Cone-Based Abstract Interpretation for Nonlinear Positive Invariant Synthesis

Guillaume O. Berger
UCLouvain
Belgium
guillaume.berger@uclouvain.be

Masoumeh Ghanbarpour
University of Colorado Boulder
USA
masoumeh.ghanbarpour@colorado.edu

Sriram Sankaranarayanan
University of Colorado Boulder
USA
first.lastname@colorado.edu

ABSTRACT

We present an abstract interpretation approach for synthesizing nonlinear (semi-algebraic) positive invariants for systems of polynomial ordinary differential equations (ODEs) and switched systems. The key behind our approach is to connect the system under study to a positive nonlinear system through a "change of variables". The positive invariance of the first orthant ($\mathbb{R}_+$) for a positive system guarantees, in turn, that the functions involved in the change of variables define a positive invariant for the original system. The challenge lies in discovering such functions for a given system. To this end, we characterize positive invariants as fixed points under an operator that is defined using the Lie derivative. Next, we use abstract-interpretation approaches to systematically compute this fixed point. Whereas abstract interpretation has been applied to the static analysis of programs, and invariant synthesis for hybrid systems to a limited extent, we show how these approaches can compute fixed points over cones generated by polynomials using sum-of-squares optimization and its relaxations. Our approach is shown to be promising over a set of small but hard-to-analyze nonlinear models, wherein it is able to generate positive invariants to place useful bounds on their reachable sets.

ACM Reference Format:
Guillaume O. Berger, Masoumeh Ghanbarpour, and Sriram Sankaranarayanan. 2024. Cone-Based Abstract Interpretation for Nonlinear Positive Invariant Synthesis. In *27th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '24), May 14–16, 2024, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3641513.3650127

1 INTRODUCTION

In this paper, we provide solutions to the problem of synthesizing semi-algebraic positive invariants for ordinary differential equations (ODEs) whose right-hand sides are defined by polynomials over the system variables. We show how our approach extends to switched systems that contain multiple modes, each described by polynomial ODEs along with transitions between these modes, governed by semi-algebraic guard conditions. Positive invariants of ODEs and switched systems help us prove bounds on the sets of states that can be reached over an infinite time horizon, thus proving that a certain set of unsafe states will never be reached. The problem of automatically synthesizing such positive invariants has been of great interest for verification. Many approaches have been studied for this problem that include constructing finite abstractions [4, 6, 50], dynamic programming-based approaches [52], approaches based on solving nonlinear constraints in order to construct barrier functions and their generalizations [7, 8, 36, 37].

The key of our approach is to relate an ODE $\dot{x} = f(x)$ to a positive system of the form $\dot{\omega} = -\lambda\omega + F(z, \omega)$, wherein $\omega \in \mathbb{R}^m$ is connected to the state $x \in \mathbb{R}^n$ through a polynomial map $(\omega_1, \ldots, \omega_m) = (g_1(x), \ldots, g_m(x))$ such that λ is a fixed scalar quantity, $g_1, \ldots, g_m$ are polynomials and F is a nonlinear function with the property that for all $z \in \mathbb{R}$ and $\omega \in \mathbb{R}^m_+$, we have $F(z, \omega) \in \mathbb{R}^m_+$. Here, z is taken to be an external input. In other words, we show that our original vector field is "f-related" to that of a positive system. Positive systems have a key property that if $\omega(0) \in \mathbb{R}^m_+$ then $\omega(t) \in \mathbb{R}^m_+$ for all times t over which the trajectory is defined. We can use this property to infer that $g_1 \geq 0, \ldots, g_m \geq 0$ is a positive invariant set for the original ODE. There are many difficulties to this approach, including: (a) choosing the dimensions of the unknown state space ω; (b) discovering the function F; and (c) synthesizing the functions $g_1, \ldots, g_m$. We propose a method to synthesize the polynomials $g_1, \ldots, g_m$ that will implicitly define the positive system, and thus, our positive invariant set. We require the user to fix the constant λ, an upper limit on the number m, and a degree limit on the polynomials $g_1, \ldots, g_m$. Our approach either converges with positive invariants (the function F gets defined implicitly by the generators upon convergence) or fails with a trivial answer.

To discover positive invariants, we iterate over finitely generated polynomial cones that are defined by a basis set of polynomials. We first derive a "refinement operator" of the cone from the given ODE. We derive a closure condition based on this operator: i.e, for a given finitely generated cone of polynomials C, our closure condition requires that every polynomial in C is mapped by the refinement operator to a related cone $\hat{C}$. If C satisfies the closure condition, then we prove that its generators form the required functions $g_1, \ldots, g_m$ that will map the original system dynamics to a positive nonlinear system, as described above. Therefore, $g_1 \geq 0, \ldots, g_m \geq 0$ will define a positive invariant. Having defined the notion of positive invariants in terms of closure of an operator, the challenge now lies in computing cones that are closed in this manner. To do so, we employ an approach based on abstract interpretation. Abstract interpretation was originally proposed by Cousot and Cousot in 1977 as an approach for establishing invariants of programs [14, 15]. It has been very successful in proving properties of large safety-critical software systems used in avionics [3, 16]. However, applying abstract interpretation to our framework is challenging since we

HSCC '24, May 14–16, 2024, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-0522-9/24/05…$15.00
https://doi.org/10.1145/3641513.3650127

will observe that the closure condition used leads us from finitely generated polynomial cones to possibly infinitely generated cones. We show how a finite set of generators can be selected and maintained using a projection operator defined in this paper. Finally, we show that a "standard" *widening* operator commonly used in abstract interpretation can force termination in finitely many steps.

We provide an empirical evaluation of our approach over a set of nonlinear polynomial ODE benchmarks and switched systems, some of which are taken from the related work. We demonstrate that our approach can yield useful positive invariant sets. Even though we employ floating-point numbers in the calculation of these invariants, they have been successfully verified using the exact-arithmetic-based nonlinear theorem prover Z3. We also compare our approach with well-established approaches for synthesizing barrier functions based on sum-of-squares programming [36].

1.1 Related Work

There have been numerous techniques to directly synthesize positive invariants for ODEs and hybrid systems. This includes barrier function synthesis [36], approaches based on constraint solving by assuming a template form of the nonlinear invariant [20, 26, 35, 43, 46, 48, 51] and abstract-interpretation approaches based on forward propagation and widening [21, 42], or in other cases through forward propagation and *extrapolation* [22]. Additionally, theorem provers such as Keymaera-X support proving properties of practical hybrid systems using positive invariant synthesis to support human reasoning [19, 32, 33, 35]. Due to space limitations, we do not expand on these approaches, noting that some of the recent textbooks cover these approaches and the theory behind them [5, 29, 34, 39].

The closest related works to our approach include the notion of comparison systems proposed by Sogokon et al. [46], the notion of a change-of-basis transformation proposed by Sankaranarayanan [40, 41], and abstract-interpretation-based iteration over polyhedral cones for linear systems first proposed by Sankaranarayanan et al. for linear hybrid systems [42]. Sogokon et al. [46] propose the notion of vector barrier function which is a vector of functions that relates the flow of a nonlinear ODE to that of a positive linear system of the form $\dot{\boldsymbol{\omega}} = \Lambda\boldsymbol{\omega} + \boldsymbol{r}(t)$, where Λ is a constant Metzler (aka. essentially nonnegative) matrix and $\boldsymbol{r}(t) \geq 0$. Their approach synthesizes the polynomials $g_1, \ldots, g_m$ by (a) assuming that the matrix Λ is given by the user and (b) the degree limits of the polynomials are specified. Our approach extends this concept to matrices Λ whose off-diagonal entries are positive definite polynomials and significantly *does not* require the user to provide us these matrices. However, the computational complexity of our approach is significant and we have to rely on heuristics to select generators from an infinitely generated cone. Nevertheless, we show success on small but interesting nonlinear systems. Sankaranarayanan [40, 41] proposes a similar idea of a change-of-basis transformation from a given nonlinear system to an *autonomous* linear system and uses an iterative abstract-interpretation-based procedure similar to what is being proposed here. However, the connection to an autonomous system essentially requires the original nonlinear system to be integrable, or in other words, have equality invariants (though these may not be necessarily be polynomial). Finally, the idea of abstract interpretation on cones was developed in Sankaranarayanan et al. [42]. But this work was restricted to linear systems where the iterations need to be over polyhedral cones. This paper goes much further and considers nonlinear systems as well as non-polyhedral cones generated by polynomials.

The synthesis of positive invariants has received much attention in the past. Taly and Tiwari provide a proof system that is sound and relatively complete for a single polynomial inequality using higher-order derivatives [49]. Liu et al. extend this to a powerful relatively complete method for synthesizing semi-algebraic positive invariants for polynomial hybrid systems [26]. Their approach fixes the form (aka. template) of the desired invariant and uses a condition based on higher-degree Lie derivative. This is essentially a refinement of the barrier set condition that states that the first non-zero higher Lie derivative must be positive at the boundaries of the invariant set. By cleverly connecting their approach to the descending chain condition for ideals on a polynomial ring, they are able to provide a relative completeness guarantee. The proof system of Liu et al. is relatively complete unlike ours which is weaker than that of Liu et al. because (a) our approach is limited to (closed) basic semi-algebraic sets and (b) we use a weaker positive-invariance condition based on relating to a positive nonlinear system. On the other hand, our approach does not use expensive quantifier elimination over semi-algebraic sets: each iteration of our approach uses sum-of-squares optimization. Ghorbal et al. provide a hierarchy of proof rules for computing semi-algebraic invariants that places the work of Liu et al. in context at the apex of a series of increasingly more complex proof rules. They also provide interesting comparisons on the types of flows that each rule can handle [20]. Our approach has two major differences: (a) we use the connection with positive systems to avoid reasoning about the boundaries of the invariant sets or *explicitly* compute higher-order derivatives; and (b) we work on basic semi-algebraic sets defined as intersections of polynomial inequalities. Our approach does not extend to general semi-algebraic sets which are unions of these basic sets.

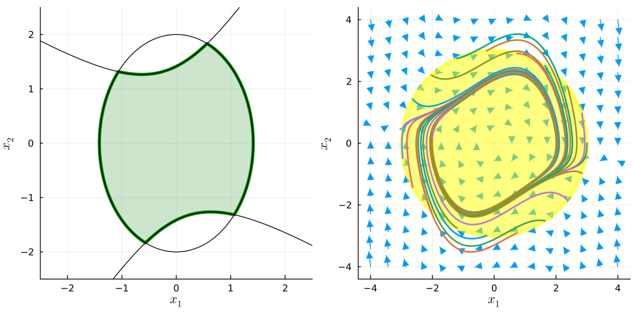
2 PROBLEM STATEMENT

Notation. Let $\mathbb{N}$ be the set of natural numbers, and $\mathbb{R}_+$ be the set of nonnegative real numbers. Given $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. We will use bold-face to denote vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{R}^k$ and capital letters to describe matrices $A, B, C \in \mathbb{R}^{m\times n}$. For a vector $\boldsymbol{x} \in \mathbb{R}^n$, the i^{th} component for $i \in [n]$ is denoted as x_i. Let $\mathbb{R}[\boldsymbol{x}]$ be the ring of polynomials over variables $\boldsymbol{x} = (x_1, \cdots, x_n)$. Given a function $f : A \to B$, let $\mathrm{dom}(f) = A$.

2.1 Polynomial Systems

Consider a continuous-time dynamical system $\mathrm{ODE}(f)$: $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$, wherein $f : \mathbb{R}^n \to \mathbb{R}^n$ is locally Lipschitz continuous. A *trajectory* of $\mathrm{ODE}(f)$ is defined as a differentiable function $\phi : [0, T) \to \mathbb{R}^n$ satisfying that for all $t \in [0, T)$, $\dot{\phi}(t) = f(\phi(t))$.

Remark 2.1. The trajectory may exist for all time, i.e, $T = \infty$. However, since our focus is on safety, we simply assume that the trajectory exists at least until some time $T > 0$, and place no bound on T. Also, since we assume local Lipschitz continuity, the trajectory must exist and be unique [28].



(a) Example 2.2 (b) Example 2.4

Figure 1: (a) The BSA set in Example 2.2 (green area). (b) Vector field (blue arrows), initial set (yellow area) and sample trajectories (colored curves) of the system in Example 2.4.

Let $\mathcal{I} \subseteq \mathbb{R}^n$ be an *initial set*. A system $\Sigma := \langle f, \mathcal{I} \rangle$ consists of a dynamical system $\mathrm{ODE}(f)$ and an initial set $\mathcal{I}$. The system Σ is said to be *safe* if no trajectory ϕ of $\mathrm{ODE}(f)$ with $\phi(0) \in \mathcal{I}$ reaches some given *unsafe set* $\mathcal{S}_{\text{unsafe}}$. We wish to prove the safety of *polynomial systems*, wherein (a) the dynamics are described by polynomials and (b) the initial set is a *basic semi-algebraic set*, described by polynomial inequalities, as follows.

Definition 2.1 (Basic Semi-Algebraic Sets). A *basic semi-algebraic set* (BSA set) is a set described by a finite set of polynomial inequalities of the form $g_i(\boldsymbol{x}) \geq 0$. More formally, for a set of polynomials $G = \{g_1, \ldots, g_m\} \subseteq \mathbb{R}[\boldsymbol{x}]$, we denote the BSA generated by G as

$$S(G) = \{\boldsymbol{x} \in \mathbb{R}^n : g_i(\boldsymbol{x}) \geq 0,\ i = 1, \ldots, m\}\,.$$

Example 2.2. Fig. 1 (left) depicts the BSA set $S(G)$ wherein $G = \{1 + \frac{1}{2}x_1^2 + \frac{1}{2}x_1x_2 - \frac{1}{2}x_2^2, 1 - x_1^2 - \frac{1}{2}x_2^2\}$.

Definition 2.3 (Polynomial System). The system $\langle f, \mathcal{I} \rangle$ is a *polynomial system* if (a) the vector field f is defined by polynomials: $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ with $f_i \in \mathbb{R}[\boldsymbol{x}]$; (b) $\mathcal{I}$ is a BSA set $S(G_{\text{init}})$ for a finite set of polynomials $G_{\text{init}} \subseteq \mathbb{R}[\boldsymbol{x}]$.

Example 2.4. Consider the following Vanderpol oscillator:

$$\mathrm{ODE}(f): \quad \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{1}{2}x_2 - x_1 - \frac{1}{2}x_1^2 x_2 \end{bmatrix},$$

with initial set $\mathcal{I} = \{x_1^2 + x_2^2 \leq \frac{1}{4}\}$, i.e., $G_{\text{init}} = \{\frac{1}{4} - x_1^2 - x_2^2\}$. The tuple $\langle f, \mathcal{I} \rangle$ is a polynomial system. The vector field and sample trajectories of $\mathrm{ODE}(f)$ are presented in Fig. 1 (right).

2.2 Constrained and Switched Polynomial Systems

Next, we define constrained and switched systems.

Definition 2.5 (Constrained Polynomial System). A *constrained polynomial system* is a triple $\langle f, \mathcal{I}, \mathcal{D} \rangle$ wherein the dynamics is given by $\mathrm{ODE}(f)$ for $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ with $f_i \in \mathbb{R}[\boldsymbol{x}]$, $\mathcal{D} \subseteq \mathbb{R}^n$ is the *domain of evolution* (or *constraint*) that restricts the state space of the system and $\mathcal{I} \subseteq \mathcal{D}$ is the initial set. Furthermore, $\mathcal{I}$ and $\mathcal{D}$ are nonempty BSA sets.

A trajectory of the constrained dynamics $\mathrm{ODE}(f, \mathcal{D})$ is a trajectory $\phi : [0, T) \to \mathbb{R}^n$ of $\mathrm{ODE}(f)$ such that for all time $t \in [0, T)$, the state is in $\mathcal{D}$, i.e., $\phi(t) \in \mathcal{D}$. Consequently, a constrained system is not allowed to reach a state outside of its domain of evolution. The constrained system $\langle f, \mathcal{I}, \mathcal{D} \rangle$ is said to be *safe* if no trajectory ϕ of $\mathrm{ODE}(f, \mathcal{D})$ with $\phi(0) \in \mathcal{I}$ reaches some given *unsafe set* $\mathcal{S}_{\text{unsafe}}$.

Next, we define a switched system through a combination of a finite number of constrained systems connected by transitions.

Definition 2.6 (Switched Polynomial System). A *switched polynomial system* is defined by a set of modes Q wherein each mode $q \in Q$ is associated with a constrained polynomial system $\langle f_q, \mathcal{I}_q, \mathcal{D}_q \rangle$ along with a finite set of transitions $\mathcal{T}$. Each transition $\tau \in \mathcal{T}$ is a triple $\langle a_\tau, b_\tau, \mathcal{G}_\tau \rangle$ with pre- and post-modes $a_\tau \in Q$ and $b_\tau \in Q$ respectively and a guard set $\mathcal{G}_\tau \subseteq \mathcal{D}_{a_\tau} \cap \mathcal{D}_{b_\tau}$. We assume that $\mathcal{G}_\tau$ is a basic semi-algebraic set given by $S(G_{\text{guard},\tau})$ for a finite set of polynomials $G_{\text{guard},\tau}$.

A trajectory of the switched dynamics $\mathrm{ODE}(Q, f_{q \in Q}, \mathcal{D}_{q \in Q}, \mathcal{T})$ is specified by (a) a finite, increasing sequence of times $0 = t_0 < t_1 < \ldots < t_k < t_{k+1}$, (b) a sequence of modes $q_0, q_1, \ldots, q_k \in Q$, (c) a sequence of transitions $\tau_1, \tau_2, \ldots, \tau_k \in \mathcal{T}$, and (d) a function $\phi : [0, t_{k+1}) \to \mathbb{R}^n$ such that the following conditions hold:

1. For each $i \in [k]$, τ_i is a transition from q_{i-1} to q_i and the guard condition $\phi(t_i) \in \mathcal{G}_{\tau_i}$ holds.
2. For each $i \in [k] \cup \{0\}$, the function $\phi_i : [0, t_{i+1} - t_i) \to \mathbb{R}^n$ defined by $\phi_i(t) = \phi(t - t_i)$ is a trajectory of $\mathrm{ODE}(f_{q_i}, \mathcal{D}_{q_i})$.

The switched system $\langle f_{q \in Q}, \mathcal{I}_{q \in Q}, \mathcal{D}_{q \in Q} \rangle$ is said to be *safe* if no trajectory $\langle \phi, q_{0:k}, t_{0:k+1} \rangle$ of $\mathrm{ODE}(Q, f_{q \in Q}, \mathcal{D}_{q \in Q}, \mathcal{T})$ with $\phi(0) \in \mathcal{I}_{q_0}$ reaches some *unsafe set* $\mathcal{S}_{\text{unsafe}}$, i.e., satisfies $\phi(t) \in \mathcal{S}_{\text{unsafe},q_i}$ for some $t \in [t_i, t_{i+1})$.

2.3 Forward Invariant Sets and Safety

An invariant of a dynamical system is a property that is preserved along the trajectories of the system. They can be used to certify safety, for instance, if the property holds for all initial conditions and for no unsafe states.

Definition 2.7. A set $\mathcal{P} \subseteq \mathbb{R}^n$ is *forward invariant* for $\mathrm{ODE}(f)$ if for all trajectories ϕ of $\mathrm{ODE}(f)$ with $\phi(0) \in \mathcal{P}$, and forall $t \in \mathrm{dom}(\phi)$, the state at time t belongs to $\mathcal{P}$: $\phi(t) \in \mathcal{P}$.

Proposition 2.8. *If $\mathcal{P}$ is a forward invariant for $\mathrm{ODE}(f)$, $\mathcal{I} \subseteq \mathcal{P}$ and $\mathcal{P} \cap \mathcal{S}_{\text{unsafe}} = \emptyset$, then the system $\langle f, \mathcal{I} \rangle$ is safe.*

Using Prop. 2.8 we prove safety of a given (constrained and switched) system as follows: Search for a forward invariant set $\mathcal{P}$ that includes the initial set $\mathcal{I}$ that excludes $\mathcal{S}$: i.e, $\mathcal{P} \cap \mathcal{S}_{\text{unsafe}} = \emptyset$. If such a forward invariant can be found, we conclude that the safety property holds for the system.

In this paper, we seek to compute safe invariants for polynomial systems in the form of *basic semi-algebraic sets*, that is, sets described by polynomial inequalities. We remind below some background of polynomial inequalities. Extension of forward invariance to constrained and switched systems will be discussed in Sec. 4.

2.4 Polynomial Inequalities

Note that the polynomial set $G = \{g_1, \ldots, g_m\}$ defining $S(G)$ is not unique. In fact, for any $\alpha_1, \alpha_2 \in \mathbb{R}_+$ and $i_1, i_2 \in [m]$, it holds that $\alpha_1 g_{i_1} + \alpha_2 g_{i_2}$ and $g_{i_1} g_{i_2}$ are also nonnegative on $S(G)$. This motivates the following definitions.

Definition 2.9 (Cone). A set $K \subseteq \mathbb{R}[\boldsymbol{x}]$ is a *cone* if for all $\alpha_1, \alpha_2 \in \mathbb{R}_+$ and $g_1, g_2 \in K$, it holds that $\alpha_1 g_1 + \alpha_2 g_2 \in K$.
Given a set $B \subseteq \mathbb{R}[\boldsymbol{x}]$, we define the *conic hull* of B as

$$\mathrm{ch}(B) = \left\{ \sum_{i=1}^{m} \alpha_i g_i : m \in \mathbb{N},\, g_i \in B,\, \alpha_i \in \mathbb{R}_+ \right\}.$$

A cone $K \subseteq \mathbb{R}[\boldsymbol{x}]$ is called *finitely generated* if there is a finite set $B = \{g_1, \ldots, g_m\} \subseteq \mathbb{R}[\boldsymbol{x}]$ such that $K = \mathrm{ch}(B)$.

Definition 2.10 (Product). Given two sets $G_1, G_2 \subseteq \mathbb{R}[\boldsymbol{x}]$, their *product* is defined by

$$G_1 \cdot G_2 = \{g_1 g_2 : g_1 \in G_1,\, g_2 \in G_2\}.$$

Given $G \subseteq \mathbb{R}[\boldsymbol{x}]$, we define $G^0 = \{1\}$, and for $\ell \in \mathbb{N}_{>0}$,

$$G^\ell = \underbrace{G \cdot G \cdots G}_{\ell \text{ times}}, \quad \text{and} \quad G^{\le \ell} = \bigcup_{\ell'=0}^{\ell} G^{\ell'}.$$

We are now able to define a set of polynomials that are nonnegative on $S(G)$:

Proposition 2.11. *Let $K \subseteq \mathbb{R}[\boldsymbol{x}]$ be a cone containing only nonnegative polynomials (i.e., $h(\boldsymbol{x}) \ge 0$ for all $h \in K$, $\boldsymbol{x} \in \mathbb{R}^n$), and let $\ell \in \mathbb{N}$. Then, the set $\mathrm{ch}(K \cdot G^{\le \ell})$ contains only polynomials nonnegative on $S(G)$ (i.e., $g(\boldsymbol{x}) \ge 0$ for all $g \in \mathrm{ch}(K \cdot G^{\le \ell})$, $\boldsymbol{x} \in S(G)$).*

Example 2.12. Consider a set $G = \{g_1, g_2, g_3\}$ of polynomials in $\mathbb{R}[\boldsymbol{x}]$ and $K = \mathbb{R}_+$ the set of all nonnegative real numbers. The set $\mathrm{ch}(K \cdot G^{\le 2})$ contains all polynomials of the form $\lambda_0 + \lambda_1 g_1 + \lambda_2 g_2 + \lambda_3 g_1 g_2 + \lambda_4 g_2 g_3 + \lambda_5 g_1 g_3 + \lambda_6 g_1^2 + \lambda_7 g_2^2 + \lambda_8 g_3^2$ for $\lambda_0, \ldots, \lambda_8 \in \mathbb{R}_+$.

An example of cone K of nonnegative polynomials is the set of sum-of-squares (SOS) polynomials.

Definition 2.13 (Sum-of-Squares). A polynomial $h \in \mathbb{R}[\boldsymbol{x}]$ is a *sum-of-squares* if $h = \sum_{i=1}^m p_i^2$ for $m \in \mathbb{N}$ and $p_i \in \mathbb{R}[\boldsymbol{x}]$. The set of SOS polynomials is denoted by $\mathrm{SOS}[\boldsymbol{x}]$.

Remark 2.2. Not all nonnegative polynomials are SOS (e.g., the so-called "Motzkin Polynomial"). However, whereas verifying that a polynomial is nonnegative is NP-hard, SOS offer a practically efficient way to certify positivity of polynomials [31, 45].

Remark 2.3. Note that Prop. 2.11 provides a sufficient but not necessary condition for characterizing a set of positive polynomials over a BSA set $S(G)$. In general, for $K = \mathrm{SOS}[\boldsymbol{x}]$ and a finite set G, the set of polynomials $\mathrm{ch}(K \cdot G^{\le 1})$ is identical to Putinar's positivstellensatz, while $\mathrm{ch}(K \cdot G^{\le |G|})$ recalls a positivstellensatz by Schmüdgen [44].

Although we assume that K is a cone of nonnegative polynomials, we do not assume that K is finitely generated (e.g., $\mathrm{SOS}[\boldsymbol{x}]$ is not finitely generated). We also assume that $1 \in K$, which implies that $\mathbb{R}_+ \subseteq K$. We denote by $\mathcal{K}$ the set of cones K of nonnegative polynomials with $1 \in K$. For instance, $\mathrm{SOS}[\boldsymbol{x}] \in \mathcal{K}$.

3 FORWARD INVARIANCE FOR POLYNOMIAL SYSTEMS

In this section, we present a sufficient condition on a BSA set to be forward invariant for a polynomial dynamical system. Consider a BSA set $\mathcal{P} \doteq S(G)$ with $G = \{g_1, \ldots, g_m\} \subseteq \mathbb{R}[\boldsymbol{x}]$. Given $g \in G$, the *Lie derivative* of g along the field f is defined by $L_f(g)(\boldsymbol{x}) = \langle \nabla g(\boldsymbol{x}), f(\boldsymbol{x}) \rangle$. Concretely, $L_f(g)(\boldsymbol{x})$ gives the rate of change of the value of g at $\boldsymbol{x}$ along a trajectory of $\mathrm{ODE}(f)$.

The *idea* of the sufficient condition for $\mathcal{P} \doteq S(G)$ to be forward invariant is that when a trajectory inside $\mathcal{P}$ reaches a point $\boldsymbol{x}$ on the boundary, i.e, some $g_i(\boldsymbol{x}) = 0$, then its Lie derivative $L_f(g_i)$ at $\boldsymbol{x}$ should be nonnegative so that that g_i stays nonnegative. We say that $\mathrm{Boundary}_f(G)$ holds iff

$$\forall\, g \in G.\ \forall\, \boldsymbol{x} \in \mathcal{P}.\ \ g(\boldsymbol{x}) = 0 \implies L_f(g)(\boldsymbol{x}) \ge 0\,. \tag{1}$$

However, the $\mathrm{Boundary}_f$ condition (inspired by the theory of Lyapunov functions and barrier certificates [36]) does not imply forward invariance of $S(G)$ under f, in general. We need additional condition (e.g., those in Rem. 3.1 or (2) below). First, we note the following counterexample by Platzer [32].

Example 3.1. Consider the set $G = \{-x^2\}$ defining $\mathcal{P} \doteq S(G) = \{x : x^2 \le 0\} = \{0\}$ and $\mathrm{ODE}(f) : \dot{x} = -1$. Clearly, $\mathcal{P}$ is not forward invariant for $\mathrm{ODE}(f)$. However, whenever $x^2 = 0$ (that is, $x = 0$), we have $L_f(-x^2) = -2x\dot{x} = 2x \ge 0$, i.e., $\mathrm{Boundary}_f(\mathcal{P})$ holds. The reason is because $L_f(-x^2) = 2x$ cannot be expressed as a Lipshitz function of the polynomials in G (compare with (2) below).

Remark 3.1. Let us note that if the condition $L_f(g)(\boldsymbol{x}) \ge 0$ in (1) is changed to $L_f(g)(\boldsymbol{x}) > 0$, then we can avoid cases such as those mentioned above and prove soundness [13].

We refine the condition (1) using the notion of cone introduced before. This will have the double advantage of (i) ensuring invariance, and (ii) making the condition easier to verify/enforce numerically. We say that the predicate $\mathrm{Forward}_f(G; \lambda, K, \ell)$ holds iff

$$\forall\, g \in G.\ \ L_f(g) + \lambda g \in \mathrm{ch}(K \cdot G^{\le \ell}), \tag{2}$$

wherein $\lambda \in \mathbb{R}$, $K \in \mathcal{K}$ and $\ell \in \mathbb{N}$ are fixed.

Example 3.2. Consider the dynamics $\mathrm{ODE}(f)$ with $f(x) = x - x^3$, and let $G = \{g_1, g_2\}$ with $g_1 = x + 2$ and $g_2 = 2 - x$. We show that $\mathrm{Forward}_f(G; \lambda, K, \ell)$ holds for $\lambda = 2$, $K = \mathrm{SOS}[\boldsymbol{x}]$ and $\ell = 1$. Indeed, $L_f(g_1) + \lambda g_1 = -x^3 + 3x + 4 = (x+1)^2 \cdot (2 - x) + 2$, where $(x+1)^2, 2 \in \mathrm{SOS}[\boldsymbol{x}]$. The proof is similar for g_2; thus omitted.

Remark 3.2. If $\mathrm{Forward}_f(G; \lambda, K, \ell)$ holds then $\mathrm{Boundary}_f(G)$ holds, but not vice-versa. In Example 3.1 above, we have $\mathrm{Boundary}_f(G)$ but we can show that for any choice of $K \in \mathcal{K}$, $\lambda \in \mathbb{R}$ and $\ell \in \mathbb{N}$, $\mathrm{Forward}_f(G; \lambda, K, \ell)$ does not hold.

We now prove that the condition Forward is sufficient to ensure the forward invariance of $\mathcal{P}$. The proof is by relating the evolution of $g_1(\phi(t)), \ldots, g_m(\phi(t))$ over a trajectory ϕ and a nonlinear (internally) positive system. We briefly define such systems.

Definition 3.3. A dynamical system of the form $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ with state $\boldsymbol{x} \in \mathbb{R}^n$, input $\boldsymbol{u} \in \mathbb{R}^k$ and f Lipschitz continuous in $\boldsymbol{x}$ and $\boldsymbol{u}$ is *(internally) positive* if every trajectory $\phi : [0, T) \to \mathbb{R}^n$ of the system with $\phi(0) \in \mathbb{R}_+^n$ (and arbitrary input signal) satisfies that for all $t \in \mathrm{dom}(\phi)$, $\phi(t) \in \mathbb{R}_+^n$.

Theorem 3.4. *Let $F : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^n$ be Lipschitz continuous and satisfy that for all $\boldsymbol{x} \in \mathbb{R}_+^n$ and $\boldsymbol{u} \in \mathbb{R}^k$, $F(\boldsymbol{x}, \boldsymbol{u}) \in \mathbb{R}_+^n$. Let $\lambda \in \mathbb{R}$. The dynamical system $\dot{\boldsymbol{x}} = -\lambda \boldsymbol{x} + F(\boldsymbol{x}, \boldsymbol{u})$ is (internally) positive.*

This theorem is proved in [23, Theorem 1] using ideas explained in Arnold's textbook [10]. Song [47] proves it from Nagumo theorem [30] using the convexity of $\mathbb{R}_+^n$.

Proof. (Sketch) Using an argument based on Picard iteration, we first show that for any trajectory if $\phi(0) \in \mathbb{R}_+^n$, then there exists an interval $[0, \epsilon]$ such that $\phi(t) \in \mathbb{R}_+^n$ for $t \in [0, \epsilon]$. This is proved by induction on the Picard iterates that converge to the solution ϕ starting with $\phi^{(0)}(t) = \phi(0)$.

Assume that there exists $t \in \mathrm{dom}(\phi)$ such that $\phi(t) \notin \mathbb{R}_+^n$ and let t_* be the infimum of all such times t. By the continuity of ϕ and since $\phi(0) \in \mathbb{R}_+^n$, it follows that $\phi(t_*) \in \mathbb{R}_+^n$. Then, by using Picard iteration argument above, we conclude that there is $\epsilon > 0$ such that $\phi(t) \in \mathbb{R}_+^n$ for all $t \in [t_*, t_* + \epsilon]$. This is a contradiction with the definition of t_*, concluding the proof. ◻

Soundness of the Forward *condition:* We now proceed to prove the soundness of the Forward condition (2) for establishing forward invariance of a set $S(G)$ under a flow $\mathrm{ODE}(f)$.

Theorem 3.5. *If* $\mathrm{Forward}_f(G; \lambda, K, \ell)$ *holds, then* $\mathcal{P} \doteq S(G)$ *is forward invariant for* $\mathrm{ODE}(f)$.

Proof. Let ϕ be a trajectory of $\mathrm{ODE}(f)$ with $\phi(0) \in \mathcal{P}$. We will show that for all $g \in G$ and $t \in \mathrm{dom}(\phi)$, $g(\phi(t)) \geq 0$. To do this, consider the function $\boldsymbol{\omega} : \mathrm{dom}(\phi) \to \mathbb{R}^m$ defined by $\boldsymbol{\omega}(t) = (g_1(\phi(t)), \ldots, g_m(\phi(t)))$. It holds that $\boldsymbol{\omega}(0) \in \mathbb{R}_+^m$. Furthermore, for all $t \in \mathrm{dom}(\phi)$, $\dot{\boldsymbol{\omega}}(t) = (L_f(g_1)(\phi(t)), \ldots, L_f(g_m)(\phi(t)))$. Hence, $\mathrm{Forward}_f(G; \lambda, K, \ell)$ implies that for each $i \in [m]$, $L_f(g_i) + \lambda g_i = \sum_{j=1}^{s_i} h_{i,j} g_{i,j}$ for some $s_i \in \mathbb{N}$, $h_{i,j} \in K$ and $g_{i,j} \in G^{\leq \ell}$. It follows that $\boldsymbol{\omega}$ is a trajectory of the dynamical system

$$\dot{\boldsymbol{\omega}} = -\lambda \boldsymbol{\omega}(t) + F(\boldsymbol{\omega}, \phi(t)), \tag{3}$$

wherein $F(\boldsymbol{\omega}, \boldsymbol{x}) = (F_1(\boldsymbol{\omega}, \boldsymbol{x}), \ldots, F_m(\boldsymbol{\omega}, \boldsymbol{x}))$ satisfies that for all $i \in [m]$, $\boldsymbol{\omega} \in \mathbb{R}_+^m$ and $\boldsymbol{x} \in \mathbb{R}^n$, $F_i(\boldsymbol{\omega}, \boldsymbol{x}) \in \mathbb{R}_+$. Thus, by Theorem 3.4, (3) is a positive system and $\boldsymbol{\omega}(t) \in \mathbb{R}_+^m$ for all $t \in \mathrm{dom}(\phi)$. ◻

Finally, note that Forward is monotonic with respect to λ.

Proposition 3.6. *Let $K \in \mathcal{K}$, and $\lambda_1, \lambda_2 \in \mathbb{R}$ such that $\lambda_1 \leq \lambda_2$. It holds that* $\mathrm{Forward}_f(G; \lambda_1, K, \ell) \Rightarrow \mathrm{Forward}_f(G; \lambda_2, K, \ell)$.

Proof. Let us assume $\mathrm{Forward}_f(G; \lambda_1, K, \ell)$ holds. Then, for each $i \in [m]$, $L_f(g_i) + \lambda_1 g_i = \sum_{j=1}^{s_i} h_{i,j} g_{i,j}$ for some $s_i \in \mathbb{N}$, $h_{i,j} \in K$ and $g_{i,j} \in G^{\leq \ell}$. We have

$$\begin{aligned} L_f(g_i) + \lambda_2 g_i &= L_f(g_i) + \lambda_1 g_i + (\lambda_2 - \lambda_1) g_i \\ &= (\lambda_2 - \lambda_1) g_i + \sum_{j=1}^{s_i} h_{i,j} g_{i,j} = \sum_{j=1}^{s_i} h'_{i,j} g_{i,j} \end{aligned}$$

wherein $h'_{i,j} = h_{i,j} + (\lambda_2 - \lambda_1)$ if $g_{i,j} = g_i$ and $h'_{i,j} = h_{i,j}$ otherwise. Note that $h'_{i,j} \in K$ since $h_{i,j} \in K$, $\lambda_2 - \lambda_1 \in \mathbb{R}_+ \subseteq K$ and K is closed under addition since it is a cone. ◻

4 FORWARD INVARIANCE FOR CONSTRAINED AND SWITCHED POLYNOMIAL SYSTEMS

We extend the condition $\mathrm{Forward}_f(G; \lambda, K, \ell)$ to cover forward invariance for constrained and switched polynomial systems. First, we consider a constrained continuous-time dynamics

$$\mathrm{ODE}(f, \mathcal{D}) : \quad \dot{\boldsymbol{x}} = f(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{D},$$

wherein $\mathcal{D} = S(G_{\mathrm{dom}})$ is a BSA set with finite set $G_{\mathrm{dom}} \subseteq \mathbb{R}[\boldsymbol{x}]$.

Definition 4.1. A set $\mathcal{P} \subseteq \mathbb{R}^n$ is *forward invariant* for $\mathrm{ODE}(f, \mathcal{D})$ if for all trajectory ϕ of $\mathrm{ODE}(f, \mathcal{D})$ with $\phi(0) \in \mathcal{P}$, it holds that for all $t \in \mathrm{dom}(\phi)$, $\phi(t) \in \mathcal{P}$.

We say that $\mathrm{Forward}_{f,\mathcal{D}}(G; \lambda, K, \ell)$ holds iff

$$\forall g \in G.\ L_f(g) + \lambda g \in \mathrm{ch}(K \cdot (G \cup G_{\mathrm{dom}})^{\leq \ell}), \tag{4}$$

wherein $\lambda \in \mathbb{R}$, $K \in \mathcal{K}$ and $\ell \in \mathbb{N}$ are fixed. Comparing (4) with (2) for ODEs without constraints, we note that the set of polynomials G_{dom} is combined with G. The soundness of the condition "Forward" extends to constrained systems:

Theorem 4.2. *If* $\mathrm{Forward}_{f,\mathcal{D}}(G; \lambda, K, \ell)$ *holds, then* $\mathcal{P} \doteq S(G)$ *is forward invariant for* $\mathrm{ODE}(f, \mathcal{D})$.

Proof. The proof follows the same structure as Theorem 3.5. Again, we fix a trajectory ϕ of $\mathrm{ODE}(f, \mathcal{D})$ with $\phi(0) \in \mathcal{P}$. Since for all $t \in \mathrm{dom}(\phi)$, $\phi(t) \in \mathcal{D}$, we conclude that for all $h \in G_{\mathrm{dom}}$ and $t \in \mathrm{dom}(\phi)$, $h(\phi(t)) \geq 0$. Consider the function $\boldsymbol{\omega} : \mathrm{dom}(\phi) \to \mathbb{R}^m$ defined by $\boldsymbol{\omega}(t) = (g_1(\phi(t)), \ldots, g_m(\phi(t)))$. It holds that $\boldsymbol{\omega}(0) \in \mathbb{R}_+^m$. Furthermore, $\mathrm{Forward}_{f,\mathcal{D}}(G; \lambda, K, \ell)$ implies that for each $i \in [m]$, $L_f(g_i) + \lambda g_i = \sum_{j=1}^{s_i} h_{i,j} g_{i,j}$ for some $s_i \in \mathbb{N}$, $g_{i,j} \in G^{\leq \ell}$ and $h_{i,j} \in K \cdot G_{\mathrm{dom}}^{\leq \ell}$. Thus, the only change from (3) in the proof of Theorem 3.5 is that $F(\boldsymbol{\omega}, \boldsymbol{x})$ can be written in terms of polynomials $h_{i,j} \in K \cdot G_{\mathrm{dom}}^{\leq \ell}$. Since $h_{i,j}(\phi(t)) \geq 0$ for all $t \in \mathrm{dom}(\phi)$, we conclude that for all $\boldsymbol{\omega} \in \mathbb{R}_+^m$, $t \in \mathrm{dom}(\phi)$, $F(\boldsymbol{\omega}, \phi(t)) \in \mathbb{R}_+^m$. The rest of the proof is identical to that of Theorem 3.5. ◻

4.1 Forward Invariance for Switched Polynomial Systems

We recall switched dynamics of the form $\mathrm{ODE}(Q, f_{q \in Q}, \mathcal{D}_{q \in Q}, \mathcal{T})$ and their semantics from Section 2.2: Q is the finite set of modes and each mode $q \in Q$ is associated with a constrained polynomial system $\langle f_q, \mathcal{D}_q \rangle$. Furthermore, $\mathcal{T}$ is a finite set of transitions wherein each $\tau \in \mathcal{T}$ is a triple $\langle a_\tau, b_\tau, \mathcal{G}_\tau \rangle$ for pre-/post modes $a_\tau, b_\tau \in Q$ and guard set $\mathcal{G}_\tau = S(G_{\mathrm{guard},\tau})$.

We will consider a collection of sets $\mathfrak{P} = \{\mathcal{P}_q\}_{q \in Q}$ wherein for each $q \in Q$, $\mathcal{P}_q = S(G_q)$ for a finite set $G_q \subseteq \mathbb{R}[\boldsymbol{x}]$.

Definition 4.3 (Forward Invariance for Switched System). $\mathfrak{P}$ is a forward invariant for $\mathrm{ODE}(Q, f_{q \in Q}, \mathcal{D}_{q \in Q}, \mathcal{T})$ if for all trajectory $\langle \phi, q_{0:k}, t_{0:k+1} \rangle$ of $\mathrm{ODE}(Q, f_{q \in Q}, \mathcal{D}_{q \in Q}, \mathcal{T})$ with $\phi(0) \in \mathcal{P}_{q_0}$, it holds that for all $t \in \mathrm{dom}(\phi)$, $\phi(t) \in \mathcal{P}_{q_i}$ if $t \in [t_i, t_{i+1})$.

The condition $\mathrm{Forward}_{(Q, f_{q \in Q}, \mathcal{D}_{q \in Q}, \mathcal{T})}(\{G_q\}_{q \in Q}; \lambda, K, \ell)$ is:

(1) For each $q \in Q$, $\mathrm{Forward}_{f_q, \mathcal{D}_q}(G_q; \lambda, K, \ell)$ holds.

(2) For all $\tau \in \mathcal{T}$, we require that $\mathcal{P}_{a_\tau} \cap \mathcal{G}_\tau \subseteq \mathcal{P}_{b_\tau}$. For that, we enforce a sufficient condition over G_{a_τ} and G_{b_τ}:

$$G_{b_\tau} \subseteq \mathrm{ch}(K \cdot (G_{a_\tau} \cup G_{\mathrm{guard},\tau})^{\leq \ell}).$$

Theorem 4.4. *If* $\mathrm{Forward}_{\mathfrak{S}}(\{G_q\}_{q\in Q}; \lambda, K, \ell)$ *holds, then* $\mathfrak{P}$ *is forward invariant for* $\mathfrak{S} = \mathrm{ODE}(Q, f_{q\in Q}, \mathcal{D}_{q\in Q}, \mathcal{T})$.

Proof. Let $\langle \phi, q_{0:k}, t_{0:k+1}\rangle$ be a trajectory of $\mathfrak{S}$ with $\phi(t_0) \in \mathcal{P}_{q_0}$), and let $\tau_1, \ldots, \tau_k$ be the associated sequence of transitions. We will establish by induction the following two facts for all $i \in [k] \cup \{0\}$: (a) $\phi(t_i) \in \mathcal{P}_{q_i}$; and (b) for all times $t \in [t_i, t_{i+1})$, $\phi(t) \in \mathcal{P}_{q_i}$. These two facts will establish the forward invariance of $\mathfrak{P}$.

For the base case $i = 0$, we note that $\phi(t_0) \in \mathcal{P}_{q_0}$ by assumption. Since $\mathrm{Forward}_{\mathfrak{S}}(\{G_q\}_{q\in Q}; \lambda, K, \ell)$ implies $\mathrm{Forward}_{(f_{q_0}, \mathcal{D}_{q_0})}(G_{q_0})$, it follows by Theorem 4.2 that $\mathcal{P}_{q_0}$ is forward invariant for the mode q_0. Therefore, $\phi(t) \in \mathcal{P}_{q_0}$ for all $t \in [t_0, t_1)$. The base case is thus established.

Now, let us look at the case $i = 1$. By continuity of ϕ and because $\mathcal{P}_{q_0}$ is a closed set, we have that $\phi(t_1) \in \mathcal{P}_{q_0}$. Since $\phi(t_1) \in \mathcal{G}_{\tau_1}$, we have $\phi(t_1) \in \mathcal{P}_{q_0} \cap \mathcal{G}\tau_1$. Note that $\mathrm{Forward}_{\mathfrak{S}}(\{G_q\}_{q\in Q}; \lambda, K, \ell)$ implies that $\mathcal{P}_{q_0} \cap \mathcal{G}_{\tau_1} \subseteq \mathcal{G}_{q_1}$. Thus, $\phi(t_1) \in \mathcal{P}_{q_1}$. We can then conclude in the same as the base case that for all $t \in [t_1, t_2)$, $\phi(t) \in \mathcal{P}_{q_1}$. The case $i = 1$ is thus established.

The proof for the cases $i > 1$ is identical. □

We will now turn our attention to computing such forward invariants for a given system with initial conditions.

5 REFINEMENT OPERATORS AND FIXED POINT FORMULATION

Given a polynomial system $\langle f, \mathcal{I}\rangle$, we wish to compute a BSA set $\mathcal{P} = S(G)$ for a finite set of polynomials $G \subseteq \mathbb{R}[\boldsymbol{x}]$ such that $\mathcal{P} \supseteq \mathcal{I}$ (the initial set is contained) and $\mathcal{P}$ is forward invariant for $\mathrm{ODE}(f)$. Note that we do not explicitly enforce in this paper that $\mathcal{P} \cap \mathcal{S}_{\mathrm{unsafe}} = \emptyset$, as is common in many approaches to invariant synthesis based on abstract interpretation. Explicit use of $\mathcal{S}_{\mathrm{unsafe}}$ (example for early termination) will be considered in future work.

Our approach is based on using the framework of abstract interpretation, first introduced by Cousot and Cousot to compute invariants for programs [14, 15]. We will present a similar approach to compute polynomial invariants for polynomial systems. The first step is to define the invariant we week as a pre-fixed point G_* of a monotone operator on the space of cones in $\mathbb{R}[\boldsymbol{x}]$. The set $S(G_*)$ will give us the invariant set $\mathcal{P}$.

Let $G \subseteq \mathbb{R}[\boldsymbol{x}]$ be a finite set of polynomials. Recall that $\mathrm{ch}(G)$ contains all the conic combinations of elements in G. Let us fix a cone $K \in \mathcal{K}$ of nonnegative polynomials. We define a refinement operator that takes us from $\mathrm{ch}(G)$ to a new cone $\mathrm{ch}(G')$.

Definition 5.1 (Refinement Operator). Given $G = \{g_1, \ldots, g_m\} \subseteq \mathbb{R}[\boldsymbol{x}]$, we define the *refinement* of G as follows:

$$\partial_f(G; \lambda, K, \ell) = \{g \in \mathrm{ch}(G) : L_f(g) + \lambda g \in \mathrm{ch}(K \cdot G^{\leq \ell})\}.$$

First, note that $\partial_f(G; \lambda, K, \ell)$ is a cone (one can easily show that it satisfies the axioms of Def. 2.9). Furthermore, as a corollary of Theorem 3.5, it holds that any finitely generated pre-fixed point of the refinement operator is a forward invariant set.

Algorithm 1: Fixed-Point Iterations

Data: BSA set $\mathcal{I} = S(G_{\mathrm{init}})$, $\lambda \in \mathbb{R}$, $K \in \mathcal{K}$, $\ell \in \mathbb{N}$.
Result: (Possibly trivial) forward invariant $\mathcal{P}$ for $\mathrm{ODE}(f)$ such that $\mathcal{I} \subseteq \mathcal{P}$.

1 Let $G_0 \subseteq \mathbb{R}[\boldsymbol{x}]$ be a finite set such that $\mathcal{I} \subseteq S(G_0)$
2 **for** $\sigma = 0, 1, \ldots$ **do**
3   **if** $\mathrm{IsFixedPoint}(G_\sigma; \lambda, K, \ell)$ **then return** G_σ
4   **else** let $G_{\sigma+1} = \mathrm{FiniteRefinement}(G_\sigma; \lambda, K, \ell)$

Corollary 5.2. *Let* $G \subseteq \mathbb{R}[\boldsymbol{x}]$ *be finite and satisfy that* $G \subseteq \partial_f(G; \lambda, K, \ell)$*. Then,* $S(G)$ *is forward invariant.*

Finally, as a corollary of Prop. 2.11, it holds that if $S(G)$ includes a set $\mathcal{S}$, then so does $S(G')$ for any finite G' in the refinement.[1]

Corollary 5.3. *Let* $G, G' \subseteq \mathbb{R}[\boldsymbol{x}]$ *be finite sets and satisfy that* $G' \subseteq \partial_f(G; \lambda, K, \ell)$*. Then,* $S(G) \subseteq S(G')$*. In particular, if* $\mathcal{I} \subseteq S(G)$*, then* $\mathcal{I} \subseteq S(G')$.

Using the refinement operator and Corollaries 5.2 and 5.3, we would apply the refinement operator iteratively: $G_{i+1} = \partial_f(G_i; \lambda, K, \ell)$ until $G_{j+1} = G_j$ for some $j \in \mathbb{N}$, wherein the initial iterate G_0 satisfies $\mathcal{I} \subseteq S(G_0)$. However, we face two problems: (a) the refined set $\partial_f(G; \lambda, K, \ell)$ need not be finitely generated even if G is finite; and (b) the process is not guaranteed to terminate in finitely many steps even if we managed to keep the iterates G_i finite. We will address both issues using ideas from abstract interpretation theory, which has been widely used in static analysis of programs [14] as well as dynamical systems [38, 41]. Algo. 1 presents a high level view of the approach: the procedure $\mathrm{IsFixedPoint}(G; \lambda, K, \ell)$ checks whether $G \subseteq \partial_f(G; \lambda, K, \ell)$, whereas $\mathrm{FiniteRefinement}(G; \lambda, K, \ell)$ extracts a finite subset from $\partial_f(G; \lambda, K, \ell)$ in a manner that will guarantee termination of this process in finitely many steps.

The choice of the initial iterate (line 1) will be discussed in Sec. 5.2 and how to guarantee termination of the above procedure will be discussed in Sec. 5.3.

Implementation for finitely generated K. As a warm-up, we start with a specific case for which $\mathrm{FiniteRefinement}(G; \lambda, K, \ell)$ can be implemented in a *lossless* way, meaning that its output generates $\partial_f(G; \lambda, K, \ell)$.[2] This comes from the observation that if K is finitely generated, then $\partial_f(G; \lambda, K, \ell)$ is finitely generated as well. Examples of finitely generated cones of positive definite polynomials include $K = \mathbb{R}_+$ (the set of nonnegative reals) and the cone of *diagonally dominant* SOS polynomials introduced by Ali Ahmadi et al [1].

Lemma 5.4. *Assume that* K *is finitely generated. Then, for any finite set* $G \subseteq \mathbb{R}[\boldsymbol{x}]$, $\partial_f(G; \lambda, K, \ell)$ *is finitely generated, i.e., there is a finite set* $G' \subseteq \mathbb{R}[\boldsymbol{x}]$ *such that* $\partial_f(G; \lambda, K, \ell) = \mathrm{ch}(G')$.

Proof. Let $G = \{g_1, \ldots, g_m\}$, and let $H = \{h_1, \ldots, h_s\} \subseteq \mathbb{R}[\boldsymbol{x}]$ be such that $K = \mathrm{ch}(H)$. Note that any element of $\mathrm{ch}(G)$ can be written as $\sum_{i=1}^m \alpha_i g_i$ for multipliers $\alpha_i \geq 0$ and any element of K can

[1] The property extends straightforwardly to infinite sets G' but we focus here on finite sets G', because this is sufficient for our needs and the operator $S(\cdot)$ is defined only for finite sets.

[2] This result is presented for completeness, but as we will see the resulting implementation becomes rapidly intractable, so that a different approach will be proposed in Sec. 5.1.

Algorithm 2: IsFixedPoint using Projections

Data: Fine set $G \subseteq \mathbb{R}[\boldsymbol{x}]$, $\lambda \in \mathbb{R}$, $K \in \mathcal{K}$, $\ell \in \mathbb{N}$.

Let $H = \partial_f(G; \lambda, K, \ell)$

if *for all* $g \in G$, $\mathrm{Proj}(g; H) = g$ **then return** True

else return False

be written as $\sum_{i=1}^{s} \alpha_i h_i$ for multipliers $\alpha_i \geq 0$. Also, note that $G^{\leq \ell}$ is finite, i.e., $G^{\leq \ell} = \{\hat{g}_1, \ldots, \hat{g}_k\}$. Any element of $K \cdot G^{\leq \ell}$ can thus be written as $(\sum_{i=1}^{s} \beta_i h_i)\hat{g}_j$ for multipliers $\beta_i \geq 0$. It follows that any element of $\mathrm{ch}(K \cdot G^{\leq \ell})$ can be written as $\sum_{j=1}^{k} \left(\sum_{i=1}^{s} \beta_{i,j} h_i\right) \hat{g}_j$ for multipliers $\beta_{i,j} \geq 0$. Hence, the condition that $p \in \mathrm{ch}(G)$ and $L_f(p) + \lambda p \in \mathrm{ch}(K \cdot G^{\leq \ell})$ can be written as two equality constraints that are linear in the variables α_i and $\beta_{i,j}$. This, plus the nonnegativity constraints $\alpha_i \geq 0$ and $\beta_{i,j} \geq 0$, defines a polyhedral cone P over these variables. Now, the projection P' of P over the m variables α_i is also a polyhedral cone. Hence, P' is finitely generated. Finally, since $\partial_f(G; \lambda, K, \ell) = \{\sum_{i=1}^{m} \alpha_i g_i : (\alpha_1, \ldots, \alpha_m) \in P'\}$, it holds that $\partial_f(G; \lambda, K, \ell)$ is finitely generated, concluding the proof. □

The constructive proof of Lemma 5.4 provides a way of implementing IsFixedPoint and FiniteRefinement when K is finitely generated. However, working with finitely generated cones K is impractical for two reasons: (i) Using a small set of generators H often leads to an overly conservative refinement operator, thereby preventing us from proving invariance even for simple systems; (ii) The number of polynomials in G_σ grows extremely fast (super-exponential in the number of iterations in the worst case), thereby preventing from applying more than a few iteration.

Therefore, in the next section, we consider a different approach using the SOS cone and wherein the size of G_σ is kept constant.

5.1 Bounded-Size Iterates

In this approach, we let K be the set of SOS polynomials of degree at most $2d$, denoted by $\mathrm{SOS}_d[\boldsymbol{x}]$. Note that $\mathrm{SOS}_d[\boldsymbol{x}]$ is *not* finitely generated. The idea of the approach is to compute a finite set $G' = \{g'_1, \ldots, g'_m\}$ included in $\partial_f(G; \lambda, K, \ell)$, wherein $G = \{g_1, \ldots, g_m\}$. Note that $|G'| = |G|$.

Sampling from a convex set is a problem that has received some attention in the literature. We can for instance mention the *hit-and-run* algorithm which is a Monte-Carlo method to sample random points inside a given set [12, 27]. However, in this work, we consider another approach based on projections.

Concretely, let $\|\cdot\|$ be a norm on $\mathbb{R}[\boldsymbol{x}]$. Given a subset $H \subseteq \mathbb{R}[\boldsymbol{x}]$, define the *projection* on H by $\mathrm{Proj}(g; H) = \arg\min_{h \in H} \|h - g\|$. The key insight is that using sum-of-squares (SOS) optimization, we can compute $\mathrm{Proj}(g; H)$ where $H = \partial_f(G; \lambda, K, \ell)$ when G is finitely generated set of polynomials and $K = \mathrm{SOS}_d[\boldsymbol{x}]$. Using the projection operator allows us to implement IsFixedPoint and FiniteRefinement as shown in Algos. 2 and 3. The IsFixedPoint procedure simply checks that $\mathrm{Proj}(g_i; H) = g_i$ for each $g_i \in G$. Similarly, the FiniteRefinement procedure computes the set $G' = \{\mathrm{Proj}(g_i; H) \mid g_i \in G\}$ wherein $G = \partial_f(G; \lambda, K, \ell)$.

Note that $H = \partial_f(G; \lambda, K, \ell)$ in Algos. 2 and 3 is a convex set that can be represented by Linear Matrix Inequalities [24, 31]. Hence,

Algorithm 3: FiniteRefinement using Projections

Data: $G = \{g_1, \ldots, g_m\} \subseteq \mathbb{R}[\boldsymbol{x}]$, $\lambda \in \mathbb{R}$, $K \in \mathcal{K}$, $\ell \in \mathbb{N}$.

1 Let $H = \partial_f(G; \lambda, K, \ell)$

2 **for** $i = 1, \ldots, m$ **do** let $g'_i = \mathrm{Proj}(g_i; H)$

3 **return** $\{g'_1, \ldots, g'_m\}$

Algorithm 4: FiniteRefinement using Robust Projections

Data: $G = \{g_1, \ldots, g_m\} \subseteq \mathbb{R}[\boldsymbol{x}]$, $\epsilon > 0$, $\lambda \in \mathbb{R}$, $K \in \mathcal{K}$, $\ell \in \mathbb{N}$.

Let $H = \partial_f(G, \epsilon; \lambda, K, \ell)$

`/* Same as lines 2-5 in Algo. 3 */`

computing $\mathrm{Proj}_S(g_i)$ can be done efficiently, e.g., using semidefinite programming [11].

As mentioned before, computing $G_{\sigma+1}$ as the output of Algo. 3 is advantageous because it keeps the size of G_σ constant throughout the process. However, it might be slow to make progress, so that a large number of iterations might be needed before finding invariant (if we eventually find one). Another limitation of the approach is the sensitivity to numerical errors. Indeed, when using a numerical solver to compute $\mathrm{Proj}(g; H)$, we may get something close to but not exactly in H. In this case, we cannot certify that a "numerical" fixed point is an actual fixed point. To address these limitations, we define a robust version of the above approach.

5.1.1 Robust Projection and Acceleration. The robust projection relies on an inner-approximation of $\partial_f(G; \lambda, K, \ell)$ parameterized by a robustness parameter $\epsilon > 0$:

$$\partial_f(G, \epsilon; \lambda, K, \ell) = \{g \in \mathrm{ch}(G) : L_f(g) + \lambda g - \epsilon\|g\| \in \mathrm{ch}(K \cdot G^{\leq \ell})\}.$$

The resulting implementation of FiniteRefinement is the same as in Algo. 3 but with $H = \partial_f(G, \epsilon; \lambda, K, \ell)$; see Algo. 4.

The main result of this section is that if the output of Algo. 4 is close to G, then $S(G)$ is forward invariant.

Lemma 5.5. *Assume that $S(G)$ is compact and let $\epsilon > 0$. Denote $H = \partial_f(G, \epsilon; \lambda, K, \ell)$. There exists a constant $\kappa > 0$ depending only on G, f, λ and ϵ, such that for all $g \in G$, if $\|g - \mathrm{Proj}(g; H)\| \leq \kappa \|g\|$, then $S(G)$ is forward invariant.*

Proof. Let $D : \mathbb{R}[\boldsymbol{x}] \to \mathbb{R}[\boldsymbol{x}]$ be defined by $D(g) = L_f(g) + \lambda g$, and let $\delta = \max\{|D(g)(\boldsymbol{x})| : \boldsymbol{x} \in S(G),\ \|g\| \leq 1\}$. Without loss of generality, let $g \in G$ be such that $\|g\| = 1$, and assume that there is $g' \in H$ such that $\|g - g'\| \leq \kappa$ wherein we choose $\kappa < \frac{\epsilon}{\epsilon + \delta}$. Let us set $\kappa(G) = \kappa$. First, note that this implies that $\|g'\| \geq 1 - \kappa$. Second, by definition of δ, we note that $|D(\frac{g - g'}{\kappa})| \leq \delta$ since $\|g - g'\| \leq \kappa$. $D(\cdot)$ being a linear operator, we obtain $D(g) \geq D(g') - \delta\kappa$ on $S(G)$. Finally, since $g' \in H$, it holds that $D(g') \geq \epsilon \|g'\|$ on $S(G)$. Hence, $D(g') \geq \epsilon - \epsilon\kappa$ on $S(G)$. This implies that $D(g) \geq \epsilon - \epsilon\kappa - \delta\kappa$ on $S(G)$. By our choice of κ, it follows that $D(g) > 0$ on $S(G)$. Hence, $S(G)$ is forward invariant. □

Hence, in our algorithmic process, if at some point G' is "close enough" to G, then we stop the algorithm and return G. However, estimating the value of κ, in practice, is hard. We will simply set it to be a constant smaller than ϵ, typically $\frac{\epsilon}{10}$, and use an SMT solver such as Z3 to validate the final positive invariant.

Algorithm 5: Initial iterate using Sample Points

Data: Finite set $G_{\text{init}} \subseteq \mathbb{R}[\boldsymbol{x}]$, $K \in \mathcal{K}$, template $\{p_1, \ldots, p_k\} \subseteq \mathbb{R}[\boldsymbol{x}]$, sample points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^n$.

1 Let $\mathcal{B} = \left\{ \sum_{i=1}^{k} \alpha_j p_j : \alpha_j \in \mathbb{R} \right\}$
2 Let $\mathcal{G} = \left\{ g \in \mathcal{B} : g(\boldsymbol{x}_i) \geq 0,\ i = 1, \ldots, N \right\}$
3 Let $G_{-1} \subseteq \mathbb{R}[\boldsymbol{x}]$ be a finite set such that $\mathcal{G} = \text{ch}(G_{-1})$
4 Let $G_0 = \{\text{Proj}(g; \text{ch}(K \cdot G_{\text{init}})) : g \in G_{-1}\}$

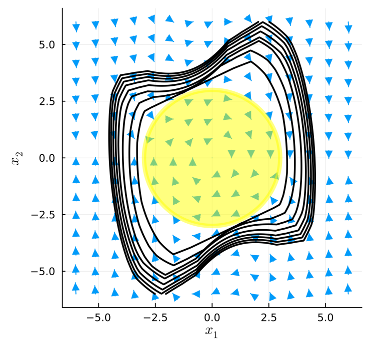


Figure 2: The sets $S(G_\sigma)$ for $\sigma = 0, \ldots, 6$ for the Vanderpol oscillator from Example 2.4 and the template in Example 5.7. The inner most curve corresponds to $\sigma = 0$, then $\sigma = 1$, etc.

5.2 Initial Iterate from Simulations

The only constraint we have on the initial iterate G_0 (line 1) is that $\mathcal{I} \subseteq S(G_0)$. Hence, one could for instance just set $G_0 = G_{\text{init}}$. However, the more functions in G_0, the more expressive the subsequent iterates since they are all subsets of $\text{ch}(G_0)$. This is illustrated in the example below.

Example 5.6. Let $G_{\text{init}} = \{1 - x_1^2 - x_2^2\}$, so that $\mathcal{I}$ is the ball of radius one around the origin. If $G_0 = G_{\text{init}}$, then all sets G_σ are subsets of $\text{ch}(G_{\text{init}}) = \{\alpha + \beta - \alpha x_1^2 - \alpha x_2^2 : \alpha, \beta \geq 0\}$. Hence, the iterates can only describe balls of radius at least one around the origin. However, for instance, if we let $G_0 = \{1 - x_1^2, 1 - x_2^2, 1 - 2x_1x_2\}$, then we still have that $\mathcal{I} \subseteq S(G_0)$. However, the iterates can describe a larger variety of sets such as ellipsoids.

In order to define a rich initial iterate, we use sample points from simulations. Then, we define the initial iterate as the set of all polynomials *in a given template* that are nonnegative at the sample points. Finally, we project on $\text{ch}(K \cdot G_{\text{init}})$. This is implemented in Algo. 5. Note that $\mathcal{B}$ is a linear subspace; then, $\mathcal{G}$ is a subset of $\mathcal{B}$ obtained by imposing linear inequality constraints; hence, $\mathcal{G}$ is a finitely generated cone; this allows to compute G_{-1} in line 3.

Example 5.7. Consider the system of Example 2.4. The sample points are given by the trajectories in Fig. 1. We use $\{1, x_1^2, x_1x_2, x_2^2\}$ as template to allow any homogeneous quadratic curves. The first seven iterates of the overall procedure are depicted in Fig. 2.

5.3 Finite Termination

Finitely, we discuss the termination of the algorithm. Unfortunately, the approach in Sec. 5.1 does not guarantee that the iteration will converge in finite time to a fixed point. Therefore, we introduce

Algorithm 6: FiniteRefinement with Widening

Data: $G = \{g_1, \ldots, g_m\} \subseteq \mathbb{R}[\boldsymbol{x}]$, $\lambda \in \mathbb{R}$, $K \in \mathcal{K}$, $\ell \in \mathbb{N}$.

1 Let $H = \partial_f(G; \lambda, K, \ell)$
2 Let $G' = \emptyset$
3 **for** $i = 1, \ldots, m$ **do**
4   **if** $\text{Proj}(g_i; H) = g_i$ **then** Add g_i to G'
5 **return** G'

a widening of the refinement operator, that can be applied only a finite number of times before obtaining the empty set, thereby ensuring termination of the algorithm in finite time.

The widened operator removes the generators that are not in the refinement. This is implemented in Algo. 6. Note that the condition in line 4 can be checked efficiently, e.g., if $K = \text{SOS}_d[\boldsymbol{x}]$ (see Sec. 5.1). It holds that the output G' of Algo. 6 has cardinality strictly lower than the input G if G is not a fixed point.

Lemma 5.8. *If G is not a fixed point, i.e., $G \not\subseteq \partial_f(G; \lambda, K, \ell)$, then the output G' of Algo. 6 satisfies $|G'| < |G|$.*

Proof. If $G \not\subseteq \partial_f(G; \lambda, K, \ell)$, then there is some $g_i \in G$ such that $\text{Proj}(g_i; H) \neq g_i$ so that $g_i \notin G'$, concluding the proof. □

As a corollary, Algo. 6 can be applied recursively at most $|G_0|$ times before reaching a (possibly trivial) fixed point. Note that the widening operator is applied only after several iterations of the less conservative projection-based refinement operator is applied.

6 NUMERICAL EXPERIMENTS

We applied the algorithmic process from Sec. 5 to compute invariant BSA sets for several polynomial systems.

Implementation Details. We implemented the algorithm in Julia. To compute the projections in Secs. 5.1 and 5.2, we used the package SumOfSquares.jl [53] with the SDP solver Mosek [9]. To compute a finite set of initial iterates in Sec. 5.2, we used Polyhedra.jl [25] with CDDLib [18]. We used only the robust projection approach (Sec. 5.1.1); in particular, no widening as in Sec. 5.3 was needed. The parameters we used for the robust projection were $\lambda = 1$, $K = \text{SOS}_d[\boldsymbol{x}]$ with d inferred automatically by the solver, $\ell = 1$, $\epsilon = 0.1$ and $\kappa = 10^{-8}$. We believe that ϵ is sufficiently large and κ sufficiently small to ensure sound invariants despite possible numerical inaccuracies inherent to SDP solvers. However, a rigorous analysis of the robustness to numerical errors is beyond the scope of this paper. Therefore, whenever possible, we verified the returned invariant using the SMT solver Z3 [17]. We compared our approach, which uses multiple polynomials $g_1, \ldots, g_m$, with the one using a single polynomial g possibly of higher degree.

Results. All computations were made on a laptop with processor Intel Core i7-7600u and 16GB RAM running Windows. The timing results, number of iterations and number of polynomials in the invariants for the different numerical experiments are reported in Table. 1. The invariants are reported in the Appendix.

Table 1: Results from numerical experiments. *: a modification of the invariant was verified. T/O: Time Out (>6 hours).

Experiment	Time	# Iterations	# Invariants	Z3
Sec. 6.1	50 sec.	94	$8 \longrightarrow 3$	Valid*
Sec. 6.2 (H_1)	<1 sec.	8	1	Valid
Sec. 6.2 (H_2)	50 sec.	17	11	Valid
Sec. 6.3 (H_1)	1 sec.	4	12	Valid
Sec. 6.3 (H_2)	12 sec.	3	1	Valid
Sec. 6.4 (H_1)	2.5 sec.	8	11	Valid
Sec. 6.4 (H_2)	152 sec.	8	$15 \longrightarrow 11$	Valid*
Sec. 6.5	5.5 sec.	4	$7 \longrightarrow 1$	Valid*
Sec. 6.6	372 sec.	6	16	T/O
Sec. 6.7	110 sec.	3 + 3	41 + 4	Valid
Sec. 6.8	115 sec.	5 + 9 + 19	8 + 10 + 1	Valid

6.1 Vanderpol Oscillator

We finish the example of the Vanderpol oscillator (Example 2.4). The algorithm generated a BSA set described by 10 polynomials, depicted in Fig. 3a. We tried to verify this invariant using Z3, but the SMT solver timed out (>6 hours) before returning an output (so that we could not validate or invalidate the invariant). However, we manually picked a subset of three polynomials from the invariant, and this time Z3 was able to verify the invariant. This new invariant is depicted in Fig. 3a as well. We synthesized a forward invariant described by a single SOS polynomial g. For degree $d \in \{2, 4, 6\}$, no such g could be found. For $d = 8$, the solver returned the polynomial given in the Appendix and depicted in Fig. 3a. However, due to the high degree of the polynomial, Z3 timed could not verify the polynomial within the 12 hour timeout.

6.2 Stable 2D Nonlinear System

We consider the dynamical system given by

$$\dot{x_1} \;=\; -\frac{1}{2}x_1^3 + 2x_2, \quad \dot{x_2} \;=\; -2x_2\,.$$

with initial set $\mathcal{I} = \{x_1^2 + x_2^2 \le \frac{1}{4}\}$, depicted in Fig. 3b. We considered two templates to compute an invariant set for this system: $H_1 = \{1, x_1^2, x_2^2\}$ and $H_2 = \{1, x_1^2, x_1x_2, x_2^2\}$. The invariants obtained by the algorithm using each template are represented in Fig. 3b. Both invariants were verified using Z3. A larger template provides a stronger invariant, but requires a longer computation time. We synthesized a forward invariant described by a single SOS polynomial g with degree 4 (given in the Appendix and depicted in Fig. 3b). Z3 was able to verify the invariant. As seen in Fig. 3b, the forward invariant is larger than those given by our approach.

6.3 System from [2, Example 6]

We consider the dynamical system given by

$$\dot{x_1} \;=\; -x_1^3 + \frac{1}{2}x_2, \quad \dot{x_2} \;=\; -x_1 - 2x_2\,,$$

with initial set $\mathcal{I} = \{x_1^2+x_2^2 \le \frac{1}{4}\}$, depicted in Fig. 3c. We considered two templates to compute an invariant set for this system: $H_1 = \{1, x_1, x_2\}$ and $H_2 = \{1, x_1^2, x_1x_2, x_2^2\}$. The invariants obtained by the algorithm using each template are represented in Fig. 3c. Both invariants were verified using Z3.

6.4 Unstable 2D Nonlinear System

We consider the dynamical system given by

$$\dot{x_1} \;=\; 1 - x_1^3 + x_2^3, \quad \dot{x_2} \;=\; -\frac{1}{2} + x_1^3 - x_2\,,$$

with initial set $\mathcal{I} = \{x_1^2+x_2^2 \le \frac{1}{4}\}$, depicted in Fig. 3d. We considered two templates to compute an invariant set for this system: $H_1 = \{1, x_1, x_2\}$ and $H_2 = \{1, x_1^2, x_1x_2, x_2^2\}$. The invariants obtained by the algorithm using each template are represented in Fig. 3d. Note that in this case, the invariants are not compact. Thus, Lemma 5.5 does not apply. Hence, it is important to verify the invariants with Z3. The first invariant (with H_1) was easily verified using Z3. However, the verification of the second invariant (with H_2) using Z3 timed out. Nevertheless, by manually removing four polynomials from the invariant, we obtained a new set (depicted in Fig. 3d), for which we could prove the invariance with Z3. We synthesized a forward invariant described by a single SOS polynomial g. For degree $d \in \{2, 4\}$, no such g could be found. For $d = 6$, the solver returned such a g (given in the Appendix and depicted in Fig. 3d). However, due to the high degree of the polynomial, Z3 timed out in the verification of the invariant (>12 hours).

6.5 3D Nonlinear System

We consider the dynamical system, inspired by [2, Example 7]:

$$\begin{cases} \dot{x_1} \;=\; 1 - x_1x_3^2 - x_1^3 + x_1x_3^4 - x_1^3x_3^2, \\ \dot{x_2} \;=\; -x_2 - x_2x_3^2 - x_1^2x_2 - x_1^2x_2x_3^2, \\ \dot{x_3} \;=\; -4x_3 - x_3^3 + 3x_1^2x_3 + 3x_1^2x_3^3, \end{cases}$$

with initial set $\mathcal{I} = \{x_1^2+x_2^2+x_3^2 \le \frac{1}{4}\}$. We considered the template $H = \{1, x_1^2, x_2^2, x_3^2\}$. The algorithm generated a BSA set described by 7 polynomials. The verification using Z3 timed out. However, by manually selecting and keeping only one polynomial, we obtained a larger set, for which we could verify the invariance with Z3.

6.6 4D Nonlinear System

We consider the dynamical system, inspired by [2, Example 9]:

$$\begin{cases} \dot{x_1} \;=\; -x_1 - 3x_3x_4 + x_2^3, \quad \dot{x_2} \;=\; \frac{1}{2} - x_1 - x_2^3, \\ \quad \dot{x_3} \;=\; -x_3 + x_1x_4, \quad \dot{x_4} \;=\; x_1x_3 - x_4^3, \end{cases}$$

with initial set $\mathcal{I} = \{x_1^2 + x_2^2 + x_3^2 + x_4^2 \le \frac{1}{4}\}$. We considered the template $H = \{1, x_1^2, x_2^2, x_3^2, x_4^2\}$. The algorithm generated a BSA set described by 16 polynomials. The verification using Z3 timed out.

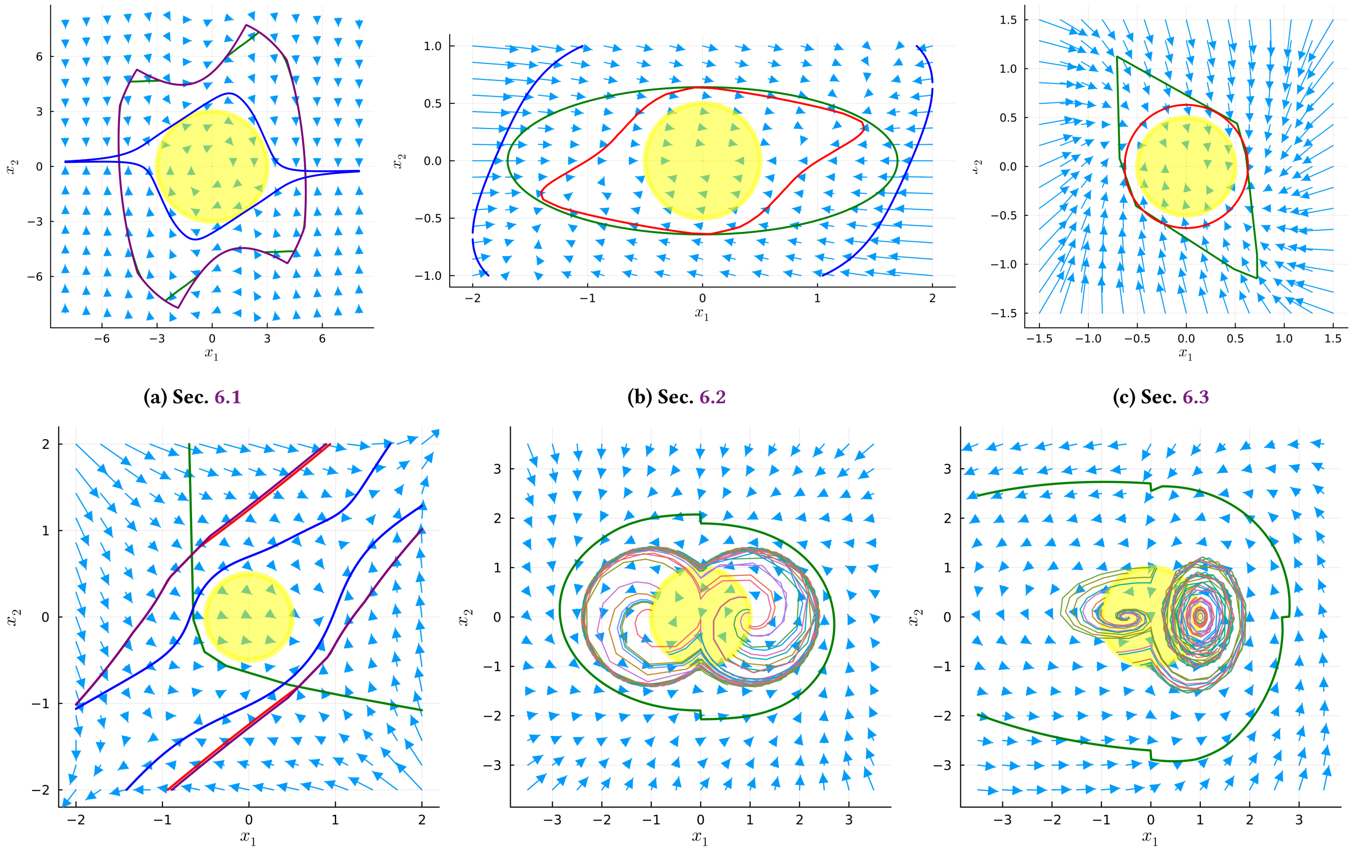
6.7 Switched System with Limit Cycle

We consider the switched dynamical system

$$\begin{cases} \dot{x_1} \;=\; -\frac{1}{4} - x_2 - \frac{1}{4}x_1 + \frac{1}{4}x_2^2 + \frac{3}{4}x_1^2 - \frac{1}{4}x_1x_2^2 - \frac{1}{4}x_1^3, \\ \dot{x_2} \;=\; -1 + \frac{1}{4}x_2 + x_1 + \frac{1}{2}x_1x_2 - \frac{1}{4}x_2^3 - \frac{1}{4}x_1^2x_2, \end{cases} \quad \text{if } x_1 \ge 0.$$

$$\begin{cases} \dot{x_1} \;=\; \frac{1}{4} - x_2 - \frac{1}{4}x_1 - \frac{1}{4}x_2^2 - \frac{3}{4}x_1^2 - \frac{1}{4}x_1x_2^2 - \frac{1}{4}x_1^3, \\ \dot{x_2} \;=\; 1 + \frac{1}{4}x_2 + x_1 + \frac{1}{2}x_1x_2 - \frac{1}{4}x_2^3 - \frac{1}{4}x_1^2x_2, \end{cases} \quad \text{if } x_1 \le 0.$$

with initial set $\mathcal{I} = \{x_1^2 + x_2^2 \le \frac{1}{4}\}$, depicted in Fig. 3e. We considered the template $H = \{1, x_2, x_1^2, x_2^2\}$. Our approach to compute an invariant for this system was: (i) compute an invariant $\mathcal{P}_1$ for the



(a) Sec. 6.1 **(b) Sec. 6.2** **(c) Sec. 6.3**

(d) Sec. 6.4 **(e) Sec. 6.7** **(f) Sec. 6.8**

Figure 3: *Green:* invariant, or invariant using H_1 (if applicable). *Red:* invariant using H_2 (if applicable). *Purple:* invariant obtained by picking some polynomials (if applicable). *Blue:* invariant obtained with a single SOS polynomial (if applicable).

first mode containing $\mathcal{I}$; (ii) compute the intersection $\mathcal{I}_{12}$ between $\mathcal{P}_1$ and the guard $\mathcal{G}_{12}$ from the first to the second mode, which is a BSA set; (iii) compute an invariant $\mathcal{P}_2$ for the second mode containing $\mathcal{I}$ and $\mathcal{I}_{12}$; (iv) compute the intersection $\mathcal{I}_{21}$ between $\mathcal{P}_2$ and the guard $\mathcal{G}_{21}$ from the second to the first mode, which is a BSA set; (v) check that $\mathcal{I}_{21}$ is contained in $\mathcal{P}_1$; since it was the case, we stopped. The resulting invariant was verified using Z3.

6.8 Switched Bistable System

We consider the switched dynamical system

$$\left\{\begin{array}{lcl} \dot{x_1} &=& 0.1 - x_2 - 0.3x_1 + 0.3x_1^2 - 0.1x_1^3, \\ \dot{x_2} &=& -2 + 2x_1 - 0.1x_2^3, \end{array}\right. \quad \text{if } x_1 \geq 0.$$

$$\left\{\begin{array}{lcl} \dot{x_1} &=& -0.125 - 2x_2 - 0.25x_1, \\ \dot{x_2} &=& 0.25 - 0.25x_2 + 0.5x_1, \end{array}\right. \quad \text{if } x_1 \leq 0.$$

with initial set $\mathcal{I} = \{x_1^2 + x_2^2 \leq \frac{1}{4}\}$, depicted in Fig. 3f. We considered the template $H = \{1, x_1^2, x_1x_2, x_2^2\}$. We split the first mode into two "sub-"modes, one for $x_2 \geq 0$ and one for $x_2 \leq 0$ and applied the procedure described in Sec. 6.7 to compute invariants that were successfully verified using Z3.

7 CONCLUSIONS

We provided an algorithmic framework to compute semi-algebraic invariants for polynomial systems. We expressed the invariance condition as a fixed point of a refinement operator over cones of polynomials. A key element of the framework is the introduction of the projection operator to compute the refinement, which allows to keep the complexity and convergence of the refinement process under control. In future work, we plan to apply this refine-by-projection approach to other fixed-point approaches in abstract interpretation. We also plan to improve the approach by allowing to detect when the size of the iterates can be reduced because some polynomials have become redundant.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their detailed comments and suggestions. This research was funded in part by the Belgian-American Education Foundation (BAEF) and the US National Science Foundation (NSF) under award numbers 1836900 and 1932189.

REFERENCES

[1] Amir Ali Ahmadi and Anirudha Majumdar. Dsos and sdsos optimization: More tractable alternatives to sum of squares and semidefinite optimization. *SIAM Journal on Applied Algebra and Geometry*, 3(2):193–230, 2019. doi: 10.1137/18M118935X.

[2] Daniele Ahmed, Andrea Peruffo, and Alessandro Abate. Automated and sound synthesis of lyapunov functions with smt solvers. In *Tools and Algorithms for the Construction and Analysis of Systems: 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings, Part I 26*, pages 97–114. Springer, 2020.

[3] Fernando Alegre, Eric Feron, and Santosh Pande. Using ellipsoidal domains to analyze control systems software. *CoRR*, abs/0909.1977, 2009. URL http://arxiv.org/abs/0909.1977.

[4] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000. doi: 10.1109/5.871304.

[5] Rajeev Alur. *Principles of Cyber-Physical Systems*. MIT Press, 2015.

[6] Rajeev Alur, Thao Dang, and Franjo Ivančić. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embed. Comput. Syst.*, 5(1):152–199, feb 2006. ISSN 1539-9087. doi: 10.1145/1132357.1132363. URL https://doi.org/10.1145/1132357.1132363.

[7] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *European Control Conference (ECC)*, pages 3420–3431, 2019. doi: 10.23919/ECC.2019.8796030.

[8] Mahathi Anand, Vishnu Murali, Ashutosh Trivedi, and Majid Zamani. Safety verification of dynamical systems via k-inductive barrier certificates. In *2021 60th IEEE Conference on Decision and Control (CDC)*, page 1314–1320. IEEE Press, 2021. doi: 10.1109/CDC45484.2021.9682889. URL https://doi.org/10.1109/CDC45484.2021.9682889.

[9] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 10.0.*, 2022. URL http://docs.mosek.com/9.0/toolbox/index.html.

[10] Vladimir I. Arnold. *Ordinary Differential Equations*. Springer (Universitext), 2006. Translated from Russian by R.Cooke.

[11] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[12] Dimitris Bertsimas and Santosh Vempala. Solving convex programs by random walks. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 109–115, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581134959. doi: 10.1145/509907.509926. URL https://doi.org/10.1145/509907.509926.

[13] Franco Blanchini and Stefano Miani. *Set-Theoretic Methods in Control*. Birkhäuser, Boston, MA, USA, 2008. ISBN 978-0-8176-4606-6. URL https://link.springer.com/book/10.1007/978-0-8176-4606-6.

[14] Patrick Cousot. *Principles of Abstract Interpretation*. The MIT Press, Cambridge, MA, USA, September 2021. ISBN 978-0-26204490-5. URL https://www.amazon.com/Principles-Abstract-Interpretation-Patrick-Cousot/dp/0262044900.

[15] Patrick Cousot and Rhadia 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.

[16] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. The astrée analyzer. In M. Sagiv, editor, *European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 21–30. Springer-Verlag, 2005.

[17] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[18] Komei Fukuda. Cddlib reference manual. *Report version 093a, McGill University, Montréal, Quebec, Canada*, 2003.

[19] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In Amy P. Felty and Aart Middeldorp, editors, *CADE*, volume 9195 of *LNCS*, pages 527–538. Springer, 2015. doi: 10.1007/978-3-319-21401-6_36.

[20] Khalil Ghorbal, Andrew Sogokon, and André Platzer. A hierarchy of proof rules for checking positive invariance of algebraic and semi-algebraic sets. *Computer Languages, Systems & Structures*, 47:19–43, January 2017. doi: 10.1016/j.cl.2015.11.003. URL https://doi.org/10.1016/j.cl.2015.11.003.

[21] Nicolas Halbwachs, Yann Eric Proy, and Pascal Raymond. Verification of linear hybrid systems by means of convex approximations. In Baudouin Le Charlier, editor, *Static Analysis*, pages 223–237, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[22] Thomas A. Henzinger and Pei-Hsin Ho. A note on abstract interpretation strategies for hybrid automata. In Panos Antsaklis, Wolf Kohn, Anil Nerode, and Shankar Sastry, editors, *Hybrid Systems II*, pages 252–264, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[23] Tadeusz Kaczorek and Kamil Borawski. Stability of positive nonlinear systems. In *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 564–569. IEEE, 2017.

[24] Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817, 2001.

[25] Benoît Legat. Polyhedral computation. In *JuliaCon*, July 2023. URL https://pretalx.com/juliacon2023/talk/JP3SPX/.

[26] Jiang Liu, Naijun Zhan, and Hengjun Zhao. Computing semi-algebraic invariants for polynomial dynamical systems. In *Proc. of ACM International Conference on Embedded Software (EMSOFT)*, EMSOFT '11, page 97–106, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307147.

[27] László Lovász and Santosh Vempala. Hit-and-run from a corner. *SIAM Journal on Computing*, 35(4):985–1005, 2006.

[28] James D. Meiss. *Differential Dynamical Systems*. SIAM, 2017.

[29] Sayan Mitra. *Verifying Cyber-Physical Systems: A Path to Safe Autonomy*. MIT Press. ISBN 978-0-262-04480-6. URL https://mitpress.mit.edu/contributors/sayan-mitra.

[30] M. Nagumo. Uber die lage der integralkurven gewöhnlicher differentialgleichungen. 24:551–559, 1942.

[31] Pablo A. Parrilo. *Polynomial Optimization, Sums of Squares, and Applications*, chapter 3, pages 47–157. doi: 10.1137/1.9781611972290.ch3.

[32] André Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, Heidelberg, 2010. ISBN 978-3-642-14508-7. doi: 10.1007/978-3-642-14509-4.

[33] André Platzer. A complete uniform substitution calculus for differential dynamic logic. *J. Autom. Reas.*, 59(2):219–265, 2017. doi: 10.1007/s10817-016-9385-1.

[34] André Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Cham, 2018. ISBN 978-3-319-63587-3. doi: 10.1007/978-3-319-63588-0.

[35] André Platzer and Edmund M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, pages 176–189, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[36] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, pages 477–492, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24743-2.

[37] Stephen Prajna, Ali Jadbabaie, and George J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Transactions on Automatic Control*, 52(8):1415–1428, 2007. doi: 10.1109/TAC.2007.902736.

[38] Pierre Roux, Romain Jobredeaux, Pierre-Loïc Garoche, and Éric Féron. A generic ellipsoid abstract domain for linear time invariant systems. In *Proceedings of Hybrid Systems: Computation and Control*, HSCC '12, page 105–114, New York, NY, USA, 2012. Association for Computing Machinery.

[39] Ricardo Sanfelice. Hybrid feedback control, 2021.

[40] Sriram Sankaranarayanan. Automatic abstraction of non-linear systems using change of variables transformations. In *Hybrid Systems: Computation and Control (HSCC)*, pages 143–152. ACM Press, 2011.

[41] Sriram Sankaranarayanan. Change of basis abstractions for non-linear hybrid systems. *Nonlinear Analysis: Hybrid Systems*, 19:107–133, 2016.

[42] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Fixed point iteration for computing the time elapse operator. In *HSCC*, volume 3927 of *Lecture Notes in Computer Science*, pages 537–551. Springer, 2006.

[43] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 32(1):25–55, 2008.

[44] K. Schmüdgen. The k-moment problem for compact semi-algebraic sets. *Math. Ann.*, 289:203–206, 1991.

[45] Naum Z. Shor. An approach to obtaining global extrema in polynomial problems of mathematical programming. *Kibernetika (Kiev)*, (5):102–6, 1987.

[46] Andrew Sogokon, Khalil Ghorbal, Yong Kiam Tan, and André Platzer. Vector barrier certificates and comparison systems. In K. Havelund, J. Peleska, B. Roscoe, and E. de Vink, editors, *FM'18*, volume 10951, pages 418–437. Springer, 2018.

[47] Yunfei Song. Positive invariance condition for continuous dynamical systems based on nagumo theorem, 2022.

[48] Thomas Sturm and Ashish Tiwari. Verification and synthesis using real quantifier elimination. In *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation*, ISSAC '11, page 329–336, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306751. doi: 10.1145/1993886.1993935. URL https://doi.org/10.1145/1993886.1993935.

[49] Ankur Taly and Ashish Tiwari. Deductive Verification of Continuous Dynamical Systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 383–394, 2009. ISBN 978-3-939897-13-2.

[50] A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In C. J. Tomlin and M. R. Greenstreet, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, March 2002.

[51] A. Tiwari and G. Khanna. Nonlinear systems: Approximating reach sets. In R. Alur and G. Pappas, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2993 of *LNCS*, pages 600–614. Springer, March 2004.

[52] C.J. Tomlin, I. Mitchell, A.M. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.

doi: 10.1109/JPROC.2003.814621.

[53] Tillmann Weisser, Benoît Legat, Chris Coey, Lea Kapelevich, and Juan Pablo Vielma. Polynomial and moment optimization in julia and jump. In *JuliaCon*, 2019. URL https://pretalx.com/juliacon2019/talk/QZBKAU/.

A COMPUTED INVARIANTS FOR THE NUMERICAL EXPERIMENTS

A.1 Invariants in Sec. 6.1

Computed invariant:

```
-0.993031954986876 + 0.037781538498899998*x2
    ^2 - 0.0666256032861187*x1*x2 -
    0.0895607096351955 7*x1^2,
-0.9992011592628854 + 0.007955071759038876*
    x2^2 + 0.004101804719455346 6*x1*x2 +
    0.038947854357559354*x1^2,
-0.9991994975155781 + 0.007562020589211854*
    x2^2 + 0.004248519887888844*x1*x2 +
    0.039052913950008275*x1^2,
-0.9992137430067909 + 0.008771034196869584*
    x2^2 + 0.003688019844171276 6*x1*x2 +
    0.03848848210965402*x1^2,
-0.9992827480108409 + 0.011526972259621198*
    x2^2 + 0.001663939042857424*x1*x2 +
    0.03603262057205512*x1^2,
-0.9993785554131969 + 0.015373693376587 55*x2
    ^2 - 0.0017643808674354974*x1*x2 +
    0.031670798732419 48*x1^2,
-0.9985171136578682 + 0.032865540164738846*
    x2^2 - 0.035792642202647794*x1*x2 -
    0.024542142656933 37*x1^2,
-0.9988802212890721 + 0.031494103922956214*
    x2^2 - 0.031358857509594 62*x1*x2 -
    0.016218723478656986*x1^2.
```

Invariant after removing seven polynomial:

```
-0.993031954986876 + 0.037781538498899998*x2
    ^2 - 0.0666256032861187*x1*x2 -
    0.0895607096351955 7*x1^2,
-0.9991994975155781 + 0.007562020589211854*
    x2^2 + 0.004248519887888844*x1*x2 +
    0.039052913950008275*x1^2,
-0.9993785554131969 + 0.015373693376587 55*x2
    ^2 - 0.0017643808674354974*x1*x2 +
    0.031670798732419 48*x1^2.
```

Single SOS polynomial:

```
-14.147180353695786 + 3.0119346001218656e
    -16*x2 - 2.722035062544444e-16*x1 -
    1.144646613906247 4*x2^2 +
    2.364989150845086*x1*x2 -
    0.585605292809049 8*x1^2 -
    3.431534141322688e-16*x2^3 +
    1.1896397017199832e-15*x1*x2^2 -
    1.065866286140602 1e-15*x1^2*x2 +
    1.430034012737953e-16*x1^3 +
    0.081363293851595 24*x2^4 -
    0.2704342780642278*x1*x2^3 +
    0.9431646915713372*x1^2*x2^2 -
    1.1022331889452621*x1^3*x2 +
    0.10534350132410865*x1^4 +
    3.785239197771556 5e-17*x2^5 -
    1.8099073204583052e-16*x1*x2^4 +
    2.717877171967598 4e-16*x1^2*x2^3 -
    4.647936726088389e-16*x1^3*x2^2 +
    1.547447895294992e-16*x1^4*x2 +
    8.370924115653621e-20*x1^5 -
    0.028676934757914243*x2^6 +
    0.1228202134074245 4*x1*x2^5 -
    0.3819521807112903*x1^2*x2^4 +
    0.24627925175346396*x1^3*x2^3 -
    0.3452458713742735 4*x1^4*x2^2 +
    0.10771573535369378*x1^5*x2 +
    3.978130888933918e-6*x1^6 -
    9.770977101795868e-19*x2^7 +
    6.6637423968859206e-18*x1*x2^6 -
    1.3603234786515431e-17*x1^2*x2^5 +
    3.414552747312814e-17*x1^3*x2^4 -
    1.6599099673281645e-17*x1^4*x2^3 +
    4.065547544821152e-17*x1^5*x2^2 +
    2.4742376740606324e-20*x1^6*x2 -
    1.0723543159897107e-22*x1^7 +
    0.002680016807049046*x2^8 -
    0.01420463919099943 5*x1*x2^7 +
    0.044234100069372326*x1^2*x2^6 -
    0.04910471879176629*x1^3*x2^5 +
    0.0613504579236852 7*x1^4*x2^4 -
    0.00990263283303217*x1^5*x2^3 +
    0.027040573261397 65*x1^6*x2^2 -
    8.062124627215044e-9*x1^8.
```

A.2 Invariants in Sec. 6.2

Computed invariant with H_1:

```
-0.3772320184363528 + 0.9167844427079844*x2
    ^2 + 0.1311575002622633*x1^2.
```

Computed invariant with H_2:

```
-0.37162601840205317 + 0.9180933839414407*x2
    ^2 + 0.05818669652154412*x1*x2 +
    0.12495178733215878*x1^2,
-0.36289098847525536 + 0.8581254736347127*x2
    ^2 - 0.2968404612144774*x1*x2 +
    0.20932401335878398*x1^2,
-0.3654505214636404 + 0.8658484454133306*x2
    ^2 - 0.2731556539932166*x1*x2 +
    0.20527633723546868*x1^2,
-0.35555787489000495 + 0.5642858490142215*x2
    ^2 - 0.6868775947801282*x1*x2 +
    0.2887200166165638*x1^2,
-0.3466720162664112 + 0.4056451291563513*x2
    ^2 - 0.7892214122321266*x1*x2 +
    0.303973855461018*x1^2,
-0.3157508057881853 - 0.006483350150517508*
    x2^2 - 0.8980827744908498*x1*x2 +
    0.306115541875539*x1^2,
-0.27099616631001205 - 0.34953984829125423*
    x2^2 - 0.8559147999107475*x1*x2 +
    0.2679418362173937*x1^2,
-0.3305122564921238 + 0.1764518798902288*x2
    ^2 - 0.8735393478570359*x1*x2 +
    0.3107336321307906*x1^2,
-0.36139139688398925 + 0.67178362788344*x2^2
    - 0.5868761074415546*x1*x2 +
    0.2714395882701188*x1^2.
```

Single SOS polynomial:

```
-1.3180825884858496 - 4.105382662498511e-17*
    x2 + 1.0621053483771272e-16*x1 -
    0.5880063521402954*x2^2 -
    0.3131075552389977*x1*x2 -
    0.37633860633493227*x1^2 +
    1.2762431718729704e-16*x2^3 -
    4.5733982603058425e-17*x1*x2^2 +
    1.4448342975755364e-16*x1^2*x2 -
    1.0527221549852131e-17*x1^3 +
    1.0799054678022022*x2^4 +
    0.35650897319712777*x1^2*x2^2 -
    0.21125714773835375*x1^3*x2 +
    0.2362070686303661*x1^4.
```

A.3 Invariants in Sec. 6.3

Computed invariant with H_1:

```
-0.5367809350302005 + 0.7364709881468191*x2
    + 0.41167549284132915*x1,
-0.5826353338500777 + 0.005118306539458857*
    x2 + 0.8127175835968954*x1,
-0.525462818319698 4 + 0.09430856351039767*x2
    + 0.8455736049641833*x1,
-0.5785554785195461 + 0.011247915805675145*
    x2 + 0.8155654741742367*x1,
-0.5254539445790127 + 0.22757778768343134*x2
    + 0.8198210186860734*x1,
-0.6209349329339116 - 0.7270290130231652*x2
    - 0.2930334849209576*x1,
-0.6182634217029666 - 0.7273093101463118*x2
    - 0.29794547950696576*x1,
-0.5254420058132805 - 0.7146509275155337*x2
    - 0.4617193415139804*x1,
-0.5254551868283345 - 0.2785035904128786*x2
    - 0.8039481306417321*x1,
-0.5618213044784532 - 0.021696949140147616*
    x2 - 0.8269740408453817*x1.
```

Computed invariant with H_2:

```
-0.2702171797894011 + 0.6808019873006584*x2
    ^2 + 1.571660199888246e-8*x1*x2 +
    0.68080197549224*x1^2.
```

A.4 Invariants in Sec. 6.4

Computed invariant with H_1:

```
-0.5494478511158494 - 0.01967978663892927*x2
    - 0.8352962138678838*x1,
-0.7084226824177525 - 0.014091865778762787*
    x2 - 0.7056477324805271*x1,
-0.9981365603664998 - 7.156732576621838e-5*
    x2 - 0.06101968320018036*x1,
-0.5724744460351411 - 0.8063258458465716*x2
    - 0.14869982836765083*x1,
-0.9219306401837807 - 0.38735496799295877*x2
    - 0.0001531714660703783*x1,
-0.5612419612399377 - 0.8113201777029977*x2
    - 0.16360632688109705*x1,
-0.5254741199710673 - 0.8143011604485618*x2
    - 0.24655743617412532*x1,
-0.5254687744128949 - 0.7146370028416534*x2
    - 0.46171043012532076*x1,
-0.5803704215825563 - 0.018596748271763663*
    x2 - 0.8141402426522119*x1,
-0.5254709394084618 - 0.2784999641909676*x2
    - 0.8039390908413512*x1,
-0.5406994519179427 - 0.01999582787652706*x2
    - 0.8409781623580773*x1.
```

Computed invariant with H_2:

```
-0.6267041594156744 + 0.40575758632504677*x2
    ^2 - 0.6304192506252365*x1*x2 +
    0.21254233966878333*x1^2,
-0.6133900277700439 + 0.3137496111572383*x2
    ^2 - 0.6448074954421678*x1*x2 +
    0.3309639695685099*x1^2,
```

```
-0.6167085200410728 + 0.11314322183028681*x2
    ^2 - 0.6868515751717251*x1*x2 +
    0.36756513211502*x1^2,
-0.6332960744235467 + 0.05489446827649687*x2
    ^2 - 0.6468844236616733*x1*x2 +
    0.4212638388187038*x1^2,
-0.6785689434622717 + 0.25813959512251694*x2
    ^2 - 0.5985695368764566*x1*x2 +
    0.3385596667088148*x1^2,
-0.7387678166947391 + 0.30757171959887847*x2
    ^2 - 0.54938996235212*x1*x2 +
    0.24040053990333135*x1^2,
-0.7639271328813163 + 0.32239414845369285*x2
    ^2 - 0.5235828206629485*x1*x2 +
    0.1958018860919908*x1^2,
-0.7245646226772867 + 0.11971534102676137*x2
    ^2 - 0.5487267537981958*x1*x2 +
    0.39946626184630585*x1^2,
-0.7561425268451387 + 0.13877899404732386*x2
    ^2 - 0.5240940437640208*x1*x2 +
    0.36648915836425316*x1^2,
-0.8040538669652494 + 0.24971470248063635*x2
    ^2 - 0.4854117256836518*x1*x2 +
    0.2356170684650319*x1^2,
-0.6797764489635506 + 0.024276733350190823*
    x2^2 - 0.5747840390401819*x1*x2 +
    0.4549043065491946*x1^2,
-0.6929429879867485 + 0.05571424244515854*x2
    ^2 - 0.5666823473370078*x1*x2 +
    0.44226355921031735*x1^2,
-0.6797412642498962 + 0.024265954560279853*
    x2^2 - 0.5748144017130071*x1*x2 +
    0.45491909248628926*x1^2,
-0.6322792310032767 - 0.04268066607914269*x2
    ^2 - 0.6377175075714147*x1*x2 +
    0.43785581567663184*x1^2,
-0.6478252149391743 + 0.3946513020780266*x2
    ^2 - 0.6164715941035325*x1*x2 +
    0.21103462824991598*x1^2.
```

Invariant after removing four polynomials:

```
-0.6133900277700439 + 0.3137496111572383*x2
    ^2 - 0.6448074954421678*x1*x2 +
    0.3309639695685099*x1^2,
-0.6167085200410728 + 0.11314322183028681*x2
    ^2 - 0.6868515751717251*x1*x2 +
    0.36756513211502*x1^2,
-0.6785689434622717 + 0.25813959512251694*x2
    ^2 - 0.5985695368764566*x1*x2 +
    0.3385596667088148*x1^2,
-0.7387678166947391 + 0.30757171959887847*x2
    ^2 - 0.54938996235212*x1*x2 +
    0.24040053990333135*x1^2,
-0.7245646226772867 + 0.11971534102676137*x2
    ^2 - 0.5487267537981958*x1*x2 +
    0.39946626184630585*x1^2,
-0.8040538669652494 + 0.24971470248063635*x2
    ^2 - 0.4854117256836518*x1*x2 +
    0.2356170684650319*x1^2,
-0.6797764489635506 + 0.024276733350190823*
    x2^2 - 0.5747840390401819*x1*x2 +
    0.4549043065491946*x1^2,
-0.6929429879867485 + 0.05571424244515854*x2
    ^2 - 0.5666823473370078*x1*x2 +
    0.44226355921031735*x1^2,
-0.6797412642498962 + 0.024265954560279853*
    x2^2 - 0.5748144017130071*x1*x2 +
    0.45491909248628926*x1^2,
-0.6322792310032767 - 0.04268066607914269*x2
    ^2 - 0.6377175075714147*x1*x2 +
    0.43785581567663184*x1^2,
-0.6478252149391743 + 0.3946513020780266*x2
    ^2 - 0.6164715941035325*x1*x2 +
    0.21103462824991598*x1^2.
```

Single SOS polynomial:

```
-0.28675786254623603 - 0.18280086902711773*
    x2 + 0.12140102417765512*x1 -
    0.15460754380061761*x2^2 +
    0.13094023733957572*x1*x2 +
    0.0179862099898157*x1^2 +
    2.2518593469162838*x2^3 -
    5.263539809426801*x1*x2^2 +
    5.324546770702873*x1^2*x2 -
    2.119308286925807*x1^3 -
    4.230287442095737*x2^4 +
    15.587357182751107*x1*x2^3 -
    22.392477842104498*x1^2*x2^2 +
    14.889470233666147*x1^3*x2 -
    3.833726478458332*x1^4 -
    0.038895473765656496*x2^5 +
    0.683051665353735*x1*x2^4 -
    2.799091160948933*x1^2*x2^3 +
    3.6150946275380034*x1^3*x2^2 -
    1.809841753837602*x1^4*x2 +
    0.5953443364300465*x1^5 +
    6.3660968391039425*x2^6 -
    26.921765398893317*x1*x2^5 +
    51.09020271546622*x1^2*x2^4 -
    60.67932780655035*x1^3*x2^3 +
    50.30016099540679*x1^4*x2^2 -
    26.462877385173247*x1^5*x2 +
    6.039783458388755*x1^6.
```

A.5 Invariants in Sec. 6.5

Computed invariant:

```
-0.871872297547406 - 0.21177461319034813*x3
    ^2 + 0.10501545362584444*x2^2 +
    0.42890787411429493*x1^2,
-0.8535189933667975 - 0.17888097824420462*x3
    ^2 + 0.007807118789072992*x2^2 +
    0.4893321698812969*x1^2,
-0.8650501279681401 - 0.200096067640876*x3^2
     + 0.0877118073301994*x2^2 +
    0.45161541013547407*x1^2,
-0.6906501877035409 + 0.6121202891787498*x3
    ^2 - 0.0030550649675006266*x2^2 +
    0.38509964473994507*x1^2,
-0.3329552584068616 + 0.9316113015966564*x3
    ^2 + 0.028997394138460 04*x2^2 +
    0.1428297229913382*x1^2,
-0.33337674127559513 + 0.9332427750444002*x3
    ^2 - 0.0032308722117315894*x2^2 +
    0.13381865590657924*x1^2,
-0.33307532808420715 + 0.932148346252807 5*x3
    ^2 - 0.0022873854889847497*x2^2 +
    0.14196849744707027*x1^2.
```

Invariant after removing all but one polynomial:

```
-0.3329552584068616 + 0.9316113015966564*x3
    ^2 + 0.028997394138460 04*x2^2 +
    0.1428297229913382*x1^2,
```

A.6 Invariant in Sec. 6.6

Computed invariant:

```
-0.2965286550303196 + 0.31208170970389876*x4
    ^2 + 0.7082105072170928*x3^2 +
    0.21640225618507145*x2^2 +
    0.516026844456294*x1^2,
-0.2931285206057089 + 0.15293571645387932*x4
    ^2 + 0.7322811557930203*x3^2 +
    0.23679999551908032*x2^2 +
    0.5462384168408964*x1^2,
-0.30196009738646706 - 0.17922236853437762*
    x4^2 + 0.6674810292357307*x3^2 +
    0.2631004599603873*x2^2 +
    0.6016200343924106*x1^2,
-0.27948055374478004 - 0.19415016054986303*
    x4^2 + 0.7177643331842496*x3^2 +
    0.25043753432728816*x2^2 +
    0.5534363004408106*x1^2,
-0.28404229752500015 - 0.10384997528433605*
    x4^2 + 0.7361300027472623*x3^2 +
    0.25059687513512807*x2^2 +
    0.5512249822694784*x1^2,
-0.28351426258665674 - 0.15651295060362 8*x4
    ^2 + 0.7340202628325854*x3^2 +
    0.2535420196275728*x2^2 +
    0.5404202598327011*x1^2,
-0.28737838494253265 + 0.09521687543953784*
    x4^2 + 0.7494453999743359*x3^2 +
    0.2388086684777887*x2^2 +
    0.5381908795367366*x1^2,
-0.28698981149979436 + 0.3037797367028826*x4
    ^2 + 0.7330534717006845*x3^2 +
    0.20694708802792727*x2^2 +
    0.4951365771667198*x1^2,
-0.3653836818924293 + 0.6429343959932746*x4
    ^2 + 0.3628981863579539*x3^2 +
    0.14439663437203434*x2^2 +
    0.5482560038664711*x1^2,
-0.328972906493326 + 0.5498664516444461*x4^2
     + 0.5717317791434137*x3^2 +
    0.13562023364037898*x2^2 +
    0.4941190515393182*x1^2,
-0.29127724671601685 + 0.31864066519563333*
    x4^2 + 0.7227184596407003*x3^2 +
    0.20228956722902733*x2^2 +
    0.5003825048041624*x1^2,
```

```
-0.3126218214440174 + 0.32722389772312327*x4
    ^2 + 0.6647074360040862*x3^2 +
    0.22993280710647002*x2^2 +
    0.5481669875621151*x1^2,
-0.290576915490207 - 0.18985024245076754*x4
    ^2 + 0.7622007966561133*x3^2 +
    0.26171008178364713*x2^2 +
    0.4796662592836452*x1^2,
-0.29141740062966376 + 0.36885793208443624*
    x4^2 + 0.7502101116845885*x3^2 +
    0.2020704172890032*x2^2 +
    0.4187744731134993*x1^2,
-0.29992514407576354 + 0.6598063821397299*x4
    ^2 + 0.6626714563508616*x3^2 +
    0.10052522509054947*x2^2 +
    0.15956712097777387*x1^2,
-0.3422412216869133 + 0.8264029971341018*x4
    ^2 + 0.41570130597352023*x3^2 +
    0.09779104617254651*x2^2 +
    0.13250799223598367*x1^2.
```

A.7 Invariants in Sec. 6.7

Computed invariant for mode 1:

```
Not shown because too long.
```

Computed invariant for mode 2:

```
-0.9002692709392969 + 0.4274611056124662*x2
    - 0.047001881841985366*x2^2 +
    0.06769834631885185*x1^2,
-0.8983116911689949 + 0.43272028254777545*x2
     - 0.039202954103108485*x2^2 +
    0.06521035937996048*x1^2,
-0.8994367184266815 + 0.42966260630358305*x2
     - 0.04379822681334729*x2^2 +
    0.06697275280486484*x1^2,
-0.9640047759616999 - 0.05184587153905228*x2
     + 0.24143150505809027*x2^2 +
    0.09857801931857868*x1^2.
```

A.8 Invariants in Sec. 6.8

Computed invariant for mode 1:

```
-0.9857936879547902 + 0.09909691865399034*x2
    ^2 + 0.000387618927225279*x1*x2 +
    0.13561141269578034*x1^2,
-0.986140544056003 + 0.0972059879302515*x2^2
     - 0.003027492093934748*x1*x2 +
    0.13441970678087045*x1^2,
-0.9847674411170849 + 0.10467633738192324*x2
    ^2 + 0.01082587545043865*x1*x2 +
    0.13841514270030641*x1^2,
-0.9834456725348417 + 0.11233213610927176*x2
    ^2 + 0.025778192014470788*x1*x2 +
    0.13982698304006194*x1^2,
-0.9834072352067391 + 0.11296145964890551*x2
    ^2 + 0.0269975268512748*x1*x2 +
    0.13935943427317987*x1^2,
-0.9834072342091401 + 0.11295953658755152*x2
    ^2 + 0.026995375099488933*x1*x2 +
    0.13936141690687895*x1^2,
-0.9850066759345267 + 0.11835293336418753*x2
    ^2 + 0.03755265898642679*x1*x2 +
    0.11976739678040782*x1^2,
-0.9842843646648725 + 0.1157526692019295*x2
    ^2 + 0.03238089022904035*x1*x2 +
    0.12937189415352954*x1^2.
```

Computed invariant for mode 2:

```
-0.9817527300995846 + 0.14126842778666962*x2
    ^2 - 0.022113062682126854*x1*x2 +
    0.12536275647699704*x1^2,
-0.9430521512391482 + 0.1274811491934949*x2
    ^2 - 0.28290212098114437*x1*x2 +
    0.1198648680301727*x1^2,
-0.959675273017751 + 0.06963989317132586*x2
    ^2 - 0.24338182490186436*x1*x2 +
    0.1222249685818302*x1^2,
-0.9460438614272869 + 0.11497499259730252*x2
    ^2 - 0.2780506258895415*x1*x2 +
    0.12028970353040606*x1^2,
-0.944755121852421 + 0.12013698286746648*x2
    ^2 - 0.2803195319354359*x1*x2 +
    0.12010755636768729*x1^2,
-0.9664456245483404 + 0.04994292733097908*x2
    ^2 - 0.21980562893445577*x1*x2 +
    0.12318297077806359*x1^2,
-0.9715222141302825 + 0.1495897116723925*x2
    ^2 - 0.1357599376728176*x1*x2 +
    0.12384161229666077*x1^2,
-0.9653797047811981 + 0.1389581820496891*x2
    ^2 - 0.18333015075898298*x1*x2 +
    0.12297440815348881*x1^2,
-0.9571361696395247 + 0.12862903115612198*x2
    ^2 - 0.22913553228295144*x1*x2 +
    0.12182706167771154*x1^2,
-0.9401064342228134 + 0.14261960375083502*x2
    ^2 - 0.28564367648818584*x1*x2 +
    0.11944551494775421*x1^2.
```

Computed invariant for mode 3:

```
-0.9902072318403451 + 0.13563419510408165*x2
    ^2 + 0.018812787726133103*x1*x2 +
    0.02718606531884196*x1^2.
```