Implicit Semi-Algebraic Abstraction for Polynomial Dynamical Systems

Sergio Mover[1], Alessandro Cimatti[2], Alberto Griggio[2], Ahmed Irfan[3], and Stefano Tonetta[2]

[1] LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris
[2] Fondazione Bruno Kessler
[3] Stanford University

Abstract. Semi-algebraic abstraction is an approach to the safety verification problem for polynomial dynamical systems where the state space is partitioned according to the sign of a set of polynomials. Similarly to predicate abstraction for discrete systems, the number of abstract states is exponential in the number of polynomials. Hence, semi-algebraic abstraction is expensive to explicitly compute and then analyze (e.g., to prove a safety property or extract invariants).
In this paper, we propose an implicit encoding of the semi-algebraic abstraction, which avoids the explicit enumeration of the abstract states: the safety verification problem for dynamical systems is reduced to a corresponding problem for infinite-state transition systems, allowing us to reuse existing model-checking tools based on Satisfiability Modulo Theory (SMT). The main challenge we solve is to express the semi-algebraic abstraction as a first-order logic formula that is linear in the number of predicates, instead of exponential, thus letting the model checker lazily explore the exponential number of abstract states with symbolic techniques. We implemented the approach and validated experimentally its potential to prove safety for polynomial dynamical systems.

1 Introduction

Non-linear dynamical systems are characterized by continuous evolution resulting from ordinary differential equations containing non-linear polynomials. Proving safety properties for non-linear dynamical systems is extremely challenging, and several approaches have been proposed. Semi-automatic deductive verification techniques based on theorem proving include proving hybrid programs using differential dynamic logic [27] or hybrid Cyber Physical System (CPS) using Hybrid Hoare Logic (HHL) [21]). Among various automatic techniques (e.g, [31]), an important line of work applies symbolic model checking to abstractions of hybrid systems, both with linear and non-linear dynamics, using qualitative predicate abstraction ([35]). Unfortunately, the problem with the above techniques is twofold. On one side, the abstractions are often unable to precisely lift important information, thus resulting in an abstract system that is not strong enough to prove the property. On the other side, the abstraction computation may be too expensive to compute, especially in the non-linear case.

To tackle the first problem, we consider the semi-algebraic decomposition for dynamical systems of [33]. The idea is to build an abstraction from a given set of polynomials, partitioning the concrete state space according to the sign of each polynomial. The abstraction is *exact*: there is a transition from an abstract state to another abstract state if and only if there is (at least) a concrete transition from the two concretizations of the abstract states. Semi-algebraic decomposition is also appealing because it can be made *more precise* adding new polynomials.

The abstraction can be computed by means of logical operations (by repeatedly checking the satisfiability of quantifier-free formulas interpreted over the reals). However, the second problem remains: the explicit computation of the abstraction is extremely costly, since it requires the enumeration of all possible transitions between abstract states, which are exponential in the number of considered polynomials.

Interestingly, an effective use of abstraction is at the core of the most successful verification techniques for discrete infinite-state transition systems. The technique of predicate abstraction [16] was originally adapted for symbolic verification in [9] and then optimized in [19]. This idea has been further developed in *implicit predicate abstraction* [36], which eliminates the burden of an up-front exponential blowup in the computation of the abstract states by embedding the abstraction in the symbolic encoding of the transitions. This approach has been used also in combination with IC3 [5,1,6].

In this paper, we propose a new approach to the verification of dynamical systems with non-linear polynomial dynamics based on the use of semi-algebraic decomposition. The contributions of the paper are the following:

- We cast the problem of computing and verifying properties of dynamical systems using the semi-algebraic decomposition in the framework of verification via implicit predicate abstraction (i.e., a first-order logic characterization of the semi-algebraic decomposition abstraction). Thus, we apply SMT-based model checking techniques to prove safety properties of polynomial dynamical systems.
- We define a linear symbolic encoding for the abstraction. Note that the naive formulation of the predicate abstraction problem (which follows from the explicit computation approach proposed in [33]) is not effective in practice: in fact, the number of abstract states is exponential in the total number of polynomials that define the abstraction, and the encoding requires to enumerate all the possible pairs of abstract states to check the existence of an abstract transition. We exploit the properties of the LZZ formulation to define a concise encoding that is linear in the number of the polynomials, hence making the approach feasible in practice.
- We implement and experimentally evaluate the approach. The results show how the reduction to the verification of discrete infinite-state transition systems is complementary to reachability analysis techniques and proves cases that were previously out of reach for the state-of-the-art tools.

Outline: The rest of the paper is structured as follows: Sec. 2 gives an overview of the approach with a motivating example; Sec. 3 provides the background def-

initions; Sec. 4 shows the naive encoding of the abstraction, while in Sec. 5 we derive the linear encoding and define the related implicit semi-algebraic abstraction; in Sec. 6, we present the experimental results; in Sec. 7, we discuss the related work and, finally, in Sec. 8, we draw some conclusions and directions for future work.

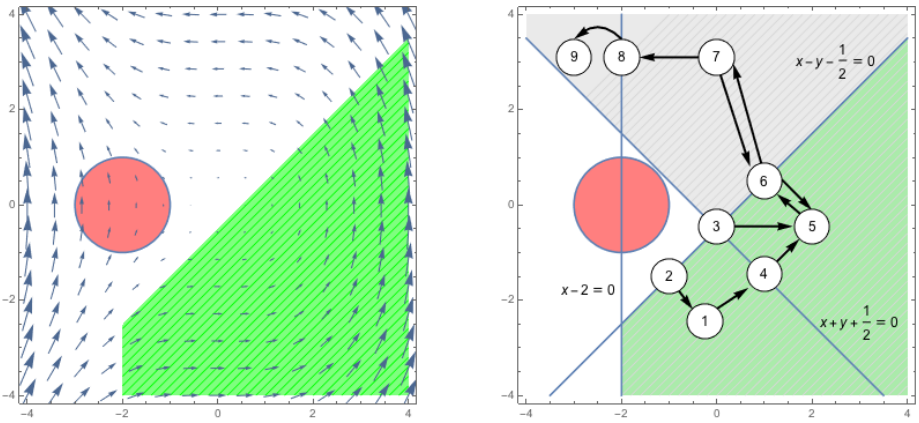
2 Overview of the approach

Consider a verification problem (adapted from [22]) on the non-linear dynamical system with two variables x and y, and differential equations $\dot{x} = -2y, \dot{y} = x^2$. We want to prove that the system cannot reach the set of bad states $(x+2)^2 + y^2 - 1 \leq 0$ (i.e., it never leaves the safe region $(x+2)^2 + y^2 - 1 > 0$) when starting from the initial set of states $x - y - \frac{1}{2} \geq 0 \wedge x + 2 > 0$. Note that although in this example the evolution of the system is not restricted, our approach can deal with the more general case in which the evolution is constrained by an invariant condition that must always hold. The system is safe and avoids the set of bad states (see system's dynamic in Figure 1a).

We can prove that the system is safe by first constructing and then model checking a discrete semi-algebraic abstraction [33]: given the set of polynomials $\mathbb{A} := \{x - y - \frac{1}{2}, x + y + \frac{1}{2}, x + 2\}$, the semi-algebraic abstraction partitions the state space according to the sign ($\{>, <, =\}$) of the polynomials in $\mathbb{A}$ (an example of abstract state is the state $x + 2 > 0 \wedge x - y - \frac{1}{2} < 0 \wedge x + y + \frac{1}{2} < 0$ represented as ① in Figure 1b). There exists a transition from an abstract state to another one if the two states are neighbors and there exists at least one trajectory of the dynamical system going from one state to the other. The existence of such condition can be checked using the *LZZ* algorithm [22], which checks if a semi-algebraic set ψ is a differential invariant for a polynomial dynamical system $\boldsymbol{f}$ when its execution is restricted to the domain H (another semi-algebraic set). The algorithm reduces the invariant check to the satisfiability of the Non-Linear Real Arithmetic Theory formula $LZZ_{\psi,\boldsymbol{f},H}(Z)$, where Z is a set of real-valued variables. We can systematically check if there exists a transition from an abstract state s_1 to the abstract state s_2 proving that s_1 *is not invariant* when restricted to the domain $s_1 \vee s_2$ (i.e., checking that $LZZ_{s_1,\boldsymbol{f},s_1 \vee s_2}(Z)$ is false).

Furthermore, we can use an algorithm, called *LazyReach* [33], to compute the forward set of reachable abstract states starting from the initial states. As usual, if no abstract states intersect the set of bad states then the system is safe, and the reachable set of abstract states is a continuous invariant for the system. Figure 1b shows the state space of the dynamical system: the initial and bad states of the verification problem (represented with the green and red region respectively), the solution of the polynomials from $\mathbb{A}$ (represented as blue lines), and further superimpose the set of reachable abstract states and transitions (represented as numbered circles and arrows between the circles). The abstraction shown in Figure 1b is the result after applying *LazyReach* to the verification problem.

A main challenge for the *LazyReach* algorithm is to explicitly enumerate the reachable states and transitions among them, since their number is exponential



(a) (b)

Fig. 1. Safety verification problem and reachable states of the abstraction for the non-linear dynamical system $\dot{x} = -2y, \dot{y} = x^2$, bad states $(x+2)^2 + y^2 - 1 \leq 0$ (red circle), and initial set of states $x - y - \frac{1}{2} \geq 0 \wedge x + 2 > 0$ (green region). Figure **(a)** shows the verification problem and the system's vector field. Figure **(b)** shows the reachable abstract states and the transitions of the algebraic abstraction (numbered circles and arrows) computed using *LazyReach* and the differential invariant (green and gray regions) obtained from the set of polynomials $\mathbb{A} = \{x - y - \frac{1}{2}, x + y + \frac{1}{2}, x + 2\}$ (blue lines), computed using *Implicit Abstraction*. Abstract states represent different combinations of signs for the abstraction's polynomials. Examples of abstract states are ① $x+2 > 0 \wedge x-y-\frac{1}{2} < 0 \wedge x+y+\frac{1}{2} < 0$, ② $x+2 > 0 \wedge x-y-\frac{1}{2} = 0 \wedge x+y+\frac{1}{2} < 0$, and ③ $x + 2 > 0 \wedge x - y - \frac{1}{2} = 0 \wedge x + y + \frac{1}{2} = 0$.

in the number of polynomials $\mathbb{A}$ (i.e., the number of total states is already $3^{|\mathbb{A}|}$). For the example above, where we have 3 polynomials, the maximum number of states would be 27, with an even bigger number of transitions (e.g., one must consider the transition between each pair of neighbouring abstract states). Even if *LazyReach* enumerates the reachable abstract states on-the-fly, the explosion in the number of states and transitions is still a bottleneck. Our implementation of *LazyReach* applied to the above example explores a total of 9 states and checks the existence of 27 transitions, taking about 12 seconds to complete.

A possible solution to tackle the state explosion problem is the *DWCL* algorithm, proposed in [33].The *DWCL* algorithm[4] tries to reduce the number of abstract states by checking if the sign of a polynomial $a \in \mathbb{A}$ is invariant, that is if:

- the sign of the polynomial a does not change in the initial states (i.e., the predicate $a \bowtie 0$, with $\bowtie \in \{<, >, =\}$, holds for all the initial states); and
- $a \bowtie 0$ is a continuous invariant for the dynamical system (this can be checked with $LZZ_{a \bowtie 0, \boldsymbol{f}, H}(Z)$).

[4] We provide the main intuition behind the *DWCL* algorithm and we refer the reader to [33] for a detailed exposition.

When a predicate $a \bowtie 0$ is a continuous invariant, the algorithm strengthens the invariant of the dynamical system (by adding $a \bowtie 0$ to the invariants), allowing to remove a from the set of polynomials $\mathbb{A}$. While the *DWCL* algorithm may already find a strong-enough invariant to prove the safety property, the algorithm falls back to the *LazyReach* algorithm in the general case to explore the abstract state space, hopefully with a strengthened invariant domain and a smaller set of polynomials. In practice, the state-space explosion problem of *LazyReach* still exists in the case "not enough" polynomials are sign-invariant, as it happens in our motivating example. In the example, no polynomials are sign-invariant[5]: this means that the *DWCL* algorithm will not remove any polynomials from the set $\mathbb{A}$ and *LazyReach* will still suffer from the state-space explosion problem.

The semi-algebraic abstraction is a specific instance of predicate abstraction [16] of the dynamical system $\boldsymbol{f}$. For discrete-state systems, there exist efficient algorithms to either *explicitly* compute the abstraction using Satisfiability Modulo Theory (SMT) solvers [20,19] or to *implicitly* represent the abstraction and directly verify a safety property (e.g., implicit predicate abstraction [36]). Since these algorithms work on a fully symbolic representation of the abstract state space, they can cope with the state-space explosion due to the number of predicates of the abstraction. However, applying the same symbolic-state techniques to compute or verify the semi-algebraic abstraction is still challenging, mainly because it requires to express the transition relation $T(X, X')$ of the semi-algebraic abstraction in a first-order logic formula. We notice that such transition relation T can be directly obtained from the abstraction's definition[6]:

$$\exists Z. \left(\bigvee_{(s_1, s_2) \in 3^{\mathbb{A}}} s_1(X) \wedge s_2(X') \wedge (\neg LZZ_{s_1, \boldsymbol{f}, s_1 \vee s_2}(Z)) \right).$$

The above transition relation enumerates all the possible pairs of abstract states and its size is exponential in the number of polynomials in $\mathbb{A}$. The additional variables Z are copies of the state variables of the system and are used to encode the LZZ condition. Clearly, even creating such formula is not scalable and hinders the application of the standard abstraction and verification techniques used for discrete systems. Note that, while the LZZ algorithm works for semi-algebraic sets (i.e., the candidate invariant ψ and the invariant states H are both arbitrary Boolean combinations of non-linear arithmetic terms), here we apply LZZ to check the existence of a transition between two abstract states, hence we still have to explicitly enumerate the abstract transitions.

Our main contribution, presented in Section 5, is a compact formulation of the above transition relation that has a size *linear* in the number of the polynomials $\mathbb{A}$. The steps to obtain such exponentially smaller transition are:

[5] The differential-cut (DC) and the differential divide-and-conquer (DDC) proof rules used in *DWCL* fail for all the polynomials from $\mathbb{A}$, so *DWCL* would not remove any polynomial.

[6] For clarity, here we do not include additional constraints in the transition relation, such as the neighborhood relation, which instead we consider later in Section 4.

1. We specialize the LZZ formula $\neg LZZ_{s_1,\boldsymbol{f},s_1 \vee s_2}(Z)$ to encode the existence of a transition between two abstract states s_1 and s_2. The resulting formula is a disjunction, and each disjunct encodes the necessary and sufficient condition for a continuous transition to s_2 to exist, either *inside* the set $s_1(Z)$ or *outside* the set $\neg s_1(Z)$. Intuitively, we obtain a specific encoding for checking the existence of an abstract transition, instead of reusing the LZZ as a "black box".
2. We "lift" the above disjunction to the disjunction of all the abstract states, obtaining the formula:

$$\exists Z.(\mathit{InsExpl}_{\boldsymbol{f}}(X, X', Z) \vee \mathit{OutExpl}_{\boldsymbol{f}}(X, X', Z)),$$

   where $\mathit{InsExpl}_{\boldsymbol{f}}(X, X', Z)$ encodes the "inside condition" for all the pairs of transitions (and similarly for the "outside condition" $\mathit{OutExpl}_{\boldsymbol{f}}(X, X', Z)$).
3. The formula $\mathit{InsExpl}_{\boldsymbol{f}}(X, X', Z)$ still contains an explicit enumeration on the pairs of abstract states. We show how we obtain an equivalent formula, $\mathit{InsSymb}_{\boldsymbol{f}}(X, X', Z)$, that encodes the same condition for each polynomial $a \in \mathbb{A}$ in the abstraction, obtaining a linear, instead of exponential, encoding. We apply the same reasoning on $\mathit{OutExpl}_{\boldsymbol{f}}(X, X', Z)$.

We then use the concise transition relation of T to obtain a symbolic transition system $S_{Impl,\mathbb{P}}$ that implicitly encodes the semi-algebraic abstraction for the dynamical system $\boldsymbol{f}$ with the polynomials $\mathbb{A}$ and predicates $\mathbb{P} = \{a \bowtie 0 \mid a \in \mathbb{A} \wedge \ \bowtie \in \{>, <, =\}\}$. Technically, instead of computing the predicate abstraction, we encode an implicit abstraction [36]. Consequently, we avoid the expensive quantifier elimination step. We can then verify the safety property on the transition system $S_{Impl,\mathbb{P}}$ using an SMT-based model checking algorithm. We use the algorithm from [4], since $S_{Impl,\mathbb{P}}$ contains non-linear arithmetic formulas. Our approach verifies the example of Figure 1 and finds the continuous invariant:

$$(x - y < \frac{1}{2} \vee x \geq -2) \wedge (x - y \geq \frac{1}{2} \vee x + y \geq -\frac{1}{2}) \wedge (x - y \geq \frac{1}{2} \vee x + y > -\frac{1}{2}),$$

which is shown in the union of the green and gray regions in Figure 1b.

3 Preliminaries

In this work, we consider first order logic formulas in the theory of non-linear arithmetic over the reals (NRA). We denote with $\phi(X)$ the formula ϕ containing free variables from the set $X = \{x_1, \ldots, x_n\}$. We simplify the notation of the formula $\phi(X)$ to ϕ when the set X is clear from the context.

Invariant Verification for Polynomial Dynamical Systems

Safety Verification of Dynamical Systems. Given a set of variables X we write $\boldsymbol{X} = [x_1, \ldots, x_n]^T$ to specify a vector containing all the variables in X ordered lexicographically. We use the subscript $\boldsymbol{X}_i$ to access to the i-th element of the

vector. We focus on *polynomial dynamical systems* of ordinary differential equations (ODEs) $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X})$, where $\dot{\boldsymbol{X}}$ is the vector of first-order derivatives of the variables $\boldsymbol{X}$ and $\boldsymbol{f}(\boldsymbol{X})$ is a vector of polynomials (i.e., $\boldsymbol{f}_i(\boldsymbol{X})$ is a polynomial). The *safety verification problem* consists of proving that every trajectory of the dynamical system $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X})$ starting inside the initial set of states ψ and while being inside the evolution domain constraints H remains inside the safe set of states ϕ. We write the problem using the *differential dynamic logic* [28] formula:

$$\psi \rightarrow [\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}) \ \& \ H]\phi, \tag{1}$$

asserting that if the system is in a state satisfying the pre-condition ϕ (the initial states) this implies ($\rightarrow$ operator) that all the trajectories evolving according to the ODE $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X})$ and evolution domain H (box modality []) will satisfy the post-condition ϕ (the safe states). Formally, the system is safe if:

$$\forall \boldsymbol{x}_0 \in \psi. \forall \tau \geq 0. \forall t \in [0, \tau]. ((\varphi(\boldsymbol{x}_0, t) \in H) \rightarrow \varphi(\boldsymbol{x}_0, t) \in \phi),$$

were the differentiable function $\varphi : \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n$, such that $\frac{d}{dt}(\varphi(\boldsymbol{x}_0, t)) = \boldsymbol{f}(\varphi(\boldsymbol{x}_0, t))$, is the solution to the initial value problem $\boldsymbol{x}_0 \in \mathbb{R}^n$ (i.e., $\varphi(\boldsymbol{x}_0, t)$ describes the state the dynamical system $\boldsymbol{f}$ reaches after $t \in \mathbb{R}$ time when starting in the initial state $\boldsymbol{x}_0$).
The problem of proving the system is safe can be reduced to find a formula $\theta(X)$ such that: i) $H \wedge \psi \rightarrow \theta$, ii) $\theta \rightarrow [\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}) \ \& \ H] \ \theta$, and iii) $\theta \rightarrow \phi$. $\theta(X)$ is a *continuous invariant* [29] that contains the initial states and that is contained in the safe states.

LZZ Algorithm [22]. The LZZ algorithm reduces the problem of checking if θ is a continuous invariant to checking the validity of the following formula:

$$\begin{aligned} LZZ_{\theta,\boldsymbol{f},H}(X) \doteq ((\theta(X) \wedge H(X) \wedge In_{\boldsymbol{f},H}(X)) \rightarrow In_{\boldsymbol{f},\theta}(X)) \wedge \\ ((\neg\theta(X) \wedge H(X) \wedge In_{-\boldsymbol{f},H}(X)) \rightarrow \neg In_{-\boldsymbol{f},\theta}(X)), \end{aligned} \tag{2}$$

where the formula $In_{\boldsymbol{f},\gamma}(X)$ for the ODEs $\boldsymbol{f}$ and the formula γ represents the set of states which will evolve inside the set γ for some non-zero time in the future. Respectively, the formula $In_{-\boldsymbol{f},\gamma}(X)$ represents the set of states evolved inside the set γ for some non-zero time in the past, and $-\boldsymbol{f}$ represents the dynamical system evolving in "reverse". Note that the construction of the formula $In_{\boldsymbol{f},\gamma}(X)$ assumes γ to be in disjunctive normal form (DNF):

$$\gamma = \bigvee_{d \in disj(\gamma)} \bigwedge_{a \bowtie 0 \in pred(d)} a(X) \bowtie 0,$$

where $disj(\gamma)$ are the disjuncts of a formula γ, $pred(d)$ are the predicates in the disjunct d, and $\bowtie \in \{>, \geq\}$[7]. The formula $In_{\boldsymbol{f},\gamma}(X)$ is defined as:

$$In_{\boldsymbol{f},\gamma}(X) = \bigvee_{d \in disj(\gamma)} \bigwedge_{a \bowtie 0 \in pred(d)} In_{\boldsymbol{f},a\bowtie 0}(X). \tag{3}$$

[7] Later we also consider equalities (i.e., predicates of the form $a = 0$). The construction of $In_{\boldsymbol{f},a=0}(X)$ can be found in [12].

The formula $In_{\boldsymbol{f},a\bowtie 0}(X)$ encodes the set for a single predicate $a \bowtie 0$ using the Lie derivatives of the polynomial $a(X)$. The i-th *Lie derivative* $L^i_{\boldsymbol{f}}a$ of a polynomial $a(X)$ with respect to the ODEs $\boldsymbol{f}$ is defined recursively as:

$$L^{(0)}_{\boldsymbol{f}}a \doteq a, \qquad L^{(i)}_{\boldsymbol{f}}a \doteq \frac{\partial}{\partial \boldsymbol{X}} L^{(i-1)}_{\boldsymbol{f}}a\boldsymbol{f}.$$

$In_{\boldsymbol{f},a>0}(X)$ encodes that the first non-zero Lie derivative of a must be positive in order for the trajectories of the system to enter the set $a > 0$ and stay inside the set for a positive time[8](see [22] and [12] for a thorough explanation):

$$In_{\boldsymbol{f},a>0}(X) \doteq \bigvee_{0\leq i\leq N_{a,f}} \left(\bigwedge_{0\leq j<i} L^{(j)}_{\boldsymbol{f}}a = 0 \wedge L^{(i)}_{\boldsymbol{f}}a > 0 \right), \tag{4}$$

$$In_{\boldsymbol{f},a\geq 0}(X) \doteq In_{\boldsymbol{f},a>0}(X) \vee \bigvee_{0\leq i\leq N_{a,f}} L^{(i)}_{\boldsymbol{f}}a = 0, \tag{5}$$

where $N_{a,\boldsymbol{f}}$ is an integer constant and is an upper bound on the minimum integer number r (called *rank*) such that $L^{(r)}_{\boldsymbol{f}}a \neq 0$ (for all $x \in \mathbb{R}^n$). $N_{a,\boldsymbol{f}}$ can be computed using Gröbner basis as explained in [22].

In the following, we will only use the fact that the formula $In_{\boldsymbol{f},\gamma}(X)$ for the DNF formula γ is the DNF formula where In_f is applied to the predicates (as shown in Formula (3)).

Semi-Algebraic Abstraction [33]. The *semi-algebraic abstraction* of the dynamical system $\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X})$ partitions its state space with respect to a set of polynomials $\mathbb{A} \doteq \{a_1, \ldots, a_m\}$. The abstraction is the (explicit state) transition system $S_{\mathbb{A}} \doteq \langle 3^{\mathbb{A}}, I_{f,\mathbb{A}}, T_{f,\mathbb{A}} \rangle$ where:

- $3^{\mathbb{A}} \doteq \{s = \bigwedge_{a\in\mathbb{A}} a \bowtie 0 \mid \bowtie \in \{>, <, =\}\}$ is the set of abstract states;
- $I_{f,\mathbb{A}} \doteq \{s \in 3^{\mathbb{A}} \mid s \wedge \psi \text{ is satisfiable}\}$ is the set of abstract initial states; and
- $T_{f,\mathbb{A}} \subseteq 3^{\mathbb{A}} \times 3^{\mathbb{A}}$ is the abstract transition relation. A transition $(s_1, s_2) \in T_{f,\mathbb{A}}$ if:
  - s_1 is an abstract state *adjacent* to s_2. The abstraction exploits the continuity assumption on $\boldsymbol{f}$ and does not allow the system to transition directly from a state where a predicate is greater than 0 (e.g., $a > 0$) to a state where the same predicate is less than 0 (e.g., $a < 0$), and vice-versa. The abstraction does not visit two abstract states containing predicates with opposite signs, forcing instead to visit the intermediate state where the predicate is equal to 0.
  - There exists a continuous trajectory from s_1 to s_2. This condition corresponds to checking that the following differential dynamic logic formula is *not valid* (i.e. s_1 is not a differential invariant when restricting the evolution domain to $s_1 \vee s_2$):

$$s_1 \rightarrow [\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}) \;\&\; s_1 \vee s_2]s_1,$$

[8] In our implementation we encode $In_{\boldsymbol{f},a>0}(X)$ using the remainders of the Lie derivative, as in [12].

which can be checked using the sound and complete LZZ algorithm, i.e. checking the satisfiability of the first-order formula $\neg LZZ_{s_1,\boldsymbol{f},s_1 \vee s_2}(Z)$).

Since the number of states $3^{\mathbb{A}}$ is finite we can compute the set of reachable states. The concretization of this set, θ contains the initial states and is a differential invariant. If θ further implies the safe states ψ, then we prove the safety verification problem 1. However, the computation of the abstract transition relation is exponential in the number of polynomials in $\mathbb{A}$ because we would need to enumerate all the possible pairs of transitions $(s_1, s_2) \in 3^{\mathbb{A}} \times 3^{\mathbb{A}}$.

Predicate Abstraction.

A *symbolic transition system* S is a tuple $S \doteq \langle V, I, T \rangle$, where V is a set of (state) variables, $I(V)$ is a formula representing the initial states, and $T(V, V')$ is a formula representing the transition relation. A *state* s of S is an interpretation of the state variables V. A (finite) *path* π of S is a finite sequence $\pi \doteq s_0, s_1, \ldots, s_k$ of states with the same domain and interpretation of symbols in the signature Σ such that $s_0 \models I$ and for all i, $0 \leq i < k$, $s_i, s'_{i+1} \models T$. We say that a state s is *reachable* in S iff there exists a path of S ending in s. Given a formula $P(V)$ and a transition system S, the *invariant verification problem*, denoted with $S \models P$, checks if for all the finite paths $s_0, s_1, \ldots, s_k$ of S, for all i, $0 \leq i \leq k$, $s_i \models P$.

Predicate Abstraction [16] partitions the concrete system $S = \langle V, I, T \rangle$ according to a finite set of predicates $\mathbb{P} \doteq \{p_1, \ldots, p_k\}$ in a finite symbolic transition system:

$$\widehat{S}_{\mathbb{P}} = \langle V_{\mathbb{P}}, \widehat{I}_{\mathbb{P}}(V_{\mathbb{P}}), \widehat{T}_{\mathbb{P}}(V_{\mathbb{P}}, V'_{\mathbb{P}}) \rangle$$

using a new abstract Boolean variable v_p for each predicate p ($V_{\mathbb{P}} = \{v_p \mid v \in V\}$ is the set of those new variables). The abstraction relation $H_{\mathbb{P}}(V, V_{\mathbb{P}}) \doteq \bigwedge_{p \in \mathbb{P}} v_p \leftrightarrow p(V)$ defines how a set of concrete states is abstracted to the abstract states. We compute the abstraction of a formula $\psi(V)$ by existentially quantifying the concrete variables V:

$$\widehat{\psi}_{\mathbb{P}}(V_{\mathbb{P}}) \doteq \exists V.(\psi(V) \wedge H_{\mathbb{P}}(V, V_{\mathbb{P}})).$$

Similarly, we compute the abstract transition relation for $T(V, V')$:

$$\widehat{T}_{\mathbb{P}}(V_{\mathbb{P}}, V'_{\mathbb{P}}) \doteq \exists V, V'.(T(V, V') \wedge H_{\mathbb{P}}(V, V_{\mathbb{P}}) \wedge H_{\mathbb{P}}(V', V'_{\mathbb{P}})).$$

The above formulation is sufficient to compute the predicate abstraction for an infinite-state transition system $S = \langle V, I, T \rangle$ and a set of predicates $\mathbb{P}$. However, the main challenge in computing the abstraction is to eliminate the quantifiers, since quantifier elimination is expensive to compute.

Implicit Predicate Abstraction. Implicit Predicate Abstraction [36] is a model checking algorithm that avoids computing the abstract version of the initial states, safety property, and transition relation, instead it encodes the existence of a path in the abstract system. It exploits the fact that the abstraction induces

an equivalence relation among concrete states of the system (i.e., two concrete states are equivalent if they belong to the same abstract state) and that this relation can be expressed as a quantifier free formula:

$$EQ_{\mathbb{P}}(V, \overline{V}) \doteq \bigwedge_{p \in \mathbb{P}} p(V) \leftrightarrow p(\overline{V}). \tag{6}$$

We use the equivalence $EQ_{\mathbb{P}}(V, \overline{V})$ to relate two sets of concrete states and we encode the problem of reaching a set of target states $\neg P$ in k steps of the transition system S as follows:

$$\begin{aligned} BMC^{k}_{\mathbb{P}} \doteq\ & I(V^0) \wedge EQ_{\mathbb{P}}(V^0, \overline{V}^0) \wedge \\ & \bigwedge_{1 \le h < k} \Big(T(\overline{V}^{h-1}, V^h) \wedge EQ_{\mathbb{P}}(V^h, \overline{V}^h) \Big) \wedge T(\overline{V}^{k-1}, V^k) \wedge \\ & EQ_{\mathbb{P}}(V^k, \overline{V}^k) \wedge (\neg P(\overline{V}^k)). \end{aligned}$$

The formula $BMC^{k}_{\mathbb{P}}$ is satisfiable iff there exists a path in the abstract transition system $\widehat{S}_{\mathbb{P}}$ of length k starting from the (abstracted) initial states $\widehat{I}_{\mathbb{P}}(V_{\mathbb{P}})$ and reaching the (abstracted) bad states $\widehat{\neg P}_{\mathbb{P}}(V_{\mathbb{P}})$.

4 Explicit Computation of the Semi-Algebraic Abstraction

We frame the problem of computing the semi-algebraic abstraction as a predicate abstraction problem. This formulation allows us to use the standard techniques to compute or analyze the predicate abstraction for discrete systems.

We consider the invariant verification problem $\psi \rightarrow [\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X})\ \&\ H]\phi$ as in Equation (1) and a set of polynomials $\mathbb{A} = \{a_1, \ldots, a_m\}$ for the abstraction. We construct a symbolic transition system of the semi-algebraic abstraction:

$$\widehat{S}_{\mathbb{P}} \doteq \langle V_{\mathbb{P}}, \widehat{I}_{\mathbb{P}}(V_{\mathbb{P}}), \widehat{T}_{\mathbb{P}}(V_{\mathbb{P}}, V'_{\mathbb{P}}) \rangle,$$

where the set of predicates of the abstraction is $\mathbb{P} = \{a \bowtie 0 \mid a \in \mathbb{A} \wedge\ \bowtie \in \{>, <, =\}\}$, and the set of abstract variables $V_{\mathbb{P}}$ is defined as in Section 3 (i.e., the abstraction contains a Boolean variable v_p for each predicates $p \in \mathbb{P}$). We similarly use the formula $H_{\mathbb{P}}(X, V_{\mathbb{P}})$ to describe the equivalence relation of the concrete states. The formulas $\widehat{I}_{\mathbb{P}}(V_{\mathbb{P}})$ and $\widehat{\neg P}_{\mathbb{P}}(V_{\mathbb{P}})$ are the semi-algebraic abstraction of the initial states ψ and of the unsafe states $\neg\phi$:

$$\widehat{I}_{\mathbb{P}}(V_{\mathbb{P}}) \doteq \exists X.(\psi(X) \wedge H_{\mathbb{P}}(X, V_{\mathbb{P}})), \qquad \widehat{\neg P}_{\mathbb{P}}(V_{\mathbb{P}}) \doteq \exists X.(\neg\phi(X) \wedge H_{\mathbb{P}}(X, V_{\mathbb{P}})),$$

and we obtain the abstraction by existentially quantifying the concrete variables X. The definition of the abstract transition relation $\widehat{T}_{\mathbb{P}}(V_{\mathbb{P}}, V'_{\mathbb{P}})$, which differs

from the encoding of the semi-algebraic decomposition, is:

$$\widehat{T}_{\mathbb{P}}(V_{\mathbb{P}}, V'_{\mathbb{P}}) \doteq \exists X, X'. \Big(N(X, X') \wedge H(X) \wedge H(X') \wedge \\ H_{\mathbb{P}}(X, V_{\mathbb{P}}) \wedge H_{\mathbb{P}}(X', V'_{\mathbb{P}}) \wedge \exists Z. T_{\mathbb{A}}(X, X', Z) \Big), \tag{7}$$

where $N(X, X')$ encodes the adjacent relation between abstract states:

$$N(X, X') = \bigwedge_{a \in \mathbb{A}} \Big((a(X) < 0 \rightarrow a(X') \leq 0) \wedge (a(X) > 0 \rightarrow a(X') \geq 0) \Big),$$

and $T_{\mathbb{A}}(X, X', Z)$ encodes the existence of a transition in the dynamical system $\boldsymbol{f}$ for each pair of abstract states $(s_1, s_2) \in 3^{\mathbb{A}}$:

$$T_{\mathbb{A}}(X, X', Z) \doteq \bigvee_{(s_1, s_2) \in 3^{\mathbb{A}}} \Big(s_1(X) \wedge s_2(X') \wedge \neg LZZ_{s_1, \boldsymbol{f}, s_1 \vee s_2}(Z) \Big). \tag{8}$$

Theorem 1. *The transition systems $S_{\mathbb{A}}$ and $\widehat{S}_{\mathbb{P}}$ are bisimilar.*

Corollary 1. *$\widehat{S}_{\mathbb{P}} \models \widehat{\neg\neg P}_{\mathbb{P}}(V_{\mathbb{P}})$ implies $\psi \rightarrow [\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}) \ \& \ H]\phi$.*

Proof (sketch). The proof follows directly from Theorem 1. □

While the encoding of the transition relation $\widehat{T}_{\mathbb{P}}(V_{\mathbb{P}}, V'_{\mathbb{P}})$ is symbolic, it (and in particular the sub-formula $T_{\mathbb{A}}(X, X', Z)$) explicitly enumerates an exponential number of abstract pairs of states. Clearly, this encoding is not practical and defeats the purpose of using symbolic techniques to compute the abstraction.

5 Linear Encoding of the Semi-Algebraic Abstraction

Specializing the LZZ formula for checking abstract transitions

The construction of the semi-algebraic abstraction uses the formula $\neg LZZ_{s_1, \boldsymbol{f}, s_1 \vee s_2}(Z)$ to encode the existence of a transition from the abstract state s_1 to the abstract state s_2. We observe that here the LZZ algorithm is applied to formulas with a specific structure – the abstract states $s_1(Z)$ and $s_2(Z)$, in contrast to arbitrary semi-algebraic sets as in the general case of $LZZ_{\theta, \boldsymbol{f}, H}(X)$ where the formulas θ and H are in DNF. Instead, in the case of $LZZ_{s_1, \boldsymbol{f}, s_1 \vee s_2}(Z)$, each abstract state $s_i(X)$ assigns a sign to each polynomial $a \in \mathbb{A}$ and is represented as conjunctions of predicates $s_i = a_1 \bowtie_1 0 \wedge a_2 \bowtie_2 0 \wedge \ldots a_m \bowtie_m 0$, where $\bowtie_j \in \{>, <, =\}$. We will write the conjunction representing a state $s_i(X)$ as $\bigwedge_{a \bowtie 0 \in s_i} a(X) \bowtie 0$. Also note that the evolution domain constraints are also a disjunction of two abstract states $s_1 \vee s_2$.

We specialize Eq. (2) to the specific case of $LZZ_{s_1,\boldsymbol{f},s_1\vee s_2}(Z)$. We will use such specialization to obtain a compact (linear in the number of polynomials) encoding later in the section. Instantiating the formula (2) to the case of $LZZ_{s_1,\boldsymbol{f},s_1\vee s_2}(Z)$, we get:

$$
\begin{aligned}
&LZZ_{s_1,\boldsymbol{f},s_1\vee s_2}(Z) \doteq \\
&\quad ((s_1(Z) \wedge (s_1(Z) \vee s_2(Z)) \wedge In_{\boldsymbol{f},s_1\vee s_2}(Z)) \rightarrow In_{\boldsymbol{f},s_1}(Z)) \wedge \qquad (9)\\
&\quad ((\neg s_1(Z) \wedge (s_1(Z) \vee s_2(Z)) \wedge In_{-\boldsymbol{f},s_1\vee s_2}(Z)) \rightarrow \neg In_{-\boldsymbol{f},s_1}(Z))\\
&\text{Applying the Boolean identities: } (\alpha \wedge (\alpha \vee \beta)) \leftrightarrow \alpha, \quad (\neg\alpha \wedge (\alpha \vee \beta)) \leftrightarrow \neg\alpha \wedge \beta\\
&\Longleftrightarrow ((s_1(Z) \wedge In_{\boldsymbol{f},s_1\vee s_2}(Z)) \rightarrow In_{\boldsymbol{f},s_1}(Z)) \wedge \qquad (10)\\
&\quad ((\neg s_1(Z) \wedge s_2(Z) \wedge In_{-\boldsymbol{f},s_1\vee s_2}(Z)) \rightarrow \neg In_{-\boldsymbol{f},s_1}(Z))\\
&\text{Rewriting the implication and applying De Morgan's laws:}\\
&\Longleftrightarrow (\neg s_1(Z) \vee \neg In_{\boldsymbol{f},s_1\vee s_2}(Z) \vee In_{\boldsymbol{f},s_1}(Z)) \wedge \qquad (11)\\
&\quad (s_1(Z) \vee \neg s_2(Z) \vee \neg In_{-\boldsymbol{f},s_1\vee s_2}(Z) \vee \neg In_{-\boldsymbol{f},s_1}(Z))\\
&\text{Expanding the definition of } In \text{ (Eq. (3)): } In_{\boldsymbol{f},\alpha\vee\beta} \doteq (In_{-\boldsymbol{f},\alpha} \vee In_{\boldsymbol{f},\beta})\\
&\qquad\qquad\qquad\qquad\qquad\qquad In_{-\boldsymbol{f},\alpha\vee\beta} \doteq (In_{-\boldsymbol{f},\alpha} \vee In_{-\boldsymbol{f},\beta})\\
&\Longleftrightarrow (\neg s_1(Z) \vee \neg(In_{\boldsymbol{f},s_1}(Z) \vee In_{\boldsymbol{f},s_2}(Z)) \vee In_{\boldsymbol{f},s_1}(Z)) \wedge \qquad (12)\\
&\quad (s_1(Z) \vee \neg s_2(Z) \vee \neg(In_{-\boldsymbol{f},s_1}(Z) \vee In_{-\boldsymbol{f},s_2}(Z)) \vee \neg In_{-\boldsymbol{f},s_1}(Z))\\
&\text{Applying the Boolean identities: } (\neg(\alpha \vee \beta) \vee \alpha) \leftrightarrow (\neg\beta \vee \alpha), (\neg(\alpha \vee \beta) \vee \neg\alpha) \leftrightarrow \neg\alpha\\
&\Longleftrightarrow (\neg s_1(Z) \vee \neg In_{\boldsymbol{f},s_2}(Z) \vee In_{\boldsymbol{f},s_1}(Z)) \wedge \qquad (13)\\
&\quad (s_1(Z) \vee \neg s_2(Z) \vee \neg In_{-\boldsymbol{f},s_1}(Z)).
\end{aligned}
$$

Note that, while In does not distribute over arbitrary Boolean formulas (see [12]), when we expand the definition of $In_{\boldsymbol{f},s_1\vee s_2}$ (Eq. (12)), the formula $s_1 \vee s_2$ is in DNF. Thus, Formula (13) is equivalent to the initial Formula (9) of $LZZ_{s_1,\boldsymbol{f},s_1\vee s_2}(Z)$. We then write the negation of the Formula 13 as:

$$
\begin{aligned}
\neg LZZ_{s_1,\boldsymbol{f},s_1\vee s_2}(Z) \doteq\ & (s_1(Z) \wedge In_{\boldsymbol{f},s_2}(Z) \wedge \neg In_{\boldsymbol{f},s_1}(Z)) \vee \qquad (14)\\
& (\neg s_1(Z) \wedge s_2(Z) \wedge In_{-\boldsymbol{f},s_1}(Z)).
\end{aligned}
$$

Linear Encoding of the Semi-Algebraic Transition Relation

In the following steps, we revise the formula $T_{\mathbb{A}}(X, X', Z)$ that encodes the existence of the transitions in the abstraction, still enumerating all possible pairs of states, using the specialized LZZ encoding from Eq. (14). We substitute the subformula $\neg LZZ_{s_1,\boldsymbol{f},s_1\vee s_2}(Z)$ with the specialized LZZ encoding (Eq. (16)); we then distribute the conjunction $s_1(X) \wedge s_2(X')$ over the disjunction present in the definition of $\neg LZZ_{s_1,\boldsymbol{f},s_1\vee s_2}(Z)$ (Eq. (17)), and then over possible pairs of states (Eq. (18)). We rename the two disjuncts in Eq. (18) as $InsExpl_{\boldsymbol{f}}(X, X', Z)$ and $OutExpl_{\boldsymbol{f}}(X, X', Z))$ (Eq. (19)). The formulas $InsExpl_{\boldsymbol{f}}(X, X', Z)$ and $OutExpl_{\boldsymbol{f}}(X, X', Z))$ still enumerate explicitly the abstract states. However, each of these formulas is

a conjunction of predicates, application of the $In_{\boldsymbol{f}}$ operator to a conjunction of predicates, and negations of the application of $In_{\boldsymbol{f}}$.

$$
\begin{aligned}
&T_{\mathbb{A}}(X, X', Z) \doteq \\
&\quad \exists Z. \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} (s_1(X) \wedge s_2(X') \wedge \neg LZZ_{s_1,\boldsymbol{f},s_1\vee s_2}(Z)) && (15)\\
&\iff \exists Z. \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} \begin{pmatrix} s_1(X)\wedge s_2(X')\wedge((s_1(Z)\wedge In_{\boldsymbol{f},s_2}(Z)\wedge\neg In_{\boldsymbol{f},s_1}(Z))\vee \\ (\neg s_1(Z)\wedge s_2(Z)\wedge In_{-\boldsymbol{f},s_1}(Z))) \end{pmatrix} && (16)\\
&\iff \exists Z. \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} \begin{pmatrix} (s_1(X)\wedge s_2(X')\wedge s_1(Z)\wedge In_{\boldsymbol{f},s_2}(Z)\wedge\neg In_{\boldsymbol{f},s_1}(Z))\vee \\ ((s_1(X)\wedge s_2(X')\wedge\neg s_1(Z)\wedge s_2(Z)\wedge In_{-\boldsymbol{f},s_1}(Z)) \end{pmatrix} && (17)\\
&\iff \exists Z. \begin{pmatrix} \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} (s_1(X)\wedge s_2(X')\wedge s_1\wedge In_{\boldsymbol{f},s_2}(Z)\wedge\neg In_{\boldsymbol{f},s_1}(Z))\vee \\ \bigvee_{(s_1,s_2)\in 3^{\mathbb{A}}} (s_1(X)\wedge s_2(X')\wedge\neg s_1\wedge s_2\wedge In_{-\boldsymbol{f},s_1}(Z)) \end{pmatrix} && (18)\\
&\iff \exists Z. (InsExpl_{\boldsymbol{f}}(X, X', Z) \vee OutExpl_{\boldsymbol{f}}(X, X', Z)). && (19)
\end{aligned}
$$

We now show how we obtain a formula $InsExpl_{\boldsymbol{f}}(X, X', Z)$ with a linear size. We expand the definition of the formula $InsExpl_{\boldsymbol{f}}(X, X', Z)$ with respect to the predicates in s_1 and s_2. Recall that each abstract state is a conjunction of predicates obtained from the set of polynomial $\mathbb{A}$ (i.e., $s \doteq \bigwedge_{a\in\mathbb{A}} a \bowtie_a 0$, $\bowtie_a \in \{>, <, =\}$) and that we use $a \bowtie 0 \in s$ to enumerate the predicates in s.

$$
\begin{aligned}
InsExpl_{\boldsymbol{f}}(X, X', Z) \doteq \bigvee_{s_1,s_2\in 3^{\mathbb{A}}} \Bigg(& \bigwedge_{a\bowtie 0\in s_1} a(X) \bowtie 0 \wedge \bigwedge_{a\bowtie 0\in s_2} a(X') \bowtie 0 \wedge \\
& \bigwedge_{a\bowtie 0\in s_1} a(Z) \bowtie 0 \wedge \bigwedge_{a\bowtie 0\in s_2} In_{\boldsymbol{f},a\bowtie 0}(Z) \wedge \\
& \bigvee_{a\bowtie 0\in s_1} \neg In_{\boldsymbol{f},a\bowtie 0}(Z) \Bigg). && (20)
\end{aligned}
$$

In the above formula, we used De Morgan rules to rewrite the formula $\neg \bigwedge_{a\bowtie 0\in s_1} In_{\boldsymbol{f},a\bowtie 0}(Z)$ as the formula $\bigvee_{a\bowtie 0\in s_1} \neg In_{\boldsymbol{f},a\bowtie 0}(Z)$. We express the formula $InsExpl_{\boldsymbol{f}}(X, X', Z)$ as an enumeration of the predicates, over the variables X and X', determining the abstract states s_1 and s_2, instead of the pairs of abstract states:

$$
\begin{aligned}
InsSymb_{\boldsymbol{f}}(X, X', Z) \doteq & \bigwedge_{a\in\mathbb{A},\bowtie\in\{>,<,=\}} \Big(a(X) \bowtie 0 \rightarrow a(Z) \bowtie 0\Big) \wedge \\
& \bigwedge_{a\in\mathbb{A},\bowtie\in\{>,<,=\}} \Big(a(X') \bowtie 0 \rightarrow In_{\boldsymbol{f},a\bowtie 0}(Z)\Big) \wedge \\
& \bigvee_{a\in\mathbb{A},\bowtie\in\{>,<,=\}} \Big(a(X) \bowtie 0 \wedge (\neg In_{\boldsymbol{f},a\bowtie 0}(Z))\Big). && (21)
\end{aligned}
$$

Lemma 1. *$InsExpl_{\boldsymbol{f}}(X, X', Z)$ and $InsSymb_{\boldsymbol{f}}(X, X', Z)$ are equivalent.*

Proof (sketch).
⇒) We show that $\mu \models \mathit{InsExpl}_{\boldsymbol{f}}(X, X', Z)$ implies $\mu \models \mathit{InsSymb}_{\boldsymbol{f}}(X, X', Z)$. Since $\mu \models \mathit{InsExpl}_{\boldsymbol{f}}(X, X', Z)$ we have that μ is an interpretation for one of the disjuncts on the possible pairs of states of $\mathit{InsExpl}_{\boldsymbol{f}}(X, X', Z)$:

$$\bigwedge_{a \bowtie 0 \in s_1} a(X) \bowtie 0 \wedge \bigwedge_{a \bowtie 0 \in s_2} a(X') \bowtie 0 \wedge \bigwedge_{a \bowtie 0 \in s_1} a(Z) \bowtie 0 \wedge$$
$$\bigwedge_{a \bowtie 0 \in s_2} \mathit{In}_{\boldsymbol{f}, a \bowtie 0}(Z) \wedge \bigvee_{a \bowtie 0 \in s_1} \neg \mathit{In}_{\boldsymbol{f}, a \bowtie 0}(Z).$$

Hence, there exist two (and exactly two) abstract states s_1, s_2, such that $\mu \models s_1(X)$ and $\mu \models s_2(X')$. This means that any predicate $a \bowtie 0 \notin s_1$ is such that $\mu \not\models a \bowtie (X)$ and similarly for predicates not in the state s_2 for the variables X' (recall that, given a polynomial $a \in \mathbb{A}$, the possible abstraction predicates $a > 0$, $a < 0$, and $a = 0$ are mutually exclusive). We show that μ is an interpretation for all the conjuncts in $\mathit{InsSymb}_{\boldsymbol{f}}(X, X', Z)$. We have that $\mu \models \bigwedge_{a \in \mathbb{A}, \bowtie \in \{>,<,=\}} \Big(a(X) \bowtie 0 \rightarrow a(Z) \bowtie 0\Big)$ since for all $a \in \mathbb{A}$, $\mu \models a(X) \bowtie 0 \rightarrow a(Z) \bowtie 0$ (when $a \in s_1$ we have $\mu \models a(Z) \bowtie 0$, while when $a \notin s_1$ the implication trivially holds). Similarly, this happens for $\bigwedge_{a \in \mathbb{A}, \bowtie \in \{>,<,=\}} \Big(a(X) \bowtie 0 \rightarrow a(Z) \bowtie 0\Big)$. We can see the disjunction: $\bigvee_{a \in \mathbb{A}, \bowtie \in \{>,<,=\}} \Big(a(X) \bowtie 0 \wedge (\neg \mathit{In}_{\boldsymbol{f}, a \bowtie 0}(Z))\Big)$ as:

$$\bigvee_{a \bowtie 0 \in s_1} \Big(a(X) \bowtie 0 \wedge (\neg \mathit{In}_{\boldsymbol{f}, a \bowtie 0}(Z))\Big) \vee \bigvee_{a \bowtie 0 \notin s_1} \Big(a(X) \bowtie 0 \wedge (\neg \mathit{In}_{\boldsymbol{f}, a \bowtie 0}(Z))\Big).$$

We have that μ satisfies the first disjunct (and hence the whole disjunction) because when $a \bowtie 0 \in s_1$ we have that $\mu \models \bigvee_{a \bowtie 0 \in s_1} \neg \mathit{In}_{\boldsymbol{f}, a \bowtie 0}(Z)$.
⇐) We show that $\mu \models \mathit{InsSymb}_{\boldsymbol{f}}(X, X', Z)$ implies $\mu \models \mathit{InsExpl}_{\boldsymbol{f}}(X, X', Z)$. As before, we notice that are only two predicates s_1, s_2 such that $\mu \models s_1(X)$ and $\mu \models s_2(X')$ and that all the predicates not in s_1 and not in s_2 do not hold in μ. Thus, from $\mu \models \mathit{InsSymb}_{\boldsymbol{f}}(X, X', Z)$ we have that

$$\mu \models \bigwedge_{a \in s_1} a(Z) \bowtie 0 \wedge \bigwedge_{a \in s_2} \mathit{In}_{\boldsymbol{f}, a \bowtie 0}(Z) \wedge \bigvee_{a \in s_1} \neg \mathit{In}_{\boldsymbol{f}, a \bowtie 0}(Z).$$

Hence, μ is a model for at least one of the disjuncts in $\mathit{InsExpl}_{\boldsymbol{f}}(X, X', Z)$. □

We similarly define the compact encoding of $\mathit{OutExpl}_{\boldsymbol{f}}(X, X', Z)$:

$$\begin{aligned} \mathit{OutSymb}_{\boldsymbol{f}}(X, X', Z) \doteq & \bigwedge_{a \in \mathbb{A}, \bowtie \in \{>,<,=\}} \Big(a(X) \bowtie 0 \rightarrow \mathit{In}_{-\boldsymbol{f}, a \bowtie 0}(Z)\Big) \wedge \\ & \bigwedge_{a \in \mathbb{A}, \bowtie \in \{>,<,=\}} \Big(a(X') \bowtie 0 \rightarrow a(Z) \bowtie 0\Big) \wedge \\ & \bigvee_{a \in \mathbb{A}, \bowtie \in \{>,<,=\}} \Big(a(X) \bowtie 0 \wedge \neg a(Z) \bowtie 0\Big). \end{aligned} \tag{22}$$

Lemma 2. *$OutExpl_{\boldsymbol{f}}(X, X', Z)$ and $OutSymb_{\boldsymbol{f}}(X, X', Z)$ are equivalent.*

Proof. The proof of Lemma 2 is similar to the proof of Lemma 1. □

We now express the transition relation from Eq. (7) in a compact form:

$$\widehat{T_{Symb_{\mathbb{P}}}}(V_{\mathbb{P}}, V'_{\mathbb{P}}) \doteq \exists X, X'. \Big(N(X, X') \wedge H(X) \wedge H(X') \wedge \tag{23}$$
$$H_{\mathbb{A}}(X, V_{\mathbb{P}}) \wedge H_{\mathbb{A}}(X', V'_{\mathbb{P}}) \wedge$$
$$\exists Z. (InsSymb_{\boldsymbol{f}}(X, X', Z) \vee OutSymb_{\boldsymbol{f}}(X, X', Z)) \Big).$$

Theorem 2. *$\widehat{T}_{\mathbb{P}}(V_{\mathbb{P}}, V'_{\mathbb{P}})$ and $\widehat{T_{Symb_{\mathbb{P}}}}(V_{\mathbb{P}}, V'_{\mathbb{P}})$ are equivalent.*

Proof. Follows directly from Lemma 2 and Lemma 1. □

Implicit Semi-Algebraic Abstraction

The formula $\widehat{T_{Symb_{\mathbb{P}}}}(V_{\mathbb{P}}, V'_{\mathbb{A}})$ represents the transition relation of the semi-algebraic abstraction. Computing the finite-state transition system representing the semi-algebraic abstraction requires to eliminate the existential quantifiers from the initial states, transition relation, and safety property formulas. However, the above formula $\widehat{T_{Symb_{\mathbb{P}}}}(V_{\mathbb{P}}, V'_{\mathbb{A}})$ contains non-linear real arithmetic terms from the polynomials and the Lie derivatives we compute in $In_{\boldsymbol{f}}$, so removing the quantifiers from the formula requires to apply a quantifier elimination algorithm (e.g., Cylindrical Algebraic Decomposition [8]) that does not scale, even when the number of polynomials is small. Instead, we construct a symbolic transition system that *implicitly* encodes the abstraction:

$$S_{Impl,\mathbb{P}} \doteq \langle X \cup \overline{X} \cup Z, \psi(X) \wedge EQ_{\mathbb{P}}(X, \overline{X}), T_{Impl,\mathbb{P}}(\overline{X}, X', Z) \wedge EQ_{\mathbb{P}}(X', \overline{X'}) \rangle,$$

where

$$T_{Impl,\mathbb{P}}(\overline{X}, X', Z) \doteq N(\overline{X}, X') \wedge H(\overline{X}) \wedge H(X') \wedge$$
$$(InsSymb_{\boldsymbol{f}}(\overline{X}, X', Z) \vee OutSymb_{\boldsymbol{f}}(\overline{X}, X', Z)).$$

The above encoding is a an implicit predicate abstraction [36] that preserves reachability properties and is such that:

Theorem 3. *$S_{Impl,\mathbb{P}} \models P(\overline{X})$ if and only if $\widehat{S}_{\mathbb{P}} \models \widehat{\neg\neg P}_{\mathbb{P}}(V_{\mathbb{P}})$.*

Thus, we can model check the transition system $S_{Impl,\mathbb{P}} \models P(\overline{X})$ to prove a property P holds on the dynamical system. Note that, to this purpose, we can apply standard SMT-based model checking algorithms.

The transition system $S_{Impl,\mathbb{P}}$ doubles the state space introducing a copy of the state variables $\overline{X}$ and encodes the equivalence relation between pairs of

concrete states in X and in $\overline{X}$ with the formula $EQ_{\mathbb{P}}(X, \overline{X})$ (c.f. Formula 6). The transition relation $T_{Impl,\mathbb{P}}(\overline{X}, X', Z)$ then encodes a transition in the semi-algebraic abstraction with the linear encoding $InsSymb_{\boldsymbol{f}}(\overline{X}, X', Z)$ and $OutSymb_{\boldsymbol{f}}(\overline{X}, X', Z)$. In this way, a transition in the transition system $S_{Impl,\mathbb{P}}$ corresponds to a transition in the semi-algebraic abstraction, and vice-versa.

6 Experimental Evaluation

Research Questions

We evaluate the performance of our approach (*Implicit Abstraction*) for the verification of invariant properties on the semi-algebraic abstraction of dynamical systems. *Implicit Abstraction* first encodes the semi-algebraic abstraction in a transition system (as we show in Section 5), and then model checks the invariant on the transition system with an off-the-shelf model checker. Our experiments aim to answer the following research questions:
RQ 1: How does *Implicit Abstraction* compare with the *LazyReach* algorithm [33], which explicitly enumerates the reachable states of the abstraction?
RQ 2: How does *Implicit Abstraction* compare with the *DWCL* algorithm [33], which applies a divide-and-conquer strategy to reduce the number of polynomials in the abstraction?

Experimental Setup

We implemented the construction of the implicit abstraction transition system in Python using *PySMT* [11] to manipulate formulas, and *SymPy* [23] for polynomial manipulation and Gröbner bases computation (i.e., to compute the Lie derivatives' ranks). We verify the implicit abstraction transition system with the model checking algorithm for symbolic transition systems with NRA constraints from [4]. The algorithm abstracts the non-linear transition system into a linear transition system, which is checked by the algorithm in [6] and is implemented using the *MathSAT* [7] SMT solver. We implemented both the *LazyReach* and the *DWCL* algorithms in the same Python tool. Our implementation of *DWCL* can use different backends to decide the satisfiability of NRA formulas, namely *MathSAT*[9], the *z3* SMT solver [25], or *Mathematica* [17].

We consider 90 invariant verification problems for dynamical systems from the *KeyMaera X* theorem prover [10]. These problems are a superset of the ones used in [33] and are used in the Applied Verification of Continuous and Hybrid Systems (ARCH) competition [24]. We obtain a total of 180 benchmark instances using, for each problem, two sets of polynomials for the semi-algebraic abstraction[10]. The first set contains all the factors of the right-hand side of the

[9] *MathSAT* uses a different decision procedure [4] than *z3* and *Mathematica* based on incremental linearization rather than cylindrical algebraic decomposition.

[10] The benchmarks have 321 sign-invariant polynomials (c.f. Section 2) over a total of 1089 polynomials that *DWCL* will use to split the state space.

ODEs; the second set extends the first one by including also the Lie derivatives of the polynomials. The latter set induces an abstraction that is more precise but also has a larger state-space.

We evaluate the performance of the algorithms *Implicit Abstraction*, *LazyReach*, and *DWCL* to solve the above verification problems. The underlying problem requires to decide the satisfiability of NRA formulas, and the decision procedures for this problem are efficient for different subsets of problems. For this reason we further evaluate different configurations of the *LazyReach* and *DWCL* algorithms using three different solvers for NRA formulas (*MathSAT*, *z3*, and *Mathematica*). Note that, while in principle, we could use multiple SMT backends also in the model checking algorithm [4] and replace the *MathSAT* SMT solver with another SMT solver (e.g., *z3*), this change would not significantly impact the overall performance, because the algorithm abstracts the non-linear formulas with linear ones where both *MathSAT* and *z3* have comparable performance.

We run the *Implicit Abstraction*, *LazyReach*, and *DWCL* algorithm on all the 180 benchmark instances with a time out of 100 seconds, and we measure the execution times to either prove (*safe* result) or find an abstract counterexample (*unknown* result) for each instance. An archive containing the necessary to reproduce the experiments is available online at `http://www.sergiomover.eu/cav2021.html`.

Results

RQ 1 - Implicit Abstraction *vs.* LazyReach. From the cumulative plot in Figure 2, we see that *Implicit Abstraction* almost always outperforms *LazyReach*.

From the cumulative plot in Figure 2a we see that *Implicit Abstraction* significantly outperforms *LazyReach* on *safe* instances. For better readability, in the plot we only show the (virtual) portfolio algorithm running each configuration of *LazyReach*, *Virtual Best LazyReach*, obtained by considering the best run time among the different configurations of *LazyReach* using different backend solvers. *Virtual Best LazyReach* solves a total of 42 safe instances, while *Implicit Abstraction* solves 100 safe instances. The scatter plots shown in the first row of Figure 3 confirms the same intuition (note that the safe instances represented as blue circles are mostly in the lower-right triangle of the plot).

Figure 2b shows the cumulative plot when verifying *unknown* instances. Note that the total number of unknown instances in the benchmarks are much smaller than the safe ones (combining the results of all the algorithms we have 123 safe instances, 19 unknown instances, and 38 still unsolved instances). From Figure 2b, we see that the performance of *Implicit Abstraction* is comparable with *LazyReach*, solving a total of 8 instances and 11 instances respectively.

RQ 2 - Implicit Abstraction *vs.* DWCL. From the cumulative plots in Figure 2, the *Virtual Best DWCL* solves 37 more instances than *Implicit Abstraction*. However, we also see from Figure 2 that the global *Virtual Best* solves more instances and is faster than *Virtual Best DWCL*. In fact, *Implicit Abstraction* is *orthogonal* to *DWCL* and is comparable to *DWCL* when fixing either *Mathematica* or *z3*

(*Implicit Abstraction* solves 108 instances, *DWCL Mathematica* solves 109, and *DWCL z3* solves 114).

The scatter plots in the second row of Figure 3 compare *Implicit Abstraction* with *DWCL MathSAT*, *DWCL Mathematica*, and *DWCL z3*. From these plots, we see that there are several instances that are solved by only one of the two algorithms compared in each plot. While we see similar data when comparing *Implicit Abstraction* with *Virtual Best DWCL* (always in the scatter plots of Figure 3), the number of instances solved uniquely by *Implicit Abstraction* seems smaller. We get a more precise picture of the complementarity of *Implicit Abstraction*, *DWCL Mathematica*, and *DWCL z3* from the diagrams in Figure 4, where we can clearly see that *Implicit Abstraction* is orthogonal to both *DWCL Mathematica* and *DWCL z3*. From the diagram, we see that when using a different backend (i.e., *Mathematica* or *z3*) *DWCL* solves a different set of instances. This difference in performance using *Mathematica* and *z3* is not surprising since *Mathematica* and *z3* uses different algorithms to solve formulas in NRA.

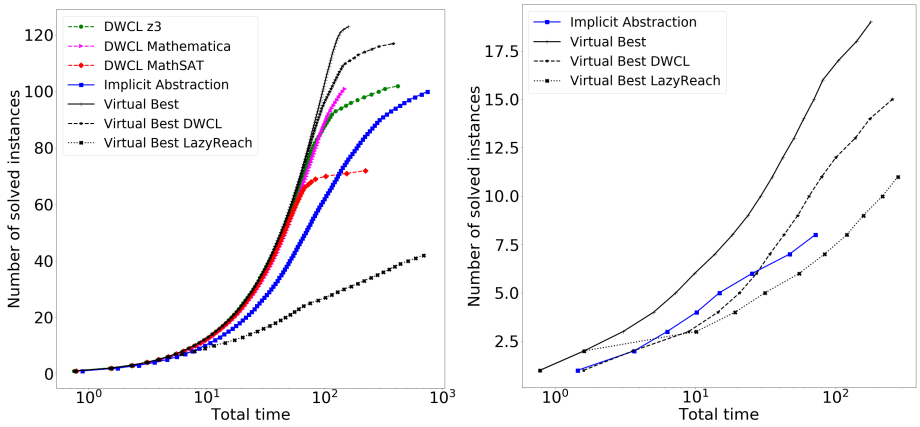
We further notice that *Implicit Abstraction* uses the *MathSAT* SMT solver in the backend, and from our experiments (see again Figure 3) *DWCL MathSAT* performs quite poorly compared to both *DWCL Mathematica* and *DWCL z3*. While naively replacing *MathSAT* in the model checking algorithm we use [4] would not provide a significant performance improvement, it is reasonable to think that investigating a tighter integration with either *z3* or *Mathematica* could improve the model checking performance. However, we believe this integration to be beyond the scope for this paper, where we enable the use of symbolic model checking techniques to analyze the semi-algebraic decomposition.

7 Related work

In this work, we focus on the (unbounded time) safety verification problem for polynomial dynamical systems. Such problem is relevant when proving safety for hybrid programs [27] with Keymaera X [10] or for hybrid CPS with the HHL Prover [37]. Our reduction to transition systems may be used as sub-procedure in both theorem provers to automate the search of a continuous invariant.

There exist different techniques to prove safety properties for polynomial dynamical systems (see e.g., [13]): barrier certificates [30,18], first integrals [14], and Darboux Polynomials [15]. All these techniques are orthogonal to semi-algebraic abstraction, and can be used to find invariant polynomials to restrict the abstract state space. Pegasus [34] implements all the above techniques, the *LazyReach*, and *DWCL* algorithms. Our algorithm can be integrated in Pegasus. The LZZ [22] procedure has been originally proposed to synthesize a continuous invariant. Instead, we use the LZZ procedure to encode the abstract transition relation, and then we prove a safety property in the abstraction. We also provide a specialized encoding of LZZ to check the existence of abstract transitions.

The semi-algebraic abstraction [33] is a qualitative abstraction [35,38]. In this work, we propose a different algorithm to verify semi-algebraic abstractions that allows us to explore the abstract state-space symbolically, in contrast to



(a) Safe instances (b) Unknown instances

Fig. 2. Plots the total number of instances (on the y axis) as a function of the cumulative time (in seconds, on the x axis) took by *Implicit Abstraction*, *LazyReach*, and *DWCL* to solve (a) *safe* and (b) *unknown* instances. The comparison includes the results of *LazyReach* and *DWCL* using different (*MathSAT*, *z3*, and *Mathematica*), as well as virtual portfolios combining the best results obtained by a given algorithm when run with multiple backends. We omit some configurations in (b) to improve readability.

the *LazyReach* algorithm [33]. In principle, our technique is orthogonal to the *DWCL* algorithm [33], since we could replace *LazyReach*, which is used in *DWCL* as a sub-routine, with our approach (i.e., model check the implicit abstraction).

Relational abstraction [32] abstracts the dynamical system's trajectories with a discrete transition relation, reducing the verification problem on the continuous system to a verification problem on the discrete system. The implicit encoding of the semi-algebraic abstraction can be seen as an instance of relational abstraction, where a trajectory of the dynamical system is mapped to a sequence of abstract transitions (similarly to what happen with relational abstractions for time-sampled systems in [39,2]). Since relational abstractions can be composed with each other (e.g., see [26]), we can strengthen the implicit semi-algebraic abstraction encoding with a relational abstraction. This composition is useful in the case the semi-algebraic abstraction cannot easily capture the system's behavior (e.g., a precise relation of the time elapsed in a transition [26]).

Predicate abstraction [16] is a commonly used abstraction techniques to verify infinite-state systems. Several symbolic techniques [19,20,3] focus on the efficient computation of the predicate abstraction. In principle, we can also use those technique to explicitly compute the semi-algebraic abstraction. However, the up-front, explicit computation of the abstraction is a bottleneck and can be avoided with implicit predicate abstraction [36] when the goal is to verify a safety property on the abstract system. We use implicit abstraction to obtain an implicit encoding of the semi-algebraic abstraction. The transition system of

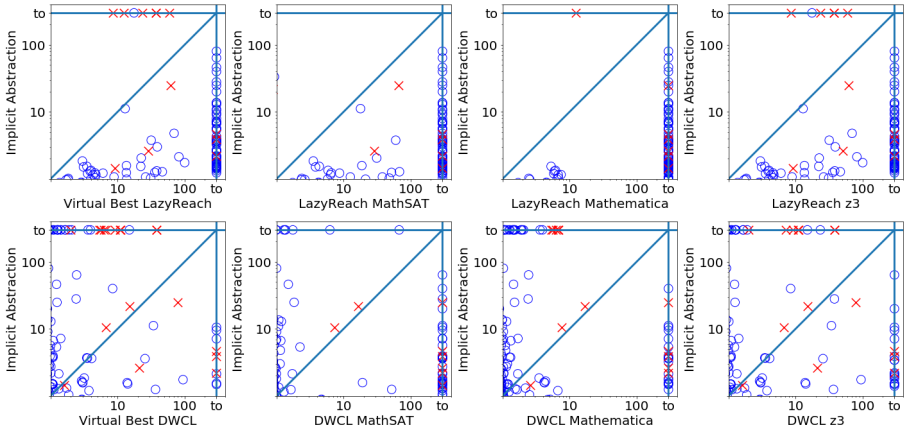


Fig. 3. Scatter plots comparing the run time (in seconds) of *Implicit Abstraction* (on the y axis) with *LazyReach* (first row, on the x axis) and *DWCL* (second row, on the x axis). Blue circles represent safe verification problems. Red crosses are instances where the algorithm found an abstract counterexample. When *Implicit Abstraction* runs for more than the 100 seconds time out, we plot the instance on the vertical line marked as *to*, and similarly for *LazyReach* and *DWCL* on the horizontal line.

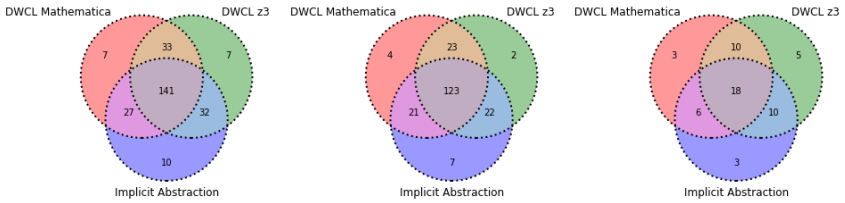
the semi-algebraic abstraction contains NRA formulas (the polynomials can be non-linear or the Lie derivative of the polynomials are non-linear). While there are few algorithms and tool that can verify such transition systems (e.g., [4]), our technique is agnostic to the underlying model checking algorithm.

8 Conclusions and Future Work

In this paper, we addressed the safety problem of polynomial dynamical systems. We built on the LZZ algorithm to define a symbolic encoding of the abstraction based on a set of polynomials. The encoding is linear in the number of polynomials and can be used to implicitly represent the abstraction without the need of enumerating the abstract states, enabling the use of SMT-based model checking techniques. The experimental evaluation showed that the approach is promising and complementary to existing techniques solving a number of new instances.

The main directions for future works are, on one side, refining the abstraction discovering new polynomials that are able to remove spurious abstract counterexamples, and, on the other side, the application of the approach to hybrid systems where the continuous dynamics depends on the discrete state of the system.

Acknowledgements S. Mover was partially supported by the academic chair "Complex Systems Engineering", École Polytechnique-ENSTA Paris-Télécom Paris-Thalés-Dassault Aviation-Naval Group-DGA-Fondation ParisTech-FX and the AID project "Validation of Autonomous Drones and Swarms of Drones". A.



(a) All the instances (b) Safe instances (c) Unknown instances

Fig. 4. Diagrams representing the distribution of unique instances solved combining different algorithms (*DWCL Mathematica*, *DWCL z3*, and *Implicit Abstraction*). Each set, displayed as a dotted circle enclosed by a dotted line, represents the set of instances solved with one algorithm. The number shown in each partition is the number of instances solved uniquely by the sets forming the partition. For example, the central partition (i.e, the intersection of all the sets) of the diagram (a) shows that *DWCL Mathematica*, *DWCL z3*, and *Implicit Abstraction* solved the same set of 141 instances.

Cimatti, A. Griggio, and S. Tonetta were partially supported by ECSEL JU under grant agreement No 876852. The JU receives support from EU's H2020 programme, Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey.

References

1. Birgmeier, J., Bradley, A.R., Weissenbacher, G.: Counterexample to Induction-Guided Abstraction-Refinement (CTIGAR). In: CAV. pp. 831–848 (2014)
2. Chen, X., Mover, S., Sankaranarayanan, S.: Compositional Relational Abstraction for Nonlinear Hybrid Systems. ACM Trans. Embedded Comput. Syst. **16**(5), 187:1–187:19 (2017)
3. Cimatti, A., Franzén, A., Griggio, A., Kalyanasundaram, K., Roveri, M.: Tighter integration of bdds and smt for predicate abstraction. In: DATE. pp. 1707–1712. IEEE (2010)
4. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental Linearization for Satisfiability and Verification Modulo Nonlinear Arithmetic and Transcendental Functions. ACM Trans. Comput. Log. **19**(3), 19:1–19:52 (2018)
5. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: IC3 Modulo Theories via Implicit Predicate Abstraction. In: TACAS. pp. 46–61 (2014)
6. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: Infinite-state invariant checking with IC3 and predicate abstraction. Formal Methods in System Design **49**(3), 190–218 (2016)
7. Cimatti, A., Griggio, A., Schaafsma, B.J., Sebastiani, R.: The MathSAT5 SMT Solver. In: TACAS. pp. 93–107 (2013)
8. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In: Automata theory and formal languages, pp. 134–183. Springer (1975)
9. Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: POPL. pp. 191–202 (2002)
10. Fulton, N., Mitsch, S., Quesel, J., Völp, M., Platzer, A.: KeYmaera X: An Axiomatic Tactical Theorem Prover for Hybrid Systems. In: CADE. pp. 527–538 (2015)
11. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT Workshop 2015 (2015)
12. Ghorbal, K., Sogokon, A.: Characterizing Positively Invariant Sets: Inductive and Topological Methods. CoRR **abs/2009.09797** (2020), `https://arxiv.org/abs/2009.09797`
13. Ghorbal, K., Sogokon, A., Platzer, A.: A hierarchy of proof rules for checking positive invariance of algebraic and semi-algebraic sets. Computer Languages, Systems & Structures **47**, 19–43 (2017)
14. Goriely, A.: Integrability and nonintegrability of dynamical systems (2001)
15. Goubault, E., Jourdan, J., Putot, S., Sankaranarayanan, S.: Finding non-polynomial positive invariants and lyapunov functions for polynomial systems through darboux polynomials. In: ACC. pp. 3571–3578 (2014)
16. Graf, S., Saïdi, H.: Construction of Abstract State Graphs with PVS. In: CAV. Lecture Notes in Computer Science, vol. 1254, pp. 72–83. Springer (1997)
17. Inc., W.R.: Mathematica, Version 12.2, `https://www.wolfram.com/mathematica`, champaign, IL, 2020
18. Kong, H., He, F., Song, X., Hung, W.N.N., Gu, M.: Exponential-Condition-Based Barrier Certificate Generation for Safety Verification of Hybrid Systems. In: CAV. pp. 242–257 (2013)
19. Lahiri, S.K., Bryant, R.E., Cook, B.: A Symbolic Approach to Predicate Abstraction. In: CAV. pp. 141–153 (2003)

20. Lahiri, S.K., Nieuwenhuis, R., Oliveras, A.: SMT Techniques for Fast Predicate Abstraction. In: CAV. pp. 424–437 (2006)
21. Liu, J., Lv, J., Quan, Z., Zhan, N., Zhao, H., Zhou, C., Zou, L.: A calculus for hybrid CSP. In: APLAS. Lecture Notes in Computer Science, vol. 6461, pp. 1–15. Springer (2010)
22. Liu, J., Zhan, N., Zhao, H.: Computing semi-algebraic invariants for polynomial dynamical systems. In: EMSOFT. pp. 97–106 (2011)
23. Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.: Sympy: symbolic computing in python. PeerJ Computer Science **3**, e103 (Jan 2017)
24. Mitsch, S., Munive, J.J.H.Y., Jin, X., Zhan, B., Wang, S., Zhan, N.: Arch-comp20 category report:hybrid systems theorem proving. In: ARCH20. EPiC Series in Computing, vol. 74, pp. 153–174. EasyChair (2020)
25. de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: TACAS. pp. 337–340 (2008)
26. Mover, S., Cimatti, A., Tiwari, A., Tonetta, S.: Time-aware relational abstractions for hybrid systems. In: EMSOFT. pp. 14:1–14:10 (2013)
27. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reason. **41**(2), 143–189 (2008)
28. Platzer, A.: Differential Dynamic Logic for Hybrid Systems. J. Autom. Reason. **41**(2), 143–189 (2008)
29. Platzer, A., Clarke, E.M.: Computing differential invariants of hybrid systems as fixedpoints. Formal Methods in System Design **35**(1), 98–120 (2009)
30. Prajna, S.: Barrier certificates for nonlinear model validation. Autom. **42**(1), 117–126 (2006)
31. Roohi, N., Prabhakar, P., Viswanathan, M.: HARE: A Hybrid Abstraction Refinement Engine for Verifying Non-linear Hybrid Automata. In: TACAS. pp. 573–588 (2017)
32. Sankaranarayanan, S., Tiwari, A.: Relational Abstractions for Continuous and Hybrid Systems. In: CAV. pp. 686–702 (2011)
33. Sogokon, A., Ghorbal, K., Jackson, P.B., Platzer, A.: A Method for Invariant Generation for Polynomial Continuous Systems. In: VMCAI. pp. 268–288 (2016)
34. Sogokon, A., Mitsch, S., Tan, Y.K., Cordwell, K., Platzer, A.: Pegasus: A Framework for Sound Continuous Invariant Generation. In: FM. pp. 138–157 (2019)
35. Tiwari, A.: Abstractions for hybrid systems. Formal Methods Syst. Des. **32**(1), 57–83 (2008)
36. Tonetta, S.: Abstract Model Checking without Computing the Abstraction. In: FM. pp. 89–105 (2009)
37. Wang, S., Zhan, N., Zou, L.: An Improved HHL Prover: An Interactive Theorem Prover for Hybrid Systems. In: ICFEM. pp. 382–399 (2015)
38. Zaki, M.H., Denman, W., Tahar, S., Bois, G.: Integrating Abstraction Techniques for Formal Verification of Analog Designs. J. Aerosp. Comput. Inf. Commun. **6**(5), 373–392 (2009)
39. Zutshi, A., Sankaranarayanan, S., Tiwari, A.: Timed Relational Abstractions for Sampled Data Control Systems. In: CAV. pp. 343–361 (2012)