Journal of Systems Architecture 174 (2026) 103711

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Formal design of safety-critical systems with MARS☆

Yihao Yin [a,b,c], Hao Wu [b,c], Wan Liu [e], Shuling Wang [d,c,*], Xiong Xu [d], Wang Lin [e], Fanjiang Xu [d,c], Naijun Zhan [f]

[a] Hangzhou Institute for Advanced Study (HIAS), UCAS, China
[b] Key Laboratory of System Software, ISCAS, China
[c] University of Chinese Academy of Sciences, China
[d] National Key Laboratory of Space Integrated Information System, ISCAS, China
[e] School of Information Science and Technology, Zhejiang Sci-Tech University, China
[f] School of Computer Science, Peking University, China

ARTICLE INFO

Keywords:
Simulink/Stateflow
Model-based design
HCSP
neural network
Code generation
Verification

ABSTRACT

MARS is a toolchain, supporting model-based design of cyber–physical systems (CPS), which integrates informal and formal design. With MARS, a system under development can be graphically modeled by the combination of AADL and Simulink/Stateflow, then the analysis of the graphical model can be conducted via simulation. Furthermore, the graphical model can be automatically transformed to Hybrid Communicating Sequential Processes (HCSP) for formal verification with HHLProver. Within HHLProver, both interactive and automated theorem proving of HCSP processes are supported, with the help of an invariant generator, which is used for automated synthesis of differential invariants of ordinary differential equations based on both numerical methods and neural networks. Finally, ANSI-C code or SystemC code can be generated from the verified HCSP formal model with the guarantee of correctness. As a case study of CPS, this paper applies the MARS toolchain to the design of an intelligent temperature control system, including its modeling, simulation, verification and code generation. This case study demonstrates the advantages of the design of CPS with MARS, including the integration of modeling, simulation, verification and code generation; the integration of informal and formal design, thus providing balance between efficiency and rigidity and enabling a **correct-by-construction** design flow.

1. Introduction

The applications of embedded systems (nowadays called Cyber–Physical Systems (CPS)) are extremely broad encompassing nearly every aspect of modern life, especially in many safety-critical areas such as autonomous driving, medical devices, aerospace and so on. For such systems, any mistake of them may result in catastrophic consequences. However, complex CPS involve closely coupling of discrete control, continuous plants and communications, thus how to efficiently design reliable CPS is very challenging. Both industrial and academic communities have paid increasing attention to design safe CPS, which can be categorized into simulation-based, formal methods based, and their combination. Simulation-based approaches are advocated by industry, such as Simulink/Stateflow (S/S) [1] and AADL [2]. S/S has become a *de facto* model-based design tool in embedded industry. Nevertheless, it is insufficient for the design of safety-critical CPS because of the inherent incompleteness of simulation. AADL provides architecture modeling and analysis of CPS by simulation, and furthermore supports the automated code generation from AADL models to C code. However, it cannot support modeling continuous physical processes as well as their combination with software. Formal methods based approaches are advocated by academic community, which can be further classified into model-checking based and theorem proving based. In model checking based approaches, a CPS is modeled as a hybrid automaton [3,4], and verification is done by computing reachable states [5–7]. In theorem proving based approaches, a CPS is modeled by a compositional modeling language, and verification is conducted through theorem proving, e.g. differential dynamic logic (dL) [8,9]. SCADE [10] tried to combine formal and informal, but failed to bridge the gap between informal graphical models and formal algorithm models.

In order to bridge the gap between informal and formal model-based design for CPS, in our previous work, we developed a toolchain called MARS [11], supporting modeling, analysis, verification, and code generation for CPS. MARS starts to design a graphical model for the system to be developed using the combination of AADL and S/S, by considering the functionality, physicality and architecture of

☆ This article is part of a Special issue entitled: 'SETTA 2024' published in Journal of Systems Architecture.

* Corresponding author at: National Key Laboratory of Space Integrated Information System, ISCAS, China.
E-mail address: wangsl@ios.ac.cn (S. Wang).

https://doi.org/10.1016/j.sysarc.2026.103711
Received 7 June 2025; Received in revised form 1 November 2025; Accepted 18 January 2026
Available online 30 January 2026

the system in a unified framework [12]. Then, formal analysis and verification of the combined graphical model can be conducted via the translation of AADL and S/S into Hybrid CSP (HCSP), an extension of CSP for formally modeling hybrid systems [13]. The HCSP models can be simulated using the HCSP simulator. Additionally, to complement incomplete simulation, they can be verified using HHLProver (Hybrid Hoare Logic prover) implemented in Isabelle/HOL [14], as well as a more automated HHLPy prover [15]. Both of the provers require differential invariants to be annotated for verifying continuous evolution modeled using ordinary differential equations (ODEs), which are completed through an invariant generator. Currently, our invariant generator supports both numerical methods and neural network based approaches for synthesizing differential invariants for ODEs. Specifically, compared to the conference version [16], we represent barrier certificates via neural networks and formally verify their correctness using SMT solvers. This contrasts with [16]'s numerical methods that synthesize invariants with approximation errors. Finally, implementations in SystemC or C can be automatically generated from verified HCSP models [17,18]. The transformation from the combined AADL and S/S to HCSP, and the one from HCSP to SystemC or ANSI-C, are both guaranteed to be correct [18,19]. MARS provides model-based design of safety-critical CPS by allowing switching between formal and informal design, depending on the efficiency, cost and rigidity.

In this paper, we apply MARS to the design of an intelligent temperature control system (ITCS), including its modeling, simulation, verification, and code generation. Specifically, the graphical model of the system is constructed using S/S, and then it is translated into an HCSP model, based on which simulations are performed. Then, we verify the generated HCSP model by synthesizing differential invariants in the form of barrier certificates using two distinct methods, the numerical method based on Semidefinite Programming (SDP) and the neural network method. The numerical method is computationally efficient and easily implementable for low-dimensional systems, but may contain numerical approximation errors; in contrast, the neural network method can avoid potential numerical errors and provide formal guarantees through subsequent SMT verification, but sometimes faces the scalability limitation of SMT solvers. Hence, these two methods complement each other and offer more flexible choices to users. From the verified HCSP model, we continue to generate ANSI-C code, which is guaranteed to be reliable given the correctness of the translation proved. The goal of this paper is to demonstrate the entire process of the model-based design approach, by applying MARS for the modeling, simulation, verification, and code generation of the case study, thereby validating the applicability of MARS for the design of complex embedded systems.

This paper is an extension of our work in SETTA 2024 [16], which contains the following updates and extensions:

- First, we employ neural networks to automatically generate differential invariants (i.e., barrier certificates) for the hybrid system in our case study, with correctness formally verified via SMT solvers. It eliminates the potential unsoundness due to numerical errors in [16] and ensures that the safety properties hold for the formal models and are consistently preserved through all subsequent transformations, particularly in the automatically generated SystemC/ANSI-C code, thereby strengthening the reliability of the overall model-based design process.
- Second, inspired by the neural network derived invariants, we redesigned the template for numerical methods and a linear invariant is generated for the case study. The comparison shows that the differential invariant derived from the two methods are consistent.
- Third, we have expanded the paper to include the following important aspects, including the subset of Simulink blocks supported by MARS, an extended description of the MARS toolchain, the numerical and neural network based barrier certificate synthesis, and the discussion of model-driven development and invariant generation in terms of barrier certificates in the related work.

Paper organization. After the related work in Section 2, the rest of the paper is organized as follows. Section 3 introduces some preliminary knowledge of this paper. Section 4 introduces the ITCS case study, and Section 5 presents the modeling, simulation, verification, and code generation of ITCS using MARS. Finally Section 7 concludes the paper.

2. Related work

Model-based design for CPSs. Several unified model-based design frameworks have been proposed for CPSs. Metropolis [20] is a platform-based design environment for heterogeneous systems, which provides simulation, verification, and code synthesis by transforming all models to a unified meta-model language. Ptolemy [21] aims to design heterogeneous systems that combine different models of computation in terms of actors and provides modeling and simulation techniques for the combined models. Functional Mock-up Interface (FMI 3.0) [22] enables the exchange and co-simulation of dynamic component models and couples different simulation tools at system level by coordinating and synchronizing different models. However, all of the above work support very limited facilities to handle continuous behaviors, and furthermore, neither Ptolemy nor FMI is designed for architecture modeling and analysis. There are some works that focus on separate aspects of model-based design of CPSs. To overcome the lack of formal semantics and limited verification capabilities in Simulink, the work [23,24] proposed a contract-based semantics for hybrid Simulink diagrams based on a set of contract composition operators and a verification framework via refinement calculus. The work [25] presented the modeling and formal timing analysis of AADL components and their Simulink-based functional behaviors in a polychronous model of computation, but the continuous behavior is not considered. SCADE [26] builds upon synchronous data-flow language Lustre [27], and supports formal verification of the models and automatic code generation from their models with formal correctness guarantee. Zélus [28] extends Lustre with ODEs for designing and implementing hybrid systems, and supports analysis of models and code generation [28–30]. There are extended Event-B framework based on refinement and proof for modeling and verifying hybrid systems, for instance, core hybrid Event-B [31–33]. Differential dynamic logic is developed for modeling and reasoning about hybrid systems [34], for which KeYmaera X prover [35] is implemented for safety verification of dynamic systems.

Synthesis of differential invariants. Throughout the whole design of systems using MARS, one crucial step is the safety verification of hybrid systems with combined discrete and continuous behavior modeled using ordinary differential equations (ODEs). Mars provides an invariant generator for synthesizing differential invariants for ODE systems. The primary approach for representing differential invariants is using barrier certificates, which are real-valued functions encoding invariant conditions. Prajna et al. first proposed the definition of barrier certificates for verifying hybrid system safety, transforming the problem into SDP via Sum-of-Squares (SOS) techniques [36,37]. Subsequent efforts attempted to improve the expressiveness of the barrier certificate conditions as well as the computational tractability of the resulting constraints [38–42]. Typically, these methods rely on predefined polynomial templates and the translated constraints are solved by existing numerical solvers. However, when dealing with high dimensional systems, the constraint solving procedure can be inefficient and numerically unstable.

In recent years, machine learning techniques have been increasingly applied to the verification of complex dynamical systems and hybrid systems. Zhao et al. first proposed a neural network-based approach for synthesizing barrier certificates [43,44]. They employ feedforward neural networks to represent barrier certificates and encode the conditions into the loss function to guide training. The networks are trained on data sampled from the systems' state spaces and then verified using Satisfiability Modulo Theories (SMT) solvers. The works [45,

46] designed a Counter-Example Guided Inductive Synthesis (CEGIS) framework, using counter-examples generated by SMT solvers to guide the training process. The framework has been further extended and implemented into a tool, called FOSSIL [45,47], which allows verifying more temporal properties for more complex hybrid systems. The tool SynHbc [48] took a different idea by combining learning-based framework with optimization techniques. It targets at polynomial barrier certificates but employs neural networks to learn the unknown coefficients. In the verification phase, it utilizes numerical methods to either prove the correctness of the learned barrier certificates or generate counter-example points.

3. Background

In this section, we introduce some preliminary knowledge for this paper, including Simulink, HCSP, and the MARS toolchain.

3.1. Simulink

Simulink [1] is a graphical environment for model-based design of dynamical systems, supporting description of both discrete-time and continuous-time behavior. A Simulink model contains a set of blocks, subsystems, and wires, where blocks and subsystems cooperate by exchanging data flows through connected wires. Wires can be considered as variables holding these data values. As basic units for building Simulink models, each block is defined with input and output ports, and methods that define how outputs and internal states are changed. Blocks can be grouped into subsystems to establish hierarchical diagrams. To ease modeling, Simulink provides an extensive library of pre-defined blocks and subsystems for building and managing diagrams, and also a rich set of fixed-step and variable-step solvers for analyzing dynamical systems through simulation.

3.2. HCSP

Hybrid CSP (HCSP) [49] is a formal language for describing HSs, which is an extension of CSP by introducing ODEs (ordinary differential equations) for modeling continuous evolution.

HCSP includes common constructs such as assignment, internal choice, sequential composition and conditional statement. Besides, it includes more constructs explained as follows:

- Input $ch?x$ receives a value along the channel ch and assigns it to variable x. Output $ch!e$ sends the value of e along ch.
- Repetition c^* executes c for a nondeterministic finite number of times.
- Continuous evolution $\langle \dot{\vec{x}} = \vec{e} \& B \rangle$ evolves continuously according to the differential equation $\dot{\vec{x}} = \vec{e}$ as long as the *domain* B holds, and terminates whenever B becomes false. Communication interruption $\langle \dot{\vec{x}} = \vec{e} \& B \rangle \trianglerighteq \llbracket_{i \in I} (ch_i * \to c_i)$ behaves like $\langle \dot{\vec{x}} = \vec{e} \& B \rangle$, except it is preempted as soon as one of the communication events $ch_i *$ takes place, and then is followed by the corresponding c_i.
- Parallel composition $pc_1 ||_{cs} pc_2$ behaves as pc_1 and pc_2 run independently except that all communications along the set of common channels cs between pc_1 and pc_2 are synchronized.

MARS enriches HCSP with constructs on modularity, including **module** for encapsulating a sequential process and **system** for parallel composition of modules.

3.3. Barrier certificates

One of the key sub-tasks in HCSP verification is the safety verification problem of a single ODE system. Formally, given an ODE system $\dot{\mathbf{x}} = f(\mathbf{x})$ defined over the domain X_D, the safety verification problem asks to prove that all trajectories starting from the initial region X_I will never reach the unsafe region X_D. The deductive reasoning approach utilizes the concept of differential invariants to witness systems' safety, where a differential invariant is a subset of states $\Omega \subset X_D$ satisfying the following three conditions: (1) the initial region X_I is contained in Ω (*Initial Condition*); (2) the unsafe region X_D and Ω are disjoint (*Saturation Condition*); (3) Ω keeps continuous (differential) inductiveness, i.e., all trajectories starting from Ω remain within Ω (*Differential Inductive Condition*). If such a differential invariant exists, one can deduce that the reachable set of states and the unsafe set are disjoint, viz., the system cannot reach a state in the unsafe set from the initial set.

To synthesize desired different invariants, the authors of [50] first proposed the idea of barrier certificates. A barrier certificate is a continuously differentiable real-valued function $B(\mathbf{x})$ defined over the system domain X_D. It has been proved (first in [50] with a flaw, later in [51]) that the zero sub-level set $\{\mathbf{x} \in \mathbb{R} \mid B(\mathbf{x}) \leq 0\}$ is a differential invariant if the following conditions are satisfied:

$$B(\mathbf{x}) \leq 0 \quad \forall \mathbf{x} \in X_I \tag{1}$$

$$B(\mathbf{x}) > 0 \quad \forall \mathbf{x} \in X_U \tag{2}$$

$$L_f B(\mathbf{x}) < 0 \quad \forall \mathbf{x} \in X_D \text{ s.t. } B(\mathbf{x}) = 0. \tag{3}$$

The conditions above are often referred to as the non-convex barrier certificate conditions, in the sense that the set of functions $B(\mathbf{x})$ satisfying these conditions form a nonconvex set. To make the conditions amenable to convex optimization techniques, the work [38] proposes exponential-type conditions by strengthening 3 into

$$L_f B(\mathbf{x}) \leq \lambda B(\mathbf{x}) \quad \forall \mathbf{x} \in X_D \tag{4}$$

where λ is a real number. There also exist several other type of barrier certificate conditions [40–42,52]. Solving these conditions for the unknown function $B(\mathbf{x})$ yields a concrete barrier certificate.

The most popular approach for solving barrier certificate conditions is by using polynomial optimization. The main idea is to first set the function $B(\mathbf{x})$ to certain polynomial forms with parameters, and then translate the conditions into constraints using algebraic techniques. Finally, the constraints are solved by numerical solvers, such as SDP solvers. Such numerical approaches are highly efficient when dealing with relatively low-dimensional dynamical systems. However, their results may contain numerical errors that require ad-hoc numerical analysis techniques (such as [53]) to mitigate. In Section 5.3, we will illustrate this approach with our case study.

3.4. Synthesizing barrier certificates using neural networks

Since neural networks have the capability to approximate arbitrary functions, using them as templates for barrier certificates offers advantages over polynomials [43]. A typical neural network consists of many interconnected neurons organized in a hierarchical structure. Each neuron serves as a unit that responds to weighted inputs received from other neurons and produces an output. Assume that $\mathbf{x} \in \mathbb{R}^n$ represents the n-dimensional input vector of the neural network, $L-1$ represents the number of hidden layers, and y is the scalar output of the neural network with respect to the input $\mathbf{x}$. Thus, the feedforward neural network can be expressed with the following structure:

$$\begin{cases} \mathbf{z}_l = \mathbf{W}_l * \mathbf{x}_{l-1} + \mathbf{b}_l & 0 < l \leqslant L \\ \mathbf{x}_l = \sigma(\mathbf{z}_l) & 0 < l \leqslant L, \\ y = \mathbf{x}_L \end{cases} \tag{5}$$

where,

- $\mathbf{z}_l$ denotes the output vector of the linear operation in the lth layer;
- $\mathbf{W}_l$ and $\mathbf{b}_l$ denote the weight matrix and bias vector for the lth layer of the neural network, respectively;

- σ denotes the activation function. Common activation functions include the Tanh function, Sigmoid function, ReLU function.

Based on barrier certificate conditions 1–3, a loss function is designed for the training dataset:

$$L_B = \sum_{\mathbf{x}\in D_I} \max\{B(\mathbf{x}) + \tau_I\} + \sum_{\mathbf{x}\in D_U} \max\{-B(\mathbf{x}) + \tau_U\} + \sum_{\mathbf{x}\in D_D, B(\mathbf{x})=0} \max\{L_f B(\mathbf{x}) + \tau_D\}, \tag{6}$$

where τ_I, τ_U, τ_D serve as offsets that are included to enhance the numerical stability during training, and D_I, D_U and D_D are the datasets sampled from X_I, X_U and X_D, respectively. Note that the loss function is evaluated over sampled points, so we can target at the original nonconvex barrier certificate, thus avoiding the additional conservativeness introduced by exponential-type conditions.

To generate a neural barrier certificate candidate $B(\mathbf{x})$, we employ gradient descent techniques to minimize L_B. When the loss decreases to 0, it indicates that the learned neural network can be a candidate barrier certificate. Since the learned neural candidate cannot formally satisfy the barrier certificate conditions, we design a counterexample-guided framework based on the SMT solver for verification, which is designed to find states that violate the barrier conditions in 1–3. To achieve this SMT-based CEGIS framework, we formulate the negation of the three barrier conditions and verify the conditions are unsatisfiable, as follows:

$$\begin{aligned} &x \in X_I \wedge B(\mathbf{x}) > 0 \\ &x \in X_U \wedge B(\mathbf{x}) \leq 0 \\ &x \in X_D \wedge L_f B(\mathbf{x}) \geq 0 \wedge B(\mathbf{x}) = 0. \end{aligned} \tag{7}$$

We choose dReal as our SMT verifier, which ensures the correctness of its unsat decisions. Therefore, when dReal returns unsat for the given formula (7), it confirms that no solution exists within the specified precision, and the candidate barrier certificate $B(\mathbf{x})$ is valid. Otherwise, it means that dReal has found a counterexample that violates the safety properties of the barrier certificate.

This neural-network-based method does not depend on polynomial templates and can deal with general non-polynomial systems. However, its bottleneck primarily lies in the computation burden of SMT solving, especially when dealing with high-dimensional systems with highly nonlinear dynamics. One possible solution is to decompose a system into sub-systems and learn barrier certificates for each individual subsystem, as we will show in Section 5.4. In practice, the numerical methods and the neural network methods complement each other. This dual-methodology ensures that users can select the most appropriate synthesis technique based on the system's dimensionality, complexity, and tolerance for numerical approximations, thereby maximizing the likelihood of successfully generating verifiable differential invariants.

3.5. MARS

The architecture of the toolchain MARS [11] is shown in Fig. 1, where AADL and S/S indicate graphical models that serve as input to the toolchain; HCSP indicates formal models, with their simulation and verification tools; and SystemC and ANSI C indicate generated code. To design a safety-critical system using MARS, users can choose to build graphical models from the top layer and do analysis through simulation by transforming the combined model to C, or build the formal HCSP models directly, which also have a simulator implemented. Mars implements an automatic translator from combined AADL and S/S graphical models to formal HCSP models for further verification. Currently, Mars can handle a wide subset of Simulink blocks, presented in Fig. 2. We have also evaluated some industrial-grade case studies such as the Toyota case in [54] and the only few exception is variable time/transport delay that closely relate to the setting of ODE solvers during simulation.

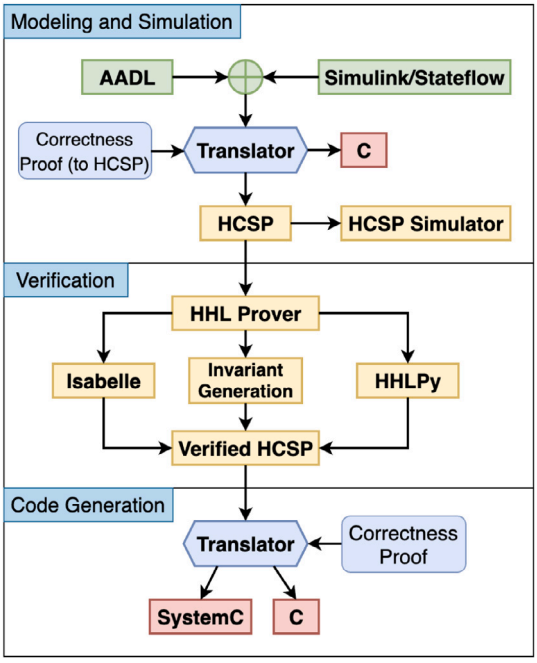


Fig. 1. The architecture of MARS.

Formal verification of HCSP models is done by HHL Prover, which includes the interactive verifier implemented in Isabelle/HOL [9], the automatic verifier HHLPy for verifying HCSP sequential subset [15], and differential invariant generator for reasoning about ODEs [55]. Our invariant generator supports both numerical methods and neural network based approaches for ODE differential invariant synthesis, by generating barrier certificates presented above, which will be used in this paper for the verification of the case study.

Finally, implementations in SystemC [17] or ANSI-C [18] are automatically generated from the verified HCSP processes. Both the transformation from graphical models to HCSP models and the one from HCSP to SystemC or ANSI-C, can be done automatically and furthermore are guaranteed to be correct by proving the consistency between the models in different layers based on their formal semantics [13, 18]. MARS supports the transformation of subsets of AADL and S/S, which include the main features of CPS including discrete-time control, continuous evolution, event-based control, *etc*. Our approach allows model-based design of safety-critical CPS based on graphical and formal models and proven-correct translation procedures.

4. An intelligent temperature control system (ITCS)

The case study of ITCS is taken from the official website of S/S [1], as shown in Fig. 3. The system is modeled as an S/S diagram, assembled from a combination of continuous blocks, discrete blocks and subsystems, mimicking a real-world scenario wherein the indoor temperature is regulated by automatically toggling the heater on and off in response to changes in outdoor temperature. In this section, we introduce the S/S model of the case study from the overall top structure and the encapsulated subsystems respectively.

4.1. The overall of ITCS

The system receives inputs from the left two constant blocks, which set the average outdoor temperature to 50°F and the house temperature to 70°F respectively. The control system is designed to maintain the indoor temperature at approximately 70°F, with allowance of given

Category	Blocks
Sources	`Constant, Clock, Sine Wave, Signal Editor, Signal Builder, From Workspace, Discrete Pulse Generator`
Continuous	`Integrator, Integrator Limited, Transfer Function`
Discontinuities	`Hit Crossing, Saturation`
Discrete	`Discrete PID Controller, Unit Delay`
Logical and Bit Operations	`Logical Operators, Relational Operators`
Math Operations	`Add, Bias, Gain, MinMax, Abs, Product, Sqrt, Square, Sign`
Signal Routing	`Data Store Memory, Bus Creator, Mux, Demux, Selector, Switch, Merge`
Signal Attributes	`IC` (Initial Condition for signals), `Unit Conversion`
Ports & Subsystems	`Normal Subsystem, Enabled Subsystem, Triggered Subsystem, Model` (Reference the specified model)
Lookup Tables	`n-D Lookup Table`
Sinks	`Scope, Display, Terminator`

Fig. 2. The Simulink blocks supported by MARS.

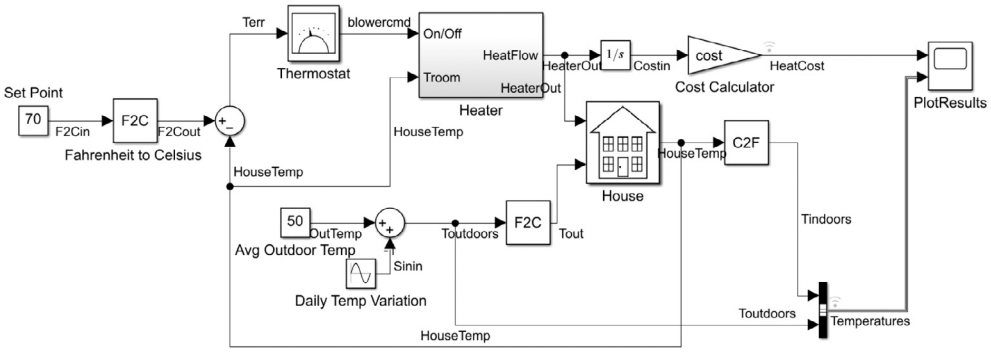


Fig. 3. The S/S Model of ITCS [1].

up and down fluctuations. The system uses a sine function to represent the daily outdoor temperature variation and superimposes it over `OutTemp` (i.e. 50) to model the changing outdoor temperature. The F2C block converts the temperature from Fahrenheit to Celsius, then the converted `Tout` is sent to the House subsystem as one input. Concurrently, the constant house temperature 70°F is also converted via F2C block and then the difference of it with the actual house temperature is calculated (i.e. `Terr`), to be input of the Thermostat subsystem, which determines whether the heater should be activated or not. The Thermostat block then transmits its judgment (i.e. `blowercmd`) as input to the Heater subsystem. The Heater subsystem receives the actual house temperature as another input and calculates the heat flow, i.e. `HeaterOut`. The heat flow is sent to the House subsystem as another input, and moreover, it is integrated via an integrator block and then multiplied with a constant via a gain block, to obtain the final cost, i.e. `HeatCost`. At the same time, the system calculates the real-time indoor temperature `HouseTemp` through the House subsystem. The final output graph is a line chart composed of the indoor and outdoor temperatures in the form of Fahrenheit degree, and the cost of the heater. The subsystems Thermostat, House and Heater will be explained subsequently.

4.2. The subsystems

The model of ITCS consists of three subsystems: Thermostat, House and Heater, explained in the following parts.

4.2.1. Thermostat subsystem

The Thermostat subsystem contains only one Relay block, as shown in Fig. 4(a). It maintains an internal state that records the status of the switch. When the input signal exceeds a certain threshold (rise threshold, 215/9 °C here), the switch closes and the output is 0, turning off the heater; when the input signal is below another threshold (fall threshold, 165/9 °C), the switch opens and the output is 1, turning on the heater; otherwise, when the input signal is between 165/9 °C and 215/9 °C, the switch is not changed and the output keeps the value of the switch state. Due to the control of Thermostat subsystem, the indoor temperature can be maintained within a certain range.

4.2.2. Heater subsystem

The Heater subsystem implements the heater as shown in Fig. 4(b). The input `On/Off` receives the output command of Thermostat subsystem (1 or 0), `Troom` is the actual house temperature from the House subsystem, and `Theater` is the temperature of the hot air from the heater, which is set to constant 50 °C here. When the heater is on, the output `HeatFlow` is calculated by the following equation:

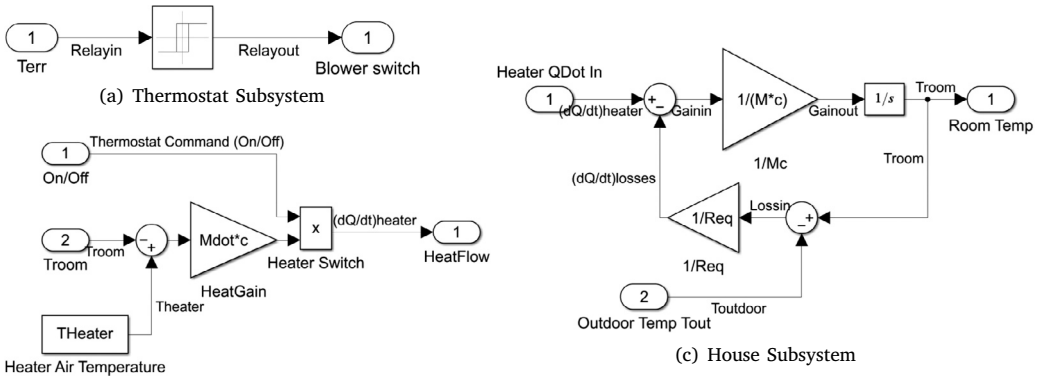
$$(\frac{dQ}{dt})_{heater} = (T_{heater} - T_{room}) * Mdot * C$$

where, $(\frac{dQ}{dt})_{heater}$ represents the heat flow from the heater into the room, $Mdot$ the air mass flow rate through the heater (kg/hr), C the heat capacity of air at constant pressure, and T_{heater}, T_{room} correspond to `Theater` and `Troom` respectively. The output `HeaterFlow` will serve as an input to both the integrator block and the House subsystem, as shown in Fig. 3.

4.2.3. House subsystem

The House subsystem controls the indoor temperature of the house, as depicted in Fig. 4(c). The input `In` receives the heat flow generated by the Heater, and `Tout` inputs the outdoor temperature. It calculates the final indoor temperature based on the inputs using the following equations:

$$(\frac{dQ}{dt})_{losses} = \frac{T_{room} - T_{outdoor}}{R_{eq}}$$



(a) Thermostat Subsystem

(b) Heater Subsystem

(c) House Subsystem

Fig. 4. The subsystems.

$$\dot{T}_{room} = \frac{1}{M * C} * ((\frac{dQ}{dt})_{heater} - (\frac{dQ}{dt})_{losses})$$

where, $T_{outdoor}$ represents the outdoor temperature `Tout`, T_{room} and C defined as above, R_{eq} the equivalent thermal resistance of the house, M the mass of air inside the house. As shown in the equations, $(\frac{dQ}{dt})_{losses}$ represents the loss rate of the heat in the environment, which is determined by the difference between the actual room temperature and the outdoor temperature, divided by the house thermal resistance. $\dot{T}_{room}$, the final gained heat rate of the room, also the derivative of the room temperature with respect to time, is the difference between the heat flow rate and the loss rate, divided by $M * C$. A loop is formed as the room temperature is also taken as an input of the Lossin block to calculate the loss rate of the heat. In S/S, the diagrams with algebraic loops are considered invalid, while the blocks which maintain internal states such as Integrator or Unit Delay blocks can break the loop. So with existence of the integrator block for `Troom` in House subsystem, the loop in this subsystem and also the main loop in the top diagram (due to the backward transition from House to previous parts) are both valid. Notice that different from $\dot{T}_{room}$, both $(\frac{dQ}{dt})_{losses}$ and $(\frac{dQ}{dt})_{heater}$ are named in the form of differential equations for ease of understanding their meanings and the relations between each other, without actual differential operations.

5. Formal design of ITCS

In this section, we show how to conduct formal design of ITCS starting from the built graphical model, including constructing its formal model represented by HCSP, its simulation based on the HCSP model, verification of the HCSP model, and code generation from the verified HCSP model.[1]

5.1. Translation to HCSP model

We first apply the toolchain MARS to transform the S/S model of ITCS to HCSP formal model. Taking the S/S model presented in Fig. 3 in .xml format as input, the tool generates the HCSP model as shown in Fig. 5, which is to be served as the foundation for subsequent simulation and verification of ITCS. The generated HCSP model is a `system` from the overall structure, which contains one `module` transformed from the S/S model.

Before introducing the HCSP model, we briefly explain the strategy of MARS for transforming a S/S model. It first determines the sample times of all blocks, including the ones inside subsystems, based on which each block is classified as either discrete or continuous; then separates the whole diagram into discrete and continuous parts; finally, transforms the discrete and continuous parts individually first and then put them together in correct execution order to form the whole HCSP model of the S/S diagram. The complexity of the transformation is $O(n^2)$ where n denotes the number of blocks in source models due to the sorting procedure in correct execution order. The generated HCSP model for a given S/S diagram $\mathcal{D}$ has the following structure:

$$HCSP(\mathcal{D}) \mathrel{\widehat{=}} \text{Output; Init;} \left(\text{ Discrete; } \langle \dot{t} = 1, \dot{\mathbf{y}} = \Gamma(\mathbf{x}) \& t < period \rangle \text{; TimeUpdate; } \right)^* \quad (8)$$

It starts from Output, which is a sequence of assignments to the outputs of $\mathcal{D}$ by their respective values, followed by the initialization of variables and then a repetition process. Init initializes some variables including internal state variables, the outputs of integrators and discrete constant blocks, and the auxiliary time variables introduced for managing the execution time of the whole model. Discrete represents the transformed process of discrete blocks of $\mathcal{D}$, and $\dot{\mathbf{y}} = \Gamma(\mathbf{x})$ is the combined vector of the ODEs for all integrator blocks after variable substitution corresponding to other non-integrator continuous blocks; TimeUpdate defines the update of the auxiliary time variables after each loop. The loop period *period*, constraining the domain of the ODEs, is the great common divisor of sample times of all discrete blocks of $\mathcal{D}$.

As presented in Fig. 5, the output list (Line 1) includes *HeatCost* representing the cost of heat, and the joint *Temperatures* for the indoor and outdoor temperatures, that correspond to the outputs of the S/S model. Each of the outputs is assigned to its respective values. The main body (Lines 4–20) implements the whole S/S diagram. It starts from the initialization of a sequence of variables, among which *Costin* and *HouseTemp* are the outputs of two integrator blocks, *F2Cin* the output of a discrete constant block, *Thermostat_sub_Relay1_state* the internal state of Relay block, and $tt, t, \_tick$ the auxiliary time variables, then followed by a repetition process. At each round of the repetition, first the discrete blocks of the case study are executed in a correct order, then the two ODEs defining the derivatives of *Costin* and *HouseTemp*, plus the one with *tt* recording the execution time of each round, are put together to constitute the transformed process of the continuous part. Notice that variable substitution is performed on the right hand sides of each ODE, by replacing recursively the outputs of each non-integrator continuous block as functions of its inputs, till the equations only contain ODE variables and variables from the separate discrete part, e.g. *blowercmd* and *HouseTemp* occurring in the ODEs. Here 0.001 is the period of the whole diagram, and variables t and $\_tick$ represent the accumulated execution time and the number of execution loops respectively.

Notice that the equation of $HouseTemp$ depends on the value of *blowercmd*, which is either 0 or 1, the output of Thermostat subsystem. We will consider the two different cases separately in the verification of ITCS.

[1] The implementation code of MARS and the case study can be found at https://gitee.com/bhzhan/mars.

```
module P():
output HeatCost = Costin * cost,
    Temperatures = [9 / 5 *
    HouseTemp + 32, 50 + 15 *
    sin(0.262 * t)];
begin
  t := 0;
  _tick := 0;
  F2Cin := 70;
  Thermostat_sub_Relay1_state := 0;
  tt := 0;
  Costin := 0;
  HouseTemp := 20;
  {
    F2Cout := 5 / 9 * (70 - 32);
    Terr := F2Cout - HouseTemp;
    blowercmd := (if Terr > 5*(5/9)
        then 1 else (if Terr <
        -5*(5/9) then 0 else
        Thermostat_sub_Relay1_state))
    Thermostat_sub_Relay1_state :=
        blowercmd;
    {tt_dot = 1, Costin_dot =
        blowercmd * ((-HouseTemp +
        50) * (Mdot * c)),
        HouseTemp_dot = (blowercmd *
        ((-HouseTemp + 50) * (Mdot *
        c)) - (HouseTemp - 5 / 9 *
        (50 + 15 * sin(0.262 * t) -
        32)) * (1 / Req)) * (1 / (M
        * c)) & tt < 0.001}
    t := t + tt;
    _tick := _tick + 1;
    tt := 0;
  }*
end endmodule
system P=P() endsystem
```

Fig. 5. The HCSP model of ITCS.

Fig. 6. Simulator interface after importing the HCSP model.

5.2. Simulation

MARS guarantees that the generated HCSP model and the corresponding source S/S diagram are consistent by formally defining their semantics and building the equivalence between them [13]. But for visualizing the behavior of the generated HCSP model, we also utilize HCSP simulator integrated in MARS to analyze the generated HCSP model, which reflects the behavior of the source S/S model as expected. HCSP simulator is designed to calculate the execution paths of HCSP processes and visualize them in the graphical interface. As shown in Fig. 6, the left "Read HCSP file" button is used to load the input HCSP models, and on the right, the number of simulation steps, starting position, and ending position can be set. On the left side of the interface, the loaded HCSP process is displayed, with the current executed statement highlighted, and below it, the values of process variables changing over time are displayed; On the right, the trace of all events produced during the execution is shown, including discrete steps, time progress or communication events.

Fig. 6 shows the simulation result of the HCSP model of ITCS, by setting the step size to 0.001 s and number of steps to 200,000 respectively. Through the result, we can check whether its behavior aligns with expectation. On the left bottom of Fig. 6, the lines from top to bottom represent the indoor temperature change, the outdoor temperature change, and the cost incurred after the heater is turned on, respectively. The changes and fluctuations of the three curves conform to the design requirement, especially, the house temperature is always within a safe range, to be given in detail in the following verification part.

5.3. Verification with numerical methods

To remedy incomplete simulation, the verification of the HCSP model is needed to guarantee the design requirement strictly. In MARS, this is achieved by HHL Prover through a Hoare-logic style deductive verification method [9,56]. HHL Prover contains three parts: HHLPy [15] for deductive verification of sequential HCSP processes covering ODEs, achieving automation of verification through the annotation of differential and loop invariants and the integration with SMT solvers for solving logical formulas; the interactive Isabelle/HOL prover, that is implemented for the whole HCSP with concurrency and communication, conducting proof of HCSP specifications with pre-/post-conditions by manually choosing corresponding inference rules; and the invariant generation, which synthesizes differential invariants for reasoning about ODEs through template-based numeric methods [50,55] or neural network-based methods [43,45], to be employed for the former two provers if necessary.

This case study demonstrates the procedure for verifying a safety specification: "if the initial house temperature is within the range of 165/9 °C to 215/9 °C, it will always remain between 145/9 °C and 235/9 °C (i.e. *the safe range*)". For simplicity, the initial system state is defined by the region $165/9 \le T \le 215/9$, while the unsafe region is $T \le 145/9 \vee 235/9 \le T$, where T stands for $HouseTemp$. Our objective is

to verify that all system trajectories originating from the initial region will never enter the unsafe region. In the following, we show how to synthesize a barrier certificate using numerical methods.

Step 1: Simplifying system. From the HCSP model, we see that the system consists of two modes, depending on whether *blowercmd* is 1 or 0, where the dynamics of the temperature are described by

$$\dot{T} = -3.334 \cdot T + 10.916 \cdot \sin(0.262t) + 114.315, \quad \text{(mode 1)}$$

and

$$\dot{T} = -1.310 \cdot T + 10.916 \cdot \sin(0.262t) + 13.099, \quad \text{(mode 2)}$$

respectively. The system will switch to (mode 1) when $T < 165/9$, and switch to (mode 2) when $T > 215/9$.

Note that the above expressions contain a trigonometric function, which will be difficult to reason about. To address this issue, inspired by [57], two fresh variables v and u are introduced to represent $\sin(0.262t)$ and $\cos(0.262t)$. Then, both (mode 1) and (mode 2) can be transformed to a 3-dimensional system in variables (T, v, u) with polynomial dynamics as follows:

$$\begin{pmatrix} \dot{T} \\ \dot{v} \\ \dot{u} \end{pmatrix} = \begin{pmatrix} -3.334 \cdot T + 10.916 \cdot v + 114.315 \\ 0.262 \cdot u \\ -0.262 \cdot v \end{pmatrix}, \quad \text{(mode 1')}$$

and

$$\begin{pmatrix} \dot{T} \\ \dot{v} \\ \dot{u} \end{pmatrix} = \begin{pmatrix} -1.310 \cdot T + 10.916 \cdot v + 13.099 \\ 0.262 \cdot u \\ -0.262 \cdot v \end{pmatrix}. \quad \text{(mode 2')}$$

Note that the derivative $\dot{v}$ is obtained by $\dot{v} = \frac{\mathrm{d}\sin(0.262t)}{\mathrm{d}t} = 0.262\cos(0.262t) = 0.262u$. The computation of $\dot{u}$ is similar. Moreover, we add a new constraint $u^2 + v^2 = 1$ as a state space constraint because $\sin^2(x) + \cos^2(x) = 1$.

Step 2: Setting templates. Template-based synthesis leverages parameterization. In this step, we first set a parameterized template for the target barrier certificate, and then formulate the conditions with respect to the template.

Since the system's two subsystems are linear, we introduce two linear barrier certificates, Φ_1 and Φ_2 for (mode 1') and (mode 2'), respectively. Each Φ_i is a parameterized linear expression in variable T, v, u of the following form

$$\Phi_i = c_0 + c_1 \cdot T + c_2 \cdot v + c_3 \cdot u, \quad (9)$$

where $c_i \in \mathbb{R}$, for $0 \le i \le 3$, are unknown real coefficients to be determined during invariant synthesis. In general, the template can be set to polynomial forms.

Here, we use the exponential-type barrier certificate conditions [38] explained in the Background section. The conditions for Φ_1 and Φ_2 are given as follows, for $i = 1, 2$,

$$\forall(T, v, u).\ 165/9 \le T \le 215/9 \wedge u^2 + v^2 = 1 \implies \Phi_i(T, v, u) \le 0, \quad (10)$$

$$\forall(T, v, u).\ \mathsf{Unsafe}_i(T) \wedge u^2 + v^2 = 1 \implies \Phi_i(T, v, u) > 0,$$
$$\text{with } \mathsf{Unsafe}_1(T) = T > 235/9, \mathsf{Unsafe}_1(T) = T < 145/9 \quad (11)$$

$$\forall(T, v, u).\ 120/9 \le T \le 250/9 \wedge u^2 + v^2 = 1 \implies \mathfrak{L}_{f_i}\Phi_i(T, v, u) \le \lambda_i \Phi_i(T, v, u), \quad (12)$$

$$\forall(T, v, u).\ \mathsf{Switch}_i(T) \wedge u^2 + v^2 = 1 \implies \Phi_{3-i}(T, v, u) \le \mu_i \Phi_i(T, v, u),$$
$$\text{with } \mathsf{Switch}_1(T) = 214/9 \le T \le 216/9, \mathsf{Switch}_2(T) = 164/9 \le T \le 166/9, \quad (13)$$

where λ_i is any real number, μ_i is any non-negative real number, $\mathfrak{L}_{f_i}(\Phi_i) = \langle \nabla\Phi_i, f_i \rangle$ is the Lie derivative of function Φ_i w.r.t. f_i (here, ∇ is the gradient notation and $\langle \cdot, \cdot \rangle$ is the dot product). In our experiments, we set $\lambda_i = -1$ and $\mu_i = 0.1$ for $i = 1, 2$. Intuitively, constraint (10) ensures the initial region is contained within the region defined by $\Phi_i(T, v, u) \le 0$, while (11) guarantees exclusion of the unsafe region from this region. Constraint (12) establishes that $\Phi_i(T, v, u) \le 0$ is a differential invariant, ensuring mode i's safety. Finally, (13) maintains safety during mode switches.

Step 3: Solving constraints. To solve these constraints, we employ sum-of-squares optimization techniques to transform them into a hierarchy of semidefinite programming (SDP) relaxations, as detailed in [38,41,42]. In our experiments, we formulate the SDP constraints using Julia package TSSOS [58] and solve them using the Mosek solver [59]. We obtain the following solutions for Φ_1 and Φ_2:

$$\Phi_1 = -2.914 + 1.159 \cdot 10^{-1} T - 2.552 \cdot 10^{-4} v + 6.685 \cdot 10^{-5} u$$
$$\Phi_2 = 9.307 - 5.491 \cdot 10^{-1} T + 3.211 \cdot 10^{-2} v - 8.412 \cdot 10^{-3} u$$

where we scale the coefficients by a positive factor (the region $\Phi_i \le 0$ remains unchanged) and keep four significant figures.

Currently, the differential invariant generation procedure for HCSP is not fully automated, as it often requires manual template refinement. Once a suitable differential invariant is found, it can be used in HHLPy or Isabelle/HOL provers of HCSP for further verification. We would like to note that, due to the reliance on numerical solvers, it is possible that the obtained results contain small numerical errors and cannot be verified in symbolic provers. In this case, some numerical analysis techniques are required to certify the existence of a barrier certificate, see [53] for more details. This limitation also motivates us to introduce the learning-based approach, described in the coming subsection.

5.4. Verification with neural networks

In this subsection, we focus on synthesizing neural barrier certificates Φ_1 and Φ_2 for mode 1' and mode 2' respectively. The constraints for Φ_1 and Φ_2 are given as follows:

$$\forall(T, v, u).\ 165/9 \le T \le 215/9 \wedge u^2 + v^2 = 1 \implies \Phi_i(T, v, u) \le 0, \quad (14)$$

$$\forall(T, v, u).\ \mathsf{Unsafe}_i(T) \wedge u^2 + v^2 = 1 \implies \Phi_i(T, v, u) > 0, \quad (15)$$

$$\forall(T, v, u).\ 120/9 \le T \le 250/9 \wedge u^2 + v^2 = 1 \wedge \Phi_i(T, v, u) = 0 \implies \mathfrak{L}_{f_i}\Phi_i(T, v, u) \le 0, \quad (16)$$

$$\forall(T, v, u).\ \mathsf{Switch}_i(T) \wedge u^2 + v^2 = 1 \implies \Phi_{3-i}(T, v, u) \le \mu_i \Phi_i(T, v, u). \quad (17)$$

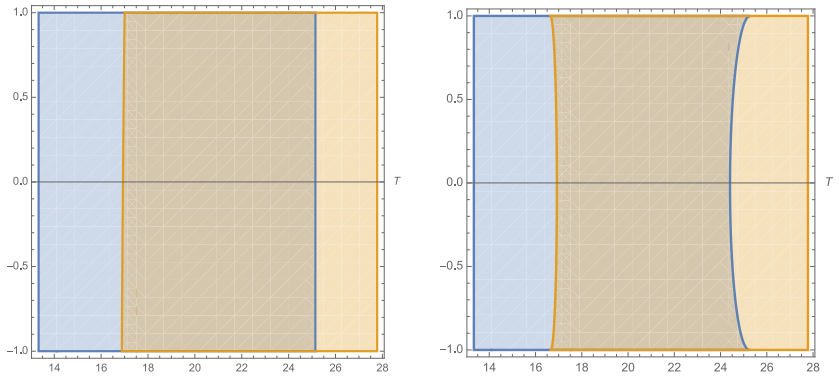
To do it, we define the loss function as follows:

$$\begin{aligned} L_\Phi = &\sum_{(T,v,u)\in D_1} \max\{\Phi_i(T, v, u) + \tau_1\} + \sum_{(T,v,u)\in D_2} \max\{-\Phi(T, v, u) + \tau_2\} \\ &+ \sum_{(T,v,u)\in D_3, \Phi_i(T,v,u)=0} \max\{L_{f_i}\Phi_i(T, v, u) + \tau_3\} \\ &+ \sum_{(T,v,u)\in D_4} \max\{\Phi_{3-i}(T, v, u) - \mu_i\Phi_i(T, v, u) + \tau_4\}, \end{aligned} \quad (18)$$

where $\tau_1, \ldots, \tau_4$ serve as offsets that are included to enhance the numerical stability during training, and $D_1, \ldots, D_4$ are the datasets sampled under the associated state constraints given in (14) to (17). The terms in (18) offer intuitive measures of the extent to which the neural networks Φ_1 and Φ_2 violate the barrier conditions. Here, we set $\mu_i = 0.1$ for $i = 1, 2$.

For the learning process, we initialize two neural networks, both with three inputs, one output, one hidden layer of 10 neurons and linear activations, to train the neural barrier candidates Φ_1 and Φ_2. We employ the Adam optimizer with a learning rate of 0.01 to update the network parameters, until L_Φ reaches zero. Finally, we obtain the barrier certificate candidates as follows:

$$\Phi_1 = 0.2864T - 4.800 \cdot 10^{-3} v + 0.2425u - 7.233$$
$$\Phi_2 = -0.1476T - 1.200 \cdot 10^{-3} v + 4.010 \cdot 10^{-2} u + 2.458.$$



(a) Numerical Method (b) Neural Method

Fig. 7. Portraits of differential invariants. blue region: $\Phi_1 \leq 0$; brown region: $\Phi_2 \leq 0$;x-axis: $T \in [120/9, 250/9]$, y-axis: $v \in [-1, 1]$. (u is replaced by $\sqrt{1-v^2}$.

For the verification process, we employ the dReal SMT solver to verify whether the negation of the barrier conditions is unsatisfiable, formalized as follows:

$$\begin{aligned}
&\exists(T, v, u).\ \big(165/9 \leq T \leq 215/9 \wedge u^2 + v^2 = 1 \implies \Phi_i(T, v, u) > 0\big)\\
&\vee \big(\mathsf{Unsafe}_i(T) \wedge u^2 + v^2 = 1 \implies \Phi_i(T, v, u) \leq 0\big)\\
&\vee \big(\ 120/9 \leq T \leq 250/9 \wedge u^2 + v^2 = 1 \vee \Phi_i(T, v, u) = 0 \implies \mathfrak{L}_{f_i}\Phi_i(T, v, u) > 0\big)\\
&\vee \big(\mathsf{Switch}_i(T) \wedge u^2 + v^2 = 1 \implies \Phi_{3-i}(T, v, u) > \mu_i\Phi_i(T, v, u)\big).
\end{aligned}$$

Through this verification process, we formally confirm the correctness of the derived invariants. The results of both methods are visualized in Fig. 7.

In general, the neural-network-based approach eliminates the numerical approximation errors inherent in purely numerical methods and provides formal soundness guarantees through SMT verification. However, this comes at the cost of increased computational resources during training and SMT verification. When applied to systems with extremely high-dimensional continuous dynamics, the scalability of SMT based verification can still be affected due to the scalability limitation of SMT solvers, and the learning process may occasionally fail to terminate.

5.5. Code generation

The MARS toolchain supports the automatic code generation from HCSP model to C, with correctness guarantee, i.e. the generated C code and the source HCSP model are proved to satisfy the approximate bisimulation relation between their reachable states with given precision allowed in ODE discretization [18]. As a result, the safety properties (that can be considered as sets of system states) proved for the HCSP model are preserved for the generated code with tolerance of given precision. No more verification needs to be re-done at the code level. For this case study, given any precision $\epsilon > 0$ allowed by the house temperature, our tool can generate the C code that is guaranteed to be approximate bisimilar with the HCSP model thus the original S/S model satisfies the given design requirement that the house temperature is always within the safe range with ϵ tolerance, i.e. (145/9 °C-ϵ, 235/9 °C+ϵ).

By using MARS, the C code for ITCS is generated, part of which is presented in Fig. 8. The whole C implementation of ITCS consists of 97 lines, that is significantly less than the code automatically generated from S/S (to be shown later). Fig. 8 presents the discretization code corresponding to the ODE part (Line 16 of Fig. 5), where h is the discretized step with respect to given precision. The ODEs of $tt, Costin, HouseTemp$ are discretized using Runge–Kutta method, implemented by a while loop: a sequence of discrete assignments on calculating the approximate values of continuous variables is performed in each loop, and when the boundary of the ODEs is reached, the loop breaks. The system's running results can be observed by executing the generated C code. Among the results in Fig. 10, we can see that the execution results of the generated C code from HCSP are almost identical to the ones of the HCSP model.

We also use S/S to generate the C code of ITCS, which amounts to 382 lines in total and is partly shown in Fig. 9. It includes the functions for updating continuous states with the specified solver for ODEs, the step function for executing the whole model, and the main function that initializes, steps in a while loop, and terminates the execution in sequence. The reason for the lengthy code from S/S includes: on one hand, the HCSP model transformed from the S/S diagram combines all the blocks of each connected part of the diagram with integrator blocks into one ODE vector, through variable substitution by hiding all outputs of intermediate non-integrator blocks, and as a result, the C code generated from the HCSP model will not include the local assignments corresponding to these blocks; On the other hand, S/S Coder needs to do settings related to ODE solver types, data logging, *etc* for each S/S instance, while in our tool, all these settings are determined, which indeed lacks feasibility for some extreme cases, but promotes efficiency for normal cases that can be handled using the general ODE solver based on Runge–Kutta method. We will give a more comparison in next section. Fig. 10 presents the comparison of execution results of the C code generated from HCSP model and S/S, the HCSP model and the original S/S model respectively. We can see that all of them are mostly consistent, except for some small fluctuations.

5.6. Other cases in Mars

We have also applied Mars to other case studies. These include: modeling and verifying the descent guidance control phase of the lunar lander and the recently launched Tianwen-I Mars lander [60,61], both of which involve non-linear dynamics. We applied a variable transformation method to convert them into polynomial dynamics, allowing for the synthesis of invariants using exponential-condition-based barrier certificates via the sum-of-squares relaxation approach. We also applied MARS to modeling and verifying safety of the Chinese high-speed Train Control System [62], which has simple continuous dynamics for modeling the train movement, but involves complex interactions between the train, the train control system and the driver through communications and parallel composition. During verification, the time and value synchronization between parallel processes are handled through interactive theorem proving. More recently, we have implemented HHLPar, an automated theorem prover that enables the verification of parallel HCSP processes with communication and parallel composition [63]. In [12], we applied MARS for the complete formal design of a cruise control system adapted from self-driving car systems. This case involved architecture modeling and analysis, as well

```
while (1) {
    double tt_ori = tt;
    double Costin_ori = Costin;
    double HouseTemp_ori = HouseTemp;
    double tt_dot1 = 1;
    double Costin_dot1 = blowercmd *
        ((-HouseTemp + 50) * (Mdot *
        c));
    double HouseTemp_dot1 =
        (blowercmd * ((-HouseTemp +
        50) * (Mdot * c)) -
        (HouseTemp - 5 / 9 * (50 +
        15 * sin(0.262 * t) - 32)) *
        (1 / Req)) * (1 / (M * c));
    tt = tt_ori + tt_dot1 * h / 2;
    Costin = Costin_ori +
        Costin_dot1 * h / 2;
    HouseTemp = HouseTemp_ori +
        HouseTemp_dot1 * h / 2;
    ... ...
    tt = tt_ori + (tt_dot1 + 2 *
        tt_dot2 + 2 * tt_dot3 +
        tt_dot4) * h / 6;
    Costin = Costin_ori +
        (Costin_dot1 + 2 *
        Costin_dot2 + 2 *
        Costin_dot3 + Costin_dot4) *
        h / 6;
    HouseTemp = HouseTemp_ori +
        (HouseTemp_dot1 + 2 *
        HouseTemp_dot2 + 2 *
        HouseTemp_dot3 +
        HouseTemp_dot4) * h / 6;
    delay(threadNumber, h);
    if (!(tt < 0.001)) {
        break;
    }
}
```

Fig. 8. Part of the C code generated from MARS.

```
void ODEUpdateStates(SolverInfo *si)
{/* the solver specified*/}
void house_step()
{ ...
  if (IsMajorTimeStep(house)) {
    ODEUpdateStates(&house->solver);
  ... ...
}
void main()
{ ...
  house_initialize();
  while ((rtmGetErrorStatus(house)
      == (NULL)) &&
        !rtmGetStopRequested(house))
          {
  house_step();
  }
  house_terminate();
  ...
}
```

Fig. 9. Part of the C code generated from Simulink.

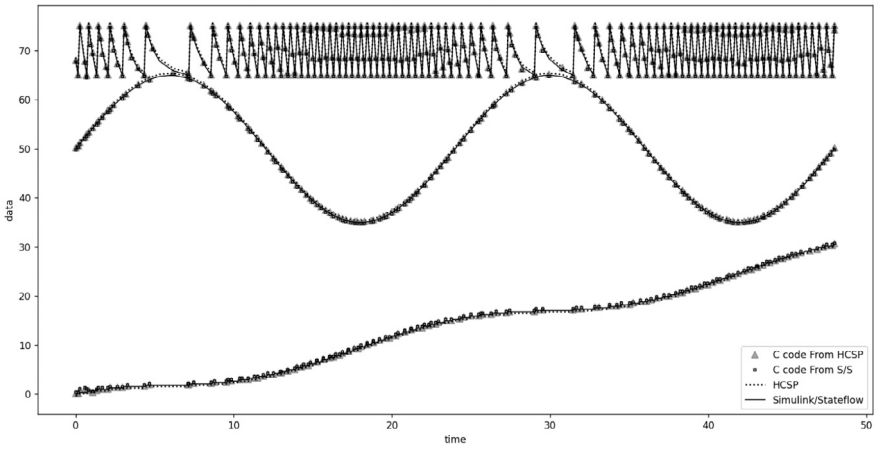


Fig. 10. Comparison of the execution results.

as Stateflow control systems, the modeling aspects that are not covered in the present case study.

6. Comparison with S/S

We compare our approach with S/S from three aspects throughout model-based development of systems: modeling and analysis, verification, code generation.

Modeling and analysis. Based on a rich set of individually simple blocks and their hierarchical composition, S/S offers a powerful graphical modeling language for building embedded systems. Especially, it is capable of modeling dynamic systems involved with continuous physical plants and complex control logics. S/S does not have an official formal semantics, and instead, system analysis and design validation within S/S are based on numerical simulation, which provides a variety of ODE solvers especially the varying-step ones for solving ODEs with both efficiency and accuracy. MARS reuses S/S for the graphical modeling of software functionality and continuous plants of systems, and to remedy S/S, it further integrates AADL for the modeling of architectures [12]. MARS also provides HCSP language for the formal modeling of hybrid systems, with formal semantics defined, and supports the transformation from S/S diagrams to HCSP formal models. The transformation covers a subset of S/S graphical syntax related to the design of hybrid systems, which is also the focus of HCSP, and the correctness of the transformation is guaranteed by defining both the formal semantics of S/S and HCSP and proving their bisimulation between each other. MARS implements a HCSP simulator that invokes Python's Scipy package to have fixed-step solvers for solving ODEs.

Verification. Formal verification is necessary in the development process of safety-critical systems. S/S has a well integrated commercial verification toolset called Simulink Design Verifier (SLDV) [64], which offers static analysis and discrete-time verification of S/S models with a high degree of automation. However, same as Simulink, the verifier does not have a formal specification language with formal semantics, and instead, it represents the property to be proved also as a S/S model. As a result, the result of SLDV cannot guarantee soundness. In [65], the authors use SLDV to formally verify an automotive Simulink controller model and detect some bugs of SLDV. MARS reduces the verification of S/S models to the verification of the transformed HCSP formal model, due to the consistency guarantee of the transformation from S/S to HCSP. As shown in Fig. 1, MARS integrates HCSP verification tools based on a sound hybrid Hoare logic for reasoning about HCSP, implemented via interactive and automated theorem proving. Furthermore, it is able to reason about continuous time behavior of HCSP models involved with ODEs based on differential invariant generation. Due to the complexity of hybrid systems, the verification of HCSP related to invariant synthesis, communications and parallel composition, needs to be done manually, but its soundness and capability of handling these behaviors are very important for designing safety-critical control systems.

Code generation. In the previous section, we have already made some comparisons between S/S and our approach for code generation of the case study. S/S has an integrated code generator, which is well developed and applied to many scalable practical embedded systems. However, the auto-generated C code may differ from the behavior of the original S/S model due to the lack of formal semantics, or potential bugs in the translation procedure from S/S to C. Thus, the code generated from S/S needs further verification for the safety. In [66], the authors perform formal verification of C code that is automatically generated from S/S controller models and find errors that are not inconsistent with the design requirement. Although these errors are found to exist as well for the original S/S model, it does not mean that the translation is correct, and in contrary, it shows the consequence of the original S/S lacking formal semantics and verification. In our tool, we implement a formally verified code generator from HCSP to C, which solves the above problem, but honestly, it is challenging for our tool to be applied for the development of large scale systems (mostly due to the verification intrinsic difficulty of complex systems).

7. Conclusion

In this paper, we show how to design an intelligent temperature control system with MARS. It consists of a S/S graphical model, a HCSP formal model transformed from the graphical model, the simulation and verification of the HCSP model, and the C code automatically generated from the verified HCSP formal model. Especially, we use both numerical and neural network based methods to synthesize differential invariants of hybrid system verification. These verification enhancements not only improve soundness of verification but also strengthen the overall design process, ensuring that the generated code inherits formally verified safety properties. Compared with the development of the system with S/S, the advantages of the design of CPS with MARS include: (i) the correctness and reliability of the generated C code; (ii) integration of modeling, simulation, verification and code generation, as well as integration of formal and informal design for CPS.

CRediT authorship contribution statement

Yihao Yin: Writing – original draft, Modelling and Simulation. **Hao Wu:** Numerical method-based verification, tool implemenation. **Wan Liu:** Neural network-based verification, tool implemenation. **Shuling Wang:** Writing – original draft, Code generation. . **Xiong Xu:** Tool implementation and support. **Wang Lin:** Neural network-based verification. **Fanjiang Xu:** Writing – review & editing. **Naijun Zhan:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been partially funded by the National Key R&D Program of China under grant No. 2022YFA1005101 and 2022YFA1005104, the National NSF of China under grant No. 62572459 and W2511064.

References

[1] MathWorks Inc., Simulink user's guide, 2013, http://www.mathworks.com/help/pdf_doc/simulink/sl_using.pdf.
[2] P. Feiler, D. Gluch, Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language, Addison-Wesley, 2012.
[3] R. Alur, C. Courcoubetis, T.A. Henzinger, P.-H. Ho, Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems, in: Hybrid Systems'92, LNCS 736, Springer, 1993, pp. 209–229.
[4] T. Henzinger, The theory of hybrid automata, in: LICS'96, IEEE Computer Society, 1996, pp. 278–292.
[5] G. Frehse, C.L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, SpaceEx: Scalable verification of hybrid systems, in: CAV'11, Springer, 2011, pp. 379–395.
[6] X. Chen, S. Sankaranarayanan, E. Ábrahám, Under-approximate flowpipes for non-linear continuous systems, in: FMCAD'14, 2014, pp. 59–66.
[7] S. Kong, S. Gao, W. Chen, E.M. Clarke, dReach: δ-reachability analysis for hybrid systems, in: TACAS 2015, Springer, 2015, pp. 200–205.
[8] A. Platzer, Differential dynamic logic for hybrid systems, J. Autom. Reason. 41 (2) (2008) 143–189.
[9] N. Zhan, B. Zhan, S. Wang, D.P. Guelev, X. Jin, A generalized hybrid Hoare logic, 2023, CoRR arXiv:2303.15020.
[10] Ansys Inc., Esterel technologies, SCADE suite, 2018, http://www.esterel-technologies.com/products/scade.
[11] B. Zhan, X. Xu, Q. Gao, Z. Ji, X. Jin, S. Wang, N. Zhan, Mars 2.0: A toolchain for modeling, analysis, verification and code generation of cyber-physical systems, 2024, arXiv arXiv:2403.03035.
[12] X. Xu, S. Wang, B. Zhan, X. Jin, J. Talpin, N. Zhan, Unified graphical co-modeling, analysis and verification of cyber-physical systems by combining AADL and Simulink/Stateflow, Theoret. Comput. Sci. 903 (2022) 1–25.
[13] X. Xu, B. Zhan, S. Wang, J.-P. Talpin, N. Zhan, A denotational semantics of Simulink with higher-order UTP, J. Log. Algebraic Methods Program. 130 (2023) 100809.
[14] S. Wang, N. Zhan, L. Zou, An improved HHL prover: an interactive theorem prover for hybrid systems, in: ICFEM'15, Springer, 2015, pp. 382–399.
[15] H. Sheng, A. Bentkamp, B. Zhan, HHLPy: Practical verification of hybrid systems using Hoare logic, in: FM'23, Springer, 2023, pp. 160–178.
[16] Y. Yin, H. Wu, S. Wang, X. Xu, F. Xu, N. Zhan, The design of intelligent temperature control system of smart house with MARS, in: SETTA, in: LNCS, vol. 15469, Springer, 2025, pp. 217–235.
[17] G. Yan, L. Jiao, S. Wang, L. Wang, N. Zhan, Automatically generating SystemC code from HCSP formal models, ACM TOSEM 29 (1) (2020) 4:1–4:39.
[18] S. Wang, Z. Ji, X. Xu, B. Zhan, Q. Gao, N. Zhan, Formally verified C code generation from hybrid communicating sequential processes, in: ICCPS'24, IEEE, 2024, pp. 123–134.
[19] X. Xu, J. Talpin, S. Wang, B. Zhan, N. Zhan, Semantics foundation for cyber-physical systems using higher-order UTP, ACM Trans. Softw. Eng. Methodol. 32 (1) (2023) 9:1–9:48.
[20] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, Q. Zhu, A next-generation design framework for platform-based design, in: DVCon 2007, Citeseer, 2007.
[21] C. Ptolemaeus (Ed.), System Design, Modeling, and Simulation Using Ptolemy II, Ptolemy.org, 2014, URL http://ptolemy.org/books/Systems.
[22] A. Junghanns, C. Gomes, C. Schulze, K. Schuch, P. R., M. Blaesken, I. Zacharias, A. Pillekeit, K. Wernersson, T. Sommer, C. Bertsch, T. Blochwitz, M. Najafi, The functional mock-up interface 3.0 - New features enabling new applications, in: Proceedings of 14th Modelica Conference 2021, 2021.
[23] Q. Sun, W. Zhang, C. Wang, Z. Liu, A contract-based semantics and refinement for hybrid Simulink block diagrams, J. Syst. Archit. 143 (2023) 102963.
[24] W. Zhang, Q. Sun, C. Wang, Z. Liu, Towards correctness proof for hybrid Simulink block diagrams, J. Syst. Archit. 141 (2023) 102922.
[25] H. Yu, Y. Ma, T. Gautier, L. Besnard, P. Le Guernic, J.-P. Talpin, Polychronous modeling, analysis, verification and simulation for timed software architectures, J. Syst. Archit. 59 (10, Part D) (2013) 1157–1170.

[26] F.X. Dormoy, SCADE 6: A model based solution for safety critical software development, in: ERTS 2008, Esterel Technologies, 2008.
[27] N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud, The synchronous data flow programming language LUSTRE, Proc. IEEE 79 (9) (1991) 1305–1320.
[28] T. Bourke, M. Pouzet, Zélus: a synchronous language with ODEs, in: HSCC 2013, ACM, 2013, pp. 113–118, http://dx.doi.org/10.1145/2461328.2461348.
[29] A. Benveniste, T. Bourke, B. Caillaud, M. Pouzet, Non-standard semantics of hybrid systems modelers, J. Comput. System Sci. 78 (3) (2012) 877–910, http://dx.doi.org/10.1016/J.JCSS.2011.08.009.
[30] T. Bourke, J.-L. Colaço, B. Pagano, C. Pasteur, M. Pouzet, A synchronous-based code generator for explicit hybrid systems languages, in: B. Franke (Ed.), CC 2015, in: LNCS, vol. 9031, Springer, 2015, pp. 69–88.
[31] R. Banach, M.J. Butler, S. Qin, N. Verma, H. Zhu, Core hybrid Event-B I: single hybrid Event-B machines, Sci. Comput. Program. 105 (2015) 92–123.
[32] R. Banach, M.J. Butler, S. Qin, H. Zhu, Core hybrid Event-B II: multiple cooperating hybrid Event-B machines, Sci. Comput. Program. 139 (2017) 1–35.
[33] R. Banach, Core hybrid Event-B III: fundamentals of a reasoning framework, Sci. Comput. Program. 231 (2024) 103002.
[34] A. Platzer, Logical Foundations of Cyber-Physical Systems, Springer, 2018, http://dx.doi.org/10.1007/978-3-319-63588-0.
[35] N. Fulton, S. Mitsch, J. Quesel, M. Völp, A. Platzer, Keymaera X: an axiomatic tactical theorem prover for hybrid systems, in: CADE 2015, in: LNCS, vol. 9195, Springer, 2015, pp. 527–538, http://dx.doi.org/10.1007/978-3-319-21401-6_36.
[36] S. Prajna, A. Jadbabaie, Safety verification of hybrid systems using barrier certificates, in: Computation and Control, HSCC, Springer, Berlin, Heidelberg, 2004, pp. 477–492.
[37] S. Prajna, A. Jadbabaie, G.J. Pappas, A framework for worst-case and stochastic safety verification using barrier certificates, IEEE Trans. Automat. Control 52 (8) (2007) 1415–1429.
[38] H. Kong, F. He, X. Song, W.N.N. Hung, M. Gu, Exponential-condition-based barrier certificate generation for safety verification of hybrid systems, in: CAV'13, Springer, 2013, pp. 242–257.
[39] X. Zeng, L. Wang, Z. Yang, X. Chen, L. Wang, Darbouxtype barrier certificates for safety verification of nonlinear hybrid systems, in: Proceedings of the 13th International Conference on Embedded Software, ACM, 2016, pp. 1–10.
[40] A. Sogokon, K. Ghorbal, Y.K. Tan, A. Platzer, Vector barrier certificates and comparison systems, in: FM, in: LNCS, vol. 10951, Springer, 2018, pp. 418–437.
[41] Q. Wang, M. Chen, B. Xue, N. Zhan, J. Katoen, Encoding inductive invariants as barrier certificates: Synthesis via difference-of-convex programming, Inform. and Comput. 289 (Part) (2022) 104965.
[42] H. Wu, S. Feng, T. Gan, J. Wang, B. Xia, N. Zhan, On completeness of SDP-based barrier certificate synthesis over unbounded domains, in: FM'24, in: LNCS, vol. 14934, Springer, 2024, pp. 248–266.
[43] H. Zhao, X. Zeng, T. Chen, Z. Liu, Synthesizing barrier certificates using neural networks, in: Proceedings of the 23rd ACM International Conference on Hybrid Systems: Computation and Control, HSCC, ACM, 2020, pp. 25:1–25:11.
[44] H. Zhao, X. Zeng, T. Chen, Z. Liu, J. Woodcock, Learning safe neural network controllers with barrier certificates, Form. Asp. Comput. 33 (3) (2021) 437–455.
[45] A. Abate, D. Ahmed, A. Edwards, M. Giacobbe, A. Peruffo, FOSSIL: a software tool for the formal synthesis of lyapunov functions and barrier certificates using neural networks, in: Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control, Association for Computing Machinery, 2021, pp. 1–11.
[46] A. Peruffo, D. Ahmed, A. Abate, Automated and formal synthesis of neural barrier certificates for dynamical models, in: TACAS (1), in: Lecture Notes in Computer Science, vol. 12651, Springer, 2021, pp. 370–388.
[47] A. Edwards, A. Peruffo, A. Abate, Fossil 2.0: Formal certificate synthesis for the verification and control of dynamical models, in: HSCC, ACM, 2024, pp. 26:1–26:10.
[48] H. Zhao, B. Liu, L. Dehbi, H. Xie, Z. Yang, H. Qian, Polynomial neural barrier certificate synthesis of hybrid systems via counterexample guidance, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 43 (2024) 3756–3767.
[49] J. He, From CSP to hybrid systems, in: A Classical Mind, Prentice Hall International (UK) Ltd., 1994, pp. 171–189.
[50] S. Prajna, A. Jadbabaie, Safety verification of hybrid systems using barrier certificates, in: HSCC'04, Springer, 2004, pp. 477–492.
[51] A. Taly, A. Tiwari, Deductive verification of continuous dynamical systems, in: FSTTCS, in: LIPIcs, vol. 4, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009, pp. 383–394.
[52] L. Dai, T. Gan, B. Xia, N. Zhan, Barrier certificates revisited, J. Symbolic Comput. 80 (2017) 62–86.
[53] P. Roux, Y.-L. Voronin, S. Sankaranarayanan, Validating numerical semidefinite programming solvers for polynomial invariants, Form. Methods Syst. Des. 53 (2) (2018) 286–312.
[54] I. Dragomir, V. Preoteasa, S. Tripakis, Compositional semantics and analysis of hierarchical block diagrams, in: SPIN 2016, in: LNCS, vol. 9641, Springer, 2016, pp. 38–56.
[55] J. Liu, N. Zhan, H. Zhao, Computing semi-algebraic invariants for polynomial dynamical systems, in: EMSOFT'11, 2011, pp. 97–106.
[56] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, L. Zou, A calculus for hybrid CSP, in: APLAS'10, Springer, 2010, pp. 1–15.
[57] J. Liu, N. Zhan, H. Zhao, L. Zou, Abstraction of elementary hybrid systems by variable transformation, in: FM'15, Springer, 2015, pp. 360–377.
[58] J. Wang, V. Magron, J. Lasserre, TSSOS: a moment-SOS hierarchy that exploits term sparsity, SIAM J. Optim. 31 (1) (2021) 30–58.
[59] MOSEK ApS, MOSEK optimizer API for Julia. Version 10.1.13, 2019, URL https://docs.mosek.com/latest/juliaapi/index.html.
[60] H. Zhao, M. Yang, N. Zhan, B. Gu, L. Zou, Y. Chen, Formal verification of a descent guidance control program of a lunar lander, in: FM 2014, in: LNCS, vol. 8442, Springer, 2014, pp. 733–748.
[61] B. Zhan, B. Gu, X. Xu, X. Jin, S. Wang, B. Xue, X. Li, Y. Chen, M. Yang, N. Zhan, Brief industry paper: Modeling and verification of descent guidance control of mars lander, in: 2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium, RTAS, 2021, pp. 457–460.
[62] L. Zou, J. Lv, S. Wang, N. Zhan, T. Tang, L. Yuan, Y. Liu, Verifying Chinese train control system under a combined scenario by theorem proving, in: VSTTE'13, in: LNCS, vol. 8164, 2013, pp. 262–280.
[63] X. Jin, B. Zhan, S. Wang, N. Zhan, HHLPar: Automated theorem prover for parallel hybrid communicating sequential processes, 2024, arXiv:2407.08936.
[64] MathWorks Inc., Simulink Design Verifier – User's guide, https://de.mathworks.com/help/pdf_doc/sldv/sldv_ug.pdf,
[65] J. Nellen, T. Rambow, M.T.B. Waez, E. Ábrahám, J.-P. Katoen, Formal verification of automotive Simulink controller models: Empirical technical challenges, evaluation and recommendations, in: FM'18, Springer, 2018, pp. 382–398.
[66] P. Berger, J.-P. Katoen, E. Ábrahám, M.T.B. Waez, T. Rambow, Verifying auto-generated C code from Simulink, in: FM'18, Springer, 2018, pp. 312–328.

Yihao Yin is a Master's student in Artificial Intelligence at the Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences (UCAS). His research focuses on formal verification and artificial intelligence. He received his Bachelor's degree in Computer Science from Shandong Normal University.

Hao Wu is a final-year Ph.D. student at the Institute of Software, University of Chinese Academy of Sciences (UCAS), under the supervision of Prof. Naijun Zhan. His work centers on addressing challenges in formal verification of cyber–physical systems by combining techniques from symbolic computation and numerical optimization. He completed his undergraduate studies in Computer Science, also at UCAS.

Wan Liu is currently pursuing the M.E. degree in software engineering at Zhejiang Sci-Tech University, Hangzhou, China. He received his B.E. degree in software engineering from Nanyang Institute of Technology, Nanyang, China, in 2023. His research interests include reinforcement learning and formal verification.

Shuling Wang is currently an Associate Researcher at the Institute of Software, Chinese Academy of Sciences (ISCAS). She received her Ph.D. from Peking University and subsequently conducted postdoctoral research at the United Nations University International Institute for Software Technology and ISCAS. Her research interests include the formal modeling and verification of cyber–physical systems and embedded systems, program verification, and interactive theorem proving.

Xiong Xu is a research assistant in Institute of Software Chinese Academy of Sciences (ISCAS). He got his bachelor degree from Tianjin Polytechnic University (Tiangong University), and his Ph.D. from ISCAS. Prior to join ISCAS as a research assistant, he completed a postdoctoral fellowship at ISCAS in collaboration with Inria. His research interests cover formal design of cyber–physical systems and model-based design.

Wang Lin is currently a Professor with the School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou, China. He received the Ph.D. degree in Computer Science and Technology from East China Normal University, Shanghai, China, in 2013. His current research interests include design and analysis of hybrid and cyber–physical systems, and software verification.

Fanjiang Xu is a research professor at the Institute of Software, Chinese Academy of Sciences. His primary research focuses on intelligent information processing, including the concept of development of software-defined satellite technology, and the intelligent optics research that integrates physical models with data-driven approaches.

Naijun Zhan is a Boya distinguished professor in the School of Computer Science of Peking University. He got his bachelor degree and master degree both from Nanjing University, and his Ph.D. from Institute of Software Chinese Academy of Sciences (ISCAS). Prior to join Peking University, he worked at the Faculty of Mathematics and Informatics, Mannheim University, Germany as a research fellow, and afterwards worked at ISCAS as an associate professor, a full professor, and a distinguished professor. His research interests cover formal design of real-time, embedded and hybrid systems, program verification.