Generalized Property-Directed Reachability for Hybrid Systems

Kohei Suenaga[1,2](✉) and Takuya Ishizawa[1]

[1] Kyoto University, Kyoto, Japan
ksuenaga@gmail.com
[2] JST PRESTO, Tokyo, Japan

Abstract. *Generalized property-directed reachability* (GPDR) belongs to the family of the model-checking techniques called IC3/PDR. It has been successfully applied to software verification; for example, it is the core of Spacer, a state-of-the-art Horn-clause solver bundled with Z3. However, it has yet to be applied to hybrid systems, which involve a continuous evolution of values over time. As the first step towards GPDR-based model checking for hybrid systems, this paper formalizes HGPDR, an adaptation of GPDR to hybrid systems, and proves its soundness. We also implemented a semi-automated proof-of-concept verifier, which allows a user to provide hints to guide verification steps.

Keywords: Hybrid systems · Property-directed reachability · IC3 · Model checking · Verification

1 Introduction

A *hybrid system* is a dynamical system that exhibits both continuous-time dynamics (called a *flow*) and discrete-time dynamics (called a *jump*). This combination of flows and jumps is an essential feature of *cyber-physical systems (CPS)*, a physical system governed by software. In the modern world where safety-critical CPS are prevalent, their correctness is an important issue.

Model checking [14,19] is an approach to guaranteeing hybrid system safety. It tries to prove that a given hybrid system does not violate a specification by abstracting its behavior and by exhaustively checking that the abstracted model conforms to the specification.

In the area of software model checking, an algorithm called *property-directed reachability (PDR)*, also known as *IC3*, is attracting interest [5,7,12]. IC3/PDR was initially proposed in the area of hardware verification; it was then transferred to software model checking by Cimatti et al. [10]. Its effectiveness for software model checking is now widely appreciated. For example, the SMT solver Z3 [29] comes with a Horn-clause solver Spacer [21] that uses PDR internally; Horn-clause solving is one of the cutting-edge techniques to verify functional programs [6,8,17] and programs with loops [6].

We propose a model checking method for hybrid automata [3] based on the idea of PDR; the application of PDR to hybrid automata is less investigated compared to its application to software systems. Concretely, we propose an adaptation of a variant of PDR called *generalized property-directed reachability (GPDR)* proposed by Hoder and Bjørner [20]. Unlike the original PDR, which is specialized to jump-only automata-based systems, GPDR is parametrized over a map over predicates on states (i.e., a *forward predicate transformer*); the detail of the underlying dynamic semantics of a verified system is encapsulated into the forward predicate transformer. This generality of GDPR enables the application of PDR to systems outside the scope of the original PDR by itself; for example, Hoder et al. [20] show how to apply GPDR to programs with recursive function calls.

An obvious challenge in an adaptation of GPDR to hybrid automata is how to deal with flow dynamics that do not exist in software systems. To this end, we extend the logic on which the forward predicate transformer is defined so that it can express flow dynamics specified by an ordinary differential equation (ODE). Our extension, inspired by the differential dynamic logic ($d\mathcal{L}$) proposed by Platzer [32], is to introduce *continuous reachability predicates (CRP)* of the form $\langle \mathcal{D} \mid \varphi_I \rangle \varphi$ where $\mathcal{D}$ is an ODE and φ_I and φ are predicates. This CRP is defined to hold under valuation σ if there is a continuous transition from σ to certain valuation σ' that satisfies the following conditions: (1) the continuous transition is a solution of $\mathcal{D}$, (2) the valuation σ' makes φ true, and (3) φ_I is true at every point on the continuous transition. With this extended logic, we define a forward predicate transformer that faithfully encodes the behavior of a hybrid automaton. We find that we can naturally extend GPDR to hybrid automata by our predicate transformer.

We formalize our adaptation of GPDR to hybrid automata, which we call HGPDR. In the formalization, we define a forward predicate transformer that precisely expresses the behavior of hybrid automata [3] using $d\mathcal{L}$. We prove the soundness of HGPDR. We also describe our proof-of-concept implementation of HGPDR and show how it verifies a simple hybrid automaton with human intervention.

In order to make this paper self-contained, we detail GPDR for discrete-time systems before describing our adaptation to hybrid automata. After fixing the notations that we use in Sect. 2, we define a discrete-time transition system and hybrid automata in Sect. 3. Section 4 then reviews the GPDR procedure. Section 5 presents HGPDR, our adaptation of GPDR to hybrid automata, and states the soundness of the procedure. We describe a proof-of-concept implementation in Sect. 6. After discussing related work in Sect. 7, we conclude in Sect. 8.

For readability, several definitions and proofs are presented in the appendices.

2 Preliminary

We write $\mathbb{R}$ for the set of reals. We fix a finite set $V := \{x_1, \ldots, x_N\}$ of *variables*. We often use primed variables x' and x''. The prime notation also applies to a set of variables; for example, we write V' for $\{x'_1, \ldots, x'_N\}$. We use metavariable

$\boldsymbol{x}$ for a finite sequence of variables. We write **Fml** for the set of quantifier-free first-order formulas over $V \cup V' \cup V''$; its elements are ranged over by φ. We call elements of the set $\Sigma := (V \cup V' \cup V'') \to \mathbb{R}$ a *valuations*; they are represented by metavariable σ. We use the prime notation for valuations. For example, if $\sigma \in V \to \mathbb{R}$, then we write σ' for $\{x'_1 \mapsto \sigma(x_1), \ldots, x'_N \mapsto \sigma(x_N)\}$. We write $\sigma[x \mapsto r]$ for the valuation obtained by updating the entry for x in σ with r. We write $\sigma \models \varphi$ if σ is a model of φ; $\sigma \not\models \varphi$ if $\sigma \models \varphi$ does not hold; $\models \varphi$ if $\sigma \models \varphi$ for any σ; and $\not\models \varphi$ if there exists σ such that $\sigma \not\models \varphi$. We sometimes identify a valuation σ with a logical formula $\bigwedge_{x \in V} x = \sigma(x)$.

3 State-Transition Systems and Verification Problem

We review the original GPDR for discrete-time systems [20] in Sect. 4 before presenting our adaptation for hybrid systems in Sect. 5. This section defines the models used in these explanations (Sects. 3.1 and 3.2) and formally states the verification problem that we tackle (Sect. 3.3).

3.1 Discrete-Time State-Transition Systems (DTSTS)

We model a discrete-time program by a state-transition system.

Definition 3.1. *A* discrete-time state-transition system (DTSTS) *is a tuple* $\langle Q, q_0, \varphi_0, \delta \rangle$. *We use metavariable* $\mathcal{S}_D$ *for DTSTS.* $Q = \{q_0, q_1, q_2, \ldots\}$ *is a set of* locations. q_0 *is the initial location.* φ_0 *is the formula that has to be satisfied by the initial valuation.* $\delta \subseteq Q \times \mathbf{Fml} \times \mathbf{Fml} \times Q$ *is the* transition relation. *We write* $\langle q, \sigma_1 \rangle \to_\delta \langle q', \sigma_2 \rangle$ *if* $\langle q, \varphi, \varphi_c, q' \rangle \in \delta$ *where* $\sigma_1 \models \varphi$ *and* $\sigma_1 \cup \sigma'_2 \models \varphi_c$*; we call relation* $\to_\delta$ *the* jump transition. *A* run *of a DTSTS* $\langle Q, q_0, \varphi_0, \delta \rangle$ *is a finite sequence* $\langle q^0, \sigma_0 \rangle, \langle q^1, \sigma_1 \rangle, \ldots, \langle q^N, \sigma_N \rangle$ *where (1)* $q^0 = q_0$*, (2)* $\sigma_0 \models \varphi_0$*, and (3)* $\langle q^i, \sigma_i \rangle \to_\delta \langle q^{i+1}, \sigma_{i+1} \rangle$ *for any* $i \in [0, N-1]$.

$\langle q, \varphi, \varphi_c, q' \rangle \in \delta$ intuitively means that, if the system is at the location q with valuation σ_1 and $\sigma_1 \models \varphi$, then the system can make a transition to the location q' and change its valuation to σ'_2 such that $\sigma_1 \cup \sigma'_2 \models \varphi_c$. We call φ the *guard* of the transition. φ_c is a predicate over $V \cup V'$ that defines the *command* of the transition; it defines how the value of the variables may change in this transition. The elements of V represent the values before the transition whereas those of V' represent the values after the transition.

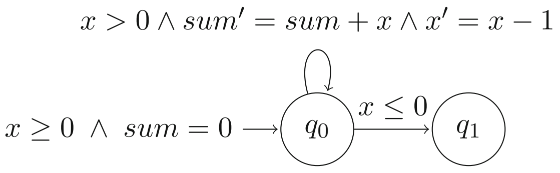
$x > 0 \wedge sum' = sum + x \wedge x' = x - 1$
$x \geq 0 \wedge sum = 0$
q_0
$x \leq 0$
q_1

Fig. 1. An example of DTSTS

Example 3.2. Figure 1 is an example of a DTSTS that models a program to compute the value of $1 + \cdots + x$; $Q := \{q_0, q_1\}$ and $\varphi_0 := x \geq 0 \wedge \mathit{sum} = 0$. In the transition from q_0 to q_0, the guard is $x > 0$; the command is $\mathit{sum}' = \mathit{sum} + x \wedge x' = x - 1$. In the transition from q_0 to q_1, the guard is $x \leq 0$; the command is $x' = x \wedge \mathit{sum}' = \mathit{sum}$ because this transition does not change the value of x and sum. Therefore, the transition relation $\delta = \{\langle q_0, x > 0, \mathit{sum}' = \mathit{sum} + x \wedge x' = x - 1, q_0\rangle, \langle q_0, x \leq 0, x' = x \wedge \mathit{sum}' = \mathit{sum}, q_1\rangle\}$. The finite sequence $\langle q_0, \{x \mapsto 3, \mathit{sum} \mapsto 0\}\rangle, \langle q_0, \{x \mapsto 2, \mathit{sum} \mapsto 3\}\rangle, \langle q_0, \{x \mapsto 1, \mathit{sum} \mapsto 5\}\rangle, \langle q_0, \{x \mapsto 0, \mathit{sum} \mapsto 6\}\rangle, \langle q_1, \{x \mapsto 0, \mathit{sum} \mapsto 6\}\rangle$ is a run of the DTSTS Fig. 1.

3.2 Hybrid Automaton (HA)

We model a hybrid system by a hybrid automaton (HA) [3]. We define an HA as an extension of DTSTS as follows.

Definition 3.3. *A* hybrid automaton (HA) *is a tuple $\langle Q, q_0, \varphi_0, F, \mathit{inv}, \delta\rangle$. The components Q, q_0, φ_0, and δ are the same as Definition 3.1. We use metavariable $\mathcal{S}_H$ for HA. F is a map from Q to ODE on V that specifies the flow dynamics at each location; inv is a map from Q to* **Fml** *that specifies the* stay condition[1] *at each state.*

A state of a hybrid automaton is a tuple $\langle q, \sigma\rangle$. A run of $\langle Q, q_0, \varphi_0, F, \mathit{inv}, \delta\rangle$ is a sequence of states $\langle q_0, \sigma_0\rangle \langle q_1, \sigma_1\rangle \ldots \langle q_n, \sigma_n\rangle$ where $\sigma_0 \models \varphi_0$. The system is allowed to make a transition from $\langle q_i, \sigma_i\rangle$ to $\langle q_{i+1}, \sigma_{i+1}\rangle$ if (1) σ_i reaches a valuation σ' along with the flow dynamics specified by $F(q_i)$, (2) $\mathit{inv}(q_i)$ holds at every point on the flow, and (3) $\langle q_i, \sigma'\rangle$ can jump to $\langle q_{i+1}, \sigma_{i+1}\rangle$ under the transition relation δ. In order to define the set of runs formally, we need to define the continuous-time dynamics that happens within each location.

Definition 3.4. *Let $\mathcal{D}$ be an ordinary differential equation (ODE) on V and let $x_1(t), \ldots, x_n(t)$ be a solution of $\mathcal{D}$ where t is the time. Let us write $\sigma^{(t)}$ for the valuation $\{x_1 \mapsto x_1(t), \ldots, x_n \mapsto x_n(t)\}$. We write $\sigma \rightarrow_{\mathcal{D},\varphi} \sigma'$ if (1) $\sigma = \sigma^{(0)}$ and (2) there exists $t' \geq t$ such that $\sigma' = \sigma^{(t')}$ and $\sigma^{(t'')} \models \varphi$ for any $t'' \in (0, t']$. We call relation $\rightarrow_{\mathcal{D},\varphi}$ the* flow transition.

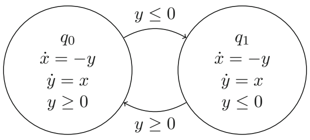

Fig. 2. An example of a hybrid automaton.

Intuitively, the relation $\sigma \rightarrow_{\mathcal{D},\varphi} \sigma'$ means that there is a trajectory from the state represented by σ to that represented by σ' such that (1) the trajectory is a solution of $\mathcal{D}$ and (2) φ holds at any point on the trajectory. For example, let $\mathcal{D}$ be $\dot{x} = v, \dot{v} = 1$, where x and v are time-dependent variables; $\dot{x}$ and $\dot{v}$ are their time derivative. The solution of $\mathcal{D}$ is $v = t + v_0$ and

[1] We use the word "stay condition" instead of the standard terminology "invariant" following Kapur et al. [23].

$x = \frac{t^2}{2} + v_0 t + x_0$ where t is the elapsed time, x_0 is the initial value of x, and v_0 is the initial value of v. Therefore, $\{x \mapsto 0, v \mapsto 0\} \rightarrow_{\mathcal{D},true} \{x \mapsto \frac{1}{2}, v \mapsto 1\}$ holds because $(x, v) = (\frac{1}{2}, 1)$ is the state at $t = 1$ on the above solution with $x_0 = 0$ and $v_0 = 0$. $\{x \mapsto 0, v \mapsto 0\} \rightarrow_{\mathcal{D},x\geq 0} \{x \mapsto \frac{1}{2}, v \mapsto 1\}$ also holds because the condition $x \geq 0$ continues to hold along with the trajectory from $(x, v) = (0, 0)$ to $(\frac{1}{2}, 1)$. However, $\{x \mapsto 0, v \mapsto 0\} \rightarrow_{\mathcal{D},x\geq \frac{1}{4}} \{x \mapsto \frac{1}{2}, v \mapsto 1\}$ does *not* hold because the condition $x \geq \frac{1}{4}$ does not hold for the initial $\frac{1}{\sqrt{2}}$ seconds in this trajectory.

Using this relation, we can define a run of an HA as follows.

Definition 3.5. *A finite sequence $\langle q^0, \sigma_0 \rangle, \langle q^1, \sigma_1 \rangle, \ldots, \langle q^N, \sigma_N \rangle$ is called a* run *of an HA $\langle Q, q_0, \varphi_0, F, inv, \delta \rangle$ if (1) $q^0 = q_0$, (2) $\sigma_0 \models \varphi_0$, (3) for any i, if $0 \leq i \leq N-2$, there exists $\langle q^i, \varphi_i, \varphi_c^i, q^{i+1} \rangle \in \delta$ and σ^I such that $\sigma_i \rightarrow_{F(q^i), inv(q^i)} \sigma^I$ and $\sigma^I \models \varphi_i$ and $\langle q^i, \sigma^I \rangle \rightarrow_\delta \langle q^{i+1}, \sigma_{i+1} \rangle$, and (4) $\sigma_{N-1} \rightarrow_{F(q^{N-1}), inv(q^{N-1})} \sigma_N$.*

Remark 3.6. This definition is more complicated than that of runs of DTSTS because we need to treat the last transition from $\langle q^{N-1}, \sigma_{N-1} \rangle$ to $\langle q^N, \sigma_N \rangle$ differently than the other transitions. Each transition from $\langle q^i, \sigma_i \rangle$ to $\langle q^{i+1}, \sigma_{i+1} \rangle$, if $0 \leq i \leq N-2$, is a flow transition followed by a jump transition; however, the last transition consists only of a flow transition.

Example 3.7. Figure 2 shows a hybrid automaton with $Q := \{q_0, q_1\}$ schematically. Each circle represents a location q; we write $F(q)$ for the ODE associated with each circle. Each edge between circles represents a transition; we present the guard of the transition on each edge. We omit the φ_c part; it is assumed to be the do-nothing command represented by $\wedge_{x \in V} x' = x$.

Both locations are equipped with the same flow that is the anticlockwise circle around the point $(x, y) = (0, 0)$ on the xy plane. The system can stay at q_0 as long as $y \geq 0$ and at q_1 as long as $y \leq 0$. $y = 0$ holds whenever a transition is invoked. Indeed, for example, $inv(q_0) = y \geq 0$ and the guard from q_0 to q_1 is $y \leq 0$; therefore, when the transition is invoked, $inv(q_0) \wedge y \leq 0$ holds, which is equivalent to $y = 0$.

Starting from the valuation $\sigma_0 := \{x \mapsto 1, y \mapsto 0\}$ at location q_0, the system reaches $\sigma_1 := \{x \mapsto -1, y \mapsto 0\}$ by the flow $F(q_0)$ along which $inv(q_0) \equiv y \geq 0$ continues to hold; then the transition from q_0 to q_1 is invoked. After that, the system reaches $\sigma_2 := \{x \mapsto 0, y \mapsto -1\}$ by $F(q_1)$. Therefore, $\langle q_0, \sigma_0 \rangle \langle q_1, \sigma_1 \rangle \langle q_1, \sigma_2 \rangle$ is a run of this HA.

3.3 Safety Verification Problem

Definition 3.8. *We say that σ is* reachable *in DTSTS $\mathcal{S}_D$ (resp., HA $\mathcal{S}_H$) if there is a run of $\mathcal{S}_D$ (resp., $\mathcal{S}_H$) that reaches $\langle q, \sigma \rangle$ for some q. A* safety verification problem (SVP) *for a DTSTS $\langle \mathcal{S}_D, \varphi \rangle$ (resp., HA $\langle \mathcal{S}_H, \varphi \rangle$) is the problem to decide whether $\sigma' \models \varphi$ holds for all the reachable valuation σ' of the given $\mathcal{S}_D$ (resp., $\mathcal{S}_H$).*

If an SVP is affirmatively solved, then the system is said to be *safe*; otherwise, the system is said to be *unsafe*. One of the major strategies for proving the safety of a system is discovering its *inductive invariant*.

Definition 3.9. – *Let $\langle \mathcal{S}_D, \varphi_P \rangle$ be an SVP for DTSTS where $\mathcal{S}_D = \langle Q, q_0, \varphi_0, \delta \rangle$. Then, a function $R : Q \to \mathbf{Fml}$ is called an inductive invariant if (1) $\models \varphi_0 \implies R(q_0)$; (2) if $\sigma \models R(q)$ and $\langle q, \sigma \rangle \to_\delta \langle q', \sigma' \rangle$, then $\sigma' \models R(q')$; and (3) $\models R(q) \implies \varphi_P$ for any q.*

– *Let $\langle \mathcal{S}_H, \varphi_P \rangle$ be an SVP for HA where $\mathcal{S}_H = \langle Q, q_0, \varphi_0, F, inv, \delta \rangle$. Then, a function $R : Q \to \mathbf{Fml}$ is called an inductive invariant if (1) $\models \varphi_0 \implies R(q_0)$; (2) if $\sigma \models R(q)$ and $\langle q, \sigma \rangle \to_{F(q), inv(q)} \langle q'', \sigma'' \rangle$ and $\langle q'', \sigma'' \rangle \to_\delta \langle q', \sigma' \rangle$, then $\sigma' \models R(q')$; and (3) $\models R(q) \implies \varphi_P$ for any q.*

Unsafety can be proved by discovering a *counterexample*.

Definition 3.10. *Define $\mathcal{S}_D$, φ_P, and $\mathcal{S}_H$ as in Definition 3.9. A run $\langle \sigma_0, q_0 \rangle \dots \langle \sigma_N, q_N \rangle$ of $\mathcal{S}_D$ (resp. $\mathcal{S}_H$) is called a counterexample to the SVP $\langle \mathcal{S}_D, \varphi_P \rangle$ (resp. $\langle \mathcal{S}_H, \varphi_P \rangle$) if $\sigma_N \models \neg\varphi_P$.*

GPDR is a procedure that tries to find an inductive invariant or a counterexample to a given SVP. SVP is in general undecidable. Therefore, the original GPDR approach [20] and our extension with hybrid systems presented in Sect. 5 do not terminate for every input.

4 GPDR for DTSTS

Before presenting our extension of GPDR with hybrid systems, we present the original GPDR procedure by Hoder and Bjørner [20] in this section. (The GPDR presented here, however, is slightly modified from the original one; see Remark 4.4.)

Given a safety verification problem $\langle \mathcal{S}_D, \varphi_P \rangle$ where $\mathcal{S}_D = \langle Q, q_0, \varphi_0, \delta \rangle$, GPDR tries to find (1) an inductive invariant to prove the safety of $\mathcal{S}_D$, or (2) a counterexample to refute the safety. To this end, GPDR (nondeterministically) manipulates a data structure called *configurations*. A configuration is either **Valid**, **Model** M, or an expression of the form $M \mid\mid R_0, \dots, R_N; N$. We explain each component of the expression $M \mid\mid R_0, \dots, R_N; N$ in the following. (**Valid** and **Model** M are explained later.)

- $R_0, \dots, R_N$ is a finite sequence of maps from Q to $\mathbf{Fml}$ (i.e., elements of $\mathbf{Fml}$). Each R_i is called a *frame*. The frames are updated during an execution of GPDR so that $R_i(q_j)$ is an overapproximation of the states that are reachable within i steps from the initial state in $\mathcal{S}_D$ and whose location is q_j.
- N is the index of the last frame.
- M is a finite sequence of the form $\langle \sigma_i, q_i, i \rangle, \langle \sigma_i, q_i, i+1 \rangle, \dots, \langle \sigma_N, q_N, N \rangle$. This sequence is a candidate partial counterexample that starts from the one that is i-step reachable from the initial state and that ends up with a state $\langle \sigma_N, q_N \rangle$ such that $\sigma_N \models \neg\varphi_P$. Therefore, in order to prove the safety of $\mathcal{S}_D$, a GPDR procedure needs to prove that $\langle q_i, \sigma_i \rangle$ is unreachable within i steps from an initial state.

In order to formalize the above intuition, GPDR uses a *forward predicate transformer* determined by $\mathcal{S}_D$. In the following, we fix an SVP $\langle \mathcal{S}_D, \varphi_P \rangle$.

Definition 4.1. *$\mathcal{F}(R)(q')$, where $\mathcal{F}$ is called the* forward predicate transformer *determined by $\mathcal{S}_D$, is the following formula:*

$$(q' = q_0 \land \varphi_0) \lor \bigvee_{(q,\varphi,\varphi_c,q') \in \delta} \exists \boldsymbol{x''}. \begin{pmatrix} [\boldsymbol{x''}/\boldsymbol{x}]R(q) \\ \land [\boldsymbol{x''}/\boldsymbol{x}]\varphi \land [\boldsymbol{x}/\boldsymbol{x'}, \boldsymbol{x''}/\boldsymbol{x}]\varphi_c \end{pmatrix},$$

where $\boldsymbol{x''}$ is the sequence $x''_1, \dots, x''_N$.

Notice that $\mathcal{F}(\lambda q.\mathit{false})$ is equivalent to φ_0. Intuitively, $\sigma' \models \mathcal{F}(R)(q')$ holds if $\langle q', \sigma' \rangle$ is an initial state (i.e., $q' = q_0$ and $\sigma' \models \varphi_0$) or $\langle q', \sigma' \rangle$ is reachable in 1-step transition from a state that satisfies R. The latter case is encoded by the second disjunct of the above definition: The valuation σ' satisfies the second disjunct if there are q, φ, and φ_c such that $(q, \varphi, \varphi_c, q') \in \delta$ (i.e., q' is 1-step after q in δ) and there is a valuation σ such that $\sigma \models R(q) \land \varphi$ (i.e., σ satisfies the precondition $R(q)$ and the guard φ) and σ' is a result of executing command c under σ.

The following lemma guarantees that $\mathcal{F}$ soundly approximates the transition of an DTSTS.

Lemma 4.2. *If $\sigma_1 \models R(q_1)$ and $\langle q_1, \sigma_1 \rangle \to_\delta \langle q_2, \sigma_2 \rangle$, then $\sigma_2 \models \mathcal{F}(R)(q_2)$.*

Proof. Assume $\sigma_1 \models R(q_1)$ and $\langle q_1, \sigma_1 \rangle \to_\delta \langle q_2, \sigma_2 \rangle$. Then, by definition, $(q_1, \varphi, \varphi', q_2) \in \delta$ and $\sigma_1 \models \varphi$ and $\sigma_1 \cup \sigma'_2 \models \varphi_c$ for some φ and φ_c. $\sigma''_1 \cup \sigma_2 \models [\boldsymbol{x''}/\boldsymbol{x}]R(q_1)$ follows from $\sigma_1 \models R(q_1)$. $\sigma''_1 \cup \sigma_2 \models [\boldsymbol{x''}/\boldsymbol{x}]\varphi$ follows from $\sigma_1 \models \varphi$. $\sigma''_1 \cup \sigma_2 \models [\boldsymbol{x}/\boldsymbol{x'}, \boldsymbol{x''}/\boldsymbol{x}]\varphi_c$ follows from $\sigma_1 \cup \sigma'_2 \models \varphi_c$. Therefore, $\sigma''_1 \cup \sigma_2 \models [\boldsymbol{x''}/\boldsymbol{x}]R(q_1) \land [\boldsymbol{x''}/\boldsymbol{x}]\varphi \land [\boldsymbol{x}/\boldsymbol{x'}, \boldsymbol{x''}/\boldsymbol{x}]\varphi_c$. Hence, we have $\sigma_2 \models \exists \boldsymbol{x''}.[\boldsymbol{x''}/\boldsymbol{x}]R(q) \land [\boldsymbol{x''}/\boldsymbol{x}]\varphi \land [\boldsymbol{x}/\boldsymbol{x'}, \boldsymbol{x''}/\boldsymbol{x}]\varphi_c$ as required.

By using the forward predicate transformer $\mathcal{F}$, we can formalize the intuition about configuration $M \mid\mid R_0, \dots, R_N; N$ explained so far as follows.

Definition 4.3. *Let $\mathcal{S}_D$ be $\langle Q, q_0, \varphi_0, \delta \rangle$, $\mathcal{F}$ be the forward predicate transformer determined by $\mathcal{S}_D$, and φ_P be the safety condition to be verified. A configuration C is said to be* consistent *if it is (1) of the form* **Valid**, *(2) of the form* **Model** *$\langle \sigma, q_0, 0 \rangle M$, or (3) of the form $M \mid\mid R_0, \dots, R_N; N$ that satisfies all of the following conditions:*

- *(Con-A) $R_0(q_0) = \varphi_0$ and $R_0(q_i) = \mathit{false}$ if $q_i \neq q_0$;*
- *(Con-B) $\models R_i(q) \implies R_{i+1}(q)$ for any q;*
- *(Con-C) $\models R_i(q) \implies \varphi_P$ for any q and $i < N$;*
- *(Con-D) $\models \mathcal{F}(R_i)(q) \implies R_{i+1}(q)$ for any $i < N$ and q;*
- *(Con-E) if $\langle \sigma, q, N \rangle \in M$, then $\sigma \models R_N(q) \land \neg\varphi_P$[2]; and*
- *(Con-F) if $\langle \sigma_1, q_1, i \rangle, \langle \sigma_2, q_2, i+1 \rangle \in M$ and $i < N$, then $\langle q_1, \varphi, \varphi_c, q_2 \rangle \in \delta$ and $\sigma_1, \sigma'_2 \models R_i(q_1) \land \varphi \land \varphi_c$.*

If C is consistent, we write **Con**(C).

INITIALIZE		$\rightsquigarrow \emptyset \mid\mid \langle R_0 := \mathcal{F}(\lambda q.\mathit{false}); N := 0\rangle$
		if $\forall q \in Q.\ \models R_0(q) \implies \varphi_P$
VALID	$M \mid\mid A$	$\rightsquigarrow$ **Valid**
		if $\exists i < N. \forall q \in Q.\ \models R_i(q) \implies R_{i-1}(q)$
UNFOLD	$M \mid\mid A$	$\rightsquigarrow \emptyset \mid\mid A[R_{N+1} := \lambda q.\mathit{true}; N := N+1]$
		if $\forall q \in Q.\ \models R_N(q) \implies \varphi_P$
INDUCTION	$M \mid\mid A$	$\rightsquigarrow \emptyset \mid\mid A[R_j := \lambda q.R_j(q) \wedge R(q)]_{j=1}^{i+1}$
		if $\forall q \in Q.\ \models \mathcal{F}(\lambda q.R_i(q) \wedge R(q))(q) \implies R(q)$
CANDIDATE	$\emptyset \mid\mid A$	$\rightsquigarrow \langle \sigma, q, N\rangle \mid\mid A$
		if $\sigma \models R_N(q) \wedge \neg\varphi_P$
DECIDE	$\langle \sigma_2, q_2, i+1\rangle\, M \mid\mid A$	$\rightsquigarrow \langle \sigma_1, q_1, i\rangle \langle \sigma_2, q_2, i+1\rangle\, M \mid\mid A$
		if $\langle q_1, \varphi, \varphi_c, q_2\rangle \in \delta$ **and** $\sigma_1, \sigma_2' \models R_i(q_1) \wedge \varphi \wedge \varphi_c$
MODEL	$\langle \sigma, q_0, 0\rangle\, M \mid\mid A$	$\rightsquigarrow$ **Model** $\langle \sigma, q_0, 0\rangle\, M$
CONFLICT	$\langle \sigma', q', i+1\rangle\, M \mid\mid A$	$\rightsquigarrow \emptyset \mid\mid A[R_j \leftarrow \lambda q.R_j(q) \wedge R(q)]_{j=1}^{i+1}$
		if $\models R(q') \implies \neg\sigma'$ **and** $\forall q \in Q.\ \models \mathcal{F}(R_i)(q) \implies R(q)$

Fig. 3. The rules for the original PDR. Recall that $\neg\sigma'$ in the rule CONFLICT denotes the formula $\neg\left(\bigwedge_{x \in V} x = \sigma'(x)\right)$.

The GPDR procedure rewrites a configuration following the (nondeterministic) rewriting rules in Fig. 3. We add a brief explanation below; for more detailed exposition, see [20]. Although the order of the applications of the rules in Fig. 3 is arbitrary, we fix one scenario of the rule applications in the following for explanation.

1. The procedure initializes M to $\emptyset$, R_0 to $\mathcal{F}(\lambda q.\mathit{false})$, and N to 0 (INITIALIZE).
2. If there are a valuation σ and a location q such that $\sigma \models R_N(q) \wedge \neg\varphi_P$ (CANDIDATE), then the procedure adds $\langle \sigma, q, N\rangle$ to M. The condition $\sigma \models R_N(q) \wedge \neg\varphi_P$ guarantees that the state $\langle q, \sigma\rangle$ violates the safety condition φ_P; therefore, the candidate $\langle \sigma, q, N\rangle$ needs to be refuted. If not, then the frame sequence is extended by setting N to $N+1$ and R_{N+1} to $\lambda q.\mathit{true}$ (UNFOLD); this is allowed since $\forall q \in Q.\ \models R_N(q) \implies \varphi_P$ in this case.
3. The discovered $\langle q, \sigma\rangle$ is backpropagated by successive applications of DECIDE: In each application of DECIDE, for $\langle q_2, \sigma_2, i+1\rangle$ in M, the procedure tries to find σ and q such that $\langle q_1, \varphi, \varphi_c, q_2\rangle \in \delta$ and $\sigma_1, \sigma_2' \models R_i(q_1) \wedge \varphi \wedge \varphi_c$ where σ_2' is the valuation obtained by replacing the domain of σ_2 with their primed counterpart. These conditions in combination guarantee $\langle q_1, \sigma_1\rangle \rightarrow_\delta \langle q_2, \sigma_2\rangle$ and $\sigma_1 \models R_i(q_1)$.
   (a) If this backpropagation reaches R_0 (the rule MODEL), then it reports the trace of the backpropagation returning **Model** M.
   (b) If it does not reach R_0, in which case there exists i such that $\sigma' \wedge \mathcal{F}(R_i)(q')$ is not satisfiable, then we pick a frame R such that $\models R(q') \implies \neg\sigma'$ and $\models \mathcal{F}(R_i)(q) \implies R(q)$ for any q (the rule CONFLICT). Intuitively, R is a frame that separates (1) the union of the initial states denoted by φ_0 and the states that are one-step reachable from a state denoted by $R_i(q')$ and

[2] We hereafter write $\langle \sigma, q, i\rangle \in M$ to express that the element $\langle \sigma, q, i\rangle$ exists in the sequence M although M is a sequence, not a set.

(2) the state denoted by $\langle q', \sigma' \rangle$. In a GPDR term, R is a *generalization* of $\neg\sigma'$. This formula is used to strengthen R_j for $j \in \{1, \ldots, i+1\}$.

4. The frame R obtained in the application of the rule Conflict is propagated forward by applying the rule Induction. The condition $\forall q \in Q. \models \mathcal{F}(\lambda q. R_i(q) \wedge R(q))(q) \implies R(q)$ forces that R holds in the one-step transition from a states that satisfies R_i. If this condition holds, then R holds for $i+1$ steps (Theorem 4.5); therefore, we conjoin R to $R_1(q), \ldots, R_{i+1}(q)$. In order to maintain the consistency conditions (Con-E) and (Con-F), this rule clears M to the empty set to keep its consistency to the updated frames.[3]
5. If $\forall q \in Q. \models R_i(q) \implies R_{i-1}(q)$ for some $i < N$, then the verification succeeds and R_i is an inductive invariant (Valid). If such i does not exist, then we go back to Step 2.

Remark 4.4. One of the differences of the above GPDR from the original one [20] is that ours deals with the locations of a given DTSTS explicitly. In the original GPDR, information about locations are assumed to be encoded using a variable that represents the program counter. Although such extension was proposed for IC3 by Lange et al. [26], we are not aware of a variant of GPDR that treats locations explicitly.

Soundness. We fix one DTSTS $\langle Q, q_0, \varphi_0, \delta \rangle$ in this section. The correctness of the GPDR procedure relies on the following lemmas.

Lemma 4.5. Con *is invariant to any rule application of Fig. 3.*

Theorem 4.6. *If the GPDR procedure is started from the rule* Initialize *and leads to* **Valid**, *then the system is safe. If the GPDR procedure is started from the rule* Initialize *and leads to* **Model** $\langle \sigma_0, q_0, 0 \rangle \ldots \langle \sigma_N, q_N, N \rangle$, *then the system is unsafe.*

5 HGPDR

We now present our procedure HGPDR that is an adaptation of the original GPDR to hybrid systems. An adaptation of GPDR to hybrid systems requires the following two challenges to be addressed.

1. The original definition of $\mathcal{F}$ (Definition 4.1) captures only a discrete-time transition. In our extension of GPDR, we need a forward predicate transformer that can mention a flow transition.
2. A run of an HA (Definition 3.5) differs from that of DTSTS in that its last transition consists only of flow dynamics; see Remark 3.6.

In order to address the first challenge, we extend the logic on which $\mathcal{F}$ is defined to be able to mention flow dynamics and define $\mathcal{F}$ on the extended logic (Sect. 5.1). To address the second challenge, we extend the configuration used by GPDR so that it carries an overapproximation of the states that are reachable from the last frame by a flow transition; the GPDR procedure is also extended to maintain this information correctly (Sect. 5.2).

[3] We could filter M so that it is consistent for the updated frame. We instead discard M here for simplicity.

5.1 Extension of Forward Predicate Transformer

In order to extend $\mathcal{F}$ to accommodate flow dynamics, we extend the logic on which $\mathcal{F}$ is defined with *continuous reachability predicates (CRP)* inspired by the differential dynamic logic ($d\mathcal{L}$) proposed by Platzer [33].

Definition 5.1. *Let $\mathcal{D}$ be an ODE over $Y := \{y_1, \dots, y_k\} \subseteq V$. Let us write σ for $\{y_1 \mapsto e_1, \dots, y_k \mapsto e_k\}$ and σ' for $\{y_1 \mapsto e'_1, \dots, y_k \mapsto e'_k\}$. We define a predicate $\langle \mathcal{D} \mid \varphi \rangle \varphi'$ by: $\sigma \models \langle \mathcal{D} \mid \varphi \rangle \varphi'$ iff. $\exists \sigma'. \sigma \rightarrow_{\mathcal{D},\varphi} \sigma' \wedge \sigma' \models \varphi'$. We call a predicate of the form $\langle \mathcal{D} \mid \varphi_I \rangle \varphi$ a* continuous reachability predicate (CRP)*.*

Using the above predicate, we extend $\mathcal{F}$ as follows.

Definition 5.2. *For an HA $\langle Q, q_0, \varphi_0, F, inv, \delta \rangle$, the forward predicate transformer $\mathcal{F}_{\mathcal{H}}(R)(q')$ is the following formula:*

$$(q' = q_0 \wedge \varphi_0) \vee \bigvee_{(q,\varphi,\varphi_c,q') \in \delta} \exists \boldsymbol{x''}. \begin{pmatrix} [\boldsymbol{x''}/\boldsymbol{x}]R(q) \\ \wedge \langle [\boldsymbol{x''}/\boldsymbol{x}]F(q) \mid [\boldsymbol{x''}/\boldsymbol{x}]inv(q) \rangle ([\boldsymbol{x''}/\boldsymbol{x}]\varphi \wedge [\boldsymbol{x}/\boldsymbol{x'}, \boldsymbol{x''}/\boldsymbol{x}]\varphi_c) \end{pmatrix}.$$

In the above definition, $[\boldsymbol{x''}/\boldsymbol{x}]F(q)$ is the ODE obtained by renaming the variables $\boldsymbol{x}$ that occur in ODE $F(q)$ with $\boldsymbol{x''}$.

We also define predicate $\mathcal{F}_{\mathcal{C}}(R)(q')$ as follows:

$$\exists \boldsymbol{x''}.([\boldsymbol{x''}/\boldsymbol{x}]R(q') \wedge \langle [\boldsymbol{x''}/\boldsymbol{x}]F(q') \mid [\boldsymbol{x''}/\boldsymbol{x}]inv(q') \rangle \boldsymbol{x} = \boldsymbol{x''}).$$

Intuitively, $\sigma' \models \mathcal{F}_{\mathcal{H}}(\varphi)(q')$ holds if either (1) $\langle q', \sigma' \rangle$ is an initial state or (2) it is reachable from R by a flow transition followed a jump transition. Similarly, $\sigma' \models \mathcal{F}_{\mathcal{C}}(R)(q')$ holds if σ' is reachable in a flow transition (not followed by a jump transition) from a state denoted by $R(q')$. This definition of $\mathcal{F}_{\mathcal{H}}$ is an extension of Definition 4.1 in that it encodes the "flow-transition" part of the above intuition by the CRP. In the case of $\mathcal{F}_{\mathcal{C}}$, the postcondition part of the CRP is $\boldsymbol{x} = \boldsymbol{x''}$ because we do not need a jump transition in this case.

Lemma 5.3. *If $\sigma_1 \models R(q_1)$ and $\sigma_1 \rightarrow_{F(q_1),inv(q_1)} \sigma^I$ and $\langle q_1, \sigma^I \rangle \rightarrow_\delta \langle q_2, \sigma_2 \rangle$, then $\sigma_2 \models \mathcal{F}_{\mathcal{H}}(R)(q_2)$.*

Proof. Assume (1) $\sigma_1 \models R(q_1)$, (2) $\sigma_1 \rightarrow_{F(q_1),inv(q_1)} \sigma^I$, and (3) $\langle q_1, \sigma^I \rangle \rightarrow_\delta \langle q_2, \sigma_2 \rangle$. Then, by definition, (4) $(q_1, \varphi, \varphi_c, q_2) \in \delta$ and (5) $\sigma^I \models \varphi$ and (6) $\sigma^I \cup \sigma'_2 \models \varphi_c$ for some φ and φ_c. We show $\exists \boldsymbol{x''}. ([\boldsymbol{x''}/\boldsymbol{x}]R(q) \wedge \langle [\boldsymbol{x''}/\boldsymbol{x}]F(q) \mid [\boldsymbol{x''}/\boldsymbol{x}]inv(q) \rangle ([\boldsymbol{x''}/\boldsymbol{x}]\varphi \wedge [\boldsymbol{x}/\boldsymbol{x'}, \boldsymbol{x''}/\boldsymbol{x}]\varphi_c))$. (5) implies (7) $\sigma^I \cup \sigma'_2 \models \varphi$. (6) and (7) imply (8) $\sigma^{I''} \cup \sigma'_2 \models [\boldsymbol{x''}/\boldsymbol{x}]\varphi \wedge [\boldsymbol{x''}/\boldsymbol{x}]\varphi_c$. (2) implies (9) $\sigma''_1 \rightarrow_{[\boldsymbol{x''}/\boldsymbol{x}]F(q_1),[\boldsymbol{x''}/\boldsymbol{x}]inv(q_1)} \sigma^{I''}$. Therefore, from (8) and (9), we have (10) $\sigma''_1 \cup \sigma_2 \models \langle [\boldsymbol{x''}/\boldsymbol{x}]F(q_1) \mid [\boldsymbol{x''}/\boldsymbol{x}]inv(q_1) \rangle ([\boldsymbol{x''}/\boldsymbol{x}]\varphi \wedge [\boldsymbol{x''}/\boldsymbol{x}, \boldsymbol{x}/\boldsymbol{x'}]\varphi_c)$. (Note that the variables in $\boldsymbol{x'}$ appear only in φ_c.) $\sigma''_1 \cup \sigma_2 \models [\boldsymbol{x''}/\boldsymbol{x}]R(q_1)$ follows from (1); therefore, we have $\sigma''_1 \cup \sigma_2 \models [\boldsymbol{x''}/\boldsymbol{x}]R(q_1) \wedge \langle [\boldsymbol{x''}/\boldsymbol{x}]F(q_1) \mid [\boldsymbol{x''}/\boldsymbol{x}]inv(q_1) \rangle ([\boldsymbol{x''}/\boldsymbol{x}]\varphi \wedge [\boldsymbol{x''}/\boldsymbol{x}, \boldsymbol{x}/\boldsymbol{x'}]\varphi_c)$. This implies $\exists \boldsymbol{x''}. ([\boldsymbol{x''}/\boldsymbol{x}]R(q) \wedge \langle [\boldsymbol{x''}/\boldsymbol{x}]F(q) \mid [\boldsymbol{x''}/\boldsymbol{x}]inv(q) \rangle ([\boldsymbol{x''}/\boldsymbol{x}]\varphi \wedge [\boldsymbol{x}/\boldsymbol{x'}, \boldsymbol{x''}/\boldsymbol{x}]\varphi_c))$ as required.

$$
\begin{array}{lrl}
\textsc{Initialize} & & \rightsquigarrow \emptyset \mid\mid \langle R_0 := \mathcal{F}_{\mathcal{H}}(\lambda q.false); R_{rem} := \lambda q.true; N := 0\rangle \\
 & & \quad \textbf{if}\, \forall q \in Q.\models R_0(q) \implies \varphi_P \\
\textsc{Valid} & M \mid\mid A & \rightsquigarrow \textbf{Valid} \\
 & & \quad \textbf{if}\, \exists i < N.\forall q \in Q.\models R_i(q) \implies R_{i-1}(q) \\
\textsc{Unfold} & M \mid\mid A & \rightsquigarrow \emptyset \mid\mid A[R_{N+1} := \lambda q.true; R_{rem} := \lambda q.true; N := N+1] \\
 & & \quad \textbf{if}\, \forall q \in Q.\models R_{rem}(q) \implies \varphi_P \\
\textsc{Induction} & M \mid\mid A & \rightsquigarrow \emptyset \mid\mid A[R_j := \lambda q.R_j(q) \wedge R(q)]_{j=1}^{i+1} \\
 & & \quad \textbf{if}\, \forall q \in Q.\models \mathcal{F}_{\mathcal{H}}(\lambda q.R_i(q) \wedge R(q))(q) \implies R(q) \\
\textsc{Decide} & \langle \sigma_2, q_2, i+1\rangle\, M \mid\mid A & \rightsquigarrow \langle \sigma_1, q_1, i\rangle\, \langle \sigma_2, q_2, i+1\rangle\, M \mid\mid A \\
 & & \quad \textbf{if}\, \langle q_1, \varphi, \varphi_c, q_2\rangle \in \delta\, \textbf{and}\, \sigma_1, \sigma_2' \models R_i(q_1) \wedge \langle F(q_1) \mid inv(q_1)\rangle(\varphi \wedge \varphi_c) \\
\textsc{Model} & \langle \sigma, q_0, 0\rangle\, M \mid\mid A & \rightsquigarrow \textbf{Model}\, \langle \sigma, q_0, 0\rangle\, M \\
\textsc{Conflict} & \langle \sigma', q', i+1\rangle\, M \mid\mid A & \rightsquigarrow \emptyset \mid\mid A[R_j := \lambda q.R_j(q) \wedge R(q)]_{j=1}^{i+1} \\
 & & \quad \textbf{if}\, \models R(q') \implies \neg\sigma'\, \textbf{and}\, \forall q \in Q.\models \mathcal{F}_{\mathcal{H}}(R_i)(q) \implies R(q) \\
\textsc{PropagateCont} & M \mid\mid A & \rightsquigarrow M \mid\mid A[R_{rem} := \lambda q.R_{rem}(q) \wedge R(q)] \\
 & & \quad \textbf{if}\, \forall q \in Q.\models R_N(q) \vee \mathcal{F}_{\mathcal{C}}(R_N)(q) \implies R(q) \\
\textsc{CandidateCont} & \emptyset \mid\mid A & \rightsquigarrow \langle \sigma, q, rem\rangle \mid\mid A \\
 & & \quad \textbf{if}\, \sigma \models R_{rem}(q) \wedge \neg\varphi_P \\
\textsc{DecideCont} & \langle \sigma_2, q, rem\rangle \mid\mid A & \rightsquigarrow \langle \sigma_1, q, N\rangle\, \langle \sigma_2, q, rem\rangle \mid\mid A \\
 & & \quad \textbf{if}\, \sigma_1, \sigma_2' \models R_N(q) \wedge \langle F(q) \mid inv(q)\rangle(\boldsymbol{x} = \boldsymbol{x'}) \\
\textsc{ConflictCont} & \langle \sigma', q', rem\rangle \mid\mid A & \rightsquigarrow \emptyset \mid\mid A[R_{rem} := \lambda q.R_{rem}(q) \wedge R(q)] \\
 & & \quad \textbf{if}\, R(q') \implies \neg\sigma', \textbf{and} \models R_N(q') \vee \mathcal{F}_{\mathcal{C}}(R_N)(q') \implies R(q')
\end{array}
$$

Fig. 4. The rules for HGPDR.

Lemma 5.4. *If* $\sigma_1 \models R(q_1)$ *and* $\sigma_1 \rightarrow_{F(q_1),inv(q_1)} \sigma_2$*, then* $\sigma_2 \models \mathcal{F}_{\mathcal{C}}(R)(q_1)$*.*

Proof. Almost the same argument as the proof of Lemma 5.3.

5.2 Extension of GPDR

We present our adaptation of GPDR for hybrid systems, which we call HGPDR. Recall that the original GPDR in Sect. 4 maintains a configuration of the form $M \mid\mid R_0, \ldots, R_N; N$. HGPDR uses a configuration of the form $M \mid\mid R_0, \ldots, R_N; R_{rem}; N$. In addition to the information in the original configurations, we add R_{rem} which we call *remainder frame*. R_{rem} overapproximates the states that are reachable from R_N within one flow transition.

Figure 4 presents the rules for HGPDR. The rules from Initialize to Conflict are the same as Fig. 3 except that (1) Initialize and Unfold are adapted so that they set the remainder frame to $\lambda q.true$ and (2) Candidate is dropped. We explain the newly added rules.

- PropagateCont discovers a fact that holds in R_{rem}. The side condition $\models R_N(q) \vee \mathcal{F}_{\mathcal{C}}(R_N)(q) \implies R(q)$ for any q guarantees that $R(q)$ is true at the remainder frame; hence R is conjoined to R_{rem}.
- CandidateCont replaces Candidate in the original procedure. It tries to find a candidate from the frame R_{rem}. The candidate $\langle q, \sigma\rangle$ found here is added to M in the form $\langle \sigma, q, rem\rangle$ to denote that $\langle q, \sigma\rangle$ is found at R_{rem}.
- DecideCont propagates a counterexample $\langle \sigma', q', rem\rangle$ found at R_{rem} to the previous frame R_N. This rule computes the candidate to be added to M by deciding $\sigma \cup \sigma' \models R_N(q) \wedge \langle F(q) \mid inv(q)\rangle(\boldsymbol{x} = \boldsymbol{x'})$, which guarantees that σ evolves to σ' under the flow dynamics determined by $F(q)$ and $inv(q)$.
- Conflict uses $\mathcal{F}_{\mathcal{H}}$ instead of $\mathcal{F}$ in the original GPDR. As in the rule Conflict in GPDR, the frame R in this rule is a generalization of $\neg\sigma'$ which is not backward reachable to R_i.

- ConflictCont is the counterpart of Conflict for the frame R_{rem}. This rule is the same as Conflict except that it uses $\mathcal{F}_{\mathcal{C}}$ instead of $\mathcal{F}_{\mathcal{H}}$; hence, R separates σ' from both the states denoted by φ_0 and the states that are reachable from R_i in a flow transition (*not* followed by a jump transition).

5.3 Soundness

In order to prove the soundness of HGPDR, we adapt the definition of **Con** in Definition 4.3 for HGPDR.

Definition 5.5. *Let $\mathcal{S}_H$ be $\langle Q, q_0, \varphi_0, F, inv, \delta\rangle$, $\mathcal{F}_{\mathcal{H}}$ and $\mathcal{F}_{\mathcal{C}}$ be the forward predicate transformers determined by $\mathcal{S}_H$, and φ_P be the safety condition to be verified. A configuration C is said to be* consistent *if it is* **Valid**, **Model** $\langle \sigma, q_0, 0\rangle\, M$, *or* $\mathbf{Con}_H(M \mid\mid R_0, \ldots, R_N; R_{rem}; N)$ *that satisfies all of the following:*

- *(Con-A)* $R_0(q_0) = \varphi_0$ *and* $R_0(q_i) = false$ *if* $q_i \neq q_0$*;*
- *(Con-B-1)* $\models R_i(q) \implies R_{i+1}(q)$ *for any* q *and* $i < N$*;*
- *(Con-B-2)* $\models R_N(q) \implies R_{rem}(q)$ *for any* q*;*
- *(Con-C)* $\models R_i(q) \implies \varphi_P$ *if* $i < N$*;*
- *(Con-D-1)* $\models \mathcal{F}_{\mathcal{H}}(R_i)(q) \implies R_{i+1}(q)$ *for any* $i < N$ *and* q*;*
- *(Con-D-2)* $\models \mathcal{F}_{\mathcal{C}}(R_N)(q) \implies R_{rem}(q)$ *for any* q*;*
- *(Con-E) if* $\langle \sigma, q, rem\rangle \in M$*, then* $\sigma \models R_{rem}(q) \wedge \neg\varphi_P$*;*
- *(Con-F-1) if* $\langle \sigma_1, q_1, i\rangle, \langle \sigma_2, q_2, i+1\rangle \in M$ *and* $i < N$*, then* $\langle q_1, \varphi, \varphi_c, q_2\rangle \in \delta$ *and* $\sigma_1, \sigma_2' \models R_i(q_1) \wedge \varphi \wedge \varphi_c$*; and*
- *(Con-F-2) if* $\langle \sigma_1, q_1, N\rangle, \langle \sigma_2, q_2, rem\rangle \in M$*, then* $\langle q_1, \varphi, \varphi_c, q_2\rangle \in \delta$ *and* $\sigma_1, \sigma_2' \models R_i(q_1) \wedge \varphi \wedge \varphi_c$*.*

The soundness proof follows the same strategy as that of the original GPDR.

Lemma 5.6. $\mathbf{Con}_H$ *is invariant to any rule application of Fig. 4.*

Theorem 5.7. *If* HGPDR *is started from the rule* Initialize *and leads to* Valid*, then the system is safe. If* HGPDR *is started from the rule* Initialize *and leads to* **Model** $\langle \sigma_0, q_0, 0\rangle \ldots \langle \sigma_N, q_N, N\rangle\, \langle \sigma_{rem}, q_{rem}, rem\rangle$*, then the system is unsafe.*

5.4 Operational Presentation of HGPDR

The definition of HGPDR in Fig. 4 is declarative and nondeterministic. For the sake of convenience of implementation, we derive an operational procedure from HGPDR; we call the operational version DetHybridPDR, whose definition is in Algorithm 1.

```
Input: Hybrid automaton $\mathcal{S}_H := \langle Q, q_0, \varphi_0, F, inv, \delta\rangle$
Output: Model($M$) if $\mathcal{S}_H$ is unsafe; $M$ is a witnessing trace. Valid($R$) if $\mathcal{S}_H$ is
        safe; $R$ is an inductive invariant.
// Initialize
1  $N := 0$; $R_0 := \lambda q.(\textbf{if}\ q = q_0\ \textbf{then}\ \varphi_0\ \textbf{else}\ \mathit{false})$
2  $R_1 := \mathit{true}$; $R_{rem} := \mathit{true}$; $M := \emptyset$
3  while $\mathit{true}$ do
4      for $q \in Q$ do
5          switch $\mathit{querySat}(R_{rem}(q) \wedge \neg\varphi_P)$ do
6              case $\mathit{Sat}(\sigma')$ do
                   // CandidateCont
7                  $M := \langle q, \sigma, rem\rangle$
8                  switch $\mathit{RemoveTrace}(M,\ R_0, \dots, R_N, R_{rem},\ N)$ do
9                      case Valid($R$) do
10                         return Valid($R$)
11                     case Cont($R_0, \dots, R_N, R_{rem}$) do
12                         $M := \emptyset$
13                         Update $R_0, \dots, R_N, R_{rem}$ to the returned frames
14                     case Model($M$) do
15                         return Model($M$)
16                 end
17             case $\mathit{Unsat}$ do
                   // Unfold
18                 $M := \emptyset$; $R_{N+1} := \lambda q.\mathit{true}$; $R_{rem} := \lambda q.\mathit{true}$; $N := N + 1$
19         end
20     end
21 end
```

Algorithm 1. Definition of DetHybridPDR.

Discharging Verification Conditions. An implementation of HGPDR needs to discharge verification conditions during verification. In addition to verification conditions expressed as a satisfiability problem of a first-order predicate, which can be discharged by a standard SMT solver, DetHybridPDR needs to discharge conditions including a CRP predicate. Specifically, DetHybridPDR needs to deal with the following three types of problems.

- Checking whether $\delta := \psi \wedge \langle \mathcal{D} \mid \varphi_I\rangle(\wedge_{x\in V} x = \sigma'(x))$ is satisfiable or not for given first-order predicates ψ and φ_I, an ODE $\mathcal{D}$, and a valuation σ'. DetHybridPDR needs to discharge this type of predicates when it decides which of DecideCont and ConflictCont should be applied if the top of M is $\langle \sigma', q', rem\rangle$. We use Algorithm 3 for discharging δ. This algorithm searches for a valuation σ_i that witnesses the satisfiability of δ by using a time-inverted simulation of $\mathcal{D}$ as follows. Concretely, this algorithm numerically simulates $\mathcal{D}^{-1}$, the time-inverted ODE of $\mathcal{D}$, starting from the point $\{\boldsymbol{x} \mapsto \sigma'(x)\}$. If it reaches a point σ_i that satisfies ψ and if all $\sigma_{i+1} \dots \sigma'$ in the obtained solution satisfy φ_I, then σ_i witnesses the satisfiability of δ. If such σ_i does

```
Input: Hybrid automaton $\mathcal{S}_H := \langle Q, q_0, \varphi_0, F, inv, \delta\rangle$; Trace of counterexamples $M$; Frames
       $R_0, \dots, R_N, R_{rem}$; Natural number $N$.
Output:
while $M \neq \emptyset$ do
    if $M = \langle q', \sigma', rem\rangle\, M'$ then
        switch $querySat_C(R_N(q') \wedge \langle F(q') \mid inv(q')\rangle(\boldsymbol{x} = \sigma'(\boldsymbol{x}))$ do
            // DecideCont
            case $Sat(\sigma)$ do
                $M := \langle q', \sigma, N\rangle\, M$
            // ConflictCont
            case $Unsat(R)$ do
                $M := \emptyset; R_{rem} := \lambda q.R_{rem}(q) \wedge R(q)$
                // PropagateCont
                for $\psi \in Formulas(R_N(q'))$ do
                    switch $querySat_C(R_N(q') \wedge \langle F(q') \mid inv(q')\rangle\neg\psi)$ do
                        case $Unsat$ do
                            $R_{rem}(q') := R_{rem}(q') \wedge \psi$
                    end
                end
        end
    else if $M = \langle q', \sigma', 0\rangle\, M'$ then
        // Model
        return Model($M$)
    else if $M = \langle q', \sigma', i\rangle\, M'$ and $0 < i \neq rem$ then
        for $\langle q, \varphi, \varphi_c, q'\rangle \in \delta$ do
            switch $querySat_C(R_{i-1}(q) \wedge \langle F(q) \mid inv(q)\rangle(\varphi \wedge \varphi_c \wedge \boldsymbol{x} = \sigma'(\boldsymbol{x})))$ do
                // Decide
                case $Sat(\sigma)$ do
                    $M := \langle q, \sigma\rangle\, M$
                // Conflict
                case $Unsat(R)$ do
                    for $j \in [1, i+1]$ do
                        $R_j := \lambda q.R_j(q) \wedge R(q); M := \emptyset;$
                    end
                // Induction
                for $i \in [1, N-1], \psi \in Formulas(R_i(q'))$ do
                    switch $querySat_C(R_i(q') \wedge \psi \wedge \langle F(q') \mid inv(q')\rangle\neg\psi)$ do
                        case $Unsat$ do
                            $R_j(q') := R_j(q') \wedge \psi$ for $j \in [1, i+1]$
                    end
                end
            end
        end
    end
end
if There exists $i$ such that $\forall q.\ \models R_{i+1}(q) \implies R_i(q)$ then
    // Valid
    return Valid($R_i$)
else
    // Inductive invariant is not reached yet.
    return Cont($R_0, \dots, R_N, R_{rem}$)
end
```

Algorithm 2. Definition of *RemoveTrace*.

not exist but there is σ_i such that $\sigma_i \not\models \varphi_I$, then ψ is not backward reachable from σ' and hence δ is unsatisfiable. In this case, Algorithm 3 needs to return a predicate that can be used as ψ' in the rule ConflictCont in Fig. 4. Currently, we assume that the user provides this predicate. We expect that we can help this step of discovering ψ' by using techniques for analyzing

```
Input: Formula $\delta := \psi \wedge \langle \mathcal{D} \mid \varphi_I \rangle (\wedge_{x \in V} x = \sigma'(x))$ to be discharged; Number $T > 0$.
Output: $Sat(\sigma_i)$ if $\sigma_i \models \delta$; $Unsat(\psi')$ if $\delta$ is unsatisfiable and $\psi'$ is a generalization of $\sigma'$;
        aborts if satisfiability nor unsatisfiability is proved.
// $\mathcal{D}^{-1}$ is the time-inverted ODE of $\mathcal{D}$. Therefore, $p$ is the backward solution of $\mathcal{D}$
   from $\sigma'$.
1  $\mathcal{D}^{-1}$ := the ODE obtained by replacing all the occurrences of the variable $t$ corresponding
     to the time to $-t$ and negating each time derivative;
2  Solve $\mathcal{D}^{-1}$ numerically from the initial point $\sigma_0 := \sigma'$;
3  Let $p := \sigma_0 \sigma_1 \dots \sigma_{T-1}$ be the solution obtained at the Step 2;
   // $i_1$ is set to $\infty$ if there is no such $i$.
4  $i_1$ := the minimum $i$ such that $\sigma_j \models \varphi_I$ for any $j < i$ and $\sigma_i \models \psi$;
   // $i_2$ is set to $\infty$ if there is no such $i$.
5  $i_2$ := the minimum $i$ such that $\sigma_j \models \varphi_I$ for any $j < i$;
6  if $i_1 < \infty$ then
       // $\sigma_{i_1}$ witnesses the satisfiability of $\delta$.
7      return $Sat(\sigma_{i_1})$
8  else if $i_2 < \infty$ then
       // $\sigma_{i_2}$ is the end point of the $\mathcal{D}^{-1}$ with the stay condition $\varphi_I$, but $\sigma_{i_2} \not\models \psi$.
          Therefore, $\psi$ is not backward reachable from $\sigma'$ along with $\mathcal{D}$. Currently, the
          user needs to provide a predicate that can be used for further refinement.
9      Obtain $\psi'$ such that $\models \exists \boldsymbol{x_0}.[\boldsymbol{x_0}/\boldsymbol{x}]\psi \wedge \langle [\boldsymbol{x_0}/\boldsymbol{x}]\mathcal{D} \mid [\boldsymbol{x_0}/\boldsymbol{x}]\varphi_I \rangle \boldsymbol{x_0} = \boldsymbol{x} \implies \psi'$ and
         $\sigma' \not\models \psi'$ from the user;
10     return $Unsat(\psi')$
11 end
   // Cannot conclude neither satisfiability nor unsatisfiabililty.
12 abort
```

Algorithm 3. Algorithm for discharging $\delta := \psi \wedge \langle \mathcal{D} \mid \varphi_I \rangle (\wedge_{x \in V} x = \sigma'(x))$.

continuous dynamics (e.g., automated synthesizer of barrier certificates [34] and Flow* [9] in combination with Craig interpolant synthesis procedures [2, 31]). If neither holds, then we give up the verification by aborting; this may happen if, for example, the value of T is too small.

– Checking whether $\delta' := \psi \wedge \langle F(q) \mid inv(q) \rangle (\varphi \wedge \varphi_c \wedge \boldsymbol{x} = \sigma'(\boldsymbol{x}))$ is satisfiable or not. DetHybridPDR needs to solve this problem in the choice between Decide and Conflict. This query is different from the previous case in that the formula that appears after $\langle F(q) \mid inv(q) \rangle$ in δ' is $\varphi \wedge \varphi_c \wedge \boldsymbol{x} = \sigma'(\boldsymbol{x})$, not $\boldsymbol{x} = \sigma'(\boldsymbol{x})$; therefore, we cannot use numerical simulation to discharge δ'. Although it is possible to adapt Algorithm 3 to maintain the sequence of *predicates* $\alpha_0 \alpha_1 \dots \alpha_{T-1}$ instead of *valuations* so that each α_i becomes the preimage of α_{i-1} by $\mathcal{D}$, the preimage computation at each step is prohibitively expensive. Instead, the current implementation restricts the input system so that there exists at most one σ such that $\sigma \models \varphi \wedge \varphi_c \wedge \boldsymbol{x} = \sigma'(\boldsymbol{x})$ for any σ'; if this is met, then one can safely use Algorithm 3 for discharging δ'. Concretely, we allow only φ_c that corresponds to the command whose syntax is given by $c ::= \textbf{skip} \mid x := r_1 x + r_2 \mid x := r_1 x - r_2$ where **skip** is a command that does nothing; r_1 and r_2 are real constants.

– Checking whether $\varphi_1 \wedge \langle \mathcal{D} \mid \varphi_I \rangle \neg \varphi_2$ is unsatisfiable. DetHybridPDR needs to discharge this type of queries when it applies Induction or PropagateCont. This case is different from the previous case in that (1) DetHybridPDR may answer *Otherwise* without aborting the entire verification if unsatisfiability nor satisfiability is proved, and (2) DetHybridPDR

```
Input: Formula $\varphi_1 \wedge \langle \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \mid \varphi_I \rangle \neg\varphi_2$ to be discharged; Number $r > 0$.
Output: Unsat or Otherwise; if Unsat is returned then the input formula is
        unsatisfiable.
1  if $\varphi_1 \wedge \varphi_2$ is satisfiable then
2      return Otherwise
3  end
4  Let $dt$ be a fresh symbol;
   // Checking $\varphi_1$ is invariant throughout the dynamics determined by
      $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ and $\models \varphi_1 \implies \neg\varphi_2$.
5  if $r > dt > 0 \wedge \varphi_1 \wedge \varphi_I \wedge \neg[\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x})dt/\boldsymbol{x}]\varphi_1$ and $\varphi_1 \wedge \varphi_2$ are unsatisfiable then
6      return Unsat
7  end
   // Checking $\neg\varphi_2$ is invariant throughout in the dynamics determined
      by $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ and $\models \varphi_1 \implies \neg\varphi_2$.
8  if $r > dt > 0 \wedge \neg\varphi_2 \wedge \varphi_I \wedge [\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x})dt/\boldsymbol{x}]\varphi_2$ and $\varphi_1 \wedge \varphi_2$ are unsatisfiable then
9      return Unsat
10 end
11 return Otherwise
```

Algorithm 4. Algorithm for discharging $\varphi_1 \wedge \langle \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \mid \varphi_I \rangle \neg\varphi_2$.

does not need to return a generalization if the given predicate is unsatisfiable. We use Algorithm 4 to discharge this type of queries. This algorithm first checks the satisfiability of $\varphi_1 \wedge \varphi_2$ in Step 1; if it is satisfiable, then so is the entire formula. Then, Step 5 tries to prove that the entire formula is unsatisfiable by proving (1) φ_1 is invariant with respect to the dynamics specified by $\mathcal{D}$ and φ_I and (2) $\varphi_1 \wedge \varphi_2$ is unsatisfiable. In order to prove the former, the algorithm tries the following sufficient condition: For any positive *dt* that is smaller than a positive real number r, $\models \varphi_i \wedge \varphi_I \implies [\boldsymbol{x} + \boldsymbol{f}(\boldsymbol{x})dt/\boldsymbol{x}]\varphi_1$, where $\mathcal{D} \equiv \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$.[4] Step 8 tries the same strategy but tries to prove that $\neg\varphi_2$ is invariant. If both attempts fail, then the algorithm returns *Otherwise*.[5] This algorithm could be further enhanced by incorporating automated invariant-synthesis procedures [15,28,35]; exploration of this possibilities is left as future work.

6 Proof-of-Concept Implementation

We implemented DetHybridPDR as a semi-automated verifier. We note that the current implementation is intended to be a proof of concept; extensive experiments are left as future work. The snapshot of the source code as of writing can be found at https://github.com/ksuenaga/HybridPDR/tree/master/src.

[4] This strategy is inspired by the previous work by one of the authors on nonstandard programming [18,30,36,37].

[5] If the flow specified by $\mathcal{D}$ is a linear or a polynomial, then we can apply the procedure proposed by Liu et al. [28], which is proved to be sound and complete for such a flow.

The verifier takes a hybrid automaton $\mathcal{S}_H$ specified with SpaceEx modeling language [27], the initial location q_0, the initial condition φ_0, and the safety condition φ_P as input; then, it applies DetHybridPDR to discover an inductive invariant or a counterexample. The frontend of the verifier is implemented with OCaml; in the backend, the verifier uses Z3 [29] and ODEPACK [1] to discharge verification conditions.

As we mentioned in Sect. 5.4, when a candidate counterexample $\langle q', \sigma', i+1\rangle$ turns out to be backward unreachable to R_i, then our verifier asks for a generalization of σ' to the user; concretely, for example in an application of the rule Conflict, the user is required to give ψ such that $\models \psi \implies \neg\sigma'$ **and** $\models (q, \varphi, \varphi_c, q') \in \delta \wedge [\boldsymbol{x_0}/\boldsymbol{x}]R_i(q) \wedge \langle [\boldsymbol{x_0}/\boldsymbol{x}]F(q) \mid [\boldsymbol{x_0}/\boldsymbol{x}]inv(q)\rangle[\boldsymbol{x_0}/\boldsymbol{x}, \boldsymbol{x_0}/\boldsymbol{x}](\varphi \wedge \varphi_c) \implies \psi$ **and** $\models R_0(q') \implies \psi$. Instead of throwing this query at the user in this form, the verifier asks the following question in order to make this process easier for the user for each $(q, \varphi, \varphi_c, q') \in \delta$:

`Pre:`$R_i(q)$`; Flow:`$F(q)$`; Stay:`$inv(q)$`; Guard:`φ`; Cmd:`φ_c`; CE:`σ'`; Init:`$R_0(q')$`.`

In applying ConflictCont, the verifier omits the fields `Guard` and `Cmd`.

We applied the verifier to the hybrid automaton in Fig. 2 with several initial conditions and the safety condition $\varphi_P := x \le 1$. We remark that the outputs from the verifier presented here are post-processed for readability. We explain how verification is conducted in each setting; we write $\mathcal{D}$ for the ODE $\dot{x} = -y, \dot{y} = x$.

- Initial condition $x = 0 \wedge y = 0$ at location q_0: The verifier finds the inductive invariant $\{q_0 \mapsto x = 0 \wedge y = 0, q_1 \mapsto x = 0 \wedge y = 0\}$ after asking for proofs of unsatisfiability to the user 5 times.
- Initial condition $x \le \frac{1}{2}$ at location q_0: The verifier finds a counterexample $\{x \mapsto 0.490533, y \mapsto 1.93995\}$, from which the system reaches $\{x \mapsto 2.00100, y \mapsto 0\}$. The verifier asks 5 questions, one of which is the following:

  `Pre:` $(x \le 1 \wedge y \ge 0) \vee x \le 0.5$`; Flow:` $\mathcal{D}$`; Stay:` $y \ge 0$`;`
  `Guard:` $y \le 0$`; Cmd:` **skip**`; CE:` $\{x \mapsto 0.998516; y \mapsto -1.889365\}$`;`
  `Init:` $x \le 0.5$`.`

  Notice that the stay condition is $y \ge 0$ and the guard is $y \le 0$; therefore the predicate $y = 0$ holds when a jump transition happens. Since the flow specified by $\mathcal{D}$ is an anticlockwise circle whose center is $\{x \mapsto 0, y \mapsto 0\}$ with the stay condition $y \ge 0$, the states after the flow dynamics followed by a jump transition is $x \le 0.5 \wedge y = 0$, which indeed does not intersect with $x = 0.998516 \wedge y = -1.889365$. The verification proceeds by giving $y \ge 0$ as a generalization in this case.
- Initial condition $0 \le x \le \frac{1}{2} \wedge 0 \le y \le \frac{1}{2}$ at location q_0: The verifier finds an inductive invariant

$$R := \left\{ \begin{array}{l} q_0 \mapsto (y = 0 \wedge 0 \le x \le 0.707107) \vee (0 \le x \le 0.5 \wedge 0 \le y \le 0.5), \\ q_1 \mapsto y = 0 \wedge -0.707107 \le x \le 0 \end{array} \right\}$$

after asking for 8 generalizations to the user. This is indeed an inductive invariant. Noting $0.707107 \approx \frac{1}{\sqrt{2}}$, we can confirm that (1) the states that are reachable by flow dynamics followed by a jump transition is the set denoted by $R(q_0)$; the same holds for the transition from $R(q_1)$; (2) it contains the initial condition $0 \leq x \leq 0.5 \wedge 0 \leq y \leq 0.5$ at location q_0; and (3) it does not intersect with the unsafe region $x > 1$. The following is one of the questions that are asked by the verifier:

> `Pre:` $(y = 0 \wedge -0.707107 \leq x \leq 0) \vee (0 \leq x \leq 0.5 \wedge 0 \leq y \leq 0.5)$`;`
> `Flow:` $\mathcal{D}$`; Stay:` $y \leq 0$`; CE:` $\{x \mapsto 0.998516; y \mapsto -1.889365\}$`;`
> `Init:` *false*.

Instead of a precise overapproximation $(x^2 + y^2 = 0.5 \wedge y \leq 0) \vee (0 \leq x \leq 0.5 \wedge 0 \leq y \leq 0.5)$ of the reachable states, we give $(-0.707107 \leq y \leq 0 \wedge -0.707107 \leq x \leq 0.707107) \vee (0 \leq x \leq 0.5 \wedge 0 \leq y \leq 0.5)$, which progresses the verification.

7 Related Work

Compared to its success in software verification [5,10,12,20,21], IC3/PDR for hybrid systems is less investigated. HyComp [11,13] is a model checker that can use several techniques (e.g., IC3, bounded model checking, and k-induction) in its backend. Before verifying a hybrid system, HyComp discretizes its flows so that the verification can be conducted using existing SMT solvers that do not directly deal with continuous-time dynamics. Compared to HyComp, HGPDR does not necessarily require prior discretization for verification. We are not aware of an IC3/PDR-based model checking algorithm for hybrid systems that does not require prior discretization.

Kindermann et al. [24,25] propose an application of PDR for a timed system—a system that is equipped with *clock variables*; the flow dynamics of a clock variable c is limited to $\dot{c} = 1$. A clock variable may be also reset to a constant in a jump transition. Kindermann et al. finitely abstract the state space of clock variables by using region abstraction [38]. The abstracted system is then verified using the standard PDR procedure. Later Isenberg et al. [22] propose a method that abstracts clock variables by using zone abstraction [4]. They do not deal with a hybrid system whose flow behavior at each location cannot be described by $\dot{c} = 1$; the system in Fig. 2 is out of the scope of their work.

Our continuous-reachability predicates (CRP) are inspired by Platzer's $d\mathcal{L}$ [33]. We may be able to use the theorem prover KeYmaera X for $d\mathcal{L}$ predicates [16] for our purpose of discharging CRP.

8 Conclusion

We proposed an adaptation of GPDR to hybrid systems. For this adaptation, we extended the logic on which the forward predicate transformer is defined with the continuous reachability predicates $\langle \mathcal{D} \mid \varphi_I \rangle \varphi$ inspired by the differential dynamic logic $d\mathcal{L}$. The extended forward predicate transformer can precisely express the behavior of hybrid systems. We formalized our procedure HGPDR and proved its soundness. We also implemented it as a semi-automated procedure, which proves the safety of a simple hybrid system in Fig. 2.

On top of the current proof-of-concept implementation, we plan to implement a GPDR-based model checker for hybrid systems. We expect that we need to improve the heuristic used in the application of the rule Induction, where we currently check sufficient conditions of the verification condition. We are also looking at automating part of the work currently done by human in verification; this is essential when we apply our method to a system with complex continuous-time dynamics.

Acknowledgements. We appreciate the comments from the anonymous reviewers, John Toman, and Naoki Kobayashi. This work is partially supported by JST PRESTO Grant Number JPMJPR15E5, JSPS KAKENHI Grant Number 19H04084, and JST ERATO MMSD project.

References

1. Hindmarsh, A.C.: ODEPACK, a systematized collection of ODE solvers. In: Stepleman, R.S., et al. (eds.) Scientific Computing, North-Holland, Amsterdam, vol. 1 of IMACS Transactions on Scientific Computation, pp. 55–64 (1983). http://www.llnl.gov/CASC/nsde/pubs/u88007.pdf
2. Albarghouthi, A., McMillan, K.L.: Beautiful interpolants. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 313–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_22
3. 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 1991-1992. LNCS, vol. 736, pp. 209–229. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57318-6_30
4. Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone-based abstractions of timed automata. STTT **8**(3), 204–215 (2006)
5. Birgmeier, J., Bradley, A.R., Weissenbacher, G.: Counterexample to induction-guided abstraction-refinement (CTIGAR). In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 831–848. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_55
6. Bjørner, N., Gurfinkel, A., McMillan, K., Rybalchenko, A.: Horn clause solvers for program verification. In: Beklemishev, L.D., Blass, A., Dershowitz, N., Finkbeiner, B., Schulte, W. (eds.) Fields of Logic and Computation II. LNCS, vol. 9300, pp. 24–51. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23534-9_2
7. Bradley, A.R.: SAT-based model checking without unrolling. In: Jhala, R., Schmidt, D. (eds.) VMCAI 2011. LNCS, vol. 6538, pp. 70–87. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-18275-4_7
8. Champion, A., Chiba, T., Kobayashi, N., Sato, R.: ICE-based refinement type discovery for higher-order functional programs. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10805, pp. 365–384. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89960-2_20

9. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_18
10. Cimatti, A., Griggio, A.: Software model checking via IC3. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 277–293. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_23
11. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Parameter synthesis with IC3. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20–23, 2013, pp. 165–168 (2013). http://ieeexplore.ieee.org/document/6679406/
12. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: IC3 modulo theories via implicit predicate abstraction. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 46–61. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_4
13. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: HyComp: an SMT-based model checker for hybrid systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 52–67. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_4
14. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
15. Colón, M.A., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using non-linear constraint solving. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 420–432. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45069-6_39
16. Fulton, N., Mitsch, S., Quesel, J.-D., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 527–538. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_36
17. Hashimoto, K., Unno, H.: Refinement type inference via horn constraint optimization. In: Blazy, S., Jensen, T. (eds.) SAS 2015. LNCS, vol. 9291, pp. 199–216. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48288-9_12
18. Hasuo, I., Suenaga, K.: Exercises in *nonstandard static analysis* of hybrid systems. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 462–478. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_34
19. Henzinger, T.A., Ho, P., Wong-Toi, H.: HYTECH: a model checker for hybrid systems. STTT **1**(1–2), 110–122 (1997). https://doi.org/10.1007/s100090050008
20. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 157–171. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31612-8_13
21. Hoder, K., Bjørner, N., de Moura, L.: μZ – an efficient engine for fixed points with constraints. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 457–462. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_36
22. Isenberg, T., Wehrheim, H.: Timed automata verification via IC3 with zones. In: Merz, S., Pang, J. (eds.) ICFEM 2014. LNCS, vol. 8829, pp. 203–218. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11737-9_14
23. Kapur, A., Henzinger, T.A., Manna, Z., Pnueli, A.: Proving safety properties of hybrid systems. In: Langmaack, H., de Roever, W.-P., Vytopil, J. (eds.) FTRTFT 1994. LNCS, vol. 863, pp. 431–454. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58468-4_177

24. Kindermann, R.: SMT-based verification of timed systems and software. Ph. D. thesis, Aalto University, Helsinki, Finland (2014). https://aaltodoc.aalto.fi/handle/123456789/19852
25. Kindermann, R., Junttila, T., Niemelä, I.: SMT-based induction methods for timed systems. In: Jurdziński, M., Ničković, D. (eds.) FORMATS 2012. LNCS, vol. 7595, pp. 171–187. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33365-1_13
26. Lange, T., Neuhäußer, M.R., Noll, T.: IC3 software model checking on control flow automata. In: Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27–30, 2015, pp. 97–104 (2015)
27. Lebeltel, O., Cotton, S., Frehse, G.: The SpaceEx modeling language (December 2010)
28. Liu, J., Zhan, N., Zhao, H.: Computing semi-algebraic invariants for polynomial dynamical systems. In: Proceedings of the 11th International Conference on Embedded Software, EMSOFT 2011, Part of the Seventh Embedded Systems Week, ESWeek 2011, Taipei, Taiwan, October 9–14, 2011, pp. 97–106 (2011). https://doi.org/10.1145/2038642.2038659
29. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
30. Nakamura, H., Kojima, K., Suenaga, K., Igarashi, A.: A nonstandard functional programming language. In: Chang, B.-Y.E. (ed.) APLAS 2017. LNCS, vol. 10695, pp. 514–533. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71237-6_25
31. Okudono, T., Nishida, Y., Kojima, K., Suenaga, K., Kido, K., Hasuo, I.: Sharper and simpler nonlinear interpolants for program verification. In: Chang, B.-Y.E. (ed.) APLAS 2017. LNCS, vol. 10695, pp. 491–513. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71237-6_24
32. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reason. **41**(2), 143–189 (2008). https://doi.org/10.1007/s10817-008-9103-8
33. Platzer, A.: Differential dynamic logics. KI **24**(1), 75–77 (2010). https://doi.org/10.1007/s13218-010-0014-6
34. 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). https://doi.org/10.1007/978-3-540-24743-2_32
35. Sankaranarayanan, S., Sipma, H., Manna, Z.: Non-linear loop invariant generation using gröbner bases. In: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14–16, 2004, pp. 318–329 (2004). https://doi.org/10.1145/964001.964028
36. Suenaga, K., Hasuo, I.: Programming with infinitesimals: a WHILE-language for hybrid system modeling. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6756, pp. 392–403. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22012-8_31
37. Suenaga, K., Sekine, H., Hasuo, I.: Hyperstream processing systems: nonstandard modeling of continuous-time signals. In: The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013, Rome, Italy - January 23–25, 2013, pp. 417–430 (2013). https://doi.org/10.1145/2429069.2429120
38. Wang, F.: Efficient verification of timed automata with bdd-like data structures. STTT **6**(1), 77–97 (2004). https://doi.org/10.1007/s10009-003-0135-4