Formal Modelling, Analysis and Verification of Hybrid Systems

Naijun Zhan, Shuling Wang, and Hengjun Zhao

State Key Lab. of Comput. Sci., Inst. of Software, Chinese Academy of Sciences

Abstract. Hybrid systems is a mathematical model of embedded systems, and has been widely used in the design of complex embedded systems. In this chapter, we will introduce our systematic approach to formal modelling, analysis and verification of hybrid systems. In our framework, a hybrid system is modelled using Hybird CSP (HCSP), and specified and reasoned about by Hybrid Hoare Logic (HHL), which is an extension of Hoare logic to hybrid systems. For deductive verification of hybrid systems, a complete approach to generating polynomial invariants for polynomial hybrid systems is proposed; meanwhile, a theorem prover for HHL that can provide tool support for the verification has been implemented. We give some case studies from real-time world, for instance, Chinese High-Speed Train Control System at Level 3 (CTCS-3). In addition, based on our invariant generation approach, we consider how to synthesize a switching logic for a considered hybrid system by reduction to constraint solving, to meet a given safety, liveness, optimality requirement, or any of their combinations. We also discuss other issues of hybrid systems, e.g., stability analysis.

Keywords: Hybrid systems, Hybrid CSP, Hybrid Hoare Logic, Invariant, Theorem proving.

1 Introduction

Our modern life increasingly depends on embedded systems. How to develop correct complex embedded systems is a grand challenge for computer science and control theory. The model-based method is thought to be an effective method to designing complex embedded systems. Using this approach at the very beginning, an abstract model of the system to be developed with precise mathematical semantics is defined. Extensive analysis and verification on the abstract model are then committed so that errors can be identified and corrected at the very early stage. Then, a higher-level abstract model is refined to a lower-level abstract model, even to source code, step by step, using model-transformation techniques.

Hybrid systems, combining formal models for discrete *reactive systems* and continuous models for *dynamical systems* [1,59], is a mathematical model of embedded systems. There are hugely numerous work that have been done related to hybrid systems. Please refer to [4,42] for a survey. Modeling discrete components by finite automata, and attaching state-dependent ordinary differential equations

Z. Liu, J. Woodcock, and H. Zhu (Eds.): Theories of Programming, LNCS 8050, pp. 207–281, 2013.

to the discrete states in order to capture the impact of the discrete component on the continuous environment, yields hybrid automata [1], which are by far the most widely used model of hybrid systems in academia. Hybrid automata are, however, analogs of state machine, with little support for structured description, and consequently, a number of formalisms have been proposed to facilitate modular descriptions of complex systems. These include modeling environments such as SHIFT [28] and PTOLEMY [31] for hierarchical specification of hybrid behavior; models such as hybrid I/O automata [58], hybrid modules [2], and CHARON [9], for compositional treatment of concurrent hybrid behavior; Hybrid CSP (HCSP) [37,98] for process algebra based specification and verification of hybrid behavior; and differential dynamic logic [65,66] and hybrid Hoare logic (HHL) [55,86] for logic-based specification and compositional analysis of hybrid behavior. Industrial variants include the Simulink/Stateflow environment, which does, however, lack a uniquely defined —depending on the use case, there are semantical variations— and comprehensive formal semantics.

With most hybrid system models being rooted in automata-based models, their pertinent verification techniques have accordingly adopted the automaton-based approach to verification in the past, mainly being based on directly computing exact representations or safe approximations of reachable state sets. For example, based on model-checking [22,71], the reachability problems of some simple hybrid systems, like timed automata [8], multirate automata [1], initialized rectangular automata [70,41], and so on, have been solved; based on the decision procedure of Tarski algebra [83], in [53] methods for computing reachable sets for three classes of special linear hybrid systems were investigated. Due to infiniteness of the underlying state spaces, symbolic representations, often paired with safe approximation within a computationally reasonably efficient symbolic representation system, and abstraction techniques are generally applied in reachable set computation. For example, the tool HYTECH [3] was the first model checker to implement exact symbolic reachability analysis of linear hybrid automata[1] by using polyhedra-based technique; while the tools CHECKMATE [20] and **d/dt** [11] compute over-approximations of reachable sets of linear hybrid systems[1] using polyhedral representations; related techniques use lazy theorem proving for analyzing bounded reachability problems of linear [12] or non-linear [34] hybrid automata. Furthermore, discretization of continuous dynamics based on gridding or predicate abstraction has been extended and adopted for hybrid systems [40,7,21,73,25].

To deal with more complicated systems, recently, a deductive method for the verification of hybrid systems has been established and successfully applied in practice [65,66]. This method can be seen as a generalization of the so-called Floyd-Hoare-Naur inductive assertion method [32,43,62]. The inductive assertion

[1] With hybrid systems being an interdisciplinary domain bridging control theory and computer science, terminology often is subtle due to different roots of naming conventions. A *linear hybrid automaton* is a system featuring (piecewise) constant differential inclusions while a *linear hybrid system* features linear or often even affine differential equations.

method is thought to be the dominant method for the verification of sequential programs. To generalize the inductive method to hybrid systems, a modeling language with compositionality for hybrid systems and a Hoare-style logic for the language with the ability of dealing with continuous dynamics are prerequisites. For example, a differential-algebraic dynamic logic for hybrid programs [64] was invented by extending dynamic logic with continuous statements. Recently, we [55] had another effort by extending Hoare logic to hybrid systems modeled by HCSP [37,98] for the same purpose.

The concept of *invariant* is at the core of deductive methods. An *invariant* of a hybrid system is a property ϕ that holds in all the reachable states of the system. An *inductive invariant* of a hybrid system is an assertion ϕ that holds at the initial states of the system, and preserved by all discrete and continuous dynamics. In fact, any inductive invariant is also an invariant, but the inverse is not true in general. The problem of (inductive) invariant generation has received wide attention in the analysis and verification of programs [13,23,76,48] and hybrid systems [75,69,65,81,55]. Many properties of hybrid systems like safety, stability, liveness etc., can be characterized and inferred via invariants without solving differential equations, while differential equations have to be exactly solved or approximated in the methods based on directly computing reachable sets.

The key issue in generating inductive invariants of a hybrid system is to deal with continuous dynamics, i.e. to generate so-called *continuous invariant* (CI) of the continuous dynamics at each mode of the system. A method based on constraint solving was proposed in [75] for generating CIs containing a single polynomial equation. This method was generalized in [74] to construct CIs containing infinitely many polynomial equations, i.e. the so-called *invariant ideal*. The basic idea of these methods is to reduce the CI generation problem to a constraint solving problem using techniques from the theory of ideals over polynomial rings. For the polynomial inequality case, it was considered in [69,67] how to generate CIs containing one polynomial inequality. The basic idea of their method is to utilize a certain function, called a *barrier certificate*, to enclose the invariant. With some stronger constraints, generation of more general CIs was considered in [65], wherein the CIs are Boolean combinations of polynomial equations and inequalities. By restricting the invariant sets to have smooth boundaries, a sound but incomplete method for constructing invariants involving non-strict polynomial inequalities was proposed in [81,80]. While in [56], we presented a relatively complete method for generating *semi-algebraic invariants* (SAIs) for polynomial continuous dynamical systems by employing higher-order Lie derivatives and the theory of polynomial ideal.

As a complementation of verification, synthesis focuses on designing a controller that controls the underlying subsystems so that the whole system is guaranteed to satisfy the given requirement, that may be safety, liveness (e.g. reachability to a given set of states), optimality criterion, or a desired combination of them. Numerous work have been done on controller synthesis for safety and/or reachability requirements. For example, in [10,85], a general framework

relying on *backward reachable set* computation and *fixed point iteration* was proposed, for synthesizing controllers for hybrid automata to meet a given safety requirement; while in [79], a symbolic approach based on templates and constraint solving to the same problem was proposed, and in [82], the symbolic approach is extended to meet both safety and reachability requirements. Compared with controller synthesis for safety, the optimal controller synthesis problem is more involved, also quite important in the design of hybrid systems. In the literature, few work has been done on the problem. Larsen et al proposed an approach based on energy automata and model-checking [18], while Jha, Seshia and Tiwari gave a solution to the problem using unconstrained numerical optimization and machine learning [44]. In [94], we proposed a "hybrid" approach for synthesizing optimal controllers of hybrid systems subject to safety requirements. The basic idea is as follows. Firstly, we reduce optimal controller synthesis subject to safety requirements to quantifier elimination (QE for short). Secondly, in order to make our approach scalable, we discuss how to combine QE with numerical computation, but at the same time, keep arising errors due to discretization manageable and within bounds. A major advantage of our approach is not only that it avoids errors due to numerical computation, but it also gives a better optimal controller.

All the aforementioned verification or synthesis approaches aim at showing or avoiding unreachability of undesirable states, i.e. total absence of undesirable behavior. In realistic applications, this often is an overly ambitious goal, being economically unattainable or even technically impossible to achieve due to uncontrollable environmental influences, unavoidable manufacturing tolerances, component breakdown, etc. Therefore, the existing, qualitative safety analysis methods for hybrid systems have to be complemented by quantitative methods, quantifying the likelihood of residual error or related performance figures (MTBF, MTTF, etc.) in systems subject to uncertain, stochastic behavior (both in the embedded system and its environment) as well as noise. It is therefore necessary to address such stochastic issues in the model of embedded systems, i.e. hybrid systems, by adding models of stochastic behavior to the modeling language and corresponding analysis techniques to the verification. Some first attempts on introducing probability and stochasticity in hybrid models have been pursued, e.g. [45,46,34,33], yet expressiveness of the models and scalability of the analysis tools remain pressing issues.

1.1 Synopsis

In Sec. 2, some basic notions, notations and mathematical foundations that will be used later are provided.

In Sec. 3, we introduce our approach for generating semi-algebraic invariants for polynomial continuous dynamical systems and its extension to hybrid systems. This is the first relatively complete approach for discovering polynomial invariants for these systems in the literature. This section is mainly based on our previous joint work with Liu reported in [56,54].

In Sec. 4, we first introduce how to synthesize switching controllers for hybrid systems subject to safety requirement based on continuous invariant generation reported in Sec. 3. To improve the efficiency, qualitative analysis [47] is adopted. This part is based on our recent joint work with Kapur [49]. Then, we consider optimal controller synthesis problems of hybrid systems by reducing to constraint solving, which is based on a joint work with Kapur and Larsen [94].

In Sec. 5, we introduce Hybrid CSP due to He, Zhou et al [37,98], which is an extension of CSP for hybrid systems. Here, we define a formal operational semantics for HCSP, which has been implemented in the HHL prover introduced later.

In Sec. 6, we introduce a specification logic for hybrid systems, called Hybrid Hoare Logic, which is achieved by combining Hoare logic with Duration Calculus (DC) [97,96]. The presentation is based on our previous work [55].

In Sec. 7, we introduce a proof assistant of HHL in Isabelle/HOL, which is based on our recent joint work with Zou et al [99].

In Sec. 8, we present a case study from a real world on Chinese high-speed train control system by using HCSP and HHL and the tool, based on the recent joint work [99].

In Sec. 9, we discuss other issues related to hybrid systems, mainly focusing on stability analysis of continuous dynamical systems based on a joint work with Liu [57].

Finally, we conclude this tutorial by Sec. 10 with some discussions of future work.

2 Preliminaries

In this section, we define the basic notions and notations that will be used in the rest of this tutorial. We also give an elementary description of several relevant mathematical theories fundamental to the understanding of this tutorial. For a comprehensive introduction of these theories the readers may refer to the cited literatures.

Throughout this tutorial, we use $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ to denote the set of *natural*, *rational* and *real* numbers respectively. Given a set A, the Cartesian product of its n duplicates is denoted by A^n; for instance, $\mathbb{R}^n$ stands for the n-dimensional Euclidean space. A vector element $(a_1, a_2, \ldots, a_n) \in A^n$ is usually abbreviated by a boldface letter $\mathbf{a}$ when its dimension is clear from the context.

2.1 Continuous Dynamical Systems

We introduce some basic theories of continuous dynamical systems here. For details please refer to [50,84].

Typically, a continuous dynamical systems (CDS for short) is modeled by first-order autonomous ordinary differential equations

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) \ , \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a vector function, called a *vector field* in $\mathbb{R}^n$.

If $\mathbf{f}$ in (1) satisfies the *local Lipschitz condition*, then given $\mathbf{x}_0 \in \mathbb{R}^n$, there exists a unique *differentiable* vector function $\mathbf{x}(\mathbf{x}_0;t) : (a,b) \to \mathbb{R}^n$, where (a,b) is an open interval containing 0, such that $\mathbf{x}(\mathbf{x}_0;0) = \mathbf{x}_0$ and the derivative of $\mathbf{x}(\mathbf{x}_0;t)$ w.r.t. t satisfies

$$\forall t \in (a,b).\ \frac{\mathrm{d}\mathbf{x}(\mathbf{x}_0;t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(\mathbf{x}_0;t)) \ .$$

Such $\mathbf{x}(\mathbf{x}_0;t)$ is called the solution to (1) with initial value $\mathbf{x}_0$.

If for any $\mathbf{x}_0 \in \mathbb{R}^n$, there is a solution $\mathbf{x}(\mathbf{x}_0;t)$ to (1) that exists for all time $t \in \mathbb{R}$, then the vector field $\mathbf{f}$ is called *complete*. A *globally Lipschitz continuous* vector field $\mathbf{f}$ guarantees the existence, uniqueness and completeness of solutions to (1).

If $\mathbf{f}$ is *analytic* at $\mathbf{x}_0 \in \mathbb{R}^n$, i.e. $\mathbf{f}$ is given by a convergent power series in a neighborhood of $\mathbf{x}_0$, then there exists a unique *analytic* solution $\mathbf{x}(\mathbf{x}_0;t)$ to (1) defined in a neighborhood of 0.

According to the evolution direction w.r.t. time, the solutions to (1) induce two sorts of geometrical curves as follows.

Definition 1. *Suppose $\mathbf{x}(\mathbf{x}_0;t)$ is the solution to (1) with initial value $\mathbf{x}_0$. Then*

- *$\mathbf{x}(\mathbf{x}_0;t)$ with $t \geq 0$ is called the* **trajectory** *of $\mathbf{f}$ starting from $\mathbf{x}_0$;*
- *$\mathbf{x}(\mathbf{x}_0;-t)$ with $t \geq 0$ is called the* **inverse trajectory** *of $\mathbf{f}$ starting from $\mathbf{x}_0$, where $\mathbf{x}(\mathbf{x}_0;-t)$ is obtained by substituting $-t$ for t in $\mathbf{x}(\mathbf{x}_0;t)$.*

When $\mathbf{x}_0$ is clear from the context, we write $\mathbf{x}(\mathbf{x}_0;t)$ and $\mathbf{x}(\mathbf{x}_0;-t)$ as $\mathbf{x}(t)$ and $\mathbf{x}(-t)$ for brevity.

The notion of *Lie derivative* is important for the study of CDSs and plays a central role in several subsequent sections of this tutorial. Let $\sigma(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ be a scalar function and $\mathbf{f}$ be a vector field in $\mathbb{R}^n$. Suppose both σ and $\mathbf{f}$ are *smooth functions*, i.e. differentiable in $\mathbf{x}$ at any order $k \in \mathbb{N}$. Then we can inductively define the *Lie derivatives* of σ along $\mathbf{f}$, i.e. $L^k_{\mathbf{f}}\sigma : \mathbb{R}^n \to \mathbb{R}$ for $k \in \mathbb{N}$, as follows:

- $L^0_{\mathbf{f}}\sigma(\mathbf{x}) = \sigma(\mathbf{x})$,
- $L^k_{\mathbf{f}}\sigma(\mathbf{x}) = \Big(\nabla L^{k-1}_{\mathbf{f}}\sigma(\mathbf{x}), \mathbf{f}(\mathbf{x})\Big)$, for $k > 0$,

where ∇ stands for the *gradient* operator, i.e. for any differentiable function $\varrho(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$,

$$\nabla\varrho(\mathbf{x}) \widehat{=} \left(\frac{\partial\varrho(\mathbf{x})}{\partial x_1}, \frac{\partial\varrho(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial\varrho(\mathbf{x})}{\partial x_n}\right) \ ,$$

and $(\cdot,\cdot)$ is the *inner product* of two vectors, i.e. $(\mathbf{a},\mathbf{b}) = \sum_{i=1}^n a_i b_i$ for $\mathbf{a} = (a_1,\ldots,a_n)$ and $\mathbf{b} = (b_1,\ldots,b_n)$.

2.2 Hybrid Systems

Hybrid systems are those systems that exhibit both continuous evolutions and discrete transitions between different modes. A widely adopted model of hybrid systems is *hybrid automata* [5,63,39], the extension of finite automata with

continuous components. In this tutorial, the discussion of invariant generation and controller synthesis of hybrid systems will be cast in the setting of hybrid automata. A separate section (Sec. 5) of this tutorial will be devoted to a compositional language named HCSP, which is more suitable for modelling complex hybrid systems. The formal definition of hybrid automata in the literature differs slightly from each other. Here the presentation is based on [85] and [75].

Definition 2 (Hybrid Automaton). *A hybrid automaton (HA) is a system* $\mathcal{H} \widehat{=} (Q, X, f, D, E, G, R, \Xi)$*, where*

- $Q = \{q_1, \ldots, q_m\}$ *is a finite set of discrete states (or modes);*
- $X = \{x_1, \ldots, x_n\}$ *is a finite set of continuous state variables, with* $\mathbf{x} = (x_1, \ldots, x_n)$ *ranging over* $\mathbb{R}^n$*;*
- $f : Q \to (\mathbb{R}^n \to \mathbb{R}^n)$ *assigns to each mode* $q \in Q$ *a locally Lipschitz continuous vector field* $\mathbf{f}_q$*;*
- D *assigns to each mode* $q \in Q$ *a mode domain* $D_q \subseteq \mathbb{R}^n$*;*
- $E \subseteq Q \times Q$ *is a finite set of discrete transitions;*
- G *assigns to each transition* $e \in E$ *a switching guard* $G_e \subseteq \mathbb{R}^n$*;*
- R *assigns to each transition* $e \in E$ *a reset function* R_e*:* $\mathbb{R}^n \to \mathbb{R}^n$*;*
- Ξ *assigns to each* $q \in Q$ *a set of initial states* $\Xi_q \subseteq \mathbb{R}^n$*.*

The state space of $\mathcal{H}$ is $\mathbb{H} \widehat{=} Q \times \mathbb{R}^n$, the domain of $\mathcal{H}$ is $D_{\mathcal{H}} \widehat{=} \bigcup_{q \in Q}(\{q\} \times D_q)$, and the set of all initial states is denoted by $\Xi_{\mathcal{H}} \widehat{=} \bigcup_{q \in Q}(\{q\} \times \Xi_q)$. The semantics of $\mathcal{H}$ can be characterized by the set of *hybrid trajectories* accepted by $\mathcal{H}$ or the *reachable set* of $\mathcal{H}$.

Definition 3 (Hybrid Time Set). *A hybrid time set is a sequence of intervals* $\tau = \{I_i\}_{i=0}^{N}$ *(*N *can be* ∞*) such that:*

- $I_i = [\tau_i, \tau_i']$ *with* $\tau_i \leq \tau_i' = \tau_{i+1}$ *for all* $i < N$*;*
- *if* $N < \infty$*, then* $I_N = [\tau_N, \tau_N'\rangle$ *is a right-closed or right-open nonempty interval (*τ_N' *may be* ∞*);*
- $\tau_0 = 0$.

Given a hybrid time set, let $\langle\tau\rangle = N$ and $\|\tau\| = \sum_{i=0}^{N}(\tau_i' - \tau_i)$. Then τ is called *infinite* if $\langle\tau\rangle = \infty$ or $\|\tau\| = \infty$, and *zeno* if $\langle\tau\rangle = \infty$ but $\|\tau\| < \infty$.

Definition 4 (Hybrid Trajectory). *A hybrid trajectory of* $\mathcal{H}$ *starting from an initial point* $(q_0, \mathbf{x}_0) \in \Xi_{\mathcal{H}}$ *is a triple* $\omega = (\tau, \alpha, \beta)$*, where* $\tau = \{I_i\}_{i=0}^{N}$ *is a hybrid time set, and* $\alpha = \{\alpha_i : I_i \to Q\}_{i=0}^{N}$ *and* $\beta = \{\beta_i : I_i \to \mathbb{R}^n\}_{i=0}^{N}$ *are two sequences of functions satisfying:*

1. *Initial condition:* $\alpha_0[0] = q_0$ *and* $\beta_0[0] = \mathbf{x}_0$*;*
2. *Discrete transition: for all* $i < \langle\tau\rangle$*,* $e = \big(\alpha_i(\tau_i'), \alpha_{i+1}(\tau_{i+1})\big) \in E$*,* $\beta_i(\tau_i') \in G_e$ *and* $\beta_{i+1}(\tau_{i+1}) = R_e(\beta_i(\tau_i'))$*;*
3. *Continuous evolution: for all* $i \leq \langle\tau\rangle$ *with* $\tau_i < \tau_i'$*, if* $q = \alpha_i(\tau_i)$*, then*
   (1) for all $t \in I_i$*,* $\alpha_i(t) = q$*,*
   (2) $\beta_i(t)$ *is the solution to the differential equation* $\dot{\mathbf{x}} = \mathbf{f}_q(\mathbf{x})$ *over* I_i *with initial value* $\beta_i(\tau_i)$*, and*
   (3) for all $t \in [\tau_i, \tau_i')$*,* $\beta_i(t) \in D_q$.

The set of trajectories starting from an initial state $(q_0, \mathbf{x}_0)$ of $\mathcal{H}$ is denoted by $\mathcal{T}r(\mathcal{H})(q_0, \mathbf{x}_0)$, and the set of all trajectories of $\mathcal{H}$ by $\mathcal{T}r(\mathcal{H})$.

A hybrid trajectory $\omega = (\tau, \alpha, \beta)$ is called *infinite* or *zeno*, if τ is infinite or zeno respectively. An HA $\mathcal{H}$ is called *non-blocking* if for any $(q_0, \mathbf{x}_0) \in \Xi_{\mathcal{H}}$ there exists an infinite trajectory in $\mathcal{T}r(\mathcal{H})(q_0, \mathbf{x}_0)$, and *blocking* otherwise; $\mathcal{H}$ is called *non-zeno* if there exists no zeno trajectory in $\mathcal{T}r(\mathcal{H})$, and *zeno* otherwise.

Another way to interpret hybrid automata is using reachability relation.

Definition 5 (Reachable Set). *Given an HA $\mathcal{H}$, the reachable set of $\mathcal{H}$, denoted by $\mathcal{R}_{\mathcal{H}}$, consists of those $(q, \mathbf{x})$ for which there exists a finite sequence*

$$(q_0, \mathbf{x}_0), (q_1, \mathbf{x}_1), \ldots, (q_l, \mathbf{x}_l)$$

such that $(q_0, \mathbf{x}_0) \in \Xi_{\mathcal{H}}$, $(q_l, \mathbf{x}_l) = (q, \mathbf{x})$, and for any $0 \leq i \leq l-1$, one of the following two conditions holds:

- *(Discrete Jump): $e = (q_i, q_{i+1}) \in E$, $\mathbf{x}_i \in G_e$ and $\mathbf{x}_{i+1} = R_e(\mathbf{x}_i)$; or*
- *(Continuous Evolution): $q_i = q_{i+1}$, and there exists a $\delta \geq 0$ s.t. the solution $\mathbf{x}(\mathbf{x}_i; t)$ to $\dot{\mathbf{x}} = \mathbf{f}_{q_i}$ satisfies*
  - *$\mathbf{x}(\mathbf{x}_i; t) \in D_{q_i}$ for all $t \in [0, \delta]$; and*
  - *$\mathbf{x}(\mathbf{x}_i; \delta) = \mathbf{x}_{i+1}$.*

Note that there is a subtle difference between Definition 4 and 5 in how to treat a continuous state $\mathbf{x}$ which terminates a piece of continuous evolution and evokes a discrete jump. Definition 4 is less restrictive because such $\mathbf{x}$ is not required to be inside the mode domain before jump happens. Nevertheless, if all mode domains are assumed to be *closed* sets, then the above two definitions are consistent with each other, that is, $\mathcal{R}_{\mathcal{H}}$ is exactly the set of states that are covered by $\mathcal{T}r(\mathcal{H})$.

One of the major concerned properties of hybrid systems is *safety*. Given an HA $\mathcal{H}$, a safety requirement $\mathcal{S}$ assigns to each mode $q \in Q$ a safe region $S_q \subseteq \mathbb{R}^n$, i.e. $\mathcal{S} = \bigcup_{q \in Q}(\{q\} \times S_q)$. We say that $\mathcal{H}$ satisfies $\mathcal{S}$ if $\mathbf{x} \in S_q$ for all $(q, \mathbf{x}) \in \mathcal{R}_{\mathcal{H}}$.

The prominent feature that distinguishes hybrid systems from traditional discrete programs and makes them more difficult to study is continuous behavior. To facilitate the investigation of continuous parts of hybrid systems, the following definition is proposed.

Definition 6 (Constrained CDS). *A constrained continuous dynamical system (CCDS) is a pair $(D, \mathbf{f})$, where $D \subseteq \mathbb{R}^n$ and $\mathbf{f}$ is a locally Lipschitz continuous vector field in $\mathbb{R}^n$.*

Thus an HA can be regarded as a composition of a finite set of CCDSs, one for each mode, together with discrete transitions among the CCDSs.

2.3 Polynomials and Polynomial Ideals

The tractability of the problems of analysis, verification and synthesis of hybrid systems depends on the language used to specify the hybrid systems, as well as the concerned properties. In this tutorial, we will focus on the class of *polynomial*

expressions, which have powerful modeling ability and are easy to manipulate. We will give a brief overview of the theory of polynomials and polynomial ideals here. For more details please refer to [24].

A *monomial* in n variables $x_1, x_2, \ldots, x_n$ (or briefly $\mathbf{x}$) is a product form $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$, or briefly $\mathbf{x}^{\boldsymbol{\alpha}}$, where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{N}^n$. The number $\sum_{i=1}^{n} \alpha_i$ is called the *degree* of $\mathbf{x}^{\boldsymbol{\alpha}}$.

Let $\mathbb{K}$ be a number field, which can be either $\mathbb{Q}$ or $\mathbb{R}$ in this tutorial. A *polynomial* $p(\mathbf{x})$ in $\mathbf{x}$ (or briefly p) with coefficients in $\mathbb{K}$ is of the form $\sum_{\boldsymbol{\alpha}} c_{\boldsymbol{\alpha}} \mathbf{x}^{\boldsymbol{\alpha}}$, where all $c_{\boldsymbol{\alpha}} \in \mathbb{K}$. The *degree* of p, denoted by $\mathtt{deg}(p)$, is the maximal degree of its component monomials. It is easy to see that a polynomial in $x_1, x_2, \ldots, x_n$ with degree d has at most $\binom{n+d}{d}$ many coefficients. The set of all polynomials in $x_1, x_2, \ldots, x_n$ with coefficients in $\mathbb{K}$ form a *polynomial ring*, denoted by $\mathbb{K}[\mathbf{x}]$.

A *parametric* polynomial is of the form $\sum_{\boldsymbol{\alpha}} u_{\boldsymbol{\alpha}} \mathbf{x}^{\boldsymbol{\alpha}}$, where $u_{\boldsymbol{\alpha}} \in \mathbb{R}$ are not constants but undetermined parameters. It can also be regarded as a standard polynomial $p(\mathbf{u}, \mathbf{x})$ in $\mathbb{Q}[\mathbf{u}, \mathbf{x}]$, where $\mathbf{u} = (u_1, u_2, \ldots, u_w)$ is the set of all parameters. It is easy to see that a parametric polynomial with degree d (in $\mathbf{x}$) has at most $\binom{n+d}{d}$ many indeterminates. In practice, one only keeps some of the $u_{\boldsymbol{\alpha}}$'s as unknowns, by judiciously fixing the coefficients of specific monomials. For any $\mathbf{u}_0 \in \mathbb{R}^w$, we call $p_{\mathbf{u}_0}(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$, obtained by substituting $\mathbf{u}_0$ for $\mathbf{u}$ in $p(\mathbf{u}, \mathbf{x})$, an *instantiation* of $p(\mathbf{u}, \mathbf{x})$.

A vector field $\mathbf{f}$ is called a *polynomial vector field* (PVF) if each element of $\mathbf{f}$ is a polynomial. Given a polynomial $p \in \mathbb{K}[\mathbf{x}]$ and a PVS $\mathbf{f} \in \mathbb{K}^n[\mathbf{x}]$, according to Section 2.1, the Lie derivatives $L_{\mathbf{f}}^k p(\mathbf{x})$ is defined for all $k \in \mathbb{N}$ and all polynomials in $\mathbb{K}[\mathbf{x}]$. The Lie derivatives of a parametric polynomial $p(\mathbf{u}, \mathbf{x}) \in \mathbb{K}[\mathbf{u}, \mathbf{x}]$ can be defined similarly by setting the gradient as

$$\nabla p(\mathbf{u}, \mathbf{x}) \widehat{=} \left(\frac{\partial p}{\partial x_1}, \frac{\partial p}{\partial x_2}, \cdots, \frac{\partial p}{\partial x_n}\right).$$

In this way all $L_{\mathbf{f}}^k p(\mathbf{u}, \mathbf{x})$ are still polynomials in $\mathbb{K}[\mathbf{u}, \mathbf{x}]$.

We next recall the basic theory of polynomial ideals.

Definition 7 (Polynomial Ideal). *A subset $I \subseteq \mathbb{K}[\mathbf{x}]$ is called an* **ideal** *if the following conditions are satisfied:*

1. $0 \in I$;
2. *If $p, g \in I$, then $p + g \in I$;*
3. *If $p \in I$ and $h \in \mathbb{K}[\mathbf{x}]$, then $hp \in I$.*

Let $g_1, g_2, \ldots, g_s \in \mathbb{K}[\mathbf{x}]$. It is easy to check that the set

$$\langle g_1, g_2, \ldots, g_s \rangle \widehat{=} \left\{ \sum_{i=1}^{s} h_i g_i : h_1, h_2, \ldots, h_s \in \mathbb{K}[\mathbf{x}] \right\}$$

is an ideal, called the ideal *generated* by $g_1, g_2, \ldots, g_s$. If $I = \langle g_1, g_2, \ldots, g_s \rangle$, then $\{g_1, g_2, \ldots, g_s\}$ is called a *basis* of I.

Theorem 1 (Hilbert Basis Theorem). *Every ideal $I \subseteq \mathbb{K}[\mathbf{x}]$ has a basis, that is, $I = \langle g_1, g_2, \ldots, g_s \rangle$ for some $g_1, g_2, \ldots, g_s \in \mathbb{K}[\mathbf{x}]$.*

In particular, every ideal $I \subseteq \mathbb{K}[\mathbf{x}]$ has a *Gröbner basis* which possesses very nice properties. To illustrate this, we need to fix an ordering of monomials. First, suppose the list of variables $x_1, x_2, \ldots, x_n$ are ordered by $x_1 \succ x_2 \succ \cdots \succ x_n$. Then $\succ$ induces a total ordering on the set of monomials $\mathbf{x}^{\boldsymbol{\alpha}}$ with $\boldsymbol{\alpha} \in \mathbb{N}^n$. One example is the *lexicographic* (lex for short) order, i.e. $\mathbf{x}^{\boldsymbol{\alpha}} \succ \mathbf{x}^{\boldsymbol{\beta}}$ if and only if there exists $1 \leq i \leq n$ such that $\alpha_i > \beta_i$, and $\alpha_j = \beta_j$ for all $1 \leq j < i$. It can be shown that the lex order of monomials is a *well-ordering*, that is, every nonempty set of monomials has a *least* element. Besides, the lex order is preserved under multiplication, i.e. $\mathbf{x}^{\boldsymbol{\alpha}} \succ \mathbf{x}^{\boldsymbol{\beta}}$ implies $\mathbf{x}^{\boldsymbol{\alpha}}\mathbf{x}^{\boldsymbol{\gamma}} \succ \mathbf{x}^{\boldsymbol{\beta}}\mathbf{x}^{\boldsymbol{\gamma}}$ for any $\boldsymbol{\gamma} \in \mathbb{N}^n$. Such an ordering of monomials as the lex order is called a *monomial ordering*.

Given a monomial ordering $\succ$ and a polynomial $g \in \mathbb{K}[\mathbf{x}]$, rearrange the monomials in p in a descending order as

$$g = c_1\mathbf{x}^{\boldsymbol{\alpha_1}} + c_2\mathbf{x}^{\boldsymbol{\alpha_2}} + \cdots + c_k\mathbf{x}^{\boldsymbol{\alpha_k}} \ ,$$

where all c_i's are nonzero. Then $c_1\mathbf{x}^{\boldsymbol{\alpha}_1}$ is called the *leading term* of g, denoted by $\mathsf{lt}(g)$; c_1 is called the *leading coefficient* of g, denoted by $\mathsf{lc}(g)$; and $\mathbf{x}^{\boldsymbol{\alpha_1}}$ is called the *leading monomial* of g, denoted by $\mathsf{lm}(g)$. For a polynomial $p \in \mathbb{K}[\mathbf{x}]$, if p has a nonzero term $c_{\boldsymbol{\beta}}\mathbf{x}^{\boldsymbol{\beta}}$ and $\mathbf{x}^{\boldsymbol{\beta}}$ is divisible by $\mathsf{lm}(g)$, i.e. $\mathbf{x}^{\boldsymbol{\beta}} = \mathbf{x}^{\boldsymbol{\gamma}}\mathsf{lm}(g)$ for some $\boldsymbol{\gamma} \in \mathbb{N}^n$, then we say p is *reducible* modulo g, and call

$$p' = p - \frac{c_{\boldsymbol{\beta}}}{\mathsf{lc}(g)}\mathbf{x}^{\boldsymbol{\gamma}}g$$

the one-step *reduction* of p modulo g.

Given a finite set of polynomials $G \subsetneq \mathbb{K}[\mathbf{x}]$ and a polynomial $p \in \mathbb{K}[\mathbf{x}]$, we can do a muli-step reduction on p using polynomials in G, until p is reduced to p^* which is not further reducible modulo G. Such p^* is called the *normal form* of p w.r.t. G, denoted by $\mathsf{nf}(p, G)$. For general G, the above process of reduction is guaranteed to terminate; however, the final result $\mathsf{nf}(p, G)$ may vary, depending on the sequence of polynomials chosen from G during reduction. Fortunately, we have

Proposition 1. *Given a monomial ordering, then every ideal $I \subseteq \mathbb{K}[\mathbf{x}]$ other than $\{0\}$ has a basis $G = \{g_1, g_2, \ldots, g_s\}$, such that for any $p \in \mathbb{K}[\mathbf{x}]$, $\mathsf{nf}(p, G)$ is unique. Such G is called a* **Gröbner basis** *of I.*

Furthermore,

Proposition 2. *Let G be a Gröbner basis of an ideal $I \subseteq \mathbb{K}[\mathbf{x}]$. Then for any $p \in \mathbb{K}[\mathbf{x}]$, $p \in I$ if and only if $\mathsf{nf}(p, G) = 0$.*

Most importantly, for any ideal $I = \langle h_1, h_2, \ldots, h_l\rangle \subseteq \mathbb{K}[\mathbf{x}]$, the Gröbner basis G of I can be computed from the h_i's using *Buchberger's Algorithm* [24]. Then by Proposition 2, we get that the *ideal membership* problem, that is to decide whether a polynomial $p \in \mathbb{K}[\mathbf{x}]$ lies in a given ideal $\langle h_1, h_2, \ldots, h_l\rangle \subseteq \mathbb{K}[\mathbf{x}]$, is algorithmically solvable.

The following theorem, which can be deduced from Hilbert Basis Theorem, is key to the proof of several main results in this tutorial.

Theorem 2 (Ascending Chain Condition). *For any ascending chain of ideals*

$$I_1 \subseteq I_2 \subseteq \cdots \subseteq I_l \subseteq \cdots$$

in $\mathbb{K}[\mathbf{x}]$*, there exists an* $N \in \mathbb{N}$ *such that* $I_l = I_N$ *for any* $l \geq N$*.*

2.4 First-Order Theory of Reals

From a logical point of view, polynomials can be used to construct the first-order theory $T(\mathbb{R})$ of *real numbers* (actually of all *real closed fields*), which is very useful in formulating problems arising in the study of hybrid systems. The language of $T(\mathbb{R})$ consists of

- variables: $x, y, z, \ldots, x_1, x_2, \ldots$, which are interpreted over $\mathbb{R}$;
- relational symbols: $>, <, \geq, \leq, =, \neq$;
- Boolean connectives: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \ldots$; and
- quantifiers: $\forall, \exists$.

A *term* of $T(\mathbb{R})$ over a finite set of variables $\{x_1, x_2, \ldots, x_n\}$ is a polynomial $p \in \mathbb{Q}[x_1, x_2, \ldots, x_n]$. An *atomic formula* of $T(\mathbb{R})$ is of the form $p \rhd 0$, where $\rhd$ is any relational symbol. A *quantifier-free formula* (QFF) of $T(\mathbb{R})$ is a Boolean combination of atomic formulas. A generic formula of $T(\mathbb{R})$ is built up from atomic formulas using Boolean connectives as well as quantifiers.

A profound result about $T(\mathbb{R})$ is that it admits *quantifier elimination* (QE) [83]. That is, any formula φ in $T(\mathbb{R})$ has a quantifier-free equivalent φ_{QF} involving only *free* variables of φ, and φ_{QF} can be computed from φ using QE algorithms. An immediate consequence of this result is the *decidability* of $T(\mathbb{R})$: the truth value of any formula in $T(\mathbb{R})$ can be decided.

Formulas in $T(\mathbb{R})$ define a special class of sets:

Definition 8 (Semi-algebraic Set). *A subset* $A \subseteq \mathbb{R}^n$ *is called a* semi-algebraic set *(SAS), if there exists a QFF* ϕ *in* $T(\mathbb{R})$ *over variables* $x_1, x_2, \ldots, x_n$*, or briefly* $\mathbf{x}$*, such that*

$$A = \{\mathbf{x} \in \mathbb{R}^n \mid \phi(\mathbf{x}) \text{ is true}\} \ .$$

Let $\mathcal{A}(\phi)$ denote the SAS defined by a QFF ϕ. Then from Definition 8 it is easy to check that SASs are closed under common set operations:

- $\mathcal{A}(\phi_1) \cap \mathcal{A}(\phi_2) = \mathcal{A}(\varphi_1 \wedge \varphi_2)$;
- $\mathcal{A}(\phi_1) \cup \mathcal{A}(\phi_2) = \mathcal{A}(\varphi_1 \vee \phi_2)$;
- $\mathcal{A}(\phi_1)^c = \mathcal{A}(\neg\phi_1)$;
- $\mathcal{A}(\phi_1) \setminus \mathcal{A}(\phi_2) = \mathcal{A}(\phi_1) \cap \mathcal{A}(\phi_2)^c = \mathcal{A}(\phi_1 \wedge \neg\phi_2)$,

where A^c and $A \setminus B$ stand for the complement and subtraction operation of sets respectively. Moreover, checking of emptiness, inclusion and equality of SASs can be done by the decidability of $T(\mathbb{R})$.

For convenience, in the rest of this tutorial, we do not distinguish between an SAS $\mathcal{A}(\phi)$ and its defining formula ϕ. That is, we will use $T(\mathbb{R})$-formulas to

represent SASs and use Boolean connectives as set operators. Besides, it is easy to check that any SAS can be represented by a QFF in the form of

$$\phi(\mathbf{x}) \widehat{=} \bigvee_{k=1}^{K} \bigwedge_{j=1}^{J_k} p_{kj}(\mathbf{x}) \triangleright 0\,,$$

where $p_{kj}(\mathbf{x}) \in \mathbb{Q}[\mathbf{x}]$ and $\triangleright \in \{\geq, >\}$. Therefore restricting SASs to formulas of this shape will not lose any generality.

Definition 9 (Semi-algebraic Template). *A* semi-algebraic template *with degree d is of the form*

$$\phi(\mathbf{u}, \mathbf{x}) \widehat{=} \bigvee_{k=1}^{K} \bigwedge_{j=1}^{J_k} p_{kj}(\mathbf{u}_{kj}, \mathbf{x}) \triangleright 0\,,$$

where $p_{kj} \in \mathbb{Q}[\mathbf{u}_{kj}, \mathbf{x}]$ are parametric polynomials with degree d (in $\mathbf{x}$), $\mathbf{u}$ is the collection of parameters appearing in each p_{kj} (i.e. $\mathbf{u}_{kj}$), and $\triangleright \in \{\geq, >\}$.

As mentioned in Section 2.3, we will focus on hybrid systems and properties described by polynomial expressions.

Definition 10. *A polynomial CDS (or CCDS, HA, safety property, etc), denoted by PCDS (or PCCDS, PHA, etc) for short, is a CDS (or CCDS, HA, safety property etc, respectively) wherein the sets are SASs and the vector fields are PVFs (with rational coefficients).*

3 Computing Invariants for Hybrid Systems

3.1 Continuous and Global Invariant

An *invariant* of a hybrid system is a property that holds at every reachable state of the system.

Definition 11 (Invariant). *An invariant of an HA $\mathcal{H}$ maps to each $q \in Q$ a subset $I_q \subseteq \mathbb{R}^n$, such that for all $(q, \mathbf{x}) \in \mathcal{R}_{\mathcal{H}}$, we have $\mathbf{x} \in I_q$.*

One effective way of finding invariants of hybrid systems is to generate so-called *inductive invariants*, as inductiveness is usually checkable [75].

Definition 12 (Inductive Invariant). *Given an HA $\mathcal{H}$, an inductive invariant maps to each $q \in Q$ a subset $I_q \subseteq \mathbb{R}^n$, such that the following conditions are satisfied:*

1. *$\Xi_q \subseteq I_q$ for all $q \in Q$;*
2. *for any $e = (q, q') \in E$, if $\mathbf{x} \in I_q \cap G_e$, then $\mathbf{x}' = R_e(\mathbf{x}) \in I_{q'}$;*
3. *for any $q \in Q$ and any $\mathbf{x}_0 \in I_q$, if there exists a $\delta \geq 0$ s.t. the solution $\mathbf{x}(\mathbf{x}_0; t)$ to $\dot{\mathbf{x}} = \mathbf{f}_q$ satisfies: (i) $\mathbf{x}(\mathbf{x}_0; \delta) = \mathbf{x}'$; and (ii) $\mathbf{x}(\mathbf{x}_0; t) \in D_q$ for all $t \in [0, \delta]$, then $\mathbf{x}' \in I_q$.*

It is easy to check that any inductive invariant is also an invariant. We assume in this section that all invariants mentioned are *inductive*.

In Definition 12, condition 1 and 2 are about initial states and discrete inductiveness, which can be checked using the standard techniques for the verification of discrete programs [92]. However, it is not so straightforward and requires special efforts to check condition 3, for which the notion of *continuous invariant*[2] [65,56] is quite useful.

Definition 13 (Continuous Invariant). *A subset $I \subseteq \mathbb{R}^n$ is called a* continuous invariant *(CI) of a CCDS $(D, \mathbf{f})$ if for any $\mathbf{x}_0 \in I$ and any $T \geq 0$, we have:*

$$(\forall t \in [0, T].\, \mathbf{x}(\mathbf{x}_0; t) \in D) \Longrightarrow (\forall t \in [0, T].\, \mathbf{x}(\mathbf{x}_0; t) \in I)\,,$$

or equivalently,

$$(\forall t \in [0, T].\, \mathbf{x}(\mathbf{x}_0; t) \in D) \Longrightarrow \mathbf{x}(\mathbf{x}_0; T) \in I\,.$$

By Definition 13, it is not difficult to check that condition 3 in Definition 12 is equivalent to

3'. for any $q \in Q$, I_q is a CI of $(D_q, \mathbf{f}_q)$.

To distinguish from CI, we refer to the inductive invariant in Definition 12 a *global invariant* (GI). Simply, a GI of an HA $\mathcal{H}$ consists of a set of CIs, one for each CCDS corresponding to a mode of the HA. Using GI, if $I_q \subseteq S_q$ for all q, then a safety property $\mathcal{S}$ can be verified without computing $\mathcal{R}_{\mathcal{H}}$. In the rest of this section, we will present an approach for automatically discovering semi-algebraic CIs (SCI) and semi-algebraic GIs (SGI) for PCCDS and PHA respectively.

3.2 Predicting Continuous Evolution via Lie Derivatives

Given a PVF $\mathbf{f}$, we can make use of Lie derivatives to investigate the tendency of $\mathbf{f}$'s trajectories in terms of a polynomial p. To capture this, look at Example 1 shown in I of Figure 1.

Example 1. Suppose $\mathbf{f} = (-x, y)$ and $p(x, y) = x + y^2$. Then

$$\begin{aligned} L^0_{\mathbf{f}} p(x, y) &= x + y^2 \\ L^1_{\mathbf{f}} p(x, y) &= -x + 2y^2 \\ L^2_{\mathbf{f}} p(x, y) &= x + 4y^2 \\ &\vdots \end{aligned}$$

[2] In some later sections of this tutorial when we talk about the Hybrid Hoare Logic (HHL), the terminology *differential invariant* is used instead of continuous invariant, with exactly the same meaning.

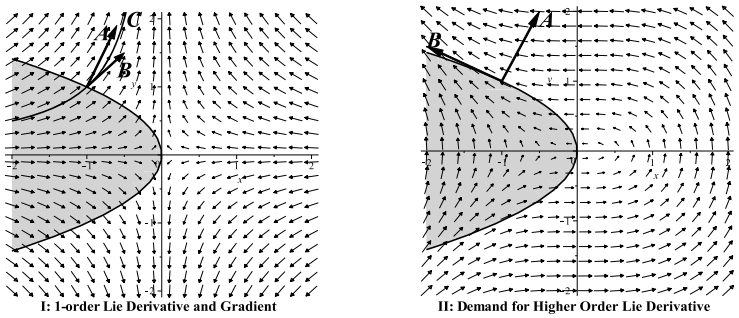

Fig. 1. Lie Derivatives

In I of Figure 1, vector B denotes the corresponding evolution direction of the vector field $\mathbf{f} = (-x, y)$ at point $(-1, 1)$. We could imagine the points on the parabola $p(x, y) = x + y^2$ with zero energy, and the points in the white area have positive energy, i.e. $p(x, y) > 0$. Vector A is the gradient $\nabla p|_{(-1,1)}$ of $p(x, y)$, which infers that the trajectory starting at $(-1, 1)$ will enter the white area immediately if the angle, between $\nabla p|_{(-1,1)}$ and the evolution direction at $(-1, 1)$, is less than $\frac{\pi}{2}$, which means equivalently that the 1-order Lie derivative $L^1_{\mathbf{f}} p|_{(-1,1)}$ is positive. Thus the 1-order Lie derivative $L^1_{\mathbf{f}} p|_{(-1,1)} = 3$ predicts that there is some positive $\epsilon > 0$ such that the trajectory starting at $(-1, 1)$ (curve C) has the property $p\big(\mathbf{x}((-1, 1); t)\big) > 0$ for all $t \in (0, \epsilon)$.

However, if the angle between the gradient and the evolution direction equals $\frac{\pi}{2}$ or the gradient is zero-vector, then the 1-order Lie derivative is zero and it is impossible to predict trajectory tendency by means of 1-order Lie derivative. In this case, we resort to nonzero higher order Lie derivatives. For this purpose, we introduce the *pointwise rank* of p with respect to $\mathbf{f}$ as the function $\gamma_{p,\mathbf{f}} : \mathbb{R}^n \to \mathbb{N} \cup \{\infty\}$ defined by

$$\gamma_{p,\mathbf{f}}(\mathbf{x}) = \min\{k \in \mathbb{N} \mid L^k_{\mathbf{f}} p(\mathbf{x}) \neq 0\}$$

if such k exists, and $\gamma_{p,\mathbf{f}}(\mathbf{x}) = \infty$ otherwise.

Example 2. Let $\mathbf{f}(x, y) = (-2y, x^2)$ and $h(x, y) = x + y^2$. Then

$$\begin{aligned} L^0_{\mathbf{f}} h(x, y) &= x + y^2 \\ L^1_{\mathbf{f}} h(x, y) &= -2y + 2x^2 y \\ L^2_{\mathbf{f}} h(x, y) &= -8y^2 x - (2 - 2x^2)x^2 \\ &\vdots \end{aligned}$$

Here, $\gamma_{h,\mathbf{f}}(0, 0) = \infty$, $\gamma_{h,\mathbf{f}}(-4, 2) = 1$, etc.

Look at II of Figure 1. At point $(-1,1)$ on curve $h(x,y)=0$, the gradient of h is $(1,2)$ (vector A) and the evolution direction is $(-2,1)$ (vector B), so their inner product is zero. Thus it is impossible to predict the tendency (in terms of curve $h(x,y)=0$) of the trajectory starting from $(-1,1)$ via the 1-order Lie derivative. By a simple computation, the 2-order Lie derivative $L_{\mathbf{f}}^2 h(-1,1)$ is 8. Hence $\gamma_{h,\mathbf{f}}(-1,1)=2$. In the sequel, we shall show how to use such high order Lie derivatives to analyze the trajectory tendency.

For analyzing trajectory tendency by high order Lie derivatives, we need the following fact.

Proposition 3. *Given a PVF* $\mathbf{f}$ *and a polynomial* p*, then for any* $\mathbf{x}_0 \in \mathbb{R}^n$*,* $p(\mathbf{x}_0)=0$ *if and only if* $\gamma_{p,\mathbf{f}}(\mathbf{x}_0)\neq 0$*. Let* $\mathbf{x}(t)\widehat{=}\mathbf{x}(\mathbf{x}_0;t)$*. Then it follows that*

(a) if $\gamma_{p,\mathbf{f}}(\mathbf{x}_0)<\infty$ *and* $L_{\mathbf{f}}^{\gamma_{p,\mathbf{f}}(\mathbf{x}_0)}p(\mathbf{x}_0)>0$*, then*

$$\exists \epsilon>0, \forall t\in(0,\epsilon).\, p(\mathbf{x}(t))>0;$$

(b) if $\gamma_{p,\mathbf{f}}(\mathbf{x}_0)<\infty$ *and* $L_{\mathbf{f}}^{\gamma_{p,\mathbf{f}}(\mathbf{x}_0)}p(\mathbf{x}_0)<0$*, then*

$$\exists \epsilon>0, \forall t\in(0,\epsilon).\, p(\mathbf{x}(t))<0;$$

(c) if $\gamma_{p,\mathbf{f}}(\mathbf{x}_0)=\infty$*, then*

$$\exists \epsilon>0, \forall t\in(0,\epsilon).\, p(\mathbf{x}(t))=0.$$

Proof. By Section 2.1, $p(\mathbf{x}(t))$ is the composition of two analytic functions, which implies [52] that the Taylor expansion of $p(\mathbf{x}(t))$ at $t=0$

$$\begin{aligned} p(\mathbf{x}(t)) &= p(\mathbf{x}_0)+\frac{\mathrm{d}p}{\mathrm{d}t}\cdot t+\frac{\mathrm{d}^2p}{\mathrm{d}t^2}\cdot\frac{t^2}{2!}+\cdots \\ &= L_{\mathbf{f}}^0 p(\mathbf{x}_0)+L_{\mathbf{f}}^1 p(\mathbf{x}_0)\cdot t+L_{\mathbf{f}}^2 p(\mathbf{x}_0)\cdot\frac{t^2}{2!}+\cdots \end{aligned} \tag{2}$$

converges in a neighborhood of zero. Then the conclusion of Proposition 3 follows immediately from formula (2) by case analysis on the sign of $L_{\mathbf{f}}^{\gamma_{p,\mathbf{f}}(\mathbf{x}_0)}p(\mathbf{x}_0)$. □

Based on this proposition, we introduce the notion of *transverse set* to indicate the tendency of the trajectories of a considered PVF in terms of the first nonzero high order Lie derivative of an underlying polynomial as follows.

Definition 14 (Transverse Set). *Given a polynomial* p *and a PVF* $\mathbf{f}$*, the transverse set of* $\mathbf{f}$ *over the domain* $P\widehat{=}p(\mathbf{x})\geq 0$ *is*

$$Trans_{\mathbf{f}\uparrow p}\widehat{=}\{\mathbf{x}\in\mathbb{R}^n \mid \gamma_{p,\mathbf{f}}(\mathbf{x})<\infty \wedge L_{\mathbf{f}}^{\gamma_{p,\mathbf{f}}(\mathbf{x})}p(\mathbf{x})<0\}.$$

Intuitively, if $\mathbf{x}\in Trans_{\mathbf{f}\uparrow p}$, then either $\mathbf{x}$ is not in P, or $\mathbf{x}$ is on the boundary of P (i.e. $p(\mathbf{x})=0$) such that the trajectory $\mathbf{x}(t)$ starting from $\mathbf{x}$ will exit P immediately.

3.3 Computing Transverse Set

The set $Trans_{\mathbf{f}\uparrow p}$ in Definition 14 plays a crucial role in developing the automatic invariant generation method. First of all, we have

Theorem 3. *Given a polynomial $p \in \mathbb{Q}[\mathbf{x}]$ and a PVF $\mathbf{f} \in \mathbb{Q}^n[\mathbf{x}]$, the set $Trans_{\mathbf{f}\uparrow p}$ is an SAS, and its explicit representation is computable.*

To prove this theorem, it suffices to show $\gamma_{p,\mathbf{f}}(\mathbf{x})$ is computable for each $\mathbf{x} \in \mathbb{R}^n$. However, $\gamma_{p,\mathbf{f}}(\mathbf{x})$ may be infinite for some $\mathbf{x}$. Thus, it seems that we have to compute $L^k_{\mathbf{f}}p(\mathbf{x})$ infinitely many times for such $\mathbf{x}$ to determine if $\mathbf{x} \in Trans_{\mathbf{f}\uparrow p}$. Fortunately, we can find a uniform upper bound on $\gamma_{p,\mathbf{f}}(\mathbf{x})$ for all $\mathbf{x}$ with finite pointwise rank. To see this, consider the polynomial ideals in ring $\mathbb{Q}[\mathbf{x}]$ generated by Lie derivatives $L^0_{\mathbf{f}}p, L^1_{\mathbf{f}}p, \ldots, L^i_{\mathbf{f}}p$ for all $i \geq 0$, i.e.

$$J_i \widehat{=} \langle L^0_{\mathbf{f}}p(\mathbf{x}), L^1_{\mathbf{f}}p(\mathbf{x}), \ldots, L^i_{\mathbf{f}}p(\mathbf{x})\rangle\,.$$

Note that

$$J_0 \subseteq J_1 \subseteq \cdots \subseteq J_l \subseteq \cdots$$

forms an ascending chain of ideals in $\mathbb{Q}[\mathbf{x}]$. By Theorem 2, the number

$$N_{p,\mathbf{f}} \widehat{=} \min\{i \in \mathbb{N} \mid J_i = J_{i+1}\}, \tag{3}$$

or equivalently,

$$N_{p,\mathbf{f}} \widehat{=} \min\{i \in \mathbb{N} \mid L^{i+1}_{\mathbf{f}}p \in J_i\}$$

is well-defined. Furthermore, $N_{p,\mathbf{f}}$ can be computed by solving the ideal membership problem with the assistance of an algebraic tool like Maple [61].

Example 3. For $\mathbf{f}$ and h in Example 2, by simple computations we get $L^1_{\mathbf{f}}h \notin \langle L^0_{\mathbf{f}}h\rangle$, $L^2_{\mathbf{f}}h \notin \langle L^0_{\mathbf{f}}h, L^1_{\mathbf{f}}h\rangle$, $L^3_{\mathbf{f}}h \in \langle L^0_{\mathbf{f}}h, L^1_{\mathbf{f}}h, L^2_{\mathbf{f}}h\rangle$, so $N_{h,\mathbf{f}} = 2$.

Actually, the integer $N_{p,\mathbf{f}}$ is the upper bound mentioned above on pointwise rank by the following two theorems.

Theorem 4 (Fixed Point Theorem). *If $J_i = J_{i+1}$, then $J_i = J_l$ for all $l > i$.*

Proof. We prove this fact by induction on l. Base case: $J_i = J_{i+1}$. Assume $J_i = J_l$ for some $l \geq i+1$. Then there are $g_j \in \mathbb{Q}[\mathbf{x}]$ for $0 \leq j \leq i$, such that $L^l_{\mathbf{f}}p = \sum_{j=0}^{i} g_j L^j_{\mathbf{f}}p$. By the definition of Lie derivatives it follows that

$$\begin{aligned} L_{\mathbf{f}}^{l+1}p &= (\nabla L_{\mathbf{f}}^{l}p, \mathbf{f}) \\ &= (\nabla \sum_{j=0}^{i} g_j L_{\mathbf{f}}^{j}p, \mathbf{f}) \\ &= (\sum_{j=0}^{i} L_{\mathbf{f}}^{j}p \nabla g_j + \sum_{j=0}^{i} g_j \nabla L_{\mathbf{f}}^{j}p, \mathbf{f}) \\ &= \sum_{j=0}^{i} (\nabla g_j, \mathbf{f}) L_{\mathbf{f}}^{j}p + \sum_{j=0}^{i} g_j L_{\mathbf{f}}^{j+1}p \\ &= \sum_{j=0}^{i} (\nabla g_j, \mathbf{f}) L_{\mathbf{f}}^{j}p + \sum_{j=1}^{i} g_{j-1} L_{\mathbf{f}}^{j}p + g_i L_{\mathbf{f}}^{i+1}p\,. \end{aligned} \tag{4}$$

By base case, $L_{\mathbf{f}}^{i+1}p \in J_i$. Then by (4) we get $L_{\mathbf{f}}^{l+1}p \in J_i$, so $J_i = J_{l+1}$. By induction, the fact follows immediately. □

Theorem 5 (Rank Theorem). *Given a polynomial p and a PVF $\mathbf{f}$, for any $\mathbf{x} \in \mathbb{R}^n$, if $\gamma_{p,\mathbf{f}}(\mathbf{x}) < \infty$, then $\gamma_{p,\mathbf{f}}(\mathbf{x}) \leq N_{p,\mathbf{f}}$, where $N_{p,\mathbf{f}}$ is defined in (3).*

Proof. If $N_{p,\mathbf{f}} < \gamma_{p,\mathbf{f}}(\mathbf{x}) < \infty$, then $\bigwedge_{i=0}^{N_{p,\mathbf{f}}} L_{\mathbf{f}}^{i}p(\mathbf{x}) = 0$. By (3) and Theorem 4 we get $L_{\mathbf{f}}^{i}p(\mathbf{x}) = 0$ for all $i \in \mathbb{N}$. Thus $\gamma_{p,\mathbf{f}}(\mathbf{x}) = \infty$, which is a contradiction. □

Now, applying the above two theorems we can prove Theorem 3.

Proof (of Theorem 3). First by Theorem 5, for any $\mathbf{x}$,

$$\mathbf{x} \in Trans_{\mathbf{f}\uparrow p} \Longleftrightarrow \gamma_{p,\mathbf{f}}(\mathbf{x}) \leq N_{p,\mathbf{f}} \wedge L_{\mathbf{f}}^{\gamma_{p,\mathbf{f}}(\mathbf{x})}p(\mathbf{x}) < 0\,. \tag{5}$$

Given p and $\mathbf{f}$, let

$$\pi^{(0)}(p, \mathbf{f}, \mathbf{x}) \mathrel{\widehat{=}} p(\mathbf{x}) < 0\,;$$

for $1 \leq i \in \mathbb{N}$,

$$\pi^{(i)}(p, \mathbf{f}, \mathbf{x}) \mathrel{\widehat{=}} \left(\bigwedge_{0 \leq j < i} L_{\mathbf{f}}^{j}p(\mathbf{x}) = 0 \right) \wedge L_{\mathbf{f}}^{i}p(\mathbf{x}) < 0\,,$$

and

$$\pi(p, \mathbf{f}, \mathbf{x}) \mathrel{\widehat{=}} \bigvee_{0 \leq i \leq N_{p,\mathbf{f}}} \pi^{(i)}(p, \mathbf{f}, \mathbf{x})\,.$$

Then from (5) we have another equivalence

$$\mathbf{x} \in Trans_{\mathbf{f}\uparrow p} \Longleftrightarrow \pi(p, \mathbf{f}, \mathbf{x})\,. \tag{6}$$

Thus $Trans_{\mathbf{f}\uparrow p}$ is actually an SAS which can be represented by $\pi(p, \mathbf{f}, \mathbf{x})$. □

In automatic invariant generation, it actually makes use of parametric polynomials $p(\mathbf{u}, \mathbf{x})$. The following theorem indicates Theorem 5 still holds after substituting $p(\mathbf{u}, \mathbf{x})$ for $p(\mathbf{x})$.

Theorem 6 (Parametric Rank Theorem). *Given a parametric polynomial* $p(\mathbf{u}, \mathbf{x})$ *and a PVF* $\mathbf{f}$*, there is an integer* $N_{p,\mathbf{f}} \in \mathbb{N}$ *such that* $\gamma_{p_{\mathbf{u}_0},\mathbf{f}}(\mathbf{x}) < \infty$ *implies* $\gamma_{p_{\mathbf{u}_0},\mathbf{f}}(\mathbf{x}) \leq N_{p,\mathbf{f}}$ *for all* $\mathbf{x} \in \mathbb{R}^n$ *and all* $\mathbf{u}_0 \in \mathbb{R}^w$*.*

The proof of this theorem is quite close to the one of Theorem 5. The difference lies in the settings of polynomials. Here, all polynomials and ideals are considered in the polynomial ring $\mathbb{Q}[\mathbf{u}, \mathbf{x}]$, and the number $N_{p,\mathbf{f}}$ is defined similarly as in (3).

3.4 Computing SCI in Simple Case

Given a PCCDS $(D, \mathbf{f})$, the task is to find SCIs for $(D, \mathbf{f})$. First of all, we illustrate how to compute an SCI of the simple form $P \widehat{=} p(\mathbf{x}) \geq 0$ for a simple domain $D \widehat{=} h(\mathbf{x}) \geq 0$.

Notice that if $\mathbf{x}_0$ is in the interior of $P \cap D$, then the trajectory $\mathbf{x}(t)$ starting at $\mathbf{x}_0$ will remain in the interior within adequately small $t > 0$. Therefore, the condition of CI could be violated only at the points $\mathbf{x}$ on the boundary of P, i.e. $p(\mathbf{x}) = 0$. Thus by Definition 14 and Proposition 3, P is an invariant of $(D, \mathbf{f})$ if and only if for all $\mathbf{x}$

$$p(\mathbf{x}) = 0 \rightarrow \mathbf{x} \notin (Trans_{\mathbf{f}\uparrow p} \setminus Trans_{\mathbf{f}\uparrow h}),$$

i.e.

$$p(\mathbf{x}) = 0 \rightarrow \mathbf{x} \in (Trans_{\mathbf{f}\uparrow p})^c \cup Trans_{\mathbf{f}\uparrow h}. \tag{7}$$

By equivalence (6), the formula (7) is equivalent to

$$p(\mathbf{x}) = 0 \rightarrow (\neg\pi(p, \mathbf{f}, \mathbf{x}) \vee \pi(h, \mathbf{f}, \mathbf{x})),$$

i.e.

$$\Big(p(\mathbf{x}) = 0 \wedge \pi(p, \mathbf{f}, \mathbf{x})\Big) \rightarrow \pi(h, \mathbf{f}, \mathbf{x}). \tag{8}$$

Let $\theta(h, p, \mathbf{f}, \mathbf{x})$ denote the formula (8). Then we obtain the following sufficient and necessary condition for P being an SCI of $(D, \mathbf{f})$.

Theorem 7 (Criterion Theorem). *Given a polynomial* p*,* $p(\mathbf{x}) \geq 0$ *is an SCI of the PCCDS* $(h(\mathbf{x}) \geq 0, \mathbf{f})$ *if and only if the formula* $\theta(h, p, \mathbf{f}, \mathbf{x})$ *defined as (8) is true for all* $\mathbf{x} \in \mathbb{R}^n$*.*

Based on Theorem 7, a constraint based method for generating SCIs in the simple form can be presented as follows.

I. First, set a simple semi-algebraic template $P \widehat{=} p(\mathbf{u}, \mathbf{x}) \geq 0$ using a parametric polynomial $p(\mathbf{u}, \mathbf{x})$.
II. Then apply QE[3] to the formula $\forall \mathbf{x}.\theta(h, p, \mathbf{f}, \mathbf{x})$. In practice, QE may be applied to a formula $\forall \mathbf{x}.(\theta \wedge \phi)$, where ϕ is a formula imposing some additional constraint on the SCI P. If the output of QE is *false*, then there is no SCI in the form of the predefined P; otherwise, a constraint on $\mathbf{u}$, denoted by $R(\mathbf{u})$, will be returned.
III. Now, use an SMT solver like [26] to pick a $\mathbf{u}_0 \in R(\mathbf{u})$ and then $p_{\mathbf{u}_0}(\mathbf{x}) \geq 0$ is an SCI of $(h(\mathbf{x}) \geq 0, \mathbf{f})$ by Theorem 7.

Example 4. Again, we make use of Example 2 to demonstrate the above method. Here, we take $D \widehat{=} h(x, y) \geq 0$ with $h(x, y) \widehat{=} -x - y^2$ as the domain.

Apply procedure (I-III), we have:

1. Set a template $P \widehat{=} p(\mathbf{u}, \mathbf{x}) \geq 0$ with $p(\mathbf{u}, \mathbf{x}) \widehat{=} ay(x - y)$, where $\mathbf{u} \widehat{=} (a)$. By a simple computation we get $N_{p,\mathbf{f}} = 2$.
2. Compute the corresponding formula

$$\theta(h, p, \mathbf{f}, \mathbf{x}) \widehat{=} p = 0 \wedge (\pi^{(0)}_{p,\mathbf{f},\mathbf{x}} \vee \pi^{(1)}_{p,\mathbf{f},\mathbf{x}} \vee \pi^{(2)}_{p,\mathbf{f},\mathbf{x}}) \longrightarrow (\pi^{(0)}_{h,\mathbf{f},\mathbf{x}} \vee \pi^{(1)}_{h,\mathbf{f},\mathbf{x}} \vee \pi^{(2)}_{h,\mathbf{f},\mathbf{x}})$$

where

$$\begin{aligned}
\pi^{(0)}_{h,\mathbf{f},\mathbf{x}} &\widehat{=} -x - y^2 < 0,\\
\pi^{(1)}_{h,\mathbf{f},\mathbf{x}} &\widehat{=} -x - y^2 = 0 \wedge 2y - 2x^2 y < 0,\\
\pi^{(2)}_{h,\mathbf{f},\mathbf{x}} &\widehat{=} -x - y^2 = 0 \wedge 2y - 2x^2 y = 0 \wedge 8xy^2 + 2x^2 - 2x^4 < 0,\\
\pi^{(0)}_{p,\mathbf{f},\mathbf{x}} &\widehat{=} ay(x - y) < 0,\\
\pi^{(1)}_{p,\mathbf{f},\mathbf{x}} &\widehat{=} ay(x - y) = 0 \wedge -2ay^2 + ax^3 - 2yax^2 < 0,\\
\pi^{(2)}_{p,\mathbf{f},\mathbf{x}} &\widehat{=} ay(x - y) = 0 \wedge -2ay^2 + ax^3 - 2yax^2 = 0\\
&\quad \wedge 40axy^2 - 16ay^3 + 32ax^3 y - 10ax^4 < 0.
\end{aligned}$$

In addition, we require the two points $\{(-1, 0.5), (-0.5, -0.6)\}$ to be contained in P. Then apply QE to the formula

$$\forall x \forall y. \big(\theta(h, p, \mathbf{f}, \mathbf{x}) \wedge 0.5a(-1 - 0.5) \geq 0 \wedge -0.6a(-0.5 + 0.6) \geq 0\big) \ .$$

The result is $a \leq 0$.
3. Just pick $a = -1$, and then $-xy + y^2 \geq 0$ is an SCI of $(D, \mathbf{f})$. The grey part of Picture III in Fig. 2 is the intersection of the invariant P and domain D.

[3] QE has been implemented in many computer algebra tools such as QEPCAD [17], Redlog [30], Mathematica [88], etc.

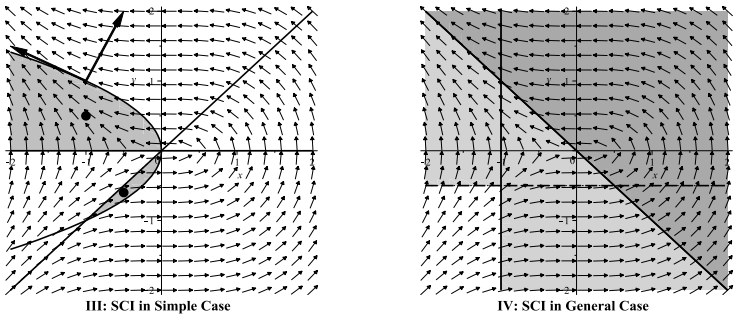

Fig. 2. Semi-Algebraic Continuous Invariants

3.5 Computing SCI in General Case

Now, consider how to automatically discover SCIs of a PCCDS in general case. Given a PCCDS $(D, \mathbf{f})$ with

$$D \widehat{=} \bigvee_{m=1}^{M} \bigwedge_{l=1}^{L_m} p_{ml}(\mathbf{x}) \triangleright 0 \quad \text{with} \quad \triangleright \in \{\geq, >\} \ , \tag{9}$$

the procedure of automatically generating SCIs with a general template

$$P \widehat{=} \bigvee_{k=1}^{K} \bigwedge_{j=1}^{J_k} p_{kj}(\mathbf{u}_{kj}, \mathbf{x}) \triangleright 0 \quad \text{with} \quad \triangleright \in \{\geq, >\}$$

for $(D, \mathbf{f})$, is essentially the same as the steps (I-III) depicted in Section 3.4. However, we must sophisticatedly handle the complex Boolean structures of the formulas herein. In what follows, the main results on general SCI generation are outlined without rigorous proofs. Please refer to [54] for details.

Necessary-Sufficient Condition for CI. First of all, we study a necessary and sufficient condition like formula (7) for P being a CI of $(D, \mathbf{f})$. To analyze the evolution tendency of trajectories of $\mathbf{f}$ in terms of a subset $A \subseteq \mathbb{R}^n$, the following notions and notations are needed.

$$\begin{aligned} \mathrm{In}_{\mathbf{f}}(A) &\widehat{=} \{\mathbf{x}_0 \in \mathbb{R}^n \mid \exists \epsilon > 0 \forall t \in (0, \epsilon). \, \mathbf{x}(\mathbf{x}_0; t) \in A\}, \\ \mathrm{IvIn}_{\mathbf{f}}(A) &\widehat{=} \{\mathbf{x}_0 \in \mathbb{R}^n \mid \exists \epsilon > 0 \forall t \in (0, \epsilon). \, \mathbf{x}(\mathbf{x}_0; -t) \in A\}. \end{aligned}$$

Intuitively, $\mathbf{x}_0 \in \mathrm{In}_{\mathbf{f}}(A)$ means that the trajectory starting from $\mathbf{x}_0$ enters A immediately and keeps inside A for a certain amount of time; $\mathbf{x}_0 \in \mathrm{IvIn}_{\mathbf{f}}(A)$ means that the trajectory through $\mathbf{x}_0$ reaches $\mathbf{x}_0$ from inside A. By the notion of CI, it can be proved that

Theorem 8. *Given a CCDS $(D, \mathbf{f})$, a subset $P \subseteq \mathbb{R}^n$ is a CI of $(D, \mathbf{f})$ if and only if*

1. $\forall \mathbf{x} \in P \cap D \cap \mathrm{In}_{\mathbf{f}}(D).\, \mathbf{x} \in \mathrm{In}_{\mathbf{f}}(P)$; *and*
2. $\forall \mathbf{x} \in P^c \cap D \cap \mathrm{IvIn}_{\mathbf{f}}(D).\, \mathbf{x} \in \big(\mathrm{IvIn}_{\mathbf{f}}(P)\big)^c$.

Necessary-Sufficient Condition for SCI. Given a PCCDS $(D, \mathbf{f})$ and a semi-algebraic template P, to encode the conditions in Theorem 8 as polynomial formulas, it is sufficient to show that $\mathrm{In}_{\mathbf{f}}(D)$, $\mathrm{In}_{\mathbf{f}}(P)$, $\mathrm{IvIn}_{\mathbf{f}}(D)$ and $\mathrm{IvIn}_{\mathbf{f}}(P)$ are all SASs if D and P are SASs, for which we have the following lemmas[4].

Lemma 1. *For any polynomial p and PVF $\mathbf{f}$,*

$$\mathrm{In}_{\mathbf{f}}(p > 0) = \psi^+(p, \mathbf{f}) \text{ and}$$
$$\mathrm{In}_{\mathbf{f}}(p \geq 0) = \psi_0^+(p, \mathbf{f}),$$

where

$$\psi^+(p, \mathbf{f}) \widehat{=} \bigvee_{0 \leq i \leq N_{p,\mathbf{f}}} \psi^{(i)}(p, \mathbf{f}) \text{ with } \psi^{(i)}(p, \mathbf{f}) \widehat{=} \Big(\bigwedge_{0 \leq j < i} L_{\mathbf{f}}^j p = 0 \Big) \wedge L_{\mathbf{f}}^i p > 0, \text{ and}$$
$$\psi_0^+(p, \mathbf{f}) \widehat{=} \psi^+(p, \mathbf{f}) \vee \Big(\bigwedge_{0 \leq j \leq N_{p,\mathbf{f}}} L_{\mathbf{f}}^j p = 0 \Big).$$

Lemma 2. *For an SAS D defined by (9) and a PVF $\mathbf{f}$, we have*

$$\mathrm{In}_{\mathbf{f}}(D) = \bigvee_{m=1}^{M} \bigwedge_{l=1}^{L_m} \mathrm{In}_{\mathbf{f}}(p_{ml} \triangleright 0).$$

Lemma 3. *For any polynomial p and PVF $\mathbf{f}$,*

$$\mathrm{IvIn}_{\mathbf{f}}(p > 0) = \varphi^+(p, \mathbf{f}) \text{ and}$$
$$\mathrm{IvIn}_{\mathbf{f}}(p \geq 0) = \varphi_0^+(p, \mathbf{f}),$$

where

$$\varphi^+(p, \mathbf{f}) \widehat{=} \bigvee_{0 \leq i \leq N_{p,\mathbf{f}}} \varphi^{(i)}(p, \mathbf{f}) \text{ with } \varphi^{(i)}(p, \mathbf{f}) \widehat{=} \Big(\bigwedge_{0 \leq j < i} L_{\mathbf{f}}^j p = 0 \Big) \wedge (-1)^i \cdot L_{\mathbf{f}}^i p > 0, \text{ and}$$
$$\varphi_0^+(p, \mathbf{f}) \widehat{=} \varphi^+(p, \mathbf{f}) \vee \Big(\bigwedge_{0 \leq j \leq N_{p,\mathbf{f}}} L_{\mathbf{f}}^j p = 0 \Big).$$

Lemma 4. *For an SAS D defined by (9) and a PVF $\mathbf{f}$, we have*

$$\mathrm{IvIn}_{\mathbf{f}}(D) = \bigvee_{m=1}^{M} \bigwedge_{l=1}^{L_m} \mathrm{IvIn}_{\mathbf{f}}(p_{ml} \triangleright 0).$$

[4] In the presentation below, we adopt the convention that $\bigvee_{i \in \emptyset} \eta_i = \mathit{false}$ and $\bigwedge_{i \in \emptyset} \eta_i = \mathit{true}$, where η_i is a logical formula.

Now the main result on automatic SCI generation can be stated as follows.

Theorem 9 (Main Result). *A semi-algebraic template $P(\mathbf{u},\mathbf{x})$ defined by*

$$\bigvee_{k=1}^{K}\left(\bigwedge_{j=1}^{j_k} p_{kj}(\mathbf{u}_{kj},\mathbf{x})\geq 0 \quad \wedge \bigwedge_{j=j_k+1}^{J_k} p_{kj}(\mathbf{u}_{kj},\mathbf{x})>0\right)$$

is a CI of the PCCDS $\big(D,\mathbf{f}\big)$ with

$$D\widehat{=}\bigvee_{m=1}^{M}\left(\bigwedge_{l=1}^{l_m} p_{ml}(\mathbf{x})\geq 0 \quad \wedge \bigwedge_{l=l_m+1}^{L_m} p_{ml}(\mathbf{x})>0\right),$$

if and only if $\mathbf{u}$ *satisfies*

$$\forall\mathbf{x}.\Big(\big(P\wedge D\wedge\Phi_D\rightarrow\Phi_P\big)\wedge\big(\neg P\wedge D\wedge\Phi_D^{\mathrm{Iv}}\rightarrow\neg\Phi_P^{\mathrm{Iv}}\big)\Big),$$

where

$$\begin{aligned}
\Phi_D &\widehat{=} \bigvee_{m=1}^{M}\left(\bigwedge_{l=1}^{l_m}\psi_0^+(p_{ml},\mathbf{f})\wedge\bigwedge_{l=l_m+1}^{L_m}\psi^+(p_{ml},\mathbf{f})\right),\\
\Phi_P &\widehat{=} \bigvee_{k=1}^{K}\left(\bigwedge_{j=1}^{j_k}\psi_0^+(p_{kj},\mathbf{f})\wedge\bigwedge_{j=j_k+1}^{J_k}\psi^+(p_{kj},\mathbf{f})\right),\\
\Phi_D^{\mathrm{Iv}} &\widehat{=} \bigvee_{m=1}^{M}\left(\bigwedge_{l=1}^{l_m}\varphi_0^+(p_{ml},\mathbf{f})\wedge\bigwedge_{l=l_m+1}^{L_m}\varphi^+(p_{ml},\mathbf{f})\right),\\
\Phi_P^{\mathrm{Iv}} &\widehat{=} \bigvee_{k=1}^{K}\left(\bigwedge_{j=1}^{j_k}\varphi_0^+(p_{kj},\mathbf{f})\wedge\bigwedge_{j=j_k+1}^{J_k}\varphi^+(p_{kj},\mathbf{f})\right),
\end{aligned}$$

with $\psi^+(p,\mathbf{f}),\psi_0^+(p,\mathbf{f}),\varphi^+(p,\mathbf{f}),\varphi_0^+(p,\mathbf{f})$ defined in Lemma 1 and 3 respectively.

Please refer to [54] for the proofs of the above results.

Example 5. Let $\mathbf{f}(x,y)=(-2y,x^2)$ and $D\widehat{=}\mathbb{R}^2$. Take a template: $P(\mathbf{u},\mathbf{x})\widehat{=}x-a\geq 0\vee y-b>0$ with $\mathbf{u}=(a,b)$. By Theorem 9, P is an SCI of $(D,\mathbf{f})$ if and only if a,b satisfy

$$\forall x\forall y.\Big((P\rightarrow\zeta)\wedge(\neg P\rightarrow\neg\xi)\Big),^{5}$$

[5] Note that in Theorem 9 φ_D and φ_D^{Iv} are trivially *true* when D equals $\mathbb{R}^n$.

where

$$\begin{aligned}
\zeta \widehat{=} & (x-a>0) \vee (x-a=0 \wedge -2y>0) \\
& \vee (x-a=0 \wedge -2y=0 \wedge -2x^2 \geq 0) \\
& \vee (y-b>0) \vee (y-b=0 \wedge x^2>0) \\
& \vee (y-b=0 \wedge x^2=0 \wedge -4yx>0) \\
& \vee (y-b=0 \wedge x^2=0 \wedge -4yx=0 \wedge 8y^2-4x^3>0) \\
\xi \widehat{=} & (x-a>0) \vee (x-a=0 \wedge -2y<0) \\
& \vee (x-a=0 \wedge -2y=0 \wedge -2x^2 \geq 0) \\
& \vee (y-b>0) \vee (y-b=0 \wedge x^2<0) \\
& \vee (y-b=0 \wedge x^2=0 \wedge -4yx>0) \\
& \vee (y-b=0 \wedge x^2=0 \wedge -4yx=0 \wedge 8y^2-4x^3<0)
\end{aligned}$$

In addition, we require the set $x+y \geq 0$ to be contained in P. By applying QE, we get $a+b \leq 0 \wedge b \leq 0$. Let $a=-1$ and $b=-0.5$, and we obtain an SCI $P \widehat{=} x+1 \geq 0 \vee y+0.5>0$, which is shown in IV of Figure 2.

3.6 SGI Generation

Now the method for generating SGIs for a PHA $\mathcal{H} \widehat{=} (Q, X, f, D, E, G, R, \Xi)$ can be stated as the following steps.

I. Predefine a familiy of semi-algebraic templates $I_q(\mathbf{u}, \mathbf{x})$[6] with degree bound d for each $q \in Q$, as the SCI to be generated at mode q.

II. Translate conditions for the family of $I_q(\mathbf{u}, \mathbf{x})$ to be a GI of $\mathcal{H}$, i.e.
   - $\Xi_q \subseteq I_q$ for all $q \in Q$;
   - for any $e=(q,q') \in E$, if $\mathbf{x} \in I_q \cap G_e$, then $\mathbf{x}' = R_e(\mathbf{x}) \in I_{q'}$;
   - for any $q \in Q$, I_q is a CI of $(D_q, \mathbf{f}_q)$

   into a set of first-order real arithmetic formulas, i.e.

   (1) $\forall \mathbf{x}. \big(\Xi_q \rightarrow I_q(\mathbf{u}, \mathbf{x})\big)$ for all $q \in Q$;

   (2) $\forall \mathbf{x}, \mathbf{x}'. \big(I_q(\mathbf{u}, \mathbf{x}) \wedge G_e \wedge \mathbf{x}' = R_e(\mathbf{x}) \rightarrow I_{q'}(\mathbf{u}, \mathbf{x}')\big)$ for all $q \in Q$ and all $e=(q,q') \in E$, where $\mathbf{x}'$ is a vector of new variables with the same dimension as $\mathbf{x}$, and $I_{q'}(\mathbf{u}, \mathbf{x}')$ is obtained by substituting $\mathbf{x}'$ for $\mathbf{x}$ in $I_{q'}(\mathbf{u}, \mathbf{x})$;

   (3) $\forall \mathbf{x}. \big((I_q(\mathbf{u}, \mathbf{x}) \wedge D_q \wedge \Phi_{D_q} \rightarrow \Phi_{I_q}) \wedge (\neg I_q(\mathbf{u}, \mathbf{x}) \wedge D_q \wedge \Phi^{\mathrm{Iv}}_{D_q} \rightarrow \neg \Phi^{\mathrm{Iv}}_{I_q})\big)$ for each $q \in Q$, as defined in Theorem 9.

   Regarding the verification of a safety property $\mathcal{S}$, there may be a fourth set of formulas:

   (4) $\forall \mathbf{x}. (I_q(\mathbf{u}, \mathbf{x}) \longrightarrow S_q)$ for all $q \in Q$.

[6] Templates at different modes have different sets of parameters. Here we simply collect all the parameters together into a w-tuple $\mathbf{u}$.

III. Take the conjunction of all the formulas in Step 2 and apply QE to get a QFF $\phi(\mathbf{u})$. Then choose a specific $\mathbf{u}_0$ from $\phi(\mathbf{u})$ with a tool like Z3 [26], and the set of instantiations $I_{q,\mathbf{u}_0}(\mathbf{x})$ form a GI of $\mathcal{H}$.

The above method is *relatively complete* with respect to the predefined set of templates, that is, if there exist SGIs in the form of the predefined templates then we are able to find one.

Example 6. The Thermostat example taken from [6] can be described by the HA in Fig. 3. The system has three modes: Cool ($q_{\rm cl}$), Heat ($q_{\rm ht}$) and Check ($q_{\rm ck}$); and 2 continuous variables: temperature T and timer clock c. All the domains, guards, reset functions and continuous dynamics are included in Fig. 3. We want to verify that under the initial condition $\Xi_{\mathcal{H}} \widehat{=} \{q_{\rm ht}\} \times X_0$ with $X_0 \widehat{=} c = 0 \wedge 5 \leq T \leq 10$, the safety property $S \widehat{=} T \geq 4.5$ is satisfied at all modes.

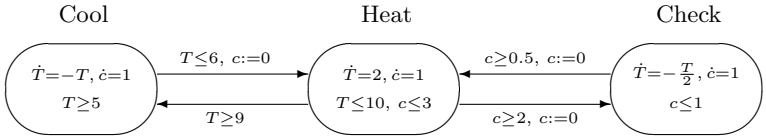

Fig. 3. A hybrid automaton describing the Thermostat system

Using the above SGI generation method, the following set of templates are predefined:

- $I_{q_{\rm ht}} \widehat{=} T + a_1 c + a_0 \geq 0 \wedge c \geq 0$;
- $I_{q_{\rm cl}} \widehat{=} T + a_2 \geq 0$;
- $I_{q_{\rm ck}} \widehat{=} T \geq a_3 c^2 - 4.5c + 9 \wedge c \geq 0 \wedge c \leq 1$

with indeterminates a_0, a_1, a_2 and a_3. By deriving verification conditions and applying QE we get the following constraint on a_0, a_1, a_2, a_3:

$$10a_3 - 9 \leq 0 \wedge 2a_3 - 1 \geq 0 \wedge a_1 + 2 = 0 \wedge a_0 + 2a_1 + 9 = 0 \wedge a_2 - a_0 = 0\,.$$

By choosing $a_0 = -5, a_1 = -2, a_2 = -5, a_3 = \frac{1}{2}$, the following SGI instantiation is obtained

- $I_{q_{\rm ht}} \widehat{=} T \geq 2c + 5 \wedge c \geq 0$;
- $I_{q_{\rm cl}} \widehat{=} T \geq 5$;
- $I_{q_{\rm ck}} \widehat{=} 2T \geq c^2 - 9c + 18 \wedge c \geq 0 \wedge c \leq 1$,

and the safety property is successfully verified.

4 Switching Controller Synthesis

4.1 Problem Description

In verification problems, a given hybrid system is proved to satisfy a desired safety (or other) property. A *synthesis* problem is harder given that the focus is on *designing* a hybrid system that will satisfy a safety requirement, reach a given set of states, or meet an optimality criterion, or a desired combination of these requirements.

In this section we talk about the synthesis of *switching controllers* for hybrid systems with safety requirements. That is, given a hybrid system and a safety requirement, we aim to identify a subset of continuous states from each original transition guard, such that if only at these states is mode switching allowed, then the system can run forever without violating the required safety property.

The formal definition of the switching controller synthesis problem w.r.t. safety requirement can be given in the way of [10]. Note that the specification of hybrid automata has been simplified by assuming that the initial condition is identical with the domain, and all reset functions are identity mappings.

Problem 1 (Switching Controller Synthesis for Safety). Given a hybrid automaton $\mathcal{H} = (Q, X, f, D, E, G)$ and a safety property $\mathcal{S}$, find a hybrid automaton $\mathcal{H}' = (Q, X, f, D', E, G')$ such that

(r1) Refinement: for any $q \in Q$, $D'_q \subseteq D_q$, and for any $e \in E$, $G'_e \subseteq G_e$;
(r2) Safety: for any $(q, \mathbf{x}) \in \mathcal{R}_{\mathcal{H}'}$, $\mathbf{x} \in S_q$;
(r3) Non-blocking: $\mathcal{H}'$ is non-blocking.

If such $\mathcal{H}'$ exists, then $\mathcal{SC} \widehat{=} \{G'_e \subseteq \mathbb{R}^n \mid e \in E\}$ is a *switching controller* satisfying the safety requirement $\mathcal{S}$, and $D_{\mathcal{H}'} \widehat{=} \bigcup_{q \in Q}(\{q\} \times D'_q)$ is the *controlled invariant set* rendered by $\mathcal{SC}$.

In the following, the theory and techniques on continuous invariant generation developed in Section 3 will be exploited to solve Problem 1.

4.2 A Synthesis Procedure Based on CI Generation

To solve Problem 1 amounts to refining the domains and guards of $\mathcal{H}$ by removing so-called *bad* states. A state $(q, \mathbf{x}) \in D_{\mathcal{H}}$ is *bad* if the hybrid trajectory starting from $(q, \mathbf{x})$ either blocks $\mathcal{H}$ or violates $\mathcal{S}$; otherwise if the trajectory starting from $(q, \mathbf{x})$ can either be extended to infinite time or execute infinitely many discrete transitions while maintaining $\mathcal{S}$, then $(q, \mathbf{x})$ is called a *good* state. By Definition 13, the set of good states of $\mathcal{H}$ can be approximated appropriately using CIs, which results in the following solution to Problem 1.

Theorem 10. *Let $\mathcal{H}$ and $\mathcal{S}$ be the same as in Problem 1. Suppose D'_q is a closed subset of $\mathbb{R}^n$ for all $q \in Q$ and $\bigcup_{q \in Q} D'_q$ is non-empty. If we have*

(c1) for all $q \in Q$, $D'_q \subseteq D_q \cap S_q$; and

(c2) for all $q \in Q$, D'_q *is a CI of* $(H_q, \mathbf{f}_q)$, *where*

$$H_q \widehat{=} \Big(\bigcup_{e=(q,q')\in E} G'_e \Big)^c \quad \textit{with} \quad G'_e \widehat{=} G_e \cap D'_{q'} \,,$$

then the HA $\mathcal{H}' = (Q, X, f, D', E, G')$ *is a solution to Problem 1.*

Please refer to [49] for the proof of this theorem.

Intuitively, by (c1), D'_q is a refinement of D_q and is also contained in the safe region S_q, thus guaranteeing (r1) and (r2) of Problem 1; by (c2), any trajectory starting from D'_q will either stay in D'_q forever[7], or finally intersect one of the transition guards enabling jumps from q to a certain q', thus guaranteeing (r3) of Problem 1.

Based on Theorem 10, the following template-based method for synthesizing switching controllers for PHA with semi-algebraic safety requirement is proposed, by incorporating the automatic SCI generation method in Section 3.4 and 3.5.

(s1) **Template Assignment:** assign to each $q \in Q$ a semi-algebraic template specifying D'_q, which will be required (see step (s3)) to be a refinement of D_q, as well as the CI to be generated at mode q;

(s2) **Guard Refinement:** refine guard G_e for each $e = (q, q') \in E$ by setting $G'_e \widehat{=} G_e \cap D'_{q'}$;

(s3) **Deriving Synthesis conditions:** encode (c1) and (c2) in Theorem 10 into first-order polynomial formulas; the encoding of condition (c1) is straightforward, while encoding of (c2) is based on Theorem 9;

(s4) **Constraint Solving:** apply QE to the fisrt-order formulas derived in (s3) and a QFF will be returned specifying the set of all possible values for the parameters appearing in templates;

(s5) **Parameters Instantiation:** a switching controller can be obtained by an appropriate instantiation of D'_q and G'_e such that D'_q are closed sets for all $q \in Q$, and D'_q is non-empty for at least one $q \in Q$; if such an instantiation is not found, we choose a new set of templates and go back to (s1).

In the above procedure, the method for SCI generation based on a necessary and sufficient criterion for SCIs is used as an integral component. As a result, the above controller synthesis method is relatively complete with respect to a given family of templates, thus having more possibility of discovering a switching controller.

The shape of chosen templates in (s1) determines the likelihood of success of the above procedure, as well as the complexity of QE in (s4). Next, heuristics for choosing appropriate templates will be discussed using the *qualitative analysis* proposed in [47].

[7] Actually in Theorem 10, for any mode $q \in Q$, $\mathbf{f}_q$ is required to be a *complete* vector field, that is, for any $\mathbf{x}_0 \in \mathbb{R}^n$, the solution $\mathbf{x}(\mathbf{x}_0; t)$ to $\dot{\mathbf{x}} = \mathbf{f}_q$ exists on $[0, \infty)$.

4.3 Heuristics for Predefining Templates

The key steps of the qualitative analysis used in [47] are as follows.

1. The evolution behavior (increasing or decreasing) of continuous state variables in each mode is inferred from the differential equations (using first or second order derivatives);
2. *control critical* modes, at which the maximal (or minimal) value of a continuous state variable is achieved, can be identified;
3. the safety requirement is imposed to obtain constraints on guards of transitions leading to control critical modes, and
4. then this information on transition guards is propagated to other modes.

Next, we illustrate how such an analysis helps in predefining templates for a nuclear reactor temperature control system discussed in [47].

Example 7. The nuclear reactor system consists of a reactor core and a cooling rod which is immersed into and removed out of the core periodically to keep the temperature of the core, denoted by x, in a certain range. Denote the fraction of the rod immersed into the reactor by p. Then the initial specification of this system can be represented using the hybrid automaton in Fig. 4. The goal is to synthesize a switching controller for this system with the global safety requirement that the temperature of the core lies between 510 and 550, i.e. $S_i \mathrel{\widehat{=}} 510 \leq x \leq 550$ for $i = 1, 2, 3, 4$.

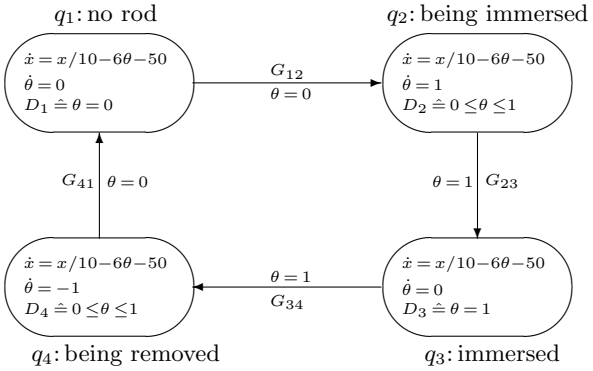

Fig. 4. Nuclear reactor temperature control

1) **Refine Domains.** Using the safety requirement, domains D_i for $i = 1, 2, 3, 4$ are refined by $D_i^s \mathrel{\widehat{=}} D_i \cap S_i$, e.g. $D_1^s \mathrel{\widehat{=}} \theta = 0 \wedge 510 \leq x \leq 550$.

2) **Infer Continuous Evolutions.** Let $l_1 \widehat{=} x/10-6\theta-50=0$ be the *zero-level* set of $\dot{x}$ and check how x and θ evolve in each mode. For example, in D_2^s, $\dot{x}>0$ on the left of l_1 and $\dot{x}<0$ on the right; since θ increases from 0 to 1, x first increases then decreases and achieves maximal value when crossing l_1. (See Fig. 5.)
3) **Identify Critical Control Modes.** By 2), q_2 and q_4 are critical control modes at which the evolution direction of x changes, and thus maximal (or minimal) value of x is achieved.
4) **Generate Control Points.** By 3), we can get a control point $E(5/6, 550)$ at q_2 by taking the intersection of l_1 and the safety upper bound $x=550$; and $F(1/6, 510)$ at q_4 is obtained by taking the intersection of l_1 and the safety lower bound $x=510$.
5) **Propagate Control Points.** E is backward propagated to $A(0,a)$ using the trajectory $\widehat{AE}$ through E defined by $\mathbf{f}_{q_2}$, and then to $C(1,c)$ using the trajectory $\widehat{CA}$ through A defined by $\mathbf{f}_{q_4}$; similarly, by propagating F we get D and B.
6) **Construct Templates.** For brevity, we only show how to construct D_2'. Intuitively, $\theta=0$, $\theta=1$, $\widehat{AE}$ and $\widehat{BD}$ form the boundaries of D_2'. In order to get a semi-algebraic template, we need to fit $\widehat{AE}$ and $\widehat{BD}$ (which are generally not polynomial curves) by polynomials using points A, E and B, D respectively. By the inference of 2), $\widehat{AE}$ has only one extreme point (also the maximum point) E in D_2^s, and is tangential to $x=550$ at E. A simple algebraic curve that can exhibit a shape similar to $\widehat{AE}$ is the parabola through A, E opening downward with $l_2 \widehat{=} \theta=\frac{5}{6}$ the axis of symmetry. Therefore to minimize the degree of terms appearing in templates, we do not resort to polynomials with degree greater than 2. This parabola can be computed using the coordinates of A, E as: $x-550-\frac{36}{25}(a-550)(\theta-\frac{5}{6})^2=0$, with a the parameter to be determined.

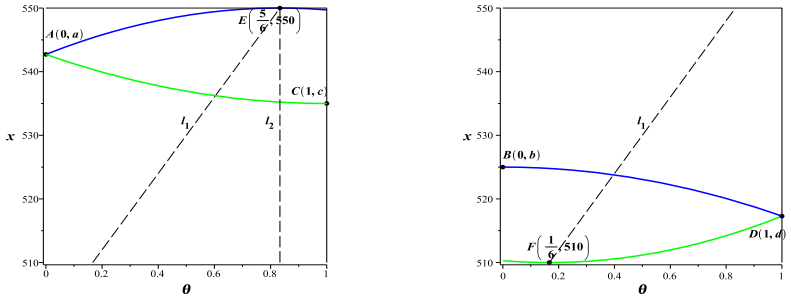

Fig. 5. Control points propagation

Through the above analysis, we generate the following templates:

- $D'_1 \widehat{=} \theta = 0 \wedge 510 \leq x \leq a$;
- $D'_2 \widehat{=} 0 \leq \theta \leq 1 \wedge x - b \geq \theta(d-b) \wedge x - 550 - \frac{36}{25}(a-550)(\theta - \frac{5}{6})^2 \leq 0$;
- $D'_3 \widehat{=} \theta = 1 \wedge d \leq x \leq 550$;
- $D'_4 \widehat{=} 0 \leq \theta \leq 1 \wedge x - a \leq \theta(c-a) \wedge x - 510 - \frac{36}{25}(d-510)(\theta - \frac{1}{6})^2 \geq 0$,

in which a, b, c, d are parameters satisfying

$$510 \leq b \leq a \leq 550 \wedge 510 \leq d \leq c \leq 550\,.$$

Note that without qualitative analysis, a single generic *quadratic* polynomial over θ and x would require $\binom{2+2}{2} = 6$ parameters.

Based on the synthesis procedure (s1)–(s5) presented in Section 4.2, we show below how to synthesize a switching controller for the system in Example 7 step by step.

(s1) The four templates are defined as the above D'_i for $1 \leq i \leq 4$.

(s2) The four guards are refined by $G'_{ij} \widehat{=} G_{ij} \cap D'_j$ and then simplified to:
- $G'_{12} \widehat{=} \theta = 0 \wedge b \leq x \leq a$;
- $G'_{23} \widehat{=} \theta = 1 \wedge d \leq x \leq 550$;
- $G'_{34} \widehat{=} \theta = 1 \wedge d \leq x \leq c$;
- $G'_{41} \widehat{=} \theta = 0 \wedge 510 \leq x \leq a$.

(s3) The derived synthesis condition, which is a first-order polynomial formula in the form of $\phi \widehat{=} \forall x \forall \theta . \varphi(a, b, c, d, x, \theta)$, is not included here due to its big size.

(s4) By applying QE to ϕ we get the following sample solution to the parameters:

$$a = \frac{6575}{12} \wedge b = \frac{4135}{8} \wedge c = \frac{4345}{8} \wedge d = \frac{6145}{12}\ . \tag{10}$$

(s5) Instantiate D'_i and G'_{ij} by (10). It is obvious that all D'_i are nonempty closed[8] sets. According to Theorem 9, we get a switching controller guaranteeing safety property for the nuclear reactor system, i.e.
- $G'_{12} \widehat{=} \theta = 0 \wedge 4135/8 \leq x \leq 6575/12$;
- $G'_{23} \widehat{=} \theta = 1 \wedge 6145/12 \leq x \leq 550$;
- $G'_{34} \widehat{=} \theta = 1 \wedge 6145/12 \leq x \leq 4345/8$;
- $G'_{41} \widehat{=} \theta = 0 \wedge 510 \leq x \leq 6575/12$.

In [47], an upper bound $x = 547.97$ for G_{12} and a lower bound $x = 512.03$ for G_{34} are obtained by solving the differential equations at mode q_2 and q_4 respectively. By (10), the corresponding bounds generated here are $x \leq \frac{6575}{12} = 547.92$ and $x \geq \frac{6145}{12} = 512.08$.

As should be evident from the above discussion, in contrast to [47], where differential equations are solved to get closed-form solutions, here good approximate results are obtained without requiring closed-form solutions. This indicates that the controller synthesis approach based on CI generation should work well for hybrid automata where differential equations for modes need not have closed form solutions.

[8] Actually all D'_i become closed sets naturally by the construction of templates, in which only $\geq, \leq, =$ relations appear conjunctively.

4.4 Synthesis of Optimal Controllers

Most of the discussion so far on switching controller synthesis is based on meeting the safety requirements. As a result, there is still considerable flexibility left in designing controllers to meet other objectives. One important criterion for further refinement of controllers is *optimality*, i.e. to optimize a *reward/penalty* function that reflects the performance of the controlled system.

The optimal switching controller synthesis problem studied in this section can be stated as follows.

Problem 2. Suppose $\mathcal{H}$ is a hybrid automaton whose transition guards are not determined but specified by a vector of parameters $\mathbf{u}$. Associated with $\mathcal{H}$ is an *objective function* g in $\mathbf{u}$. The task is to determine values of $\mathbf{u}$, or a relation over $\mathbf{u}$, such that $\mathcal{H}$ can take discrete jumps at desired conditions, thus guaranteeing

1) a safety requirement $\mathcal{S}$ is satisfied; and
2) an optimization goal $\mathcal{G}$, possibly

$$\min_{\mathbf{u}} g(\mathbf{u}),\ \max_{\mathbf{u}_2}\min_{\mathbf{u}_1} g(\mathbf{u})\,,\ \text{or}\ \min_{\mathbf{u}_3}\max_{\mathbf{u}_2}\min_{\mathbf{u}_1} g(\mathbf{u})^{9}\ \text{is achieved.}$$

The determined values of $\mathbf{u}$ or relations over $\mathbf{u}$ are called the *optimal switching controller*.

If $\mathcal{H}$ is a PHA and $\mathcal{S}$ is a semi-algebraic safety property, then Problem 2 can be solved by following the steps (s1)–(s4) in Section 4.2 and then solving an optimization problem with objective $\mathcal{G}$. In particular, if g is a polynomial function, then the optimization problem can also be encoded into first-order polynomial formulas and then solved by QE.

In detail, the approach for solving the optimal controller synthesis problem can be described as the following steps.

Step 1. *Derive constraint $\mathcal{D}(\mathbf{u})$ on $\mathbf{u}$ from the safety requirements of the system.*

The reachable set $R_{\mathcal{H}}$ (parameterized by $\mathbf{u}$) is either computed exactly, or approximated using SCIs (with $\mathbf{u}$ and possibly others as parameters). Then the safety requirement $\mathcal{S}$ is imposed to derive constraint on $\mathbf{u}$ using QE.

Step 2. *Encode the optimization problem $\mathcal{G}$ over constraint $\mathcal{D}(\mathbf{u})$ into a quantified first-order polynomial formula $\mathbf{Qu}.\varphi(\mathbf{u}, z)$, where z is a fresh variable.*

The encoding is based on the following proposition, in which all the aforementioned optimization objectives are discussed together.

Proposition 4. *Suppose $g_1(\mathbf{u}_1)$, $g_2(\mathbf{u}_1, \mathbf{u}_2)$, $g_3(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$ are polynomials, and $\mathcal{D}_1(\mathbf{u}_1)$, $\mathcal{D}_2(\mathbf{u}_1, \mathbf{u}_2)$, $\mathcal{D}_3(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$ are nonempty compact (i.e. bounded closed) SASs. Then there exist c_1, c_2, $c_3 \in \mathbb{R}$ s.t.*

[9] The elements of $\mathbf{u}$ are divided into groups $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \ldots$ according to their roles in $\mathcal{G}$.

$$\exists \mathbf{u}_1.(\mathcal{D}_1 \wedge g_1 \leq z) \iff z \geq c_1\,, \tag{11}$$

$$\forall \mathbf{u}_2.\Big(\exists \mathbf{u}_1.\mathcal{D}_2 \longrightarrow \exists \mathbf{u}_1.(\mathcal{D}_2 \wedge g_2 \leq z)\Big) \iff z \geq c_2\,, \tag{12}$$

$$\exists \mathbf{u}_3.\Big((\exists \mathbf{u}_1\mathbf{u}_2.\mathcal{D}_3) \wedge \forall \mathbf{u}_2.\Big(\exists \mathbf{u}_1.\mathcal{D}_3 \longrightarrow \exists \mathbf{u}_1.(\mathcal{D}_3 \wedge g_3 \leq z)\Big)\Big) \iff z \rhd c_3, \tag{13}$$

where $\rhd \in \{>, \geq\}$*, and* c_1, c_2, c_3 *satisfy*

$$c_1 = \min_{\mathbf{u}_1} g_1(\mathbf{u}_1) \quad \text{over}\, \mathcal{D}_1(\mathbf{u}_1)\,, \tag{14}$$

$$c_2 = \sup_{\mathbf{u}_2} \min_{\mathbf{u}_1} g_2(\mathbf{u}_1, \mathbf{u}_2) \quad \text{over}\, \mathcal{D}_2(\mathbf{u}_1, \mathbf{u}_2)\,, \tag{15}$$

$$c_3 = \inf_{\mathbf{u}_3} \sup_{\mathbf{u}_2} \min_{\mathbf{u}_1} g_3(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3) \quad \text{over}\, \mathcal{D}_3(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)\,. \tag{16}$$

The proof of this proposition can be found in [95].

Step 3. *Apply QE to* $\mathbf{Qu}.\varphi(\mathbf{u}, z)$ *and from the result we can retrieve the optimal value of* $\mathcal{G}$ *and the corresponding optimal controller* $\mathbf{u}$*.*

Using the above procedure, the issues of synthesis, verification and optimization for hybrid systems are integrated into one elegant framework. Compared to numerical approaches, using the QE-based method, the synthesized controllers are guaranteed to be correct and better optimal optimal values can be obtained.

4.5 Oil Pump: A Case Study

We illustrate the above approach on an industrial oil pump example studied in [18].

The whole system consists of a machine, an accumulator, a reservoir and a pump. The machine consumes oil periodically out of the accumulator with a period of $20s$ (second) for one consumption cycle. The profile of consumption rate is shown in Fig. 6. The pump adds oil from the reservoir into the accumulator with power $2.2l/s$ (liter/second). There is an additional physical constraint requiring a *latency* of at least $2s$ between any two consecutive operations of the pump.

Control objective for this system is to switch on/off the pump at appropriate time points

$$0 \leq t_1 \leq t_2 \leq \cdots \leq t_k \leq t_{k+1} \leq \cdots \tag{17}$$

in order to

1) maintain the oil volume $v(t)$ in the accumulator within a safe range $[V_{\min}, V_{\max}]$ at any time, where $V_{\min} = 4.9l$, $V_{\max} = 25.1l$; and
2) minimize the average accumulated oil volume in one cycle, i.e. $\frac{1}{T}\int_{t=0}^{T} v(t)$.

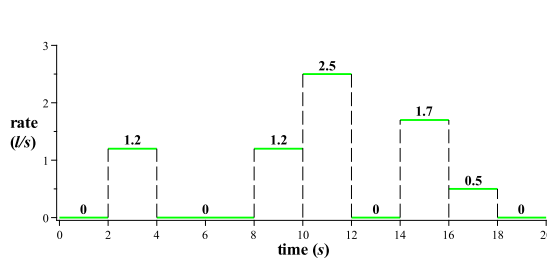

Fig. 6. Consumption rate in each cycle

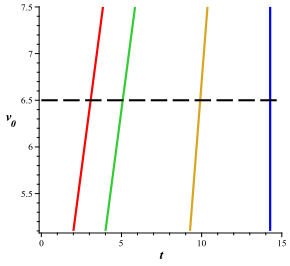

Fig. 7. Optimal switching controller for the oil pump

The second objective is important because the average oil level reflects the energy cost of the system.

Following [18], the time points to switch on/off the pump in one consumption cycle is determined by measuring the oil volume v_0 at the beginning of each cycle. Besides, it is assumed that the pump is operated (turned on/off) at most 4 times in one cycle.

The system along with the safety and optimality requirements can all be exactly modeled by first-order polynomial formulas. By applying various QE heuristics, the following results are obtained:

- The optimal switching controller is

$$t_1 = \frac{10v_0 - 25}{13} \wedge t_2 = \frac{10v_0 + 1}{13} \wedge t_3 = \frac{10v_0 + 153}{22} \wedge t_4 = \frac{157}{11} \ , \quad (18)$$

  where t_1, t_2, t_3, t_4 are the 4 time points to operate the pump in one cycle, and $v_0 \in [5.1, 7.5]$ is the measurement of the initial oil volume at the beginning of each cycle. If $v_0 = 6.5$, then by (18) the pump should be switched on at $t_1 = 40/13$, off at $t_2 = 66/13$, then on at $t_3 = 109/11$, and finally off at $t_4 = 157/11$ (dashed line in Fig. 7).
- The optimal average accumulated oil volume obtained using the strategy given by (18) is $V_{opt} = \frac{215273}{28600} = 7.53$, which is a significant improvement (over 5%) compared to the optimal value 7.95 reported in [18]. If the pump is allowed to be turned on more times, then even better controllers can be generated ($V_{opt} = 7.35$ if the pump is allowed to be turned on at most 3 times in one cycle).

More details about this case study can be found in [94,95].

5 Hybrid CSP

HCSP [37,98], which extends CSP by introducing differential equations for modelling continuous evolutions and interrupts, is a formal language for describing hybrid systems. In HCSP, exchange of data among processes is described solely by communications; no shared variable is allowed between different processes in parallel, so each process variable is local to the respective sequential component. We denote by $\mathcal{V}$ ranged over $x, y, s, \ldots$ the set of variables, and by Σ ranged over $ch, ch_1, \ldots$ the set of channels. The syntax of HCSP is given as follows:

$$
\begin{aligned}
P ::= {} & \mathbf{skip} \mid x := e \mid \text{wait } d \mid ch?x \mid ch!e \mid P; Q \mid B \to P \mid P \sqcup Q \mid [\!]_{i \in I}(ch_i * \to Q_i) \\
& \mid P^* \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq_d Q \\
& \mid \langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(ch_i * \to Q_i) \\
S ::= {} & P \mid S \| S
\end{aligned}
$$

Here $ch, ch_i \in \Sigma$, ch_i* stands for a communication event, i.e., either $ch_i?x$ or $ch_i!e$, $x, s \in \mathcal{V}$, B and e are Boolean and arithmetic expressions, d is a non-negative real constant, P, Q, Q_i are sequential processes, and S stands for a system, i.e., an HCSP process.

The intended meaning of the individual constructs is as follows:

- **skip** terminates immediately having no effect on variables.
- $x := e$ assigns the value of expression e to x and then terminates.
- wait d will keep idle for d time units keeping variables unchanged.
- $ch?x$ receives a value along channel ch and assigns it to x.
- $ch!e$ sends the value of e along channel ch. A communication takes place when both the sending and the receiving parties are ready, and may cause one side to wait.
- The sequential composition $P; Q$ behaves as P first, and if it terminates, as Q afterwards.
- The alternative $B \to P$ behaves as P if B is true; otherwise it terminates immediately.
- $P \sqcup Q$ denotes internal choice. It behaves as either P or Q, and the choice is made by the process.
- $[\!]_{i \in I}(ch_i * \to Q_i)$ denotes communication controlled external choice. I is supposed to be finite. As soon as one of the communications ch_i* takes place, the process continues as the respective guarded Q_i.
- The repetition P^* executes P for some finite number of times.
- $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$ is the continuous evolution statement (hereafter shortly *continuous*). It forces the vector s of real variables to obey the differential equations $\mathcal{F}$ as long as the boolean expression B, which defines the *domain of* s, holds, and terminates when B turns false.
- $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq_d Q$ behaves like $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$, if that continuous terminates before d time units. Otherwise, after d time units of evolution according to $\mathcal{F}$, it moves on to execute Q.

- $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(ch_i * \to Q_i)$ behaves like $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$, except that the continuous evolution is preempted as soon as one of the communications $ch_i *$ takes place, which is followed by the respective Q_i. Notice that, if the continuous part terminates before a communication from among $\{ch_i *\}_I$ occurs, then the process terminates without communicating.
- $S_1 \| S_2$ behaves as if S_1 and S_2 run independently except that all communications along the common channels connecting S_1 and S_2 are to be synchronized. The processes S_1 and S_2 in parallel can neither share variables, nor input or output channels.

Note that some primitives of CSP and timed CSP are derivable from the above syntax, e.g. $\mathbf{stop} \stackrel{\text{def}}{=} t := 0; \langle \dot{t} = 1 \& true \rangle$. Specifically, some of the constructs in the above syntax can be defined with other ones and thus are not primitive either, for instance

$$\text{wait } d \stackrel{\text{def}}{=} t := 0; \langle \dot{t} = 1 \& t < d \rangle,$$

$$[\!]_{i \in I}(ch_i * \to Q_i) \stackrel{\text{def}}{=} \mathbf{stop} \trianglerighteq [\!]_{i \in I}(ch_i * \to Q_i),$$

$$\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle \trianglerighteq_d Q \stackrel{\text{def}}{=} t := 0; \langle F(\dot{s}, s) = 0 \wedge \dot{t} = 1 \& t < d \wedge B \rangle; t \geq d \to Q.$$

Example 8. Consider the classic plant-controller example: A plant is sensed by a computer periodically (say every d time units), and receives a control (u) from the digit controller soon after the sensing. Thus, it can be modelled by the following HCSP process:

$$((\langle F(u, s, \dot{s}) = 0 \& true \rangle \trianglerighteq (c_{p2c}!s \to \mathbf{skip})); c_{c2p}?u)^* \parallel (\mathbf{wait}\ d; c_{p2c}?x; c_{c2p}!e(x))^*$$

where $\langle F(u, s, \dot{s}) = 0 \& true \rangle$ describes the behaviour of the plant. We refer this HCSP process as PLC hereafter.

In the sequel, we use $\mathcal{V}(P)$ to stand for the set of local variables and $\Sigma(P)$ for the set of channels of a process P.

5.1 Notations

In order to define the real-time behavior of HCSP processes, we use non-negative reals $\mathbb{R}^+$ to model time, and introduce a global clock *now* as a system variable to record the time in the execution of a process.

A *timed communication* is of the form $\langle ch.c, b \rangle$, where $ch \in \Sigma$, $c \in \mathbb{R}$ and $b \in \mathbb{R}^+$, representing that a communication along channel ch occurs at time b with value c transmitted. The set $\Sigma \times \mathbb{R} \times \mathbb{R}^+$ of all timed communications is denoted by $T\Sigma$. The set of all timed traces is

$$T\Sigma^*_{\leq} = \{\gamma \in T\Sigma^* \mid \text{ if } \langle ch_1.c_1, b_1 \rangle \text{ precedes } \langle ch_2.c_2, b_2 \rangle \text{ in } \gamma, \text{ then } b_1 \leq b_2\}.$$

If $X \subseteq \Sigma$, $\gamma \restriction_X$ is the projection of γ onto X.

Given two timed traces γ_1, γ_2, and $X \subseteq \Sigma$, the *alphabetized parallel* of γ_1 and γ_2 over X, denoted by $\gamma_1 \parallel_X \gamma_2$, results in a set of timed traces, defined by:

$$\langle\rangle \underset{X}{\parallel} \langle\rangle \stackrel{\text{def}}{=} \langle\rangle, \qquad \langle\rangle \underset{X}{\parallel} \gamma \stackrel{\text{def}}{=} \gamma \underset{X}{\parallel} \langle\rangle$$

$$\langle ch.a, b\rangle \cdot \gamma \underset{X}{\parallel} \langle\rangle \stackrel{\text{def}}{=} \begin{cases} \langle ch.a, b\rangle \cdot (\gamma \underset{X}{\parallel} \langle\rangle) & \text{if } ch \notin X \\ \emptyset & \text{otherwise} \end{cases}$$

$$\langle ch_1.a, t_1\rangle \cdot \gamma_1' \underset{X}{\parallel} \langle ch_2.b, t_2\rangle \cdot \gamma_2'$$

$$\stackrel{\text{def}}{=} \begin{cases} \langle ch_1.a, t_1\rangle \cdot (\gamma_1' \underset{X}{\parallel} \gamma_2') & \text{if } ch_1 = ch_2 \in X, a = b, t_1 = t_2 \\ \langle ch_1.a, t_1\rangle \cdot (\gamma_1' \underset{X}{\parallel} (\langle ch_2.b, t_2\rangle \cdot \gamma_2')) \ \cup \langle ch_2.b, t_2\rangle \cdot ((\langle ch_1.a, t_1\rangle \cdot \gamma_1') \underset{X}{\parallel} \gamma_2') & \\ \quad \text{otherwise if } ch_1, ch_2 \notin X, t_1 = t_2 & \\ \langle ch_1.a, t_1\rangle \cdot (\gamma_1' \underset{X}{\parallel} (\langle ch_2.b, t_2\rangle \cdot \gamma_2')) & \text{otherwise if } ch_1 \notin X, t_1 \le t_2 \\ \langle ch_2.b, t_2\rangle \cdot ((\langle ch_1.a, t_1\rangle \cdot \gamma_1') \underset{X}{\parallel} \gamma_2') & \text{otherwise if } ch_2 \notin X, \text{ and } t_2 \le t_1 \\ \emptyset \quad \text{otherwise} & \end{cases}$$

To model synchronization of communication events, we need to describe their readiness, and meanwhile, to record the timed trace of communications having occurred till now. Each *communication event* has the form of $\gamma.ch?$ or $\gamma.ch!$, to represent that $ch?$ (resp. $ch!$) is ready to occur, and before that the sequence of communications γ have occurred. Therefore, we introduce two system variables, rdy and tr, to represent the ready set of communication events and the timed communication trace accumulated, at each time point during process execution. In what follows, we use $\mathcal{V}^+(P)$ to denote $\mathcal{V}(P) \cup \{rdy, tr, now\}$.

For a process P, a state σ of P is an assignment to associate a value from the respective domain to each variable in $\mathcal{V}^+(P)$. Given two states σ_1 and σ_2, we say σ_1 and σ_2 are parallelable iff $Dom(\sigma_1) \cap Dom(\sigma_2) = \{rdy, tr, now\}$ and $\sigma_1(now) = \sigma_2(now)$. Paralleling them over $X \subseteq \Sigma$ results in a set of new states, denoted by $\sigma_1 \uplus \sigma_2$, any of which σ is given by

$$\sigma(v) \stackrel{\text{def}}{=} \begin{cases} \sigma_1(v) & \text{if } v \in Dom(\sigma_1) \setminus Dom(\sigma_2), \\ \sigma_2(v) & \text{if } v \in Dom(\sigma_2) \setminus Dom(\sigma_1), \\ \sigma_1(now) & \text{if } v = now, \\ \gamma, \text{ where } \gamma \in \sigma_1(tr) \underset{X}{\parallel} \sigma_2(tr) & \text{if } v = tr, \\ \sigma_1(rdy) \cup \sigma_2(rdy) & \text{if } v = rdy. \end{cases}$$

It makes no sense to distinguish any two states in $\sigma_1 \uplus \sigma_2$, so hereafter we abuse $\sigma_1 \uplus \sigma_2$ to represent any of its elements.

5.2 Operational Semantics

As mentioned above, we use *now* to record the time during process execution. A state, ranging over σ, σ_1, assigns respective value to each variable in $\mathcal{V}^+(P)$; moreover, we introduce *flow*, ranging over H, H_1, defined on a time interval, assigns a state to each point in the interval.

Each transition relation has the form of $(P, \sigma) \xrightarrow{\alpha} (P', \sigma', H)$, where P is a process, σ, σ' are states, H is a flow. It records that starting from initial

state σ, P evolves into P' and ends in state σ' and flow H, while performing event α. When the transition is discrete and thus produces a flow on an interval that contains only one point, we will write $(P,\sigma) \xrightarrow{\alpha} (P',\sigma')$ instead of $(P,\sigma) \xrightarrow{\alpha} (P',\sigma',\{\sigma(now) \mapsto \sigma'\})$. The label α represents events, which can be an internal event like skip, assignment, or a termination of a continuous *etc*, uniformly denoted by τ, or an external communication event $ch!c$ or $ch?c$, or an internal communication $ch.c$, or a time delay d that is a positive real number. We call the events but the time delay *discrete events*, and will use β to range over them. We define the dual of $ch?c$ (denoted by $\overline{ch?c}$) as $ch!c$, and vice versa, and define $comm(ch!c, ch?c)$ or $comm(ch?c, ch!c)$ as the communication $ch.c$. To make our operational semantics more expressive, we will record both the internal events and internal communications that have occurred till now in tr.

The semantics of **skip** and $x := e$ are defined as usual, except that for each, an internal event occurs. Rule (Idle) says that a terminated configuration can keep idle arbitrarily, and then evolves to itself. For input $ch?x$, the input event has to be put in the ready set if it is enabled (In-1); then it may wait for its environment for any time d during which it keeps ready (In-2), or it performs a communication and terminates, with x being assigned and tr extended by the communication, and the ready set being reduced one corresponding to the input (In-3). The semantics of output $ch!e$ is similarly defined by rules (Out-1), (Out-2) and (Out-3). The continuous evolves for d time units if B always holds within this period, during which the ready set is empty (Cont-1), and it terminates at a point when B turns out false at the point or at a right open interval (Cont-2). For communication interrupt, it evolves for d time units if none of the communications io_i is ready (IntP-1), or continues as Q_j if io_j occurs first (IntP-2); or terminates immediately when the continuous terminates (IntP-3). For $P_1 \| P_2$, we always assume that the initial states σ_1 and σ_2 are parallelable. There are four rules: both P_1 and P_2 evolve for d time units in case they can delay d time units respectively; or P_1 may progress separately on internal events or external communication events (Par-2), and the symmetric case can be defined similarly (omitted here); or they together perform a synchronized communication (Par-3); or $P_1 \| P_2$ terminates when both P_1 and P_2 terminate (Par-4). At last, the semantics for conditional, sequential, internal choice, and repetition is defined as usual.

$$(\mathbf{skip},\sigma) \xrightarrow{\tau} (\epsilon, \sigma[tr+\tau]) \qquad \text{(Skip)}$$

$$(\epsilon,\sigma) \xrightarrow{d} (\epsilon, \sigma[now \mapsto \sigma(now)+d]) \qquad \text{(Idle)}$$

$$(x := e,\sigma) \xrightarrow{\tau} (\epsilon, \sigma[x \mapsto \sigma(e), tr \mapsto \sigma(tr)\cdot\langle\tau,\sigma(now)\rangle]) \qquad \text{(Ass)}$$

$$\frac{\sigma(tr).ch? \notin \sigma(rdy)}{(ch?x,\sigma) \xrightarrow{\tau} (ch?x, \sigma[rdy \mapsto \sigma(rdy) \cup \{\sigma(tr).ch?\}])} \qquad \text{(In-1)}$$

$$\frac{\sigma(tr).ch? \in \sigma(rdy)}{(ch?x,\sigma) \xrightarrow{d} (ch?x, \sigma[now \mapsto \sigma(now)+d], H_{d,i})} \qquad \text{(In-2)}$$

$$\frac{\sigma(tr).ch? \in \sigma(rdy)}{(ch?x,\sigma) \xrightarrow{ch?b} (\epsilon,\ \sigma[x \mapsto b, tr + ch.b, rdy \mapsto \sigma(rdy)\backslash\{\sigma(tr).ch?\}]\ )} \quad \text{(In-3)}$$

$$\frac{\sigma(tr).ch! \notin \sigma(rdy)}{(ch!e,\sigma) \xrightarrow{\tau} (ch!e, \sigma[rdy \mapsto \sigma(rdy) \cup \{\sigma(tr).ch!\}])} \quad \text{(Out-1)}$$

$$\frac{\sigma(tr).ch! \in \sigma(rdy)}{(ch!e,\sigma) \xrightarrow{d} (ch!e, \sigma[now \mapsto \sigma(now) + d], H_{d,o})} \quad \text{(Out-2)}$$

$$\frac{\sigma(tr).ch! \in \sigma(rdy)}{(ch!e,\sigma) \xrightarrow{ch!\sigma(e)} (\epsilon,\ \sigma[tr + ch.\sigma(e), rdy \mapsto \sigma(rdy)\backslash\{\sigma(tr).ch!\}]\ )} \quad \text{(Out-3)}$$

$$\frac{\begin{array}{l}S(t) \text{ is a trajectory of } \mathcal{F}(\dot{s},s) = 0 \text{ s.t.}(S(0) = \sigma(s)\\ \wedge \forall t \in [0,d].(\mathcal{F}(\dot{S(t)}, S(t)) = 0 \wedge \sigma(B[s \mapsto S(t)]) = true))\end{array}}{(\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle, \sigma) \xrightarrow{d} \left(\begin{array}{l}\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle,\\ \sigma[now \mapsto \sigma(now) + d, s \mapsto S(d)], H_{d,s}\end{array}\right)} \quad \text{(Cont-1)}$$

$$\frac{\begin{array}{l}(\sigma(B) = false) \text{ or } (S(t) \text{ is a trajectory of } \mathcal{F}(\dot{s},s) = 0 \text{ s.t.}\\ \exists \varepsilon > 0.(S(0) = \sigma(s)\\ \wedge \forall t \in (0,\varepsilon].(\mathcal{F}(\dot{S(t)}, S(t)) = 0 \wedge \sigma(B[s \mapsto S(t)]) = false)))\end{array}}{(\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle, \sigma) \xrightarrow{\tau} (\epsilon, \sigma[s \mapsto \lim_{t\to 0} S(t), tr \mapsto \sigma(tr) \cdot \langle \tau, \sigma(now)\rangle])} \quad \text{(Cont-2)}$$

$$\frac{\begin{array}{l}(ch_i*; Q_i, \sigma) \xrightarrow{d} (ch_i*; Q_i, \sigma_i', H_i), \quad \forall i \in I\\ (\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle, \sigma) \xrightarrow{d} (\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle, \sigma', H)\end{array}}{\begin{array}{l}(\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(ch_i* \to Q_i), \sigma) \xrightarrow{d}\\ \qquad \left(\begin{array}{l}\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(ch_i* \to Q_i),\\ \sigma'[rdy \mapsto \cup_{i\in I}\sigma_i'(rdy)], H[rdy \mapsto \cup_{i\in I}\sigma_i'(rdy)]\end{array}\right)\end{array}} \quad \text{(IntP-1)}$$

$$\frac{(ch_j*; Q_j, \sigma) \xrightarrow{ch_j*} (Q_j, \sigma'), \exists j \in I}{(\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(ch_i* \to Q_i), \sigma) \xrightarrow{ch_j*} (Q_j, \sigma')} \quad \text{(IntP-2)}$$

$$\frac{(\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle, \sigma) \xrightarrow{\tau} (\epsilon, \sigma'))}{(\langle \mathcal{F}(\dot{s},s) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(ch_i* \to Q_i), \sigma) \xrightarrow{\tau} (\epsilon, \sigma')} \quad \text{(IntP-3)}$$

$$\frac{\begin{array}{l}(P_1,\sigma_1) \xrightarrow{d} (P_1', \sigma_1', H_1), \quad (P_2,\sigma_2) \xrightarrow{d} (P_2', \sigma_2', H_2),\\ \forall ch \in \Sigma(P_1) \cap \Sigma(P_2).\neg((P_1, \sigma_1 \uplus \sigma_2) \xrightarrow{ch*} \wedge (P_2, \sigma_1 \uplus \sigma_2) \xrightarrow{\overline{ch*}})\end{array}}{(P_1 \parallel P_2, \sigma_1 \uplus \sigma_2) \xrightarrow{d} (P_1' \parallel P_2', (\sigma_1' \uplus \sigma_2'), H_1 \uplus H_2)} \quad \text{(Par-1)}$$

$$\frac{(P_1,\sigma_1) \xrightarrow{\beta} (P_1', \sigma_1'), \quad \Sigma(\beta) \notin \Sigma(P_1) \cap \Sigma(P_2)}{(P_1 \parallel P_2, \sigma_1 \uplus \sigma_2) \xrightarrow{\beta} (P_1' \parallel P_2, \sigma_1' \uplus \sigma_2)} \quad \text{(Par-2)}$$

$$\frac{(P_1,\sigma_1) \xrightarrow{ch*} (P_1', \sigma_1'), \quad (P_2,\sigma_2) \xrightarrow{\overline{ch*}} (P_2', \sigma_2'),}{(P_1 \parallel P_2, \sigma_1 \uplus \sigma_2) \xrightarrow{comm(ch*,\overline{ch*})} (P_1' \parallel P_2', \sigma_1' \uplus \sigma_2')} \quad \text{(Par-3)}$$

$$(\epsilon \parallel \epsilon, \sigma_1 \uplus \sigma_2) \xrightarrow{\tau} (\epsilon, \sigma_1 \uplus \sigma_2) \quad \text{(Par-4)}$$

$$\frac{\sigma(B) = \mathit{true}}{(B \to P, \sigma) \xrightarrow{\tau} (P, \sigma[tr + \tau])} \quad \text{(Cond-1)} \qquad \frac{\sigma(B) = \mathit{false}}{(B \to P, \sigma) \xrightarrow{\tau} (\epsilon, \sigma[tr + \tau])} \quad \text{(Cond-2)}$$

$$\frac{(P, \sigma) \xrightarrow{\alpha} (P', \sigma', H) \quad P' \neq \epsilon}{(P; Q, \sigma) \xrightarrow{\alpha} (P'; Q, \sigma', H)} \quad \text{(Seq-1)} \qquad \frac{(P, \sigma) \xrightarrow{\alpha} (\epsilon, \sigma', H)}{(P; Q, \sigma) \xrightarrow{\alpha} (Q, \sigma', H)} \quad \text{(Seq-2)}$$

$$(P \sqcup Q, \sigma) \xrightarrow{\tau} (P, \sigma[tr + \tau]) \quad \text{(IntC-1)} \qquad (P \sqcup Q, \sigma) \xrightarrow{\tau} (Q, \sigma[tr + \tau]) \quad \text{(IntC-2)}$$

$$\frac{(P, \sigma) \xrightarrow{\alpha} (P', \sigma', H) \quad P' \neq \epsilon}{(P^*, \sigma) \xrightarrow{\alpha} (P'; P^*, \sigma', H)} \quad \text{(Rep-1)} \qquad \frac{(P, \sigma) \xrightarrow{\alpha} (\epsilon, \sigma', H)}{(P^*, \sigma) \xrightarrow{\alpha} (P^*, \sigma', H)} \quad \text{(Rep-2)}$$

$$(P^*, \sigma) \xrightarrow{\tau} (\epsilon, \sigma[tr + \tau]) \quad \text{(Rep-3)}$$

where for an internal or communication event β, $\sigma[tr + \beta]$ stands for $\sigma[tr \mapsto \sigma(tr) \cdot \langle \beta, \sigma(now) \rangle]$, and the flow $H_{d,i}$ (or $H_{d,o}$) is defined over time interval $[\sigma(now), \sigma(now) + d]$, such that for any t in the domain, $H_{d,i}(t) = \sigma[now \mapsto t]$ (or $H_{d,o}(t) = \sigma[now \mapsto t]$); and the flow $H_{d,s}$ is defined over time interval $[\sigma(now), \sigma(now) + d]$ such that for any $t \in [\sigma(now), \sigma(now) + d]$, $H_{d,s}(t) = \sigma[now \mapsto t, s \mapsto S(t - \sigma(now))]$, where $S(\cdot)$ is the trajectory as defined in the rule. For any t in the domain, $H_1 \uplus H_2(t) = H_1(t) \uplus H_2(t)$.

Given two flows H_1 and H_2 defined on $[r_1, r_2]$ and $[r_2, r_3]$ respectively, we define the *concatenation* $H_1^\frown H_2$ as the flow defined on $[r_1, r_3]$ such that $H_1^\frown H_2(t)$ is equal to $H_1(t)$ if $t \in [r_1, r_2)$, and $H_2(t)$ if $t \in [r_2, r_3)$. Given a process P and an initial state σ_0, if we have the following sequence of transitions:

$$\begin{aligned}
&(P, \sigma_0) \xrightarrow{\alpha_0} (P_1, \sigma_1, H_1) \\
&(P_1, \sigma_1) \xrightarrow{\alpha_1} (P_2, \sigma_2, H_2) \\
&\dots \\
&(P_{n-1}, \sigma_{n-1}) \xrightarrow{\alpha_{n-1}} (P_n, \sigma_n, H_n)
\end{aligned}$$

then we define the sequence $H_1^\frown \dots ^\frown H_n$ as a *flow* from P_1 to P_n starting from σ_0, and write $(P, \sigma_0) \xrightarrow{\alpha_0 \cdots \alpha_{n-1}} (P_n, \sigma_n, H_1^\frown \dots ^\frown H_n)$ as an abbreviation of the above transition sequence; and meanwhile, define the sequence $B_1^\frown \dots ^\frown B_n$ as a *behavior* from P_1 to P_n starting from σ_0, where B_i is H_i if H_i is not empty, empty otherwise if H_i is empty but H_{i+1} is not, σ_i otherwise. Thus, a flow records for each time point the rightmost state, while a behavior records for each time point all the discrete states that occur in execution. Especially, when P_n is ϵ, we will call them *complete flow* and *complete behavior* of P with respect to σ_0 respectively.

6 Hybrid Hoare Logic

HHL was first proposed in [55], which is an extension of Hoare logic to hybrid system, used to specify and reason about hybrid systems modelled by HCSP. The assertion logic of HHL consists of two parts: the first-order logic and Duration Calculus (DC) [97,96]. The former is used to specify discrete events, represented by *pre-* and *post-condition*, while the latter is used to specify continuous evolution. In HHL, a hybrid system is modelled by an HCSP process. So, the proof system of HHL consists of the following three parts: axioms and inference rules for the first-order logic, axioms and inference rules for DC, and axioms and inference rules for the constructs of HCSP. A theorem prover of the logic based on Isabelle/HOL has been implemented, and applied to model and specify Chinese High-Speed Train Control System at Level 3 (CTCS-3) [99].

However, the version of HHL given in [55] can only be used to deal with closed systems, as it lacks compositionality and therefore cannot cope with open systems. Recently, some attempts to define a compositional proof system are undertaken [86,36,93].

Here, we present a revised version of HHL given in[55].

6.1 History Formulas

As indicated before, we will use a subset of DC formulas to record execution history of HCSP processes. The formulas in this subset are denoted as HF (*history formula*) and given as follows.

$$HF ::= \ell < T \mid \ell = T \mid \ell > T \mid \lceil S \rceil^0 \mid \neg HF \mid HF_1^\frown HF_2 \mid HF_2 \vee HF_2$$

where ℓ stands for interval length, $T \in \mathbb{R}^+$ is a constant, and S is a state expression, which is a first order formula of $\mathcal{V}(P)$ interpreted as a Boolean function over the time domain, defined by

$$S ::= 1 \mid 0 \mid R(e_1, \ldots, e_n) \mid \neg S \mid S_1 \vee S_2$$

where $R(e_1, \ldots, e_n)$ is a n-ary predicate over expressions $e_1, \ldots, e_n$, normally of the form $p(x_1, \ldots, x_n) \rhd 0$ with $\rhd \in \{\geq, >, =, \neq, \leq, <\}$ and $p(x_1, \ldots, x_n)$ a polynomial in $x_1, \ldots, x_n$.

Informally, the above formulas can be understood as follows:

- $\ell < T$ (resp. $\ell = T$, $\ell > T$) means the length of the reference interval is less than (resp. equal to, greater than) T;
- $\lceil S \rceil^0$ means that the state S is satisfied at the reference point interval, i.e., the considered time point;
- $HF_1^\frown HF_2$ says that the reference interval can be split into two parts such that HF_1 is satisfied on the first segment, while HF_2 holds on the second;
- The logical connectives can be understood in the standard way.

$\lceil S \rceil$ is an abbreviation of $\neg(true^\frown\lceil\neg S\rceil^0{}^\frown\ell > 0)$, which means S holds everywhere on a considered interval, except for its right endpoint. Obviously, we have

$$\begin{array}{ll} false \Leftrightarrow (\ell < 0) & true \Leftrightarrow (\ell = 0) \vee (\ell > 0) \Leftrightarrow \neg(\ell = 0) \vee \lceil S \rceil \\ \lceil S \rceil^\frown\lceil S \rceil \Leftrightarrow \lceil S \rceil & \lceil S \rceil^\frown(\ell = 0) \Leftrightarrow \lceil S \rceil \Leftrightarrow (\ell = 0)^\frown\lceil S \rceil \end{array}$$

In addition, given a history formula HF, we use $HF^<$ to denote the internal of HF, meaning that HF holds on the interval derived from the considered interval by excluding its endpoint. $HF^<$ can be formally defined as follows:

$$\begin{array}{ll} (\ell < T)^< & \stackrel{\text{def}}{=} (\ell < T) \\ (\ell = T)^< & \stackrel{\text{def}}{=} (\ell = T) \\ (\ell > T)^< & \stackrel{\text{def}}{=} \ell > T \\ (\lceil S \rceil^0)^< & \stackrel{\text{def}}{=} \ell = 0 \\ \lceil S \rceil^< & \stackrel{\text{def}}{=} \lceil S \rceil \\ (HF_1^\frown HF_2)^< & \stackrel{\text{def}}{=} (HF_1)^<{}^\frown(HF_2)^< \\ (HF_1 \wedge HF_2)^< & \stackrel{\text{def}}{=} (HF_1)^< \wedge (HF_2)^< \\ (HF_1 \vee HF_2)^< & \stackrel{\text{def}}{=} (HF_1)^< \vee (HF_2)^< \end{array}$$

Formally, given a state σ, a state expression S is interpreted as

$$\begin{array}{rl} \sigma(1) = 1 & \\ \sigma(0) = 0 & \\ \sigma(R(e_1, \ldots, e_n)) = & \begin{cases} 1, & \text{if } R(\sigma(e_1), \ldots, \sigma(e_n)); \\ 0, & \text{otherwise} \end{cases} \\ \sigma(\neg S) = 1 - \sigma(S) & \\ \sigma(S_1 \vee S_2) = \max\{\sigma(S_1), \sigma(S_2)\} & \end{array}$$

Thus, given a flow H and a reference interval of the flow $[a, b]$ with $a, b \in \mathrm{Dom}(H)$, and $a \leq b$, we can formally define the meaning of a history formula HF inductively as follows:

- $H, [b, e] \models \ell \rhd T$ iff $e - b \rhd T$, where $\rhd \in \{\leq, >, =, \neq, \leq, <\}$;
- $H, [b, e] \models \lceil S \rceil^0$ iff $b = e$, and $H(b)(S) = 1$;
- $H, [b, e] \models \neg HF$ iff $H, [b, e] \not\models HF$;
- $H, [b, e] \models HF_1 \wedge HF_2$ iff $H, [b, e] \models HF_1$ and $H, [b, e] \models HF_2$;
- $H, [b, e] \models HF_1 \vee HF_2$ iff $H, [b, e] \models HF_1$ or $H, [b, e] \models HF_2$;
- $H, [b, e] \models HF_1^\frown HF_2$ iff there is $m \in [b, e]$ such that $H, [b, m] \models HF_1$ and $H, [m, e] \models HF_2$.

6.2 Hoare Assertion

A Hoare assertion of HHL consists of four parts: precondition, process, postcondition and history, written as

$$\{Pre\}P\{Post; HF\}$$

where *Pre* specifies values of $\mathcal{V}(P)$ before an execution of P, *Post* specifies values of $\mathcal{V}(P)$ when P terminates, and *HF* is a formula of $\mathcal{V}(P)$ from the DC subset to describe the execution history of P. HCSP has three kinds of interruptions: boundary interruption like $\langle F(\dot{s}, s) = 0 \wedge B\rangle$, timeout interruption like $\langle F(\dot{s}, s) = 0 \wedge B\rangle \trianglerighteq_d Q$ and communication interruption like $\langle F(\dot{s}, s) = 0 \wedge B\rangle \trianglerighteq [\!]_{i \in I}(ch_i * \rightarrow Q_i)$. For these three kinds of interruptions, *HF* has to join in reasoning.

Definition 15 (Validity). *We say a Hoare assertion* $\{Pre\}P\{Post; HF\}$ *is* valid*, denoted by* $\models \{Pre\}P\{Post; HF\}$*, iff for any initial state* σ_1*, if* $(P, \sigma_1) \xrightarrow{\alpha^*} (\epsilon, \sigma_2, H)$ *then* $\sigma_1 \models Pre$ *implies* $\sigma_2 \models Post$ *and* $H, [\sigma_1(now), \sigma_2(now)] \models HF$*.*

For a parallel process, say $P_1 \parallel ... \parallel P_n$, the assertion becomes

$$\{Pre_1, ..., Pre_n\}P_1 \parallel ... \parallel P_n\{Post_1, ..., Post_n; HF_1, ..., HF_n\}$$

where $Pre_i, Post_i, HF_i$ are (first order or DC) formulas of $\mathcal{V}(P_i)$ $(i = 1, ..., n)$ separately. The validity can be defined similarly.

Another role of *HF* is to specify real-time (continuous) property of an HCSP process, while *Pre* and *Post* can only describe its discrete behaviour. *HF* therefore bridges up the gap between discrete and continuous behaviour of the process. For instance, in Example 8, we may want the plant controller stable after T time units, i.e. after T time units the distance between the trajectory of s and its target s_{targ} must be small. This can be specified through the following assertion.

$$\begin{array}{l}\{s = s_0 \wedge u = u_0 \wedge Ctrl(u_0, s_0), Pre_2\}\ PLC \\ \quad \{Post_1, Post_2; (l = T)^\frown\lceil \mid s - s_{targ} \mid \leq \epsilon\rceil, HF_2\}\end{array}$$

where $Ctrl(u, s)$ may express a controllable property, and the other formulas are not elaborated here.

Note that we can essentially put *Pre* and *Post* as parts of history formula *HF* like the form $\lceil Pre\rceil^0{}^\frown HF^\frown\lceil Post\rceil^0$. But we did not adopt this way, because separation of specifying and reasoning about discrete behavior and continuous behavior can indeed improve readability and simplify our approach.

6.3 Proof System of HHL

We will omit the axioms and inference rules for the first-order logic and DC, and just concentrate on the axioms and rules for the constructs of HCSP.

1. **Monotonicity**

$$\begin{array}{l}\text{If } \{Pre_1, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}, \\ \text{and } Pre_i' \Rightarrow Pre_i, Post_i \Rightarrow Post_i', HF_i \Rightarrow HF_i'(i = 1, 2), \\ \text{then } \{Pre_1', Pre_2'\}P_1 \parallel P_2\{Post_1', Post_2'; HF_1', HF_2'\}\end{array}$$

   where we use first order logic to reason about $Pre_i' \Rightarrow Pre_i$ and $Post_i \Rightarrow Post_i'$, but use DC to reason about $HF_i \Rightarrow HF_i'$. From now on we will not repeatedly mention this.

2. **Case Analysis**

$$\begin{array}{l}\text{If } \{Pre_{1i}, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}\ (i = 1, 2),\\ \text{then } \{Pre_{11} \vee Pre_{12}, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}\end{array}$$

Symmetrically,

$$\begin{array}{l}\text{If } \{Pre_1, Pre_{2i}\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}\ (i = 1, 2),\\ \text{then } \{Pre_1, Pre_{21} \vee Pre_{22}\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}\end{array}$$

3. **Parallel vs Sequential**
These two rules show a simple relation between assertions of a parallel process and its sequential components that can ease a proof.

$$\begin{array}{l}\text{If } \{Pre_1, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}\\ \text{then } \{Pre_i\}P_i\{Post_i; HF_i\}\ (i = 1, 2)\end{array}$$

and

$$\begin{array}{l}\text{If } \{Pre_i\}P_i\{Post_i; HF_i\}\ (i = 1, 2),\\ \text{and } P_i\ (i = 1, 2) \text{ do not contain communication,}\\ \text{then } \{Pre_1, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\}\end{array}$$

4. **Skip**

$$\{Pre\}\mathbf{skip}\{Pre; l = 0\},$$

where by $l = 0$ we assume that, in comparison with physical device, computation takes no time (i.e. *super dense computation* [60])

5. **Assignment**

$$\{Pre[e/x]\}x := e\{Pre, \lceil x = e\rceil^0\}$$

The precondition and postcondition are copied from Hoare Logic. Here we use $\lceil x = e\rceil^0$ as its history to indicate that x is assigned to e, which takes place at this time point.

6. **Communication**
Since HCSP rejects sharing variables, a communication looks like the output party $(P_1; ch!e)$ assigning to variable x of the input one $(P_2; ch?x)$ a value e. Besides, in order to synchronize both parties, one may have to wait for another. During the waiting of P_i, $Post_i$ must stay true ($i = 1$ or 2). We use $const(\mathcal{V}(P))$ to denote $\wedge_{x\in\mathcal{V}(P)}\exists v.\lceil x = v\rceil$, which means that all variables of P keep unchanged except for at the endpoint.

$$\begin{array}{l}\text{If } \{Pre_1, Pre_2\}P_1 \parallel P_2\{Post_1, Post_2; HF_1, HF_2\},\\ \quad Post_1 \Rightarrow G(e), HF_1 \Rightarrow \ell = c_1, \text{and } HF_2 \Rightarrow \ell = c_2\\ \text{then } \{Pre_1, Pre_2\}(P_1; ch!e) \parallel (P_2; ch?x)\\ \qquad \{Post_1, G(x) \wedge \exists x.Post_2; HF_1{}^\frown(\lceil Post_1\rceil \wedge const(\mathcal{V}(P_1)) \wedge \ell = c - c_1),\\ \qquad\quad (HF_2{}^\frown(\lceil Post_2\rceil \wedge const(\mathcal{V}(P_2)) \wedge \ell = c - c_2))^{<\frown}\lceil x = e\rceil^0\}\\ \text{where } c = \max\{c_1, c_2\}.\end{array}$$

Note that for simplicity, in the above rule we just consider a simple case of communication; a rule for the general case of communication

$$(P_1; [\!]_{i\in I} ch_i* \to Q_{1i}) \parallel (P_2; [\!]_{j\in J} ch_j* \to Q_{2j}),$$

where $ch_i* = \overline{ch_j*}$ for some $i \in I, j \in J$, can be defined similarly.

Example 9. If

$$\{Pre_1, Pre_2\}P_1 \parallel P_2$$
$$\{y = 3, x = 1; (\lceil y = 0\rceil \wedge (l = 3))^\frown \lceil y = 3\rceil^0, \lceil x = 0\rceil \wedge (l = 5)^\frown \lceil x = e\rceil^0\},$$

we want to deduce through this rule

$$\{Pre_1, Pre_2\}P_1; ch!y \parallel P_2; ch?x\{Post_3, Post_4; HF_3, HF_4\}.$$

Since $(y = 3) \Rightarrow (3 = 3)$, $\lceil y = 0\rceil \wedge (l = 3))^\frown \lceil y = 3\rceil^0 \Rightarrow \ell = 3$, and $(\lceil x = 0\rceil \wedge \ell = 5)^\frown \lceil x = 1\rceil^0 \Rightarrow \ell = 5$, we can conclude that $Post_3$ is $y = 3$, $Post_4$ is $x = 3$, HF_3 is $((\lceil y = 0\rceil \wedge (l = 3))^\frown \lceil y = 3\rceil^{0\frown}(\lceil y = 3\rceil \wedge const(\mathcal{V}(P_1) \cup \{y\}) \wedge \ell = 2)$, and HF_4 is $(\ell = 5^\frown \lceil x = 1\rceil^0)^{<\frown} \lceil x = 3\rceil^0$, which is equivalent to $(\ell = 5^\frown \lceil x = 3\rceil^0$ by the definition of $HF^<$. □

7. **Continuous**
This is about $\langle F(\dot{s}, s) = 0 \wedge B\rangle$, where s can be a vector and F be a group of differential equations, such as

$$\langle(\dot{s}_1 = f_1, ..., \dot{s}_n = f_n) \wedge B\rangle.$$

As indicated in Sec 3, in our framework, we only deal with polynomial differential equations and *semi-algebraic* differential invariants. That is, f_js are polynomials in s_i $(i = 1, ..., n)$, B is a conjunction of polynomial equations and inequalities of s_i $(i = 1, ..., n)$, and differential invariants are also restricted to polynomial equations and inequalities. So, given a polynomial differential invariant Inv of $\langle F(\dot{s}, s) = 0 \wedge B\rangle$ with initial values satisfying $Init$, the inference rule for *continuous* can be formulated as follows:

$$\begin{array}{l}\text{If } Init \Rightarrow Inv,\\ \text{then } \{Init \wedge Pre\}\langle F(\dot{s}, s) = 0 \wedge B\rangle\{Pre \wedge \mathbf{Cl}(Inv) \wedge \mathbf{Cl}(\neg B);\\ \quad \lceil Inv \wedge Pre \wedge B\rceil\}\end{array}$$

where Pre does not contain s, $\mathbf{Cl}(G)$ stands for the *closure* of G[10].
The second rule is about explicit time.

$$\begin{array}{l}\text{If } \{Pre\}\langle F(\dot{s}, s) = 0 \& B\rangle\{Post; HF\}\\ \text{and } \{Pre \wedge t = 0\}\langle(F(\dot{s}, s) = 0, \dot{t} = 1) \& B\rangle\{t = t_0 \wedge Rg(t_0), HF'\},\\ \text{then } \{Pre\}\langle F(\dot{s}, s) = 0 \& B\rangle\{Post; HF \wedge Rg(\ell)\}\end{array}$$

where t is a clock to count the execution time, and $Rg(t)$ is a constraint on the final value of t which is an arithmetic formula.

[10] When G is constructed by polynomial inequalities through $\wedge$ and $\vee$, $\mathbf{Cl}(G)$ can be obtained from G by replacing $<$ (and $>$) with $\leq$ (and $\geq$) in G.

Example 10. According to the result given in Section 3, it is easy to see that $v \leq v_{ebi}$ is an invariant of $\langle (\dot{s} = v, \dot{v} = a) \wedge v < v_{ebi} \rangle$. Thus, by the continuous rule

$$\{(v = v_0 \leq v_{ebi})\}\langle (\dot{s} = v, \dot{v} = a) \wedge v < v_{ebi} \rangle \\ \{(v \leq v_{ebi}) \wedge (v \geq v_{ebi}); \lceil (v \leq v_{ebi}) \wedge (v < v_{ebi}) \rceil\}$$

In addition, we can prove that, if the initial values are $v = v_0$ and $t = 0$, and we assume $p \geq a \geq w$, then

$$((v_0 + wt) \leq v \leq (v_0 + pt)) \wedge (v \leq v_{ebi})$$

is an invariant of $\langle (\dot{s} = v, \dot{v} = a, \dot{t} = 1) \wedge v < v_{ebi} \rangle$. So under the assumption $(p \geq a \geq w)$

$$\{(v = v_0 \leq v_{ebi}) \wedge (t = 0)\}\langle (\dot{s} = v, \dot{v} = a, \dot{t} = 1) \wedge v < v_{ebi} \rangle \\ \{(v = v_{ebi}) \wedge ((v_0 + wt) \leq v \leq (v_0 + pt)) \wedge \tfrac{v_{ebi} - v_0}{w} \geq t \geq \tfrac{v_{ebi} - v_0}{p}; \\ \lceil (v < v_{ebi}) \wedge ((v_0 + wt) \leq v \leq (v_0 + pt)) \rceil\}$$

Therefore, assuming $(p \geq a \geq w)$ we can have

$$\{(v = v_0 \leq v_{ebi})\}\langle (\dot{s} = v, \dot{v} = a) \wedge v < v_{ebi} \rangle \\ \{(v = v_{ebi}); \lceil (v < v_{ebi}) \rceil \wedge (\tfrac{v_{ebi} - v_0}{w} \geq l \geq \tfrac{v_{ebi} - v_0}{p})\}$$

□

8. **Sequential.** The rule for sequential composition is very standard, given as follows:
   If $\{Pre_1\}P_1\{Post_1; HF_1\}$, and $\{Post_1\}P_2\{Post_2; HF_2\}$
   then $\{Pre_1\}P_1; P_2\{Post_2; HF_1^{\frown} HF_2\}$.
9. **Internal Choice.** The rule for internal choice is standard, given as follows:
   If $\{Pre\}P_1\{Post_1; HF_1\}$ and $\{Pre\}P_2\{Post_2; HF_2\}$,
   then $\{Pre\}P_1 \sqcup P_2\{Post_1 \vee Post_2; HF_1 \vee HF_2\}$.
10. **Communication Interruption**
    There are two rules for communication interruption, the first one says that the continuous part terminates before a communication happens, while the second one states that the continuous evolution is interrupted by a communication.
    Rule1: If
    (a) $\{Pre, Pre_R\}\, \langle F(\dot{s}, s) = 0 \& B \rangle \parallel R\{Post, Post_R; HF, HF_R\}$,
    (b) for all $i \in I$, $\{Pre, Pre_R\}\, ch_i * \parallel R\{Post_i, Post_R^i; HF_i, HF_R^i\}$,
    (c) $HF \Rightarrow \ell = x$, $\wedge_{i \in I}(HF_i \Rightarrow \ell = x_i) \wedge x < x_i$,
    then
    $$\{Pre, Pre_R\}\, \langle F(\dot{s}, s) = 0 \& B \rangle \trianglerighteq [\!]_{i \in I}(ch_i * \to Q_i) \parallel R \\ \{Post, Post_R; HF, HF_R\}$$
    Rule 2: Assume $j \in I$. If
    (a) $\{Pre, Pre_R\}\, \langle F(\dot{s}, s) = 0 \& B \rangle \parallel R_1; \overline{ch_j *} \to R_2\{Post, Post_R; HF, HF_R\}$,
    (b) for all $i \in I$, $\{Pre, Pre_R\}\, ch_i * \parallel R_1; \overline{ch_j *}\, \{Post_i, Post_R^i; HF_i, HF_R^i\}$,
    (c) $HF \Rightarrow \ell = x$, $\wedge_{i \in I} HF_i \Rightarrow \ell = x_i$, and $x_j \leq x \wedge \wedge_{i \neq j} x_j \leq x_i$,

(d) $HF \Rightarrow (\ell = x_j \wedge HF_s)^\frown\lceil G(s_0)\rceil^0$,
(e) $\{Post_j \wedge G(s_0), Post_R^j\}Q_j \parallel R_2\{Post^f, Post_R^f; HF^f, HF_R^f\}$,
then

$$\{Pre, Pre_R\}\,\langle F(\dot{s}, s) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(ch_i* \to Q_i) \parallel R_1; \overline{ch_j*} \to R_2$$
$$\{Post^f, Post_R^f; ((HF_s{}^\frown\lceil G(s_0)\rceil^0) \wedge HF_j)^\frown HF^f, HF_R^j{}^\frown HF_R^f\}$$

Note that for simplicity, in Rule 2, we only consider $\langle F(\dot{s}, s) = 0 \& B\rangle \trianglerighteq [\!]_{i\in I}(ch_i* \to Q_i)$ to be parallel with $R_1; \overline{ch_j*}; R_2$. For general case, the rule can be given similarly, without any difficulty.

11. **Repetition**
We can pick up rules from the literature for the repetition. Here we only show a rule which ends off an assertion reasoning.

If $\{Pre_1, Pre_2\}P_1 \parallel P_2\{Pre_1, Pre_2; HF_1, HF_2\}$,
$HF_i \;\Rightarrow\; (D_i \wedge (l = T))\ (i = 1, 2,\ T > 0)$,
and $D_i^\frown D_i \;\Rightarrow\; D_i$,
then $\{Pre_1, Pre_2\}P_1^* \parallel P_2^*\{Pre_1, Pre_2; \ell = 0 \vee D_1, \ell = 0 \vee D_2\}$

where T is the time consumed by both P_1 and P_2 that can guarantee the synchronisation of the starting point of each repetition.

6.4 Soundness

We only present the case for sequential processes.

Definition 16 (Theorem). *We say a Hoare triple* $\{Pre\}P\{Post; HF\}$ *is a* theorem, *denoted by* $\vdash \{Pre\}P\{Post; HF\}$, *iff it is derivable from the above proof system.*

The soundness of the proof system is guaranteed by the following theorem.

Theorem 11 (Soundness). *If* $\vdash \{Pre\}P\{Post; HF\}$, *then* $\models \{Pre\}P\{Post; HF\}$, *i.e. every theorem of the proof system is valid.*

7 HHL Prover

In this section, we aim to provide the tool support for verifying whether an HCSP process conforms to its specification written in HHL. Fig. 8 shows the verification architecture of our approach: given an annotated HCSP process in the form of HHL specification, by designing a verification condition generator based on HHL proof system, the specification to be proved is reduced to a set of verification conditions, each of which is either a first-order formula or a DC formula, and the validity of these logical formulas is equivalent to that of the original specification; these logical formulas can then be proved by interactive theorem proving, furthermore, some of which falling in decidable subsets of first-order logic or DC can be proved automatically by designing the corresponding decision procedures.

As shown in Fig. 8, a differential invariant generator is needed for specifying and verifying differential equations. But currently we assume for each differential equation, its invariant is annotated as given, as we have not implemented the results reported in Sec. 3 yet. As one of future work, such an invariant generator will be implemented and integrated.

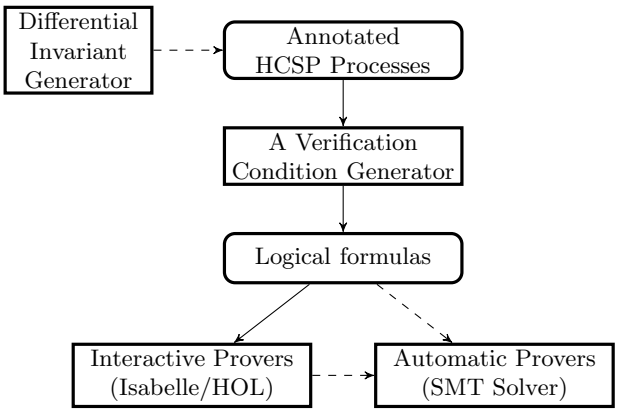

Fig. 8. Verification Architecture of HCSP Processes

We have mechanized the main part of the verification architecture connected by solid lines shown in Fig. 8 in proof assistant Isabelle/HOL, based on which implemented an interactive theorem prover called *HHL prover* for verifying HHL specifications. The mechanization mainly includes the embedding of HCSP, the assertion languages, i.e., first-order logic (FOL) and DC, and upon them, the embedding of the proof system of HHL in Isabelle/HOL. We adopt the deep embedding approach [14,87] here, which represents the abstract syntax for both HCSP and assertions by new datatypes, and then defines the semantic functions that assign meanings to each construct of the datatypes. It allows us to quantify over the syntactic structures of processes and assertions, and furthermore, make full use of deductive systems for reasoning about assertions written in FOL and DC.

The HHL prover can be downloaded at `https://github.com/iscas/HHL_prover`.

7.1 Expressions

We start from encoding the bottom construct, i.e. expressions, that are represented as a datatype `exp`:

```
datatype exp = RVar string | SVar string | BVar string | Real real
         | String string | Bool bool | exp + exp | exp − exp | exp ∗ exp
```

An expression can be a variable, that can be of three types, `RVar x` for real variable, `SVar x` and `BVar x` for string and boolean variables; a constant, that can be also of the three types, e.g. `Real 1.0`, `String ''CO''` and `Bool True`; an arithmetic expression constructed from operators $+, -, *$. Based on expressions, we can define the assertion languages and the process language HCSP respectively.

7.2 Assertion Language

As we introduced in Sec. 6, there are two assertion logics in HHL: FOL and DC, where the former is used for specifying the pre-/post-conditions and the latter for the execution history of a process respectively. The encodings for both logics consist of two parts: syntax and deductive systems. We will encode the deductive systems in Gentzen's sequent calculus style, which applies backward search to conduct proofs and thus is more widely used in interactive and automated reasoning. A sequent is written as $\Gamma \vdash \Delta$, where both Γ and Δ are sequences of logical formulas, meaning that when all the formulas in Γ are true, then at least one formula in Δ will be true. We will implement a sequent as a truth proposition. The sequent calculus deductive system of a logic is composed of a set of sequent rules, each of which is a relation between a (possibly empty) sequence of sequents and a single sequent. In what follows, we consider to encode FOL and DC respectively.

First-Order Logic. The FOL formulas are constructed from expressions by using relational operators from the very beginning, and can be represented by the following datatype `fform`:

datatype `fform = [True] | [False] | exp [=] exp | exp [<] exp`
`| [¬] fform | fform [∨] fform | [∀]` *string* `fform`

The other logical connectives including $[\wedge]$, $[\rightarrow]$, and $[\exists]$ can be derived as normal. For quantified formula $[\forall]$*string* `fform`, the name represented by a string corresponds to a real variable occurring in `fform`. We only consider the quantification over real variables here, but it can be extended to variables of other types (e.g. string and bool) without any essential difficulty. Notice that we add brackets to wrap up the logical constructors in order to avoid the name conflicts between `fform` and the FOL system of Isabelle library. But in sequel, we will remove brackets for readability when there is no confusion in context; and moreover, in order to distinguish between FOL formulas and Isabelle meta-logic formulas, we will use $\Rightarrow$, & and | to represent implication, conjunction and disjunction in Isabelle meta-logic.

Now we need to define the sequent calculus style deductive system for `fform`. The Isabelle library includes an implementation of the sequent calculus of classical FOL with equation, based upon system LK that was originally introduced by Gentzen. Our encoding of the sequent calculus for `fform` is built from it directly, but with an extension for dealing with the atomic arithmetic formulas that are defined in `fform`. We define an equivalent relation between the validity of formulas of `fform` and of *bool*, the built-in type of Isabelle logical formulas, represented as follows:

```
formT (f :: fform) ⇔ ⊢ f
```

where the function `formT` transforms a formula of type `fform` to a corresponding formula of *bool*. This approach enables us to prove atomic formulas `f` of `fform` by applying the built-in arithmetic solvers of Isabelle and proving `formT (f)` instead.

Duration Calculus. Encoding DC into different proof assistants has been studied, such as [78] in PVS, and [38,72] in Isabelle/HOL. DC can be considered as an extension of Interval Temporal Logic (ITL) by introducing state durations (here point formulas instead), while ITL an extension of FOL with the introducing of temporal variables and chop modality by regarding intervals instead of points as worlds. Therefore, both [38] and [72] apply an incremental approach to encode ITL on top of an FOL sequent calculus system, and then DC on top of ITL. We will follow a different approach here, to represent DC formulas as a datatype, as a result, the proving of DC formulas can be done by inductive reasoning on the structures of the formulas.

The datatype `dform` encodes the history formulas HF given in Sec. 6:

```
datatype dform = [[True]] | [[False]] | dexp[[=]]dexp | dexp[[<]]dexp
  | [[¬]]dform | dform[[∨]]dform |[[∀]] string dform | pf fform | dform⌢dform
```

We will get rid of double brackets for readability if without confusion in context. The datatype `dexp` defines expressions that are dependent on intervals. As seen from HF, it includes the only temporal variable ℓ for representing the length of the interval, and real constants. Given a state formula `S` of type `fform`, `pf S` encodes the point formula $\lceil S \rceil^0$, and furthermore, the following `high S` encodes formula $\lceil S \rceil$:

```
high :: fform ⇒ dform
high S ≡ ¬ (True ⌢pf (¬S)⌢ ℓ > Real 0)
```

The chop modality $\frown$ can be encoded as well.

To establish the sequent calculus style deductive system for `dform`, we first define the deductive system for the first-order logic constructors of `dform`, which can be taken directly from the one built for `fform` above, and then define the deductive system related to the new added modalities for DC, i.e. ℓ, $\frown$ and `pf`.

For ℓ and $\frown$, we encode the deductive system of ITL from [96], which is presented in Hilbert style. Thus, we need to transform the deductive system to sequent calculus style, and it is not so natural to do. We borrow the idea from [72] that for each modality, define both the left and right introducing rules, e.g., the following implementation

```
LI  : $H, P ⊢ $E ⇒ $H, P⌢(ℓ = Real 0) ⊢ $E
RI  : $H ⊢ P, $E ⇒ $H ⊢ P⌢(ℓ = Real 0), $E
```

where `$H`, `$E` represent arbitrary sequences of logical formulas of type `dform`, encodes the axiom of ITL: $P \leftrightarrow P^\frown(\ell = 0)$. In the same way, for point formula `pf`, we encode the deductive system of DC defined in [96] in sequent calculus style, e.g., the following implementation

```
PFRI : $H ⊢ (pf S1⌢pf S2), $E ⇒ $H ⊢ pf (S1 ∧ S2), $E
```

encodes the axiom of DC: $\lceil S_1 \rceil^0 \frown \lceil S_2 \rceil^0 \to \lceil S_1 \wedge S_2 \rceil^0$.

7.3 HCSP

We represent HCSP processes as a datatype `proc`, and each construct of HCSP can be encoded as a construct in datatype `proc` correspondingly. Most of the encoding is directly a syntactic translation, but with the following exceptions:

- As mentioned in previous sections, in the deductive verification of HCSP process, the role of differential equation is reflected by an differential invariant with respect to the property to be verified, which can be automatically discovered in polynomial cases. So in `proc`, instead of differential equation, we use differential invariant to describe the underlying continuous, and for aiding verification, we also add execution time range of the continuous. Thus, we encode continuous of form $\langle \mathcal{F}(\dot{s}, s) = 0 \& B \rangle$ as `<Inv&B> : Rg`, where `Inv` represents the differential invariant of the continuous, `B` the domain constraint, and `Rg` the range of execution time, of the continuous respectively; and `Inv`, `B` are implemented as formulas of type `fform`, while `Rg` of type `dform`.
- For sequential composition, we encode $P; Q$ as `P; mid; Q`, where `P` and `Q` represent the encodings of P and Q respectively, and `mid` is added to represent the intermediate assertions between P and Q. This is requisite for reducing proof of sequential composition to the ones of its components, and commonly used in theorem proving.
- For parallel composition, we remove the syntax restriction that it can only occur in the outmost scope, thus it is encoded with the same datatype `proc` as other constructs.

7.4 Semantics

In this section, we encode the semantics of HCSP, FOL and DC in Isabelle/HOL. This is done by implementing all the relevant semantic notations and functions defined in Sec. 5 and Sec. 6.

There are two disjoint sets of variables considered in the semantics of HCSP: local variables in $\mathcal{V}(P)$, and system variables $\{now, tr, rdy\}$. Notice that the system variables do not occur in HHL; therefore, we will implement the semantic functions for evaluating them separately. We define type `state` to represent states and each element of it is a function that assigns respective values to (only) process variables. Besides, we define types `now`, `trace`, `ready` to represent system time (i.e. *real*), timed traces and ready sets of communication events. Based on these definitions, we implement the behavior of a process by the following `lbevr` and `sbevr`:

```
type_synonym lbevr = now ⇒(state list)
              sbevr = now ⇒((trace*ready) list)
```

Each local behavior of type `lbevr` associates a sequence of states to each time point, while each system behavior of type `sbevr` associates a sequence of traces and ready sets to each time point. The combination of local behavior and system behavior implements the overall behavior defined in Sec. 5. It should be pointed out that the flow of a process is not implemented explicitly here, but it can always be extracted from the behavior of the process by only keeping the rightmost state in the state list for each time point[11]. Thus, in the following, we always use a behavior whenever a flow is needed.

The expressions of HCSP are interpreted over states. Given a state `s` of type `state` and an expression `e` of type `exp`, the function `evalE(s, e)` defines the value of `e` under the state `s`. Based on the evaluation of expressions, the pre-/post-conditions in the form of `fform`, can be interpreted over states. Given a state `s` of type `state` and a formula `p` of type `fform`, the function `evalF(s, p)` evaluates the truth value of `p` under the state `s`.

As defined in Sec. 6, the history formulas of DC are interpreted over flows and timed intervals. Because the history formulas do not refer to system variables, we interpret them over local behaviors instead of flows. First of all, given a local behaviour `f` of type `lbevr` and a timed interval `[c, d]`, `ievalE(f,`ℓ`, c, d)` returns the value of ℓ, that is `d-c`, under the behavior `f` and the timed interval `[c, d]`. Given a behavior `f` of type `lbevr`, a DC formula `ip`, and a timed interval `[c, d]`, `ievalF(f, ip, c, d)` evaluates the truth value of `ip` under the behavior `f` and the timed interval `[c, d]`. In particular, the point formula and chop can be defined as follows:

```
pf_eval: ievalF (f, pf (P), c, d) = (c=d & evalF (last(f(c)), P))
chop_eval: ievalF (f, P^Q, c, d) = ∃k. c<=k & k<=d & ievalF (f, P, c, k)
                                                  & ievalF (f, Q, k, d)
```

Thus, `pf(P)` holds, iff the interval is a point interval, and `P` holds at the last state of the state list that is recorded at the time point.

Finally, we implement the operational semantics of HCSP processes. Given a process `P` of type `proc`, a local behavior `f` of type `lbevr`, a system behavior `sf` of type `sbevr`, an event α of type `event`, and a time point `d`, the function `evalP (P, f, sf, d,`α`) = (P', f', sf', d')` represents that, starting to execute from behaviors `f` and `sf`, and time `d`, `P` performs an event α and evolves to `P'` at time `d'`, and produces the new local and system behaviors `f'` and `g'` respectively. It implements exactly the transition relation $(P, \sigma) \xrightarrow{\alpha} (P', \sigma', H)$ defined in Sec. 5, in particular that the initial state σ can be extracted from `f`, `sf`, and `d`, while final state σ' from `f'`, `sf'`, and `d'` respectively.

We explain the semantics of several HCSP constructors as an illustration here. For instance, the transition rule (Ass) of assignment is implemented as follows (only the case for real variables is considered):

[11] Please note the difference between flow and behavior that a flow only records the last state occurring at any time point, while the corresponding behavior records all states occurring at the time point.

```
assignR : evalP (((RVar (x)) :=e), f, sf, d, Tau) =
              (ϵ, updateVal(f, x, R, e, d), updateTr(sf, Tau, d), d)
```

where `updateVal` adds a new state corresponding to the discrete assignment to the state list recorded at time `d`, and this new state is the same as the initial state except that the value of variable `RVar (x)` is updated by `e`; and `updateTr` adds a `Tau` event to the initial trace and then pushes the resulting trace to the trace list recorded at time `d`. Notice that the termination time is still d, indicating assignment does not take time. As another instance, the transition rule (In-3) of input is implemented as follows:

```
in3 : inList((fst(last(sf(d))), (I ch)), snd(last(sf(d))))
      ⇒ evalP(ch??(RVar(x)), f, sf, d, (Inp ch e)) =
          (ϵ, updateVal(f, x, R, e, d), removeRdy(
      updateTr(sf, Com(ch, e), d), (fst(last(sf(d))), (I ch)), d), d)
```

where `ch??(RVar(x))` of type `proc` represents an input to real variable. The predicate `inList(...)` represents that the communication event corresponding to input `ch??(RVar(x))`, represented by `(fst(last(sf(d))), (I ch))` of type `ready`, is in the initial ready set (represented by `snd(last(sf(d)))`); and it implements exactly the premise in rule (In-3). It performs an input event `Inp ch e`, and results in the adding of a new state that assigns the value of `e` to `RVar(x)` to the state list at time `d`, and the adding of a new trace increased by the communication `Com(ch,e)` to the trace list at time `d`, and the adding of a new ready set with the removal of the communication event corresponding to input `ch??(RVar(x))` at time `d`. At last, the transition rule (Par-3) for parallel composition is implemented as follows:

```
par3: evalP(P, f, sf, d, (Inp ch e)) = (P', f', sf', d) &
  evalP(Q, f, sf, d, (Outp ch e)) = (Q', f'', sf'', d) ≡
    evalP((P || Q), f, sf, d, a) = ((P' || Q'), f', sf', d)
```

`P` performs an input communication event, while `Q` performs an output communication event along the same channel, as a consequence, a synchronization occurs for `P || Q`. Notice that the resulting behaviors of `P || Q` are exactly the same to those of `P`.

7.5 Proof System of HHL

With the definitions of datatypes `proc`, `fform` and `dform`, it is now easy to encode HHL assertions. First of all, a Hoare assertion for sequential process `P` is implemented as a truth proposition of the form `{Pre} P {Post;HF}`, where `Pre` and `Post` are of type `fform`, and `HF` of type `dform` respectively. A Hoare assertion for parallel process `P||Q` can be implemented in the similar way.

Verification Condition. Based on the inference rules of HHL, we implement the verification condition generator for reasoning about HCSP specifications. The inference rules encoded here are slightly different from those presented in Sec. 6, in the sense that we remove the point formulas for specifying discrete changes

in history formulas and use $\ell = 0$ instead. This will not affect the expressiveness and soundness of HHL.

In deep embedding, the effects of assignments are expressed at the level of formulas by substitution. We implement a map as a list of pairs `(exp * exp) list`, and then given a map σ and a formula `p` of type `fform`, we define function `substF(`σ`, p)` to substitute expressions occurring in `p` according to the map σ. Based on this definition, we have the following axiom for assignment `e:=f`:

axioms `Assignment :`
$\vdash$ `(p` $\rightarrow$ `substF ([(e, f)], q))` $\wedge$ `(`ℓ `= Real 0` $\rightarrow$`G)` $\Rightarrow$ `{p} (e :=f) {q; G}`

According to the rule of assignment, the weakest precondition of `e := f` with respect to postcondition `q` is `substF ([(e, f)], q)`, and on the other hand, the strongest history formula for assignment is ℓ`= Real 0`, indicating that as a discrete action, assignment takes no time. Therefore, `{p} (e :=f) {q; G}` holds, if `p` implies the weakest precondition, and moreover, `G` is implied by the strongest history formula.

For continuous `<Inv & B> : Rg`, we assume that the precondition can be separated into two conjunctive parts: `Init` referring to initial state of continuous variables, and `p` referring to other distinct variables that keep unchanged during continuous evolution. With respect to precondition `Init`$\wedge$`p`, according to the rule of continuous, when it terminates (i.e. `B` is violated), the precondition `p` not relative to initial state, the closures of `Inv` and of $\neg$`B` hold in postcondition; moreover, there are two cases for the history formula: the continuous terminates immediately, represented by ℓ`= Real 0`, or otherwise, throughout the continuous evolution, `p`, `Inv` and `B` hold everywhere except for the endpoint, represented by `high (Inv`$\wedge$`p`$\wedge$`B)`, where both cases satisfy `Rg`.

axioms `Continuous :` $\vdash$`(Init` $\rightarrow$`Inv)` $\wedge$ `((p` $\wedge$ `close(Inv)` $\wedge$ `close(`$\neg$`B))` $\rightarrow$`q)`
$\wedge$ `((((`ℓ `= Real 0)` $\vee$ `(high (Inv` $\wedge$ `p` $\wedge$ `B)))` $\wedge$ `Rg)` $\rightarrow$ `G)`
$\Rightarrow$ `{Init` $\wedge$ `p} <Inv & B> : Rg {q; G}`

where function `close` returns closure of corresponding formulas. The above axiom says that `{Init`$\wedge$`p} <Inv & B> : Rg {q;G}` holds, if the initial state satisfies invariant `Inv`, and furthermore, both `q` and `G` are implied by the postcondition and the history formula of the continuous with respect to `Init`$\wedge$`p` respectively.

For sequential composition, the intermediate assertions need to be annotated (i.e., `(m, H)` below) to refer to the postcondition and the history formula of the first component. Therefore, the specification `{p} P; (m, H); Q {q; H`$\frown$`G}` holds, if both `{p} P {m;H}` and `{m} Q {q;G}` hold, as indicated by the following axiom.

axioms `Sequence : {p} P {m; H}; {m} Q {q; G}` $\Rightarrow$`{p} P; (m, H); Q {q; H`$\frown$`G}`

The following axiom deals with communication `P1; ch!e || P2;ch?x`, where `P1` and `P2` stand for sequential processes. Let `p1` and `p2` be the preconditions for the sequential components respectively, and `(q1, H1)`, `(q2, H2)` the intermediate assertions specifying the postconditions and history formulas for `P1` and `P2` respectively. `r1` and `G1` represent the postcondition and history formula for the

left sequential component ended with `ch!e`, while `r2` and `G2` for the right component ended with `ch?x`. `Rg` stands for the execution time range of the whole parallel composition.

```
axioms Communication :
  {p1, p2} P1 || P2 {q1, q2; H1, H2};
  ⊢ (q1 →r1) ∧ (q2 →substF ([(x, e)], r2));
  ⊢ (H1 ⌢ high (q1)) →G1) ∧ (H2 ⌢ high (q2)) →G2);
  ⊢ (((H1 ⌢ high (q1)) ∧ H2) ∨ ((H2 ⌢ high (q2)) ∧ H1)) →Rg;
    ⇒ {p1, p2} ((P1; (q1, H1); ch !! e) || (P2; (q2, H2); ch ?? x))
              {r1, r2; G1 ∧ Rg, G2 ∧ Rg}
```

As shown above, to prove the final specification, the following steps need to be checked: first, the corresponding specification with intermediate assertions as postconditions and history formulas holds for `P1 || P2`; second, after the communication is done, for the sending party, `q1` is preserved, while for the receiving party, `x` is assigned to be `e`. Thus, `r1` must be implied by `q1`, and `q2` implies the weakest precondition of the communicating assignment with respect to `r2`, i.e. `substF ([(x, e)], r2)`; third, for the communication to take place, one party may need to wait for the other party to be ready, in case that `P1` and `P2` do not terminate simultaneously. The left sequential component will result in history formula `H1⌢high (q1)`, in which `high (q1)` indicates that during waiting time, the postcondition of `P1` is preserved, and similarly for the right component. Thus, `G1` and `G2` must be implied by them respectively; and finally, for both cases when one party is waiting for the other, the conjunction of their history formulas must satisfy the execution time `Rg`.

For repetition, we have the following implementation:

```
axioms Repetition :
{p1, p2} P || Q {p1, p2; H1, H2}; ⊢(H1 ⌢ H1 →H1) ∧ (H2 ⌢ H2 →H2)
   ⇒  {p1, p2} P* || Q* {p1, p2; H1 ∨ (ℓ = Real 0), H2 ∨ (ℓ = Real 0)}
```

The above axiom says that the final specification for `P*|| Q*` holds, if the same specification holds for one round of execution, i.e. `P || Q`, and moreover, `H` is idempotent with respect to chop modality. The formula `ℓ= Real 0` indicates that the repetition iterates zero time.

Soundness. To prove the soundness of HHL proof system, we need to have a big step operational semantics for HCSP first, which can be derived directly from the small step semantics given in Sec. 5. Besides, considering that the interpretation of pre-/post-conditions and history formulas are irrelevant to the system behavior and also the events, we will get rid of them in the big step operational semantics, represented by function `evalPB`.

```
axioms
  base: evalP (P, f, sf, d, α) = (ϵ, f', sf', d') ⇔
      evalPB (P, f, d) = (ϵ, f', d')
  ind: evalP (P, f, sf, d, α) = (P', f', sf', d') & evalPB (P', f', d') =
      (ϵ, f'', d'') ⇔ evalPB (P, f, d) = (ϵ, f'', d'')
```

The axioms `base` and `ind` define the cases when `P` terminates after one step transition, and after more than one step transitions respectively.

We can then define the validity of a specification `{p} P {q;H}` with respect to the big operational semantics, as follows:

```
definition Valid :: fform ⇒proc ⇒fform ⇒dform ⇒bool
where Valid (p, P, q, H) = ∀f d f' d'. evalPB (P, f, d) = (ϵ, f', d') ⇒
        evalF (f, p, d) ⇒(evalF (f', q, d') & ievalF (f', H, d, d'))
```

which says that, given a process `P`, for any initial behavior `f` and initial time `d`, if `P` terminates at behavior `f'` and time `d'`, and if the precondition `p` holds under the initial state, i.e. `last(f(d))`, the last state among the state list at initial time, then the postcondition `q` will hold under the final state, i.e. `last(f'(d'))`, the last state among the state list at termination time, and furthermore, the history formula will hold under `f'` between `d` and `d'`.

Based on the above definitions, we have proved the soundness of HHL proof system in Isabelle/HOL, i.e. all the inference rules of the proof system are valid.

8 Case Study: Chinese Train Control System

In this section, we illustrate our approach by modelling and verifying a combined operational scenario of Chinese Train Control System at Level 3 (CTCS-3) with respect to its System Requirement Specification (SRS).

A train at CTCS-3 applies for movement authorities (MAs) from Radio Block Center (RBC) via GSM-Railway (GSM-R) and is guaranteed to move safely in high speed within its MA. CTCS-2 is a backup system of CTCS-3, under which a train applies for MAs from Train Control Center (TCC) via train circuits and balises instead. There are 9 main operating modes in CTCS-3, among which the Full Supervision (FS) and Calling On (CO) modes will be involved in the combined scenario studied in this section. During FS mode, a train needs to know the complete information including its MA, line data, train data and so on; while during CO mode, the on-board equipment of the train cannot confirm explicit routes, thus a train is required to move under constant speed 40km/h.

The operating behavior of CTCS-3 is specified by 14 basic scenarios, all of which cooperate with each other to constitute normal functionality of train control system. The combined scenario considered here integrates the Movement Authority and Level Transition scenarios of CTCS-3, plus a special Mode Transition scenario.

For modeling a scenario, we model each component involved in it as an HCSP process and then combine different parts by parallel composition to form the model of the scenario. In particular, the train participates in each scenario, and the HCSP model corresponding to the train under different scenarios has a very unified structure. Let s be trajectory, v velocity, a acceleration, t clock time of a train respectively. Then we have the following general model for the train:

$$Train \mathrel{\widehat{=}} \left(\begin{array}{l} \langle \dot{s} = v, \dot{v} = a, \dot{t} = 1 \,\&\, B \rangle \trianglerighteq [\!]_{i \in I} (io_i \to P_{comp_i}); \\ Q_{comp} \end{array} \right)^*$$

where P_{comp_i} and Q_{comp} are discrete computation that takes no time to complete. The train process proceeds as follows: at first the train moves continuously at velocity v and acceleration a; as soon as domain B is violated, or a communication among $\{io_i\}_{i\in I}$ between the train and another component of CTCS-3 takes place, then the train movement is interrupted and shifted to Q_{comp}, or P_{comp_i} respectively; after the discrete computation is done, the train repeats the above process, indicated by * in the model. For each specific scenario, the domain B, communications io_i, and computation P_{comp_i} and Q_{comp} can be instantiated correspondingly. We assume the acceleration a is always in the range $[-b, A]$.

In the rest of this section, we will first model three basic scenarios separately, and then construct a combined scenario from them.

8.1 Movement Authority Scenario

Among all the scenarios, MA is the most basic one and crucial to prohibit trains from colliding with each other. Before moving, the train applies for MA from RBC in CTCS-3 or TCC in CTCS-2, and if it succeeds, it gets the permission to move but only within the MA it owns. An MA is composed of a sequence of segments. Each segment is represented as a tuple $(v_1, v_2, e, mode)$, where v_1 and v_2 represent the speed limits of emergency brake pattern and normal brake pattern by which the train must implement emergency brake and normal brake (thus v_1 is always greater than v_2), e the end point of the segment, and $mode$ the operating mode of the train in the segment. We introduce some operations on MAs and segments. Given a non-empty MA α, we define $hd(\alpha)$ to return the first segment of α, and $tl(\alpha)$ the rest sequence after removing the first segment; and given a segment seg, we define $seg.v_1$ to access the element v_1 of seg, and similarly to other elements.

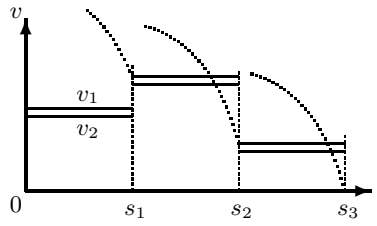

Fig. 9. Static and dynamic speed profiles

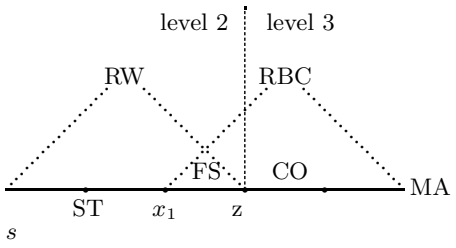

Fig. 10. Level and mode transition

Given an MA, we can calculate its static speed profile and dynamic speed profile respectively. As an illustration, Fig. 9 presents an MA with three segments, separated by points s_1, s_2, and s_3. In the particular case, we assume s_3 the end of the MA, thus the train is required to fully stop at s_3 if the MA is not extended. The static speed profile corresponds to two step functions formed by

the two speed limits (i.e. v_1 and v_2) of each segment; and for any segment seg, the dynamic speed profile is calculated down to the higher speed limit of next segment taking into account the train's maximum deceleration (i.e. constant b), and corresponds to the curved function $v^2+2b\,s < next(seg).v_1^2+2b\,seg.e$, where $next(seg)$ represents the next segment following seg in the considered MA. The train will never be allowed to run beyond the static and dynamic speed profiles.

By specializing the general model of train, we get its specific model in MA scenario. Let B_0 represent the general restriction that the train always moves forward, i.e. $v \geq 0$, or otherwise, the train has already stopped deceleration (denoted by $a \geq 0$). If B_0 fails to hold, the acceleration a needs to be set by a non-negative value in $[0, A]$. Let B_1 denote the case when the speed is less than the lower limit v_2, or otherwise the train has already started to decelerate; and B_2 the case when the speed is less than the higher limit v_1 and not exceeding the dynamic speed profile, or otherwise the train has already started an emergency brake, i.e., the acceleration a is set to be the maximum deceleration b. The above procedure is defined by $Q1_{comp}$ below. For future use, we denote the formula for specifying dynamic speed profile, i.e. $\forall seg : MA\,.\,v^2 + 2b\,s < next(seg).v_1^2 + 2b\,seg.e$, by *DSP_Form*.

$$
\begin{array}{ll}
B_0 & \widehat{=}\ (v \geq 0 \vee a \geq 0 \vee t < Temp + T_{delay}) \\
B_1 & \widehat{=}\ (\forall seg : MA\,.\,v < seg.v_2) \vee a < 0 \vee t < Temp' + T_{delay} \\
B_2 & \widehat{=}\ (\forall seg : MA\,.\,v < seg.v_1 \wedge v^2 + 2b\,s < next(seg).v_1^2 + 2b\,seg.e) \vee a = -b \\
Q1_{comp} & \widehat{=}\ \neg B_0 \rightarrow (Temp := t; \sqcup_{\{0<=c<=A\}} a := c); \\
 & \quad \neg B_1 \rightarrow (Temp' := t; \sqcup_{\{-b<=c<0\}} a := c); \\
 & \quad \neg B_2 \rightarrow a := -b;
\end{array}
$$

Notice that we add T_{delay} to clock t to guarantee that the interrupt B_0 can at most occur once every T_{delay} time units, to avoid Zeno behavior. This is in accordance with the real system to check the condition periodically. We adopt this approach several times. In parallel with the train, the RBC or TCC will send MA to the train periodically via communications, and as a consequence, the train will update the MA it owns. We omit the formalization of this process here as it is hardly related to the combined scenario.

8.2 Level Transition

Under CTCS-2, whenever a train passes some specific balises, it can apply for upgrading to CTCS-3 when necessary. It is assumed balises to be equally distributed every δ meters along the track. Let B_3 represent the negation of the case when the train is at level 2 and passing a specific balise. When B_3 is violated, then as specified in $Q2_{comp}$, the following computation will take place: first, the train sends a level upgrade application signal to RBC; as soon as RBC receives the application, it sends back the package (b, x_1, x_2) to the train, where b represents weather RBC approves the application, x_1 the location for starting level upgrade, and x_2 the location for completing level upgrade; if RBC approves the level upgrade (i.e. b is true), the train enters level 2.5 and meanwhile passes

the balise. Notice that level 2.5 does not actually exist, but is used only for modelling the middle stage between level 2 and level 3, during which the train will be supervised by both CTCS-2 and CTCS-3. Finally, as soon as the train at level 2.5 reaches location x_2 (the negation denoted by B_4), the level will be set to 3, specified in $Q3_{comp}$. RBC_{lu} defines the behavior of RBC under the level transition scenario.

$$
\begin{array}{ll}
B_3 & \widehat{=}\ level \neq 2\ \vee\ s \neq n * \delta \\
B_4 & \widehat{=}\ level \neq 2.5\ \vee\ s \leq LU.x_2 \\
Q2_{comp} & \widehat{=}\ \neg B_3 \rightarrow (CH_{LUA}!; CH_{LU}?LU; LU.b \rightarrow level = 2.5; n = n + 1); \\
Q3_{comp} & \widehat{=}\ \neg B_4 \rightarrow level := 3 \\
RBC_{lu} & \widehat{=}\ CH_{LUA}?; \sqcup_{b_{LU} \in \{true, false\}} CH_{LU}!(b, x_1, x_2)
\end{array}
$$

8.3 Mode Transition

When a train moves under CTCS-2, it will always check whether its operating mode is equal to the mode of current segment, i.e. $hd(MA).mode$. We denote this condition by B_5, and as soon as it is violated, the train will update its mode to be consistent with *mode* of the segment, specified in $Q4_{comp}$.

$$
\begin{array}{ll}
B_5 & \widehat{=}\ mode = hd(MA).mode \\
Q4_{comp} & \widehat{=}\ \neg B_5 \rightarrow mode := hd(MA).mode
\end{array}
$$

We consider the mode transition from Full Supervision (FS) to Calling On (CO) under CTCS-3, which is a little complicated. In the MA application stage, RBC can only grant the train the MAs before the CO segment. The train needs to ask the permission of the driver before moving into a CO segment at level 3. To reflect this specification in modelling, both the speed limits for CO segments are set to be 0. As a consequence, if the train fails to get the permission from the driver, it must stop before the CO segment; but if the train gets the driver's permission, the speed limits of the CO segments will be reset to be positive.

Let B_6 denote the negation of the case when the train is at level 3, and it moves to 300 meters far from the end of current segment, and the mode of next segment is CO. As soon as B_6 is violated, then as specified in $Q5_{comp}$, the following computation will take place: first, the train will report the status to the driver and ask for permission to enter next CO segment via communications; if the driver sends true, the speed limits of next CO segment will be reset to be $40km/h$ and $50km/h$ respectively (abstracted away by function $coma(MA)$). As a consequence, the train is able to enter next CO segment at a positive speed successfully. $Driver_{mc}$ defines the process for the driver under the mode transition scenario.

$$
\begin{array}{ll}
B_6 & \widehat{=}\ level \neq 3 \vee CO \neq hd(tl(MA)).mode \vee hd(MA).e - s > 300 \\
 & \quad \vee t < Temp + T_{delay} \\
Q5_{comp} & \widehat{=}\ CH_{win}!\neg B_6; \neg B_6 \rightarrow Temp := t; CH_{DC}?b_{rConf}; b_{rConf} \rightarrow coma(MA) \\
Driver_{mc} & \widehat{=}\ CH_{win}?b_{win}; b_{win} \rightarrow \sqcup_{b_{sConf} \in \{true,false\}} CH_{DC}!b_{sConf}
\end{array}
$$

8.4 Combined Scenario and Its Model

We combine the scenarios introduced above, but with the following assumptions for the occurring context:

- The train moves inside an MA it owns, and in the combined scenario, it does not need to apply for new MAs from RBC or TCC;
- There are two adjacent segments in the MA, divided by point z. The train is supervised by CTCS-2 to the left of z and by CTCS-3 to the right, and meanwhile, it is operated by mode FS to the left of z and by mode CO to the right. Thus the locations for mode transition and for level transition are coincident. At the starting point of a CO segment, i.e., location z, both speed limits are initialized to 0 by RBC;
- The train has already got the permission for level transition from RBC which sends $(true, x_1, z)$.

Please see Fig. 10 for an illustration. Based on these assumptions, the train will not cooperate with RBC or TCC temporarily in this combined scenario. Thus, only the train and the driver participate in the combined scenario.

The model of the combined scenario can then be constructed from the models of all the basic scenarios contained in it. The construction takes the following steps: firstly, decompose the process for each basic scenario to a set of sub-processes corresponding to different system components that are involved in the scenario (usually by removing parallel composition on top); secondly, as a component may participate in different basic scenarios, re-construct the process for it based on the sub-processes corresponding to it under these scenarios (usually by conjunction of continuous domain constraints and sequential composition of discrete computation actions); lastly, combine the new obtained processes for all the components via parallel composition. According to this construction process, we get the following HCSP model for the combined scenario:

$$
\begin{array}{ll}
System & \widehat{=}\ Train^* \parallel Driver^*_{mc} \\
Train & \widehat{=}\ \langle \dot{s} = v, \dot{v} = a, \dot{t} = 1 \,\&\, B_0 \wedge B_1 \wedge B_2 \wedge B_4 \wedge B_5 \wedge B_6 \rangle; P_{train} \\
P_{train} & \widehat{=}\ Q1_{comp}; Q3_{comp}; Q4_{comp}; Q5_{comp}
\end{array}
$$

According to SRS of CTCS-3, we hope to prove that the combined scenario satisfies a liveness property, i.e., the train can eventually pass through the location for level transition and mode transition.

8.5 Proof of the Combined Scenario

Under the given assumptions in Section 8.4, we check whether the combined scenario (i.e. model *System*) satisfies a liveness property, i.e., the train will eventually move beyond location z for both level transition and mode transition. In this section, instead of proving the liveness property directly, we provide a machine-checked proof for negation of the livness, which says, after moving for any arbitrary time, the train will always stay before location z. We start from encoding the model *System* and the negation property first.

According to HCSP syntax implemented by `proc`, most encoding of model *System* is a direct translation, except for continuous and sequential composition. Firstly, the continuous of *System* needs to be represented in the form of differential invariants. According to the differential invariant generation method, the differential invariant $(a = -b) \rightarrow$ *DSP_Form* is calculated for the continuous, indicating that when the train brakes with maximum deceleration b, it will never exceed the dynamic speed profile. Obviously it is a complement to the domain constraint B_2, saying that the train will never exceed the dynamic speed profile except for the case of emergency brake. We adopt the conjunction of these two formulas, that results in *DSP_Form*, as the final invariant for the continuous. Thus we represent the continuous as `<Inv&B> : Rg`, where `Inv` and `B` correspond to encodings of *DSP_Form* and the domain constraints respectively, and `Rg` is `True`, specifying the executing time of the continuous; Secondly, the intermediate formulas for all sequential composition are added. We finally get the encoding of *System*, represented by `System`, with structure `Train`*`|| Driver`*.

Now it is turn to encode the negation property, specified by pre/post-conditions, and history formula. The precondition is separated into two parts depending on whether it is relative to initial values, shown by `Init` and `Pre` below:

```
definition Init :: fform where Init ≡(x2 - s > Real 300)
definition Pre :: fform where
Pre ≡ (level = Real 2.5) ∧ (fst (snd (snd (hd (MA)))) = x2)
      ∧ (snd (snd (snd (hd (MA)))) = String ‘‘FS’’)
      ∧ (snd (snd (snd (hd (tl (MA))))) = String ‘‘CO’’)
      ∧ (fst (hd (MA)) = Real 0) ∧ (fst (snd (hd (MA))) = Real 0)
```

The `Init` represents that the initial position of the train (i.e. `s`) is more than 300 meters away from `x2`. The `Pre` indicates the following aspects: the train moves at level `2.5`, i.e. in process of level transition from CTCS-2 to CTCS-3; the end of current segment is `x2`; the mode of the train in current segment is ‘‘`FS`’’; the mode of the train in next segment is ‘‘`CO`’’; and at the end of current segment, both speed limits are initialized to be `0`. Notice that for any segment *seg*, $seg.v_1$ is implemented as `fst (seg)`, and $seg.v_2$ as `fst (snd (seg))`, and so on.

We then get a specification corresponding to the negation property, with the postcondition and history formula for the train to indicate that the train will never pass through location x_2:

```
theorem System : {Init ∧ Pre, True} System {Pre ∧ s <= x2,
                 True; (ℓ = Real 0) ∨ (high (Pre ∧ s <= x2)), True}
```

In Isabelle/HOL, we have proved this specification as a theorem. From this fact, we know that the model `System` for level transition and mode transition fails to conform to the liveness property. This reflects some design flaw for the specifications of related scenarios in CTCS-3.

9 Other Issues: Stability Analysis

In the previous sections, we have discussed the issues of modeling, invariant generation, deductive verification and controller synthesis of hybrid systems. The focus has been on safety properties, that is, properties need to hold at all time. Other important properties of hybrid systems include: *reachability*, which asks whether a given set of target states will be reached in finite time; *stability*, which reflects the influence of small perturbations of initial conditions on the system's trajectories; or *asymptotic stability* which, beyond stability, also cares about the system's convergence behavior when time approaches infinity; and so on. The issues of verification and controller synthesis of hybrid systems for reachability specifications have been investigated in works such as [51,68,29,82,35]. In this section, we will exploit the same techniques developed for invariant generation in Section 3, to automatically generate so-called *relaxed Lyapunov functions* for asymptotic stability analysis of PCDSs. For stability analysis of hybrid systems using tools like *multiple Lyapunov functions*, the readers are referred to such works as [15,16], and the survey papers [27,77] and the citations therein.

9.1 Lyapunov Stability

The following are classic results of stability theory in the sense of Lyapunov. For the details please refer to [50].

Definition 17. *A point $\mathbf{x}_e \in \mathbb{R}^n$ is called an equilibrium point or critical point of a CDS (1) if $\mathbf{f}(\mathbf{x}_e) = \mathbf{0}$.*

It is assumed that $\mathbf{x}_e = \mathbf{0}$ from now on without loss of generality.

Definition 18. *Suppose $\mathbf{0}$ is an equilibrium point of (1). Then*

- *$\mathbf{0}$ is called* **Lyapunov stable** *if for any $\epsilon > 0$, there exists a $\delta > 0$ such that if $\|\mathbf{x}_0\| < \delta$,[12] then the solution $\mathbf{x}(\mathbf{x}_0; t)$ of (1) can be extended to infinity, and $\|\mathbf{x}(\mathbf{x}_0; t)\| < \epsilon$ for all $t \geq 0$.*
- *$\mathbf{0}$ is called* **asymptotically stable** *if it is Lyapunov stable and there exists a $\delta > 0$ such that for any $\|\mathbf{x}_0\| < \delta$, the solution $\mathbf{x}(\mathbf{x}_0; t)$ of (1) satisfies $\lim_{t\to\infty} \mathbf{x}(\mathbf{x}_0; t) = \mathbf{0}$.*

Lyapunov first provided a sufficient condition, using so-called *Lyapunov functions*, for the Lyapunov stability as follows.

[12] For $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, $\|\mathbf{x}\| = \sqrt{\sum_{i=1}^n x_i^2}$ denotes the Euclidean norm of $\mathbf{x}$.

Theorem 12 (Lyapunov Stability Theorem). *Suppose* $\mathbf{0}$ *is an equilibrium point of (1). If there is an open set* $U \subset \mathbb{R}^n$ *with* $\mathbf{0} \in U$*, and a continuously differentiable function* $V : U \to \mathbb{R}$ *such that*

(a) $V(\mathbf{0}) = 0$,
(b) $V(\mathbf{x}) > 0$ *for all* $\mathbf{x} \in U \backslash \{\mathbf{0}\}$ *and*
(c) $L^1_{\mathbf{f}} V(\mathbf{x}) \leq 0$ *for all* $\mathbf{x} \in U$,

then $\mathbf{0}$ *is a stable equilibrium point. Moreover, if condition (c) is replaced by*

(c)* $L^1_{\mathbf{f}} V(\mathbf{x}) < 0$ *for all* $\mathbf{x} \in U \backslash \{\mathbf{0}\}$,

then $\mathbf{0}$ *is an asymptotically stable equilibrium point. Such* V *satisfying (a), (b) and (c) (or (c*)) is called a* **Lyapunov function**.

Basically, for asymptotic stability of an equilibrium point of a CDS, Theorem 12 requires a *positive definite* function V with *negative definite* first-order Lie derivative $L^1_{\mathbf{f}} V$ in a neighborhood of the equilibrium. If V has only *negative semi-definite* $L^1_{\mathbf{f}} V$ but no trajectories can stay identically in the zero level set of $L^1_{\mathbf{f}} V$, then the asymptotic stability can also be guaranteed, which is known as the Barbashin-Krasovskii-LaSalle (BKL) Principle.

Theorem 13 (BKL Principle). *Let* V *be such a function as stated in Theorem 12 with conditions (a), (b) and (c). If the set* $\mathcal{M} \widehat{=} \{\mathbf{x} \in U \mid L^1_{\mathbf{f}} V(\mathbf{x}) = 0\}$ *does not contain any trajectory of the system other than the trivial trajectory* $\mathbf{x}(t) \equiv \mathbf{0}$*, then* $\mathbf{0}$ *is asymptotically stable.*

9.2 Relaxed Lyapunov Function

Intuitively, a Lyapunov function in Theorem 12 with conditions (a), (b), (c) requires any trajectory starting from $\mathbf{x}_0 \in U$ to stay in the region $\{\mathbf{x} \in \mathbb{R}^n \mid V(\mathbf{x}) \leq V(\mathbf{x}_0)\}$. In the asymptotic stability case, the corresponding V forces any trajectory starting from $\mathbf{x}_0 \in U \backslash \{\mathbf{0}\}$ to transect the boundary $\{\mathbf{x} \in \mathbb{R}^n \mid V(\mathbf{x}) = V(\mathbf{x}_0)\}$, called a *Lyapunov surface*, towards the set $\{\mathbf{x} \in \mathbb{R}^n \mid V(\mathbf{x}) < V(\mathbf{x}_0)\}$. The left picture in Figure 11 illustrates how a Lyapunov function guarantees asymptotic stability.

For any $\mathbf{x}_0 \in U \setminus \{\mathbf{0}\}$, it is not difficult to see that $L^1_{\mathbf{f}} V(\mathbf{x}_0) < 0$ is only a sufficient condition for $\mathbf{x}(\mathbf{x}_0; t)$ to move towards the set $V(\mathbf{x}) < V(\mathbf{x}_0)$. When $L^1_{\mathbf{f}} V(\mathbf{x}_0) = 0$, the transection requirement may still be met if the first non-zero higher order Lie derivative of V at $\mathbf{x}_0$ is negative. In this case, the trajectory may be tangential to a Lyapunov surface at the cross point (see the right picture in Fig. 11). To formalize the above idea, and motivated by the results on invariant generation in Section 3, the following definitions are proposed.

Definition 19 (Pointwise Rank). *Let* $\mathbb{N}^+$ *be the set of positive natural numbers. Given a smooth function* σ *and a smooth vector field* $\mathbf{f}$*, the pointwise rank of* σ *w.r.t.* $\mathbf{f}$ *is defined as the function* $\nu_{\sigma,\mathbf{f}} : \mathbb{R}^n \to \mathbb{N}^+ \cup \{\infty\}$ *given by*

$$\nu_{\sigma,\mathbf{f}}(\mathbf{x}) = \begin{cases} \infty, & \text{if } \forall k \in \mathbb{N}^+ .\, L^k_{\mathbf{f}} \sigma(\mathbf{x}) = 0, \\ \min\{k \in \mathbb{N}^+ \mid L^k_{\mathbf{f}} \sigma(\mathbf{x}) \neq 0\}, & \text{otherwise.} \end{cases}$$

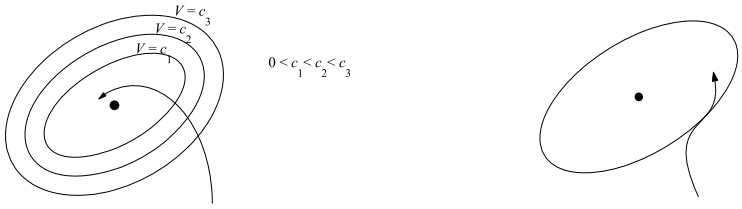

Fig. 11. Trajectories transecting Lyapunov surfaces

Example 11. For $\mathbf{f} = (-x, y)$ and $p(x, y) = x + y^2$, by Example 1, we have $\nu_{p,\mathbf{f}}(0, 0) = \infty$, $\nu_{p,\mathbf{f}}(1, 1) = 1$, $\nu_{p,\mathbf{f}}(2, 1) = 2$.

Actually, $\nu_{\sigma,\mathbf{f}}$ is almost the same as the pointwise rank function $\gamma_{p,\mathbf{f}}$ defined in Section 3.2. The only difference is that for $\nu_{\sigma,\mathbf{f}}$, the zeroth order Lie derivative is not considered.

Definition 20 (Transverse Set). *Given a smooth function σ and a smooth vector field $\mathbf{f}$, the* transverse set *of σ w.r.t $\mathbf{f}$ is defined as*

$$\mathrm{Trans}_{\sigma,\mathbf{f}} \widehat{=} \{\mathbf{x} \in \mathbb{R}^n \mid \nu_{\sigma,\mathbf{f}}(\mathbf{x}) < \infty \wedge L_{\mathbf{f}}^{\nu_{\sigma,\mathbf{f}}(\mathbf{x})}\sigma(\mathbf{x}) < 0\} \ .$$

Actually, $\mathrm{Trans}_{\sigma,\mathbf{f}}$ is defined in the same manner as the transverse set $Trans_{\mathbf{f}\uparrow p}$ in Definition 14, using a different definition of pointwise rank function.

Using transverse set, condition (c*) in Theorem 12 can be relaxed to give a new criterion for asymptotic stability.

Theorem 14. *Suppose $\mathbf{0}$ is an equilibrium point of (1) with smooth vector field $\mathbf{f}$. If there is an open set $U \subset \mathbb{R}^n$ with $\mathbf{0} \in U$, and a smooth function $V : U \to \mathbb{R}$ such that*

(a) $V(\mathbf{0}) = 0$,
(b) $V(\mathbf{x}) > 0$ *for all* $\mathbf{x} \in U\backslash\{\mathbf{0}\}$ *and*
(c) $\mathbf{x} \in \mathrm{Trans}_{V,\mathbf{f}}$ *for all* $\mathbf{x} \in U\backslash\{\mathbf{0}\}$,

then $\mathbf{0}$ is an asymptotically stable equilibrium.

Proof. First notice that condition (c) implies $L_{\mathbf{f}}^1 V(\mathbf{x}) \leq 0$ for all $\mathbf{x} \in U\backslash\{\mathbf{0}\}$. Then according to Theorem 13, in order to show the asymptotic stability of $\mathbf{0}$, it is sufficient to show that $\mathcal{M} \widehat{=} \{\mathbf{x} \in U \mid L_{\mathbf{f}}^1 V(\mathbf{x}) = 0\}$ contains no nontrivial trajectory of (1).

If not, let $\mathbf{x}(t)$, $t \geq 0$ be such a trajectory contained in $\mathcal{M}$ other than $\mathbf{x}(t) \equiv \mathbf{0}$. Then for all $t \geq 0$, $L_{\mathbf{f}}^1 V(\mathbf{x}(t)) = 0$ and $\mathbf{x}(t) \neq \mathbf{0}$. By (c), $\mathbf{x}_0 \widehat{=} \mathbf{x}(0) \in \mathrm{Trans}_{V,\mathbf{f}}$. Then by Definition 20, we get the *Taylor Formula* of $L_{\mathbf{f}}^1 V(\mathbf{x}(t))$ at $t = 0$:

$$\begin{aligned} L_{\mathbf{f}}^1 V(\mathbf{x}(t)) &= L_{\mathbf{f}}^1 V(\mathbf{x}_0) + L_{\mathbf{f}}^2 V(\mathbf{x}_0)\cdot t + \cdots \\ &\quad + L_{\mathbf{f}}^{\nu_{V,\mathbf{f}}(\mathbf{x}_0)} V(\mathbf{x}_0)\cdot \frac{t^{\nu_{V,\mathbf{f}}(\mathbf{x}_0)-1}}{(\nu_{V,\mathbf{f}}(\mathbf{x}_0)-1)!} + o(t^{\nu_{V,\mathbf{f}}(\mathbf{x}_0)-1}) \\ &= L_{\mathbf{f}}^{\nu_{V,\mathbf{f}}(\mathbf{x}_0)} V(\mathbf{x}_0)\cdot \frac{t^{\nu_{V,\mathbf{f}}(\mathbf{x}_0)-1}}{(\nu_{V,\mathbf{f}}(\mathbf{x}_0)-1)!} + o(t^{\nu_{V,\mathbf{f}}(\mathbf{x}_0)-1})\,. \end{aligned} \tag{19}$$

Since $L_{\mathbf{f}}^{\nu_{V,\mathbf{f}}(\mathbf{x}_0)} V(\mathbf{x}_0) < 0$, the formula (19) shows that there exists an $\epsilon > 0$ s.t. $\forall t \in (0,\epsilon).\, L_{\mathbf{f}}^1 V(\mathbf{x}(t)) < 0$, which contradicts the fact $\forall t \geq 0.\, L_{\mathbf{f}}^1 V(\mathbf{x}(t)) = 0$. □

Definition 21 (Relaxed Lyapunov Function). *We refer to the function V in Theorem 14 as a relaxed Lyapunov function, denoted by RLF for short.*

9.3 Automatically Discovering Polynomial RLFs for PCDSs

Given a PCDS, the process of automatically discovering polynomial RLFs is as follows:

I. A parametric polynomial $p(\mathbf{u},\mathbf{x})$ (also called a template) is predefined as a candidate for RLF;
II. The conditions for $p(\mathbf{u},\mathbf{x})$ to be an RLF, i.e. (a), (b) and (c) in Theorem 14, are encoded into a first-order polynomial formula φ;
III. Constraint ϕ on the parameters $\mathbf{u}$ is obtained by applying QE to φ, and any instantiation of $\mathbf{u}$ from ϕ yields an RLF $p_{\mathbf{u}_0}(\mathbf{x})$.

Step II in the above process, i.e. encoding of the three conditions in Theorem 14, is crucial to automatic RLF generation. In particular, we have to show that for any polynomial $p(\mathbf{x})$ and PVF $\mathbf{f}$, the transverse set $\mathrm{Trans}_{p,\mathbf{f}}$ can be represented by first-order polynomial formulas. In fact, all the results established for $Trans_{\mathbf{f}\uparrow p}$ in Section 3.3 apply to $\mathrm{Trans}_{p,\mathbf{f}}$ here.

Theorem 15 (Fixed Point Theorem). *Given a polynomial p and a PVF $\mathbf{f}$, if $L_{\mathbf{f}}^{i+1} p \in \langle L_{\mathbf{f}}^1 p, \cdots, L_{\mathbf{f}}^i p\rangle$, then for all $m > i$, $L_{\mathbf{f}}^m p \in \langle L_{\mathbf{f}}^1 p, \cdots, L_{\mathbf{f}}^i p\rangle$.*

Theorem 16 (Rank Theorem). *Given a polynomial p and a PVF $\mathbf{f}$, for any $\mathbf{x} \in \mathbb{R}^n$, if $\nu_{p,\mathbf{f}}(\mathbf{x}) < \infty$ then $\nu_{p,\mathbf{f}}(\mathbf{x}) \leq N_{p,\mathbf{f}}$, where*

$$N_{p,\mathbf{f}} \widehat{=} \min\{i \in \mathbb{N}^+ \mid L_{\mathbf{f}}^{i+1} p(\mathbf{x}) \in \langle L_{\mathbf{f}}^1 p(\mathbf{x}), \cdots, L_{\mathbf{f}}^i p(\mathbf{x})\rangle\}\,.$$

Theorem 17 (Parametric Rank Theorem). *Given a parametric polynomial $p \widehat{=} p(\mathbf{u},\mathbf{x})$ and a PVF $\mathbf{f}$, for all $\mathbf{x} \in \mathbb{R}^n$ and all $\mathbf{u}_0 \in \mathbb{R}^w$, $\nu_{p_{\mathbf{u}_0},\mathbf{f}}(\mathbf{x}) < \infty$ implies $\nu_{p_{\mathbf{u}_0},\mathbf{f}}(\mathbf{x}) \leq N_{p,\mathbf{f}}$, where*

$$N_{p,\mathbf{f}} \widehat{=} \min\{i \in \mathbb{N}^+ \mid L_{\mathbf{f}}^{i+1} p(\mathbf{u},\mathbf{x}) \in \langle L_{\mathbf{f}}^1 p(\mathbf{u},\mathbf{x}), \cdots, L_{\mathbf{f}}^i p(\mathbf{u},\mathbf{x})\rangle\}\,. \tag{20}$$

Theorem 18. *Given a parametric polynomial* $p \widehat{=} p(\mathbf{u}, \mathbf{x})$ *and a PVF* $\mathbf{f}$*, for any* $\mathbf{u}_0 \in \mathbb{R}^w$ *and any* $\mathbf{x} \in \mathbb{R}^n$*,* $\mathbf{x} \in \mathrm{Trans}_{p_{\mathbf{u}_0},\mathbf{f}}$ *if and only if* $\mathbf{u}_0$ *and* $\mathbf{x}$ *satisfy* $\varphi_{p,\mathbf{f}}$*, where*

$$\varphi_{p,\mathbf{f}} \widehat{=} \bigvee_{1 \le i \le N_{p,\mathbf{f}}} \varphi^i_{p,\mathbf{f}} \quad \textit{with} \tag{21}$$

$$\varphi^i_{p,\mathbf{f}} \widehat{=} \Big(\bigwedge_{1 \le j \le i-1} L^j_{\mathbf{f}} p(\mathbf{u}, \mathbf{x}) = 0 \Big) \wedge L^i_{\mathbf{f}} p(\mathbf{u}, \mathbf{x}) < 0$$

and $N_{p,\mathbf{f}}$ *defined in (20).*

All the proofs of the above theorems can be given in exactly the same way as in Section 3.3. The details are omitted here and can be found in [57].

Now the main result on automatically generating polynomial RLFs for PCDSs can be stated as the following theorem.

Theorem 19 (Main Result). *Given a PCDS* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ *with* $\mathbf{f}(\mathbf{0}) = \mathbf{0}$*, a parametric polynomial* $p \widehat{=} p(\mathbf{u}, \mathbf{x})$*, and* $\mathbf{u}_0 = (u_{1_0}, u_{2_0}, \ldots, u_{w_0}) \in \mathbb{R}^w$*, then* $p_{\mathbf{u}_0}$ *is an RLF of the PCDS if and only if there exists* $r_0 \in \mathbb{R}, r_0 > 0$*, such that* $(u_{1_0}, u_{2_0}, \ldots, u_{w_0}, r_0)$ *satisfies* $\phi_{p,\mathbf{f}} \widehat{=} \phi^1_{p,\mathbf{f}} \wedge \phi^2_{p,\mathbf{f}} \wedge \phi^3_{p,\mathbf{f}}$*, where*

$$\phi^1_{p,\mathbf{f}} \widehat{=} p(\mathbf{u}, \mathbf{0}) = 0\,, \tag{22}$$

$$\phi^2_{p,\mathbf{f}} \widehat{=} \forall \mathbf{x}.(\|\mathbf{x}\|^2 > 0 \wedge \|\mathbf{x}\|^2 < r^2 \to p(\mathbf{u}, \mathbf{x}) > 0)\,, \tag{23}$$

$$\phi^3_{p,\mathbf{f}} \widehat{=} \forall \mathbf{x}.(\|\mathbf{x}\|^2 > 0 \wedge \|\mathbf{x}\|^2 < r^2 \to \varphi_{p,\mathbf{f}}) \tag{24}$$

with $\varphi_{p,\mathbf{f}}$ *defined in (21).*

Proof. First, in Theorem 14, the existence of an open set U is equivalent to the existence of an open ball $\mathcal{B}(\mathbf{0}, r_0) \widehat{=} \{\mathbf{x} \in \mathbb{R}^n \mid \|\mathbf{x}\| < r_0\}$. Then according to Theorem 18, it is easy to check that (22), (23) and (24) are direct translations of conditions (a), (b) and (c) in Theorem 14. □

According to Theorem 19, we can follow the three steps at the beginning of Section 9.3 to discover polynomial RLFs for PCDSs. This method is *relatively complete* because we can discover all possible polynomial RLFs in the form of a predefined template, and thus can find all polynomial RLFs by enumerating all polynomial templates for a given PCDS.

9.4 Simplification and Implementation

When constructing $\phi_{p,\mathbf{f}}$ in Theorem 19, computation of $N_{p,\mathbf{f}}$ is a time-consuming work. Furthermore, when $N_{p,\mathbf{f}}$ is a large number the resulting $\phi_{p,\mathbf{f}}$ could be a huge formula, for which QE is infeasible in practice. Regarding this, in the following the complexity of RLF generation is reduced in two aspects:

1) some of the QE problems arising in RLF generation can be reduced to so-called *real root classification* (RRC for short) problems, which can be solved in a more efficient way than standard QE problems;

2) RLF can be searched for in a stepwise manner: if an RLF can be obtained by solving constraints involving only lower order Lie derivatives, there is no need to resort to higher order ones.

The following three lemmas are needed to explain the first aspect.

Lemma 5. *Suppose* $\mathbf{f}$ *is a smooth vector field,* σ *is a smooth function defined on an open set* $U \subseteq \mathbb{R}^n$*, and* $L^1_{\mathbf{f}}\sigma(\mathbf{x}) \leq 0$ *for all* $\mathbf{x} \in U$*. Then for any* $\mathbf{x} \in U$*,* $\nu_{\sigma,\mathbf{f}}(\mathbf{x}) < \infty$ *implies* $\mathbf{x} \in \mathrm{Trans}_{\sigma,\mathbf{f}}$ *.*

Proof. Suppose there is an $\mathbf{x}_0 \in U$ such that $\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0) < \infty$ and $L_{\mathbf{f}}^{\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)}\sigma(\mathbf{x}_0) > 0$. Let $\mathbf{x}(t)$ be the trajectory of $\mathbf{f}$ starting from $\mathbf{x}_0$. Then from

$$\begin{aligned} L^1_{\mathbf{f}}\sigma(\mathbf{x}(t)) &= L^1_{\mathbf{f}}\sigma(\mathbf{x}_0) + L^2_{\mathbf{f}}\sigma(\mathbf{x}_0)\cdot t + \cdots \\ &\quad + L_{\mathbf{f}}^{\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)}\sigma(\mathbf{x}_0)\cdot\frac{t^{\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)-1}}{(\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)-1)!} + o(t^{\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)}-1) \\ &= L_{\mathbf{f}}^{\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)}\sigma(\mathbf{x}_0)\cdot\frac{t^{\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)-1}}{(\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)-1)!} + o(t^{\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)}-1) \qquad (25) \end{aligned}$$

we can see that there exists an $\epsilon > 0$ such that $\forall t \in (0,\epsilon).\, L^1_{\mathbf{f}}\sigma(\mathbf{x}(t)) > 0$, which contradicts $L^1_{\mathbf{f}}\sigma(\mathbf{x}) \leq 0$ for all $\mathbf{x} \in U$. □

Lemma 6. *Suppose* $\mathbf{f}$ *is a smooth vector field,* σ *is a smooth function defined on an open set* $U \subseteq \mathbb{R}^n$*, and* $L^1_{\mathbf{f}}\sigma(\mathbf{x}) \leq 0$ *for all* $\mathbf{x} \in U$*. Then for any* $\mathbf{x} \in U$*,* $\nu_{\sigma,\mathbf{f}}(\mathbf{x}) < \infty$ *implies* $\nu_{\sigma,\mathbf{f}}(\mathbf{x}) = 2k+1$ *for some* $k \in \mathbb{N}$ *.*

Proof. If there is an $\mathbf{x}_0 \in U$ such that $\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0) < \infty$ and $\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0) = 2k$ for some $k \in \mathbb{N}^+$, then by Lemma 5 we have $L_{\mathbf{f}}^{\nu_{\sigma,\mathbf{f}}(\mathbf{x}_0)}\sigma(\mathbf{x}_0) < 0$. Then by (25) we can see there exists an $\epsilon > 0$ such that $\forall t \in (-\epsilon, 0).\, L^1_{\mathbf{f}}\sigma(\mathbf{x}(t)) > 0$, which contradicts $L^1_{\mathbf{f}}\sigma(\mathbf{x}) \leq 0$ for all $\mathbf{x} \in U$. □

Lemma 7. *Suppose* $\mathbf{f}$ *is a PVF and* $p(\mathbf{x})$ *is a polynomial, and* $L^1_{\mathbf{f}}p(\mathbf{x}) \leq 0$ *for all* $\mathbf{x}$ *in an open set* $U \subseteq \mathbb{R}^n$*. Then for any* $\mathbf{x} \in U$*,* $\mathbf{x} \in \mathrm{Trans}_{p,\mathbf{f}}$ *if and only if* $\mathbf{x}$ *is not a common root of the sequence of polynomials*

$$L^1_{\mathbf{f}}p(\mathbf{x}),\ L^3_{\mathbf{f}}p(\mathbf{x}),\ \ldots,\ L^{(2K_0+1)}_{\mathbf{f}}p(\mathbf{x})\ ,$$

where $K_0 \widehat{=} \lfloor \frac{N_{p,\mathbf{f}}-1}{2} \rfloor$[13] *and* $N_{p,\mathbf{f}}$ *is defined in Theorem 16 .*

Proof. ($\Rightarrow$) Actually K_0 has been chosen is such a way that $2K_0+1$ is the largest odd number less than or equal to $N_{p,\mathbf{f}}$, i.e. $2K_0+1 = N_{p,\mathbf{f}}$ or $2K_0+1 = N_{p,\mathbf{f}}-1$. Suppose $\mathbf{x}_0 \in \mathrm{Trans}_{p,\mathbf{f}}$ and $L^1_{\mathbf{f}}p(\mathbf{x}_0) = L^3_{\mathbf{f}}p(\mathbf{x}_0) = \cdots = L^{(2K_0+1)}_{\mathbf{f}}p(\mathbf{x}_0) = 0$. From Lemma 6 we know that $\nu_{p,\mathbf{f}}(\mathbf{x}_0)$ is an odd number. Thus $\nu_{p,\mathbf{f}}(\mathbf{x}_0) \geq 2K_0+1+2 > N_{p,\mathbf{f}}$, which contradicts Theorem 16 .

($\Leftarrow$) If $\mathbf{x}_0$ is not a common root of $L^1_{\mathbf{f}}p(\mathbf{x}),\ L^3_{\mathbf{f}}p(\mathbf{x}),\ \ldots,\ L^{(2K_0+1)}_{\mathbf{f}}p(\mathbf{x})$, then $\nu_{p,\mathbf{f}}(\mathbf{x}_0) < \infty$. By Lemma 5 we get $\mathbf{x}_0 \in \mathrm{Trans}_{p,\mathbf{f}}$. □

[13] For $0 \leq r \in \mathbb{R}$, we have $\lfloor r \rfloor \in \mathbb{N}$ and $r - 1 < \lfloor r \rfloor \leq r$.

Now a simplified version of Theorem 19 can be given as follows.

Theorem 20. *Given a PCDS* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ *with* $\mathbf{f}(\mathbf{0}) = \mathbf{0}$*, a parametric polynomial* $p \widehat{=} p(\mathbf{u}, \mathbf{x})$*, and* $\mathbf{u}_0 = (u_{1_0}, u_{2_0}, \ldots, u_{w_0}) \in \mathbb{R}^w$*, then* $p_{\mathbf{u}_0}$ *is an RLF of the PCDS if and only if there exists* $r_0 \in \mathbb{R}, r_0 > 0$ *such that* $(u_{1_0}, u_{2_0}, \ldots, u_{w_0}, r_0)$ *satisfies* $\psi_{p,\mathbf{f}} \widehat{=} \psi^1_{p,\mathbf{f}} \wedge \psi^2_{p,\mathbf{f}} \wedge \psi^3_{p,\mathbf{f}} \wedge \psi^4_{p,\mathbf{f}}$*, where*

$$\psi^1_{p,\mathbf{f}} \widehat{=} p(\mathbf{u}, \mathbf{0}) = 0\,, \tag{26}$$

$$\psi^2_{p,\mathbf{f}} \widehat{=} \forall \mathbf{x}.(\|\mathbf{x}\|^2 > 0 \wedge \|\mathbf{x}\|^2 < r^2 \to p(\mathbf{u}, \mathbf{x}) > 0)\,, \tag{27}$$

$$\psi^3_{p,\mathbf{f}} \widehat{=} \forall \mathbf{x}.(\|\mathbf{x}\|^2 < r^2 \to L^1_{\mathbf{f}} p(\mathbf{u}, \mathbf{x}) \leq 0)\,, \tag{28}$$

$$\psi^4_{p,\mathbf{f}} \widehat{=} \forall \mathbf{x}.(0 < \|\mathbf{x}\|^2 < r^2 \to L^1_{\mathbf{f}} p(\mathbf{x}) \neq 0 \vee L^3_{\mathbf{f}} p(\mathbf{x}) \neq 0 \vee \cdots \vee L^{(2K_0+1)}_{\mathbf{f}} p(\mathbf{x}) \neq 0) \tag{29}$$

with K_0 *defined in Lemma 7.*

Proof. By combining Theorem 14 with Lemma 7 we can get the results immediately. □

In Theorem 20, constraints (26), (27) and (28) have relatively small sizes and can be solved by QE tools, while (29) can be handled more efficiently as an RRC problem of parametric semi-algebraic systems.

Definition 22. *A parametric semi-algebraic system (PSAS for short) is a conjunction of polynomial formulas of the following form:*

$$\begin{cases} p_1(\mathbf{u}, \mathbf{x}) = 0, ..., p_r(\mathbf{u}, \mathbf{x}) = 0, \\ g_1(\mathbf{u}, \mathbf{x}) \geq 0, ..., g_k(\mathbf{u}, \mathbf{x}) \geq 0, \\ g_{k+1}(\mathbf{u}, \mathbf{x}) > 0, ..., g_l(\mathbf{u}, \mathbf{x}) > 0, \\ h_1(\mathbf{u}, \mathbf{x}) \neq 0, ..., h_m(\mathbf{u}, \mathbf{x}) \neq 0, \end{cases} \tag{30}$$

where $r \geq 1, l \geq k \geq 0, m \geq 0$ *and all* p_i*'s,* g_i*'s and* h_i*'s are in* $\mathbb{Q}[\mathbf{u}, \mathbf{x}] \setminus \mathbb{Q}$*.*

For a PSAS, the interesting problem is so-called *real root classification*, that is, to determine conditions on the parameters $\mathbf{u}$ such that the given PSAS has certain prescribed number of distinct real solutions. Theories on real root classification of PSASs were developed in [90,91]. A computer algebra tool named DISCOVERER [89] was developed to implement these theories.

Given a PSAS $\mathcal{P}$ with n indeterminates and s polynomial equations, it was argued in [19] that CAD-based QE on $\mathcal{P}$ has complexity doubly exponential in n. In contrast, the RRC approach has complexity singly exponential in n and doubly exponential in t, where t is the dimension of the ideal generated by the s polynomials. Therefore RRC can dramatically reduce the complexity of RRC problems especially when t is much less than n.

For RLF generation, to solve (29) we can define a PSAS

$$\mathcal{P} \widehat{=} \begin{cases} L^1_{\mathbf{f}} p(\mathbf{u}, \mathbf{x}) = 0,\ L^3_{\mathbf{f}} p(\mathbf{u}, \mathbf{x}) = 0,\ \ldots,\ L^{(2K_0+1)}_{\mathbf{f}} p(\mathbf{u}, \mathbf{x}) = 0 \\ -\|\mathbf{x}\|^2 > -r^2, \|\mathbf{x}\|^2 > 0 \end{cases}.$$

Then the command *RealRootClassification*$(\mathcal{P}, 0)$ in DISCOVERER returns conditions on $\mathbf{u}$ and r such that $\mathcal{P}$ has NO solutions. In practice, $\mathcal{P}$ can be constructed in a stepwise manner. That is, $L_{\mathbf{f}}^{(2i+1)}p(\mathbf{u}, \mathbf{x}) = 0$ for $0 \leq i \leq K_0$ can be added to $\mathcal{P}$ one by one. Based on the above ideas, an RLF generation algorithm (Algorithm 1) is proposed to implement Theorem 20 .

Algorithm 1. Relaxed Lyapunov Function Generation

```
1  Input: f ∈ ℚ^n[x_1, ..., x_n] with f(0) = 0, p ∈ ℚ[u_1, ..., u_w, x_1, ..., x_n]
2  Output: Res ⊆ ℝ^{w+1}
3  i := 1; Res := ∅; L_f^1 p := (∇p, f);
4  P := ||x||^2 > 0 ∧ −||x||^2 > −r^2;
5  Res^0 := QE(ψ^1_{p,f} ∧ ψ^2_{p,f} ∧ ψ^3_{p,f});
6  if Res^0 = ∅ then
7      return ∅;
8  else
9      repeat
10         P := P ∧ L_f^i p = 0;
11         Res := Res^0 ∩ RRC(P, 0);
12         if Res ≠ ∅ then
13             return Res;
14         else
15             L_f^{i+1} p := (∇L_f^i p, f);
16             L_f^{i+2} p := (∇L_f^{i+1} p, f);
17             i := i + 2;
18     until L_f^i p ∈ ⟨L_f^1 p, L_f^2 p, ..., L_f^{i−1} p⟩;
19 return ∅;
```

Remark 1. In Algorithm 1,

- $\psi^1_{p,\mathbf{f}}$ $\psi^2_{p,\mathbf{f}}$ and $\psi^3_{p,\mathbf{f}}$ in Line 5 are defined in (26), (27) and (28) respectively;
- QE in line 5 is done in a computer algebra tool like Redlog [30] or QEPCAD [17];
- RRC in line 11 stands for the *RealRootClassification* command in DISCOVERER;
- in Line 18 the loop test can be done by the *IdealMembership* command in Maple™ [61] .

Termination of Algorithm 1 is guaranteed by Theorem 15 and Theorem 17; correctness of Algorithm 1 is guaranteed by Theorem 20 .

9.5 Example

We illustrate the method for RLF generation using the following example.

Example 12. Consider the PCDS

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -x + y^2 \\ -xy \end{pmatrix} \tag{31}$$

with a unique equilibrium point $O(0,0)$. We want to establish the asymptotic stability of O.

First, the linearization of (31) at O has the coefficient matrix

$$A = \begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix}$$

with eigenvalues -1 and 0, so none of the principles of stability for linear systems can be applied. Besides, a homogeneous quadratic Lyapunov function $x^2 + axy + by^2$ for verifying asymptotic stability of (31) does not exist in $\mathbb{R}^2$, because

$$\forall x \forall y. \begin{pmatrix} x^2 + y^2 > 0 \to \big(x^2 + axy + by^2 > 0 \\ \wedge\, 2x\dot{x} + ay\dot{x} + ax\dot{y} + 2by\dot{y} < 0\big) \end{pmatrix}$$

is *false*. However, if we try to find an RLF in $\mathbb{R}^2$ for (31) using the simple template $p \widehat{=} x^2 + ay^2$ with a the indeterminate, then Algorithm 1 returns $a = 1$. This means (31) has an RLF $x^2 + y^2$, and O is asymptotically stable. See Fig. 12 for an illustration.

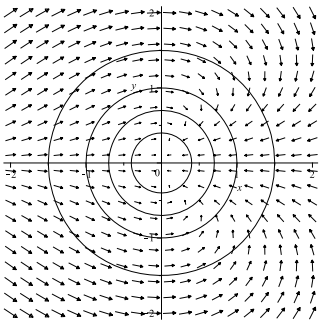

Fig. 12. Vector field and Lyapunov surfaces in Example 12

From this example, we can see that RLFs really extend the class of functions that can be used for asymptotic stability analysis, and the method for automatically discovering polynomial RLFs can save the effort in finding conventional Lyapunov functions in some cases.

10 Conclusion

In this tutorial, we have developed a theoretical and practical foundation for deductive verification of hybrid systems, which includes a selection of topics related to modeling, analysis, and logic of hybrid systems. We choose HCSP as the formal modeling language for hybrid systems, due to its more compositionality and scalability compared to automata-based approach. In order to guarantee the correct functioning of hybrid systems, we have defined a specification logic, called HHL, for specifying and reasoning about the behavior of HCSP, both discrete and continuous, based on first-order logic and DC respectively. However, the logic is not compositional, thus fails to manage more complex HCSP models. The compositionality of the specification logic is one main topic we are working on now.

The specification logic for HCSP uses differential invariants for proving correctness about differential equations instead of their solutions, because solutions of differential equations may not even be expressible. To support this, we have invented a relative complete method for generating polynomial invariants for polynomial differential equations, based on higher-order Lie derivatives and the theory of polynomial ideal.

As a complement of logic-based verification, synthesis provides another approach to ensuring hybrid systems meet given requirements. It focuses on designing a controller for a system such that under the controller, the system is guaranteed to satisfy the given requirement. Based on the differential invariant generation method, we have solved the switching controller synthesis problem with respect to a safety requirement in the context of hybrid automata; and on the other hand, we have also studied the switching controller synthesis problem with respect to an optimality requirement by reducing it to a constraint solving problem.

For tool support, we have implemented a theorem prover for verifying HCSP models in Isabelle/HOL, called HHL prover, which takes an annotated HCSP model in the form of HHL specification as input, and by interactive theorem proving, checks whether the model conforms to the annotated property. The automated verification is not considered yet, and both verification techniques and their implementation to support this will be one of our future work.

Finally, we have demonstrated that our logic-based verification techniques can be used successfully for verifying safety and liveness properties in practical train control systems. In particular, we considered a combined scenario originating from the Chinese High-Speed Train Control System at Level 3 (CTCS-3), and reached a verification result in HHL prover indicating a design error of the combined scenario in CTCS-3. We will consider how to apply our approach to more case studies, among which one direction will be on the safety checking of the other scenarios of CTCS-3 and their all possible combinations.

Acknowledgements. First of all, we thank all the collaborators of the joint work presented in this tutorial for their great contribution. The deductive approach to formal verification of hybrid systems based on Hybrid Hoare Logic

(HHL) was pioneered by Prof. Chaochen Zhou, to whom the basic principles of HHL should be mainly attributed; Prof. Dimitar P. Guelev contributes to the joint work of developing improved versions of HHL; formal modelling and verification of Chinese High-Speed Train Control System (CTCS-3), in particular the scenario reported in Sec. 8, is the result of our long-term collaboration with a research team led by Prof. Tao Tang in Beijing Jiaotong University; Dr. Jidong Lv, Mr. Zhao Quan and Mr. Liang Zou are involved intensively in modelling and verifying CTCS-3; Mr. Liang Zou also helps with the development of HHL prover; Dr. Jiang Liu is one of the major contributors to the work on invariant generation and stability analysis of hybrid systems; the part on (optimal) controller synthesis of hybrid systems is joint work with Prof. Deepak Kapur and Prof. Kim G. Larsen.

We also Thank Prof. Lu Yang, Prof. Bican Xia, Prof. Shaofa Yang, Dr. Ming Xu, Dr. Jiaqi Zhu, Yang Gao, Danqing Guo and many other colleagues for their valuable comments and helpful discussions on the topics of this tutorial.

The work in this tutorial has been supported mainly by projects NSFC-91118007, NSFC-6110006, and National Science and Technology Major Project of China (Grant No. 2012ZX01039-004).

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science 138(1), 3–34 (1995)
2. Alur, R., Dang, T., Esposito, J., Hur, Y., Ivančić, F., Kumar, V., Mishra, P., Pappas, G., Sokolsky, O.: Hierarchical modeling and analysis of embedded systems. Proceedings of the IEEE 91(1), 11–28 (2003)
3. Alur, R., Henzinger, T., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering 22(3), 181–201 (1996)
4. Alur, R.: Formal verification of hybrid systems. In: EMSOFT 2011, pp. 273–278. ACM, New York (2011)
5. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.) HS 1991 and HS 1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993)
6. Alur, R., Dang, T., Ivančić, F.: Counterexample-guided predicate abstraction of hybrid systems. Theor. Comput. Sci. 354(2), 250–271 (2006)
7. Alur, R., Dang, T., Ivančić, F.: Predicate abstraction for reachability analysis of hybrid systems. ACM Trans. Embed. Comput. Syst. 5(1), 152–199 (2006)
8. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)
9. Alur, R., Henzinger, T.A.: Modularity for timed and hybrid systems. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 74–88. Springer, Heidelberg (1997)
10. Asarin, E., Bournez, O., Dang, T., Maler, O., Pnueli, A.: Effective synthesis of switching controllers for linear systems. Proceedings of the IEEE 88(7), 1011–1025 (2000)

11. Asarin, E., Bournez, O., Dang, T., Maler, O.: Approximate reachability analysis of piecewise-linear dynamical systems. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 20–31. Springer, Heidelberg (2000)
12. Audemard, G., Bozzano, M., Cimatti, A., Sebastiani, R.: Verifying industrial hybrid systems with MathSAT. Electronic Notes in Theoretical Computer Science 119(2), 17–32 (2005)
13. Bensalem, S., Bozga, M., Fernández, J.-C., Ghirvu, L., Lakhnech, Y.: A transformational approach for generating non-linear invariants. In: Palsberg, J. (ed.) SAS 2000. LNCS, vol. 1824, pp. 58–72. Springer, Heidelberg (2000)
14. Boulton, R.J., Gordon, A., Gordon, M.J.C., Harrison, J., Herbert, J., Tassel, J.V.: Experience with embedding hardware description languages in HOL. In: Proceedings of the IFIP TC10/WG 10.2 International Conference on Theorem Provers in Circuit Design: Theory, Practice and Experience, pp. 129–156. North-Holland Publishing Co. (1992)
15. Branicky, M.: Stability of switched and hybrid systems. In: CDC 1994, vol. 4, pp. 3498–3503 (1994)
16. Branicky, M.: Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. IEEE Transactions on Automatic Control 43(4), 475–482 (1998)
17. Brown, C.W.: QEPCAD B: A program for computing with semi-algebraic sets using CADs. SIGSAM Bull. 37, 97–108 (2003)
18. Cassez, F., Jessen, J.J., Larsen, K.G., Raskin, J.-F., Reynier, P.-A.: Automatic synthesis of robust and optimal controllers – an industrial case study. In: Majumdar, R., Tabuada, P. (eds.) HSCC 2009. LNCS, vol. 5469, pp. 90–104. Springer, Heidelberg (2009)
19. Chen, Y., Xia, B., Yang, L., Zhan, N.: Generating polynomial invariants with DISCOVERER and QEPCAD. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) Formal Methods and Hybrid Real-Time Systems. LNCS, vol. 4700, pp. 67–82. Springer, Heidelberg (2007)
20. Chutinan, A., Krogh, B.H.: Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 76–90. Springer, Heidelberg (1999)
21. Clarke, E., Fehnker, A., Han, Z., Krogh, B., Stursberg, O., Theobald, M.: Verification of hybrid systems based on counterexample-guided abstraction refinement. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 192–207. Springer, Heidelberg (2003)
22. Clarke, E., Emerson, E.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
23. Colón, M.A., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using non-linear constraint solving. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 420–432. Springer, Heidelberg (2003)
24. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 2nd edn. Springer (1997)
25. Damm, W., Pinto, G., Ratschan, S.: Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. In: Peled, D.A., Tsay, Y.-K. (eds.) ATVA 2005. LNCS, vol. 3707, pp. 99–113. Springer, Heidelberg (2005)

26. de Moura, L., Bjørner, N.S.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
27. DeCarlo, R., Branicky, M., Pettersson, S., Lennartson, B.: Perspectives and results on the stability and stabilizability of hybrid systems. Proceedings of the IEEE 88(7), 1069–1082 (2000)
28. Deshpande, A., Göllü, A., Varaiya, P.: SHIFT: A formalism and a programming language for dynamic networks of hybrid automata. In: Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1996. LNCS, vol. 1273, pp. 113–133. Springer, Heidelberg (1997)
29. Ding, J., Tomlin, C.: Robust reach-avoid controller synthesis for switched nonlinear systems. In: CDC 2010, pp. 6481–6486 (2010)
30. Dolzmann, A., Seidl, A., Sturm, T.: Redlog User Manual, Edition 3.1, for Redlog Version 3.06 (Reduce 3.8) edn. (2006)
31. Eker, J., Janneck, J., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Sachs, S., Xiong, Y., Neuendorffer, S.: Taming heterogeneity — the Ptolemy approach. Proceedings of the IEEE 91(1), 127–144 (2003)
32. Floyd, R.W.: Assigning Meanings to Programs. In: Schwartz, J.T. (ed.) Proceedings of a Symposium on Applied Mathematics, vol. 19, pp. 19–31 (1967)
33. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: HSCC 2011, pp. 43–52. ACM, New York (2011)
34. Fränzle, M., Teige, T., Eggers, A.: Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. The Journal of Logic and Algebraic Programming 79(7), 436–466 (2010)
35. Girard, A.: Controller synthesis for safety and reachability via approximate bisimulation. CoRR abs/1010.4672 (2010), http://arxiv.org/abs/1010.4672
36. Guelev, D., Wang, S., Zhan, N.: Hoare reasoning about HCSP in the duration calculus (submitted, 2013)
37. He, J.: From CSP to hybrid systems. In: A Classical Mind: Essays in Honour of C. A. R. Hoare, pp. 171–189. Prentice Hall International (UK) Ltd., Hertfordshire (1994)
38. Heilmann, S.T.: Proof Support for Duration Calculus. Ph.D. thesis, Technical University of Denmark (1999)
39. Henzinger, T.: The theory of hybrid automata. In: LICS 1996, pp. 278–292 (July 1996)
40. Henzinger, T.A., Ho, P.H.: Algorithmic analysis of nonlinear hybrid systems. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 225–238. Springer, Heidelberg (1995)
41. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: STOC 1995, pp. 373–382. ACM, New York (1995)
42. Henzinger, T.A., Sifakis, J.: The embedded systems design challenge. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 1–15. Springer, Heidelberg (2006)
43. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM 12(10), 576–580 (1969)
44. Jha, S., Seshia, S.A., Tiwari, A.: Synthesis of optimal switching logic for hybrid systems. In: EMSOFT 2011, pp. 107–116. ACM, New York (2011)
45. Julius, A., Girard, A., Pappas, G.: Approximate bisimulation for a class of stochastic hybrid systems. In: American Control Conference 2006, pp. 4724–4729 (2006)

46. Julius, A., Pappas, G.: Probabilistic testing for stochastic hybrid systems. In: CDC 2008, pp. 4030–4035 (2008)
47. Kapur, D., Shyamasundar, R.K.: Synthesizing controllers for hybrid systems. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 361–375. Springer, Heidelberg (1997)
48. Kapur, D.: Automatically generating loop invariants using quantifier elimination. In: Baader, F., Baumgartner, P., Nieuwenhuis, R., Voronkov, A. (eds.) Deduction and Applications (2005)
49. Kapur, D., Zhan, N., Zhao, H.: Synthesizing switching controllers for hybrid systems by continuous invariant generation. CoRR abs/1304.0825 (2013), http://arxiv.org/abs/1304.0825
50. Khalil, H.K.: Nonlinear Systems, 3rd edn. Prentice Hall (December 2001)
51. Koo, T.J., Pappas, G.J., Sastry, S.S.: Mode switching synthesis for reachability specifications. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 333–346. Springer, Heidelberg (2001)
52. Krantz, S., Parks, H.: A Primer of Real Analytic Functions, 2nd edn. Birkhäuser, Boston (2002)
53. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computation for families of linear vector fields. Journal of Symbolic Computation 32(3), 231–253 (2001)
54. Liu, J., Zhan, N., Zhao, H.: Computing semi-algebraic invariants for polynomial dynamical systems. ArXiv e-prints (Febraury 2011), http://arxiv.org/abs/1102.0705
55. Liu, J., Lv, J., Quan, Z., Zhan, N., Zhao, H., Zhou, C., Zou, L.: A calculus for hybrid CSP. In: Ueda, K. (ed.) APLAS 2010. LNCS, vol. 6461, pp. 1–15. Springer, Heidelberg (2010)
56. Liu, J., Zhan, N., Zhao, H.: Computing semi-algebraic invariants for polynomial dynamical systems. In: EMSOFT 2011, pp. 97–106. ACM, New York (2011)
57. Liu, J., Zhan, N., Zhao, H.: Automatically discovering relaxed Lyapunov functions for polynomial dynamical systems. Mathematics in Computer Science 6(4), 395–408 (2012)
58. Lynch, N., Segala, R., Vaandrager, F., Weinberg, H.: Hybrid I/O automata. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) HS 1995. LNCS, vol. 1066, pp. 496–510. Springer, Heidelberg (1996)
59. Maler, O., Manna, Z., Pnueli, A.: From timed to hybrid systems. In: Huizing, C., de Bakker, J.W., Rozenberg, G., de Roever, W.-P. (eds.) REX 1991. LNCS, vol. 600, pp. 447–484. Springer, Heidelberg (1992)
60. Manna, Z., Pnueli, A.: Verifying hybrid systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) HS 1991 and HS 1992. LNCS, vol. 736, pp. 4–35. Springer, Heidelberg (1993)
61. Maplesoft: Maple 14 User Manual, http://www.maplesoft.com/documentation_center/
62. Naur, P.: Proof of algorithms by general snapshots. BIT Numerical Mathematics 6(4), 310–316 (1966)
63. Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: An approach to the description and analysis of hybrid systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) HS 1991 and HS 1992. LNCS, vol. 736, pp. 149–178. Springer, Heidelberg (1993)
64. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. and Comput. 20(1), 309–352 (2010)

65. Platzer, A., Clarke, E.M.: Computing differential invariants of hybrid systems as fixedpoints. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 176–189. Springer, Heidelberg (2008)
66. Platzer, A., Clarke, E.M.: Formal verification of curved flight collision avoidance maneuvers: A case study. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 547–562. Springer, Heidelberg (2009)
67. Prajna, S., Jadbabaie, A., Pappas, G.: A framework for worst-case and stochastic safety verification using barrier certificates. IEEE Transactions on Automatic Control 52(8), 1415–1428 (2007)
68. Prajna, S.: Optimization-based methods for nonlinear and hybrid systems verification. Ph.D. thesis, California Institute of Technology (January 2005)
69. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 477–492. Springer, Heidelberg (2004)
70. Puri, A., Varaiya, P.: Decidability of hybrid systems with rectangular differential inclusions. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, pp. 95–104. Springer, Heidelberg (1994)
71. Queille, J., Sifakis, J.: Specification and verification of concurrent systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Programming 1982. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)
72. Rasmussen, T.M.: Interval Logic — Proof Theory and Theorem Proving. Ph.D. thesis, Technical University of Denmark (2002)
73. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 573–589. Springer, Heidelberg (2005)
74. Sankaranarayanan, S.: Automatic invariant generation for hybrid systems using ideal fixed points. In: HSCC 2010, pp. 221–230. ACM, New York (2010)
75. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constructing invariants for hybrid systems. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 539–554. Springer, Heidelberg (2004)
76. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Non-linear loop invariant generation using Gröbner bases. In: POPL 2004, pp. 318–329. ACM, New York (2004)
77. Shorten, R., Wirth, F., Mason, O., Wulff, K., King, C.: Stability criteria for switched and hybrid systems. SIAM Rev. 49(4), 545–592 (2007)
78. Skakkebaek, J.U., Shankar, N.: Towards a duration calculus proof assistant in PVS. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994. LNCS, vol. 863, pp. 660–679. Springer, Heidelberg (1994)
79. Taly, A., Gulwani, S., Tiwari, A.: Synthesizing switching logic using constraint solving. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 305–319. Springer, Heidelberg (2009)
80. Taly, A., Gulwani, S., Tiwari, A.: Synthesizing switching logic using constraint solving. International Journal on Software Tools for Technology Transfer 13(6), 519–535 (2011)
81. Taly, A., Tiwari, A.: Deductive verification of continuous dynamical systems. In: Kannan, R., Kumar, K.N. (eds.) FSTTCS 2009. LIPIcs, vol. 4, pp. 383–394 (2009)
82. Taly, A., Tiwari, A.: Switching logic synthesis for reachability. In: EMSOFT 2010, pp. 19–28. ACM, New York (2010)
83. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press, Berkeley (1951)
84. Tenenbaum, M., Pollard, H.: Ordinary Differential Equations. Dover Publications (October 1985)

85. Tomlin, C., Lygeros, J., Sastry, S.: A game theoretic approach to controller design for hybrid systems. Proceedings of the IEEE 88(7), 949–970 (2000)
86. Wang, S., Zhan, N., Guelev, D.: An assume/Guarantee based compositional calculus for hybrid CSP. In: Agrawal, M., Cooper, S.B., Li, A. (eds.) TAMC 2012. LNCS, vol. 7287, pp. 72–83. Springer, Heidelberg (2012)
87. Wildmoser, M., Nipkow, T.: Certifying machine code safety: Shallow versus deep embedding. In: Slind, K., Bunker, A., Gopalakrishnan, G.C. (eds.) TPHOLs 2004. LNCS, vol. 3223, pp. 305–320. Springer, Heidelberg (2004)
88. Wolfram: Mathematica Documentation, `http://reference.wolfram.com/mathematica/guide/Mathematica.html`
89. Xia, B.: DISCOVERER: a tool for solving semi-algebraic systems. ACM Commun. Comput. Algebra 41(3), 102–103 (2007)
90. Yang, L.: Recent advances on determining the number of real roots of parametric polynomials. J. Symb. Comput. 28(1-2), 225–242 (1999)
91. Yang, L., Xia, B.: Real solution classification for parametric semi-algebraic systems. In: Dolzmann, A., Seidl, A., Sturm, T. (eds.) Algorithmic Algebra and Logic, pp. 281–289 (2005)
92. Yang, L., Zhou, C., Zhan, N., Xia, B.: Recent advances in program verification through computer algebra. Frontiers of Computer Science in China 4, 1–16 (2010)
93. Zhan, N., Wang, S., Guelev, D.: Extending Hoare logic to hybrid systems. Tech. Rep. ISCAS-SKLCS-13-02, State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences (2013)
94. Zhao, H., Zhan, N., Kapur, D., Larsen, K.G.: A "hybrid" approach for synthesizing optimal controllers of hybrid systems: A case study of the oil pump industrial example. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 471–485. Springer, Heidelberg (2012)
95. Zhao, H., Zhan, N., Kapur, D., Larsen, K.G.: A "hybrid" approach for synthesizing optimal controllers of hybrid systems: A case study of the oil pump industrial example. CoRR abs/1203.6025 (2012), `http://arxiv.org/abs/1203.6025`
96. Zhou, C., Hansen, M.: Duration Calculus — A Formal Approach to Real-Time Systems. Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2004)
97. Zhou, C., Hoare, C., Ravn, A.P.: A calculus of durations. Information Processing Letters 40(5), 269–276 (1991)
98. Zhou, C., Wang, J., Ravn, A.P.: A formal description of hybrid systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) HS 1995. LNCS, vol. 1066, pp. 511–530. Springer, Heidelberg (1996)
99. Zou, L., Lv, J., Wang, S., Zhan, N., Tang, T., Yuan, L., Liu, Y.: Verifying Chinese train control system under a combined scenario by theorem proving. In: Shankar, N. (ed.) VSTTE 2013. LNCS. Springer, Heidelberg (to appear, 2013)