P, Q, Q_i, P_i are sequential process terms, and S stands for an HCSP which is either a sequential process or the composition of multiple parallel processes. Parallel processes are not allowed to share variables. They can only exchange messages via synchronized communications along their common channels. Furthermore, we assume that each input or output channel can be possessed by exactly one sequential process in order to simplify the semantics of HCSP.

The informal meanings of the individual constructs are as follows:

- **skip** — Terminate immediately with variables unchanged.
- **stop** — Do nothing with time passing.
- $x := e$ — Assign the value of expression e to x and then terminate.
- wait d — Keep idle for d time units with all variables unchanged.
- $ch?x$ — Assign to x the value received from channel ch.
- $ch!e$ — Send the value of e along channel ch.
  - Note that a communication takes place only when both the sending and receiving parties are ready. So one side may have to wait until the other becomes ready.
- $P;Q$ — Execute P first, and Q afterwards if P terminates.
- $B \rightarrow P$ — Execute P if B is true. Otherwise terminate immediately.
- $P \sqcap Q$ — (Internal choice.) Execute either P or Q. The choice is made by the process.
- P^* — (Repetition.) Execute P for an arbitrary finite number of times. Assume there exists an oracle *num* s.t. $num(P^*)$ returns the upper bound of P's repeating times for a given P^*. In particular, P^k means P repeating k times. Note that $P^0 = \textbf{skip}$.

- $[]_{i\in I}(io_i \longrightarrow P_i)$ — (External choice.) Execute the P_j if io_j is the first event to occur. Make a non-deterministic choice if multiple events emerge simultaneously.
- $\langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0\&B\rangle$ — (Continuous evolution.) The vector s evolves following the dynamical system $F(\dot{\mathrm{s}}, \mathrm{s}) = 0$ as long as B holds, and terminates when B turns false. B acts as the domain of s. In particular, "wait d" can be derived as $t := 0; \langle \dot{t} = 1\&t < d\rangle$.
  Here we assume that the set of B is open so that the escape point is located at its boundary. We also restrict F to be ordinary differential equations (ODEs) by assuming the Jacobian matrix of F to be non-singular (i.e., invertible), then $F(\dot{\mathrm{s}}, \mathrm{s}) = 0$ becomes an implicit ODE system rather than Differential-Algebraic Equations whose Jacobian matrix is singular [14]. As shown in Sec. 4, we only consider F of the special form $\dot{\mathrm{s}} = \mathrm{f}(\mathrm{s})$. We will consider the discretization of such ODEs with respect to a fixed initial state.
- $\langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0\&B\rangle \trianglerighteq []_{i\in I}(io_i \longrightarrow Q_i)$ — (Communication interrupt.) Behave as $\langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0\&B\rangle$ does, unless one or more events in $\{io_i\}_{i\in I}$ occur before the continuous evolution terminates. If a communication jo_j for some j occurs earlier than others, then the corresponding branch is chosen to execute. If multiple communications in $\{io_i\}_{i\in I}$ get ready simultaneously earlier than others, then a non-deterministic choice will be made among them to decide which one to execute.
  - Note that the above two constructs are the main extension of HCSP for describing continuous behaviors.
- $P_1\|P_2\|\ldots\|P_n$ $(n \geq 2)$— Behave as if processes $P_1, P_2, \ldots, P_n$ run independently except that all communications along their common channels must be synchronized.

Remark. The HCSP syntax we present above is only a subset of the HCSP in [40, 80]. To ease the discretization process of HCSP, we put some restrictions on the syntax. For example, we adopt Kleene star rather than general recursion for characterizing loops, and only allow parallel compositions to occur at the outermost level. There is no essential difficulty in expanding this work to include the full set of HCSP , e.g., the general recursion handled in [85].

Example 2.3. For better understanding of HCSP, we model the water tank system [8], aiming at maintaining its water level within desired boundaries, as an HCSP *WTS* composed of two parallel components *Watertank* and *Controller*:

$$
\begin{aligned}
WTS &\overset{\text{def}}{=} Watertank\|Controller\\
Watertank &\overset{\text{def}}{=} v := v_0; d := d_0; (v = 1 \longrightarrow \langle \dot{d} = Q_{max} - \pi r^2\sqrt{2gd}\rangle \trianglerighteq (wl!d \longrightarrow cv?v);\\
&\qquad v = 0 \longrightarrow \langle \dot{d} = -\pi r^2\sqrt{2gd}\rangle \trianglerighteq (wl!d \longrightarrow cv?v))^*\\
Controller &\overset{\text{def}}{=} y := v_0; x := d_0; (\text{wait}\, p; wl?x; x \geq ub \longrightarrow y := 0; x \leq lb \longrightarrow y := 1; cv!y)^*
\end{aligned}
$$

where Q_{max}, π, r, g and p are constant system parameters, among which p is the period for *Controller* to interact with *Watertank*. In *Watertank*, v is the control variable taking the value of either 1 or 0 depending on whether the valve is open or not, and d is the water level variable. And v_0 and d_0 are the initial values of the control and water level variables, respectively. In *Controller*, variables x and y are used to record the values of d and v from *Watertank*, respectively. Furthermore, value transmissions are achieved via channels wl and cv.

The *Controller* works periodically. At the beginning of each period, it copies to x the value of water level d via channel wl and then computes the new value of y. y takes the value of 1 if $x \geq ub$ (the given upper bound), and 0 if $x \leq ub$ (the given lower bound). This new value is then sent back to v in *Watertank* along channel cv. The above-mentioned actions are assumed to be instantaneous. Then the water level d evolves following its own differential equation according to the status of the valve v during this period.

2.4 SystemC

SystemC is a system-level modeling language for hardware and software co-design. We choose SystemC as the target language because it is widely adopted in the design of embedded systems, thus more acceptable and practical. Moreover, its syntax and semantics are very close to HCSP's, e.g., the notion of time, communication and synchronization mechanisms, making the transformation between them more straightforward. We will give a brief introduction of SystemC in this subsection, please refer to [45] for a comprehensive understanding of SystemC.

Syntax. SystemC is a C++ class library and thus follows an object-oriented design philosophy, with most specific identifiers prefixed with *SC_* or *sc_* according to its naming convention. The main method is named *sc_main()*.

Modules are the basic blocks of a design hierarchy. A module is essentially a class containing an *SC_CTOR()* (constructor) which initializes and connects sub-designs, registers processes, etc. It often contains processes, ports, channels, events and helper functions as well.

Processes are member functions of modules, describing the actual functionality. There are two types of processes, *SC_METHOD* (method) and *SC_THREAD* (thread). A method can be invoked multiple times. Each time it begins and ends instantaneously. A thread, on the other hand, can be invoked only one time, but it can suspend and continue according to the *wait()* function. All the processes are orchestrated by the simulation kernel of SystemC.

Events ensure synchronization. To be specific, methods and threads waiting for an event will start or continue simultaneously once it occurs.

Ports and channels support communication between modules. According to the data transfer direction, there are *sc_in*, *sc_out* and *sc_inout* ports. The datatype of a port can be any C++, SystemC or user-defined type. Channels connect ports of the same datatype, hence sub-designs are connected. Data transmission is realized by calling methods of the channel such as *read()* and *write()*. One of the most commonly used channels is *sc_signal*$\langle\rangle$.

Helper functions can be called by processes or other helper functions normally.

For better understanding, a simple example is depicted below. Within the declaration of the module adder, a and b are input ports of integer type (line 3), sum is an output port of integer type (line 4), and do_add() is a process writing the value of a + b into sum (line 5-7). In the constructor, do_add is registered to the kernel as a method (line 9) with sensitive list a, b (line 10), which means that the method do_add will start whenever a or b changes.

```
1  // a simple example of adder
2  SC_MODULE(adder){ // module declaration
3      sc_in<int> a,b; // input ports
4      sc_out<int> sum; // output ports
5      void do_add(){ // process
6          sum.write(a.read() + b.read()); //adder
7      }
8      SC_CTOR(adder){ // constructor
9          SC_METHOD(do_add); // register to kernel
10         sensitive << a << b; // sensitivity list
11     }
12 };
```

Formal semantics. Unfortunately, no complete formal semantics for SystemC is given yet. Some related work on the semantics of subsets of SystemC is listed below. In [56], a simulation semantics of standard SystemC including basic SystemC operations and simulation scheduler is defined based on abstract state machines, and is extended to more complex components in [34]. After that, operational semantics, denotational semantics and algebraic semantics of the standard SystemC are studied in [57, 81–83]. But the problem is that some useful feature of SystemC such as general function statements are not considered. Recently, an executable semantics for standard SystemC is given based on guarded assignment systems in [74, 75] considering general function statements as well. It fits our purpose to define a relation between HCSP and SystemC very well, since guarded assignment systems are transition systems essentially. More details about the semantics of HCSP and SystemC will be introduced in Sec. 3 and Sec. 6, respectively.

3 APPROXIMATE BISIMULATION OF HCSP

In this section, we introduce a new approximate bisimulation relation with two precisions over transition systems inspired by [46] (Sec. 3.1), define the semantics of HCSP models based on transition systems (Sec. 3.2), and study the approximate bisimulation between two HCSP processes in bounded and unbounded time, respectively (Sec. 3.3).

3.1 Approximate Bisimulation over Transition Systems

Let $TS_i = \langle Q_i, L_i, \twoheadrightarrow_i, Q_i^0, Y, H_i\rangle$ $(i = 1, 2)$ be two metric transition systems with the same output set Y and metric $\mathbf{d}$. Let h and ε be the time and value precisions, respectively.

Definition 3.1 (Approximate bisimulation). A relation $\mathcal{B}_{h,\varepsilon} \subseteq Q_1 \times Q_2$ is called an (h, ε)-approximate bisimulation relation between TS_1 and TS_2, if for all $(\mathrm{q}_1, \mathrm{q}_2) \in \mathcal{B}_{h,\varepsilon}$:
1. $\mathrm{d}(H_1(\mathrm{q}_1), H_2(\mathrm{q}_2)) \le \varepsilon$,
2. $\forall \mathrm{q}_1 \overset{l}{\twoheadrightarrow}_1 \mathrm{q}_1', \exists \mathrm{q}_2 \overset{l'}{\Longrightarrow}_2 \mathrm{q}_2'$ s.t. $dis(l, l') \le h$ and $(\mathrm{q}_1', \mathrm{q}_2') \in \mathcal{B}_{h,\varepsilon}$, for $l \in L_1$ and $l' \in L_2$,
3. $\forall \mathrm{q}_2 \overset{l}{\twoheadrightarrow}_2 \mathrm{q}_2', \exists \mathrm{q}_1 \overset{l'}{\Longrightarrow}_1 \mathrm{q}_1'$ s.t. $dis(l, l') \le h$ and $(\mathrm{q}_1', \mathrm{q}_2') \in \mathcal{B}_{h,\varepsilon}$, for $l \in L_2$ and $l' \in L_1$.

Intuitively, approximate bisimulation requires the difference of observation values of the pair $(\mathrm{q}_1, \mathrm{q}_2)$ must be within the tolerance ε. Besides, if one of them is capable to reach a state via a single action, the other one can also reach a state via an action (possibly with sequences of τ before and after it) such that the distance between the two actions is within the tolerance h and the pair of derived states also satisfy the approximate bisimulation.

Definition 3.2. TS_1 and TS_2 are approximately bisimilar with precisions h and ε, denoted $TS_1 \cong_{h,\varepsilon} TS_2$, if there exists an (h, ε)-approximate bisimulation relation $\mathcal{B}_{h,\varepsilon}$ satisfying that for all $\mathrm{q}_1 \in Q_1^0$, there exists $\mathrm{q}_2 \in Q_2^0$ s.t. $(\mathrm{q}_1, \mathrm{q}_2) \in \mathcal{B}_{h,\varepsilon}$ and vice versa.

Now we can define the approximate bisimulation between two transition systems on time interval $[0, T]$, where $T \in \mathbb{R}^+ \cup \{+\infty\}$. $[0, T]$ is bounded if $T \in \mathbb{R}^+$, and unbounded if $T = +\infty$. It is easy to introduce a global clock for each transition system to record its execution time. The clock value is increased by the value of the delay action once it is performed. We can check approximately bisimulation between two transition systems on $[0, T]$ by only considering transitions which keep the global clock within the upper bound T.

For two (h, ε)-approximately bisimilar transition systems $TS_1 \cong_{h,\varepsilon} TS_2$, it is easy to deduce that $Reach(TS_1) \subseteq N(Reach(TS_2), \varepsilon)$ and $Reach(TS_2) \subseteq N(Reach(TS_1), \varepsilon)$ where $N(Y, \varepsilon)$ denotes the ε-neighborhood of Y, i.e., $N(Y, \varepsilon) = \bigcup_{\mathrm{y} \in Y}\{\mathrm{x} \mid \mathbf{d}(\mathrm{x}, \mathrm{y}) \le \varepsilon\}$. As an immediate consequence, this can be applied for safety verification of transition systems, especially whose reachable states cannot be precisely computed. For example, if the distance between $Reach(TS_1)$ and an unsafe set Y_U is greater than ε, $Reach(TS_2)$ must be disjoint from Y_U and hence safe. As shown in the rest of the paper, it is guaranteed that the original HCSP model, the discretized HCSP, and the resulting SystemC code all satisfy approximate bisimulation. Thus safety is preserved during the whole model transformation.

3.2 HCSP as Transition Systems

In this subsection we present an operational semantics of HCSP based on the notion of transition system in order to investigate approximate bisimulation between different HCSP processes. We will simply present its definition and give a few examples for better understanding. For more details, readers are referred to [76, 77].

A transition system $TS(S) = \langle Q, L, \rightarrow, Q^0, Y, H\rangle$ can embed an HCSP S as follows:

- $Q = (subp(S) \cup \{\surd\}) \times eval(S)$. Among them, $subp(S)$ is the set of all the sub-processes of S (i.e., all the processes occuring in S), and $\surd$ is the terminal process. $eval(S) = Var(S) \longrightarrow Val$ is the set of evaluations of variables in S where Val is the value space. A (process) state means an evaluation v instead of the state of the transition system. Besides, $fst(q)$ and $snd(q)$ mean the first and second components of a given state q, respectively.
- $L = \mathbb{R}_0^+ \cup InOut \cup \{\tau\}$. $\mathbb{R}_0^+$ stands for time delays, $InOut$ for the set of input and output communication events, and τ for a discrete action of HCSP such as assignment or evaluation of Boolean expressions.
- $Q^0 = \{(S, v) \mid v \in eval(S)\}$, meaning S is to execute from the initial process state v.
- $Y = \overrightarrow{Val}$ is the set of value vectors corresponding to $Var(S)$.
- $H(q) = vec(snd(q))$ for $q \in Q$, where function vec returns the value vector corresponding to the process state of q.
- $\rightarrow$ is the transition relation. A transition takes the form $(P, v) \xrightarrow{l} (P', v')$, indicating that process P executes to P' with v changing to v' by performing action l. It will be explained in detail in the remainder of this subsection.

Given sequential processes P_1, P_2 with transition systems $TS(P_1) = \langle Q_1, L_1, \rightarrow_1, Q_1^0, Y_1, H_1\rangle$ and $TS(P_2) = \langle Q_2, L_2, \rightarrow_2, Q_2^0, Y_2, H_2\rangle$, we can define the parallel transition system $TS(P_1\|P_2) = \langle Q, L, \rightarrow, Q^0, Y, H\rangle$ where

- $Q = ((subp(P_1) \cup \{\surd\})\|(subp(P_2) \cup \{\surd\})) \times \{v_1 \uplus v_2 \mid v_1 \in eval(P_1), v_2 \in eval(P_2)\}$, where given two sets of processes PS_1 and PS_2, $PS_1\|PS_2$ is defined as $\{\alpha\|\beta \mid \alpha \in subp(PS_1) \wedge \beta \in subp(PS_2)\}$; $v_1 \uplus v_2$ represents the disjoint union, i.e. $v_1 \uplus v_2(x)$ is $v_1(x)$ if $x \in Var(P_1)$, otherwise $v_2(x)$. This operator is well-defined because of the syntax restriction of HCSP, i.e. the processes in parallel do not share variables.
- $L = L_1 \cup L_2$.
- $Q^0 = \{(P_1\|P_2, v_1^0 \uplus v_2^0) \mid (P_i, v_i^0) \in Q_i^0 \text{ for } i = 1, 2\}$.
- $Y = Y_1 \times Y_2$.
- $H(q) = H_1(q) \times H_2(q)$.
- $\rightarrow$ is defined based on the parallel composition of transition relations of P_1 and P_2.

Below we present the transition rules (5), (6) for continuous evolution $\langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0\&B\rangle$, and (7) for synchronization, for illustration.

(5) and (6) define the transition relation for the time evolution case and termination case, respectively. In both rules, v denotes the initial state. For the evolution case (5), for any $d \geq 0$ such that B evaluates to true before d, the process can evolve for d time units according to F. $S(\cdot) : [0, d] \rightarrow \mathbb{R}^n$ defines the trajectory of F with initial value $v(\mathrm{s})$. And B holds under $v[\mathrm{s} \mapsto S(t)]$, the state at t (by substituting $S(t)$ for s), for any t in $[0, d)$. After d time units, the resulting process is $\langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0\&B\rangle$

with the new initial state updated to $v[\mathrm{s} \mapsto S(p)]$, keeping the other variables unchanged. For the termination case (6), B is false in current state v, so a τ- transition occurs leading to termination of the continuous evolution.

$$\frac{\begin{array}{c}\forall d > 0.\exists S(\cdot) : [0,d] \to \mathbb{R}^n.(S(0) = v(\mathrm{s}) \wedge (\forall t \in [0,d).\\ (F(\dot{S}(t), S(t)) = 0 \wedge v[\mathrm{s} \mapsto S(t)](B) = true)))\end{array}}{(\langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0 \& B\rangle, v) \xrightarrow{d} (\langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0 \& B\rangle, v[\mathrm{s} \mapsto S(d)])} \tag{5}$$

$$\frac{v(B) = false}{(\langle F(\dot{\mathrm{s}}, \mathrm{s}) = 0 \& B\rangle, v) \xrightarrow{\tau} (\surd, v)} \tag{6}$$

(7) defines the rule for communication synchronization. Suppose two transitions $(P_1, u) \xrightarrow{\alpha} (P_1', u')$ and $(P_2, v) \xrightarrow{\beta} (P_2', v')$ occur for P_1 and P_2, respectively. Let α be the receiving action along some channel ch (i.e., $\alpha = ch?x$), and β be the sending action along the same channel (i.e., $\beta = ch!e$), then the two actions in parallel will synchronize with each other, with an internal τ transition produced for the composite transition system.

$$\frac{(P_1, u) \xrightarrow{ch?x} (P_1', u'),\ (P_2, v) \xrightarrow{ch!e} (P_2', v')}{(P_1 \| P_2, u \uplus v) \xrightarrow{\tau} (P_1' \| P_2', u' \uplus v')} \tag{7}$$

3.3 Approximate bisimulation over HCSP processes

Let S_1 and S_2 be two HCSP processes, $T \in \mathbb{R}^+ \cup \{+\infty\}$ the upper bound of execution time, h the time precision, and ε the value precision.

Definition 3.3. S_1 and S_2 are called (h, ε)-*approximately bisimilar* on $[0, T]$, denoted by $S_1 \cong_{h,\varepsilon} S_2$ on $[0, T]$, if $TS(S_1) \cong_{h,\varepsilon} TS(S_2)$ holds on $[0, T]$ with an arbitrary initial state v_0, where $TS(S_1)$ and $TS(S_2)$ are the τ-compressed transition systems of S_1 and S_2, respectively.

We present Algorithm 1 for deciding whether two HCSP processes are approximately bisimilar within a time interval. The inputs of the algorithm include two HCSP processes S_1 and S_2, the initial state v_0, the time and value precisions h and ε, and a time bound T_b. The output is either **true** indicating $S_1 \cong_{h,\varepsilon} S_2$ on $[0, T_b]$, or **false** indicating that it does not hold. According to Def. 3.3, the algorithm will first construct the transition systems $TS(S_1)$ and $TS(S_2)$, and then check whether $TS(S_1) \cong_{h,\varepsilon} TS(S_2)$ holds or not.

In the initialization phase, $TS(S_m).Q^0$, the set of the initial states of the transition system $TS(S_m)$, is set to $\{(S_m, v_0)\}$, and the set of transitions $TS(S_m).T^0$ is initialized as $\emptyset$, for $m = 1, 2$. The iteration index i is assigned to 0. The time step d is set to d_ε, the minimal step size such that the precision ε is guaranteed for all ODEs in S_1 and S_2 (which will be presented in details in the subsequent section).

The algorithm takes two steps. The first step (lines 1-6) constructs the sets of reachable states and transitions of $TS(S_1)$ and $TS(S_2)$. The idea is to expand each set iteratively until reaching its fixpoint. At the i-th iteration, starting from a current state q in $TS(S_m).Q^i$, a new valid transition with label l is returned by function *ValidT*(q). The new transition is added to $TS(S_m).T^{i+1}$ (line 2), and its post state is added to $TS(S_m).Q^{i+1}$ (line 3). The repetition stops when a fixed point is reached, that is, $TS(S_m).T^i = TS(S_m).T^{i-1}$ (line 5). Then the state and transition sets $TS(S_m).Q$ and $TS(S_m).T$ are obtained (line 6). The function *ValidT*, given a source state q, returns a transition $q \overset{l}{\twoheadrightarrow} q'$ if it is enabled for q, and furthermore, if l is a time delay, l must be equal or less than d_ε and the global time after taking it must be within the bound T_b. Note that l is strictly less than d_ε only when the corresponding continuous evolution is interrupted or when the new global time reaches T_b.

Algorithm 1 $AB(S_1, S_2, v_0, h, \varepsilon, T_b)$ /* Deciding whether S_1 and S_2 are approximately bisimilar within time T_b */

```
Input: Processes $S_1, S_2$; initial state $v_0$; precisions $h$ and $\varepsilon$; time bound $T_b$;
Initialization: /* Initialize the initial state and transition set of transition systems generated by $S_1$ and $S_2$,
    the iteration index $i$, and step size $d_\varepsilon$ s.t. the $\varepsilon$ is guaranteed for all ODEs */
    $TS(S_m).Q^0 \leftarrow \{(S_m, v_0)\}$, $TS(S_m).T^0 \leftarrow \emptyset$ for $m = 1, 2$;
    $i \leftarrow 0$; $d \leftarrow d_\varepsilon$;
1: repeat
2:     /* add new elements into $TS(S_m).T$ and $TS(S_m).Q$ until a fixed point is reached */
       $TS(S_m).T^{i+1} \leftarrow TS(S_m).T^i \cup \{q \xrightarrow{l} q' \mid q \in TS(S_m).Q^i \wedge q \xrightarrow{l} q' \in ValidT(q)\}$;
3:     $TS(S_m).Q^{i+1} \leftarrow TS(S_m).Q^i \cup postState(TS(S_m).T^{i+1})$;
4:     $i \leftarrow i + 1$;
5: until $TS(S_m).T^i = TS(S_m).T^{i-1}$
6: /* the final reachable states and transitions of $TS(S_m)$ within $T_b$ */
   $TS(S_m).Q \leftarrow TS(S_m).Q^i$; $TS(S_m).T \leftarrow TS(S_m).T^i$;
7: /* Initialize the approximate bisimulation set for $S_1$ and $S_2$ */
   $\mathcal{B}^0_{h,\varepsilon} \leftarrow \{(q_1, q_2) \in TS(S_1).Q \times TS(S_2).Q \mid \mathbf{d}(H_1(q_1), H_2(q_2)) \leq \varepsilon\}$;
8: $i \leftarrow 0$;
9: repeat
10:    /* remove the pairs of states violating the approximate bisimulation from $\mathcal{B}^i_{h,\varepsilon}$ until a fixed point is
       reached */
       $\mathcal{B}^{i+1}_{h,\varepsilon} \leftarrow \{(q_1, q_2) \in B^i_{h,\varepsilon} \mid AppSim(q_1, q_2, \mathcal{B}^i_{h,\varepsilon}, h) \wedge AppSim(q_2, q_1, \mathcal{B}^i_{h,\varepsilon}, h)\}$;
11:    $i \leftarrow i + 1$;
12: until $\mathcal{B}^i_{h,\varepsilon} = \mathcal{B}^{i-1}_{h,\varepsilon}$
13: /* the final maximal approximate bisimulation relation */
    $\mathcal{B}_{h,\varepsilon} = \mathcal{B}^i_{h,\varepsilon}$;
14: if $((S_1, v_0), (S_2, v_0)) \in \mathcal{B}_{h,\varepsilon}$ then
15:     return true;
16: else
17:     return false;
18: end if
```

The second step (lines 7-18) checks whether S_1 and S_2 are approximately bisimilar with given precisions based on Def. 3.1 and Def. 3.2. At the beginning, $\mathcal{B}^0_{h,\varepsilon}$ stores all state pairs whose "distance" is not greater than ε (line 7), and the iteration index i is reset to 0 (line 8). Only the pairs satisfying approximate bisimulation are kept for the next iteration (lines 10-11). $AppSim(q_1, q_2, \mathcal{B}^i_{h,\varepsilon}, h)$ returns **true** if and only if for every transition $q_1 \xrightarrow{l}_1 q_1' \in TS(S_1).T$, there exists transition $q_2 \overset{l'}{\Longrightarrow}_2 q_2' \in TS(S_2).T$ such that $(q_1', q_2') \in \mathcal{B}^i_{h,\varepsilon}$ and $dis(l, l') \leq h$. And $AppSim(q_2, q_1, \mathcal{B}^i_{h,\varepsilon}, h)$ works symmetrically. When a fixed point $\mathcal{B}^i_{h,\varepsilon}$ is reached (line 12), the final approximate bisimulation relation $\mathcal{B}_{h,\varepsilon}$ (line 13) is exactly acquired. At last, the algorithm returns **true** if the pair of the initial states of $((S_1, v_0), (S_2, v_0))$ belongs to $\mathcal{B}_{h,\varepsilon}$ (line 14), indicating that $S_1 \cong_{h,\varepsilon} S_2$ on $[0, T_b]$ (line 15), and **false** otherwise (line 16-17).

Now we consider the decidability problem under bounded time ($T_b \in \mathbb{R}^+$) and unbounded time ($T_b = +\infty$), respectively.

Bounded time. For a given time step d and a bounded time T_b, the transition systems derived from S_1 and S_2 (i.e., $TS(S_1)$ and $TS(S_2)$) are *symbolic* (or *finite*). Thus, we can conclude that Algorithm 1 terminates, and thus whether $S_1 \cong_{h,\varepsilon} S_2$ on $[0, T_b]$ is decidable.

Theorem 3.4. *Given two HCSP processes with the same initial state v_0, it is decidable whether they are approximately bisimilar on* $[0, T]$ *where $T \in \mathbb{R}^+$ is an arbitrary real number.*

Proof. Let the two HCSP processes be S_1 and S_2. $TS(S_1)$ and $TS(S_2)$ denotes the transition systems constructed according to Algorithm 1, with any given initial state v_0.

We first claim that $TS(S_1)$ and $TS(S_2)$ are *symbolic*. As all ODEs in S_1 and S_2 are local Lipschitz, based on the theories of numerical solutions to ODEs, we can compute a bound d_ε such that for any discretization step $d \leq d_\varepsilon$, the error on each discretization step and the total error on $[0, T]$ are smaller than the precision ε. A method of computing such d_ε is given in the subsequent section. Since the number of τ and communication actions in S_1 and S_2 are finite, and T and d are positive real numbers, we can conclude that $TS(S_1)$ and $TS(S_2)$ are both *symbolic*.

After that, we can compute a maximal approximate bisimulation relation between them with given precisions h and ε. Then the problem whether $TS(S_1)$ and $TS(S_2)$ are (h, ε)-approximately bisimilar depends on whether all the pairs of their initial states belong to the maximal approximate bisimulation relation. As T is finite and the number of discrete actions is finite, Algorithm 1 will terminate in a finite number of steps.

Thus, we can conclude that whether two HCSP processes are approximately bisimilar on $[0, T]$ is decidable. □

Unbounded time. For $T_b = +\infty$, the transition systems derived from S_1 and S_2 are infinite and the algorithm may not terminate any more. However, if we require that all the ODEs in S_1 and S_2 satisfy the GAS condition (introduced in Sec. 2.1), the problem is still decidable. The basic idea is to approximate the reachable states of S_1 and S_2 as finite sets. Suppose the ODEs occurring in S_m are $F_1^m, \cdots, F_{k_m}^m$ and their equilibrium points are $x_1^m, \cdots, x_{k_m}^m$, respectively, for $m = 1, 2$. As a result, for each ODE F_j^m with $j \in \{1, \cdots, k_m\}$, there must exist a sufficiently large T_j^m such that the distance between the trajectory and its equilibrium point x_j^m can be sufficiently small after T_j^m time. Therefore, all the reachable states after T_j^m can be approximated as x_j^m and can be ignored without affecting the result. So Algorithm 1 terminates and the problem whether two HCSP processes are approximately bisimilar on $[0, +\infty]$ is decidable under the GAS assumption.

Theorem 3.5. *Given two HCSP processes with the same initial state v_0, if all the ODEs occurring in them are GAS, it is decidable whether they are approximately bisimilar on* $[0, +\infty]$.

Proof. Since the problem of deciding whether two HCSP processes under the GAS assumption are approximately bisimilar on $[0, +\infty]$ can always be reduced to the problem whether they are approximately bisimilar within a bounded interval (the latter problem is decidable from Theorem 3.4), we can conclude that Theorem 3.5 holds. □

4 DISCRETIZATION OF HCSP

Our approach to generating executable code from HCSP models proceeds in two phases: discretizing the HCSP model, and generating SystemC code from the discretized HCSP model. We mainly focus on the first phase in this section, leaving the second one to be introduced in Sec. 6.

Benefiting from the compositionality, the discretization of HCSP processes can be realized by defining a set of rules corresponding to its primitive constructs. As a result, discretizing an HCSP can be done in a compositional manner. Let S be an HCSP, $T \in \mathbb{R}^+ \cup \{+\infty\}$ be the upper bound of the execution time, and h and ε be the precisions for time and value, respectively. In the following, we explain how to construct a discrete HCSP process $D_{h,\varepsilon}(S)$ from S such that S is (h, ε)-approximately bisimilar to $D_{h,\varepsilon}(S)$ on the interval $[0, T]$, i.e., $S \cong_{h,\varepsilon} D_{h,\varepsilon}(S)$ holds on $[0, T]$. The crucial issue for the discretization of HCSP processes is to represent the continuous dynamics by a discrete approximation. To achieve this, we firstly propose a method for discretizing ODEs within a bounded interval in Sec. 4.1. Then, discretizing HCSP processes in bounded and unbounded time are discussed in Sec. 4.2 and Sec. 4.3, respectively. For better understanding, the approximate bisimulation relation between HCSP and its discretization will be discussed in Sec. 5 individually.

4.1 Discretizing Continuous Dynamics in Bounded Time

As analytical solutions to differential equations are difficult and impossible in general to compute, approximate solutions based on numeric computation become an alternative. Below we consider the discretization of an ODE within a bounded time $T_o \in \mathbb{R}^+$.

The ODE under consideration is $\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})$ with the initial condition $\mathrm{x}(t_0) = \widetilde{\mathrm{x}}_0$, where $\mathrm{x}, \widetilde{\mathrm{x}}_0 \in \mathbb{R}^n$, and $X(t, \widetilde{\mathrm{x}}_0)$ is assumed to be the solution of the initial value problem on the time interval $[t_0, t_0 + T_o]$. In what follows, h and ξ respectively represent the discretized time step and the precision of the discretization. In general ξ is less than the overall value precision ε for the processes under consideration throughout.

There are many well-established methods for numerically solving ODEs [66], among which the 4-stage Runge-Kutta method is more effective with higher precision. The *global discretization error* of it can be bounded by $O(h^4)$. By applying the 4-stage Runge-Kutta method, $\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})$ on $[t_0, t_0+T_o]$ is discretized as

$$(\text{wait } h; \mathrm{x} := \mathrm{x} + h\Phi(\mathrm{x}, h))^N; \text{wait } h'; \mathrm{x} := \mathrm{x} + h'\Phi(\mathrm{x}, h') \tag{8}$$

where $N = \lfloor \frac{T_o}{h} \rfloor$, $h' = T_o - Nh$, and $\Phi(\mathrm{x}, s) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ with $k_1 = \mathrm{f}(\mathrm{x})$, $k_2 = \mathrm{f}(\mathrm{x} + \frac{1}{2}sk_1)$, $k_3 = \mathrm{f}(\mathrm{x} + \frac{1}{2}sk_2)$ and $k_4 = \mathrm{f}(\mathrm{x} + sk_3)$. With the initial state x_0 at $h_0 = t_0$, the obtained sequence of approximate solutions $\{\mathrm{x}_i\}$ at time stamps $\{h_i\}$ satisfies

$$\begin{cases} \mathrm{x}_0, & h_0 = t_0, & \\ \mathrm{x}_i = \mathrm{x}_{i-1} + h\Phi(\mathrm{x}_{i-1}, h), & h_i = t_0 + i * h, & 1 \le i \le N \\ \mathrm{x}_{N+1} = \mathrm{x}_N + h'\Phi(\mathrm{x}_N, h'), & h_{N+1} = t_0 + T_o, & \text{if } Nh < T_o \end{cases} \tag{9}$$

Intuitively, T_o is divided into N intervals of length h and a possible residual interval of length h'. Φ is used for computing the approximated value of x based on the values of the vector field at the four points k_1, k_2, k_3 and k_4.

The discretization error at time h_i is defined as $\|X(h_i, \widetilde{\mathrm{x}}_0) - \mathrm{x}_i\|$, i.e., the distance between the exact solution and the approximate solution at h_i. According to Theorem **7.2.2.3** in [66], the error can be estimated as follows:

Proposition 4.1 (Global Error). *Let L be the Lipschitz constant of the ODE $\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})$ with the initial condition $\mathrm{x}(t_0) = \widetilde{\mathrm{x}}_0$, that is, for any compact set S of $\mathbb{R}^n$, $\|\mathrm{f}(\mathrm{y}_1) - \mathrm{f}(\mathrm{y}_2)\| \le L\|\mathrm{y}_1 - \mathrm{y}_2\|$ for all $\mathrm{y}_1, \mathrm{y}_2 \in S$. And $\mathrm{x}_0 \in \mathbb{R}^n$ is a state with $\|\mathrm{x}_0 - \widetilde{\mathrm{x}}_0\| \le \xi_1$. Then there exists a stepsize $h_e > 0$ s.t. for all $0 < h \le h_e$ and all $i \le \lceil \frac{T_o}{h} \rceil$, the global discretization error between $X(h_i, \widetilde{\mathrm{x}}_0)$ and x_i satisfies:*

$$\|X(h_i, \widetilde{\mathrm{x}}_0) - \mathrm{x}_i\| \le M(h), \tag{10}$$

where

$$\begin{aligned} M(h) &= \tfrac{e^{Lh'}-1}{L} C_2 (h')^4 + [1 + Lh' + \tfrac{(Lh')^2}{2} + \tfrac{(Lh')^3}{4} + \tfrac{(Lh')^4}{24}] M_N(h), \text{ and} \\ M_N(h) &= e^{NLh}\xi_1 + \tfrac{e^{NLh}-1}{L} C_1 h^4. \end{aligned} \tag{11}$$

Among them $N = \lfloor \frac{T_o}{h} \rfloor$, $h' = T_o - Nh$, h_i and x_i are as defined in (9), and C_1, C_2 are positive constants depending on the local discretization error of the 4-stage Runge-Kutta method.

Proof. For each $i \le N$, by Theorem **3** in [73] and Theorem **7.2.2.3** in [66], it follows that there exists a stepsize h_e such that for all $0 < h \le h_e$

$$\|X(h_i, \widetilde{\mathrm{x}}_0) - \mathrm{x}_i\| \le e^{NLh}\xi_1 + \frac{e^{NLh} - 1}{L} C_1 h^4$$

where C_1 is a positive constant depending on the local discretization error of the 4-stage Runge-Kutta method. In the following, we use $M_N(h)$ to denote the right side of the above inequality, i.e., $M_N(h) = e^{NLh}\xi_1 + \frac{e^{NLh}-1}{L}C_1h^4$.

When $i = N+1$, we introduce an auxiliary point $\mathrm{x}'_{N+1} = X(h_N, \widetilde{\mathrm{x}}_0) + h'\Phi(X(h_N, \widetilde{\mathrm{x}}_0), h')$, which is the approximate solution to $X(t, \widetilde{\mathrm{x}}_0)$ at h_{N+1} starting from $X(h_N, \widetilde{\mathrm{x}}_0)$ by using the 4-stage Runge-Kutta method with stepsize h'. Clearly

$$\|X(h_{N+1}, \widetilde{\mathrm{x}}_0) - \mathrm{x}'_{N+1}\| \le \frac{e^{Lh'}-1}{L}C_2(h')^4,$$

where C_2 is a constant. As $\mathrm{x}_{N+1} = \mathrm{x}_N + h'\Phi(\mathrm{x}_N, h')$,

$$\|\mathrm{x}'_{N+1} - \mathrm{x}_{N+1}\| \le \|X(h_N, \widetilde{\mathrm{x}}_0) - \mathrm{x}_N\| + h'\|\Phi(X(h_N, \widetilde{\mathrm{x}}_0), h') - \Phi(\mathrm{x}_N, h')\|.$$

Let $\Phi(X(h_N, \widetilde{\mathrm{x}}_0)) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ and $\Phi(\mathrm{x}_N, h') = \frac{1}{6}(k'_1 + 2k'_2 + 2k'_3 + k'_4)$ according to the definition of $\Phi(\mathrm{x}, s)$. Based on the Lipschitz condition, $\|k_1 - k'_1\| \le L\|X(h_N, \widetilde{\mathrm{x}}_0) - \mathrm{x}_N\| \le LM_N(h)$, $\|k_2 - k'_2\| \le L\|(X(h_N, \widetilde{\mathrm{x}}_0)) + \frac{h'}{2}k_1 - \mathrm{x}_N - \frac{h'}{2}k'_1\| \le L\|(X(h_N, \widetilde{\mathrm{x}}_0)) - \mathrm{x}_N\| + \frac{Lh'}{2}\|k_1 - k'_1\| \le (1 + \frac{Lh'}{2})LM_N(h)$, and analogously $\|k_3 - k'_3\| \le (1 + \frac{Lh'}{2} + \frac{(Lh')^2}{4})LM_N(h)$ and $\|k_4 - k'_4\| \le (1 + Lh' + \frac{(Lh')^2}{2} + \frac{(Lh')^3}{4})LM_N(h)$. Therefore

$$\begin{aligned} h'\|\Phi(X(h_N, \widetilde{\mathrm{x}}_0), h') - \Phi'(\mathrm{x}_N, h')\| &= h'\|\tfrac{k_1+2k_2+2k_3+k_4}{6} - \tfrac{k'_1+2k'_2+2k'_3+k'_4}{6}\| \\ &\le \tfrac{h'}{6}(\|k_1 - k'_1\| + 2\|k_2 - k'_2\| + 2\|k_3 - k'_3\| + |k_4 - k'_4\|) \\ &\le [Lh' + \tfrac{(Lh')^2}{2} + \tfrac{(Lh')^3}{6} + \tfrac{(Lh')^4}{24}]M_N(h), \end{aligned}$$

and

$$\|\mathrm{x}'_{N+1} - \mathrm{x}_{N+1}\| \le [1 + Lh' + \frac{(Lh')^2}{2} + \frac{(Lh')^3}{6} + \frac{(Lh')^4}{24}]M_N(h).$$

Hence we have

$$\begin{aligned} \|X(h_{N+1}, \widetilde{\mathrm{x}}_0) - \mathrm{x}_{N+1}\| &= \|X(h_{N+1}, \widetilde{\mathrm{x}}_0) - \mathrm{x}'_{N+1} + \mathrm{x}'_{N+1} - \mathrm{x}_{N+1}\| \\ &\le \|X(h_{N+1}, \widetilde{\mathrm{x}}_0) - \mathrm{x}'_{N+1}\| + \|\mathrm{x}'_{N+1} - \mathrm{x}_{N+1}\| \\ &\le \tfrac{e^{Lh'}-1}{L}C_2(h')^4 + [1 + Lh' + \tfrac{(Lh')^2}{2} + \tfrac{(Lh')^3}{6} + \tfrac{(Lh')^4}{24}]M_N(h). \end{aligned}$$

Let $M(h)$ denote $\frac{e^{Lh'}-1}{L}C_2(h')^4 + [1 + Lh' + \frac{(Lh')^2}{2} + \frac{(Lh')^3}{6} + \frac{(Lh')^4}{24}]M_N(h)$, the upper bound of $\|X(h_{N+1}, \widetilde{\mathrm{x}}_0) - \mathrm{x}_{N+1}\|$, where $h' = T_o - \lfloor \frac{T_o}{h} \rfloor h$.

As $L > 0$ and $h' \ge 0$, it is obvious that $M(h) > M_N(h)$. Therefore, for any $0 < h \le h_e$ and all $i \le \lceil \frac{T_o}{h} \rceil$, the global discretization error between $X(h_i, \widetilde{\mathrm{x}}_0)$ and x_i can be bounded by $M(h)$. This completes the proof. □

By Proposition 4.1, we can prove that an ODE and its discretization are approximately bisimilar on bounded time.

Theorem 4.2 (Correctness of Discretization of ODEs). *Assume L is the Lipschitz constant of the ODE $\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})$ with the initial condition $\mathrm{x}(t_0) = \widetilde{\mathrm{x}}_0$, and x_0 the initial approximation satisfies $\|\mathrm{x}_0 - \widetilde{\mathrm{x}}_0\| \le \xi_1$. For any precision $\xi > \xi_1 > 0$, there exists $h > 0$ s.t.*

$$\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}),\ \mathrm{x}(t_0) = \widetilde{\mathrm{x}}_0. \tag{12}$$

and

$$\mathrm{x} := \mathrm{x}_0; (\mathit{wait}\ h; \mathrm{x} := \mathrm{x} + h\Phi(\mathrm{x}, h))^N; \mathit{wait}\ h'; \mathrm{x} := \mathrm{x} + h'\Phi(\mathrm{x}, h') \tag{13}$$

are (h, ξ)-approximately bisimilar on $[t_0, t_0 + T_o]$, where $N = \lfloor \frac{T_o}{h} \rfloor$ and $h' = T_o - Nh$.

Proof. For convenience, P and DP are used to denote the transition systems of the ODE and its discrete result over $[t_0, t_0 + T_o]$, respectively. P has infinitely many states $\{X(t, \widetilde{\mathrm{x}}_0) \mid t \in [t_0, t_0 + T_o]\}$ and transitions $\{X(t, \widetilde{\mathrm{x}}_0) \overset{t'-t}{\twoheadrightarrow}_P X(t', \widetilde{\mathrm{x}}_0) \mid t < t'\}$. DP, on the contrary, has finite number of states $\{\mathrm{x}_i\}_{0\le i\le N+1}$ and transitions $\{\mathrm{x}_i \overset{h}{\twoheadrightarrow}_{DP} \mathrm{x}_i\}_{0\le i\le N} \cup \{\mathrm{x}_{N+1} \overset{h'}{\twoheadrightarrow}_{DP} \mathrm{x}_{N+1}\} \bigcup \{\mathrm{x}_i \overset{0}{\twoheadrightarrow}_{DP} \mathrm{x}_i\}_{1\le i\le N+1} \bigcup \{\mathrm{x}_i \overset{\tau}{\twoheadrightarrow}_{DP} \mathrm{x}_{i+1}\}_{0\le i\le N}$. Here h' is assumed to be positive without loss of generality.

Let $\mathcal{B}_{h,\xi} = \{(X(t_0, \widetilde{\mathrm{x}}_0), \mathrm{x}_0)\} \cup \{(X(t_i, \widetilde{\mathrm{x}}_0), \mathrm{x}_{i+1}) \mid t_i \in (h_i, h_{i+1}] \wedge 0 \le i \le \lceil \frac{T_o}{h} \rceil - 1\}$. This means that the exact values $X(t_i, \widetilde{\mathrm{x}}_0)$ where $t \in (h_i, h_{i+1}]$ are all approximated as the discrete value x_{i+1}. The subscription i of t_i emphasizes the fact that t_i is from the i-th interval $(h_i, h_i + 1]$. Especially, $\widetilde{\mathrm{x}}_0$ is approximated as x_0.

We first prove there exists such h that $\|X(t_i, \widetilde{\mathrm{x}}_0) - \mathrm{x}_{i+1}\| \le \xi$ for each pair $(X(t_i, \widetilde{\mathrm{x}}_0), \mathrm{x}_{i+1}) \in \mathcal{B}_{h,\xi}$. Firstly we have $\|X(t_i, \widetilde{\mathrm{x}}_0) - \mathrm{x}_{i+1}\| \le \|X(t_i, \widetilde{\mathrm{x}}_0) - X(h_{i+1}, \widetilde{\mathrm{x}}_0)\| + \|X(h_{i+1}, \widetilde{\mathrm{x}}_0) - \mathrm{x}_{i+1}\|$. If $t_i = h_{i+1}$, $\|X(t_i, \widetilde{\mathrm{x}}_0) - X(h_{i+1}, \widetilde{\mathrm{x}}_0)\| = 0$. For $t_i < h_{i+1}$, since the function $X(t, \widetilde{\mathrm{x}}_0)$ is continuous on $[t_i, h_{i+1}]$ and differentiable on (t_i, h_{i+1}) (by Theorem **(7.1.1)** of [66]), there always exists some $t'_i \in (t_i, h_{i+1})$ such that $\mathrm{f}(X(t'_i, \widetilde{\mathrm{x}}_0)) = \frac{X(h_{i+1}, \widetilde{\mathrm{x}}_0) - X(t_i, \widetilde{\mathrm{x}}_0)}{h_{i+1} - t_i}$. So we have $\|X(h_{i+1}, \widetilde{\mathrm{x}}_0) - X(t_i, \widetilde{\mathrm{x}}_0)\| = (h_{i+1} - t_i)\|\mathrm{f}(X(t'_i, \widetilde{\mathrm{x}}_0))\|$. Let D_m denote max $\|\mathrm{f}(X(t, \widetilde{\mathrm{x}}_0))\|$ on $[t_0, t_0 + T_o]$. If $D_m = 0$, $\mathrm{f}(X(t, \widetilde{\mathrm{x}}_0)) = \mathbf{0}$ on $[t_0, t_0 + T_o]$ and $\|X(h_{i+1}, \widetilde{\mathrm{x}}_0) - X(t_i, \widetilde{\mathrm{x}}_0)\| = 0$. If $D_m > 0$, choose $h \le h_1 = \frac{\xi - \xi_1}{2D_m}$, then $\|X(h_{i+1}, \widetilde{\mathrm{x}}_0) - X(t_i, \widetilde{\mathrm{x}}_0)\| \le (h_{i+1} - t_i)D_m < \frac{\xi - \xi_1}{2}$ for all i and all t_i. And from Proposition 4.1, $\|X(h_{i+1}, \widetilde{\mathrm{x}}_0) - \mathrm{x}_{i+1}\| \le M(h)$. $M(h)$ tends to ξ_1 as $h \to 0$, so there exists h_2 such that $M(h) < \frac{\xi + \xi_1}{2}$ for $h \le h_2$.

Therefore, choose $h = \min\{h_1, h_2\}$, then the "distance" between $X(t_i, \widetilde{\mathrm{x}}_0)$ and x_{i+1} for all i and all t_i can be bounded by $\|X(t_i, \widetilde{\mathrm{x}}_0) - \mathrm{x}_{i+1}\| < \frac{\xi - \xi_1}{2} + \frac{\xi + \xi_1}{2} = \xi$.

Below we prove that the relation $\mathcal{B}_{h,\xi}$ is a bisimulation relation between P and DP.

Consider the transition $X(t, \widetilde{\mathrm{x}}_0) \overset{t'-t}{\twoheadrightarrow}_P X(t', \widetilde{\mathrm{x}}_0)$ in P. If t' and t are from the same interval $(h_i, h_{i+1}]$, the corresponding transition of DP should be $\mathrm{x}_i \overset{0}{\twoheadrightarrow}_{D(P)} \mathrm{x}_i$. Obviously the distance is $t' - t < h$. If t' is from a different interval $(h_j, h_{j+1}]$, the corresponding compressed transition should be $\mathrm{x}_{i+1} \overset{(j-i)h}{\Longrightarrow}_{DP} \mathrm{x}_{j+1}$. Here the distance $|(t' - t) - (j - i)h| < h$.

For $\mathrm{x}_i \overset{0}{\twoheadrightarrow}_{DP} \mathrm{x}_i$, transitions in P are $X(h_{i-1} + \delta, \widetilde{\mathrm{x}}_0) \overset{h-\delta}{\twoheadrightarrow}_P X(h_i, \widetilde{\mathrm{x}}_0)$. For $\mathrm{x}_i \overset{\tau}{\twoheadrightarrow}_{DP} \mathrm{x}_{i+1}$, transitions are $X(h_i, \widetilde{\mathrm{x}}_0) \overset{\delta}{\twoheadrightarrow}_P X(h_i + \delta, \widetilde{\mathrm{x}}_0)$. For $\mathrm{x}_i \overset{h}{\twoheadrightarrow}_{DP} \mathrm{x}_i$ (resp. $\mathrm{x}_N \overset{h'}{\twoheadrightarrow}_{DP} \mathrm{x}_N$) transitions are $X(h_{i-1} + \delta, \widetilde{\mathrm{x}}_0) \overset{h-\delta}{\twoheadrightarrow}_P X(h_i, \widetilde{\mathrm{x}}_0)$ (resp. $X(h_{N-1} + \delta, \widetilde{\mathrm{x}}_0) \overset{h'-\delta}{\twoheadrightarrow}_P X(h_N, \widetilde{\mathrm{x}}_0)$).

In conclusion P and DP are (h, ξ)-approximately bisimilar on $[t_0, t_0 + T_o]$. □

4.2 Discretization of HCSP in Bounded Time

We continue to consider the discretization of HCSP processes. Below, we denote $D_{h,\varepsilon}(S)$ by the discretized process of S with time step h and value precision ε. $D_{h,\varepsilon}(S)$ within time $T \in \mathbb{R}^+$ is listed in Table 2, with the original process above the line and the discretized one below for each rule. Generally, the discretization of ODEs is as in the previous section. We replace the guard B with $N(B, \varepsilon)$, a Boolean expression that holds in the ε-neighborhood of B, i.e., $N(B, \varepsilon) = \bigcup_{\mathrm{y}\in B}\{\mathrm{x} | \mathbf{d}(\mathrm{x}, \mathrm{y}) \le \varepsilon\}$. For instance, $N(x > 2, \varepsilon)$ is $x > 2 - \varepsilon$. When B is true, $N(B, \varepsilon)$ is also true. We also test whether $N(B, \varepsilon)$ holds for the next discretized step, i.e., $N'(B, \varepsilon) := N(B, \varepsilon)[\mathrm{x} \mapsto \mathrm{x} + h\Phi(\mathrm{x}, h)]$. Thus, the discretized process will terminate in accord with the original continuous evolution. Besides, Boolean variables $ch?$ and $ch!$ are introduced for each channel ch to represent the readiness of the input and output events. Let $\overline{ch*}$ be the dual of $ch*$, e.g., if $ch* = ch?$, then $\overline{ch*} = ch!$ and vice versa. We use $\fatsemi_{i\in I}(io_i := 1)$ to represent the sequential composition of setting all the readiness variables io_i in I

$$\frac{\textbf{skip}}{\textbf{skip}} \quad \frac{\textbf{stop}}{\textbf{stop}} \quad \frac{x := e}{x := e} \quad \frac{\text{wait } d}{\text{wait } d}$$

$$\frac{ch?x}{ch? := 1; ch?x; ch? := 0} \qquad \frac{ch!e}{ch! := 1; ch!e; ch! := 0}$$

$$\frac{\talloblong_{i\in I}(io_i \longrightarrow Q_i)}{\fatsemi_{i\in I}(io_i := 1); \talloblong_{i\in I}(io_i \longrightarrow (\fatsemi_{i\in I}(io_i := 0); D_{h,\varepsilon}(Q_i)))}$$

$$\frac{P; Q}{D_{h,\varepsilon}(P); D_{h,\varepsilon}(Q)} \quad \frac{B \longrightarrow P}{B \longrightarrow D_{h,\varepsilon}(P)} \quad \frac{P \sqcap Q}{D_{h,\varepsilon}(P) \sqcap D_{h,\varepsilon}(Q)}$$

$$\frac{P^*}{(D_{h,\varepsilon}(P))^*} \qquad \frac{P\|Q}{D_{h,\varepsilon}(P)\|D_{h,\varepsilon}(Q)}$$

$$\frac{\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B\rangle}{\begin{array}{c}(N(B,\varepsilon) \wedge N'(B,\varepsilon) \longrightarrow (\text{wait } h; \mathrm{x} := \mathrm{x} + h\Phi(\mathrm{x}, h)))^{\lfloor \frac{T}{h} \rfloor}; \\ N(B,\varepsilon) \wedge N'(B,\varepsilon) \longrightarrow (\text{wait } h'; \mathrm{x} := \mathrm{x} + h'\Phi(\mathrm{x}, h')); \\ N(B,\varepsilon) \wedge N'(B,\varepsilon) \longrightarrow \textbf{stop}\end{array}}$$

$$\frac{\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B\rangle \unrhd \talloblong_{i\in I}(io_i \longrightarrow Q_i)}{\begin{array}{c}\fatsemi_{i\in I}(io_i := 1); \\ (N(B,\varepsilon) \wedge N'(B,\varepsilon) \wedge \wedge_{i\in I}(io_i \wedge \neg\overline{io_i}) \longrightarrow (\text{wait } h; \mathrm{x} := \mathrm{x} + h\Phi(\mathrm{x}, h)))^{\lfloor \frac{T}{h} \rfloor}; \\ (N(B,\varepsilon) \wedge N'(B,\varepsilon) \wedge \wedge_{i\in I}(io_i \wedge \neg\overline{io_i}) \longrightarrow (\text{wait } h'; \mathrm{x} := \mathrm{x} + h'\Phi(\mathrm{x}, h'))); \\ \neg(N(B,\varepsilon) \wedge N'(B,\varepsilon)) \wedge \wedge_{i\in I}(io_i \wedge \neg\overline{io_i}) \longrightarrow \fatsemi_{i\in I}(io_i := 0); \\ \vee_{i\in I}(io_i \wedge \overline{io_i}) \longrightarrow (\talloblong_{i\in I}(io_i \longrightarrow (\fatsemi_{i\in I}(io_i := 0); D_{h,\varepsilon}(Q_i)))); \\ (N(B,\varepsilon) \wedge N'(B,\varepsilon) \wedge \wedge_{i\in I}(io_i \wedge \neg\overline{io_i})) \longrightarrow \textbf{stop}\end{array}}$$

Table 2. The rules for discretization of HCSP in bounded time.

to 1, that is, $io_1 := 1; \ldots; io_{|I|} := 1$. And $\fatsemi_{i\in I}(io_i := 0)$ is defined similarly. Details are explained as follows:

- **skip**, **stop**, $x := e$ and wait d remain unchanged, as they do not need discretization.
- The input process $ch?x$ is discretized as: set $ch?$ to 1 (ready); read; reset $ch?$ to 0 (not ready). The output process $ch!e$ is handled in the same way.
- For $\talloblong_{i\in I}(io_i \longrightarrow Q_i)$, similarly, first set to 1 all the readiness variables of $\{io_i\}_{i\in I}$, and reset them to 0 as soon as one of them occurs, followed by the discretized result of the corresponding sequential process Q_i.
- Compound constructs including sequential composition $P; Q$, alternative construct $B \longrightarrow P$, internal choice $P \sqcap Q$, repetition P^* and parallel composition $P\|Q$ are discretized recursively.
- The continuous statement $\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B\rangle$ is discretized on $[0, T]$ by Theorem 4.2. When the $N(B,\varepsilon)$- and $N'(B,\varepsilon)$-neighborhood guards hold, assign $\mathrm{x} + h\Phi(\mathrm{x}, h)$ to x every h time units for $\lfloor \frac{T}{h} \rfloor$ times, and then assign $\mathrm{x} + h'\Phi(\mathrm{x}, h')$ to x for the residual interval $[\lfloor \frac{T}{h} \rfloor h, T]$. If the two guards still hold after that, we use **stop** to terminate the whole process. Otherwise, if $N(B,\varepsilon) \wedge N'(B,\varepsilon)$ turns false before T (near the exact termination time point, which is guaranteed by the robustly-safe requirement discussed in the next section), x will keep unchanged and the discretized process terminates.
- The communication interrupt $\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B\rangle \unrhd \talloblong_{i\in I}(io_i \longrightarrow Q_i)$ can be discretized similarly. At the beginning, all the readiness variables corresponding to $\{io_i\}_I$ are set to 1. Act as the discretized $\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B\rangle$ does when there occurs no communications. If the two neighborhood guards are violated but no communication gets ready, the process will terminate and

all its readiness variables are reset to 0. If some io_i occur, take an external choice and go on to the corresponding discretized Q_i.

Example 4.3. Following the rules in Table 2, a discretized system $WTS_{h,\varepsilon}$ of the water tank system in Example 2.3 is obtained as follows:

$$
\begin{array}{rcl}
WTS_{h,\varepsilon} & \overset{\text{def}}{=} & Watertank_{h,\varepsilon} \| Controller_{h,\varepsilon} \\
Watertank_{h,\varepsilon} & \overset{\text{def}}{=} & v := v_0; d := d_0; \\
& & (v = 1 \longrightarrow (wl! := 1; (wl! \wedge \neg wl? \longrightarrow (wait\ h; d = d + h\Phi(\mathrm{x}, h)))^{\lfloor \frac{T}{h} \rfloor}; \\
& & wl! \wedge \neg wl? \longrightarrow (wait\ h'; d = d + h'\Phi(\mathrm{x}, h')); \\
& & wl! \wedge wl? \longrightarrow (wl!d; wl! := 0; cv? := 1; cv?v; cv? := 0); \\
& & wl! \wedge \neg wl? \longrightarrow \mathbf{stop}); \\
& & v = 0 \longrightarrow (wl! := 1; (wl! \wedge \neg wl? \longrightarrow (wait\ h; d = d + h\Phi(\mathrm{x}, h)))^{\lfloor \frac{T}{h} \rfloor}; \\
& & wl! \wedge \neg wl? \longrightarrow (wait\ h'; d = d + h'\Phi(\mathrm{x}, h')); \\
& & wl! \wedge wl? \longrightarrow (wl!d; wl! := 0; cv? := 1; cv?v; cv? := 0); \\
& & wl! \wedge \neg wl? \longrightarrow \mathbf{stop}))^* \\
Controller_{h,\varepsilon} & \overset{\text{def}}{=} & y := v_0; x := d_0; (wait\ p; wl? := 1; wl?x; wl? := 0; \\
& & x \geq ub \longrightarrow y := 0; x \leq lb \longrightarrow y := 1; cv! := 1; cv!y; cv! := 0)^*
\end{array}
$$

4.3 Discretization of HCSP in Unbounded Time

In the unbounded time case, we require that all ODEs satisfy the GAS condition. Actually, most rules are the same as in bounded time, except for those containing ODEs, i.e. continuous evolution and continuous interrupt, shown in Table 3.

Here we take as an example the continuous evolution $\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B \rangle$. If $\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})$ is GAS, there must exist a sufficiently large T_{et} called *equilibrium time* after which the distance between the actual trajectory and the equilibrium point $\bar{\mathrm{x}}$ is less than ε. Therefore, it turns to the bounded time case with time bound T_{et}, except that the value of the continuous evolution is approximated by its equilibrium point $\bar{\mathrm{x}}$ after T_{et}. The communication interrupt is handled similarly.

5 CORRECTNESS OF THE DISCRETIZATION

In this section, we consider the correctness of the discretization investigated in the previous section, i.e., whether the discretized HCSP $D_{h,\varepsilon}(S)$ defined as above (Sec. 4) is approximately bisimilar to the original HCSP S in given time step and value precision. This is hard to be held in general, as the precisions, however small, may lead to different control flows when executing S and $D_{h,\varepsilon}(S)$. To avoid this, we need to put on S an extra condition called robustly safety. Below we will first present the notion of robustly safety, then propose an algorithm for calculating the robustly-safe parameters for a given HCSP, and finally investigate the approximate bisimulation between HCSP and its discretization under this condition.

5.1 Robustly Safe Processes

Boolean conditions are contained in three kinds of constructs, $B \longrightarrow P$, $\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B \rangle$ and $\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B \rangle \trianglerighteq []_{i \in I}(io_i \longrightarrow Q_i)$. In each case, execution decisions are made depending on the truth values of the Boolean condition B. Intuitively speaking, S and $D_{h,\varepsilon}(S)$ are expected to make the same choice at proximate timepoints if they start executing from the same initial state. Especially,

$$\dfrac{\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})\&B\rangle}{\begin{array}{c}(N(B,\varepsilon) \wedge N'(B,\varepsilon) \longrightarrow (\text{wait } h; \mathrm{x} := \mathrm{x} + h\Phi(\mathrm{x},h)))^{\lfloor \frac{T_{et}}{h} \rfloor};\\ N(B,\varepsilon) \wedge N'(B,\varepsilon) \longrightarrow (\text{wait } h'; \mathrm{x} := \mathrm{x} + h'\Phi(\mathrm{x},h'));\\ N(B,\varepsilon) \wedge N'(B,\varepsilon) \longrightarrow (\mathrm{x} = \bar{\mathrm{x}}; \mathbf{stop})\end{array}}$$

$$\dfrac{\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})\&B\rangle \trianglerighteq [\!]_{i\in I}(io_i \longrightarrow Q_i)}{\begin{array}{c}\fatsemi_{i\in I}(io_i := 1);\\ (N(B,\varepsilon) \wedge N'(B,\varepsilon) \wedge \wedge_{i\in I}(io_i \wedge \neg\overline{io_i}) \longrightarrow (\text{wait } h; \mathrm{x} := \mathrm{x} + h\Phi(\mathrm{x},h)))^{\lfloor \frac{T_{et}}{h} \rfloor};\\ (N(B,\varepsilon) \wedge N'(B,\varepsilon) \wedge \wedge_{i\in I}(io_i \wedge \neg\overline{io_i}) \longrightarrow (\text{wait } h'; \mathrm{x} := \mathrm{x} + h'\Phi(\mathrm{x},h')));\\ \neg(N(B,\varepsilon) \wedge N'(B,\varepsilon)) \wedge \wedge_{i\in I}(io_i \wedge \neg\overline{io_i}) \longrightarrow \fatsemi_{i\in I}(io_i := 0);\\ \vee_{i\in I}(io_i \wedge \overline{io_i}) \longrightarrow ([\!]_{i\in I}(io_i \longrightarrow (\fatsemi_{i\in I}(io_i := 0); D_{h,\varepsilon}(Q_i))));\\ (N(B,\varepsilon) \wedge N'(B,\varepsilon) \wedge \wedge_{i\in I}(io_i \wedge \neg\overline{io_i})) \longrightarrow (\mathrm{x} = \bar{\mathrm{x}}; \mathbf{stop})\end{array}}$$

Table 3. The rules for discretization of HCSP in unbounded time.

for $\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})\&B\rangle$, any violation of the boundary condition should be consistent in both the original and the discretized version. To this end we propose the (δ, ϵ)-robustly safe condition.

Let $\epsilon > 0$ be a precision. The ϵ-neighbourhood of a point v is denoted as $U(v,\epsilon) = \{\mathrm{y} \mid \mathrm{d}(\mathrm{y}, v) < \epsilon\}$. The $(-\epsilon)$-neighborhood of a formula ϕ is denoted as $N(\phi, -\epsilon) = \{\mathrm{x} \in \phi \mid \forall \mathrm{y} \in \neg\phi. \|\mathrm{x}-\mathrm{y}\| > \epsilon\}$. Intuitively, $\mathrm{x} \in N(\phi, -\epsilon)$ means that x is inside ϕ and that the distance between x and ϕ's boundary is greater than ϵ.

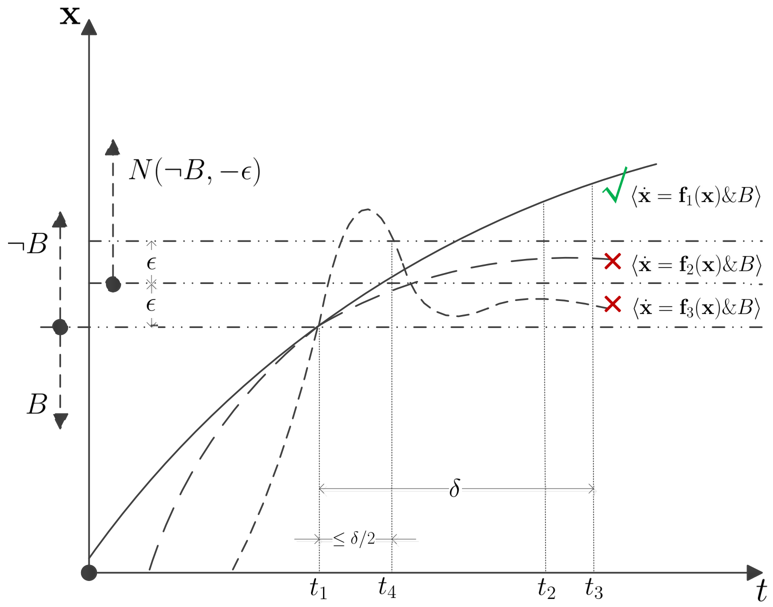
Definition 5.1 ((δ,ϵ)-robustly safe). Let $\delta > 0$ and $\epsilon > 0$ be the time and value precisions, respectively. An HCSP S is (δ, ϵ)-robustly safe w.r.t. a given initial state v_0, if the following two conditions hold:

(a) for every alternative process $B \longrightarrow P$ with B depending on continuous variables in S, the state v right before $B \longrightarrow P$ satisfies $v \in N(B, -\epsilon) \cup N(\neg B, -\epsilon)$;
(b) for every continuous evolution $\langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})\&B\rangle$ which ends up in state v at t because of the violation of B (hence $v(B) = \mathit{false}$), there exists $\widehat{t} \in (t, t+\delta)$ s.t. $U(v[\mathrm{x} \mapsto X(\widehat{t}, \widetilde{\mathrm{x}}_0)], \epsilon) \subseteq N(\neg B, -\epsilon)$, where $\widetilde{\mathrm{x}}_0$ denotes the initial value of the ODE.

S is said to be (δ, ϵ)-robustly safe if it is (δ, ϵ)-robustly safe w.r.t. any initial state.

Condition (a) means that, when entering $B \longrightarrow P$, the actual state v should fall into the $(-\epsilon)$-neighborhood of B or $\neg B$. Thus if $v \in N(B, -\epsilon)$ (or $v \in N(\neg B, -\epsilon)$) and the value precision of the discretization $\varepsilon < \epsilon$, the discrete state v' with $\mathrm{d}(v, v') \leq \varepsilon$ is also in B (or $\neg B$). Condition (b) means that, in less than δ time after B is violated, the ODE will reach a state in $\neg B$ which is more than 2ϵ far away from B's boundary. So, if $h < \delta < 2h$ and $\varepsilon < \epsilon$, (δ, ϵ)-robustly safety guarantees that any violation of the boundary condition in the continuous evolution to be detected in the discretized process, in a time bounded by h, and therefore the continuous and the discrete processes are (h, ε)-approximately bisimilar.

Example 5.2. For a better understanding, we illustrate the satisfiability and violation of (δ, ϵ)-robustly safe condition for two continuous statements in Fig. 1. As shown in the figure, suppose the Boolean condition B in all $\langle \dot{\mathrm{x}} = \mathrm{f}_1(\mathrm{x})\&B\rangle$, $\langle \dot{\mathrm{x}} = \mathrm{f}_2(\mathrm{x})\&B\rangle$ and $\langle \dot{\mathrm{x}} = \mathrm{f}_3(\mathrm{x})\&B\rangle$ turns $false$ at time t_1, meaning that the three continuous statements will terminate at t_1. Consider the discretized processes for them according to Table 2 with the value precision ε less than ϵ and the time precision h in $(\frac{\delta}{2}, \delta)$. Then we hope that the corresponding discretized process for either f_1, f_2 and f_3



x
$N(\neg B, -\epsilon)$
$\neg B$
B
ϵ
ϵ
$\langle \dot{x} = f_1(x) \& B \rangle$
$\langle \dot{x} = f_2(x) \& B \rangle$
$\langle \dot{x} = f_3(x) \& B \rangle$
δ
$\leq \delta/2$
t_1
t_4
t_2
t_3
t

Fig. 1. The examples of (δ, ϵ)-robustly safe continuous statements (Color figure online).

terminates at a time close to t_1, i.e. the distance between it and t_1 should be less than the given time precision h. From the above explanation, it is required that the ODE in the continuous statements should step forward for at least 2ϵ distance within δ. As shown in Fig. 1, $\langle \dot{x} = f_1(x) \& B \rangle$ satisfies (δ, ϵ)-robustly safety, $\langle \dot{x} = f_3(x) \& B \rangle$ violates the time constraint, and $\langle \dot{x} = f_2(x) \& B \rangle$ violates the value constraint.

We have the following theorem for robustly safe processes.

Theorem 5.3. *(1) If S is (δ, ϵ)-robustly safe w.r.t. an initial state v_0, then S is (δ, ϵ')-robustly safe w.r.t. v_0, for any $0 < \epsilon' < \epsilon$.*
(2) If S is (δ, ϵ)-robustly safe w.r.t. v_0, then S is (δ', ϵ)-robustly safe w.r.t. v_0, for any $\delta' > \delta$.

Proof. (1) δ is fixed. (a) Since S is (δ, ϵ)-robustly safe, $v \in N(B, -\epsilon) \cup N(\neg B, -\epsilon)$. As $\epsilon' < \epsilon$, we have $N(B, -\epsilon) \subset N(B, -\epsilon')$ and $N(\neg B, -\epsilon) \subset N(\neg B, -\epsilon')$, so obviously $v \in N(B, -\epsilon') \cup N(\neg B, -\epsilon')$. (b) Let $\widehat{t'} = \widehat{t}$, then we have $U(v[x \mapsto X(\widehat{t'}, \widetilde{x}_0)], \epsilon') \subset U(v[x \mapsto X(\widehat{t}, \widetilde{x}_0)], \epsilon) \subseteq N(\neg B, -\epsilon) \subseteq N(\neg B, -\epsilon')$. (2) ϵ is fixed. (a) The first condition has nothing to do with δ, and naturally holds for any $\delta' > \delta$. (b) Choose $\widehat{t'} = \widehat{t}$. Obviously $\widehat{t'} \in (t, t + \delta')$. So the condition is satisfied. □

Theorem 5.3 indicates the fact that a given HCSP S is (δ, ϵ)-robustly safe for any $\delta \geq \delta_{\min}$ and $\epsilon \leq \epsilon_{\max}$ if there exists $\delta_{\min}$ and $\epsilon_{\max}$ such that it is $(\delta_{\min}, \epsilon_{\max})$-robustly safe. Next we will present algorithms for returning the valid scopes of δ and ϵ by computing the values of $\delta_{\min}$ and $\epsilon_{\max}$, given an HCSP with an initial state.

5.2 Computing Parameters ϵ and δ

By Def. 5.1, we first compute the value of ϵ based on the Boolean conditions of the alternative constructs (that is, those B_i's in $B_i \longrightarrow P_i$, see Alg. 2), and then the value of δ based on the calculated ϵ and the domain conditions of the continuous evolutions (the B_i's in Alg. 3).

Algorithm 2 *Com_ϵ*(S, v_0) /* Computing the scope of ϵ */

```
Input: Process $S$ and its initial state $v_0$;
1: $c = +\infty$; /* The upper bound of $\epsilon$ is initialized to $+\infty$ */
   /* Find the minimal upper bound from all alternative statements in $S$ */
2: for all $S'_i; B_i \longrightarrow P_i$ in $S$ do
3:     $c_i$ = Find minimum $e$ s.t. /* Compute the upper bound of $\epsilon_i$ (i.e., $c_i$) for a specific alternative
       statement */
           $\begin{cases} \mathrm{d}(v_i, b_i) = e, \text{where} \\ v_i \in RS(v_0, S'_i), \\ b_i \in \text{boundary}(B_i, v_i); \end{cases}$
4:     $c = \min(c, c_i)$; /* Update the upper bound of $\epsilon$ */
5: end for
6: return $\epsilon \le c$; /* Return the scope of $\epsilon$ */
```

Computing ϵ. The inputs are the process S and the initial state v_0 in Alg. 2. The local variable c with initial value $+\infty$ is introduced to record the upper bound of ϵ (line 1). For every alternative statement $B_i \longrightarrow P_i$ with S'_i being the subprocess before it (line 2), we first compute c_i to be the minimal distance between v_i, a reachable state after executing S'_i from v_0, and b_i, a state on the boundary of B_i under the state v_i (line 3), and then update c with the smaller one between current c and c_i (line 4). Here the computation of c_i can be reduced to a constrained optimization problem. And the reachable set is returned by the function *RS*() using simulation methods, since it is difficult to compute and even not computable in general when ODEs are involved. Moreover, the function *boundary*() returns the boundary of B_i under a given state by computing the range of all variables contained in B_i. For instance, *boundary*(B, v) returns $x = 1 \& y = 1$ for $B = x > 1 \& y > 1$ for any v. After all alternative constructs are traversed, $\epsilon \le c$, the valid scope, is finally returned (line 6).

Notably, the error introduced by *RS*() during simulation should be small enough such that the reachable state of the previous subprocess will not overlap with the boundary of the Boolean condition, or c_i will be 0 causing the whole algorithm to fail.

Computing δ. As discussed previously, for a fixed ϵ, the distance between $X(\widehat{t}, \widetilde{x}_0)$ and $X(t, \widetilde{x}_0)$ should be at least 2ϵ. Furthermore, by requiring that the continuous evolution keeps monotonic till $\widehat{t}$ is found after B turns false, the existence of $\widehat{t}$ is guaranteed, satisfying the robustly-safe condition in Def. 5.1.

Based on this idea, the procedure of calculating δ when ϵ is given is presented in Alg. 3. The skeleton of Alg. 3 is quite similar to Alg. 2. Firstly, the local variable d, for recording the lower bound of δ, is initialized to 0 (line 1). For each continuous statement with $B_i \not\equiv true$ (line 2), the computation of the lower bound for d_i can be reduced to a constrained optimization problem (line 3). After that, we update d as the maximal value between d_i and the current value of d (line 4). Finally, $\delta \ge d$ is returned (line 6).

The formulas between line 3 and line 4 are used to determine the lower bound of d_i. $X_i(t, v_i)$ stands for the solution of the continuous evolution with respect to initial state v_i. We first compute two time points, t_i and $\widehat{t_i}$, s.t. with respect to the initial state v_i, the value of B_i at t_i becomes *false* for the first time, and the distance between $X_i(\widehat{t_i}, v_i)$ and $X_i(t_i, v_i)$ is greater than 2ϵ, and the solution between these two points is monotonic; then by searching the maximal value of $\|\mathrm{f}_i(X_i(t'_i, v_i))\|$ by ranging t'_i in $[t_i, \widehat{t_i}]$, we can get the minimal precision d_i satisfying the robustly safe property.

Algorithm 3 *Com_δ*(S, v_0, ϵ) /* Computing the scope of δ */

```
Input: Process $S$ and the value parameter $\epsilon$;
1: $d = 0$; /* The lower bound of $\delta$ is initialized to 0 */
   /* Find the maximal lower bound from all continuous statements in $S$ */
2: for all $S'_i; \langle \dot{x}_i = f_i(x_i) \& B_i \rangle$ in $S$ with $B_i \not\equiv true$ do
3:     $d_i$ = Find minimum $e$ s.t. /* Compute the lower bound of $\delta_i$ (i.e., $d_i$) for a specific continuous
       statement */
       $$\begin{cases} \frac{2\epsilon}{\|f_i(X_i(t'_i, v_i))\|} = e, \\ B_i[x_i \mapsto X_i(t_i, v_i)] = false, \\ \|X_i(t_i, v_i) - X_i(\widehat{t_i}, v_i)\| \geq 2\epsilon, \\ \text{for each } t \in [t_i, \widehat{t_i}], f_i(X_i(t, v_i)) > 0 \text{ or } f_i(X_i(t, v_i)) < 0 \\ \dot{X}_i(t, v_i) = f_i(X_i(t, v_i)), \text{ for some } t_i, \widehat{t_i} \in \mathbb{R}, \text{ where} \\ v_i \in RS(v_0, S'_i), \\ t'_i \in [t_i, \widehat{t_i}], \\ t \in \mathbb{R}; \end{cases}$$
4:     $d = \max(d, d_i)$; /* Update the upper bound of $\delta$ */
5: end for
6: return $\delta \geq d$; /* Return the scope of $\delta$ */
```

5.3 Approximate Bisimulation between HCSP and the Disretization

Based on the assumption of robustly safe, the main theorems on the approximate bisimulation between an HCSP and its discretization are presented for both the bounded and unbounded scenarios.

Theorem 5.4 (Bounded time). *Let S be an HCSP and v_0 any initial state at time 0. Assume S is (δ, ϵ)-robustly safe with respect to v_0, and for any ODE $\dot{x} = f(x)$ occurring in S, f is Lipschitz continuous. Let $\varepsilon \in (0, \epsilon)$ be a precision and $T \in \mathbb{R}^+$ a bounded time. When $\delta = 0$, if h makes Theorem 4.2 hold, then $S \cong_{h,\varepsilon} D_{h,\varepsilon}(S)$ holds on $[0, T]$. If $\delta > 0$ and there exists $h \in (\frac{\delta}{2}, \delta)$ such that Theorem 4.2 holds, we can also conclude that $S \cong_{h,\varepsilon} D_{h,\varepsilon}(S)$ on $[0, T]$.*

Before presenting the proof, we have two points to explain for the theorem.

Firstly, h must be sufficiently small to meet the requirements in Theorem 4.2, i.e. $h < h_{max}$ for some h_{max}. When $\delta = 0$, such an h can always be found. But when $\delta > 0$, h should also be in the interval $(\frac{\delta}{2}, \delta)$ derived from the assumption of (δ, ϵ)-robustly safety. If the two intervals $(0, h_{max})$ and $(\frac{\delta}{2}, \delta)$ are disjoint, h does not exist. This happens when at least one ODE in S fails to leave far enough away from the boundary of its domain B in a limited time, just like the ODE $\langle \dot{x} = f_2(x) \& B \rangle$ shown in Fig. 1.

Secondly, we point out emphatically the execution of each ODE occurring in S. If it terminates before time T because of the violation of the domain condition B, its successive processes will be continued to be executed, one by one, until time reaches T; Otherwise, its execution after T along with the executions of its successive processes will be disregarded.

Below we give a proof of Theorem 5.4.

Proof. Let t_1 and t_2 be the time points right before and after P executes in S, and σ_1 and σ_2 be the states at t_1 and t_2. Similarly, t_1^d and t_2^d are for the time points right before and after $D_{h,\varepsilon}(P)$ executes in $D_{h,\varepsilon}(S)$, and σ_1^d and σ_2^d for the states correspondingly. We denote $\mathrm{d}(\sigma_1, \sigma_1^d)$ by ε_1, and $\mathrm{d}(\sigma_2, \sigma_2^d)$ by ε_2.

We will first prove that $\varepsilon_2 \le K\varepsilon_1$ for some constant K, and the time difference (if exists) is no more than h. This is given by structural induction on P.

- Cases P = **skip** and P = **stop**: Obviously $\varepsilon_2 = \varepsilon_1$ since no variable is changed.
- Case $P = \text{wait } d$: Also $\varepsilon_2 = \varepsilon_1$.
- Case $P = (x := e)$: Suppose e is of the functional form $f(x_1, \cdots, x_n)$. After the assignment, only the value of x is changed, that is, $\sigma_2(x) = f(\sigma_1(x_1), \cdots, \sigma_1(x_n))$ and $\sigma_2^d(x) = f(\sigma_1^d(x_1), \cdots, \sigma_1^d(x_n))$. Let $\delta_i = \sigma_1^d(x_i) - \sigma_1(x_i)$, and we have $|\delta_i| \le \varepsilon_1$ for $i = 1, \ldots, n$, since d is the Euclidian distance. Without loss of generality we suppose x is x_1. By the Lagrange Mean Value Theorem, there exists $\theta \in (0, 1)$ such that $\sigma_2^d(x) - \sigma_2(x) = \sum_{i=1}^n \frac{\partial f}{\partial x_i}(\sigma_1(x_i) + \theta\delta_i)\delta_i$. So $|\sigma_2^d(x) - \sigma_2(x)| \le \sum_{i=1}^n |\frac{\partial f}{\partial x_i}(\sigma_1(x_i) + \theta\delta_i)| \cdot |\delta_i| \le nM\varepsilon_1$, where $M = \max\{|\frac{\partial f}{\partial x_i}(\sigma_1(x_i) + \theta\delta_i)| \mid \theta \in (0, 1), \delta_i \in [-\varepsilon_1, \varepsilon_1], i \in \{1, \ldots, n\}\}$. M exists because $\frac{\partial f}{\partial x_i}$ is bounded in the interval $(\sigma_1(x_i) - \varepsilon_1, \sigma_1(x_i) + \varepsilon_1)$ for each i. Therefore $\varepsilon_2 = \sqrt{|\sigma_2^d(x) - \sigma_2(x)|^2 + \sum_{i=2}^n \delta_i^2} \le \sqrt{n^2M^2 + n - 1} \cdot \varepsilon_1$.
- Case $P = ch!e$: Recall that the discretized process is $ch! := 1; ch!x; ch! := 0$. Notice that the auxiliary readiness variable $ch!$ does not introduce errors. If there is a deadlock, the communication will never be enabled. Otherwise, it takes zero or more time before the communication actually occurs, letting time advance by the same amount for both processes. No variable is changed as a consequence of an output, thus $\varepsilon_2 = \varepsilon_1$.
- Case $P = ch?x$: Time analysis is the same as in case $P = ch!x$. The difference is that, when the communication actually occurs, the value of x is changed as in case $P = (x := e)$, so the result is the same, i.e., $\varepsilon_2 \le \sqrt{n^2M^2 + n - 1} \cdot \varepsilon_1$.
- Case $P = Q; Q'$: By the induction hypothesis, the intermediate error satisfies $\varepsilon_m \le K_Q\varepsilon_1$ and $\varepsilon_2 \le K_{Q'}\varepsilon_m$. Therefore $\varepsilon_2 \le K_{Q'}\varepsilon_m \le K_QK_{Q'}\varepsilon_1$.
- Case $P = Q^*$: As $num(Q^*)$ is the upper bound of the number of repetitions of Q (defined in Sec. 2.3), we have $\varepsilon_2 \le K_Q^{num(Q^*)}\varepsilon_1$ based on the previous case.
- Case $P = Q \sqcap Q'$: Let both processes make the same choice, that is, to choose both Q and $D_{h,\varepsilon}(Q)$, or both Q' and $D_{h,\varepsilon}(Q')$. By the induction hypothesis, we have $\varepsilon_2 \le \max\{K_Q, K_{Q'}\}\varepsilon_1$.
- Case $P = B \longrightarrow Q$: Suppose $\varepsilon_1 < \varepsilon$. (This can always be realized by choosing an h small enough based on the hypothesis that $\varepsilon_1 \le Dh$.) Then we have $\varepsilon_1 < \epsilon$ since $\varepsilon < \epsilon$. When $\sigma_1(B)$ is *true*, $\sigma_1^d(B)$ is also *true*, because σ_1 and β_1 are close enough to satisfy the first condition in robustly safety, then Q and $D_{h,\varepsilon}(Q)$ will be executed afterwards, with $\varepsilon_2 \le K_Q\varepsilon_1$. When $\sigma_1(B)$ is *false*, $\sigma_1^d(B)$ is also *false* for the same reason, then both terminate immediately with $\varepsilon_2 = \varepsilon_1$.
- Case $P = \langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}) \& B \rangle$: Recall that the trajectory $X(t, \sigma_1(\mathrm{x}))$ starts with the initial value $\sigma_1(\mathrm{x})$ at t_1. Let the timing sequence be $\{t_i = t_1 + (i-1)h\}_{i \ge 1}$. Suppose B turns *false* for the first time at $t_f \in (t_N, t_{N+1}]$ ($N = +\infty$ when $B \equiv true$). Clearly $N(B, \varepsilon)$ is true at $t_1, t_2, \ldots, t_N$. Besides, the value of $N'(B, \varepsilon)$ at T, i.e. the value of $N(B, \varepsilon)$ at t_{N+1}, is also *true*, from the fact that the discrete value falls in the ε neighborhood of some $t \le t_f$ when B is *true*. (similar to the proof for Theorem 4.2). But $N(B, \varepsilon) = false$ at t_{N+2}, according to the definition of (δ, ϵ)-robustly safe. If $T < t_{N+1}$, it is just the same as the situation in Theorem 4.2. If $T \ge t_{N+1}$, $D_{h,\varepsilon}(P)$ does nothing after t_{N+1}, while P does nothing after t_f. $|t_f - t_{N+1}| < h$, and there exists some K such that $\mathrm{d}(X(t_f, \sigma_1(\mathrm{x})), \mathrm{x}_{N+1}) \le K\varepsilon_1$ holds by Theorem 4.2.

- Case $P = \langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})\&B\rangle \trianglerighteq []_{i\in I}(io_i \longrightarrow Q_i)$: Also the auxiliary variables $io_i, \overline{io_i}$ do not introduce errors. Communication interrupts are handled as Boolean conditions with higher priority. So the result is similar to that for the case $P = \langle \dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})\&B\rangle$. Notably, the time step h should be small enough such that the communication action and the violation of B would fall into different intervals to be distinguished if they don't occur simultaneously.

Now we can infer that ε_g, the global error in the discretization, is bounded by Dh for some constant D. By choosing $h < \frac{\varepsilon}{D}$, the constraint $\varepsilon_g < \varepsilon$ can be satisfied. If $\delta = 0$, that is sufficient. If $\delta > 0$, $\frac{\varepsilon}{D} > \frac{\delta}{2}$ is also needed to ensure h fall into $(\frac{\delta}{2}, \delta)$. □

Theorem 5.5 (Unbounded time). *Let S be an HCSP, which is (δ, ϵ)-robustly safe, $0 < \varepsilon < \epsilon$ be a precision. If for any ODE $\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})$ occurring in S, f is Lipschitz continuous, $\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x})$ is GAS with $\mathrm{f}(\bar{\mathrm{x}}) = 0$ for some $\bar{\mathrm{x}}$, then if there exists h satisfying $h < \delta < 2h$ if $\delta > 0$ s.t. Theorem 4.2 holds with the equilibrium time of each ODE as the upper bound, it follows $S \cong_{h,\varepsilon} D_{h,\varepsilon}(S)$ on $[0, +\infty]$.*

Proof. Assume equilibrium times of ODEs in S are $T_{et}^1, T_{et}^2, \ldots T_{et}^n$ where $n \in \mathbb{N}$ is the number of ODEs. Let $T = T_{et}^1 + T_{et}^2 \ldots + T_{et}^n$. As the distance between S and $D_{h,\varepsilon}(S)$ is never greater than ε after T, we can easily conclude that $S \cong_{h,\varepsilon} D_{h,\varepsilon}(S)$ on $[T, +\infty]$. The proof for $S \cong_{h,\varepsilon} D_{h,\varepsilon}(S)$ on $[0, T]$ is similar to the bounded scenario illustrated in Theorem 5.4. □

We can see easily that the unbounded time case can be reduced to the bounded time case, with the sum of the equilibrium time for all the ODEs occurring in S as the upper bound of the execution of S.

Remark. To ensure the discretization is approximately bisimilar with the original HCSP model, as shown in Theorem 5.4 and Theorem 5.5, several conditions need to hold. First of all, the ODEs occurring in the HCSP model must be local Lipschitz continuous, which is necessary to ensure the existence and uniqueness of the solutions to the ODEs. Second, the HCSP model should be (δ, ϵ)-robustly safe, and moreover, the relation between the parameters δ, ϵ, and the value and time precisions for the discretization must be satisfied. This requirement is especially necessary to guarantee the consistence of the control flows governed by the Boolean conditions for the whole HCSP model and its discretization, under the premise that each atomic statement is approximately bisimilar with its discretization. At last, for the unbounded time case, the ODEs are required to be GAS, which ensures that the ODE can finally reach an equilibrium point within a bounded time. This is also essential for discretizing HCSP models on unbound time.

6 FROM DISCRETIZED HCSP TO SYSTEMC

In this section, we present the generation of SystemC code from the discretized HCSP preserving a $(0, 0)$-approximate bisimulation (bisimulation for short) during the transformation. In order to prove this bisimulation, we first recall the operational semantics of a core subset of SystemC defined in [74, 75] in Sec. 6.1, and then present the function for SystemC code generation and the proof of the bisimulation based on the unified semantic framework in Sec. 6.2. As the discrete rules of HCSP for unbounded time are very close to those for bounded time (the only difference lies in the choice of the bounded execution time for the continuous statement, as seen in Tab. 2 and Tab. 3), we only present the method of code generation for the bounded time case here.

6.1 Operational Semantics of a Subset of SystemC

The operational semantics for a subset of SystemC defined in [74, 75] is based on transition systems. Formally, the transition function $\mathcal{T}$: SC_statements $\longrightarrow$ Transitions maps a SystemC statement to a set of guarded transitions, and may introduce auxiliary variables such as event, channel and function variables. The assignment statement, taken as an example, has the following semantics:

$$\mathcal{T}[\![l : v := exp; l']\!] \equiv \{P.pc = l \longrightarrow P.pc := l', v := exp\} \tag{14}$$

where l and l' are the labels of SystemC statements, $v := exp$ is an assignment statement, P is the context process which contains the assignment, and pc is the program counter of P for recording the execution progress. And for sequential composition $S_0; S_1$, the transitions are defined as the union of S_0's and S_1's corresponding transitions, that is,

$$\mathcal{T}[\![l_0 : S_0; l_1 : S_1; l_2]\!] \equiv \mathcal{T}[\![l_0 : S_0; l_1]\!] \cup \mathcal{T}[\![l_1 : S_1; l_2]\!] \tag{15}$$

Furthermore, the simulation scheduler is also regarded as a process. All in all, a SystemC program is mapped into a transition system and its execution is interpreted as transitions on the transition system.

Although the subset covers most common statements of SystemC, the for-loop statement involved in our work is not considered. The semantics of for-loop can be derived as a composition of assignment statements and a while-loop statement

$$\mathcal{T}[\![l : \textbf{for } S_1 \textbf{ do } S_2; l']\!] \equiv \mathcal{T}[\![l : A_1; l_1]\!] \cup \mathcal{T}[\![l_1 : \textbf{while } b \textbf{ do } S_2'; l']\!] \tag{16}$$

where $S_1 = A_1; b; A_2$ with A_1 and A_2 being assignments, $S_2' \equiv S_2; A_2$, and b is the Boolean expression within the loop condition.

6.2 Generating SystemC Code From Discretized HCSP

When an HSCP process S is translated into the discretized HCSP D such that they are (h, ε)-approximately bisimilar, we will go on to transform D into a piece of SystemC code SC which is semantically equivalent to D. If it succeeds, we can conclude that S and SC are (h, ε)-approximately bisimilar by transitivity. In this section, we will define a function $\mathcal{SC}[\![\cdot]\!]$ returning a piece of SystemC code for each sequential HCSP, and prove its correctness as well.

Code generation. At the outermost level, the whole parallel model D of the form $P_1 \| \ldots \| P_k$ is encapsulated as an *SC_MODULE*, with each sequential P_i as an *SC_THREAD* therein. This does not affect functionality, so we will not discuss it later. What we are concerned about is the function $\mathcal{SC}[\![\cdot]\!]$: HCSP_sequential_processes $\longrightarrow$ SystemC_statements which is recursively defined. Below is the definition for each sequential construct.

The most important is $\mathcal{SC}[\![P; Q]\!] \equiv \mathcal{SC}[\![P]\!]; \mathcal{SC}[\![Q]\!]$, whose role is to link pieces of code.

Tab. 4 lists code-generating rules for other relatively simple ones, leaving complicated ones for later. $x := e$ is mapped to an assignment $x = e$ in the syntax of SystemC. wait d is transformed into a statement wait(d, *SC_TU*), in which *SC_TU* is the time unit chosen from *SC_SEC* (second), *SC_MS* (millisecond), *SC_US* (microsecond), etc. **stop** is mapped to an infinite *while*-loop statement that waits for one and one time unit, meaning that the process is suspended forever. The alternative construct is mapped to an *if* statement. The internal choice construct is implemented as an *if-else* statement, where *oracle* returns a uncertain boolean value to guide the choice. Here *oracle* is a special *SC_METHOD* which can be seen as an external process. The repetition P^* and the bounded repetition P^k are mapped to *for* statements, where $num(P^*)$ returns the upper bound of P's iteration times.

To ensure synchronization during communication, *sc_signals* and *sc_events* are utilized. For each channel *ch*, the reading side has *ch_r*, an *sc_signal* with Boolean type, to represent the readiness to read, and *ch_r_done*, an *sc_event*, to represent the completion of reading. And the writing side has *ch_w* and *ch_w_done* similarly.

For the discretized input construct $ch? := 1; ch?x; ch? := 0$, the SystemC code is shown in the below left. It first initializes the signal *ch_r* to 1, which means getting ready to read (line 2); waits

$\mathcal{SC}[\![x := e]\!]$	$\equiv$	$x = e$
$\mathcal{SC}[\![\text{wait } d]\!]$	$\equiv$	wait(d, SC_TU)
$\mathcal{SC}[\![\textbf{stop}]\!]$	$\equiv$	while ($true$) {wait(1, SC_TU)}
$\mathcal{SC}[\![B \longrightarrow P]\!]$	$\equiv$	if (B) {$\mathcal{SC}[\![P]\!]$}
$\mathcal{SC}[\![P \sqcap Q]\!]$	$\equiv$	if ($oracle$) {$\mathcal{SC}[\![P]\!]$} else {$\mathcal{SC}[\![Q]\!]$}
$\mathcal{SC}[\![P^*]\!]$	$\equiv$	for ($i = 1; i <= num(P^*); i{+}{+}$) {$\mathcal{SC}[\![P]\!]$}
$\mathcal{SC}[\![P^k]\!]$	$\equiv$	for ($i = 1; i <= k; i{+}{+}$) {$\mathcal{SC}[\![P]\!]$}

Table 4. Function for code generation of HCSP.

until the other side gets ready and finishes writing (lines 3-5); gets the latest value from the channel and assigns it to the variable x (line 6); informs that it has finished reading (line 7); and finally resets ch_r to 0 (lines 8). Here $\mathcal{SC}[\![ch? := 1]\!]$, $\mathcal{SC}[\![ch?x]\!]$ and $\mathcal{SC}[\![ch? := 0]\!]$ correspond to the code in lines 2, 3-7 and 8, respectively. The right below is the code for the output $ch! := 1; ch!e; ch! := 0$ generated similarly. Note that the writing side writes when the other side is ready (lines 3-5), and informs its own completion (line 6), and waits for the completion of the other side (lines 7). The two pieces of code are put in pair for a clearer comparison between them.

```
1  // code for input construct
2  ch_r=1;
3  if(!ch_w)
4     wait(ch_w.posedge_event());
5  wait(ch_w_done);
6  x=ch.read();
7  ch_r_done.notify();
8  ch_r=0;
```

```
1  // code for output construct
2  ch_w=1;
3  if(!ch_r)
4     wait(ch_r.posedge_event());
5  ch.write(e);
6  ch_w_done.notify();
7  wait(ch_r_done);
8  ch_w=0;
```

For the discretized communication-based external choice $\fatsemi_{i\in I}(io_i := 1); [\!]_{i\in I} io_i \longrightarrow (\fatsemi_{i\in I}(io_i := 0); Q_i)$, we first explain the meanings of variables and lists used therein. $chan\_num$ represents the size of the number of channels in I, which is assumed to be finite for simplicity. Indexes of channels range from 0 to $chan\_num - 1$. The Boolean variable IO[i] stands for the readiness of this side of the i-th channel, and IO_d[i] for that of the other side. And k records the index of the channel whose both sides are ready. Below is the explanation of our code. First, k is initialized as -1 and the value of $chan\_num$ is calculated (lines 2-3). Second, all the IO[i] are set to 1, which means all getting ready on this side (lines 4-6). Third, wait until the other side of some channel also gets ready (line 7). Fourth, find the channel (lines 8-9), execute its communication (line 10) and record its index in k (line 11). Finally, reset all the Boolean variables in IO to 0 (lines 14-16), and execute $Q[k]$ (line 17). Here $\mathcal{SC}[\![\fatsemi_{i\in I}(io_i := 1)]\!]$, $\mathcal{SC}[\![[\!]_{i\in I} io_i]\!]$, $\mathcal{SC}[\![\fatsemi_{i\in I}(io_i := 0)]\!]$ and $\mathcal{SC}[\![Q_i]\!]$ correspond to lines 4-6, 7-13, 14-16 and 17, respectively.

```
1  // code for external choice construct
2  int k=-1;
3  int chan_num=sizeof(I)/sizeof(I[0]);
4  for(int i=0;i<chan_num;i++){
        IO[i]=1;
    }
    wait(IO_d[0].posedge_event()|...|IO_d[chan_num-1].posedge_event());
        for(int i=0;i<chan_num&&k<0;i++){
            if(IO[i]==1&&IO_d[i]==1){
                SC[[io_i]];
                k=i;
            }
        }
    for(int i=0;i<chan_num;i++){
        IO[i]=0;
    }
    SC[[Q[k]]];
```

Until now, we have defined the code generation function for all basic constructs of the discretized HCSP. The remaining two constructs can be expressed as compositions of these basic contructs, thus the code generation for them can be defined recursively.

Code for discretized continuous contruct is given below. It contains three parts. The first part in lines 2-7 corresponds to $(N(B,\varepsilon) \wedge N'(B,\varepsilon) \longrightarrow (\text{wait } h; \text{x} := \text{x} + h\Phi(\text{x},h)))^{\lfloor \frac{T}{h} \rfloor}$. where $N(B,e)$, $N\_p(B,e)$ and $R\_K(\text{x},h)$ are helper functions computing $N(B,\varepsilon)$, $N'(B,\varepsilon)$ and $\Phi(\text{x},h)$, respectively (e is used instead of ε for simplicity). The second part in lines 8-11 corresponds to $N(B,\varepsilon) \wedge N'(B,\varepsilon) \longrightarrow (\text{wait } h'; \text{x} := \text{x} + h'\Phi(\text{x},h'))$, where $T - \text{floor}(T/h) * h$ is the value of h'. And there is no more explaining about the third part in lines 12-16.

```
1   // code for continuous construct
2   for(int i=0;i<floor(T/h);i++){
3       if(N(B,e)&&N_p(B,e)){
4           wait(h,SC_TU);
5           x=x+h*R_K(x,h);
6       }
7   }
8   if(N(B,e)&&N_p(B,e)){
9       wait(T-floor(T/h)*h,SC_TU);
10      x=x+(T-floor(T/h)*h)*R_K(x,(T-floor(T/h)*h));
11  }
12  if(N(B,e)&&N_p(B,e)){
13      while(true){
14          wait(1,SC_TU);
15      }
16  }
```

For the discretized communication interrupt construct, code can be obtained by combining that of the discretized continuous construct and the communication choice construct. $⨟_{i\in I}(io_i := 1)$ is implemented as before (lines 2-6). The discretized ODE is also represented as a series of assignments and waits as before, except that the Boolean condition is augmented by the readiness variables for the channels (lines 7-16). If the neighborhood condition of B is violated, all the readiness variables are reset to 0 (lines 17-21). Then if some channel is ready when the neighborhood of B remains *true*, it communicates and records its index k (lines 22-27), resets readiness variables to

0 (lines 28-30) and executes $Q[k]$ (lines 31-33). At last, if the neighborhood of B is never violated and no channels get ready during $[0, T]$, the whole process is suspended forever (lines 34-38).

```
1  // code for communication interrupt construct
2  int k=-1;
3  int chan_num=sizeof(I)/sizeof(I[0]);
4  for(int i=0;i<chan_num;i++){
5      IO[i]=1;
6  }
7  for(int i=0;i<floor(T/h);i++){
8      if(N(B,e)&&N_p(B,e)&&IO[0]&&!IO_d[0]&&...){
9          wait(h,SC_TU);
10         x=x+h*R_K(x,h);
11     }
12 }
13 if(N(B,e)&&N_p(B,e)&&IO[0]&&!IO_d[0]&&...){
14     wait(T-floor(T/h)*h,SC_TU);
15     x=x+(T-floor(T/h)*h)*R_K(x,(T-floor(T/h)*h))
16 }
17 if(!(N(B,e)&&N_p(B,e))&&IO[0]&&!IO_d[0]&&...){
18     for(int i=0;i<chan_num;i++){
19         IO[i]=0;
20     }
21 }
22 for(int i=0;i<chan_num&&k<0;i++){
23     if(IO[i]==1&&IO_d[i]==1){
24         SC[[io_i]];
25         k=i;
26     }
27 }
28 for(int i=0;i<chan_num;i++){
29     IO[i]=0;
30 }
31 if(k>-1){
32     SC[[Q[k]]];
33 }
34 if(N(B,e)&&N_p(B,e)&&IO[0]&&!IO_d[0]&&...){
35     while(true){
36         wait(1,SC_TU);
37     }
38 }
```

Correctness of the code generation. Based on the above definition of function $\mathcal{SC}[\![.]\!]$, a piece of SystemC code $\mathcal{SC}[\![D_{h,\varepsilon}(S)]\!]$ can be automatically generated from a given discretized HCSP $D_{h,\varepsilon}(S)$. Furthermore, we can guarantee the "equivalence" between them by proving that they are bisimilar, i.e., with $h = \varepsilon = 0$ in Def. 3.1 and Def. 3.2.

Theorem 6.1. *For the discretization of HCSP model S, $D_{h,\varepsilon}(S)$ and $\mathcal{SC}[\![D_{h,\varepsilon}(S)]\!]$ are bisimilar, i.e., $(0, 0)$-approximately bisimilar.*

Proof. We first focus on transition systems, denoted as TD and TSC, generated from $D_{h,\varepsilon}(S)$ and $\mathcal{SC}[\![D_{h,\varepsilon}(S)]\!]$, respectively. Let $Var(P)$ be the set of variables occurring in P, and $In(P)$ and

$Out(P)$ be the sets of channels that P can read and write, respectively. The states in TD and TSC are evaluations over $Var(D_{h,\varepsilon}(S)) \cup In(D_{h,\varepsilon}(S)) \cup Out(D_{h,\varepsilon}(S))$. Actions are constructs of $D_{h,\varepsilon}(S)$ and pieces of $\mathcal{SC}[\![D_{h,\varepsilon}(S)]\!]$, respectively. Essentially, they are $\{\tau\} \cup \{\text{assign}(x, e)\} \cup \{d \geq 0\}$ according to their real effects. With a slight abuse of definition, the sequence of transitions based on P is abbreviated as $v \xrightarrow{P} v'$, and transitions based on $P_1; P_2 : \ldots; P_k$ is abbreviated as $v^0 \xrightarrow{P_1} v^1 \xrightarrow{P_2} \cdots \xrightarrow{P_k} v^k$ to emphasize that the end of the i-th transition is the beginning of the $i+1$-th. $v(e)$ means the value of the expression e based on the evaluation v, and $v' = v[x \mapsto y]$ means the evaluation v' is the same as v except $v'(x) = y$.

From the definition of HCSP, each P can only change the values of variables in $Var(P) \cup Out(P)$. Hence the set is mentioned as the local variables of P in the following of this proof. Two states v and v' are called locally equivalent with respect to P, denoted as $v =_P v'$, if for any $x \in Var(P) \cup Out(P)$, $v(x) = v'(x)$.

The transitions are organized by structural induction on $D_{h,\varepsilon}(S)$'s possible components. In the following, we use P to represent each component, and p_i and q_i to distinguish states of TD and TSC, respectively.

- Case $P = P_1; P_2$: Transitions are $\{p_0 \xrightarrow{P_1} p_1 \xrightarrow{P_2} p_2\}$ and $\{q_0 \xrightarrow{\mathcal{SC}[\![P_1]\!]} q_1 \xrightarrow{\mathcal{SC}[\![P_2]\!]} q_2\}$.
- Cases $P = Q^k$ and $P = Q^*$: Transitions are $\{p_0 \xrightarrow{Q} p_1 \xrightarrow{Q} \cdots \xrightarrow{Q} p_k\}$ and $\{q_0 \xrightarrow{\mathcal{SC}[\![Q]\!]} q_1 \xrightarrow{\mathcal{SC}[\![Q]\!]} \cdots \xrightarrow{\mathcal{SC}[\![Q]\!]} q_k\}$. For $P = Q^*$, k takes the value $num(Q^*)$.
- Case $P = \mathbf{skip}$: Transitions are $\{p \xrightarrow{\tau} p\}$ and $\{q \xrightarrow{\tau} q\}$.
- Case $P = \mathbf{stop}$: Transitions are $\{p \xrightarrow{d} p\}$ and $\{q \xrightarrow{d} q\}$. Here d is large enough number.
- Case $P = wait\ d$: Transitions are $\{p \xrightarrow{d} p\}$ and $\{q \xrightarrow{d} q\}$.
- Case $P = x := e$: Transitions are $\{p \xrightarrow{x:=e} p[x \mapsto p(e)]\}$ and $\{q \xrightarrow{x=e} q[x \mapsto q(e)]\}$.
- Case $P = ⨟_{i \in I}(io_i := b)$ ($b = 0$ or 1): Suppose $I = \{1, \ldots, |I|\}$. As a result, transitions are $\{p_0 \xrightarrow{io_1:=b} p_1 \xrightarrow{io_2:=b} \cdots \xrightarrow{io_{|I|}:=b} p_{|I|}\}$ and $\{q_0 \xrightarrow{IO[1]:=b} q_1 \xrightarrow{IO[2]=b} \cdots \xrightarrow{IO[|I|]=b} q_{|I|}\}$ where $p_i = p_{i-1}[io_i \mapsto b]$ and $q_i = q_{i-1}[IO[i] \mapsto b]$ for $i \in I$. Here IO[i] are alias of io_i.
- Case $P = B \longrightarrow Q$: Transitions are $\{p \xrightarrow{Q} p' \mid p(B) = 1\} \cup \{p \xrightarrow{\tau} p \mid p(B) = 0\}$ and $\{q \xrightarrow{\mathcal{SC}[\![Q]\!]} q' \mid q(B) = 1\} \cup \{q \xrightarrow{\tau} q \mid q(B) = 0\}$.
- Case $P = [\!]_{i \in I}(io_i \longrightarrow Q_i)$: Transitions are $\bigcup_{i \in I}\{p_0 \xrightarrow{d_i} p_{0i} \xrightarrow{io_i} p_{1i} \xrightarrow{Q_i} p_{2i}\}$ and $\bigcup_{i \in I}\{q_0 \xrightarrow{d_i} q_{0i} \xrightarrow{\mathcal{SC}[\![io_i]\!]} q_{1i} \xrightarrow{\mathcal{SC}[\![Q_i]\!]} q_{2i}\}$. This means that, if the i-th communication is the first to be ready after waiting d_i (d_i can also be 0), the communication io_i and the sequential Q_i will be scheduled. Here $p_{0i} =_P p_0$, and $p_{0i}(\overline{io_i}) = 1$ and $p_{0i}(\overline{io_j}) = 0$ for all $j \neq i$. Similarly, $q_{0i} =_{\mathcal{SC}[\![P]\!]} q_0$, $q_{0i}(\text{IO\_d}[i]) = 1$ and $q_{0i}(\text{IO\_d}[j]) = 0$ for all $j \neq i$. IO_d[i] have the same meaning as $\overline{io_i}$.
- Case $P = ch?x$: Transitions are $\{p_0 \xrightarrow{d} p_1 \xrightarrow{x:=p_1(ch)} p_2\}$ and $\{q_0 \xrightarrow{d} q_1 \xrightarrow{x=q_1(ch)} q_2\}$. d is the time difference between the readiness of reading and the completeness of channel updating. Notably, if the writing side gets ready first, d must be 0. Here p_1 is locally equivalent to p_0 with respect to P, that is, $p_1 =_P p_0$, and satisfies $p_1(ch!) = 1$. And $p_2 = p_1[x \mapsto p_1(ch)]$. Analyses for q_1 and q_2 are similar.
- Case $P = ch!e$: Transitions are $\{p_0 \xrightarrow{d} p_1 \xrightarrow{ch:=p_1(e)} p_2 \xrightarrow{0} p_3\}$ and $\{q_0 \xrightarrow{d} q_1 \xrightarrow{ch=q_1(e)} q_2 \xrightarrow{0} q_3\}$. d is the time difference between the readiness of writing and the readiness of the

reading. The transitions $p_2 \xrightarrow{0} p_3$ and $q_2 \xrightarrow{0} q_3$ are used for ensuring that the latest value of the channel is updated. Other analyses for p_1, p_2, q_1 and q_2 are similar to the previous case.

- Case $P = P_1 \| P_2$:
  - When there are no communications, P_1 and P_2 will execute independently, resulting in possible alternate executions of P_1's and P_2's transitions by nondeterministic choices. If P_1 and P_2 are atomic (i.e., there is only one step of transition), transitions are $\{p_{00} \xrightarrow{P_1} p_{10} \xrightarrow{P_2} p_{11}\} \cup \{p_{00} \xrightarrow{P_2} p_{01} \xrightarrow{P_1} p_{11}\}$ and $\{q_{00} \xrightarrow{\mathcal{SC}[\![P_1]\!]} q_{01} \xrightarrow{\mathcal{SC}[\![P_2]\!]} q_{11}\} \cup \{q_{00} \xrightarrow{\mathcal{SC}[\![P_2]\!]} q_{10} \xrightarrow{\mathcal{SC}[\![P_1]\!]} q_{11}\}$. If P_1 and P_2 are not atomic, the result is similar, by introducing more intermediate states.
  - When there are communications in between, it will be a little different. For example, the case where $ch! := 1; ch!e; ch! := 0$ in P_1 and $ch? := 1; ch?x; ch? := 0$ in P_2 match. There are four kinds of transitions based on P: $\{p_{00} \xrightarrow{ch!:=1} p_{10} \xrightarrow{d_1} p'_{10} \xrightarrow{ch?:=1} p_{11} \xrightarrow{ch:=p_{11}(e)} p_{21} \xrightarrow{x:=p_{21}(ch)} p_{22} \xrightarrow{ch!:=0} p_{32} \xrightarrow{ch?:=0} p_{33}\} \cup \{p_{00} \xrightarrow{ch!:=1} p_{10} \xrightarrow{d_1} p'_{10} \xrightarrow{ch?:=1} p_{11} \xrightarrow{ch:=p_{11}(e)} p_{21} \xrightarrow{x:=p_{21}(ch)} p_{22} \xrightarrow{ch?:=0} p_{23} \xrightarrow{ch!:=0} p_{33}\} \cup \{p_{00} \xrightarrow{ch?:=1} p_{01} \xrightarrow{d_2} p'_{01} \xrightarrow{ch!:=1} p_{11} \xrightarrow{ch:=p_{11}(e)} p_{21} \xrightarrow{x:=p_{21}(ch)} p_{22} \xrightarrow{ch!:=0} p_{32} \xrightarrow{ch?:=0} p_{33}\} \cup \{p_{00} \xrightarrow{ch?:=1} p_{01} \xrightarrow{d_2} p'_{01} \xrightarrow{ch!:=1} p_{11} \xrightarrow{ch:=p_{11}(e)} p_{21} \xrightarrow{x:=p_{21}(ch)} p_{22} \xrightarrow{ch?:=0} p_{23} \xrightarrow{ch!:=0} p_{33}\}$. Similarly, transitions based on $\mathcal{SC}[\![P]\!]$ are $\{q_{00} \xrightarrow{ch\_w=1} q_{10} \xrightarrow{d_1} q'_{10} \xrightarrow{ch\_r=1} q_{11} \xrightarrow{ch=q_{11}(e)} q_{21} \xrightarrow{x=q_{21}(ch)} q_{22} \xrightarrow{ch\_w=0} q_{32} \xrightarrow{ch\_r=0} q_{33}\} \cup \{q_{00} \xrightarrow{ch\_w=1} q_{10} \xrightarrow{d_1} q'_{10} \xrightarrow{ch\_r=1} q_{11} \xrightarrow{ch=q_{11}(e)} q_{21} \xrightarrow{x=q_{21}(ch)} q_{22} \xrightarrow{ch\_r=0} q_{23} \xrightarrow{ch\_w=0} q_{33}\} \cup \{q_{00} \xrightarrow{ch\_r=1} q_{01} \xrightarrow{d_2} q'_{01} \xrightarrow{ch\_w=1} q_{11} \xrightarrow{ch=q_{11}(e)} q_{21} \xrightarrow{x=q_{21}(ch)} q_{22} \xrightarrow{ch\_w=0} q_{32} \xrightarrow{ch\_r=0} q_{33}\} \cup \{q_{00} \xrightarrow{ch\_r=1} q_{01} \xrightarrow{d_2} q'_{01} \xrightarrow{ch\_w=1} q_{11} \xrightarrow{ch=q_{11}(e)} q_{21} \xrightarrow{x=q_{21}(ch)} q_{22} \xrightarrow{ch\_r=0} q_{23} \xrightarrow{ch\_w=0} q_{33}\}$. Here $p'_{10} =_{P_1} p_{10}$ with $p'_{10}(ch?) = 0$, $p'_{01} =_{P_2} p_{01}$ with $p'_{01}(ch!) = 0$, $q'_{10} =_{\mathcal{SC}[\![P_1]\!]} q_{10}$ with $q'_{10}(ch\_r) = 0$, and $q'_{01} =_{\mathcal{SC}[\![P_2]\!]} q_{01}$ with $q'_{01}(ch\_w) = 0$. And relations of other states are the same as in the previous cases.

Other constructs can be proved by combining the above-mentioned cases.

Let $\mathcal{B}$ be the identical relation over the evaluations. From the above analyses, we can easily prove that $\mathcal{B}$ is a $(0, 0)$-approximately bisimilar relation, that is, $D_{h,\varepsilon}(S)$ is bisimilar to $\mathcal{SC}[\![D_{h,\varepsilon}(S)]\!]$. □

7 CASE STUDIES

In this section, we illustrate the code generation process from HCSP to SystemC through three case studies: the water tank system introduced in Sec. 2.3, a room heating system, and the control of the slow descent phase of the China Chang'e lunar lander. The whole procedure is divided into four steps: (1) build the HCSP model of the system manually; (2) estimate the values of the critical robustly safe parameters δ and ϵ for discretization through Matlab scripts; (3) choose the value precision $\varepsilon < \epsilon$ and the bounded time $T \in \mathbb{R}^+$ (or $T = +\infty$ for unbounded time), and then infer the value of h; (4) generate SystemC code from the HCSP model based on ε and h obtained in step 3. This is implemented by a Java program, whose input is the HCSP model and computed precisions, and the output is a piece of SystemC code[1]. For the water tank system, both the bounded and unbounded time cases are discussed (since the GAS condition is satisfied). For the other two systems, only the bounded time case is considered.

[1]The tool and all example files, including the Matlab scripts for computing the discrete parameters and precisions, can be found at https://github.com/HCSP-CodeGeneration/HCSP2SystemC.

7.1 Water tank system

The water tank system and its corresponding HCSP model have been introduced in Sec. 2.3, and its discretization is in Sec. 4.2. So we directly present the results in bounded and unbounded time scenarios, respectively. Parameters are set to $Q_{max} = 2.0$, $\pi = 3.14$, $r = 0.18$, $g = 9.8$, $p = 1$, $lb = 4.1$, $ub = 5.9$, $v_0 = 1$ and $d_0 = 4.5$ here.

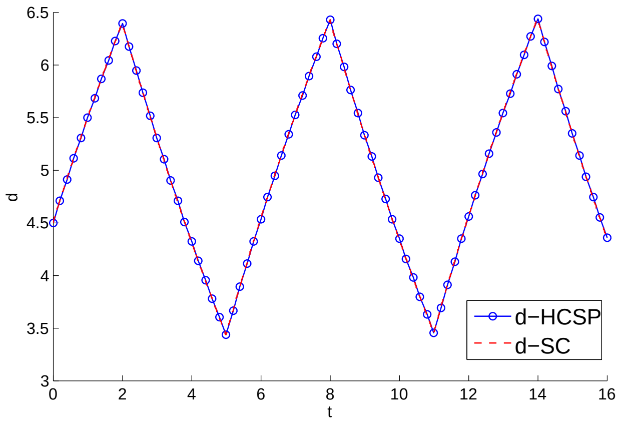


Fig. 2. The HCSP model *vs.* the SystemC code of *WTS* for the waterlevel (Color figure online).

Bounded time. Using the algorithms in Section 5.2, we get $\delta = 0$ and $\epsilon \leq 0.22$. Therefore, we set $\varepsilon = 0.2$ and $T = 16$, and figure out $h = 0.008$ such that Theorem 5.4 holds, i.e., $WTS \cong_{h,\varepsilon} D_{h,\varepsilon}(WTS)$ on $[0, 16]$. Then the SystemC code bisimilar to $D_{h,\varepsilon}(WTS)$ is automatically generated.

Fig. 2 shows the fluctuation of the water level d acquired from the simulation of the original HCSP model (*d-HCSP* represented by blue solid with cycles) and the generated SystemC model (*d-SC* represented by red dashed), respectively. It is apparent that they practically coincide with each other.

Unbounded time. In fact, the method above can be applied directly, by computing the equilibrium time for each ODE and taking the sum of all of them as the time bound.

Take as an example the *WTS* in [73] whose discretization has already been presented with the GAS condition. We first compute the time of reaching the stable state for the two ODEs $\dot{d} = Q_{max} - \pi r^2\sqrt{2gd}$ and $\dot{d} = -\pi r^2\sqrt{2gd}$ (whose equilibrium points are $\frac{Q_{max}^2}{2\pi^2 r^4 g}$ and 0, respectively), which are 82.5s and 9.5s, respectively. Second, we use our tool to generate the SystemC code from *WTS* within $82.5 + 9.5 = 92$s. Thus the SystemC code without the limitation on the total execution time is generated. Setting the values of parameters as in the bounded scenario, the result also shows the consistency between the original HCSP and its generated SystemC code (details are omitted here, as it is very close to the result in Fig. 2).

7.2 Room Heating

The second case study is inspired by the room heating benchmark in [33]. In this example, there are an n-room house and m heaters, where $m < n$ in general. In order to maintain proper room temperature, heaters are shared, that is, a heater in one room can move to its adjacent room. But each room is allowed to hold only one heater at the same time. The temperature of each room (denoted as x_i, $i = 1, \ldots, n$) is decided by those of its adjacent rooms and the outside (represented as u), and whether there is a heater working inside (defined as $h_i = 1$ when there is, and $h_i = 0$

otherwise). The state of the heater in room i is controlled by the room temperature x_i: the heater switches on when the temperature is below a lower threshold (i.e., $x_i \leq \mathrm{on_i}$), and off when it is beyond a upper threshold (i.e., $x_i \geq \mathrm{off_i}$). In addition, the shift of the heater from room j to room i happens when the following four conditions hold: (1) room i has no heater; (2) room j has one; (3) the temperature of room i is below a certain level (i.e., $x_i \leq \mathrm{get_i}$); and (4) the temperature difference between room j and room i is not less than another threshold (i.e., $x_j - x_i \geq \mathrm{diff_i}$).

Here, we consider an instance of the room heating system where $n = 2$ and $m = 1$, that is, there are two rooms and one heater. Its continuous behavior, denoted as $\mathrm{ODE_{tem}}$, is governed by

$$\begin{cases} \dot{x_1} &= -0.45x_1 + 0.25x_2 + 0.2u + 5h_1 \\ \dot{x_2} &= 0.25x_1 - 0.65x_2 + 0.15u + 5h_2 \end{cases} \tag{17}$$

where $u = 4$ is the outside temperature. Assume the system has an initial state $x_1 = x_2 = 10$, $h_1 = 1$ and $h_2 = 0$. The thresholds are $\mathrm{on_1} = \mathrm{on_2} = 10$, $\mathrm{off_1} = \mathrm{off_2} = 12$, $\mathrm{get_1} = \mathrm{get_2} = 8$, and $\mathrm{diff_1} = \mathrm{diff_2} = 0.5$. We get the following HCSP model of the room heating system:

$$\begin{array}{rcl}
RH & \overset{\mathrm{def}}{=} & Room \| Controller \\
Room & \overset{\mathrm{def}}{=} & x_1 := 10; x_2 := 10; \mathrm{R\_h_1} := 1; \mathrm{R\_h_2} := 0; \\
 & & (\langle \mathrm{ODE_{tem}} \rangle \trianglerighteq (\mathrm{ch\_x_1}!x_1 \rightarrow (\mathrm{ch\_x_2}!x_2; \mathrm{ch\_h_1}?\mathrm{R\_h_1}; \mathrm{ch\_h_2}?\mathrm{R\_h_2})))^* \\
Controller & \overset{\mathrm{def}}{=} & h_1 := 1; h_2 := 0; \mathrm{C\_x_1} := 10; \mathrm{C\_x_2} := 10; \mathrm{owner} := 1; \\
 & & (\mathrm{wait}\ 0.1; \mathrm{ch\_x_1}?\mathrm{C\_x_1}; \mathrm{ch\_x_2}?\mathrm{C\_x_2}; \\
 & & (\mathrm{owner} = 1 \& \mathrm{C\_x_2} \leq get_2 \& \mathrm{C\_x_1} - \mathrm{C\_x_2} \geq \mathrm{diff_2}) \rightarrow (\mathrm{owner} := 2; h_1 := 0; h_2 := 1); \\
 & & (\mathrm{owner} = 2 \& \mathrm{C\_x_1} \leq get_1 \& \mathrm{C\_x_2} - \mathrm{C\_x_1} \geq \mathrm{diff_1}) \rightarrow (\mathrm{owner} := 1; h_1 := 1; h_2 := 0); \\
 & & (\mathrm{owner} = 2 \& \mathrm{C\_x_2} \leq \mathrm{on_2}) \rightarrow (h_1 := 0; h_2 := 1); \\
 & & (\mathrm{owner} = 1 \& \mathrm{C\_x_1} \leq \mathrm{on_1}) \rightarrow (h_1 := 1; h_2 := 0); \\
 & & \mathrm{C\_x_2} \geq \mathrm{off_2} \rightarrow h_2 := 0; \mathrm{C\_x_1} \geq \mathrm{off_1} \rightarrow h_1 := 0; \mathrm{ch\_h_1}!h_1; \mathrm{ch\_h_2}!h_2)^*
\end{array}$$

The room heating system *RH* is modeled as the parallel composition of two processes *Room* and *Controller*. $\mathrm{ch\_x_1}$, $\mathrm{ch\_x_2}$, $\mathrm{ch\_h_1}$ and $\mathrm{ch\_h_2}$ are channels between *Room* and *Controller*, and $\mathrm{C\_x_1}$, $\mathrm{C\_x_2}$, $\mathrm{R\_h_1}$ and $\mathrm{R\_h_2}$ are used to restore values received from corresponding channels. Furthermore, the variable owner in *Controller* represents the room holding the heater, i.e., $\mathrm{owner} = i$ means that room i has the heater for $i = 1, 2$.

For discretization of the above model, we estimate that $\delta = 0$ and $\epsilon \leq 0.12$, and choose $\varepsilon = 0.1$. By letting $T = 5$, we infer $h = 0.0002$ such that $RH \cong_{h,\varepsilon} D_{h,\varepsilon}(RH)$ on $[0, 5]$ holds. At last, the SystemC code bisimilar to $D_{h,\varepsilon}(RH)$ is generated.

Like in the first case, the execution results of the HCSP model and the generated SystemC model are shown in Fig. 3. The temperatures in room 1 and 2 are represented as $x1$ and $x2$ with suffixes -HCSP and -SC, respectively. Obviously the SystemC model is indeed an approximation of the original HCSP model.

7.3 Slow Descent Phase of the Lunar Lander

In [27, 79], the formal verification of the slow descent phase of a lunar lander is presented. We mainly focus on its discretization and code generation here.

The slow descent phase is the last of the six phases in the powered descent process of a lunar lander, beginning at a height of 30m relative to the lunar surface. Just as the name indicates, its aim is to ensure that the lander shall descend slowly and smoothly to the lunar surface. During this phase, the controller periodically observes the state of the lander including the velocity and mass,

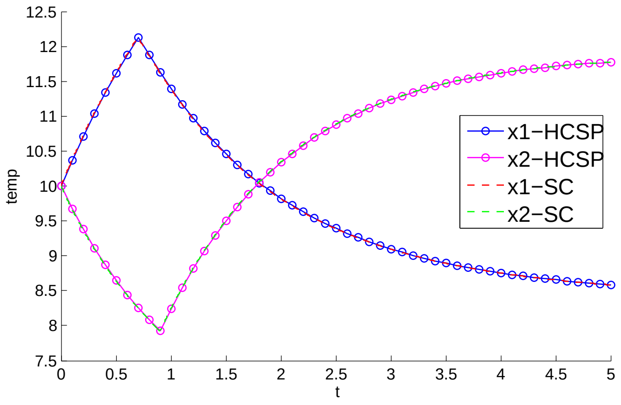


Fig. 3. The HCSP model *vs.* the SystemC code of *RH* for the temperatures (Colour figure online).

and computes a new control command, and then sends it back for the physical plant to follow in the next period, until an engine shutdown signal is received.

The dynamics of the physical plant is described by

$$\begin{cases} \dot{r} & = & v \\ \dot{v} & = & \frac{F_c}{m} - gM \\ \dot{m} & = & -\frac{F_c}{\mathrm{Isp}} \\ \dot{F_c} & = & 0 \end{cases} \tag{18}$$

and the controller reads the values of m and v periodically to calculate F_c' for the next sampling cycle:

$$F_c' = -0.01 \cdot m \cdot ((\frac{F_c}{m} - gM) - 0.6 \cdot (v+2) + gM) \tag{19}$$

whose righthand side is written as $\mathrm{Com\_Next\_F_c}$ for short. Here r is the lander's altitude relative to the lunar surface, v is its vertical velocity, m is its mass, and F_c is the thrust imposed on the lander. $gM = 1.622\mathrm{m/s^2}$ is the gravitational acceleration on the moon. And Isp denotes the *specific impulse*[2] of the lander's thrust engine, which is set to $\mathrm{Isp_1} = 2548\mathrm{N} \cdot \mathrm{s/kg}$ if $F_c \leq 3000\mathrm{N}$, and $\mathrm{Isp_2} = 2842\mathrm{N} \cdot \mathrm{s/kg}$ otherwise. Therefore there are actually two ODEs, denoted as $\mathrm{ODE_1}$ and $\mathrm{ODE_2}$ for convenience, corresponding to $\mathrm{Isp_1}$ and $\mathrm{Isp_2}$, respectively.

By assigning the initial values of m, r, v and F_c to 1250kg, 30m, −2m/s and 2027.5N, respectively, and the sampling period to 0.128s, we have the following HCSP system *Slow_ph*:

$$\begin{array}{rcl} \mathit{Slow\_ph} & \stackrel{\mathrm{def}}{=} & \mathit{Lander}\|\mathit{Controller} \\ \mathit{Lander} & \stackrel{\mathrm{def}}{=} & m := 1250; r := 30; v := -2; \mathrm{L\_F_c} := 2027.5; \\ & & (\mathrm{L\_F_c} \leq 3000 \rightarrow \langle \mathrm{ODE_1} \rangle \trianglerighteq (\mathit{ch\_m}!m \rightarrow (\mathit{ch\_v}!v; \mathit{ch\_F_c}?\mathrm{L\_F_c})); \\ & & \mathrm{L\_F_c} > 3000 \rightarrow \langle \mathrm{ODE_2} \rangle \trianglerighteq (\mathit{ch\_m}!m \rightarrow (\mathit{ch\_v}!v; \mathit{ch\_F_c}?\mathrm{L\_F_c})))^* \\ \mathit{Controller} & \stackrel{\mathrm{def}}{=} & \mathit{C\_m} := 1250; \mathit{C\_v} := -2; F_c := 2027.5; \\ & & (\mathrm{wait}\ 0.128; \mathit{ch\_m}?\mathit{C\_m}; \mathit{ch\_v}?\mathit{C\_v}; F_c := \mathrm{Com\_Next\_F_c}; \mathit{ch\_F_c}!F_c)^* \end{array}$$

[2]Specific impulse is a physical quantity describing the efficiency of rocket engines. It equals the thrust produced per unit mass of propellant burned per second.

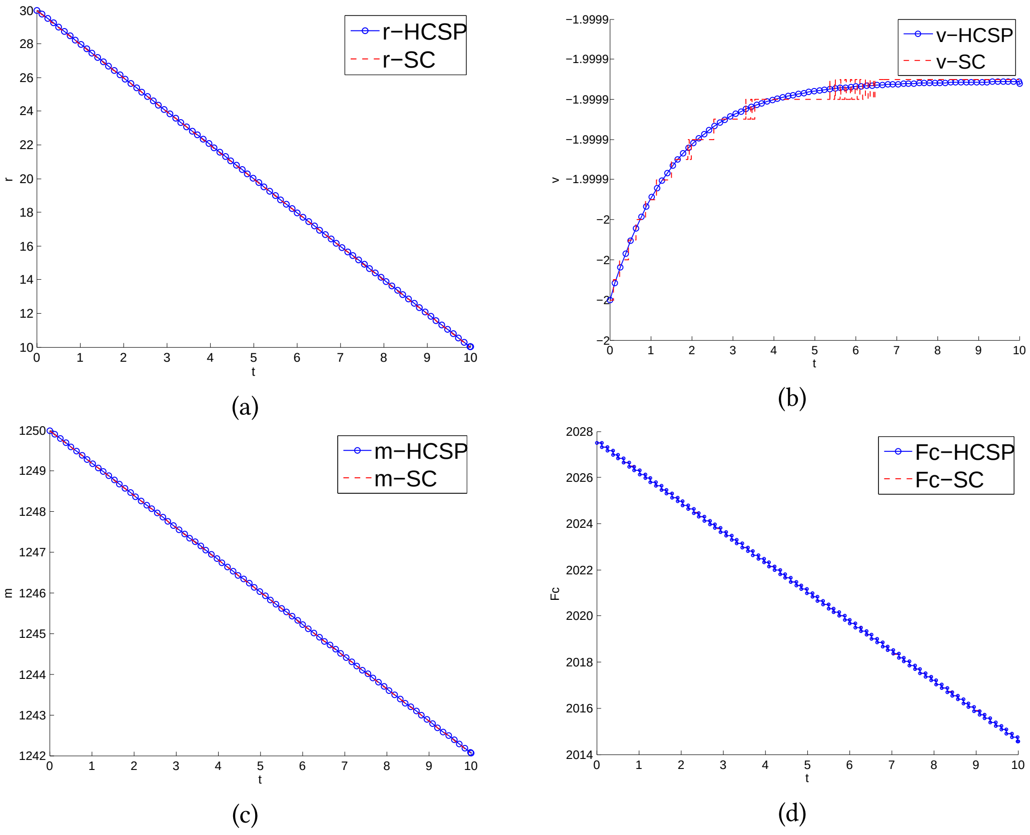


Fig. 4. The HCSP model *vs.* the SystemC code of *Slow_ph*: (a)-(d) represent the evolution of altitude *r*, velocity *v*, mass *m* and thrust F_c, of the lunar lander, respectively (Color figure online).

where ch_m, ch_v and ch_F_c are channels between *Lander* and *Controller*, and C_m, C_v and L_F_c are used to restore values received from corresponding channels.

As with the previous two cases, we can estimate that $\delta = 0$ and $\epsilon \leq 972.5$, respectively. Then, by letting $\varepsilon = 0.05$ and $T = 10$ here, we can infer $h = 0.0002$ satisfies $Slow\_ph \cong_{h,\varepsilon} D_{h,\varepsilon}(Slow\_ph)$ on $[0, 10]$. Finally, we automatically generate the SystemC code that is bisimilar to $D_{h,\varepsilon}(Slow\_ph)$, whose execution (red dashed, with suffix -SC) is given in Fig. 4, compared to the simulation of the original HCSP model (blue solid with cycles, with suffix -HCSP). In particular, all variables in *Slow_ph*, i.e., r, v, m and F_c, are observed. As show in Fig. 4, the difference between the HCSP results and the SystemC results is within the error precision, i.e., $\varepsilon = 0.05$, for each variable.

8 RELATED WORK

Approximate bisimulation. Approximate bisimulation [36] is a popular method for analyzing and verifying complex hybrid systems. Instead of requiring observational behaviors of two systems to be exactly identical, it allows errors but requires the "distance" between two systems to remain bounded by some precisions. In [35], with the use of simulation functions, a characterization of approximate simulation relations between hybrid systems is developed. A new approximate bisimulation relation with two parameters as precisions, which is very similar to the notion defined in this paper, is introduced in [46]. For control systems with inputs, the method for constructing a symbolic model which is approximately bisimilar with the original continuous system is studied in [61]. Moreover, [54] discusses the problem for building an approximately bisimilar symbolic model of a digital control system. Also, there are some works on building symbolic models for networks of control systems [62]. But for all the above works, either discrete dynamics is not considered, or it is assumed atomic actions are independent of the continuous variables. In [41, 48, 69], the abstraction of hybrid automata is considered, but it is only guaranteed that the abstract system is an approximate simulation of the original system. In [60], a discretization of hybrid programs is presented for a proof-theoretical purpose, i.e., it aims to have a sound and complete axiomatization relative to properties of discrete programs. Different from all the above works, we aim to have a discretization of HCSP, for which discrete and continuous dynamics, communications, etc., are entangled with each other tightly, to guarantee that the discretized process has the approximate equivalence with the original process.

Code generation. In the past decade, code generation from formal models has received lots of attention from the research and industrial communities [53], owe to its overwhelming advantages, such as reducing the cost for development, making domain concepts easier to write and introducing less errors [16]. For the traditional discrete systems, there are many relevant tools and works. In [52], a tool that can generate code from a high level specification, called Hybrid Control Operator Language (HCOL), is proposed. However, HCOL mainly focuses on mathematical calculation and is weak in building a whole embedded system. In [25], a tool named EventB2Java is presented for generating Java code from Event-B [7] formal models. Another tool called Asm2C++ is proposed in [18], which can transform the Abstract State Machines (ASM) [19] to C++ code. But both tools do not formally prove the conformance relation between the formal model and the code model. In [58], the authors define a complete set of rules for transforming UML State Machine (USM) to C++ code. Whereas, the semantic-conformance between USM [72] and corresponding C++ code is only tested on some benchmarks (in fact, 4 of 66 tests failed), and no formal correctness is proved. Moreover, [50] explores the possibilities of automatically generate Java code from Z formal models [67]. Unfortunately, the translation is done manually. Authors in [51] illustrate the translation from a formalized modeling language called $\langle HOE\rangle^2$ to an efficient intermediate representation, and [32] introduces a method that can automatically generate a distributed implementation in C from a formal model called LNT [26], and [65] develops a tool for transforming a deterministic finite-state automaton (FSA) [28] into a piece of C# code. Nevertheless, continuous behavior is not taken into account in $\langle HOE\rangle^2$, LNT and FSA. Other popular models for building reactive systems, such as Esterel [17], Statecharts [39], and Lustre [37, 38] also support code generation. However, continuous behavior is still not supported in them. In fact, there are few works about code generation from continuous or hybrid systems, due to the complexity of them.

A hybrid system exhibits the combination of continuous and discrete behaviors, even with concurrence and communication. However, the code inherently appears discrete. Therefore, the semantic differences between the model and the code may bring about different system behavior, which is dangerous for safety-critical systems, as it essentially determines how to sample data from the continuous evolution and the entanglement between sampling data and computing control commands, that essentially determines whether or not the controller at code level is correct. How to provide formal guarantees for the model of the system and its code is a challenging problem. In fact, many MDD approaches targeting at ESs have been proposed and used in industry and academia, many of which support code generation from control models, e.g., commercial modeling tools such as Simulink [3], Rational Rose [2], and TargetLink [6]. However, they mainly focus on the numerical errors for solving the ODEs in the model rather than errors on discrete behavior, and they all lack the formal guarantee between the model and the code generated from the model. SHIFT [29] is another modeling language for hybrid automata, and code generation is also supported in it, but the correctness issue is not addressed rigorously. In [10] and [44], code generation from a special hybrid model, named CHARON [9], was discussed on signal thread and multi-thread execution, respectively. Then, the authors extended their previous work by defining a formal criteria, named *faithful implementation*, in which error bounds on the variables and the time of discrete transitions are allowed [11]. However, the code model in [11] is only an under-approximate implementation of the original hybrid system essentially, and only the coherence of transitions between the hybrid model and its code model is considered. Moreover, there are many strict hypotheses in [11], such as the original hybrid system should be linear, all invariants and guards are μ-insensitive, etc. In [23], the authors present an approach for generating code from a synchronous language kernel with ODEs. However, the exact error introduced in the process of the discretization of ODEs is not considered. Different from the above works, in this paper, we formally identify an equivalent relation, i.e., approximate bisimulation, between a powerful modeling language for hybrid system (i.e., HCSP) and its code model (SystemC), and moreover present rules for generating code models from hybrid models and ensure they are approximately bisimilar for given precisions.

ODE solvers and zero-crossing detection. The generation of code from hybrid systems with interacting continuous evolution and discrete computation relies heavily on the development of ODE solvers and the subsequent zero-crossing detection problem caused by the value precision. Most classical ODE solvers require rollback to adjust the step sizes to decrease the numerical error and to iterate to detect the events [78]. It is of great help to promote the precision and efficiency of numerical ODE solvers to make this iteration process converge more quickly. In [21], the authors propose a more precise and efficient abstraction of the continuous variables, by computing the interval valued step functions based on GRKLib method [20] to over-approximate the continuous functions; and then in [22], they perform a guaranteed integration of the ODE semantics with the discrete system modeled using imperative languages. However, zero-crossing detection is not considered. As an alternative approach, some work on implementation or code generation of hybrid systems explores to eliminate zero crossing detection, which is very costly, by posing some restrictions on the systems. In [47], it is required that each operational mode and its corresponding guard of any event must overlap for a duration greater than the sampling time, thus the event can always be detected. In [55], they deal with a class of linear time invariant systems, and meanwhile require that the witness function obtained after solving the ODEs for each location be monotonic. Thus, the dynamic zero crossing detection is not needed. In [24], they propose a quantized state system (QSS)-based approximation for solving ODEs in Ptolemy: if the three QSS assumptions on ODEs are valid, then there will be no error due to numerical approximation of the integration, thus the zero-crossing detection is predictable in advance. However, for systems that do not satisfy the QSS assumption, the zero-crossing detection can not be guaranteed. In [15] which focuses on translating a hybrid automaton into a Simulink/Stateflow (SLSF) model, zero-crossing detection algorithms inside the simulation routines of SLSF are used for event detection. Additionally, to make sure no transitions will be missed, an ε-relaxation is introduced for each guard constraint, as long as a time step small enough is chosen.

9 CONCLUSION

This paper translates the abstract models of hybrid systems in HCSP to SystemC code, under the premise that either all ODEs of the hybrid system satisfy the GAS condition or the hybrid system executes within bounded time. In both cases, the translation can be done fully automatically, by defining a set of rules for transforming HCSP models with continuous dynamics, to discretized

HCSP models, and to final SystemC code successively. Furthermore, the translation is guaranteed to be correct, by proving that the source model with continuous evolution and the discretized intermediate model are approximately bisimilar with respect to the given precisions, and moreover, the discretized intermediate model and the target code are bisimilar. In particular, the precisions are introduced during the discretization of continuous dynamics and thus cannot be avoided. We have implemented a tool for the automatic code generation from HCSP to SystemC, and applied it to three case studies to illustrate our approach.

Together with the work reported in [77, 85, 86], it forms an integrated framework for formal design of safety-critical embedded systems. In this framework, one can build a Simulink/Stateflow graphical model of the system to be developed first, and conduct extensive simulations, then transform the graphical model automatically to HCSP formal model based on which formal verification can be done, and finally generate correct code from the verified HCSP model.

As a future work, we will concentrate on the optimization of the generated SystemC code corresponding to each construct of HCSP. We will consider transforming from SystemC code into other high-level programming languages such as RUST, C, C++, Java. In addition, we will consider to apply the framework to more complicated practical case studies.

REFERENCES

[1] *dSPACE Release 2019-A.* https://www.dspace.com/.

[2] *Rational Rose. 2017.* http://www-03.ibm.com/software/products/en/rosemod.

[3] *Simulink. 2017.* https://cn.mathworks.com/products/simulink.html.

[4] *Simulink Coder User's Guide, R2019a.* The MathWorks, Inc.

[5] *SysML V 1.5. 2017.* http://www.omg.org/spec/SysML/1.5/.

[6] *TargetLink. 2017.* https://www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm.

[7] J. Abrial. 2010. *Modeling in Event-B: System and Software Design.* Cambridge University Press, New York.

[8] E. Ahmad, Y. Dong, S. Wang, N. Zhan, and L. Zou. 2014. Adding formal meanings to AADL with hybrid annex. In *FACS 2014*. Springer, 228–247.

[9] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. 2000. Modular specification of hybrid systems in CHARON. In *HSCC 2000 (LNCS)*, Vol. 1790. Springer, 6–19.

[10] R. Alur, F. Ivancic, J. Kim, I. Lee, and O. Sokolsky. 2003. Generating embedded software from hierarchical hybrid models. *Languages Compilers and Tools for Embedded Systems* 38, 7 (2003), 171–182.

[11] M. Anand, S. Fischmeister, Y. Hur, J. Kim, and I. Lee. 2010. Generating reliable code from hybrid-systems models. *IEEE Trans. Computers* 59, 9 (2010), 1281–1294.

[12] D. Angeli. 2002. A Lyapunov approach to incremental stability properties. *IEEE Trans. Automat. Control* 47, 3 (2002), 410–421.

[13] D. Angeli and E. Sontag. 1999. Forward completeness, unboundedness observability, and their Lyapunov characterizations. *Systems and Control Letters* 38, 4 (1999), 209–217.

[14] U. Ascher and L. Petzold. 1998. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations* (1st ed.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[15] Stanley Bak, Omar Ali Beg, Sergiy Bogomolov, Taylor T. Johnson, Luan Viet Nguyen, and Christian Schilling. 2017. Hybrid automata: from verification to implementation. *International Journal on Software Tools for Technology Transfer* (12 Jun 2017). DOI:http://dx.doi.org/10.1007/s10009-017-0458-1

[16] R. Balzer. 1985. A 15 year perspective on automatic programming. *IEEE Trans. Software Eng.* 11, 11 (1985), 1257–1268.

[17] G. Berry. 2000. The foundations of Esterel. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner.* 425–454.

[18] S. Bonfanti, M. Carissoni, A. Gargantini, and A. Mashkoor. 2017. Asm2C++: a tool for code generation from abstract state machines to Arduino. In *NFM 2017 (LNCS)*, Vol. 10227. Springer, 295–301.

[19] E. Börger and R. Stark. 2003. *Abstract State Machines: A Method for High-level System Design and Analysis.* Springer, New York.

[20] O. Bouissou and M. Martel. 2006. GRKLib: A guaranteed runge-kutta library. In *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics.* IEEE, Los Alamitos.

[21] O. Bouissou and M. Martel. 2008a. Abstract interpretation of the physical inputs of embedded programs. In *VMCAI, LNCS 4905*. Springer-Verlag, 37–51.

[22] O. Bouissou and M. Martel. 2008b. A hybrid denotational semantics for hybrid systems. In *ESOP, LNCS 4960*. Springer-Verlag, 63–77.

[23] T. Bourke, J. Colaço, B. Pagano, C. Pasteur, and M. Pouzet. 2015. A synchronous-based code generator for explicit hybrid systems languages. In *CC 2015 (LNCS)*, Vol. 9031. Springer, 69–88.

[24] C. Brooks, E. A. Lee, D. Lorenzetti, T. S. Nouidui, and M. Wetter. 2015. CyPhySim: a cyber-physical systems simulator. In *HSCC 2015*. ACM, New York, NY, USA, 301–302.

[25] N. Cataño and V. Rivera. 2016. EventB2Java: a code generator for Event-B. In *NFM 2016 (LNCS)*, Vol. 9690. Springer, 166–171.

[26] D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, C. McKinty, V. Powazny, F. Lang, W. Serwe, and G. Smeding. 2015. Reference manual of the LNT to LOTOS translator (version 6.2). *INRIA/VASY and INRIA/CONVECS* (2015).

[27] M. Chen, X. Han, T. Tang, S. Wang, M. Yang, N. Zhan, H. Zhao, and L. Zou. 2015. MARS: a toolchain for modeling, analysis and verification of hybrid systems. In *ProCoS 2015*. 39–58.

[28] T. Chow. 1978. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.* 4, 3 (1978), 178–187.

[29] A. Deshpande, A. Göllü, and P. Varaiya. 1996. SHIFT: a formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems IV*. 113–133.

[30] F. Dormoy. 2008. SCADE 6: A model based solution for safety critical software development. In *ERTS 2008*. 1–9.

[31] J. Eker, J. Janneck, and et al. 2003. Taming heterogeneity - the Ptolemy approach. *Proc. IEEE* 91, 1 (2003), 127–144.

[32] H. Evrard and F. Lang. 2017. Automatic distributed code generation from formal models of asynchronous processes interacting by multiway rendezvous. *J. Log. Algebr. Meth. Program.* 88 (2017), 121–153.

[33] A. Fehnker and F. Ivancic. 2004. Benchmarks for hybrid systems verification. In *HSCC 2004*. LNCS, Vol. 2993. Springer, 326–341.

[34] A. Gawanmeh, A. Habibi, and S. Tahar. 2004. Enabling SystemC verification using abstract state machines. In *FDL 2004*. ECSI, 649–661.

[35] A. Girard, A. Julius, and G. Pappas. 2008. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems* 18, 2 (2008), 163–179.

[36] A. Girard and G. Pappas. 2007. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control* 52, 5 (2007), 782–798.

[37] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. 1991. The synchronous dataflow programming language LUSTRE. In *Proceedings of the IEEE*. 1305–1320.

[38] N. Halbwachs and P. Raymond. 2007. A Tutorial of Lustre. (2007). http://francois.touchard.perso.luminy.univ-amu.fr/INFO5/Langages/lustre/tutorial.pdf.

[39] D. Harel. 1987. Statecharts: a visual formalism for complex systems. *Sci. Comput. Program.* 8, 3 (1987), 231–274.

[40] J. He. 1994. From CSP to hybrid systems. In *A Classical Mind, Essays in Honour of C.A.R. Hoare*. Prentice Hall International (UK) Ltd., 171–189.

[41] T. Henzinger, P. Ho, and H. Wong-Toi. 1998. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control* 43, 4 (1998), 540–554.

[42] T. Henzinger and J. Sifakis. 2006. The embedded systems design challenge. In *FM 2006 (LNCS)*, Vol. 4085. Springer, 1–15.

[43] T. A. Henzinger. 1996. The theory of hybrid automata. In *LICS 1996*. IEEE, 278–292.

[44] Y. Hur, J. Kim, I. Lee, and J. Choi. 2004. Sound code generation from communicating hybrid models. In *HSCC 2004 (LNCS)*, Vol. 2993. Springer, 432–447.

[45] IEEE Std. 1666-2011. 2011. Open SystemC language reference manual. In *IEEE Standards Association*.

[46] A. Julius, A. D'Innocenzo, M. Di Benedetto, and G. Pappas. 2009. Approximate equivalence and synchronization of metric transition systems. *Systems and Control Letters* 58, 2 (2009), 94–101.

[47] J. Kim and I. Lee. 2003. Modular code generation from hybrid automata based on data dependency. In *Real-Time and Embedded Technology and Applications Symposium*. The 9th IEEE, 160–168.

[48] R. Lanotte and S. Tini. 2005. Taylor approximation for hybrid systems. In *HSCC 2005*. LNCS, Vol. 3414. Springer, 402–416.

[49] E.A. Lee. 2000. What's ahead for embedded software? *Computer* 33, 9 (2000), 18–26.

[50] P. Li, J. Sun, and H. Wang. 2017. Towards code generation from design models. In *SEKE 2017*. KSI Research Inc. and Knowledge Systems Institute Graduate School, 242–247.

[51] I. Llopard, C. Fabre, and A. Cohen. 2017. From a formalized parallel action language to its efficient code generation. *ACM Trans. Embedded Comput. Syst.* 16, 2 (2017), 37:1–37:28.

[52] T. Low and F. Franchetti. 2017. High assurance code generation for cyber-physical systems. In *HASE 2017*. IEEE, 104–111.

[53] L. Luhunu and E. Syriani. 2017. Survey on template-based code generation. In *MODELS 2017*. CEUR-WS.org, 468–471.

[54] R. Majumdar and M. Zamani. 2012. Approximately bisimilar symbolic models for digital control systems. In *CAV 2012*. LNCS, Vol. 7358. Springer, 362–377.

[55] A. Malik, P. S. Roop, S. Andalam, M. Trew, and M. Mendler. 2017. Modular compilation of hybrid systems for emulation and large scale simulation. *ACM Trans. Embed. Comput. Syst.* 16, 5s (2017), 118:1–118:21.

[56] W. Müller, J. Ruf, and W. Rosenstiel. 2003. An ASM based SystemC simulation semantics. In *SystemC: Methodologies and Applications*. 97–126.

[57] X. Peng, H. Zhu, J. He, and N. Jin. 2006. An operational semantics of an event-driven system-level simulator. In *SEW 2006*. IEEE, 190–202.

[58] V. Pham, A. Radermacher, S. Gérard, and S. Li. 2017. Complete code generation from UML state machine. In *MODELSWARD 2017*. SciTePress, 208–219.

[59] A. Platzer. 2010. Differential-algebraic dynamic logic for differential-algebraic programs. *Journal of Logic and Computation* 20, 1 (2010), 309–352.

[60] A. Platzer. 2012. The complete proof theory of hybrid systems. In *LICS 2012*. IEEE, 541–550.

[61] G. Pola, A. Girard, and P. Tabuada. 2008. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica* 44, 10 (2008), 2508–2516.

[62] G. Pola, P. Pepe, and M. Di Benedetto. 2014. Symbolic models for networks of discrete-time nonlinear control systems. In *ACC 2014*. IEEE, 1787–1792.

[63] J. Ralph, A. Abo, and V. J. 1998. *Refinement Calculus: A Systematic Introduction*. Springer.

[64] B. Selic and S. Gérard. 2013. *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems*. Elsevier.

[65] T. Shulga, E. Ivanov, M. Slastihina, and N. Vagarina. 2016. Developing a software system for automata-based code generation. *Programming and Computer Software* 42, 3 (2016), 167–173.

[66] J. Stoer and R. Bulirsch. 2013. *Introduction to Numerical Analysis*. Springer.

[67] J. Sun, J. Dong, J. Liu, and H. Wang. 2001. Object-Z web environment and projections to UML. In *WWW 2001*. ACM, 725–734.

[68] M. Tiller. 2012. *Introduction to Physical Modeling with Modelica*. Springer.

[69] A. Tiwari. 2008. Abstractions for hybrid systems. *Formal Methods in System Design* 32, 1 (2008), 57–83.

[70] J. Tsinias, N. Kalouptsidis, and A. Bacciotti. 1987. Lyapunov functions and stability of dynamical polysystems. *Mathematical Systems Theory* 19, 4 (1987), 333–354.

[71] S. Wang, N. Zhan, and L. Zou. 2015. An improved HHL prover: an interactive theorem prover for hybrid systems. In *ICFEM 2015*. LNCS, Vol. 9407. Springer, 382–399.

[72] Y. Wang, J. Talpin, A. Benveniste, and P. Guernic. 2000. A semantics of UML state-machines using synchronous pre-order transition systems. In *ISORC 2000*. IEEE Computer Society, 96–103.

[73] G. Yan, L. Jiao, Y. Li, S. Wang, and N. Zhan. 2016. Approximate bisimulation and discretization of hybrid CSP. In *FM 2016*. LNCS, Vol. 9995. Springer, 702–720.

[74] N. Zeng and W. Zhang. 2013. A SystemC semantics in guarded assignment systems and its applications with VERDS. In *APSEC 2013*. IEEE, 371–379.

[75] N. Zeng and W. Zhang. 2014. An Executable Semantics of SystemC Transaction Level Models and Its Applications with VERDS. In *ICECCS 2014*. IEEE, 198–201.

[76] N. Zhan, S. Wang, and H. Zhao. 2013. Formal Modelling, Analysis and Verification of Hybrid Systems. In *Unifying Theories of Programming and Formal Engineering Methods*. LNCS, Vol. 8050. Springer, 207–281.

[77] N. Zhan, S. Wang, and H. Zhao. 2016. *Formal Verification of Simulink/Stateflow Diagrams: A Deductive Way*. Springer.

[78] F. Zhang, M. Yeddanapudi, and P. J Mosterman. 2008. Zero-crossing location and detection algorithms for hybrid system simulation. *IFAC Proceedings* 41, 2 (2008), 7967–7972.

[79] H. Zhao, M. Yang, N. Zhan, B. Gu, L. Zou, and Y. Chen. 2014. Formal verification of a descent guidance control program of a lunar lander. In *FM 2014*. LNCS, Vol. 8442. Springer, 733–748.

[80] C. Zhou, J. Wang, and A. Ravn. 1996. A formal description of hybrid systems. In *Hybrid Systems 1996*. LNCS, Vol. 1066. Springer, 511–530.

[81] H. Zhu, J. He, X. Peng, and N. Jin. 2008. Denotational approach to an event-driven system-level language. In *UTP 2008*. Springer, 258–278.

[82] H. Zhu, F. Yang, and J. He. 2010. Generating denotational semantics from algebraic semantics for event-driven system-level language. In *UTP 2010*. Springer, 286–308.

[83] H. Zhu, Y. Zhao, and J. He. 2009. Locality-based normal form approach to linking algebraic semantics and operational semantics for an event-driven system-level language. In *ASWEC 2009*. IEEE, 297–306.

[84] L. Zou, J. Lv, S. Wang, N. Zhan, T. Tang, L. Yuan, and Y. Liu. 2014. Verifying Chinese train control system under a combined scenario by theorem proving. In *VSTTE 2013*. LNCS, Vol. 8164. Springer, 262–280.

[85] L. Zou, N. Zhan, S. Wang, and M. Fränzle. 2015. Formal verification of Simulink/Stateflow diagrams. In *ATVA 2015*.

LNCS, Vol. 9364. Springer, 464–481.

[86] L. Zou, N. Zhan, S. Wang, M. Fränzle, and S. Qin. 2013. Verifying Simulink diagrams via a hybrid Hoare logic prover. In *EMSOFT 2013*. IEEE, 1–10.

Received XXX XXXX; revised XXX XXXX; accepted XXX XXXX