Relational Abstractions For Continuous and Hybrid Systems

Sriram Sankaranarayanan[1] and Ashish Tiwari[2]

1. University of Colorado, Boulder, CO. srirams@colorado.edu
2. SRI International, Menlo Park, CA. ashish.tiwari@sri.com

Abstract. There has been much recent progress on invariant generation techniques for continuous systems whose dynamics are described by Ordinary Differential Equations (ODE). In this paper, we present a simple abstraction scheme for hybrid systems that abstracts continuous dynamics by relating any state of the system to a state that can potentially be reached at some future time instant. Such relations are then interpreted as discrete transitions that model the continuous evolution of states over time. We adapt template-based invariant generation techniques for continuous dynamics to derive relational abstractions for continuous systems with linear as well as non-linear dynamics. Once a relational abstraction has been derived, the resulting system is a purely discrete, infinite-state system. Therefore, techniques such as k-induction can be directly applied to this abstraction to prove properties, and bounded model-checking techniques applied to find potential falsifications. We present the basic underpinnings of our approach and demonstrate its use on many benchmark systems to derive simple and usable abstractions.

1 Introduction

Hybrid automata model systems that run in different modes. In each mode, a hybrid system behaves as a continuous dynamical system, whose dynamics is specified through Ordinary Differential Equations (ODEs). However, the system can also transition between modes. These transitions are specified by means of logical assertions over the current and next state variables known as transition relations.

In this paper, we present *relational abstractions* of hybrid systems. Relational abstraction transforms a given hybrid system into a purely discrete transition system by *summarizing* the effect of the continuous evolution of states over time using relations. The abstract discrete system is an infinite-state system that can be analyzed using standard techniques for verifying such systems such as k-induction and bounded model checking.

Relational abstractions preserve the discrete behavior of the hybrid system and only abstract its continuous behavior. They work by replacing the continuous dynamics in each mode by means of a relation $R(\boldsymbol{x}_0, \boldsymbol{x})$. The relation R relates a continuous state $\boldsymbol{x}_0$ with a state $\boldsymbol{x}$ that can be potentially reached at some future time instant, through some time trajectory of the system starting from $\boldsymbol{x}_0$.

Such a relation R can be interpreted in two ways: **(a)** As a positive invariant set for an associated dynamical system over $\boldsymbol{x}_0, \boldsymbol{x}$, and **(b)** As a discrete transition relation that abstracts the evolution of the continuous states over time.

The two views above provide two key advantages of the relational abstraction approach. As a consequence of the first view above, we can use techniques for generating invariants for continuous systems to generate a relational abstraction. We propose simple extensions of template-based invariant generation techniques, which can abstract systems with linear as well as non-linear dynamics, to construct relational abstractions. Template-based techniques allow us to specify the form of the relational abstraction [11, 21]. Therefore, our technique can be used to obtain relational abstractions in *linear* arithmetic for systems with either linear or *non-linear* dynamics.

As a consequence of the second view above, we obtain discrete infinite-state abstractions of hybrid systems. This enables us to use techniques such as k-induction using decision procedures [43], abstract interpretation [13, 23], or virtually any technique for discrete systems, to analyze hybrid systems.

It is well known that the problem of verifying hybrid systems is quite hard, both in theory and in practice. Recently, there have been many advances that have yielded remarkably efficient tools for integrating affine ODEs over sets and that work over large state spaces [27, 6, 45, 18, 40, 19]. However, we have observed that a significant gap in performance remains when these techniques are used to perform symbolic model checking, along the lines of tools such as HyTech and PHAVer [25, 17]. In our experience, this gap stems from the need to handle the dynamics repeatedly for the same mode, often with small variations between sub-problems. In this paper, we hypothesize that the situation with continuous dynamics is analogous to that of function calls encountered during program analysis. During inter-procedural program analysis, it is often observed that the analysis of each function call, given the state at the entry to the call, is quite efficient. However, the overall inter-procedural analysis is often not scalable due to repeated analysis of the same function with different actual parameters. Therefore, as in the case of function calls in program analysis, we propose *summarization techniques* that abstract the effect of the dynamics in each mode by a discrete transition. As a result, our technique can efficiently handle continuous dynamics. However, on the flip side, our approach may lose precision if the relational abstraction is too conservative. Furthermore, the computation of relational invariants implicitly doubles the number of state variables.

Our approach is able to prove safety properties of hybrid systems using techniques such as k-induction, as well as to discover potential violations through bounded model checking. To evaluate the idea of using relational abstractions of hybrid systems, we generate relational abstractions of some standard benchmarks and model check these abstractions. We generate relational abstractions using a combination of quantifier elimination tools (REDLOG, QEPCAD) to search for templatized invariants [48, 10, 44], and polyhedral analysis of ODEs using fixed point iteration over cones [42]. We analyze the resulting relational abstractions using the SAL framework from SRI [37, 47]. Our preliminary ex-

periments are quite promising: our approach has the ability to prove properties of hybrid systems that are known to be complex, while at the same performing much more efficiently than symbolic model checkers. The data from our experiments along with an extended version of this paper with proofs will be made available on-line [1].

We now discuss other related ideas in the literature.

Transition Invariants and Variance Analysis: The idea of defining "progress" invariant predicates over pairs of states, $\mathtt{x}, \overline{\mathtt{x}}$, is well-known in the field of program analysis. The abstraction (or summarization) of loops and function calls in programs by means of relations is a standard approach for verifying safety as well as liveness properties in programs [30]. There has been a lot of work on verifying *liveness* properties using ranking functions, transition invariants and progress invariants [5, 33, 12, 20]. However, there are some important distinctions between these various forms of relational invariants. Transition invariants [33] capture the relationship between the current state and *any* previous state (at a particular program location). Transition invariants were used to prove termination. Progress invariants capture the relationship between the current state and the *immediately* previous state (at a particular program location) [20]. Progress invariants were used to compute complexity bounds of programs. The relational abstractions presented here have a subtle difference: they capture the relationship between the current state and *all* previous states *after* the current mode was entered. When combined with the entry and exit conditions of a mode, relational invariants are exactly *summaries* of that mode. We use relational invariants to create abstractions of hybrid systems that can be used to analyze the hybrid systems; for example, to verify *safety* properties.

Podelski and Wagner provide a verification procedure for (region) stability properties of hybrid systems [34], where they derive *binary reachability relations* over trajectories of a hybrid system, similar to what is being proposed here. However, there are two key differences in our methodology: (a) Our approach deals with the dynamics at each mode upfront, deriving relational abstractions. On the other hand, Podelski et al.'s technique transforms the entire hybrid system, relying on safety verification built into a tool such as Phaver to derive the relations [17]. Our goal in this paper is to make the process more efficient using constraint-based approaches and improve hybrid system safety verification in the first place. (b) Secondly, our ultimate goal is to verify safety properties efficiently as opposed to verifying stability.

Abstractions of Hybrid Systems: Many different types of *discrete abstractions* have been studied for hybrid systems including predicate abstraction [3, 46] and abstractions based on invariants [31]. The use of counter-example guided abstraction-refinement for iterative refinement has also been investigated in the past (Cf. Alur et al. [2] and Clarke et al. [8], for example). In this paper, the proposed abstraction yields a discrete but infinite state system.

Hybridization is a technique for converting non-linear systems into affine systems by subdividing the invariant region into numerous sub-regions and ap-

[1] Cf. `http://www.csl.sri.com/~tiwari/relational-abstraction/`

proximating the dynamics as a hybrid system by means of a linear differential inclusion in each region [26, 4, 14]. However, such a subdivision can be expensive as the number of dimensions increases and may not be feasible if the invariant region is unbounded.

Reachability Analysis: Reasoning about the reachable set of states for flows of non-linear systems is an important primitive that is used repeatedly in the analysis of non-linear hybrid systems. This has been addressed using a wide variety of techniques in the past, including algebraic and semi-algebraic geometric techniques, interval analysis, constraint propagation and Bernstein polynomials [35, 29, 32, 36, 15].

2 Preliminaries

In this section, we present the basic definitions and properties of continuous systems defined by Ordinary Differential Equations (ODE). Let $\mathbb{R}$ denote the set of real numbers. We use $\boldsymbol{a}, \ldots, \boldsymbol{z}$ with subscripts to denote (column) vectors and $A, \ldots, Z$ to denote matrices. For a $m \times n$ matrix A, the row vector A_i, for $1 \leq i \leq m$, denotes the i^{th} row. We define continuous systems using vector fields.

Definition 1 (Vector Field). *A vector field $\mathfrak{F}$ over a set $X \subseteq \mathbb{R}^n$ is a function $\mathfrak{F} : X \mapsto \mathbb{R}^n$ mapping each $\boldsymbol{x} \in X$ with a field direction $\mathfrak{F}(\boldsymbol{x})$.*

Vector fields commonly arise from the definition of time invariant systems. A time invariant system defined by the ODE $\frac{dx_1}{dt} = f_1(\boldsymbol{x}), \ldots, \frac{dx_n}{dt} = f_n(\boldsymbol{x})$ can be identified with the vector field $\mathfrak{F}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$. Therefore, a continuous system $\mathcal{S} : \langle \mathfrak{F}, X \rangle$ is defined by a tuple consisting of the vector field $\mathfrak{F}$ and a domain (also referred to as a mode invariant) $X \subseteq \mathbb{R}^n$. We now define the time trajectories of a continuous system:

Definition 2 (Time Trajectories). *A time trajectory of a continuous system $\mathcal{S} : \langle \mathfrak{F}, X \rangle$ is a function $\tau : [0, T) \mapsto \mathbb{R}^n$ for some $T > 0$, such that: $\tau(t) \in X$, for all $t \in [0, T)$ and $\frac{d\tau}{dt} = \mathfrak{F}(\tau(t)),\ \forall\ t \in [0, T)$.*

Note 1. To facilitate presentation, we have (deliberately) restricted our attention to time invariant and autonomous systems. The full generalization to time variant, non-autonomous systems will be presented in an extended version.

If the continuous system $\mathcal{S}$ is defined by a *Lipschitz continuous* vector field $\mathfrak{F}$, then for any $\boldsymbol{x}_0 \in X$, we can guarantee the existence of a unique time trajectory τ such that $\tau(0) = \boldsymbol{x}_0$ [28]. Henceforth, we will assume that the systems considered are defined by Lipschitz continuous vector fields.

An affine system $\mathcal{S}$ is a continuous system whose dynamics are defined by an affine vector field $\frac{d\boldsymbol{x}}{dt} = A\boldsymbol{x} + \boldsymbol{b}$.

If $f(\boldsymbol{x})$ is continuous and differentiable over $\boldsymbol{x}$ then we write $\partial_{\boldsymbol{x}} f$ to denote the vector of its partial derivatives w.r.t each $\boldsymbol{x}_i$. The *Lie derivative* of a function g with respect to a field $\mathfrak{F}$ is given by $\mathcal{L}_F(g) := (\partial_{\boldsymbol{x}} g) \cdot \mathfrak{F}(\boldsymbol{x})$, where '$\cdot$' computes the dot product of two vectors.

Positive Invariant Set: A set $M \subseteq X$ is an invariant set for the system $\mathcal{S}$ iff for any $\boldsymbol{x} \in M$, and for each time trajectory $\tau : [0, T) \mapsto X$ such that $\tau(0) = \boldsymbol{x}$ is entirely contained in M; that is, $(\forall\ t \in [0, T))\ \tau(t) \in M$.

Let M be a closed set defined by the assertion $\bigwedge_{j=1}^{m} g_j(\boldsymbol{x}) \leq 0$ for some finite m. For technical reasons, as suggested by Blanchini and Miani, we assume that each $g_j(\boldsymbol{x})$ is a continuous and differentiable, and is a "practical set" satisfying the constraint qualification (Cf. Blanchini & Miani [7], page 104)

$$(\forall\ \boldsymbol{x} \in X),\ (\exists \boldsymbol{z})\ g_j(\boldsymbol{x}) + \partial_{\boldsymbol{x}} g_j \cdot\ \boldsymbol{z} < 0\,. \tag{1}$$

Informally, the constraint qualifications ensure that $\bigwedge_j g_j(\boldsymbol{x}) < 0$ represents the (relative) interior of the set M and $\bigvee_j g_j(\boldsymbol{x}) = 0$ represents the boundary. It can be shown that all affine functions g_j and positive-semidefinite quadratic forms (defining n-dimensional ellipsoids) satisfy these conditions.

Theorem 1. *The set $M : \bigwedge_{j=1}^{m} g_j(\boldsymbol{x}) \leq 0$ is a positive invariant for the vector field $\mathfrak{F}$ if for each $j \in [1, m]$ the following assertion holds true: $\forall\ \boldsymbol{x} \in X : g_j(\boldsymbol{x}) = 0\ \wedge\ \bigwedge_{i \neq j} g_i(\boldsymbol{x}) \leq 0\ \Rightarrow\ \mathcal{L}_F(g_j) < 0$.*

The theorem states that under appropriate conditions, a closed set M is a positive invariant set if the vector field $\mathfrak{F}$ lies in the tangent cone at each point on the boundary of the set. It is a direct consequence of Nagumo's theorem, a more general result that holds for non-Lipschitz continuous dynamics and non "practical" sets as well. The interested reader is referred to Blanchini and Miani (Chapter 4) for further details [7]. The theorem above provides a basis for various techniques for generating invariants for continuous systems using quantifier elimination and constraint solving [35, 42, 21, 32].

Hybrid Systems: Hybrid systems combine the continuous evolution of state with discrete, instantaneous jumps that can alter the state as well as the dynamics of a system. We present the standard definition for hybrid systems following Henzinger [24].

Definition 3 (Hybrid System). *A hybrid system $\mathcal{H}$ is defined by a set of* discrete modes *$\langle m_1, \ldots, m_k \rangle$, wherein, each mode m_i is defined by a continuous system $\mathcal{S}_i : \langle \mathfrak{F}_i, X_i \rangle$. The system can change modes through a set of* discrete transitions *$\tau_1, \ldots, \tau_m$. Each transition is defined by a prior mode m_0, a post-mode m_1 and a transition relation $\rho[\boldsymbol{x}, \boldsymbol{x}'] \subseteq X_{m_0} \times X_{m_1}$, that relates the state $\boldsymbol{x} \in X_{m_0}$ before the transition to the state $\boldsymbol{x}' \in X_{m_1}$ obtained as a result of taking the transition. The initial conditions are given by the initial mode m_{init} with the initial state set $\Theta \subseteq X_{\mathsf{init}}$.*

A hybrid system is a *switched system* if each discrete transition of the system does not modify the continuous state variables. In other words, each discrete transition relation $\rho[\boldsymbol{x}, \boldsymbol{x}']$ can be written as $\rho : \gamma(\boldsymbol{x})\ \wedge\ \boldsymbol{x}' = \boldsymbol{x}$, for guard $\gamma(\boldsymbol{x})$.

Example 1 (Switched system). Figure 1 shows the circuit diagram for a voltage controlled switch that closes whenever the voltage across the capacitor (V_C) exceeds $4V$, and open whenever V_C goes below $1V$.

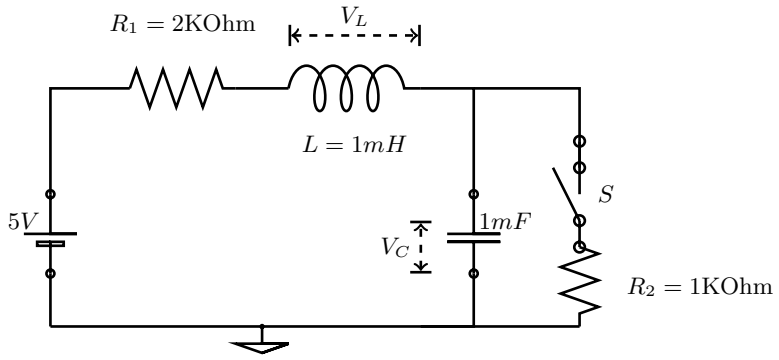

Fig. 1. Circuit diagram for an LCR circuit with a voltage controlled switch S.

With the switch S open, the dynamics of the voltage across capacitor V_C and the voltage across the inductor V_L are given by $\frac{dV_C}{dt} = 5 - V_C - V_L$, $\frac{dV_L}{dt} = -5 + V_C - V_L$.

Likewise, with the switch S closed, the dynamics of the voltage across the inductor is given by: $\frac{dV_C}{dt} = 5 - 3V_C - V_L$, $\frac{dV_L}{dt} = -5 + 3V_C - V_L$. In each mode, we assume the mode invariant $(V_C, V_L) \in [-10, 10] \times [-10, 10]$.

2.1 Constraint-Based Invariant Generation

Constraint-based invariant generation techniques fix a *template form* for the desired invariant and derive constraints over the unknown parameters that ensure that any solution is an invariant. A template form is specified as $g(\boldsymbol{c}, \boldsymbol{x}) \leq 0$, wherein g is a continuous function over unknown parameters in $\boldsymbol{c}$ and system variables $\boldsymbol{x}$. A simple example consists of affine templates:

$$c_0 + c_1 x_1 + \ldots + c_m x_m \leq 0$$

The overall technique for computing invariants using templates consists of the following key steps:

1. We first fix a template form for the desired linear invariants. The template form involves the program variables as well as unknown parameters. We represent the template as

   $$g_1(\boldsymbol{c}_1, \boldsymbol{x}) \ \leq \ 0 \ \wedge \ g_2(\boldsymbol{c}_2, \boldsymbol{x}) \leq 0 \ \wedge \ \cdots \ \wedge \ g_m(\boldsymbol{c}_m, \boldsymbol{x}) \leq 0 \,,$$

   over program variables $\boldsymbol{x}$ and unknowns $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$. We assume that each g_j satisfies the constraint qualification stated in Equation (1) (Cf. page 5). While our presentation here focusses on conjunctions of template inequalities, it is possible to extend this technique to arbitrary Boolean combinations of templates.
2. We encode the conditions required for the template to be an invariant of the system. For the case of ODEs, we encode the condition in Theorem 1,

recalled below, for each $j \in [1, m]$:

$$(\forall\, \boldsymbol{x} \in X),\ g_j(\boldsymbol{c}_j, \boldsymbol{x}) = 0 \wedge \bigwedge_{i \neq j} g_i(\boldsymbol{c}_i, \boldsymbol{x}) \leq 0 \ \Rightarrow\ \partial_{\boldsymbol{x}}(g_j(\boldsymbol{c}_j, \boldsymbol{x}))\mathfrak{F}(\boldsymbol{x}) < 0\,. \quad (2)$$

3. Eliminating the universally quantified variables in front of each of these assertions yields a system of constraints Ψ over $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m$. Solutions to these constraints upon instantiation yield the required invariants.

The key conceptual step consists of eliminating the universally quantified constraints in condition (2). If each of the templatized functions $g_j(\boldsymbol{c}_j, \boldsymbol{x})$ is a polynomial, this can be performed by using standard techniques for quantifier elimination (QE) over the theory of reals such as *cylindrical algebraic decomposition*. On the other hand, eliminating quantifiers over reals is often quite expensive and yields constraints that are hard to solve. Therefore, a standard approach consists of *dualizing* the condition (2) using *Farkas' Lemma* or Lagrangian relaxation. This converts the universal quantifiers into existential quantifiers, yielding the constraints:

$$(\exists\, \boldsymbol{\lambda}, \mu)\ \boldsymbol{\lambda} \geq 0\ \wedge\ (\mu g_j + \sum_{i \neq j} \boldsymbol{\lambda}_i g_i \equiv (\partial_{\boldsymbol{x}} g_j)\mathfrak{F}(\boldsymbol{x}))\,. \quad (3)$$

Note that we use $a(\boldsymbol{c}, \boldsymbol{x}) \equiv b(\boldsymbol{c}, \boldsymbol{x})$ for two polynomial expressions over $\boldsymbol{x}$ to signify that coefficients for each monomial on the LHS is equal to the coefficient on the RHS.

Example 2. Consider the problem of computing invariants of the LCR circuit from Example 1. We consider the mode with switch S open. We recall that the system variables are V_C, V_L with invariant $(V_C, V_L) \in [-10, 10] \times [-10, 10]$. The dynamics in this mode are recalled below:

$$\frac{dV_C}{dt} = 5 - V_C - V_L, \quad \frac{dV_L}{dt} = -5 + V_C - V_L$$

for the initial condition $V_c \in [0, 1], V_L \in [0, 1]$. Let us assume that the required invariant is of the form

$$c_0 + c_1 V_C + c_2 V_L \leq 0\,,$$

for unknown parameters $c_0, \ldots, c_2$.

We first encode the initial condition as:

$$(\forall V_C, V_L) V_C \in [0, 1]\ \wedge\ V_L \in [0, 1] \Rightarrow c_0 + c_1 V_C + c_2 V_L \leq 0$$

Dualizing using Farkas' Lemma and eliminating the resulting multipliers, we obtain the system for initiation:

$$c_0 \leq 0\ \wedge\ c_0 + c_1 \leq 0\ \wedge\ c_0 + c_2 \leq 0\ \wedge\ c_0 + c_1 + c_2 \leq 0\,.$$

Additionally, we encode that the dynamics are preserved by the flow

$$c_0 + c_1 V_C + c_2 V_L = 0\ \wedge\ (V_C, V_L) \in [-10, 10]^2 \Rightarrow$$
$$(c_1 - c_2) + (c_2 - c_1) V_c - (c_1 + c_2) V_L < 0$$

Dualizing using Farkas' lemma, yields the constraints:

$$\begin{array}{r} \lambda_1, \lambda_2, \lambda_3, \lambda_4 \geq 0 \\ \mu c_1 - \lambda_1 + \lambda_2 = c_2 - c_1 \\ \mu c_2 - \lambda_3 + \lambda_4 = -c_1 - c_2 \\ \mu c_0 - 10\lambda_1 + 10\lambda_2 - 10\lambda_3 + 10\lambda_4 > (c_1 - c_2) \end{array},$$

wherein $\boldsymbol{\lambda}, \mu$ are existentially quantified multiplier variables.

In general, the application of Farkas' lemma (or equivalently Lagrangian relaxation) yields a system of constraints of the following form:

$$(\exists \boldsymbol{\lambda}, \mu)\ \boldsymbol{\lambda} \geq 0\ \wedge\ D(\boldsymbol{\lambda}, \boldsymbol{c}, \mu) \tag{4}$$

wherein D is a set of bilinear inequalities involving multipliers $\boldsymbol{\lambda}, \mu$ and unknowns $\boldsymbol{c}$. There are many ways of eliminating the multipliers from this system:

1. Use quantifier elimination over reals, especially elimination techniques that take advantage of the fact that all polynomials in D are quadratic [11].
2. Use techniques based on constraint-logic programming to search for instantiations of $\boldsymbol{\lambda}, \mu$ by factoring so that the remaining constraints can be linearized [41].
3. Use bit-vector solvers to consider finite bit width instantiations for $\boldsymbol{\lambda}, \mu, \boldsymbol{c}$ [21]
4. Obtain constraints for $\boldsymbol{c}$ by simulating the dynamics. These constraints are linear, assuming that the templates are all linear over $\boldsymbol{c}$, and can be used to search for solutions of the system of constraints (4) [22].

2.2 Fixed Point Iteration

An alternative to template based invariant generation is a fixed point iteration over polyhedral cone. We present this approach at a high level. For more details, we refer the reader to prior work by one of the authors [42, 39, 38]. The tool TimePass (open source version developed at Stanford University) implements fixed point iteration for linear systems in conjunction with a flow pipe construction technique.

Like constraint based methods, the overall idea behind fixed point iteration is to fix a template of the form

$$g_1(\boldsymbol{c}_1, \boldsymbol{x})\ \leq\ 0\ \wedge\ g_2(\boldsymbol{c}_2, \boldsymbol{x}) \leq 0\ \wedge\ \cdots\ \wedge\ g_m(\boldsymbol{c}_m, \boldsymbol{x}) \leq 0\,.$$

Assuming that each of the functions g_j is linear over $\boldsymbol{c}_j$, we may conclude that the set of possible solutions form a cone. To obtain solutions that are also positive invariants, the technique starts from an initial cone C_0 containing all possible instantiations for $\boldsymbol{c} : (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m)$, and iteratively removes some of the inadmissible solutions $\boldsymbol{c}$ which fail to satisfy condition (2), using a monotone *refinement* operator. The refinement operator is iterated until a fixed point is obtained. The generators of the fixed point cone, when instantiated into the template yields the required invariants. Abstract interpretation techniques such as widening can be used to accelerate convergence to an approximate but sound invariant.

2.3 Flowpipe Construction

Flowpipe construction is a technique for computing sets of states that are guaranteed to enclose the reachable states over time instances that lie inside an interval $I : [\Delta_1, \Delta_2]$. There are numerous techniques for flowpipe constructions; too many to list here. These techniques differ based on the type of sets used to enclose the flowpipes and the class of dynamical systems treated.

3 Relational Abstractions

In this section, we define relational abstractions for continuous systems, and present proof rules for checking that a given relation is an abstraction of the time trajectories of a continuous system defined by ODEs.

Let $\mathcal{S} : \langle \mathfrak{F}, X \rangle$ be a continuous system defined by the vector field $\mathfrak{F}$, and domain (invariant) X. It is assumed that $\mathcal{S}$ arises from a mode of a larger hybrid system. Let $R(\boldsymbol{x}, \boldsymbol{y})$ be a relation over $X \times X$.

Definition 4 (Relational Abstraction Without Time). *The relation $R \subseteq \mathbb{R}^{2n}$ is a (timeless) relational abstraction of a continuous system $\mathcal{S}$ if for all time trajectories $\tau : [0, T) \mapsto X$ of the system $\mathcal{S}$, it is the case that $(\forall\ t \in [0, T))\ (\tau(0), \tau(t)) \in R$.*

Thus, for a time invariant system, a relational abstraction R captures all pairs of states $(\boldsymbol{x}, \boldsymbol{y})$ such that it is possible to reach $\boldsymbol{y}$ from $\boldsymbol{x}$ in a finite amount of time by evolving according to the dynamics of the system.

A relational abstraction $R \subseteq X \times X$ is said to be *complete* for a system $\mathcal{S}$ if whenever $R(\boldsymbol{x}, \boldsymbol{y})$ holds, then there exists a time trajectory $\tau : [0, T) \mapsto X$ such that $\tau(0) = \boldsymbol{x}$ and $\tau(t) = \boldsymbol{y}$, for some time $0 \leq t < T$. Likewise, a relational abstraction R is linear if it can be expressed as an assertion in the theory of linear arithmetic over reals.

Note 2. A continuous system whose dynamics are defined by constants (such as, a mode of a multi-rate hybrid automaton) has a complete, linear relational abstraction. For instance, the evolution of the ODE $\frac{dx}{dt} = 2$, $\frac{dy}{dt} = -3$ can be abstracted by the relation

$$R(x, y, x', y') :=\ x' - x \geq 0\ \wedge\ \frac{1}{2}(x' - x) = \frac{-1}{3}(y' - y)\,.$$

In fact, we can show that hybrid systems with constant dynamics in each mode are bisimilar to a purely discrete transition system through relationalization. On the other hand, linear vector fields can fail to have complete abstractions.

We now define an "extended system" $\mathcal{S}'$ from a given system $\mathcal{S}$ such that invariants of $\mathcal{S}'$ will yield relational abstractions for $\mathcal{S}$.

Definition 5 (Extended System). *Let $\mathcal{S}$ be a continuous system over $\boldsymbol{x} \in \mathbb{R}^n$ defined by vector field $\mathfrak{F}$ and invariant region X. The extended system $\mathcal{S}'$ has state variables $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{2n}$ with the dynamics.*

$$\frac{d\boldsymbol{y}}{dt} = \mathfrak{F}(\boldsymbol{y}),\ \frac{d\boldsymbol{x}}{dt} = \boldsymbol{0}\,, \tag{5}$$

invariant region given by $X \times X$ and with the initial conditions $\boldsymbol{x}(0) = \boldsymbol{y}(0) \in X$.

We now refine the notion of positive invariants from Section 2 to account for the presence of initial conditions in the system.

Definition 6 (Initialized Positive Invariant). *A set M is an initialized positive invariant for the system $\mathcal{S}$ with initial conditions $X_0 \subseteq M$ iff for all time trajectories $\tau : [0, T) \mapsto \mathbb{R}^n$ of $\mathcal{S}$ starting from $\tau(0) \in X_0$ we have $\tau(t) \in M$ for all $t \in [0, T)$.*

An initialized positive invariant is an over-approximation of all states reachable through a time trajectory starting from some pre-specified set of initial states. This is, in fact, the true analog of an invariant for a program.

Note that every positive invariant set M (following the definition in Section 2) that contains the initial set X_0 is an initialized positive invariant. On the other hand, an initialized positive invariant may not be a general positive invariant. This is because, it may be possible for trajectories that start from some state in the set $M - X_0$ to exit the invariant set M.

Lemma 1. *A relation R is a relational abstraction of $\mathcal{S}$ if and only if R is an* initialized *positive invariant for $\mathcal{S}'$.*

Proof. For a tuple of states $(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^{2n}$, we will use the notation $\pi_1(\boldsymbol{x}, \boldsymbol{y})$ to refer to $\boldsymbol{x} \in \mathbb{R}^n$ and $\pi_2(\boldsymbol{x}, \boldsymbol{y})$ to refer to $\boldsymbol{y} \in \mathbb{R}^n$.

Consider a time trajectory $\sigma : [0, T) \mapsto \mathbb{R}^{2n}$ for (5) with initial conditions $\sigma(0) = (\boldsymbol{x}_0, \boldsymbol{x}_0)$. Let us define $\boldsymbol{y}(t) = \pi_2(\sigma(t))$. We observe that $\boldsymbol{y}(t)$ is a time trajectory of $\mathcal{S}$ with $\boldsymbol{y}(0) = \boldsymbol{x}_0$. To prove this, simply verify that $\boldsymbol{y}$ indeed evolves according to the vector field in $\mathcal{S}$ and stays inside the invariant region X.

Likewise, for each time trajectory τ of $\mathcal{S}$, the trajectory $\sigma : [0, T) \mapsto (\tau(0), \tau(t))$ is a time trajectory of system (5).

[$\Leftarrow$] Assume that set M is an initialized positive invariant set for the extended system $\mathcal{S}'$ defined according to (5). The following facts are true as a consequence:

1. $(\boldsymbol{x}, \boldsymbol{x}) \in M$ for all $\boldsymbol{x} \in X$.
2. For every trajectory σ of $\mathcal{S}'$, with initial conditions $\sigma(0) = (\boldsymbol{x}, \boldsymbol{x})$, we have $\sigma(t) \in M$ for all $t \in [0, T)$.

Let us take any trajectory τ of $\mathcal{S}$, we know that the trajectory $\sigma(t) : (\tau(0), \tau(t))$ is a valid trajectory for system (5) that is initialized in the set $\boldsymbol{x}(0) = \boldsymbol{y}(0) = \tau(0) \in X$. Therefore, $(\tau(0), \tau(t)) \in M$ for all time instances t for τ. This suffices to show that M is a relational abstraction for $\mathcal{S}$.

[$\Rightarrow$] Let a relation R be a relational abstraction for system $\mathcal{S}$. Consider any trajectory $\sigma : [0, T) \mapsto \mathbb{R}^{2n}$ of $\mathcal{S}'$ such that $\sigma(0)$ satisfies the initial conditions for $\mathcal{S}'$. We wish to prove that $\sigma(t) \in R$ for all $t \in [0, T)$ in order to show that R is an initialized positive invariant. Since $\sigma(0)$ satisfies the initial conditions $\boldsymbol{x}(0) = \boldsymbol{y}(0)$, we can write $\sigma(t)$ as $(\tau(0), \tau(t))$ for some trajectory τ of $\mathcal{S}$. Therefore, by definition of the relational abstraction R, we conclude that $(\tau(0), \tau(t)) \in R$ and in turn that $\sigma(t) \in R$. As a result R is a positive invariant for the system (5).

Therefore, if we can compute initialized positive invariants of the extended system $\mathcal{S}'$ with initial states $\boldsymbol{x}(0) = \boldsymbol{y}(0)$, we may use them to obtain relational abstractions. In this work, we use various techniques that can compute positive invariants M (using the definition in Section 2) of systems $\mathcal{S}$ that contain some initial set of states X_0.

Theorem 2. *Let M be a positive invariant of the extended system $\mathcal{S}'$ containing the initial states $X_0 = \{(\boldsymbol{x}, \boldsymbol{x}) \mid \boldsymbol{x} \in X\}$. Then M is a relational abstraction of the system $\mathcal{S}$.*

Proof. We note that a positive invariant M containing the initial states X_0 is also an initialized positive invariant. I.e, for any trajectory σ starting from X_0, we know that $\sigma(t) \in M$ since $\sigma(0) \in M$. The rest of the proof simply relies on Lemma 1.

The converse of the theorem above does not hold, in general. As discussed above, a positive invariant M containing the initial set of states X_0 is not necessarily an initialized positive invariant.

Note 3. The extended system can be expressed, equivalently, using the (time reversed) system instead of the system (5).

$$\frac{d\boldsymbol{y}}{dt} = \mathbf{0},\ \frac{d\boldsymbol{x}}{dt} = -\mathfrak{F}(\boldsymbol{x}) \tag{6}$$

with the initial conditions $\boldsymbol{x}(0) = \boldsymbol{y}(0)$.

In other words, a relational abstraction $R(\boldsymbol{x}, \boldsymbol{y})$ is a positive invariant of one of two dynamical systems: System 5 where $\boldsymbol{x}$ is frozen in time and $\boldsymbol{y}$ evolves according to the vector field $\mathfrak{F}$, and System 6 where $\boldsymbol{y}$ is frozen in time and $\boldsymbol{x}$ evolves according to the time reversed field $-\mathfrak{F}$.

Proof Rule for Relational Abstractions The proof rule for relational abstractions can be derived from the proof rule for invariant sets. Furthermore, techniques for synthesizing invariants can be directly used to synthesize relational abstractions. We now present a proof rule for checking if a relation R is a sound abstraction. We assume that the relation R is specified as an assertion of the form:

$$R(\boldsymbol{x}, \boldsymbol{y}) :\ g_1(\boldsymbol{x}, \boldsymbol{y}) \leq 0\ \wedge\ \ldots\ \wedge\ g_m(\boldsymbol{x}, \boldsymbol{y}) \leq 0\,,$$

wherein $g_1, \ldots, g_m$ are continuous and differentiable functions over $\mathbb{R}^{2n}$. Furthermore, for technical reasons, we assume that the set $R \cap (X \times X)$ in $\mathbb{R}^{2n}$ defined by the relation R restricted to X is a *closed* set and g_j satisfy the constraint qualifications in (1).

Definition 7. *The following rules allow us to conclude that the relation R, as specified above, is a relational abstraction of a continuous system $\mathcal{S}$:*

Initialization: $\forall\ \boldsymbol{x}\ \in\ X,\ R(\boldsymbol{x}, \boldsymbol{x})$*, and*

Flow Preservation: *We may use the rule for forward time:*

$$\forall j \in [1,m],\ \forall\, \boldsymbol{x},\boldsymbol{y}\ \in\ X,\ \bigwedge_{i\neq j} g_i(\boldsymbol{x},\boldsymbol{y}) \leq 0 \wedge g_j(\boldsymbol{x},\boldsymbol{y}) = 0\ \Rightarrow\ (\partial_{\boldsymbol{y}} g_j)\cdot\mathfrak{F}(\boldsymbol{y}) < 0\,,$$

or the rule for time reversed dynamics:

$$\forall j \in [1,m],\ \forall\, \boldsymbol{x},\boldsymbol{y}\ \in\ X,\ \bigwedge_{i\neq j} g_i(\boldsymbol{x},\boldsymbol{y}) \leq 0 \wedge g_j(\boldsymbol{x},\boldsymbol{y}) = 0\ \Rightarrow\ (\partial_{\boldsymbol{x}} g_j)\cdot(-\mathfrak{F}(\boldsymbol{x})) < 0\,.$$

Example 3. We now consider relationalizations for the LCR circuit in Example 1. Consider the mode when the switch is open with dynamics given by $\frac{dV_C}{dt} = 5 - V_C - V_L,\quad \frac{dV_L}{dt} = -5 + V_C - V_L$.

We wish to show that the relation $R(V_{C0}, V_{L0}, V_C, V_L)$, represented by the assertion below, is a relational abstraction: $(V_{C0}, V_C, V_{L0}, V_L)\ \in\ [-10,10]^4\ \wedge\ V_C + 5V_L \leq V_{C0} + 50\ \wedge\ 4V_L \leq V_{L0} + 30\ \wedge\ 2V_L - 3V_C \leq 2V_{L0} + 30$.

Let us consider the inequality $V_C + 5V_L - V_{C0} - 50\ \leq\ 0$. For the initial condition, we set $V_C = V_{C0}$ and $V_L = V_{L0}$ and verify that $5V_{L0} \leq 50$ holds over the invariant region $(V_{C0}, V_{L0}) \in [-10,10]^2$. Likewise, the Lie derivative of the LHS expression is given by $4V_C - 6V_L - 20$. We verify the following entailment using an SMT solver

$$R(V_{C0}, V_{L0}, V_C, V_L)\ \wedge\ V_C + 5V_L - V_{C0} - 50 = 0\ \models\ 4V_C - 6V_L - 20 < 0\,.$$

The remaining constraints are similarly verified.

Disjunctive Relational Abstraction Often, the relational abstraction can be represented as the disjunction $R(\boldsymbol{x},\boldsymbol{y}) : \bigvee_{j=1}^{m} R_j(\boldsymbol{x},\boldsymbol{y})$ of finitely many relations $R_1, \ldots, R_m$, such that (a) each relation R_j is represented by an assertion over $\boldsymbol{x},\boldsymbol{y}$ satisfying the flow preservation proof rule in Def. 7, and (b) the disjunctive relation $R(\boldsymbol{x},\boldsymbol{y})$ satisfies the initialization rule.

Example 4. Consider, once again, the LCR circuit in Example 1. The relation below is a disjunctive relational abstraction for the switch open mode:

$$|V_L| \leq \max(|V_{L0}|, |V_{C0} - 5|)\ \wedge\ |V_C - 5| \leq \max(|V_{L0}|, |V_{C0} - 5|)\,.$$

Verifying this fact can be performed by expanding the definitions of max and $|\cdot|$. The resulting assertion is cast in the disjunctive normal form and the flow preservation proof rule in Def. 7 can be checked for each disjunct, and the initialization rule can be checked for the whole disjunction.

3.1 Relational Abstractions over Time Intervals

The timeless relational abstraction $R(\boldsymbol{x},\boldsymbol{y})$ (Definition 4) holds between a state $\boldsymbol{x}$ and all states that are reached at all possible future time instants starting from $\boldsymbol{x}$. It is also useful to define a similar concept that restricts the possible future time instants considered.

Let $\mathcal{S} : \langle \mathfrak{F}, X\rangle$ be a continuous system. Let $I :\ [\Delta_1, \Delta_2]$ be a non-empty interval over the real line. We assume that $\Delta_2 > \Delta_1 \geq 0$.

Definition 8 (Limited Time Relational Abstractions). *A relation* $R_I \subseteq \mathbb{R}^n \times \mathbb{R}^n$ *is a limited time relational abstraction of* $\mathcal{S}$ *over the interval* I *if for all* $\boldsymbol{x}_0 \in X$*, for all time trajectories* $\tau : [0, T) \mapsto \mathbb{R}^n$ *starting from* $\boldsymbol{x}_0$ *(i.e.,* $\tau(0) = \boldsymbol{x}_0$*), and* for all time instants $t \in I$, $(\tau(0), \tau(t)) \in R_I$.

We now relate limited time relational abstractions with timeless relational abstractions. First, we observe that limited time abstractions can be derived, once again, by searching for positive invariants of an associated system.

Lemma 2. *A relation* R_I *is a limited time relational abstraction for a system* $\mathcal{S} : \langle \mathfrak{F}, X \rangle$ *over the time interval* I *iff there is a positive invariant set* R' *of the associated system* $\mathcal{S}'$ *defined over variables* $\boldsymbol{x}, \boldsymbol{y}, s$ *with dynamics* $\frac{d\boldsymbol{x}}{dt} = 0$, $\frac{d\boldsymbol{y}}{dt} = \mathfrak{F}(\boldsymbol{y})$, $\frac{ds}{dt} = 1$ *such that* $\forall \boldsymbol{x} \in X, R'(\boldsymbol{x}, \boldsymbol{x}, 0)$ *holds and* $R_I = \exists s \in I : R'$.

Proof. The proof is simply a corrollary of Lemma 1 due to the following observation: R_I is a time limited abstraction for $\mathcal{S}$ iff it is a time unlimited abstraction for the system $\mathcal{S}' : \langle \mathfrak{F}', X' \rangle$ over variables $\boldsymbol{x} \cup \{s\}$. The new variable s represents time and has dynamics $\mathfrak{F}'(s) = 1$. For variables in the original system $\mathcal{S}$, we have $\mathfrak{F}(\boldsymbol{x}) = \mathfrak{F}'(\boldsymbol{x})$. Similarly, the invariant region X' is simply X with the added constraint $s \in I$.

An application of Lemma 1 yields the required result.

As a result, techniques for generating invariants of continuous systems can be used to generate limited time relational invariants. More importantly, however, is the fact that limited time relational abstractions over time intervals of the form $[0, \Delta]$ can be computed using flowpipe construction techniques that are commonly used to integrate ODEs over sets [27, 6, 45, 18, 40, 19]. Furthermore, given a limited time abstraction R_I over a time interval $[0, \Delta]$, we can derive a timeless abstraction R by computing the transitive closure of R_I.

The composition of $R \circ S$ of relations $R(\boldsymbol{x}, \boldsymbol{y})$ and $S(\boldsymbol{x}, \boldsymbol{y})$ is defined as $[R \circ S](\boldsymbol{x}, \boldsymbol{y}) : \ (\exists\ \boldsymbol{z})\ R(\boldsymbol{x}, \boldsymbol{z})\ \wedge\ S(\boldsymbol{z}, \boldsymbol{y})$.

Lemma 3. *Let* R_1 *be a limited time relational abstraction over* $[0, \Delta_1]$ *and* R_2 *be a limited time relational abstraction over* $[0, \Delta_2]$ *for* $\Delta_1, \Delta_2 > 0$*. Then,* $R_1 \circ R_2$ *is a limited time relational abstraction over the time interval* $[0, \Delta_1 + \Delta_2]$.

Proof. Let τ be a time trajectory over some time interval $[0, T]$ wherein $T \leq \Delta_1 + \Delta_2$. We wish to show that for any two points $\boldsymbol{x} = \tau(t_1)$ and $\boldsymbol{y} = \tau(t_2)$ wherein $0 \leq t_1 \leq t_2 \leq \Delta_1 + \Delta_2$, we have $R_1 \circ R_2(\boldsymbol{x}, \boldsymbol{y})$.

If $t_2 - t_1 \leq \Delta_1$, we note that $(\boldsymbol{x}, \boldsymbol{y}) \in R_1$. Furthermore, since $(\boldsymbol{y}, \boldsymbol{y}) \in R_2$, we obtain the required result. Similarly, we can handle the case when $t_2 - t_1 \leq \Delta_2$.

On the other hand, we have $t_2 - t_1 \leq \Delta_1 + \Delta_2$. Let us choose $t_3 \in [t_1, t_2]$ such that $0 \leq t_3 - t_1 \leq \Delta_1$ and $0 \leq t_2 - t_3 \leq \Delta_2$. Let $\boldsymbol{z} = \tau(t_3)$. We note that $R_1(\boldsymbol{x}, \boldsymbol{z})$ and $R_2(\boldsymbol{z}, \boldsymbol{y})$. Combining, these two observations yields the required result.

As a result, given a relation R_I valid over some time interval $[0, \Delta]$ for $\Delta > 0$, we obtain a timeless abstraction R^* as the fixed point of an iterative sequence of relations.

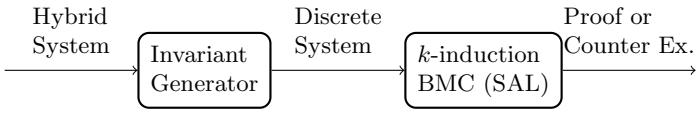

Fig. 2. Framework for implementing a safety verification engine using relationalization.

Lemma 4. *If R_I is a limited time abstraction over an interval $[0, \Delta]$, $\Delta > 0$, then any relation R that contains the fixed point of the iterative sequence*

$$R_0 := R_I, \;\; R_{n+1} := (R_n \circ R_n)$$

is a timeless abstraction.

Proof. The proof is a direct consequence of Lemma 3. We can prove by induction that R_n is a valid time limited abstraction over the interval $[0, (2^n)\Delta]$. Let R be an over-approximation of the fixed point. Let us assume that τ is any time trajectory and consider points $\boldsymbol{x} = \tau(t_1)$ and $\boldsymbol{y} = \tau(t_2)$ wherein $0 \leq t_1 \leq t_2$. Let $n = \left\lceil \log_2 \left(\frac{(t_2 - t_1)}{\Delta} \right) \right\rceil$. We note that $R_n(\boldsymbol{x}, \boldsymbol{y})$ holds. Since $R \supseteq R_n$, we conclude that $R(\boldsymbol{x}, \boldsymbol{y})$ holds. Therefore R is a time unlimited abstraction.

Thus, we can use flowpipe approximation techniques combined with a fixed point computation using abstract interpretation to generate timeless relational abstractions.

4 Implementation

Figure 2 shows the overall verification framework. It consists of two parts: (a) an invariant generator for generating the relational abstraction of the input hybrid system, and (b) a verifier for analyzing the relational abstraction using techniques such as k-induction and BMC. Note that other verification techniques/tools are equally applicable here. Our framework abstracts each mode upfront to yield a purely discrete system. It is possible, in practice, to implement the abstraction on-the-fly, depending on the verification procedure used, whenever a previously unseen mode is entered.

We now discuss the implementation of relational abstraction, restricting our attention here to techniques that have been employed in our experiments. We primarily apply template-based methods for generating relational abstractions [11, 21]. Template-based techniques formulate an unknown parameterized form for the required invariant and cast the problem of generating the invariant as an $\exists\forall$ formula. These $\exists\forall$ formulas can be solved directly using quantifier elimination techniques over the theory of reals [48, 9], or they can be first converted into $\exists$ formulas through through dualization. The $\exists$ formulas, which contains non-linear constraints over the unknown parameters, can be solved using either fixed point iteration over cones [42], or using bit-vector solvers [21], or by simulating the system numerically (along the lines of Gupta et al. [22]). In our experiments,

we use a specialized quantifier elimination technique [44] and the tool TimePass, which implements a fixed point iteration with widening over polyhedral cones for affine differential equations [42].

For each mode of the hybrid system, we consider three types of abstractions *affine*, *eigen*, and *box* abstractions.

Affine abstractions Affine abstractions employ the template: $\boldsymbol{a}\cdot\boldsymbol{x}+\boldsymbol{b}\cdot\boldsymbol{x}_0 \geq a_0$. In practice, the template: $\boldsymbol{a}(\boldsymbol{x}-\boldsymbol{x}_0) \geq a_0$ suffices after taking the initiation into account. Affine relational abstractions are computationally inexpensive to generate, but they are also of relatively poor quality.

Eigen abstractions For linear systems, such as $d\boldsymbol{x}/dt = A\boldsymbol{x}$, whenever A has real eigenvalues, useful relational abstractions can be generated using the eigenvectors of A^T corresponding to those real eigenvalues [45]. Here A^T denotes the transpose of matrix A. Specifically, if $\boldsymbol{c}$ is such that $A^T\boldsymbol{c} = \lambda\boldsymbol{c}$, then by simple algebraic manipulation, we obtain $\frac{d}{dt}(c_1x_1 + \ldots + c_nx_n) = \lambda(c_1x_1 + \ldots + c_nx_n)$ where $\boldsymbol{c} := [c_1; \ldots; c_n]$ and $\boldsymbol{x} := [x_1; \ldots; x_n]$. Let p denote the linear expression $c_1x_1 + \ldots + c_nx_n$ and let p_0 denote the linear expression $c_1x_{10} + \ldots + c_nx_{n0}$. Here x_{i0} denotes the old value of x_i. If $\lambda < 0$, then we know the value of p approaches zero monotonically. Consequently, we get the relational abstraction $(p_0 < 0 \Rightarrow p_0 \leq p < 0) \ \wedge \ (p_0 > 0 \Rightarrow p_0 \geq p > 0)$. Similarly, we can write the relational invariants for the case when $\lambda > 0$ and $\lambda = 0$.

Box abstractions Box relational abstractions take the following form:

$$Max(a_1|x_1|, \ldots, a_n|x_n|) \leq Max(a_1|x_{10}|, \ldots, a_n|x_{n0}|)$$

where a_i's are nonnegative real numbers. Note that box relational invariants are Boolean combinations of affine relational invariants. We can generate box relational invariants by finding appropriate a_i's. It turns out that this does not require expensive quantifier elimination and we can find box (relational) invariants in just $O(n^3)$ time [1]. Box invariants do not always exist. There are sufficient (and necessary) conditions for their existence. Example 4 provided a box invariant for the switch open mode.

5 Experimental Evaluation

We evaluate our approach over the navigation benchmarks [16], to experimentally evaluate the usefulness of relational abstractions for verifying hybrid systems. The navigation benchmarks model a vehicle moving in a 2-dimensional rectangular space $[0, m-1] \times [0, n-1]$. This space is partitioned in $m \times n$ cells. The vehicle has different dynamics in each of these cells. Let x, y denote the position of the vehicle and v_x, v_y denote its velocity. Then the dynamics of the vehicle in any particular cell is given by the ODEs:

$$\begin{array}{ll} \frac{dx}{dt} = v_x & \frac{dv_x}{dt} = a_{11}(v_x - b) + a_{12}(v_y - c) \\ \frac{dy}{dt} = v_y & \frac{dv_y}{dt} = a_{21}(v_x - b) + a_{22}(v_y - c) \end{array}$$

Benchmark	Affine Invs			Affine+Eigen Invs			Affine+Eigen+Box Invs		
	depth	status	time(s)	depth	status	time(s)	depth	status	time(s)
nav01	4	F	0.63	4	F	0.88	4	F	1.91
nav01	5	P	0.75	5	P	0.91	5	P	1.36
nav02	4	F	0.64	4	F	0.87	4	F	1.8
nav02	5	P	0.68	5	P	1.04	5	P	3.33
nav03	4	F	0.60	4	F	0.91	4	F	1.72
nav03	5	P	0.67	5	P	1.05	5	P	2.7
nav04	3	CE	0.49	8	F	3.21	8	F	34.883
nav04				4	P	0.75+0.99	4	P	0.98+2.21
nav05	2	CE	0.47	8	F	3.85	8	F	37.31
nav05				8	P	2.15+2.50	8	P	5.38+11.05
nav06	4	CE	0.61	8	F	18.01	8	F	494.5
nav06*	4	CE	1.03	8	P	21.80+7.42	8	P	40.22+35.08
nav07	5	CE	0.66	-	-	-	5	F	69.9
nav07				-	-	-	6	P	6.25
nav08	4	CE	0.52	-	-	-	6	CE	0.95
nav09	4	CE	0.57	4	CE	1.45	4	CE	19.87
nav10	3	CE	0.44	3	CE	0.99	3	CE	0.95

Table 1. Comparison of various abstractions over the NAV benchmarks. All experiments performed on a Intel Xeon E5630 2.53GHz single-core processor (x86_64 arch) with 4GB RAM running Ubuntu Linux 2.6.32-26. Proofs are attempted at a given depth k using k-induction. Legend — **depth:** depth for which k-induction was run, **time:** time taken by verifier, **status:** status of the verification run, **P**: Proved Property, **CE:** k-induction base case fails and counter-example is produced, **F**: Inductive step fails, no proofs or counter example. **Note:** Relational eigeninvariants are inapplicable for nav07, nav08 since A matrix has no negative real eigenvalue (indicated by -). Some of the k-induction timings are reported as $t_1 + t_2$, indicating that an auxilliary lemma was used. t_1 is the time to prove the property, and t_2 is the time to discharge the auxilliary lemma.

where the matrix $A := [a_{11}, a_{12}; a_{21}, a_{22}]$ and the direction (b, c) are parameters that can potentially vary (for each of the cells).[2]

Every benchmark in the suite [16] is specified by fixing the matrix A, the number of cells $m \times n$, the direction (b, c) in each cell and initial intervals for each of the four state variables x, y, v_x, v_y. A distinct cell in each benchmark (marked B) represents the unsafe region. Our experiments focus on proving the unreachability of B. Further details are available elsewhere [16].

In our experiments, we verify the safety property for the navigation benchmarks using k-induction over the relational abstraction. We use the SAL infinite bounded model checker, with the k-induction flag turned on, (`sal-inf-bmc -i`), which uses the SMT solver Yices in the back-end. Table 1 reports the results. For each benchmark, we report the depth used for performing k-induction (under "depth"), the output of k-induction (under "status"), and the time it took (under "time"). There are three possible outputs: (a) the base case of k-induction

[2] The matrix A is typically Hurwitz, which means that the dynamics are such that (v_x, v_y) asymptotically converges to (b, c).

fails and a counter-example is found (denoted by "CE") (b) the base case is proved, but the induction step fails; i.e, no counter-example is found, but no proof is found either (denoted by "F") (c) the base case and the induction step are successfully proved (denoted by "P"). Since we perform k-induction on an abstraction, the counter-examples may be spurious, but the proofs are not. As Table 1 indicates, relational abstractions are sufficient to establish safety of the benchmarks nav01–nav05, nav06* and nav07. The system nav06* is the same as nav06 but with a slightly smaller set of initial states. However, the proof fails on nav06 and nav08–nav10. There are two reasons for failure: (a) Poor quality of abstraction, which is reflected in entries "CE" in Table 1. (b) Inability to find suitable k-inductive lemmas. This happens in the case of nav06, where the proof fails without yielding a counter-example. As discussed in Section 4, we employed three kinds of relational abstractions for each mode of the hybrid system: affine, eigen, and box abstractions. Table 1 also shows the performance of each of these techniques.

Affine abstractions In Table 1, Columns (2)–(4) report results on affine relational abstractions. We note that affine abstractions are sufficient to prove safety of benchmarks nav01–nav03, but they fail on all other benchmarks.

Eigen abstractions The dynamics in each mode of the benchmarks nav01–nav06 and nav10 have negative real eigenvalues. In Table 1, Columns (5)–(7) present results using a relational abstraction obtained by combining affine and eigen abstractions. For nav04–nav06, the combination eliminates the spurious counter-examples. However, no such benefit is seen on nav08–nav10 benchmarks. The dynamics in benchmarks nav07–nav08 do not have any real eigenvalues.

Box abstractions The dynamics of all modes of all benchmarks in Table 1 satisfy all the conditions for the existence of box invariants, which enables us to generate box relational invariants for each of them. Columns (8)–(10) report results using a relational abstraction obtained by combining affine, eigen and box relational invariants. In the case of nav07, where there are no eigen invariants, addition of box invariants eliminated the counter examples from the model and even enabled verification of safety using k-induction with depth 6. However, no such benefit is seen for benchmarks nav08–nav10. Also, when eigen invariants exist, then adding box invariants does not seem to improve the quality of abstraction. Note that the use of box invariants increases the time taken to perform k-induction: this is expected since box invariants have a complex Boolean structure, which increases the search space of the SMT solver.

Comparison with Other Tools: Comparing our timings with those reported in the literature for the same nav benchmarks, especially previous work by one of the authors [40], we note that our techniques are at least an order of magnitude faster on the larger benchmarks (10s of seconds vs. $100s - 1000s$ of seconds using template-based flowpipes [40]). A detailed comparison will be made available in our extended version.

Disjunctive and conjunctive relational invariants One plausible reason for the failure to prove nav08–nav10 benchmarks is that we do not consider invariants of richer Boolean structure, such as 2-disjunctive invariants of the

Type of Relational Invariant	Time (without state invariant)	Time (with state invariant)
Affine Inequality	60ms	6740ms
Affine Equality + Eigen	70ms	340ms

Table 2. Time (in milliseconds) to generate all affine equality, inequality and eigen relational invariants for all modes of all the benchmarks.

form $p(\boldsymbol{x}_0) \geq 0 \Rightarrow p(\boldsymbol{x}, \boldsymbol{x}_0) \geq 0$. Even though eigen invariants have this form, but there may be other invariants of this form that are not related to the eigenvectors of the A-matrix. We also do not consider conjunctive invariants of the form $p_1 \geq 0 \wedge p_2 \geq 0$. Note that $p_1 \geq 0$ and $p_2 \geq 0$ need not separately be inductive, but their conjunction could be inductive. For this reason, we often fail to find them by just considering templates for $p_1 \geq 0$ and $p_2 \geq 0$ separately.

Overcoming limitations of k-induction Even if the relational abstractions are strong enough to rule out all unsafe behaviors, we may still fail to prove the system safe using k-induction. This will happen if the safety property is not k-inductive for any k. This is possibly the case for benchmarks nav04-nav06. However, we are able to successfully prove safety of nav04 and nav05 by using an auxiliary lemma. The auxiliary lemma was itself verified by k-induction again. For nav06, we are unable to find any suitable auxiliary lemma at this time.

Another plausible cause for the failure of k-induction is the introduction of spurious loops in the relational abstraction, where no such loops exist in the concrete system. Analysis of the counter-examples to the induction step in nav06 (generated by `sal-inf-bmc -i -ice`) strongly indicates this possibility.

One way to eliminate spurious loops in the abstract is based on assuming that the concrete system stays in a mode for some small, but fixed, amount of time. Under the assumption that the concrete system stayed in a mode for at least 1/10 seconds, we strengthened the affine invariants of nav06, allowing us to prove safety of nav06 (for a slightly smaller set of initial states than what is specified in the nav06 benchmarks). These results are reported in row nav06* in Table 1. We conjecture that this trick will eliminate all the spurious counter-examples in the other navigation benchmarks.

Quantifier Elimination for generating relational invariants The new redlog/qepcad combination [44] is quite effective in generating all the affine and eigen invariants used in our experiments. Table 2 provides the time taken by *all* runs of the quantifier elimination process to generate these invariants. We report times for two cases depending on whether we used a template of the form $\psi[\boldsymbol{x}_0] \Rightarrow R(\boldsymbol{x}_0, \boldsymbol{x})$, with a state invariant antecedent guarding the relation. The times are negligible since the benchmarks are 4-dimensional systems (they all involve only 4 real-valued variables) with relatively simple (linear) dynamics in each mode. As a final remark, note that quantifier elimination does not return specific values for the parameters, but constraints on the unknown parameters. We choose values by solving a satisfiability problem. In our examples, the constraints after elimination were simple enough to perform this step manually. The redlog files that were used

to generate the relational invariants and SAL models of the relational abstraction are publicly available[3].

6 Conclusions

We have presented an approach for verifying hybrid systems based on constructing a relational abstraction of the hybrid system, and then using fast SMT based verification techniques, such as k-induction, for verifying the abstraction. Relational abstractions can be constructed compositionally by abstracting each mode separately. Our initial results are quite encouraging. The technique successfully solves some of the standard benchmark examples an order of magnitude faster than symbolic model checkers. Yet, the abstractions can be coarse and k-induction itself can be challenging to apply on hybrid systems in practice. Our future work will focus on improving the quality of relational abstractions while at the same time making the technique suitable for deriving fast proofs for complex systems. We also wish to apply our our techniques to non-linear hybrid systems in order to derive linear arithmetic abstractions.

References

1. A. Abate, A. Tiwari, and S. Sastry. Box invariance in biologically-inspired dynamical systems. *Automatica*, 45(7):1601–1610, July 2009.
2. R. Alur, T. Dang, and F. Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *TACAS*, volume 2619 of *LNCS*, pages 208–223. Springer, 2003.
3. R. Alur, T. A. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proc. of IEEE*, 88(7):971–984, 2000.
4. E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43:451—476, 2007.
5. J. Berdine, A. Chawdhary, B. Cook, D. Distefano, and P. W. O'Hearn. Variance analyses from invariance analyses. In *POPL*, pages 211–224. ACM, 2007.
6. M. Berz and K. Makino. Performance of Taylor model methods for validated integration of ODEs. *LNCS*, 3732:65–74, 2005.
7. F. Blanchini and S. Miani. *Set-Theoretic Methods in Control*. Springer-Verlag, 2008.
8. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5):752–794, 2003.
9. G. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In H.Brakhage, editor, *Automata Theory and Formal Languages*, volume 33 of *LNCS*, pages 134–183. Springer, 1975.
10. G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, sep 1991.
11. M. Colón, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. In *CAV*, volume 2725 of *LNCS*, pages 420–433. Springer, July 2003.

[3] `http://www.csl.sri.com/~tiwari/relational-abstraction/`

12. M. Colon and H. Sipma. Synthesis of linear ranking functions. In T. Margaria and W. Yi, editors, *Tools and Algorithms for Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 67–81. Springer, April 2001.
13. P. Cousot and R. Cousot. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Principles of Programming Languages*, pages 238–252, 1977.
14. T. Dang, O. Maler, and R. Testylier. Accurate hybridization of nonlinear systems. In *HSCC '10*, pages 11–20. ACM, 2010.
15. T. Dang and D. Salinas. Image computation for polynomial dynamical systems using the bernstein expansion. In *CAV*, volume 5643 of *LNCS*, pages 219–232. Springer, 2009.
16. A. Fehnker and F. Ivanĉić. Benchmarks for hybrid systems verification. In *HSCC*, volume 2993 of *LNCS*, pages 326–341. Springer, 2004.
17. G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. *STTT*, 10(3), June 2008.
18. A. Girard. Reachability of uncertain linear systems using zonotopes. In *HSCC*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.
19. C. L. Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250 – 262, 2010.
20. S. Gulwani, S. Jain, and E. Koskinen. Control-flow refinement and progress invariants for bound analysis. In *PLDI*, 2009.
21. S. Gulwani and A. Tiwari. Constraint-based approach for hybrid systems. In *CAV*, volume 5123 of *LNCS*, pages 190–203, 2008.
22. A. Gupta, R. Majumdar, and A. Rybalchenko. From tests to proofs. In *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 262–276. Springer, 2009.
23. N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
24. T. A. Henzinger. The theory of hybrid automata. In *LICS'96*, pages 278–292. IEEE, 1996.
25. T. A. Henzinger and P. Ho. HyTech: The Cornell hybrid technology tool. In *Hybrid Systems II*, volume 999 of *LNCS*, pages 265–293. Springer, 1995.
26. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.
27. A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *HSCC*, volume 1790 of *LNCS*, pages 202–214. Springer, 2000.
28. J. D. Meiss. *Differential Dynamical Systems*. SIAM publishers, 2007.
29. V. Mysore, C. Piazza, and B. Mishra. Algorithmic algebraic model checking II: Decidability of semi-algebraic model checking and its applications to systems biology. In *ATVA*, volume 3707 of *LNCS*, pages 217–233. Springer, 2005.
30. F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.
31. M. Oishi, I. Mitchell, A. M. Bayen, and C. J. Tomlin. Invariance-preserving abstractions of hybrid systems: Application to user interface design. *IEEE Trans. on Control Systems Technology*, 16(2), Mar 2008.
32. A. Platzer and E. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Formal Methods in Systems Design*, 35(1):98–120, 2009.
33. A. Podelski and A. Rybalchenko. Transition invariants. In *LICS*, pages 32–41. IEEE Computer Society, 2004.
34. A. Podelski and S. Wagner. Model checking of hybrid systems: From reachability towards stability. In *HSCC*, volume 3927 of *LNCS*, pages 507–521. Springer, 2006.

35. S. Prajna and A. Jadbabaie. Safety verification using barrier certificates. In *HSCC*, volume 2993 of *LNCS*, pages 477–492. Springer, 2004.
36. S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *HSCC*, volume 3414 of *LNCS*, pages 573–589. Springer, 2005.
37. J. Rushby, P. Lincoln, S. Owre, N. Shankar, and A. Tiwari. Symbolic analysis laboratory (sal). Cf. `http://www.csl.sri.com/projects/sal/`.
38. S. Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In *Hybrid Systems: Computation and Control*, pages 221–230. ACM Press, 2010.
39. S. Sankaranarayanan, T. Dang, and F. Ivančić. A policy iteration technique for time elapse over template polyhedra. In *HSCC*, volume 4981 of *LNCS*, pages 654–657. Springer, 2008.
40. S. Sankaranarayanan, T. Dang, and F. Ivančić. Symbolic model checking of hybrid systems using template polyhedra. In *TACAS*, volume 4963 of *LNCS*, pages 188–202. Springer, 2008.
41. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constraint-based linear-relations analysis. In *Static Analysis Symposium (SAS 2004)*, volume 3148 of *LNCS*, pages 53–69. Springer, August 2004.
42. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Fixed point iteration for computing the time-elapse operator. In *HSCC*, LNCS. Springer, 2006.
43. M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a sat-solver. In *FMCAD*, volume 1954 of *LNCS*, pages 108–125. Springer, 2000.
44. T. Sturm and A. Tiwari. Verification and synthesis using real quantifer elimination, 2011. Submitted.
45. A. Tiwari. Approximate reachability for linear systems. In *HSCC*, volume 2623 of *LNCS*, pages 514–525. Springer, 2003.
46. A. Tiwari. Abstractions for hybrid systems. *Formal Methods in Systems Design*, 32:57–83, 2008.
47. A. Tiwari and S. International. HybridSAL: A tool for abstracting HybridSAL specifications to SAL specifications, 2007. Cf. `http://sal.csl.sri.com/hybridsal/`.
48. V. Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. In *Applied Algebra and Error-Correcting Codes (AAECC) 8*, pages 85–101, 1997.