Synthesizing Invariants for Polynomial Programs by Semidefinite Programming

HAO WU, State Key Lab. of Computer Sciences, Institute of Software, Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China
QIUYE WANG, State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China and Fermat Labs, Huawei Inc., Dongguan, China
BAI XUE, Key Lab. of System Software and State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China
NAIJUN ZHAN, Key Laboratory of High Confidence Software Technology, School of Computer Science, Peking University, Beijing, China and State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
LIHONG ZHI, Key Lab. of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China
ZHI-HONG YANG, School of Mathematics and Statistics, Central South University, Changsha, China

Constraint-solving-based program invariant synthesis takes a parametric invariant template and encodes the (inductive) invariant conditions into constraints. The problem of characterizing the set of all valid parameter assignments is referred to as the *strong invariant synthesis problem*, while the problem of finding a concrete valid parameter assignment is called the *weak invariant synthesis problem*. For both problems, the challenge lies in solving or reducing the encoded constraints, which are generally non-convex and lack efficient solvers.

In this article, we propose two novel algorithms for synthesizing invariants of polynomial programs using semidefinite programming (SDP): (1) The Cluster algorithm targets the strong invariant synthesis problem for polynomial invariant templates. Leveraging robust optimization techniques, it solves a series of SDP relaxations and yields a sequence of increasingly precise under-approximations of the set of valid parameter assignments. We prove the algorithm's soundness, convergence, and weak completeness under a specific robustness assumption on templates. Moreover, the outputs can simplify the weak invariant synthesis

H. Wu and N. Zhan are funded by the Sponsor National Key R&D Program of China under Grant No. #2022YFA1005101 and Sponsor Natural Science Foundation of China under Grant No. #62192732. L. Zhi is supported by the Sponsor National Key R&D Program of China under Grant No. #2023YFA1009401. Z.-H. Yang is supported by the Sponsor Natural Science Foundation of China under Grant No. #12201425.

Authors' Contact Information: Hao Wu, State Key Lab. of Computer Sciences, Institute of Software, Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China; e-mail: wuhao@ios.ac.cn; Qiuye Wang, State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China and Fermat Labs, Huawei Inc., Dongguan, China; e-mail: wangqiuye2@huawei.com; Bai Xue, Key Lab. of System Software and State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China; e-mail: xuebai@ios.ac.cn; Naijun Zhan (corresponding author), Key Laboratory of High Confidence Software Technology, School of Computer Science, Peking University, Beijing, China and State Key Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China; e-mail: njzhan@pku.edu.cn; Lihong Zhi, Key Lab. of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China and University of Chinese Academy of Sciences, Beijing, China; e-mail: lzhi@mmrc.iss.ac.cn; Zhi-Hong Yang, School of Mathematics and Statistics, Central South University, Changsha, China; e-mail: yangzhihong@csu.edu.cn.

ACM 1558-4593/2025/2-ART1
https://doi.org/10.1145/3708559

problem. (2) The Mask algorithm addresses the weak invariant synthesis problem in scenarios where the aforementioned robustness assumption does not hold, rendering the Cluster algorithm ineffective. It identifies a specific subclass of invariant templates, termed masked templates, involving parameterized polynomial equalities and known inequalities. By applying variable substitution, the algorithm transforms constraints into an equivalent form amenable to SDP relaxations. Both algorithms have been implemented and demonstrated superior performance compared to state-of-the-art methods in our empirical evaluation.

CCS Concepts: • **Theory of computation** → **Invariants**; *Program verification*; Logic and verification; • **Mathematics of computing** → *Semidefinite programming*;

Additional Key Words and Phrases: program verification, invariant synthesis, sum-of-squares relaxations, semidefinite programming

ACM Reference format:
Hao Wu, Qiuye Wang, Bai Xue, Naijun Zhan, Lihong Zhi, and Zhi-Hong Yang. 2025. Synthesizing Invariants for Polynomial Programs by Semidefinite Programming. *ACM Trans. Program. Lang. Syst.* 47, 1, Article 1 (February 2025), 35 pages.
https://doi.org/10.1145/3708559

1 Introduction

The dominant approach to program verification is *Floyd-Hoare-Naur's inductive assertion method* [27, 35, 65], which is based on Hoare logic [35]. The central concept of Hoare logic is the *Hoare triple* with the form

$$\{P\}C\{Q\},$$

where C is a piece of program to be verified, P is the precondition, and Q is the postcondition. The program C is said to be *partially correct* with respect to specifications P and Q if, assuming the precondition P holds before executing C and the program C terminates, then the postcondition Q will hold upon the completion of C.

In Hoare logic, an *invariant* is an assertion associated with a particular program location, and it holds whenever the location is reached during program execution. An *inductive invariant* is a specific type of invariant that satisfies the inductive property: If the assertion holds at a program location, then it is preserved during subsequent visits to that location. The difference between invariants and inductive invariants is discussed in detail in [77]. For the purpose of this article, we will solely focus on inductive invariants, and for simplicity, we will refer to them simply as "invariants," unless otherwise stated.

Invariant generation stands as a crucial aspect of Hoare-style program verification. The effectiveness of the verification process heavily relies on the ability to discover appropriate invariants that accurately capture the behavior and properties of the program throughout its execution. Though this has been shown to be undecidable in general [63], many efforts have been put into this area, resulting in various invariant synthesis techniques, including approaches based on constraint solving (discussed below), recurrence analysis [40, 45], abstract interpretation [71, 73], Craig interpolation [29, 54], machine learning [81, 90], and so on.

Constraint-solving-based invariant synthesis, also known as template-based invariant synthesis, is promising and therefore prominent for discovering program invariants. The general workflow of these methods can be summarized as follows: The algorithm takes a user-specified parametric formula as input, encodes the invariant conditions into constraints, and attempts to reduce or solve these constraints. The *strong invariant synthesis problem* aims to characterize the set of all valid parameter assignments that satisfy the invariant conditions. This typically involves reducing the

initial constraints into new constraints solely on the parameters. In contrast, the *weak invariant synthesis problem* seeks to find a single concrete invariant that satisfies the constraints. This usually involves solving the constraints to identify a valid parameter assignment that defines a specific invariant.

In this article, we consider both the strong and the weak invariant synthesis problem for polynomial programs over real-valued variables, where the invariant templates are given as conjunctions of polynomial inequalities. In this setting, the conditions for a template to be an invariant can be expressed as a first-order logic formula. It is worth noting that if both the program and the specifications (precondition and postcondition) are polynomial, the truth of the formula is decidable, as per Tarski's theorem [82]. For the weak invariant synthesis problem, one major approach is to use Putinar's Positivstellensatz (see Theorem 1) to transform the invariant conditions into constraints containing **sum-of-squares (SOS)** polynomials, i.e., **bilinear matrix inequalities (BMIs)**. However, solving the resulting constraints is still NP-hard [11, 83], and hence existing works mostly rely on general-purpose solvers for non-linear real arithmetic [14, 30, 89] or heuristic strengthening strategies [1, 18, 55] to tackle them. For the strong invariant synthesis problem, only a general doubly exponential upper bound based on quantifier elimination is known [43].

Contributions. We propose two novel **semidefinite programming (SDP)**-based approaches for the invariant synthesis problem of polynomial programs over real-valued variables. This demonstrates that, by carefully encoding, the original invariant synthesis problem can be transformed into convex optimization problems amenable to numerical solvers.

The Cluster algorithm (Sections 3 and 4) tackles the strong invariant synthesis problem for invariant templates expressed as conjunctions of parameterized polynomial inequalities. To characterize the set of valid assignments of parameters $\boldsymbol{a}$, called the *valid set*, we employ Lasserre's technique [51] to construct a series of SOS relaxations of the invariant conditions. Notably, our encoding of SOS relaxations can be solved as SDPs, which is not the case as in [14, 30]. For any $D \in \mathbb{N}$, our Cluster algorithm produces a series of under-approximations $R_{I,1}, R_{I,2}, \ldots, R_{I,D}$ of the valid set. For each d such that $1 \leq d \leq D$, the dth under-approximation is defined by $R_{I,d} = \{\boldsymbol{a} \mid h_d(\boldsymbol{a}) \leq 0\}$, where $h_d(\boldsymbol{a})$ is a polynomial of degree at most d. As D goes to infinity, the sequence of under-approximations converges to the valid set. Furthermore, under the robustness assumption that the valid set has an interior point, we establish a semi-completeness result: For sufficiently large D, the Cluster algorithm will produce a non-empty under-approximation $R_{I,d}$ of the valid set for some $1 \leq d \leq D$. In such cases, the weak invariant synthesis problem reduces to solving $h_d(\boldsymbol{a}) \leq 0$.

The Mask algorithm (Section 5) is designed for the weak invariant synthesis problem in scenarios where the Cluster algorithm fails due to the violation of the robustness assumption. Such scenarios often arise when the invariant templates include equalities. The Mask algorithm focuses on *masked templates*, a specific subclass of invariant templates containing parameterized polynomial equalities and known inequalities. By using variable substitution, the Mask algorithm transforms the invariant conditions into constraints that again allow for a hierarchy of SDP relaxations.

The two algorithms have been implemented and tested on two sets of benchmarks, depending on whether the invariant templates include equalities. Compared with state-of-the-art constraint-solving-based and learning-based methods, both of our approaches demonstrate advantages in terms of effectiveness and efficiency.

Structure. The rest of this article is organized as follows: In Section 2, we introduce basic notions and algebraic tools that will be used. Section 3 explains Lasserre's technique and proposes the Cluster algorithm. Section 4 discusses additional extensions to enhance the expressiveness of the algorithm. Section 5 introduces the definition of masked templates and presents the Mask algorithm.

We report the experimental results in Section 6 and discuss related work in Section 7. Finally, Section 8 concludes the article.

2 Preliminaries

In the following, we first fix the notation used throughout the rest of this article. In Section 2.1, we formally define the invariant synthesis problems of interest. In Section 2.2, we give a brief introduction to SOS relaxations, which serves as the fundamental technique in our algorithms.

Basic Notations. The following basic notions will be used. Let $\mathbb{R}$ and $\mathbb{N}$ denote the set of real numbers and the set of natural numbers, respectively. We use boldface letters to denote vectors (such as $\boldsymbol{x}$), vector-valued functions (such as $\boldsymbol{f}(\boldsymbol{x})$), and vector of constants (such as $\mathbf{0}$). The comparison between vectors is element-wise. Given a function $f : \mathbb{R}^n \to \mathbb{R}$, the 0-sublevel set of f is the set $\{\boldsymbol{x} \in \mathbb{R}^n \mid f(\boldsymbol{x}) \leq 0\}$. For a vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, we use $\|\boldsymbol{x}\|_1 = \sum_{i=1}^n |x_i|$ and $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$ to denote its l_1-norm and l_2-norm, respectively. We say $\boldsymbol{x}_0 \in \mathbb{R}^n$ is an interior point of a set $S \subseteq \mathbb{R}^n$ if there exists $\epsilon > 0$ such that $\boldsymbol{x} \in S$ for all $\boldsymbol{x}$ satisfying $\|\boldsymbol{x} - \boldsymbol{x}_0\|_2 \leq \epsilon$. We use μ to denote the Lebesgue measure. Given a hyper-rectangle $C = [a_1, b_1] \times \cdots \times [a_n, b_n] \in \mathbb{R}^n$ with $-\infty < a_i < b_i < \infty$, the volume of C is $\mu(C) = \prod_{i=1}^n (b_i - a_i)$.

We will also use the following notations from real algebraic geometry. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$ denote a vector of variables in $\mathbb{R}^n$. $\mathbb{R}[\boldsymbol{x}]$ denotes the ring of polynomials in variables $\boldsymbol{x}$, and $\mathbb{R}^d[\boldsymbol{x}]$ denotes the set of polynomials of degree less than or equal to d in variables $\boldsymbol{x}$, where $d \in \mathbb{N}$. For convenience, we do not explicitly distinguish a polynomial $p \in \mathbb{R}[\boldsymbol{x}]$ and the function $p(\boldsymbol{x})$ it introduces. A basic semialgebraic set $\mathcal{K} \subseteq \mathbb{R}^n$ is of the form $\{\boldsymbol{x} \in \mathbb{R}^n \mid p_1(\boldsymbol{x}) \diamond 0, \ldots, p_m(\boldsymbol{x}) \diamond 0\}$, where $p_i(\boldsymbol{x}) \in \mathbb{R}[\boldsymbol{x}]$ and each $\diamond$ can be one of $\{<, \leq, =, \geq, >\}$. A basic closed semialgebraic set is of the form $\{\boldsymbol{x} \in \mathbb{R}^n \mid p_1(\boldsymbol{x}) \geq 0, \ldots, p_m(\boldsymbol{x}) \geq 0\}$. A semialgebraic set is of the form $\bigcup_{i=1}^n \mathcal{K}_i$, where each $\mathcal{K}_i$ is a basic semialgebraic set. We say a polynomial $p(\boldsymbol{x})$ is non-negative (respectively, strictly positive) over $\mathcal{K}$ if $p(\boldsymbol{x}) \geq 0$ (respectively, $p(\boldsymbol{x}) > 0$) for all $\boldsymbol{x} \in \mathcal{K}$.

2.1 Problem Formulation

Program Model. In this article, we focus on synthesizing invariants for loops of the form in Code 1. In Section 4.1, we will discuss how to handle nested loops.

Code 1: The Program Model

```
// Program variables: x ∈ ℝ^n
// Precondition: Pre = {x | q_pre(x) ≤ 0}
while (g(x) ≤ 0) {
    case (c_1(x) ≤ 0) : x ← f_1(x);
    case (c_2(x) ≤ 0) : x ← f_2(x);
    ...
    case (c_k(x) ≤ 0) : x ← f_k(x);
}
// Postcondition: Post = {x | q_post(x) ≤ 0}
```

In our program model, program variables $\boldsymbol{x} \in \mathbb{R}^n$ are assumed to take real values. The loop consists of a loop guard $\boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0}$ and a switch-case loop body, where each branch contains a branch conditional $\boldsymbol{c}_i(\boldsymbol{x}) \leq \mathbf{0}$ and an assignment statement $\boldsymbol{x} = \boldsymbol{f}_i(\boldsymbol{x})$ for $i = 1, \ldots, k$. Here, we require that $\boldsymbol{g}(\boldsymbol{x}), \boldsymbol{c}_i(\boldsymbol{x}), \boldsymbol{f}_i(\boldsymbol{x})$ are all (vectors of) polynomials. The branch conditionals $\boldsymbol{c}_i(\boldsymbol{x}) \leq \mathbf{0}$ are tested

in parallel. This means that if more than one branch conditionals are satisfied, the program will *non-deterministically* choose a satisfied branch.

The goal is to prove the correctness of the program, i.e., for any state satisfying the precondition ($\boldsymbol{x} \in \mathit{Pre}$), if the loop terminates, the final state must satisfy the postcondition ($\boldsymbol{x} \in \mathit{Post}$). Here *Pre* and *Post* are basic semialgebraic sets defined by polynomial inequalities $\boldsymbol{q}_{pre}(\boldsymbol{x}) \leq \boldsymbol{0}$ and $\boldsymbol{q}_{post}(\boldsymbol{x}) \leq \boldsymbol{0}$, respectively.

We make one *assumption* in our model: Throughout the execution of the program, the program state $\boldsymbol{x}$ remains within a known hyper-rectangle $C_{\boldsymbol{x}} \subseteq \mathbb{R}^n$. In our algorithms, we consider $C_{\boldsymbol{x}}$ to be of the form $\{\boldsymbol{x} \in \mathbb{R}^n \mid x_1^2 - N^2 \leq 0, \ldots, x_n^2 - N^2 \leq 0\} = [-N, N]^n$, where $N \in \mathbb{N}$ is a constant.

This assumption is a technical assumption corresponding to the Archimedean condition in Putinar's Positivstellensatz (later Theorem 1). In most cases, many real-world programs have natural bounds for program variables. Additionally, in practical programming languages like `C`, variables are typically assigned types and have known value ranges. Therefore, the assumption is often reasonable. In later Remark 1, we will explain that why this assumption is not essential and how to remove it. Moreover, our algorithms remain sound even without this assumption, which is similar to other works based on Putinar's Positivstellensatz [1, 14, 30].

Invariant Synthesis Problem. The formal definition of loop invariants is formulated as follows:

Definition 1 (Invariant). $\mathit{Inv} \subseteq \mathbb{R}^n$ is an invariant of the program in Code 1 if it satisfies the following three conditions, also called the invariant conditions:

$$\boldsymbol{x} \in \mathit{Pre} \implies \boldsymbol{x} \in \mathit{Inv}, \qquad \text{(Initial Cond.)}$$

$$\boldsymbol{x} \in \mathit{Inv} \wedge \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \wedge \boldsymbol{c}_i(\boldsymbol{x}) \leq \boldsymbol{0} \implies \boldsymbol{f}_i(\boldsymbol{x}) \in \mathit{Inv}, \quad \text{for } i = 1, \ldots, k \qquad \text{(Inductive Cond.)}$$

$$\boldsymbol{x} \in \mathit{Inv} \wedge \neg(\boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}) \implies \boldsymbol{x} \in \mathit{Post} \qquad \text{(Saturation Cond.)}$$

The existence of an invariant implies the correctness of the loop. However, directly searching for a satisfactory *Inv* within the entire space of all subsets of $\mathbb{R}^n$ could be challenging. To address this issue, one common approach is to impose constraints on the invariants *Inv* to adhere to specific types of parametric formulas. For explanation, we primarily focus on polynomial templates, which are defined as follows. The extension to basic semialgebraic templates (defined in Section 4.2) is straightforward.

Definition 2 (Polynomial Template). A polynomial template is a polynomial $I(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$ defined over $C_{\boldsymbol{a}} \times C_{\boldsymbol{x}}$, where $C_{\boldsymbol{a}} \subseteq \mathbb{R}^{n'}$ is a hyper-rectangle and $\boldsymbol{a} = (a_1, a_2, \ldots, a_{n'}) \in C_{\boldsymbol{a}}$ are referred to as parameters. Given a parameter assignment $\boldsymbol{a}_0 \in C_{\boldsymbol{a}}$, the instantiation of the invariant *Inv* w.r.t. $\boldsymbol{a}_0$ is the set $\{\boldsymbol{x} \in C_{\boldsymbol{x}} \mid I(\boldsymbol{a}_0, \boldsymbol{x}) \leq 0\}$, where $C_{\boldsymbol{x}} = [-N, N]^n$ for some user-defined $N \in \mathbb{N}$.

The reason for the assumption $\boldsymbol{a} \in C_{\boldsymbol{a}}$ is similar to that of $\boldsymbol{x}$. However, when $I(\boldsymbol{a}, \boldsymbol{x})$ is linear in $\boldsymbol{a}$ as in Equation (6), we can take $C_{\boldsymbol{a}}$ to be $[-1, 1]^{n'}$ without loss of generality. This is because the parameters $\boldsymbol{a}$ can be scaled by any positive constant without changing the invariant candidate they define.

When a polynomial template $I(\boldsymbol{a}, \boldsymbol{x})$ is fixed, the invariant conditions can be expressed as constraints in first-order logic:

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}.\ \boldsymbol{q}_{pre}(\boldsymbol{x}) \leq \boldsymbol{0} \implies I(\boldsymbol{a}, \boldsymbol{x}) \leq \boldsymbol{0}, \tag{1}$$

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}.\ I(\boldsymbol{a}, \boldsymbol{x}) \leq \boldsymbol{0} \wedge \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \wedge \boldsymbol{c}_i(\boldsymbol{x}) \leq \boldsymbol{0} \implies I(\boldsymbol{a}, \boldsymbol{f}_i(\boldsymbol{x})) \leq \boldsymbol{0}, \quad i = 1, \ldots, k, \tag{2}$$

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}.\ I(\boldsymbol{a}, \boldsymbol{x}) \leq \boldsymbol{0} \wedge \neg(\boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}) \implies \boldsymbol{q}_{post}(\boldsymbol{x}) \leq \boldsymbol{0}. \tag{3}$$

Definition 3 (Valid and Valid Set). Given a program as presented in Code 1 and a polynomial template $I(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$, a parameter assignment $\boldsymbol{a}_0 \in C_{\boldsymbol{a}}$ is valid if it satisfies constraints (1)–(3),

meaning that the set $\{\boldsymbol{x} \mid I(\boldsymbol{a}_0, \boldsymbol{x}) \leq 0\}$ is an invariant of the program. The valid set, denoted by R_I, represents the collection of all valid parameter assignments for the polynomial template $I(\boldsymbol{a}, \boldsymbol{x})$.

Given a polynomial template for Code 1, we are interested in two problems:

(1) The *weak invariant synthesis problem* asks for an invariant satisfying the template, i.e., finding a valid parameter assignment $\boldsymbol{a}_0 \in C_{\boldsymbol{a}}$.
(2) The *strong invariant synthesis problem* [14] asks for a characterization of all possible invariants satisfying the template, i.e., characterizing the valid set R_I.

2.2 SOS Relaxations

The SOS relaxation is a well-established technique in polynomial optimization. Its basic idea is to approximate a non-convex polynomial optimization problem by a sequence of convex optimization problems. In this part, we introduce necessary concepts related to this technique and demonstrate a typical application. For more comprehensive technical details, please refer to [50, 59].

Putinar's Representation Theorem. A polynomial $p(\boldsymbol{x}) \in \mathbb{R}[\boldsymbol{x}]$ is said to be an SOS polynomial if it can be expressed as $p(\boldsymbol{x}) = \sum_{i=1}^{m} p_i(\boldsymbol{x})^2$, where $p_i(\boldsymbol{x}) \in \mathbb{R}[\boldsymbol{x}]$ and $m \in \mathbb{N}$. Similar to $\mathbb{R}[\boldsymbol{x}]$ and $\mathbb{R}^d[\boldsymbol{x}]$, we use $\Sigma[\boldsymbol{x}]$ and $\Sigma^d[\boldsymbol{x}]$ to denote the set of SOS polynomials and the set of SOS polynomials of degree less than or equal to d in variables $\boldsymbol{x}$, respectively. Since the degree of an SOS polynomial must be even, we have $\Sigma^{2d}[\boldsymbol{x}] = \Sigma^{2d+1}[\boldsymbol{x}]$ for any $d \in \mathbb{N}$.

Definition 4 (Quadratic Module [59]). A subset $\mathbf{Q}$ of $\mathbb{R}[\boldsymbol{x}]$ is called a quadratic module if it contains 1 and is closed under addition and multiplication with squares, i.e.,

$$1 \in \mathbf{Q}, \quad \mathbf{Q} + \mathbf{Q} \subseteq \mathbf{Q}, \quad \text{and} \quad p^2\mathbf{Q} \subseteq \mathbf{Q} \text{ for all } p \in \mathbb{R}[\boldsymbol{x}].$$

Given a vector of polynomials

$$\boldsymbol{p}(\boldsymbol{x}) = \big(p_1(\boldsymbol{x}), \ldots, p_m(\boldsymbol{x})\big), \tag{4}$$

and let $\mathcal{K}$ be the basic closed semialgebraic set defined by $\boldsymbol{p}(\boldsymbol{x}) \geq \mathbf{0}$, i.e.,

$$\mathcal{K} = \big\{\boldsymbol{x} \in \mathbb{R}^n \;\big|\; p_1(\boldsymbol{x}) \geq 0, \ldots, p_m(\boldsymbol{x}) \geq 0\big\} \tag{5}$$

we define the quadratic module generated by $\boldsymbol{p}$ as follows:

Definition 5. Let $\boldsymbol{p}(\boldsymbol{x})$ be defined as above, we denote by $\mathbf{Q}(\boldsymbol{p})$ the smallest quadratic module generated by polynomials in $\boldsymbol{p}$, i.e.,

$$\mathbf{Q}(\boldsymbol{p}) = \left\{\sigma_0 + \sum_{i=1}^{m} \sigma_i p_i \;\middle|\; \sigma_i \in \Sigma[\boldsymbol{x}]\right\}.$$

Furthermore, a quadratic module $\mathbf{Q}(\boldsymbol{p})$ is called Archimedean, or satisfies the Archimedean condition, if $N - \|\boldsymbol{x}\|_2^2 \in \mathbf{Q}(\boldsymbol{p})$ for some constant $N \in \mathbb{N}$.

Since SOS polynomials in $\Sigma[\boldsymbol{x}]$ are non-negative over $\mathbb{R}^n$, it is easy to see that the following lemma holds.

Lemma 1. *Given $\boldsymbol{p}(\boldsymbol{x})$ and $\mathcal{K}$ as defined in Equations (4) and (5), respectively. If a polynomial $f \in \boldsymbol{Q}(\boldsymbol{p})$, then $f(\boldsymbol{x})$ is* non-negative *over $\mathcal{K}$.*

By Lemma 1, the Archimedean condition $N - \|\boldsymbol{x}\|_2^2 = N - \sum_{i=1}^{n} x_i^2 \geq 0$ over $\mathcal{K}$ implies that $\mathcal{K}$ is bounded. On the other hand, if $\mathcal{K}$ is known to be bounded in a n-dimensional ball $\{\boldsymbol{x} \in \mathbb{R}^n \mid N - \|\boldsymbol{x}\|_2^2 \geq 0\}$ for some $N \in \mathbb{N}$, we can introduce a redundant polynomial $p_{m+1} = N - \|\boldsymbol{x}\|_2^2$ and denote $\boldsymbol{p}'(\boldsymbol{x}) = \big(p_1(\boldsymbol{x}), \ldots, p_m(\boldsymbol{x}), p_{m+1}(\boldsymbol{x})\big)$, then $\mathbf{Q}(\boldsymbol{p}')$ is Archimedean. Therefore, the Archimedean condition can be intuitively interpreted as compactness, i.e., being closed and bounded.

Now, we present an important representation theorem in real algebraic geometry, called Putinar's Positivstellensatz [69], which provides a characterization of polynomials that are *locally positive* over a compact basic closed semialgebraic set. For convenience, we use the formulation in [50].

Theorem 1 (Putinar's Positivstellensatz [50, Theorem 2.14]). *Given $\boldsymbol{p}(\boldsymbol{x})$ and $\mathcal{K}$ as defined in Equations (4) and (5), respectively. If $\boldsymbol{Q}(\boldsymbol{p})$ is Archimedean and a polynomial $f \in \mathbb{R}[\boldsymbol{x}]$ is* strictly positive *over $\mathcal{K}$, then $f \in \boldsymbol{Q}(\boldsymbol{p})$.*

Remark 1. When the Archimedean condition is violated (e.g., $\mathcal{K}$ is unbounded), we need to resort to variants of Theorem 1, such as the homogenization formulation [39] and Putinar–Vasilescu Positivstellensatz [58]. For example, the homogenization formulation has been applied in invariant generation for continuous-time dynamical systems [87] and Craig interpolation synthesis [88], a logic inference technique very related to invariant generation. While these extensions allow handling unbounded domains, they come at the cost of increased complexity in the resulting constraints. Nevertheless, these constraints can still be efficiently solved (as SDP, introduced later). To maintain focus on the core technique presented in this article, we will limit our study to the scenario where program variables are bounded. Moreover, our algorithms remain sound even when this condition is violated.

SOS Relaxations. We demonstrate the use of Theorem 1 on a typical template-based synthesis problem. Let $l(\boldsymbol{a}, \boldsymbol{x})$ be a parameterized polynomial that is linear in parameters $\boldsymbol{a}$. For example, one can consider $l(\boldsymbol{a}, \boldsymbol{x})$ to be a polynomial template of degree d in variable $\boldsymbol{x}$ and the parameters $\boldsymbol{a}$ represent the unknown coefficients, i.e.,

$$l(\boldsymbol{a}, \boldsymbol{x}) = \sum_{\|\boldsymbol{\beta}\|_1 \le d} \boldsymbol{a}_{\boldsymbol{\beta}} \boldsymbol{x}^{\boldsymbol{\beta}}, \tag{6}$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n) \in \mathbb{N}^n$ are exponents such that $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^n \beta_j \le d$. For example, when $\boldsymbol{x} = (x_1, x_2)$, a template could be $l(\boldsymbol{a}, \boldsymbol{x}) = a_1 x_1^2 + a_2 x_1 x_2 + a_3 x_2^2 + a_4 x_1 + a_5 x_2 + a_6$, which represents any polynomial in (x_1, x_2) of degree not exceeding 2.

Suppose we want to find a valid parameter assignment $\boldsymbol{a}_0$ such that $l(\boldsymbol{a}_0, \boldsymbol{x}) \le 0$ over the compact basic closed semialgebraic set $\mathcal{K}$ defined by polynomials $\{p_1, \ldots, p_m\}$ as in Equation (5), the problem can be formalized as follows:

$$\begin{aligned} &\text{find} \quad \boldsymbol{a} \\ &s.t. \quad \forall \boldsymbol{x}. \bigwedge_{i=1}^{m} p_i(\boldsymbol{x}) \ge 0 \implies l(\boldsymbol{a}, \boldsymbol{x}) \le 0. \end{aligned} \tag{7}$$

Since $\mathcal{K}$ is bounded, we can assume that the quadratic module $\mathbf{Q}(\boldsymbol{p})$ is Archimedean. By applying Theorem 1, one can construct the following program:

$$\begin{aligned} &\inf \quad \gamma \\ &s.t. \quad \gamma - l(\boldsymbol{a}, \boldsymbol{x}) = \sigma_0(\boldsymbol{x}) + \sum_{i=1}^{m} \sigma_i(\boldsymbol{x}) \cdot p_i(\boldsymbol{x}), \\ &\qquad \sigma_0 \in \Sigma[\boldsymbol{x}], \sigma_i \in \Sigma[\boldsymbol{x}], \quad \text{for } i = 1, \ldots, m, \end{aligned} \tag{8}$$

where γ is a newly introduced variable serving as an upper bound of $l(\boldsymbol{a}, \boldsymbol{x})$ over $\mathcal{K}$. Let γ^* denote the optimal value of Equation (8). The relation between Equations (7) and (8) is stated by the following two theorems.

Theorem 2 (Soundness). *If $\gamma^* < 0$ or the infimum $\gamma^* = 0$ is attainable in Equation (8), then Equation (7) is feasible.*

Proof. Using the assumption, there must exist a feasible solution $\gamma_0 \leq 0$ and $\boldsymbol{a}_0$ such that the constraint in Equation (8) holds. By Lemma 1, we have $\gamma_0 - l(\boldsymbol{a}_0, \boldsymbol{x}) \geq 0$ when $\boldsymbol{x} \in \mathcal{K}$. Therefore, $\boldsymbol{a}_0$ is a solution to Equation (7). □

Theorem 3 (Semi-Completeness). *If Equation (7) is feasible, then $\gamma^* \leq 0$ in Equation (8).*

Proof. If Equation (7) is feasible, let $\boldsymbol{a}_0$ be one feasible solution, then we know $\gamma - l(\boldsymbol{a}_0, \boldsymbol{x}) > 0$ over $\boldsymbol{x} \in \mathcal{K}$ for any $\gamma > 0$,. By Theorem 1, there exist SOS polynomials σ_i for $0 \leq i \leq m$ such that the constraint in Equation (8) holds for any $\gamma > 0$. Hence, the infimum $\gamma^* \leq 0$. □

In [68], Parrilo showed that Equation (8) can be approximated by solving a series of its relaxations. The idea is to impose restrictions on the highest degree of involved polynomials in the constraints. Recall that d is the degree of $l(\boldsymbol{a}, \boldsymbol{x})$ in variable $\boldsymbol{x}$. Given a relaxation order $d_r \in \mathbb{N}$ with $2d_r \geq \max\{d, \deg(p_1), \ldots, \deg(p_m)\}$, we set the degrees of the unknown SOS polynomials appropriately such that the maximum degree of polynomials involved in Equation (8) equals $2d_r$. The resulting program is referred to as the d_rth SOS relaxation of Equation (8):

$$\begin{aligned} \min \quad & \gamma \\ s.t. \quad & \gamma - l(\boldsymbol{a}, \boldsymbol{x}) = \sigma_0(\boldsymbol{x}) + \sum_{i=1}^{m} \sigma_i(\boldsymbol{x}) \cdot p_i(\boldsymbol{x}), \\ & \sigma_0 \in \Sigma^{2d_r}[\boldsymbol{x}], \sigma_i \in \Sigma^{2\lfloor \frac{2d_r - \deg(p_i)}{2} \rfloor}[\boldsymbol{x}], \quad \text{for } i = 1, \ldots, m, \end{aligned} \tag{9}$$

where $\lfloor \cdot \rfloor$ returns the largest integer less than or equal to the argument. As the relaxation order d_r increases and approaches infinity, the optimal value of Equation (9) converges to the optimal value of Equation (8) [49]. In other words, the series of SOS relaxations yields progressively more accurate approximations of the original problem Equation (7). One distinct advantage of Equation (9) is that it can be solved as an SDP.

Remark 2. Whether the infimum γ^* is attainable in Theorem 2 depends on whether the solutions of the SOS relaxations Equation (9) converge in finitely many steps. Hence, this is also referred to as the finite convergence property; [66, Theorem 1.1] shows that this property is decidable by checking that the constraints are not in some pathological forms, which generally holds. In practice, since we only deal with SOS relaxations, we do not need to consider this problem and treat it as a technical assumption.

SDP Translation. Let S_n denote the set of symmetric $n \times n$ real matrices. A matrix $X \in S_n$ is *positive semidefinite* if all its eigenvalues are non-negative, denoted by $X \succeq \mathbf{0}$.

Definition 6 (Standard Form SDP [13, Section 4.6.2]). A standard form SDP has linear equality constraints and a matrix non-negativity constraint on the variable $X \in S_n$:

$$\begin{aligned} \min \quad & \mathbf{tr}(CX) \\ s.t. \quad & \mathbf{tr}(A_i X) = b_i, \quad i = 1, \ldots, k \\ & X \succeq \mathbf{0}. \end{aligned} \tag{10}$$

where $C, A_1, \ldots, A_k \in S_n$ for some $k \in \mathbb{N}$, and $\mathbf{tr}(\cdot)$ is the trace operator, i.e., $\mathbf{tr}(CX) = \sum_{i,j=1}^{n} C_{ij} X_{ji}$.

Let $\mathbf{m}_d(\boldsymbol{x})$ be a column vector with all monomials in $\boldsymbol{x} \in \mathbb{R}^n$ of degree up to d. For example, when $\boldsymbol{x} = (x_1, x_2)$, $\mathbf{m}_2(\boldsymbol{x}) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$. Any polynomial $p(\boldsymbol{x}) \in \mathbb{R}^{2d}[\boldsymbol{x}]$ can be represented by

$$p(\boldsymbol{x}) = \mathbf{m}_d^\top(\boldsymbol{x}) C_p \mathbf{m}_d(\boldsymbol{x}),$$

where $\mathbf{m}_d^\top(\boldsymbol{x})$ is the transpose of $\mathbf{m}_d(\boldsymbol{x})$ and $C_p \in S_{\binom{n+d}{d}}$ is called the *Gram matrix* of p. An important theorem states that a polynomial p is an SOS polynomial if and only if its Gram matrix is positive semidefinite, i.e., $C_p \succeq \mathbf{0}$ [50, Proposition 2.1].

For each relaxation order $d_r \in \mathbb{N}$, Equation (9) can be translated into an equivalent SDP:

$$\begin{aligned} \min \quad & \gamma \\ s.t. \quad & C_{\gamma - l(\boldsymbol{a},\boldsymbol{x})} = C_{\sigma_0} + \sum_{i=1}^{m} C_{\sigma_i \cdot p_i} \\ & \operatorname{diag}(C_{\sigma_0}, C_{\sigma_1}, \ldots, C_{\sigma_m}) \succeq \mathbf{0}, \end{aligned} \tag{11}$$

where $C_{\sigma_i \cdot p_i}$ denotes the Gram matrix of the product $\sigma_i \cdot p_i$ for $i = 1, \ldots, m$, and $\operatorname{diag}(C_{\sigma_0}, C_{\sigma_1}, \ldots, C_{\sigma_m})$ is a block-diagonal matrix. One can check that Equation (11) conforms to the standard form of SDP in Definition 6.

Roughly speaking, the complexity for solving Equation (11) depends on the maximum size of Gram matrices C_{σ_i}, which is at most $\binom{n+d_r}{n} \times \binom{n+d_r}{n}$ [75]. Note that $\binom{n+d_r}{n}$ is a polynomial in n for a fixed d_r and vice versa, but is not a polynomial in both n and d_r. Since an SDP can be solved in polynomial time, for example, by interior point methods [13], the complexity for solving Equation (9) is polynomial in $\binom{n+d_r}{n}$.

In the following remark, we briefly explain the difficulty in applying Theorem 1 to the invariant synthesis problems presented at the end of Section 2.1.

Remark 3. Let $I(\boldsymbol{a}, \boldsymbol{x})$ be a polynomial template that is linear in the parameters $\boldsymbol{a}$. For the invariant condition Equation (1), since the parameters $\boldsymbol{a}$ occur after the implication symbol, the SOS relaxations of Equation (1) can still be translated to SDP, similar to Equation (7). However, Equations (2) and (3) cannot be handled in the same manner as the parameters $\boldsymbol{a}$ occur before the implication symbol. In other words, solving Equations (2) and (3) requires one to solve a program of the following form:

$$\begin{aligned} \mathit{find} \quad & \boldsymbol{a} \\ s.t. \quad & \forall \boldsymbol{x}. \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0 \implies l(\boldsymbol{a}, \boldsymbol{x}) \leq 0, \end{aligned} \tag{12}$$

where $p_i(\boldsymbol{a}, \boldsymbol{x}), l(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$.

Obviously, Equation (12) is a generalization of Equation (7), allowing polynomials p_i to contain unknown parameters $\boldsymbol{a}$. Unfortunately, Equation (12) is much harder to solve. To see this, we apply Theorem 1 to transform it into a program involving SOS polynomials, assuming the Archimedean condition:

$$\begin{aligned} \min \quad & \gamma \\ s.t. \quad & \gamma - l(\boldsymbol{a}, \boldsymbol{x}) = \sigma_0(\boldsymbol{x}) + \sum_{i=1}^{m} \sigma_i(\boldsymbol{x}) \cdot p_i(\boldsymbol{a}, \boldsymbol{x}), \\ & \sigma_0 \in \Sigma[\boldsymbol{x}], \sigma_i \in \Sigma[\boldsymbol{x}], \quad \text{for } i = 1, \ldots, m, \end{aligned} \tag{13}$$

where the decision variables are parameters $\boldsymbol{a}$ and the unknown coefficients in SOS polynomials σ_i, for $i = 0, \ldots, m$. Similarly, by restricting the highest degree of involved polynomials in constraints, we obtain a series of SOS relaxations of the above program.

However, the resulting SOS relaxations of Equation (13) cannot be translated into SDPs because the Gram matrices of the products $\sigma_i(\boldsymbol{x}) \cdot p_i(\boldsymbol{a}, \boldsymbol{x})$ contain bilinear terms arising from the product of unknown coefficients in σ_i and parameters $\boldsymbol{a}$. These constraints, known as BMIs in optimization theory, are incompatible with the linear matrix inequalities allowed in SDPs. As shown in [83]

and [11], solving general BMI optimization problems is **NP**-hard. In [14] and [30], the constraints are further encoded into quadratic programming by applying Cholesky decomposition [31] to Gram matrices of SOS polynomials. However, solving non-convex quadratic programming is still **NP**-hard [76].

3 Synthesizing Strong Invariants from Polynomial Templates

In this section, we propose the *Cluster algorithm* to give an approximate solution to the strong invariant synthesis problem. To this end, we leverage Lasserre's technique from [51] to construct a series of SOS relaxations for under-approximating the valid set R_I. By solving these relaxations as SDPs, we obtain a sequence of polynomials $h_d(\boldsymbol{a})$ for $d \in \mathbb{N}$ whose 0-sublevel sets are subsets of R_I. Furthermore, we show that these under-approximations possess desirable properties, including soundness, convergence, and semi-completeness. Moreover, these under-approximations can be utilized to simplify the weak invariant synthesis problem, which reduces to finding a solution to $h_d(\boldsymbol{a}) \leq 0$ over $\boldsymbol{a} \in C_{\boldsymbol{a}}$.

3.1 Lasserre's Technique

In this part, we demonstrate how to apply Lasserre's technique [51] to deal with Equation (12). Instead of attempting to find a valid assignment of $\boldsymbol{a} \in C_{\boldsymbol{a}}$, our objective is to under-approximate the set of valid assignments of $\boldsymbol{a}$, denoted by

$$R = \left\{ \boldsymbol{a} \in C_{\boldsymbol{a}} \;\middle|\; \forall \boldsymbol{x}. \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0 \implies l(\boldsymbol{a}, \boldsymbol{x}) \leq 0 \right\}, \tag{14}$$

where $p_i(\boldsymbol{a}, \boldsymbol{x}), l(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$. We assume that variables $\boldsymbol{x}$ and parameters $\boldsymbol{a}$ are both bounded within some predefined hyper-rectangles $C_{\boldsymbol{x}}$ and $C_{\boldsymbol{a}}$, respectively. Without loss of generality, we can further assume that $p_i(\boldsymbol{a}, \boldsymbol{x})$ include those polynomials that define $C_{\boldsymbol{x}}$ and $C_{\boldsymbol{a}}$, ensuring that $\mathbf{Q}(p_1, \ldots, p_m)$ is Archimedean.

Let $\mathcal{K}_{\boldsymbol{a}} = \{\boldsymbol{x} \in C_{\boldsymbol{x}} \mid \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0\}$ be a basic closed semialgebraic set parameterized by $\boldsymbol{a}$. When $\mathcal{K}_{\boldsymbol{a}}$ is non-empty for every $\boldsymbol{a} \in C_{\boldsymbol{a}}$, we can express R as

$$R = \{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid J(\boldsymbol{a}) \leq 0\}, \tag{15}$$

where $J(\boldsymbol{a}) = \sup_{\boldsymbol{x} \in \mathcal{K}_{\boldsymbol{a}}} l(\boldsymbol{a}, \boldsymbol{x})$.

Therefore, if we can find a function $h(\boldsymbol{a})$ such that $h(\boldsymbol{a}) \geq J(\boldsymbol{a})$ for all $\boldsymbol{a} \in C_{\boldsymbol{a}}$, then the 0-sublevel set of $h(\boldsymbol{a})$ serves as an under-approximation of R, i.e.,

$$\{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid h(\boldsymbol{a}) \leq 0\} \subseteq \{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid J(\boldsymbol{a}) \leq 0\} = R. \tag{16}$$

Now the problem boils down to finding such a function $h(\boldsymbol{a})$. Ideally, we would like $h(\boldsymbol{a})$ to be a simple expression that closely approximates $J(\boldsymbol{a})$. Fortunately, according to the following Lemma 2 and Theorem 4, we can restrict our search to polynomials for $h(\boldsymbol{a})$.

Lemma 2 [51, Lemma 1]. *The function J as defined in Equation (15) is upper semi-continuous, i.e., for all $\boldsymbol{a}_0 \in C_{\boldsymbol{a}}$, we have*

$$\limsup_{\boldsymbol{a} \to \boldsymbol{a}_0} J(\boldsymbol{a}) \leq J(\boldsymbol{a}_0). \tag{17}$$

Theorem 4 [51, Theorem 1]. *Let $C_{\boldsymbol{a}} \subset \mathbb{R}^{n'}$ be a compact set and $J(\boldsymbol{a}) : C_{\boldsymbol{a}} \to \mathbb{R}$ be a bounded and upper semi-continuous function. Then there exists a sequence of polynomials $\{h_i(\boldsymbol{a}) \mid i \in \mathbb{N}\} \subset \mathbb{R}[\boldsymbol{a}]$ such that $h_i(\boldsymbol{a}) \geq J(\boldsymbol{a})$ over $\boldsymbol{a} \in C_{\boldsymbol{a}}$ for all $i \in \mathbb{N}$ and*

$$\lim_{i \to \infty} \int_{C_{\boldsymbol{a}}} |h_i(\boldsymbol{a}) - J(\boldsymbol{a})| \mathrm{d}\boldsymbol{a} = 0. \tag{18}$$

In what follows, we show how to compute such a polynomial $h(\boldsymbol{a})$ by employing SOS relaxations. First, using the definition of $J(\boldsymbol{a})$, we have $h(\boldsymbol{a}) - J(\boldsymbol{a}) \geq 0$ over $\boldsymbol{a} \in C_{\boldsymbol{a}}$ if and only if

$$\forall(\boldsymbol{a}, \boldsymbol{x}) \in C_{\boldsymbol{a}} \times \mathcal{K}_{\boldsymbol{a}}.\ h(\boldsymbol{a}) - l(\boldsymbol{a}, \boldsymbol{x}) \geq 0. \tag{19}$$

Thus, the problem amounts to solving the following program:

$$\begin{aligned} \inf \quad & \frac{1}{\mu(C_{\boldsymbol{a}})} \int_{C_{\boldsymbol{a}}} h(\boldsymbol{a}) \mathrm{d}\boldsymbol{a} \\ s.t. \quad & \forall(\boldsymbol{a}, \boldsymbol{x}).\ \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0 \implies h(\boldsymbol{a}) - l(\boldsymbol{a}, \boldsymbol{x}) \geq 0, \end{aligned} \tag{20}$$

where $\mu(C_{\boldsymbol{a}})$ is the volume of $C_{\boldsymbol{a}}$ and the objective function is the scaled integral of $h(\boldsymbol{a})$ over C_a. Since $J(\boldsymbol{a})$ is fixed, minimizing $\int_{C_{\boldsymbol{a}}} h(\boldsymbol{a}) \mathrm{d}\boldsymbol{a}$ is the same as minimizing $\int_{C_{\boldsymbol{a}}} |h(\boldsymbol{a}) - J(\boldsymbol{a})| \mathrm{d}\boldsymbol{a}$, i.e., the gap between $h(\boldsymbol{a})$ and $J(\boldsymbol{a})$ over $C_{\boldsymbol{a}}$.

The main difference between Equations (20) and (13) is how parameters $\boldsymbol{a}$ are quantified in constraints: In Equation (13), $\boldsymbol{a}$ are associated with an (implicit) existential quantifier, while in Equation (20) a universal quantifier. As a result, for Equation (20), we can treat parameters $\boldsymbol{a}$ equally as variables $\boldsymbol{x}$. After applying Theorem 1, we have

$$\begin{aligned} \inf \quad & \frac{1}{\mu(C_{\boldsymbol{a}})} \int_{C_{\boldsymbol{a}}} h(\boldsymbol{a}) \mathrm{d}\boldsymbol{a} \\ s.t. \quad & h(\boldsymbol{a}) - l(\boldsymbol{a}, \boldsymbol{x}) = \sigma_0(\boldsymbol{a}, \boldsymbol{x}) + \sum_{i=1}^{m} \sigma_i(\boldsymbol{a}, \boldsymbol{x}) \cdot p_i(\boldsymbol{a}, \boldsymbol{x}), \\ & \sigma_0(\boldsymbol{a}, \boldsymbol{x}), \sigma_i(\boldsymbol{a}, \boldsymbol{x}) \in \Sigma[\boldsymbol{a}, \boldsymbol{x}], \text{ for } i = 1, \ldots, m. \end{aligned} \tag{21}$$

It is worthwhile to note that SOS polynomials $\sigma_i(\boldsymbol{a}, \boldsymbol{x})$ belong to $\Sigma[\boldsymbol{a}, \boldsymbol{x}]$, while SOS polynomials $\sigma_i(\boldsymbol{x}) \in \Sigma[\boldsymbol{x}]$ in Equation (13).

When $h(\boldsymbol{a})$ is of degree d, it can be expressed as $h(\boldsymbol{a}) = \sum_{\boldsymbol{\beta}} \mathrm{h}_{\boldsymbol{\beta}} \boldsymbol{a}^{\boldsymbol{\beta}}$ where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{n'}) \in \mathbb{N}^{n'}$ are exponents such that $\sum_{j=1}^{n'} \beta_j \leq d$ and $\mathrm{h}_{\boldsymbol{\beta}}$ are unknown coefficients in $\mathbb{R}$. Then, the objective function in Equation (21) can be simplified into a linear expression in coefficients $h_{\boldsymbol{\beta}}$:

$$\begin{aligned} \frac{1}{\mu(C_{\boldsymbol{a}})} \int_{C_{\boldsymbol{a}}} h(\boldsymbol{a}) \mathrm{d}\boldsymbol{a} &= \frac{1}{\mu(C_{\boldsymbol{a}})} \int_{C_{\boldsymbol{a}}} \left(\sum_{\boldsymbol{\beta}} \mathrm{h}_{\boldsymbol{\beta}} \boldsymbol{a}^{\boldsymbol{\beta}} \right) \mathrm{d}\boldsymbol{a} \\ &= \sum_{\boldsymbol{\beta}} \left(\frac{1}{\mu(C_{\boldsymbol{a}})} \int_{C_{\boldsymbol{a}}} \boldsymbol{a}^{\boldsymbol{\beta}} \mathrm{d}\boldsymbol{a} \right) \mathrm{h}_{\boldsymbol{\beta}}. \end{aligned} \tag{22}$$

When $C_{\boldsymbol{a}}$ is a hyper-rectangle (or other simple shapes like ellipses), the integral $\int_{C_{\boldsymbol{a}}} \boldsymbol{a}^{\boldsymbol{\beta}} \mathrm{d}\boldsymbol{a}$ will be easy to compute.

Now we present SOS relaxations of Equation (21). Assuming that $h(\boldsymbol{a})$ is of degree d, let d_r be the smallest natural number such that $2d_r \geq \max\{d, \deg(l), \deg(p_1), \ldots, \deg(p_m)\}$, then the d_rth

SOS relaxation of Equation (21) is given by

$$\begin{aligned} \min \quad & \sum_{\boldsymbol{\beta}} \gamma_{\boldsymbol{\beta}} \mathrm{h}_{\boldsymbol{\beta}} \\ s.t. \quad & h(\boldsymbol{a}) = \sum_{\boldsymbol{\beta}} \mathrm{h}_{\boldsymbol{\beta}} \boldsymbol{a}^{\boldsymbol{\beta}} \in \mathbb{R}^{d}[\boldsymbol{a}], \\ & h(\boldsymbol{a}) - l(\boldsymbol{a}, \boldsymbol{x}) = \sigma_0(\boldsymbol{a}, \boldsymbol{x}) + \sum_{i=1}^{m} \sigma_i(\boldsymbol{a}, \boldsymbol{x}) \cdot p_i(\boldsymbol{a}, \boldsymbol{x}), \\ & \sigma_0 \in \Sigma^{2d_r}[\boldsymbol{x}], \sigma_i \in \Sigma^{2\lfloor \frac{2d_r - \deg(p_i)}{2} \rfloor}[\boldsymbol{x}], \quad \text{for } i = 1, \ldots, m, \end{aligned} \tag{23}$$

where $\gamma_{\boldsymbol{\beta}} = \frac{1}{\mu(C_{\boldsymbol{a}})} \int_{C_{\boldsymbol{a}}} \boldsymbol{a}^{\boldsymbol{\beta}} \mathrm{d}\mu(\boldsymbol{a})$.

For each $d \in \mathbb{N}$, mirroring the process in Section 2.2, Equation (23) can be translated into an SDP. If this SDP is solvable, we can use the solutions $\{h_{\boldsymbol{\beta}}\}_{\boldsymbol{\beta}}$ to construct a polynomial $h_d(\boldsymbol{a}) = \sum_{\boldsymbol{\beta}} \mathrm{h}_{\boldsymbol{\beta}} \boldsymbol{a}^{\boldsymbol{\beta}}$, which serves as the dth approximation of $h(\boldsymbol{a})$. If the translated SDP is not solvable, we set $h_d(\boldsymbol{a}) = 1$. In this case, $R_d = \{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid 1 \leq 0\} = \emptyset$ is a trivial under-approximation of R. Moreover, we have the following theorem:

Theorem 5 [51, Theorem 5]. *Assume that R has non-empty interior and $\mathcal{K}_{\boldsymbol{a}}$ is non-empty for every $\boldsymbol{a} \in C_{\boldsymbol{a}}$, then $h_d(\boldsymbol{a})$ converges to $J(\boldsymbol{a})$ (from above) as d goes to ∞, i.e.,*

$$\lim_{d \to \infty} \int_{C_{\boldsymbol{a}}} |h_d(\boldsymbol{a}) - J(\boldsymbol{a})| \mathrm{d}\boldsymbol{a} = 0.$$

3.2 Cluster Algorithm

In this part, we show how to apply Lasserre's technique to solve the strong invariant synthesis problem.

Similar to Equation (15), we first express the valid set R_I as a 0-sublevel set. Mimicking the definition of J, let us define $J_1(\boldsymbol{a}), \ldots, J_{k+2}(\boldsymbol{a})$ as follows (recall that k is the number of branches):

$$J_1(\boldsymbol{a}) = \sup_{\boldsymbol{x} \in \mathcal{K}_{\boldsymbol{a},1}} I(\boldsymbol{a}, \boldsymbol{x}), \text{ with}$$
$$\mathcal{K}_{\boldsymbol{a},1} = \{\boldsymbol{x} \in C_{\boldsymbol{x}} \mid \boldsymbol{q}_{pre}(\boldsymbol{x}) \leq \boldsymbol{0}\}, \tag{24}$$
$$J_{i+1}(\boldsymbol{a}) = \sup_{\boldsymbol{x} \in \mathcal{K}_{\boldsymbol{a},i+1}} I(\boldsymbol{a}, \boldsymbol{f}_i(\boldsymbol{x})), \text{ with}$$
$$\mathcal{K}_{\boldsymbol{a},i+1} = \{\boldsymbol{x} \in C_{\boldsymbol{x}} \mid I(\boldsymbol{a}, \boldsymbol{x}) \leq \boldsymbol{0}, \boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0}, \boldsymbol{c}_i(\boldsymbol{x}) \leq \boldsymbol{0}\}, \text{ for } i = 1, \ldots, k, \tag{25}$$
$$J_{k+2}(\boldsymbol{a}) = \sup_{\boldsymbol{x} \in \mathcal{K}_{\boldsymbol{a},k+2}} \boldsymbol{q}_{post}(\boldsymbol{x}), \text{ with}$$
$$\mathcal{K}_{\boldsymbol{a},k+2} = \{\boldsymbol{x} \in C_{\boldsymbol{x}} \mid I(\boldsymbol{a}, \boldsymbol{x}) \leq \boldsymbol{0} \wedge \neg(\boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0})\}. \tag{26}$$

It is straightforward to see that Equations (24)–(26) correspond to Equations (1)–(3), respectively. Then, we define

$$J(\boldsymbol{a}) = \max\{J_1(\boldsymbol{a}), \cdots, J_{k+2}(\boldsymbol{a}), -1\}, \tag{27}$$

where an addition constant -1 (or any other negative real number) is introduced to ensure $J(\boldsymbol{a}) \neq -\infty$ in an extreme case when $\mathcal{K}_{\boldsymbol{a},i} = \emptyset$ for $i = 1, \ldots, k+2$. By definition, the valid set R_I is the 0-sublevel set of function $J(\boldsymbol{a})$, i.e.,

$$R_I = \{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid J(\boldsymbol{a}) \leq 0\}. \tag{28}$$

In order to obtain a close under-approximation of R_I, we try to find a polynomial $h(\boldsymbol{a})$ that approaches $J(\boldsymbol{a})$ over $C_{\boldsymbol{a}}$ from above. According to Equation (20), the problem is reduced to the following program:

$$\begin{aligned}&\inf \quad \frac{1}{\mu(C_{\boldsymbol{a}})}\int_{C_{\boldsymbol{a}}} h(\boldsymbol{a})\mathrm{d}\boldsymbol{a}\\ &s.t. \quad \forall(\boldsymbol{a},\boldsymbol{x})\in C_{\boldsymbol{a}}\times C_{\boldsymbol{x}}.\\ &\begin{cases} \boldsymbol{q}_{pre}(\boldsymbol{x})\le \boldsymbol{0} \implies h(\boldsymbol{a})-I(\boldsymbol{a},\boldsymbol{x})\ge 0,\\ I(\boldsymbol{a},\boldsymbol{x})\le \boldsymbol{0}\wedge \boldsymbol{g}(\boldsymbol{x})\le \boldsymbol{0}\wedge \boldsymbol{c}_i(\boldsymbol{x})\le \boldsymbol{0} \implies h(\boldsymbol{a})-I(\boldsymbol{a},\boldsymbol{f}_i(\boldsymbol{x}))\ge 0, \text{ for } i=1,\dots,k,\\ I(\boldsymbol{a},\boldsymbol{x})\le \boldsymbol{0}\wedge \neg(\boldsymbol{g}(\boldsymbol{x})\le \boldsymbol{0}) \implies h(\boldsymbol{a})-\boldsymbol{q}_{post}(\boldsymbol{x})\ge 0,\\ h(\boldsymbol{a})+1\ge 0, \end{cases}\end{aligned} \tag{29}$$

where the first three constraints correspond to Equations (24)–(26) (or the invariant conditions), and the last constraint corresponds to the additional value -1 in Equation (27).

For simplicity, we assume that $\boldsymbol{q}_{pre}, \boldsymbol{q}_{post}, \boldsymbol{g}, \boldsymbol{c}_i$ are polynomials (instead of vectors of polynomials) and use q_{pre}, q_{post}, g, and c_i instead. We also assume that $C_{\boldsymbol{x}} = \{\boldsymbol{x}\in\mathbb{R}^n \mid N-x_1^2\ge 0,\dots,N-x_n^2\ge 0\}$ and $C_{\boldsymbol{a}} = \{\boldsymbol{x}\in\mathbb{R}^{n'} \mid 1-a_1^2\ge 0,\dots,1-a_{n'}^2\ge 0\}$. Let polynomial $h(\boldsymbol{a})$ be of degree d, we translate Equation (29) into constraints with SOS polynomials:

$$\begin{aligned}&\inf \quad \sum_{\boldsymbol{\beta}}\gamma_{\boldsymbol{\beta}}\mathrm{h}_{\boldsymbol{\beta}}\\ &s.t. \quad h(\boldsymbol{a}) = \sum_{\boldsymbol{\beta}}\mathrm{h}_{\boldsymbol{\beta}}\boldsymbol{a}^{\boldsymbol{\beta}}\in\mathbb{R}^d[\boldsymbol{a}],\\ &\quad h(\boldsymbol{a})-I(\boldsymbol{a},\boldsymbol{x}) = \sigma_{0,0}-\sigma_{0,1}\cdot q_{pre}(\boldsymbol{a},\boldsymbol{x})+\sum_{j=1}^{n}\sigma_{0,j}^{\boldsymbol{x}}\cdot(N-x_j^2)+\sum_{j=1}^{n'}\sigma_{0,j}^{\boldsymbol{a}}\cdot(1-a_j^2),\\ &\quad h(\boldsymbol{a})-I(\boldsymbol{a},f_i(\boldsymbol{x})) = \sigma_{i,0}-\sigma_{i,1}\cdot I(\boldsymbol{a},\boldsymbol{x})-\sigma_{i,2}\cdot g(\boldsymbol{x})-\sigma_{i,3}\cdot c_i(\boldsymbol{x})\\ &\qquad\qquad +\sum_{j=1}^{n}\sigma_{i,j}^{\boldsymbol{x}}\cdot(N-x_j^2)+\sum_{j=1}^{n'}\sigma_{i,j}^{\boldsymbol{a}}\cdot(1-a_j^2), \text{ for } i=1,\dots,k\\ &h(\boldsymbol{a})-q_{post}(\boldsymbol{x}) = \sigma_{k+1,0}-\sigma_{k+1,1}\cdot I(\boldsymbol{a},\boldsymbol{x})+\sigma_{k+1,2}\cdot g(\boldsymbol{x})\\ &\qquad\qquad +\sum_{j=1}^{n}\sigma_{k+1,j}^{\boldsymbol{x}}\cdot(N-x_j^2)+\sum_{j=1}^{n'}\sigma_{k+1,j}^{\boldsymbol{a}}\cdot(1-a_j^2),\\ &h(\boldsymbol{a})+1 = \sigma_{k+2,0}+\sum_{j=1}^{n}\sigma_{k+2,j}^{\boldsymbol{x}}\cdot(N-x_j^2)+\sum_{j=1}^{n'}\sigma_{k+2,j}^{\boldsymbol{a}}\cdot(1-a_j^2),\\ &\sigma_{i,j},\sigma_{i,j}^{\boldsymbol{x}},\sigma_{i,j}^{\boldsymbol{a}}\in\Sigma[\boldsymbol{a},\boldsymbol{x}] \text{ for all pair } (i,j),\end{aligned} \tag{30}$$

where, according to the definition of $C_{\boldsymbol{a}}$,

$$\gamma_{\boldsymbol{\beta}} = \frac{1}{2^{n'}}\int_{C_{\boldsymbol{a}}}\boldsymbol{a}^{\boldsymbol{\beta}}\mathrm{d}\boldsymbol{a} = \begin{cases}0 & \text{if } \beta_i \text{ is odd for some } i,\\ \prod_{i=1}^{n}(\beta_i+1)^{-1} & \text{otherwise.}\end{cases} \tag{31}$$

The Cluster algorithm takes as input a program of the form Code 1, a polynomial template $I(\boldsymbol{a},\boldsymbol{x})\in\mathbb{R}[\boldsymbol{a},\boldsymbol{x}]$, and an upper bound $D\in\mathbb{N}$ on the degree of $h(\boldsymbol{a})$. For each degree d such that $1\le d\le D$, the algorithm tries to find a polynomial $h_d(\boldsymbol{a})$ of degree d by solving the d_rth SOS relaxation of Equation (30), where d_r is the smallest natural number such that $2d_r$ is larger than or

Algorithm 1: The Cluster Algorithm

Input : A program $\mathcal{P}$ of the form Code 1, a polynomial template $I(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$, and an upper bound $D \in \mathbb{N}$ on the degree of $h(\boldsymbol{a})$.

Output : A sequence of under-approximations $\{R_{I,i} \mid 1 \le i \le D\}$.

1 Construct Equation (30) using $\mathcal{P}$ and $I(\boldsymbol{a}, \boldsymbol{x})$;
2 $d \leftarrow 1$;
3 **while** $d \le D$ **do**
4 $d_{\max} \leftarrow$ the largest degree of polynomials in Equation (30);
5 $d_r \leftarrow \lfloor \frac{d_{\max}+1}{2} \rfloor$;
6 **if** *the d_r-th SOS relaxation of Equation (30) is solvable* **then**
7 $\{h_{\boldsymbol{\beta}}\}_{\boldsymbol{\beta}} \leftarrow$ Solve the d_r-th SOS relaxation of Equation (30);
8 $h_d(\boldsymbol{a}) \leftarrow \sum_{\boldsymbol{\beta}} \mathrm{h}_{\boldsymbol{\beta}} \boldsymbol{a}^{\boldsymbol{\beta}}$; ▷ construct $h_d(\boldsymbol{a})$ using coefficients
9 **else**
10 $h_d(\boldsymbol{a}) \leftarrow 1$; ▷ constant polynomial
11 **end**
12 $R_{I,d} \leftarrow \{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid h_d(\boldsymbol{a}) \le 0\}$;
13 ▷ a valid parameter assignment can be obtained by solving $h_d(\boldsymbol{a}) \le 0$
14 $d \leftarrow d + 1$;
15 **end**
16 **return** $R_{I,1}, \ldots, R_{I,D}$; ▷ a sequence of under-approximations of R_I

equal to the maximum degree of polynomials occurring in Equation (29). If the program is solvable, an under-approximation of R_I is given by

$$R_{I,d} = \{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid h_d(\boldsymbol{a}) \le 0\} \subseteq R_I. \tag{32}$$

If not solvable, we set $h_d(\boldsymbol{a}) = 1$ and $R_{I,d} = \{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid 1 \le 0\} = \emptyset$. Finally, the algorithm outputs the sequence $\{R_{I,d} \mid 1 \le d \le D\}$. The pseudocode of the algorithm is presented in Algorithm 1.

The Cluster algorithm can also be adapted to solve the weak invariant synthesis problem. When $h_d(\boldsymbol{a})$ is obtained for some $d \in \mathbb{N}$, we know that any parameter assignment $\boldsymbol{a} \in R_{I,d}$ is valid. From this perspective, the polynomial $h_d(\boldsymbol{a})$ characterizes a *cluster* of invariants of similar shapes. To synthesize a valid assignment $\boldsymbol{a}$ such that $h(\boldsymbol{a}) \le 0$, we only need to solve the constraint $\exists \boldsymbol{a} \in C_{\boldsymbol{a}}.h(\boldsymbol{a}) \le 0$ (see the comment on line 13 in Algorithm 1), which is usually simpler than the original invariant conditions Equations (1)–(3) and can be tackled by many modern optimization tools or SMT solvers.

Remark 4. One minor (theoretical) advantage of the Cluster algorithm is that it can handle templates $I(\boldsymbol{a}, \boldsymbol{x})$ non-linear in parameters $\boldsymbol{a}$, as we only assume that $I(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$. This is not the case for many existing approaches [1, 14, 30, 55] (discussed in Section 7), where the templates are required to contain linear parameters so that the constraints can be encoded into desired forms. However, since in practice parameters often represent unknown coefficients, we still mainly focus on the case when templates are linear in parameters.

3.3 Soundness, Convergence, and Semi-Completeness

Now we prove the output $\{R_{I,d} \mid 1 \le d \le D\}$ of the Cluster algorithm has many desired properties.

Theorem 6 (Soundness). *Given $D \in \mathbb{N}$, $R_{I,d}$ is an under-approximation of the valid set R_I, i.e., $R_{I,d} \subseteq R_I$, for any d such that $1 \le d \le D$.*

Proof. We first show that a feasible solution to Equation (30) is also a solution to Equation (29). This is achieved by applying Lemma 1 to each constraint in Equation (30). For example, assume that there exists a polynomial $h(\boldsymbol{a})$ such that the following constraint in Equation (30) holds:

$$h(\boldsymbol{a}) - I(\boldsymbol{a},\boldsymbol{x}) = \sigma_{0,0} - \sigma_{0,1}\cdot q_{pre}(\boldsymbol{a},\boldsymbol{x}) + \sum_{j=1}^{n}\sigma_{0,j}^{\boldsymbol{x}}\cdot(N-x_j^2) + \sum_{j=1}^{n'}\sigma_{0,j}^{\boldsymbol{a}}\cdot(1-a_j^2)$$

for some SOS polynomials $\sigma_{0,0},\ldots,\sigma_{0,j}^{\boldsymbol{a}} \in \Sigma[\boldsymbol{a},\boldsymbol{x}]$. By applying Lemma 1, we have

$$\bigwedge_{j=1}^{n} N - x_j^2 \ge 0 \wedge \bigwedge_{j=1}^{n'} N - a_j^2 \ge 0 \wedge q_{pre}(\boldsymbol{a},\boldsymbol{x}) \le 0 \implies h(\boldsymbol{a}) - I(\boldsymbol{a},\boldsymbol{x}) \ge 0,$$

which corresponds to the first constraint in Equation (29), i.e.,

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}, \boldsymbol{a} \in C_{\boldsymbol{a}}.\ q_{pre}(\boldsymbol{a},\boldsymbol{x}) \le 0 \implies h(\boldsymbol{a}) - I(\boldsymbol{a},\boldsymbol{x}) \ge 0.$$

Therefore, if the d_rth SOS relaxation of Equation (30) is solvable, we have $h_d(\boldsymbol{a}) \ge J(\boldsymbol{a})$ over $C_{\boldsymbol{a}}$, which implies $R_{I,d} \subseteq R_I$. If not solvable, we have $R_{I,d} = \emptyset \subseteq R_I$. □

Theorem 7 (Convergence). *If the set $\{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid J(\boldsymbol{a}) = 0\}$ has Lebesgue measure zero, then we have*

$$\lim_{D\to\infty} \mu(R_I \setminus R_{I,D}) = 0.$$

Proof. Essentially the same as the proof of [51, Theorem 3]. □

The condition in Theorem 7 is to ensure that

$$\mu(\{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid J(\boldsymbol{a}) \le 0\}) = \mu(\{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid J(\boldsymbol{a}) < 0\}), \tag{33}$$

meaning that the infimum value of Equation (30) is not attainable at most over a region of measure 0. This assumption is made similar to the assumption in Theorem 2. As mentioned in Remark 2, this is just a technical assumption and usually holds in practice.

Before presenting the semi-completeness result, we introduce the following definition.

Definition 7 (Robustness). A polynomial template $I(\boldsymbol{a},\boldsymbol{x})$ is said to be robust (w.r.t. the program model Code 1) if there exists a valid parameter assignment $\boldsymbol{a}_0 \in C_{\boldsymbol{a}}$ and a small constant $\epsilon > 0$ such that any $\boldsymbol{a}$ satisfying $\|\boldsymbol{a} - \boldsymbol{a}_0\|_2 < \epsilon$ is still valid.

Proposition 8. *Checking the robustness of a polynomial template is decidable.*

Proof. By Definition 7, when the polynomial template is robust, the valid set R_I contains an interior point. Let $\varphi(\boldsymbol{a})$ denote the conjunction of formulas in Equations (1)–(3). Then the problem reduces to checking whether the following first-order logic formula holds:

$$\exists \epsilon > 0, \exists \boldsymbol{a}_0 \in C_{\boldsymbol{a}}, \forall \boldsymbol{a} \in C_{\boldsymbol{a}}.\ \|\boldsymbol{a} - \boldsymbol{a}_0\|_2 < \epsilon \implies \varphi(\boldsymbol{a}), \tag{34}$$

which is decidable due to Tarski's result [82]. □

Theorem 9 (Semi-Completeness). *If the set $\{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid J(\boldsymbol{a}) = 0\}$ has Lebesgue measure zero and there exists a robust polynomial template $I(\boldsymbol{a}_0,\boldsymbol{x})$ for some $\boldsymbol{a}_0 \in C_{\boldsymbol{a}}$, the Cluster algorithm will yield a non-empty under-approximation $R_{I,D} \subseteq R_I$ for some $D \in \mathbb{N}$ large enough.*

Proof. By definition, a polynomial template $I(\boldsymbol{a}, \boldsymbol{x})$ is robust if R_I has an interior point. Combining Theorems 5 and 7, $R_{I,d}$ has a positive Lebesgue measure when d is large enough, which implies that $R_{I,d}$ is a non-empty under-approximation of the valid set R_I. □

Combining Theorems 7 and 9, we know that the valid set R_I admits an arbitrarily close approximation by solving Equation (30) for sufficiently large d (and d_r), which gives an approximation solution to the strong invariant synthesis problem. Finally, we end this section with an illustrative example.

Example 1. *Consider a discrete-time dynamical system presented in Code 2.*

Code 2: A Discrete-Time Dynamical System

```
// Program variables: (x, y) ∈ ℝ²
// Range: C_{x,y} = {(x, y) | 4 − x² ≥ 0, 4 − y² ≥ 0}
// Precondition: {(x, y) | x² + y² − 1 ≤ 0}
while (x² − 0.81 ≤ 0) {
    // omit case(−1 ≤ 0)
    x ← 0.95(x − 0.1y²);
    y ← 0.95(y + 0.2xy);
}
// Postcondition: {(x, y) | 0.25 − x² − (y − 1.5)² ≤ 0}
```

Suppose that we are searching for ellipsoid-shaped invariants centered at the origin of the form

$$x^2 + \hat{a}y^2 + \hat{b} \in \mathbb{R}[\hat{a}, \hat{b}, x, y], \tag{35}$$

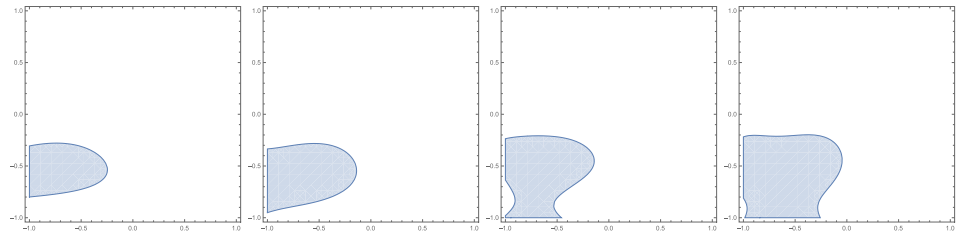
where $\hat{a}$ and $\hat{b}$ are parameters within the range $(\hat{a}, \hat{b}) \in [-10, 10]^2$. By replacing $\hat{a}$ and $\hat{b}$ by $10a$ and $10b$, we denote the polynomial template by

$$I(a, b, x, y) = x^2 + 10ay^2 + 10b, \tag{36}$$

where $(a, b) \in C_{a,b} = \{(a, b) \mid 1 - a^2 \geq 0, 1 - b^2 \geq 0\} = [-1, 1]^2$.

Let $D = 6$ be the upper bound on the degree of $h(a, b)$. By applying Lasserre's technique, we solve the following program to obtain $h(a, b)$, for $1 \leq d \leq D$:

$$\begin{aligned}
\min \quad & \sum_{\beta_1+\beta_2 \leq d} \boldsymbol{\gamma}_{\beta_1,\beta_2} \mathrm{h}_{\beta_1,\beta_2} \\
s.t. \quad & h(a, b) = \sum_{\beta_1+\beta_2 \leq d} \mathrm{h}_{\beta_1,\beta_2} a^{\beta_1} b^{\beta_2} \in \mathbb{R}^d[a, b], \\
& h(a, b) - I(a, b, x, y) = \sigma_{0,0} + \sigma_{0,1} \cdot (1 - x^2 - y^2) \\
& \qquad + \sigma_{0,1}^{a,b} \cdot (1 - a^2) + \sigma_{0,2}^{a,b} \cdot (1 - b^2) + \sigma_{0,1}^{x,y} \cdot (4 - x^2) + \sigma_{0,2}^{x,y} \cdot (4 - y^2), \\
& h(a, b) - I\big(a, b, 0.95(x - 0.1y^2), 0.95(y + 0.2xy)\big) = \sigma_{1,0} - \sigma_{1,1} \cdot I(a, b, x, y) + \sigma_{1,2} \cdot (0.81 - x^2) \\
& \qquad + \sigma_{1,1}^{a,b} \cdot (1 - a^2) + \sigma_{1,2}^{a,b} \cdot (1 - b^2) + \sigma_{1,1}^{x,y} \cdot (4 - x^2) + \sigma_{1,2}^{x,y} \cdot (4 - y^2), \\
& h(a, b) - \big(0.25 - x^2 - (y - 2)^2\big) = \sigma_{2,0} - \sigma_{2,1} \cdot I(a, b, x, y) \\
& \qquad + \sigma_{2,1}^{a,b} \cdot (1 - a^2) + \sigma_{2,2}^{a,b} \cdot (1 - b^2) + \sigma_{2,1}^{x,y} \cdot (4 - x^2) + \sigma_{2,2}^{x,y} \cdot (4 - y^2),
\end{aligned}$$



(a) $d = 3$ (b) $d = 4$ (c) $d = 5$ (d) $d = 6$

x-axis: a from -1.0 to 1.0; y-axis: b from -1.0 to 1.0

Fig. 1. Under-approximations of R_I defined by $R_{I,d} = \{(a, b) \in [-1, 1]^2 \mid h_d(a, b) \leq 0\}$ for $d = 3, 4, 5, 6$.

$$
\begin{aligned}
&h(a,b) + 1 = \sigma_{3,0} + \sigma_{3,1}^{a,b} \cdot (1 - a^2) + \sigma_{3,2}^{a,b} \cdot (1 - b^2) + \sigma_{3,1}^{x,y}(4 - x^2) + \sigma_{3,1}^{x,y} \cdot (4 - y^2),\\
&\sigma_{0,0}, \sigma_{1,0}, \sigma_{2,0}, \sigma_{3,0} \in \Sigma^{2d_r}[a, b, x, y], \sigma_{1,1}, \sigma_{2,1} \in \Sigma^{2d_r-3}[a, b, x, y],\\
&\text{all other SOS polynomials are in } \Sigma^{2d_r-2}[a, b, x, y],
\end{aligned} \tag{37}
$$

where

$$
\gamma_{\beta_1,\beta_2} = \frac{1}{2^2}\int_{-1}^{1}\int_{-1}^{1} a^{\beta_1} b^{\beta_2} \mathrm{d}a\mathrm{d}b = \begin{cases} 0 & \text{if either } \beta_1 \text{ or } \beta_2 \text{ is odd,} \\ \frac{1}{(\beta_1+1)(\beta_2+1)} & \text{otherwise,} \end{cases} \tag{38}
$$

and d_r is the smallest natural number such that $2d_r \geq \max\{d, 3\}$. Here $\deg(I) = 3$ is the largest degree of polynomials in constraints other than $h(a, b)$.

In this example, we use YALMIP [57] to formulate Equation (37) and MOSEK solver [6] to solve the translated SDP. For $d = 1$ and 2, the program is not solvable. For $d = 3, 4, 5,$ and 6, we obtain, rounding to five decimal places,

$$
\begin{aligned}
h_3(a, b) &= 2.69187 + \cdots - 1.50005ab^2 - 2.10735a^3,\\
h_4(a, b) &= 2.34708 + \cdots - 0.00418a^3b - 0.74973a^4,\\
h_5(a, b) &= 2.09144 + \cdots + 2.23803a^4b + 2.57913a^5,\\
h_6(a, b) &= 1.74276 + \cdots - 1.82314a^5b + 1.67067a^6,
\end{aligned}
$$

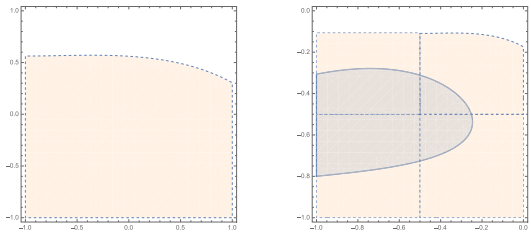
which give under-approximations to the valid set R_I. In Figure 1, we plot the 0-sublevel sets of the above four polynomials. Therefore, any point (a, b) in the light blue region is a valid parameter assignment for the polynomial template $I(a, b, x, y)$.

It's important to note that for small values of d, the inequality $h_d(\boldsymbol{a}) \leq 0$ might not have any solutions, i.e., $R_{I,d} = \emptyset$. When this scenario occurs, we need to increase D to search for polynomials $h_d(\boldsymbol{a})$ of higher degrees. Alternatively, we can also enhance the approximation by shrinking the size of $C_{\boldsymbol{a}}$. For example, suppose we want to obtain a more accurate approximation than the 0-sublevel set of $h_3(a, b)$ over the domain $C'_{\boldsymbol{a}} = [-1, 0] \times [-1, 0]$. We first partition the range $C'_{\boldsymbol{a}}$ into four smaller boxes. Then, we solve Equation (37) with respect to each small box, still with relaxation order $d = 3$. For instance, over the box $[-0.5, 0] \times [-0.5, 0]$, we obtain

$$
h'_3(a, b) = -0.5311 + \cdots + 0.0478ab^2 + 0.54927b^3,
$$

where the range of a, b is scaled to $[-1, 1] \times [-1, 1]$. The 0-sublevel set $h'_3(a, b) \leq 0$ is depicted in Figure 2(a). After scaling back, we can see the combination of under-approximations over these four smaller boxes cover the original approximation at $d = 3$, as shown in Figure 2(b).

Furthermore, using the solutions above, the weak invariant synthesis problem becomes significantly simpler. It can be reduced to solving a single polynomial inequality of the form $h_d(a, b) \leq 0$ for



(a) (b)

(a): $h_3'(a,b) \leq 0$ over $[-0.5,0] \times [-0.5,0]$.
(b): the comparison between $h_3(a,b) \leq 0$ (light blue) and $h_3'(a,b) \leq 0$ (light orange) over $[-1,0] \times [-1,0]$.

Fig. 2. A more accurate under-approximation of R_I by partitioning.

$a, b \in C_{\boldsymbol{a}}$. When the degree and the number of variables are not large, this type of problem can be efficiently addressed using general-purpose numerical optimization solvers or symbolic computation tools. For example, when $d = 3$, a solution $a = -1$, $b = -0.79971$ is returned by FindInstance function in MATHEMATICA, which implies

$$x^2 - 10y^2 - 7.9971 \leq 0$$

is an invariant of Code 2. Even though this example seems simple enough, many existing invariant synthesizing tools do not support this non-linear template or fail to synthesize a suitable invariant [17, 37, 45]. On the other hand, directly applying symbolic solvers such as Z3 or Redlog [25] to constraints Equations (1)–(3) also fails to produce a satisfying assignment of parameters in an hour.

4 Extensions of Cluster Algorithm

In this section, we extend our approach to allow for more complex program structures and invariant templates.

4.1 Nested Loops

In Section 3, we have focused on unnested conditional loops of the form Code 1. In fact, our approach can be extended to nested loops or even control flow graphs without substantial changes.

Code 3: A Simple Nested Loop

```
// Program variables: x ∈ ℝ^n
// Precondition: Pre = {x | q_pre(x) ≤ 0}
// Invariant (outer): I_1(a, x) with a ∈ R^{n'_1}
while (g_1(x) ≤ 0) {
    x ← f_1(x);
    // Invariant (inner): I_2(b, x) with b ∈ R^{n'_2}
    while (g_2(x) ≤ 0) {
        x ← f_2(x);
    }
}
// Postcondition: Post = {x | q_post(x) ≤ 0}
```

To illustrate the main idea, let us consider a simple nested loop of the form Code 3. Synthesizing invariants for Code 3 is more challenging compared to Code 1 because it involves two invariants:

$I_1(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$ for the outer while loop and $I_2(\boldsymbol{b}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{b}, \boldsymbol{x}]$ for the inner while loop. Similar to the unnested cases, we assume that $\boldsymbol{a} \in C_{\boldsymbol{a}}, \boldsymbol{b} \in C_{\boldsymbol{b}}$ for some known hyper-rectangle $C_{\boldsymbol{a}}, C_{\boldsymbol{b}}$. The goal is to find valid assignments of the parameters $\boldsymbol{a}$ and $\boldsymbol{b}$ satisfying the following constraints:

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}.\ \boldsymbol{q}_{pre}(\boldsymbol{x}) \le 0 \implies I_1(\boldsymbol{a}, \boldsymbol{x}) \le 0, \tag{39}$$

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}.\ I_1(\boldsymbol{a}, \boldsymbol{x}) \le 0 \wedge g_1(\boldsymbol{x}) \le 0 \implies I_2(\boldsymbol{b}, \boldsymbol{f}_1(\boldsymbol{x})) \le 0, \tag{40}$$

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}.\ I_2(\boldsymbol{b}, \boldsymbol{x}) \le 0 \wedge g_2(\boldsymbol{x}) \le 0 \implies I_2(\boldsymbol{b}, \boldsymbol{f}_2(\boldsymbol{x})) \le 0, \tag{41}$$

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}.\ I_2(\boldsymbol{b}, \boldsymbol{x}) \le 0 \wedge g_2(\boldsymbol{x}) \ge 0 \implies I_1(\boldsymbol{a}, \boldsymbol{x}) \le 0, \tag{42}$$

$$\forall \boldsymbol{x} \in C_{\boldsymbol{x}}.\ I_1(\boldsymbol{a}, \boldsymbol{x}) \le 0 \wedge g_1(\boldsymbol{x}) \ge 0 \implies \boldsymbol{q}_{post}(\boldsymbol{x}) \le 0. \tag{43}$$

Here Equations (40)–(42) play two roles: For the inner loop, they encode the conditions of I_2 to be an invariant with I_1 serving as both precondition and postcondition; for the outer loop, they collectively encode the inductive condition of I_1.

We can observe that constraints Equations (39)–(43) still adhere to the form of the constraint in Equation (12), albeit parameters $\boldsymbol{a}$ are replaced by $(\boldsymbol{a}, \boldsymbol{b})$. As a result, our approach in Section 3 remains applicable, and the soundness, convergence, and semi-completeness results carry over. Our approach can generally be applied to programs represented by control flow graphs.

4.2 Semialgebraic Templates

In this subsection, we discuss the extensions of our approach to deal with more general polynomial templates, called (basic) semialgebraic templates.

Definition 8 ((Basic) Semialgebraic Template). A semialgebraic template is a finite collection of polynomials $I_{t,r}(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$ defined over $C_{\boldsymbol{a}} \times C_{\boldsymbol{x}}$, where $C_{\boldsymbol{a}} \subseteq \mathbb{R}^{n'}$ is a hyper-rectangle. Given a parameter assignment $\boldsymbol{a}_0 \in \mathbb{R}^{n'}$, the instantiation of the invariant Inv w.r.t. $\boldsymbol{a}_0$ is the set $\{\boldsymbol{x} \mid \bigvee_t \bigwedge_r I_{t,r}(\boldsymbol{a}_0, \boldsymbol{x}) \le 0\}$. If t belongs to a singleton set, then the template is called a basic semialgebraic template, and the instantiation of the invariant is the set $\{\boldsymbol{x} \mid \bigwedge_r I_r(\boldsymbol{a}_0, \boldsymbol{x}) \le 0\}$.

The robustness of (basic) semialgebraic templates is defined similar to that of polynomial templates.

Definition 9 (Robustness). A semialgebraic template $\{I_{t,r}(\boldsymbol{a}, \boldsymbol{x})\}_{t,r}$ is said to be robust (w.r.t. the program model Code 1) if there exists a valid parameter assignment $\boldsymbol{a}_0 \in C_{\boldsymbol{a}}$ and a small constant $\epsilon > 0$ such that any $\boldsymbol{a}$ satisfying $\|\boldsymbol{a} - \boldsymbol{a}_0\| < \epsilon$ is still valid.

In the following, we briefly show that techniques in Section 3 can be directly applied to the cases when templates are basic semialgebraic (instead of only polynomial) without substantial changes. After that, we discuss how to deal with semialgebraic templates in the general form.

When given a basic semialgebraic template or a semialgebraic template, the invariant conditions Equations (1)–(3) should be modified by replacing $I(\boldsymbol{a}, \boldsymbol{x})$ with $\bigwedge_r I_r(\boldsymbol{a}, \boldsymbol{x}) \le 0$ or $\bigvee_t \bigwedge_r I_{t,r}(\boldsymbol{a}_0, \boldsymbol{x}) \le 0$, respectively. Recall that the original constraints Equations (1)–(3) take the form of Equation (12), the new constraints can be viewed as replacing the constraint in Equation (12) by

$$\forall \boldsymbol{x}. \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \ge 0 \implies \bigwedge_r l_r(\boldsymbol{a}, \boldsymbol{x}) \le 0, \tag{44}$$

or

$$\forall \boldsymbol{x}. \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \ge 0 \implies \bigvee_t \bigwedge_r l_{t,r}(\boldsymbol{a}, \boldsymbol{x}) \le 0, \tag{45}$$

where $p_i(\boldsymbol{a}, \boldsymbol{x}), l_r(\boldsymbol{a}, \boldsymbol{x}), l_{t,r}(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$.

As for basic semialgebraic templates, the problem can be reduced to the polynomial case with minor modifications. This is because the constraint Equation (44) can be rewritten as

$$\forall \boldsymbol{x}. \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0 \implies l_r(\boldsymbol{a}, \boldsymbol{x}) \leq 0, \quad \text{for each } r. \tag{46}$$

Consequently, the derived SOS relaxations will be like Equation (23) but will involve more constraints. After that, all other results can be derived similarly.

For general semialgebraic templates, unfortunately, simply rewriting the constraints no longer works due to the existence of disjunction. To address this issue, we resort to the *lifting* technique introduced in [53] to reduce Equation (45) to the form of Equation (12).

First, note that Equation (45) is equivalent to

$$\forall \boldsymbol{x}. \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0 \implies s(\boldsymbol{a}, \boldsymbol{x}) \leq 0, \tag{47}$$

where $s(\boldsymbol{a}, \boldsymbol{x}) = \min_t \max_r l_{t,r}(\boldsymbol{a}, \boldsymbol{x})$.

Let us say a function $f(\boldsymbol{x})$ is a semialgebraic function if its graph $\{(\boldsymbol{x}, f(\boldsymbol{x})) \in \mathbb{R}^{n+1} \mid \boldsymbol{x} \in \mathbb{R}^n\}$ is a semialgebraic set. By Tarski-Seidenberg principle [12, Proposition 2.2.4] and the definition of $s(\boldsymbol{a}, \boldsymbol{x})$, we can prove $s(\boldsymbol{a}, \boldsymbol{x})$ is a semialgebraic function. Since the graph of every semi-algebraic function is the projection of a basic semialgebraic set in the lifted space [52, Lemma 3], we know that the graph

$$\left\{ (\boldsymbol{a}, \boldsymbol{x}, s(\boldsymbol{a}, \boldsymbol{x})) \;\middle|\; \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0 \right\} \subset \mathbb{R}^{n+n'+1} \tag{48}$$

is the projection of some basic semialgebraic set $\hat{\mathcal{K}} \subseteq \mathbb{R}^{n+n'+1+u}$ for some $u \in \mathbb{N}$, i.e.,

$$\left\{ (\boldsymbol{a}, \boldsymbol{x}, v) \;\middle|\; v = s(\boldsymbol{a}, \boldsymbol{x}) \wedge \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0 \right\} = \left\{ (\boldsymbol{a}, \boldsymbol{x}, v) \;\middle|\; \exists \boldsymbol{w} \in \mathbb{R}^u. (\boldsymbol{a}, \boldsymbol{x}, v, \boldsymbol{w}) \in \hat{\mathcal{K}} \right\}, \tag{49}$$

where $v \in \mathbb{R}$ and $\boldsymbol{w} \in \mathbb{R}^u$ are fresh variables

Here, $\hat{\mathcal{K}}$ is called the *lifting* of $\mathcal{K}$ and can be computed following the techniques in [53]. For now, we assume that $\hat{\mathcal{K}}$ is already obtained as

$$\hat{\mathcal{K}} = \left\{ (\boldsymbol{a}, \boldsymbol{x}, v, \boldsymbol{w}) \;\middle|\; \bigwedge_{i=1}^{\hat{m}} \hat{p}_i(\boldsymbol{a}, \boldsymbol{x}, v, \boldsymbol{w}) \geq 0 \right\}, \tag{50}$$

where $\hat{p}_i(\boldsymbol{a}, \boldsymbol{x}, v, \boldsymbol{w}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}, v, \boldsymbol{w}]$. Then, Equation (45) is equivalent to the following constraint in higher dimensions:

$$\forall (\boldsymbol{x}, v, \boldsymbol{w}). \bigwedge_{i=1}^{m} p_i(\boldsymbol{a}, \boldsymbol{x}) \geq 0 \wedge \bigwedge_{i=1}^{\hat{m}} \hat{p}_i(\boldsymbol{a}, \boldsymbol{x}, v, \boldsymbol{w}) \geq 0 \implies v \leq 0, \tag{51}$$

which conforms to the form of Equation (12) when treating $v, \boldsymbol{w}$ equivalently as $\boldsymbol{x}$.

Remark 5. In practice, our algorithm is less efficient for general semialgebraic templates compared to polynomial and basic semialgebraic templates. The main reason lies in the lifting process, which dramatically increases either the degree of defining polynomials or the number of parameters, sometimes even both.

5 Synthesizing Weak Invariants from Masked Templates

Recall that the semi-completeness of our Cluster algorithm hinges on the assumption that the given invariant templates are robust. In Section 5.1, we delve into situations where this assumption is violated, which implies that the Cluster algorithm may fail to produce a non-trivial under-approximation of the valid set. In these scenarios, in Section 5.2, we identify a special subclass of basic semialgebraic templates, called *Masked templates*, consisting of parametric polynomial equalities and some known polynomial inequalities. Regarding these templates, Section 5.3 proposes the *Mask algorithm* to translate the invariant conditions by exploiting the structure of masked templates. The resulting constraints can also be solved by SOS relaxations.

5.1 On Robustness of Semialgebraic Templates

The semi-completeness result (Theorem 9) of the Cluster algorithm relies on the assumption that the given invariant templates are robust. When the valid set R_I is non-empty but the assumption is violated, solving Equation (30) will never yield a non-empty under-approximation $R_{I,d}$ for any $d \in \mathbb{N}$. In this case, the Cluster algorithm is ineffective.

While checking the robustness of a template is decidable (see Proposition 8), the decision procedure involves quantifier elimination, a computationally expensive process. Fortunately, we have the following empirical observation: In most cases, when a basic semialgebraic template does not contain equalities, either the template is robust or the valid set R_I is empty. In this context, having an equality in a basic semialgebraic template means that, for some polynomial $I_i(\boldsymbol{a}, \boldsymbol{x}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{x}]$, both $I_i(\boldsymbol{a}, \boldsymbol{x})$ and $-I_i(\boldsymbol{a}, \boldsymbol{x})$ are contained in the template.

To illustrate the intuition behind this observation, consider a simple loop of the form

$$\text{while } (-1 \le 0) \text{ do } \{\boldsymbol{x} \leftarrow \boldsymbol{x}\}.$$

Suppose that we are given a precondition $Pre = \{\boldsymbol{x} \mid \boldsymbol{x} - \boldsymbol{x_0} \le 0, \boldsymbol{x_0} - \boldsymbol{x} \le 0\}$ for some $\boldsymbol{x}_0 \in \mathbb{R}^n$ and a basic semialgebraic template

$$\{I_1(\boldsymbol{a}, \boldsymbol{x}), -I_1(\boldsymbol{a}, \boldsymbol{x}), \ldots, I_m(\boldsymbol{a}, \boldsymbol{x}), -I_m(\boldsymbol{a}, \boldsymbol{x}), I_{m+1}(\boldsymbol{a}, \boldsymbol{x}), \ldots, I_s(\boldsymbol{a}, \boldsymbol{x})\}, \tag{52}$$

where $\boldsymbol{a}$ is linear in $I_i(\boldsymbol{a}, \boldsymbol{x})$ for $i = 1, \ldots, s$. Since the loop never terminates, the postcondition is irrelevant. In this setting, the invariant conditions Equations (1)–(3) can be reduced to the following single constraint:

$$\begin{aligned} \forall \boldsymbol{x}.\ \boldsymbol{x} - \boldsymbol{x}_0 = 0 \implies & I_1(\boldsymbol{a}, \boldsymbol{x}) = 0 \wedge \cdots \wedge I_m(\boldsymbol{a}, \boldsymbol{x}) = 0 \\ & \wedge I_{m+1}(\boldsymbol{a}, \boldsymbol{x}) \le 0 \wedge \cdots \wedge I_s(\boldsymbol{a}, \boldsymbol{x}) \le 0, \end{aligned} \tag{53}$$

which is equivalent to a linear system of equations in parameters $\boldsymbol{a}$:

$$\begin{aligned} & I_i(\boldsymbol{a}, \boldsymbol{x}_0) = 0 \text{ for } i = 1, \ldots, m \\ & I_j(\boldsymbol{a}, \boldsymbol{x}_0) \le 0 \text{ for } j = m+1, \ldots, s. \end{aligned} \tag{54}$$

Then, the valid set R_I is exactly the solution set to this linear system. Suppose the linear expressions $I_i(\boldsymbol{a}, \boldsymbol{x}_0)$ are linearly independent. Based on the standard knowledge of linear algebra, the dimension of the set of solutions is generally $n' - m$, where n' is the dimension of $\boldsymbol{a}$. However, R_I having an interior point means that R_I is of dimension n', which requires $m = 0$, i.e., there are no equalities in the template.

In practice, when the precondition and the postcondition contain equalities, we tend to need a basic invariant template with equalities. Even though the Cluster algorithm may fail in such situations, we can utilize the structures of the equalities to design more efficient algorithms. This motivates the definition of masked templates and the Mask algorithm below.

5.2 Masked Templates

Before presenting the definitions, we fix some notations. Recall that $\boldsymbol{f}_i : \mathbb{R}^n \to \mathbb{R}^n$ is the assignment function of the ith branch. We slightly abuse the notation $\boldsymbol{f}_{i,\boldsymbol{y}}$ to denote the projection of $\boldsymbol{f}_i$ onto variables $\boldsymbol{y} \subseteq \boldsymbol{x}$. For example, if $\boldsymbol{f}_1(x_1, x_2, x_3) = (x_1, x_2^2, x_3^3)$ and $\boldsymbol{y} = (x_2, x_3)$, then $\boldsymbol{f}_{1,\boldsymbol{y}} = (x_2^2, x_3^3)$.

Definition 10 (Core Variables). Given a loop as in Code 1, if the program variables can be divided into two non-empty parts $\boldsymbol{x} = (\boldsymbol{y}, \boldsymbol{z})$, where $\boldsymbol{y} \in \mathbb{R}^{n_{\boldsymbol{y}}}$ and $\boldsymbol{z} \in \mathbb{R}^{n_{\boldsymbol{z}}}$ with $n_{\boldsymbol{y}} + n_{\boldsymbol{z}} = n$, such that:

(1) for each y in $\boldsymbol{y}$, $\boldsymbol{f}_{i,y} \in \mathbb{R}[\boldsymbol{y}]$;
(2) for each z in $\boldsymbol{z}$, $\boldsymbol{f}_{i,z} \in \mathbb{R}[\boldsymbol{y}, z]$ and is linear in z;
(3) the loop guard $\boldsymbol{g}$ and branch conditionals $\boldsymbol{c}_i$ are independent of $\boldsymbol{z}$, and the postcondition $\boldsymbol{q}_{post}(\boldsymbol{y}, \boldsymbol{z})$ is linear in $\boldsymbol{z}$.

Then we call $\boldsymbol{y}$ core variables and $\boldsymbol{z}$ non-core variables.

Definition 11 (Masked Template). Given core variables $\boldsymbol{y}$ and non-core variables $\boldsymbol{z}$, a masked template is a finite collection of polynomials $I_r(\boldsymbol{a}, \boldsymbol{y}) \in \mathbb{R}[\boldsymbol{a}, \boldsymbol{y}]$ linear in $\boldsymbol{a}$ and polynomials $I_t(\boldsymbol{y}) \in \mathbb{R}[\boldsymbol{y}]$, where $r \in \{1, \ldots, n_{\boldsymbol{z}}\}$ and $t \in T$ for some finite index set T. Given a parameter assignment $\boldsymbol{a}_0 \in \mathbb{R}^{n'}$, the instantiation of the invariant Inv w.r.t. $\boldsymbol{a}_0$ is the set $\{\boldsymbol{x} \mid \bigwedge_{r=1}^{n_{\boldsymbol{z}}} z_r = I_r(\boldsymbol{a}_0, \boldsymbol{y}) \wedge \bigwedge_{t \in T} I_t(\boldsymbol{y}) \leq 0\}$.

In other words, the program variables are partitioned into two non-empty parts, core variables $\boldsymbol{y}$ and non-core variables $\boldsymbol{z}$; and the non-core variables $\boldsymbol{z}$ will only occur in the precondition, the postcondition, and their assignment functions. In a masked template, we wish to express non-core variables $\boldsymbol{z}$ by core variables $\boldsymbol{y}$.

Example 2. *Code 4 is the algorithm for finding the closest integer (variable r) to the square root (of variable y), taken from the benchmark set [70]. Here, we treat integers as a subset of real numbers.*

Code 4: freire1

```
// Program variables: (x, y, r) ∈ ℝ^n
// Precondition: Pre = {(x, y, r) | −y ≤ 0, x = y/2, r = 0}
// Invariant template: {y = poly[(x, r), 2], −x ≤ 0}
while (r − x ≤ 0) { // Real invariant: y = 2x + r^2 − r ∧ x ≥ 0
    x ← x − r;
    y ← y;
    r ← r + 1;
}
// Postcondition: Post = {y − r^2 − r ≤ 0, r^2 − r − y ≤ 0}
```

In this program, we can view (x, r) as core variables and y as the only non-core variable. In the invariant template, the highlighted part $poly[(x, r), 2]$ represents some unknown polynomial in x, r of degree 2. In other words, we are given a masked template of the form $\{y = I(\boldsymbol{a}, x, r), -x \leq 0\}$, where $I(\boldsymbol{a}, x, r) = a_1x^2 + a_2xr + a_3r^2 + a_4x + a_5r + a_6$ with parameters $\boldsymbol{a} = (a_1, \ldots, a_6)$.

In this case, the valid set R_I does not contain an interior point because the following constraint

$$\forall (x, y, r) \in C_{\boldsymbol{x}}. y \geq 0 \wedge x = y/2 \wedge r = 0 \implies y = I(\boldsymbol{a}, x, r) \wedge x \geq 0$$

implies that $a_1 = a_6 = 0$ and $a_4 = 2$.

Our observation suggests that core variables often correspond to local variables (such as x, r), while non-core variables tend to align with the input arguments (such as y). The motivation for the definition of masked templates is that, for most programs, the invariants include two parts: (i) equalities of the form that a part of variables are expressed by the other variables; (ii) inequalities that are derived from conditionals and monotonicity. Besides, the name of masked templates is inspired by the so-called "masked programs" in [30].

5.3 Mask Algorithm

In this part, we show how to transform the invariant conditions for masked templates based on variable substitution. The outstanding property of the resulting constraints is that they can be directly solved by SOS relaxations.

Given a masked template as in Definition 11, we explicitly write down the invariant conditions w.r.t. Equations (1)–(3), with $\boldsymbol{x}$ replaced by $(\boldsymbol{y}, \boldsymbol{z})$

$$\forall(\boldsymbol{y},\boldsymbol{z}) \in C_{\boldsymbol{x}}.\boldsymbol{q}_{pre}(\boldsymbol{y},\boldsymbol{z}) \le \boldsymbol{0} \implies \bigwedge_{r=1}^{n_z} z_r = I_r(\boldsymbol{a},\boldsymbol{y}) \wedge \bigwedge_{t\in T} I_t(\boldsymbol{y}) \le 0 \tag{55}$$

$$\begin{aligned}\forall(\boldsymbol{y},\boldsymbol{z}) \in C_{\boldsymbol{x}}. \bigwedge_{r=1}^{n_z} z_r = I_r(\boldsymbol{a},\boldsymbol{y}) \wedge \bigwedge_{t\in T} I_t(\boldsymbol{y}) \le 0 \wedge \boldsymbol{g}(\boldsymbol{y}) \le \boldsymbol{0} \wedge \boldsymbol{c}_i(\boldsymbol{y}) \le \boldsymbol{0} \\ \implies \bigwedge_{r=1}^{n_z} f_{i,z_r}(\boldsymbol{y},\boldsymbol{z}) = I_r(\boldsymbol{a}, \boldsymbol{f}_{i,\boldsymbol{y}}(\boldsymbol{y})) \wedge \bigwedge_{t\in T} I_t(\boldsymbol{f}_{i,\boldsymbol{y}}(\boldsymbol{y})) \le 0, \quad i = 1,\dots,k,\end{aligned} \tag{56}$$

$$\begin{aligned}\forall(\boldsymbol{y},\boldsymbol{z}) \in C_{\boldsymbol{x}}. \bigwedge_{r=1}^{n_z} z_r = I_r(\boldsymbol{a},\boldsymbol{y}) \wedge \bigwedge_{t\in T} I_t(\boldsymbol{y}) \le 0 \wedge \neg(\boldsymbol{g}(\boldsymbol{y}) \le 0) \\ \implies \boldsymbol{q}_{post}(\boldsymbol{y},\boldsymbol{z}) \le 0,\end{aligned} \tag{57}$$

where we omit $\boldsymbol{z}$ in $\boldsymbol{g}(\boldsymbol{y},\boldsymbol{z})$, $c_i(\boldsymbol{y},\boldsymbol{z})$, and $\boldsymbol{f}_{i,\boldsymbol{y}}(\boldsymbol{y},\boldsymbol{z})$.

As discussed in Remark 3, the SOS relaxations of Equations (56) and (57) cannot be translated into SDPs because parameters $\boldsymbol{a}$ occur in the left-hand-side of the implications. However, after a simple variable substitution procedure, the above constraints can be converted into a desired form. The substitution is based on the following observation in first-order logic: the formula

$$\forall y, z.\ \big(z = f(y) \wedge A(y)\big) \implies B(y,z) \tag{58}$$

is equivalent to

$$\forall y.\ A(y) \implies B(y, f(y)), \tag{59}$$

where f is a function and A, B are formulas.

Exploiting this idea, Equations (56) and (57) can be transformed into

$$\begin{aligned}\forall \boldsymbol{y} \in C_{\boldsymbol{y}}. \bigwedge_{t\in T} I_t(\boldsymbol{y}) \le 0 \wedge \boldsymbol{g}(\boldsymbol{y}) \le \boldsymbol{0} \wedge \boldsymbol{c}_i(\boldsymbol{y}) \le \boldsymbol{0} \\ \implies \bigwedge_{r=1}^{n_{z'}} f_{i,z_r}(\boldsymbol{y},\boldsymbol{z'}) = I_r\big(\boldsymbol{a}, \boldsymbol{f}_{i,\boldsymbol{y}}(\boldsymbol{y})\big) \wedge \bigwedge_{t\in T} I_t\big(\boldsymbol{f}_i(\boldsymbol{y})\big) \le 0, \quad i = 1,\dots,k,\end{aligned} \tag{60}$$

$$\forall \boldsymbol{y} \in C_{\boldsymbol{y}}. \bigwedge_{t\in T} I_t(\boldsymbol{y}) \le 0 \wedge \neg(\boldsymbol{g}(\boldsymbol{y}) \le 0) \implies \boldsymbol{q}_{post}(\boldsymbol{y},\boldsymbol{z'}) \le 0, \tag{61}$$

where $\boldsymbol{z'} = (I_1(\boldsymbol{a},\boldsymbol{y}),\dots,I_{n_z}(\boldsymbol{a},\boldsymbol{y}))$ and C_y is the domain of $\boldsymbol{y}$.

Algorithm 2: The Mask Algorithm

Input : A program $\mathcal{P}$ of the form Code 1, a masked template, and an upper bound $D \in \mathbb{N}$ on the relaxation order.

Output : A valid parameter assignment $\boldsymbol{a}_0$.

```
Construct Equation (62) using P and the masked template;
d_max ← the largest degree of polynomials in Equation (62);
d_r ← ⌊(d_max + 1)/2⌋;
while d_r ≤ D do
    if the d_r-th SOS relaxation of Equation (62) is solvable then
        a_0 ← Solve the d_r-th SOS relaxation of Equation (62);
        return a_0; ▷ a valid parameter assignment
    else
        d_r ← d_r + 1;
    end
end
return ∅; ▷ either D is not large enough or the template has no solution
```

Therefore, finding a valid parameter assignment for Equations (55)–(57) is reduced to solving the following program:

$$\begin{aligned} \text{find} \quad & \boldsymbol{a} \\ s.t. \quad & \text{Constraints Equations (55), (60), and (61).} \end{aligned} \tag{62}$$

According to the definition of masked templates, constraints Equations (60) and (61) contain no non-linear terms in parameters $\boldsymbol{a}$. Therefore, Equation (62) conforms to the form of Equation (7) and can be solved by using the standard SOS relaxation techniques in Section 2.2. As a consequence, the soundness result (Theorem 2) and semi-completeness result (Theorem 3) carry over.

Example 2 *(Continued). After performing a variable substitution, we construct Equation (62) as follows:*

$$\begin{aligned} \textit{find} \quad & \boldsymbol{a} \\ s.t. \quad & \forall (x,y,r) \in C_{x,y,r}. y \geq 0 \wedge x = y/2 \wedge r = 0 \implies y = I(\boldsymbol{a},x,r) \wedge x \geq 0, \\ & \forall (x,r) \in C_{x,r}. x \geq 0 \wedge x \geq r \implies I(\boldsymbol{a},x,r) = I(\boldsymbol{a},x-r,r+1) \wedge x \geq 0, \\ & \forall (x,r) \in C_{x,r}. x \geq 0 \wedge x \leq r \implies r^2 + r \geq I(\boldsymbol{a},x,r) \wedge r^2 - r \leq I(\boldsymbol{a},x,r). \end{aligned}$$

In particular, by solving the 2nd SOS relaxation of the above constraints, we obtain the following numerical result:

$$\begin{aligned} I(\boldsymbol{a}_0,x,r) = {} & 0.0000000020 + 1.9999999314 \cdot x - 0.9999999624 \cdot r + \\ & 0.9999999665 \cdot r^2 - 0.0000000041 \cdot x^2 - 0.0000000098 \cdot x \cdot r, \end{aligned}$$

where we round off the coefficients to 10 decimal places just to demonstrate the numerical errors. In our experiments, we use a technique called rationalization (explained later) to eliminate the numerical errors, obtaining

$$I(\boldsymbol{a}_0,x,r) = 2x + r^2 - r,$$

which can be verified to be an invariant.

6 Experiments

Research Questions (RQs). Our experiments aim to answer the following RQs:

— *RQ1*: How does the *Cluster algorithm* perform on *strong* invariant synthesis problems with a relatively *small* number of parameters?
— *RQ2*: How does the *Mask algorithm* perform on *weak* invariant synthesis problems with a relatively *large* number of parameters?

Note that the number of parameters significantly impacts the computational complexity of our two algorithms. For the Cluster algorithm, Equation (30) involves SOS polynomials with Gram matrices of size $\binom{n+n'+d_r}{d_r}$. To make it tractable, we need to keep the sum $n + n'$ (i.e., the number of $\boldsymbol{x}$ and $\boldsymbol{a}$) small. In contrast, Equation (62) of the Mask algorithm involves SOS polynomials with Gram matrices of size $\binom{n_y+d_r}{d_r}$, suggesting that it could be more scalable for problems with a larger number of parameters.

Implementation. We have developed prototypical implementations of our two SDP-based algorithms in MATLAB (R2020a), interfaced with YALMIP [57] and MOSEK [5] to solve the underlying SOS relaxations. The implementation and benchmarks can be found online via link https://github.com/EcstasyH/invSDP. All experiments were performed on a 2.50 GHz Intel Core i9-12900H laptop running 64-bit Windows 11 with 16 GB of RAM and Nvidia GeForce RTX 3060 GPU.

Comparison. Since the strong invariant synthesis algorithms in [43] and [14] are not implemented, we mainly compare with the weak invariant synthesis tools. For our Cluster algorithm, we add one extra step to synthesize weak invariants: When an under-approximation $R_{I,d} = \{\boldsymbol{a} \in C_{\boldsymbol{a}} \mid h_d(\boldsymbol{a}) \leq 0\}$ is synthesized, we use the symbolic tool MATHEMATICA to solve $h_d(\boldsymbol{a}) \leq 0$ (see the comment on line 13 in Algorithm 1).

We wish to compare our tool with the most relevant tool in [14], but, unfortunately, their implementation is not publicly available. Instead, we primarily compare with POLYSYNTH [30], which is a recent template-based synthesis tool that supports both generations of programs and invariants. When focusing on synthesizing (weak) invariants, POLYSYNTH adopts the same strategy as in [14] to encode the invariant conditions into quadratic constraints. It also employs additional techniques and heuristics for further speedup. To compare with [43], which is based on quantifier elimination, we try solving the constraints Equations (1)–(3) in Z3 [23]. Since most problem instances used in our experiments have basic semialgebraic templates, we do not compare with [1, 55], which only support polynomial templates. Apart from template-based approaches, we also provide a comparison with the state-of-the-art machine learning approach LIPuS [91].

Benchmarks. Our two algorithms are applicable to different situations depending on whether the basic semialgebraic templates contain equalities. Regarding this, we design two sets of benchmarks:

— *Cluster benchmarks* include two groups of problem instances, modified programs and dynamical systems. (1) The modified programs are obtained from the corresponding programs in the masked benchmarks by relaxing the specifications and adjusting the templates. Such modification ensures that the selected templates are robust so that the Cluster algorithm should be able to find a solution, when D is large enough. We also include two benchmarks, freire1-2 and cohencu-2, with templates quadratic in parameters $\boldsymbol{a}$, to validate the statement in Remark 4. The other benchmarks use templates that are linear in parameters $\boldsymbol{a}$ by default. (2) For programs abstracted from dynamical systems in literature, we try to search for ellipsoid-shaped invariants and do not know whether the template is robust. In all these examples, we restrict the number of parameters $\boldsymbol{a}$ to not exceed 3. Since POLYSYNTH lacks support for

Table 1. Experimental Results over Cluster Benchmarks

			Cluster (Algorithm 1)			PolySynth [30]		Z3 [23]	
Benchmark	n_x	n_a	D	Result	Time	Result	Time	Result	Time
freire1-1	3	2	1	✓	0.7 s	✓	3.6 s	✓	**0.1 s**
freire1-2	3	2	2	✓	**4.2 s**	TO	>600 s	TO	>600 s
freire1-3	3	2	1	✓	**0.8 s**	✓	3.1 s	✓	61.0 s
cohencu-1	2	3	2	✓	5.4 s	TO	>600 s	✓	**0.2 s**
cohencu-2	2	3	3	✓	**9.4 s**	TO	>600 s	TO	>600 s
cohencu-3	2	3	3	✓	11.2 s	TO	>600 s	✓	**0.2 s**
Example 1	2	2	3	✓	**3.0 s**	TO	>600 s	TO	>600 s
circuit [4]	2	2	2	✓	**2.8 s**	TO	>600 s	TO	>600 s
unicycle [79]	2	3	≥ 7	TO	>600 s	TO	>600 s	TO	>600 s
overview [21]	2	3	≥ 8	TO	>600 s	TO	>600 s	TO	>600 s

n_x, n_a: The number of variables and parameters, respectively. For the Cluster algorithm: D is the smallest degree upper bound such that a non-empty under-approximation can be found, "time" includes the solving SDP time and the posterior verification time. TO: timeout, 600 s. The boldface marks the winner.

specified domains, we do not supply C_x to it in these benchmarks. For these benchmarks, we primarily compare our tool with complete approaches PolySynth and Z3, as LIPuS does not support floating-point data.

— *Masked benchmarks* include programs without nested loops from [70]. Some benchmarks (like euclidex2 and cohendiv) are adapted so that they do not involve integer operations (such as $\gcd(x, y)$). We also added a manually constructed parameterized benchmark sum-k-power-d, which can be found at the end of this section. In these benchmarks, the program variables and constants are mostly integers, treated as a subset of $\mathbb{R}$. For these programs, as in Example 2, we set the invariant templates to contain all monomials of core variables up to a given degree (determined by the real invariants obtained by experts). Moreover, since program variables are unbounded in these benchmarks, we do not add the bound restriction (i.e., $C_x = \mathbb{R}^n$) and the Mask algorithm remains sound (see Remark 1). For these benchmarks, the comparisons encompass PolySynth, LIPuS, and Z3.

For our two algorithms and Z3, we manually encode the information of the programs and the templates. For PolySynth, the inputs are the programs and the invariant templates. LIPuS does not need templates and the inputs are just the programs. The experimental results are reported in Tables 1 and 2, respectively.

Remark 6. It is worth noting that the results of PolySynth and LIPuS are not directly comparable with the results reported in their original papers, as the specifications and templates can be different. For example, in freire1 example, the template in our experiment is given as in Example 2 with six parameters, while in [30] they assume two parameters in the invariant, i.e., $y = a_1 x + a_2 + r^2 - r$. For many other examples in [30], such as berkeley, they assume the invariants are known and try to synthesize unknown parameters in the loop body.

Numeric vs. Symbolic. One important feature of our approach is that we use a numerical solver. While benefiting from the efficiency of numerical algorithms, the produced results may be unsound due to numerical errors. For example, as shown in Example 2, there is a tiny gap between the

Table 2. Experimental Results over Masked Template Benchmarks

					Mask (Algorithm 2)		PolySynth [30]		LIPuS [91]	
Benchmark	n_y	n_z	n_a	k	Result	Time	Result	Time	Result	Time[a]
berkeley	4	1	5	4	✓	3.8 s	TO	>600 s	TO	>600 s
cohencu	1	3	12	1	✓	1.7 s	TO	>600 s	TO	>600 s
cohendiv	4	1	15	1	✓	0.6 s	✓	6.8 s	(✓)	(142.0 s)
euclidex2	6	2	56	2	✓	4.0 s	TO	>600 s	TO	>600 s
fermat2	4	1	15	2	✓	0.9 s	✓	10.3 s	TO	>600 s
firefly	4	1	5	7	✓	10.9 s	TO	>600 s	TO	>600 s
freire1	2	1	6	1	✓	0.7 s	✓	70.4 s	(✓)	(460.0 s)
freire2	4	-	-	1	NS	-	TO	>600 s	NS	-
illinois	4	1	5	10	✓	17.0 s	TO	>600 s	TO	>600 s
lcm	6	1	28	2	✓	1.3 s	✓	17.1 s	TO	>600 s
mannadiv	4	1	15	3	✓	1.2 s	TO	>600 s	TO	>600 s
mesi	4	1	5	4	✓	4.5 s	TO	>600 s	(✓)	(592.5 s)
moesi	5	1	6	5	✓	7.7 s	TO	>600 s	(✓)	(117.2 s)
petter	1	1	7	1	✓	0.6 s	✓	4.4 s	TO	>600 s
readerswriters	5	1	21	4	✓	3.4 s	TO	>600 s	TO	>600 s
sqrt	4	-	-	1	NS	-	TO	>600 s	(✓)	(421.0 s)
wensley	7	-	-	2	NS	-	TO	>600 s	NS	-
z3sqrt	4	1	15	2	✓	1.1 s	TO	>600 s	NS	-
sum2power10	2	1	66	1	✓	8.6 s	TO	>600 s	TO	>600 s
sum2power15	2	1	136	1	✗	200.3 s	TO	>600 s	TO	>600 s
sum3power6	3	1	84	1	✓	9.9 s	TO	>600 s	TO	>600 s
sum3power8	3	1	102	1	✗	102.4	TO	>600 s	TO	>600 s
sum5power4	5	1	126	1	✓	26.6 s	TO	>600 s	TO	>600 s
sum5power5	5	1	252	1	TO	>600 s	TO	>600 s	TO	>600 s
sum8power3	8	1	165	1	✓	184.6 s	TO	>600 s	TO	>600 s

[a]The results in the last column were provided by the authors of LIPuS using their computational environment in [91].
n_y, n_z, n_a: The number of core variables, non-core variables, and parameters in templates, respectively. k: the number of branches in loop body. For the Mask algorithm, "time" includes the solving SDP time and the posterior verification time. TO: timeout, 600 s. NS: unsupported benchmarks where there are no non-core variables (Mask algorithm) or containing floating-point variables (not allowed in LIPuS). ✗: fail to synthesize an invariant or unsound invariant.

numerical result and the real invariant. Though there exist exact solvers (such as [33, 34]) and arbitrary-precision solvers (such as [41]), our experience shows that these tools do not scale for the SDP problems translated from our benchmarks. In our experiments, we employ the following strategies to deal with numerical errors:

— *Rounding Off*: For the Cluster algorithm, since the invariant template is robust, the obtained valid parameter assignment $\boldsymbol{a}_0$ is usually an interior point of the valid set. This makes the result tolerant to small numerical errors. Even with slight perturbations, $\boldsymbol{a}_0 + \epsilon$ remains valid for small ϵ. While SDP solvers usually offer accuracies around 10^{-8} [75], there is no guarantee of the accuracy of solutions for SOS relaxations. In our experience, rounding off to five decimal places can achieve relatively accurate results.
— *Rationalization*: For the Mask algorithm, the invariant template is not robust as equalities are involved. As a result, solely relying on the rounding off strategy may produce incorrect answers. In addition, the benchmarks contain problem instances that require rational

coefficients like $\frac{5}{12}$, which cannot be expressed by floating point numbers. To this end, we employ the idea from [42]: We first rationalize the numerical result, then verify its correctness. The rationalization is achieved using the built-in function rat in Matlab, which returns the rational fraction approximation of the input to within a specified tolerance (10^{-5}). For example, when a numerical coefficient 0.41667 is obtained, the function rat transforms it into a continued fractional expansion $0 + 1/(2 + 1/(3 + 1/(-2)))$, which can then be simplified to $\frac{5}{12}$.

— *Posterior Verification*: The above two strategies cannot address all numerical problems. To guarantee soundness, when a numerical solution is obtained, we use MATHEMATICA to verify the correctness of the corresponding invariant candidate.

Experimental Results over Cluster Benchmarks. For all benchmarks in the first group, our algorithm successfully synthesized a non-empty under-approximation $\{\boldsymbol{a} \mid h_D(\boldsymbol{a}) \leq 0\}$ with $D \leq 3$, but there were a few cases where directly using Z3 outperformed our approach. Note that both POLYSYNTH and Z3 failed in synthesizing a valid invariant for freire1-2 and cohencu-2, this was possibly attributed to the fact that the templates are quadratic in parameters $\boldsymbol{a}$. Although the problems in the first group are relatively easy, they already pose a challenge for POLYSYNTH and Z3. As for the second group, our algorithm succeeded in the first two problem instances, while the other two tools failed on all problems.

This answers *RQ1*: The Cluster algorithm can efficiently synthesize under-approximations to strong invariant synthesis problems with a relatively small number of parameters. Additionally, its outputs simplify the weak invariant synthesis problem. However, our approach also has limitations. The Cluster algorithm lacks a termination criterion when a non-empty under-approximation cannot be found. This means that the algorithm might run indefinitely in such cases. In addition, as the degree bound D increases, the time required to solve both the SDP and the polynomial inequality $h_D(\boldsymbol{a}) \leq 0$ also grows significantly.

Experimental Results over Masked Benchmarks. In the first group of benchmarks, our algorithm demonstrated its effectiveness by successfully synthesizing valid invariants for 14 out of 18 problem instances. The runtimes of our algorithm for this set of benchmarks were typically under 10 s, demonstrating its practical applicability and efficiency. However, the cohencu benchmark failed due to a small numerical error (approximately 10^{-5}). In the three unsupported benchmarks, we were unable to identify the core variables according to our definition. In comparison, POLYSYNTH solved only five instances, while LIPuS and Z3 (omitted from the table) failed to produce results for all instances in 10 minutes. To fully understand LIPuS's capacity, we collaborated with its creators to evaluate our benchmarks within their environment. The results are displayed in the final column of Table 2, distinguished by parentheses.

The second group of benchmarks demonstrates the scalability of the Mask algorithm, as it can handle problem instances with up to a few hundred parameters within a 10-minute timeout. However, its performance degrades when the degree of the template increases due to numerical issues in the underlying solvers. This is evident in cases like sum2power15 and sum3power8, where the solver incorrectly returns "infeasible" despite the existence of a real invariant. The sumpower benchmarks are tailored to the Mask algorithm. Therefore, a direct comparison with other tools might not be entirely fair.

Overall, the experimental results answer *RQ2*: Compared to existing methods, the Mask algorithm demonstrated superior efficiency and scalability on benchmarks with a relatively large number of parameters.

Sum-k-Power-d Benchmark. Code 5 is manually constructed example with no practical meanings, solely intended to demonstrate how the number of parameters will influence the efficiency of

our algorithm. It is easy to see $n_{\boldsymbol{a}} = \binom{k+d}{d}$. The template $poly[(n_1, \ldots, n_k), d]$ is a polynomial in variables $n_1, \ldots, n_k$ of degree d.

Code 5: sum-k-power-d

```
// Program variables: (n_1, n_2, ..., n_k, s) ∈ ℝ^n
// Precondition: s = (n_1 + n_2 + ··· + n_k)^d
// Invariant template: {s = poly[(n_1, ..., n_k), d]}
while ( true ) { // Real invariant s = (n_1 + n_2 + ··· + n_k)^d
    n_1 ← n_1 + 1;
    ...;
    n_k ← n_k + 1;
    s ← (n_1 + n_2 + ··· + n_k + k)^d − (n_1 + n_2 + ··· + n_k)^d;
}
// Postcondition: true
```

7 Related Work

In this section, we present different methods for invariant synthesis and compare our approaches with the most related works.

Constraint Solving. As constraint-solving techniques have made significant advancements in recent years, constraint-solving-based approaches have become increasingly relevant and promising. Specifically, for synthesizing linear invariants, [17] proposes the first complete approach based on Farkas' lemma, which can be seen as a linear version of Putinar's Positivstellensatz (Theorem 1), and solves the resulting non-linear constraints by quantifier elimination. However, due to the doubly exponential time complexity of quantifier elimination procedures [22], this method is impractical even for programs of moderate size. Therefore, many works consider using heuristics to solve the non-linear constraints for better scalability [56, 78].

The problem of synthesizing polynomial invariants for polynomial programs is more challenging. For both the weak and the strong invariant synthesis problem, [43] introduces the first complete approach based on quantifier elimination. For the strong invariant synthesis problem, when coefficients in templates are rational numbers, [14] shows that the complexity bound can be improved to subexponential in the length of the template. For the weak invariant synthesis problem, subsequent works can be broadly categorized into two classes: One group focuses on efficiently solving the general constraints of invariant conditions [14, 30, 89], while the other group strengthens the invariant conditions to make the constraints easier to solve [1, 18, 55]. A concise overview of these approaches is presented in Table 3. In the following, we compare the technical differences between our work and some related works which also use Theorem 1.

Comparison with [14, 30]: These two papers provide a systematic way to encode the invariant conditions of programs represented by control-flow graphs. As discussed in Remark 3, they also employ Theorem 1 to translate the invariant conditions into constraints involving SOS polynomials (essentially BMIs). To handle these constraints, they further encode them into quadratic programs and rely on general-purpose solvers. In contrast, our algorithms encode constraints into SDPs. This is achieved by utilizing Lasserre's technique [51] (in the Cluster algorithm) and by exploiting specific patterns in templates (in the Mask algorithm).

Comparison with [1, 55]: These two papers deal with the weak invariant synthesis problem by strengthening the invariant conditions in the form of Equation (8), allowing for standard

Table 3. Summary of Constraint-Solving-Based Approaches for Polynomial Invariant Synthesis

Invariant	Algorithm	Loop	Template	Constraint	Guarantee	Size of σ
Strong	Cluster (Section 3)	Nested	Basic semi.	SDP	Convergence*	$\binom{n+n'+d_r}{d_r}$
	[14, 43]	Nested	Basic semi.	FOL	Complete	-
Weak	Mask (Section 5)	Simple	Masked	SDP	Semi-complete	$\binom{n_{\boldsymbol{y}}+d_r}{d_r}$
	[1, 55]	Simple	Poly.	SDP	-	$\binom{n+d_r}{d_r}$
	[14, 30]	Nested	Basic semi.	QP	Semi-complete	$\binom{n+d_r}{d_r}$

Invariant: the strong or weak invariant synthesis problem. *Loop*: the structure of loop models: "nested" means nested loops, and "simple" means non-nested loops. *Template*: the type of invariant templates. *Constraint*: the form of encoded constraints: "QP" means quadratic programming, "FOL" means first-order logic in reals. *Guarantee*: the theoretical guarantee (all these approaches are sound), and the asterisk (*) means the robustness assumption is needed. *Size of* σ: the size of Gram matrices of SOS polynomials (measuring the magnitude of the constraints for methods based on Theorem 1), where d_r is the relaxation order and n, n', and $n_{\boldsymbol{y}}$ are the dimensions of $\boldsymbol{x}$, $\boldsymbol{a}$, and $\boldsymbol{y}$, respectively.

SOS relaxations. However, their techniques are limited to polynomial templates, which means the invariant must be the 0-sublevel set of a single polynomial and do not have completeness guarantees. As a result, these approaches cannot synthesize invariants for programs in our masked template benchmarks.

Craig Interpolation. Craig interpolation is a power tool for local and modular reasoning. In first-order logic, if a formula P implies a formula Q, then there exists a formula I, called a *Craig interpolation* or simply *interpolation*, such that P implies I, I implies Q, and every non-logical symbol in I occurs in both P and Q. In program verification, interpolants can serve as invariants, though they may not always be inductive. Lin et al. [54] introduce an algorithmic framework for generating interpolants and subsequently strengthening them into inductive invariants. Similar approaches have been explored in [21, 28, 29] for synthesizing non-linear invariants for polynomial programs. Craig interpolation has also been employed in model checking techniques for invariant generation, leading to a diverse range of algorithms [10, 16, 36, 47, 60, 61].

Abstract Interpretation. Abstract interpretation is a widely used and classic method for invariant generation [2, 7, 62, 71, 74]. The process involves fixing an abstract domain and iteratively performing forward propagation until a fixed point is reached, which serves as an invariant. The effectiveness and efficiency of abstract interpretation approaches heavily rely on the choice of abstract domains. Different abstract domains may lead to varying levels of precision and scalability in the obtained invariants. In most cases, there is no theoretical guarantee of the accuracy of generated invariants. In other words, it is uncertain whether the obtained invariant is strong enough to accurately represent the desired properties of the system under analysis. The absence of such guarantees necessitates careful consideration of the abstract domains and fine-tuning of the analysis to strike a balance between precision and tractability.

Recurrence Analysis. Recurrence-based methods [26, 40, 45, 48, 72, 74] typically involve these steps: (1) extracting recurrences from loops; (2) computing closed-form solutions for loop variables; and (3) deriving (equality) invariants from the solutions. One major limitation of these methods is that they are restricted to loops with *solvable mappings* [72, 74] (and minor generalizations thereof), a restricted subclass of loops where the underlying recurrences are solvable. Intuitively, solvable mappings generalize affine mappings by allowing certain acyclic nonlinear dependencies between variables. Recent advancements [3, 19, 84] aim to handle the cases where recurrences for loop

variables do not exist or are not solvable, by employing techniques to generate recurrences for expressions over program variables.

Other Methods. Recently, methods based on machine learning [32, 81, 90, 91] and logical inference [24, 46, 67, 80] have shown significant promise. Beyond classic programs, the problem of invariant generation is also being actively explored in the context of hybrid systems [20, 85, 86] and stochastic systems [8, 9, 15], combining techniques from differential equations and probability theory.

We would like to emphasize that our definition of strong invariants differs from the concept of "strongest (affine/polynomial) invariants" commonly used in research on computability results, such as [37, 38, 44, 63, 64]. These studies typically focus on computing sets of affine or polynomial *equalities* that serve as loop invariants. In contrast, our work considers *inequalities* of specific parameterized forms. For a comprehensive overview of these related results, we recommend consulting [64].

8 Conclusions and Future Work

In this article, we present two novel SDP-based approaches to synthesize invariants for polynomial programs, expanding the boundaries of constraint-solving-based invariant synthesis methods. For the strong invariant synthesis problem, the Cluster algorithm employs a technique from robust optimization [51] to under-approximate the valid set. For the weak invariant synthesis problem, the Mask algorithm relies on identifying special structures in program invariants. Both our algorithms are sound and semi-complete.

Currently, the Cluster algorithm becomes impractical when the template includes an excessive number of parameters. This limitation arises because the size of SOS polynomials in the relaxations depends on the total number of program variables and parameters. To address this problem, we consider exploring the internal structure of the constraints to improve the algorithm. Moreover, we plan to extend the techniques presented in this article to invariant synthesis for hybrid systems and probabilistic programs.

Acknowledgments

We are grateful to the anonymous reviewers for their insightful feedback and constructive criticism, which significantly improved the quality of this article. We also thank Dr. Shiwen Yu for his assistance in producing the benchmark results of LIPuS.

References

[1] Assalé Adjé, Pierre-Loïc Garoche, and Victor Magron. 2015. Property-based polynomial invariant generation using sums-of-squares optimization. In *22nd International Symposium on Static Analysis*, Lecture Notes in Computer Science, Vol. 9291. Springer, 235–251. DOI: https://doi.org/10.1007/978-3-662-48288-9_14

[2] Assalé Adjé, Stéphane Gaubert, and Eric Goubault. 2012. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. *Logical Methods in Computer Science* 8, 1 (2012), 1–32. DOI: https://doi.org/10.2168/LMCS-8(1:1)2012

[3] Daneshvar Amrollahi, Ezio Bartocci, George Kenison, Laura Kovács, Marcel Moosbrugger, and Miroslav Stankovic. 2022. Solving invariant generation for unsolvable loops. In *29th International Symposium on Static Analysis (SAS '22)*, Lecture Notes in Computer Science, Vol. 13790. Springer, 19–43. DOI: https://doi.org/10.1007/978-3-031-22308-2_3

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

[5] Erling D. Andersen, Cornelis Roos, and Tamás Terlaky. 2003. On implementing a primal-dual interior-point method for conic quadratic optimization. *Mathematical Programming* 95, 2 (2003), 249–277.

[6] MOSEK ApS. 2019. *The MOSEK Optimization Toolbox for MATLAB Manual. Version 9.0.* Retrieved from http://docs.mosek.com/9.0/toolbox/index.html

[7] Roberto Bagnara, Enric Rodríguez-Carbonell, and Enea Zaffanella. 2005. Generation of basic semi-algebraic invariants using convex polyhedra. In *12th International Symposium on Static Analysis* (Lecture Notes in Computer Science, Vol. 3672). Springer, 19–34. DOI: https://doi.org/10.1007/11547662_4

[8] Jialu Bao, Nitesh Trivedi, Drashti Pathak, Justin Hsu, and Subhajit Roy. 2022. Data-driven invariant learning for probabilistic programs. In *34th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 13371. Springer, Haifa, Israel, 33–54. DOI: https://doi.org/10.1007/978-3-031-13185-1_3

[9] Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. 2023. Probabilistic program verification via inductive synthesis of inductive invariants. In *29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, Vol. 13994. Springer, Paris, France, 410–429. DOI: https://doi.org/10.1007/978-3-031-30820-8_25

[10] Martin Blicha, Grigory Fedyukovich, Antti E. J. Hyvärinen, and Natasha Sharygina. 2022. Transition power abstractions for deep counterexample detection. In *28th International Conference Tools and Algorithms for the Construction and Analysis of Systems (TACAS '22)*, Lecture Notes in Computer Science, Vol. 13243. Springer, 524–542. DOI: https://doi.org/10.1007/978-3-030-99524-9_29

[11] Vincent Blondel and John N. Tsitsiklis. 1995. NP-hardness of some linear control design problems. In *34th IEEE Conference on Decision and Control (CDC)*, Vol. 3, 2910–2915. DOI: https://doi.org/10.1109/CDC.1995.478584

[12] Jacek Bochnak, Michel Coste, and Marie-Françoise Roy. 1998. *Real Algebraic Geometry*, Vol. 36. Springer Science & Business Media.

[13] Stephen Boyd, Stephen P. Boyd, and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

[14] Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Ehsan Kafshdar Goharshady. 2020. Polynomial invariant generation for non-deterministic recursive programs. In *41st ACM SIGPLAN International Conference on Programming Language Design and Implementation*. ACM, 672–687. DOI: https://doi.org/10.1145/3385412.3385969

[15] Yu-Fang Chen, Chih-Duo Hong, Bow-Yaw Wang, and Lijun Zhang. 2015. Counterexample-guided polynomial loop invariant generation by Lagrange interpolation. In *27th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 9206. Springer, San Francisco, CA, 658–674. DOI: https://doi.org/10.1007/978-3-319-21690-4_44

[16] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. 2016. Infinite-state invariant checking with IC3 and predicate abstraction. *Formal Methods in System Design* 49, 3 (2016), 190–218. DOI: https://doi.org/10.1007/s10703-016-0257-4

[17] Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. 2003. Linear invariant generation using non-linear constraint solving. In *15th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 2725. Springer, 420–432. DOI: https://doi.org/10.1007/978-3-540-45069-6_39

[18] Patrick Cousot. 2005. Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming. In *6th International Conference on Verification, Model Checking, and Abstract Interpretation* , Lecture Notes in Computer Science, Vol. 3385. Springer, 1–24. DOI: https://doi.org/10.1007/978-3-540-30579-8_1

[19] John Cyphert and Zachary Kincaid. 2024. Solvable polynomial ideals: The ideal reflection for program analysis. *Proceedings of the ACM on Programming Languages* 8, POPL (2024), 724–752. DOI: https://doi.org/10.1145/3632867

[20] Liyun Dai, Ting Gan, Bican Xia, and Naijun Zhan. 2017. Barrier certificates revisited. *Journal of Symbolic Computation* 80 (2017), 62–86.

[21] Liyun Dai, Bican Xia, and Naijun Zhan. 2013. Generating non-linear interpolants by semidefinite programming. In *25th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 8044. Springer, 364–380. DOI: https://doi.org/10.1007/978-3-642-39799-8_25

[22] James H. Davenport and Joos Heintz. 1988. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation* 5, 1–2 (1988), 29–35.

[23] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: An efficient SMT solver. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, Vol. 4963. Springer, 337–340. DOI: https://doi.org/10.1007/978-3-540-78800-3_24

[24] Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. 2013. Inductive invariant generation via abductive inference. In *2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*. ACM, 443–456. DOI: https://doi.org/10.1145/2509136.2509511

[25] Andreas Dolzmann and Thomas Sturm. 1996. *Redlog User Manual*.

[26] Azadeh Farzan and Zachary Kincaid. 2015. Compositional recurrence analysis. In *Formal Methods in Computer-Aided Design*. IEEE, 57–64. DOI: https://doi.org/10.1109/FMCAD.2015.7542253

[27] Robert W. Floyd. 1967. Assigning meanings to programs. *Mathematical Aspects of Computer Science* 19, 19–32 (1967), 1.

[28] Ting Gan, Liyun Dai, Bican Xia, Naijun Zhan, Deepak Kapur, and Mingshuai Chen. 2016. Interpolant synthesis for quadratic polynomial inequalities and combination with EUF. In *8th International Joint Conference on Automated Reasoning*, Lecture Notes in Computer Science, Vol. 9706. Springer, 195–212. DOI: https://doi.org/10.1007/978-3-319-40229-1_14

[29] Ting Gan, Bican Xia, Bai Xue, Naijun Zhan, and Liyun Dai. 2020. Nonlinear Craig interpolant generation. In *32nd International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 12224. Springer, 415–438. DOI: https://doi.org/10.1007/978-3-030-53288-8_20

[30] Amir Kafshdar Goharshady, S. Hitarth, Fatemeh Mohammadi, and Harshit J. Motwani. 2023. Algebro-geometric algorithms for template-based synthesis of polynomial programs. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 727–756. DOI: https://doi.org/10.1145/3586052

[31] Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations*, 3rd ed. Johns Hopkins University Press.

[32] Jingxuan He, Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2020. Learning fast and precise numerical analysis. In *41st ACM SIGPLAN International Conference on Programming Language Design and Implementation*. ACM, 1112–1127. DOI: https://doi.org/10.1145/3385412.3386016

[33] Didier Henrion, Simone Naldi, and Mohab Safey El Din. 2019. SPECTRA - A Maple library for solving linear matrix inequalities in exact arithmetic. *Optimization Methods and Software* 34, 1 (2019), 62–78. DOI: https://doi.org/10.1080/10556788.2017.1341505

[34] Didier Henrion, Simone Naldi, and Mohab Safey El Din. 2021. Exact algorithms for semidefinite programs with degenerate feasible set. *Journal of Symbolic Computation* 104 (2021), 942–959. DOI: https://doi.org/10.1016/j.jsc.2020.11.001

[35] Charles Antony Richard Hoare. 1969. An axiomatic basis for computer programming. *Communications of the ACM* 12, 10 (1969), 576–580.

[36] Hossein Hojjat and Philipp Rümmer. 2018. The ELDARICA horn solver. In *Formal Methods in Computer Aided Design (FMCAD '18)*. IEEE, 1–7. DOI: https://doi.org/10.23919/FMCAD.2018.8603013

[37] Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. 2018. Polynomial invariants for affine programs. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 530–539. DOI: https://doi.org/10.1145/3209108.3209142

[38] Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. 2023. On strongest algebraic program invariants. *Journal of the ACM* 70, 5 (2023), 29:1–29:22. DOI: https://doi.org/10.1145/3614319

[39] Lei Huang, Jiawang Nie, and Ya-Xiang Yuan. 2023. Homogenization for polynomial optimization with unbounded sets. *Mathematical Programming* 200, 1 (2023), 105–145. DOI: https://doi.org/10.1007/s10107-022-01878-5

[40] Andreas Humenberger, Maximilian Jaroschek, and Laura Kovács. 2018. Invariant generation for multi-path loops with polynomial assignments. In *19th International Conference on Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science, Vol. 10747. Springer, 226–246. DOI: https://doi.org/10.1007/978-3-319-73721-8_11

[41] Mioara Joldes, Jean-Michel Muller, and Valentina Popescu. 2017. Implementation and performance evaluation of an extended precision floating-point arithmetic library for high-accuracy semidefinite programming. In *24th IEEE Symposium on Computer Arithmetic*. IEEE Computer Society, 27–34. DOI: https://doi.org/10.1109/ARITH.2017.18

[42] Erich L. Kaltofen, Bin Li, Zhengfeng Yang, and Lihong Zhi. 2012. Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients. *Journal of Symbolic Computation* 47, 1 (2012), 1–15. DOI: https://doi.org/10.1016/J.JSC.2011.08.002

[43] Deepak Kapur. 2005. Automatically generating loop invariants using quantifier elimination. In *Deduction and Applications (Dagstuhl Seminar Proceedings)*, Vol. 05431. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. Retrieved from http://drops.dagstuhl.de/opus/volltexte/2006/511

[44] Michael Karr. 1976. Affine relationships among variables of a program. *Acta Informatica* 6, 2 (1976), 133–151.

[45] Zachary Kincaid, John Cyphert, Jason Breck, and Thomas W. Reps. 2018. Non-linear reasoning for invariant synthesis. *Proceedings of the ACM on Programming Languages* 2, POPL (2018), 54:1–54:33. DOI: https://doi.org/10.1145/3158142

[46] Jason R. Koenig, Oded Padon, Sharon Shoham, and Alex Aiken. 2022. Inferring invariants with quantifier alternations: Taming the search space explosion. In *28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, Vol. 13243. Springer, 338–356. DOI: https://doi.org/10.1007/978-3-030-99524-9_18

[47] Anvesh Komuravelli, Arie Gurfinkel, and Sagar Chaki. 2014. SMT-based model checking for recursive programs. In *26th International Conference on Computer Aided Verification (CAV '14)*, Lecture Notes in Computer Science, Vol. 8559. Springer, 17–34. DOI: https://doi.org/10.1007/978-3-319-08867-9_2

[48] Laura Kovács. 2008. Reasoning algebraically about P-solvable loops. In *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08)*, Lecture Notes in Computer Science, Vol. 4963. Springer, 249–264. DOI: https://doi.org/10.1007/978-3-540-78800-3_18

[49] Jean Bernard Lasserre. 2000. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization* 11 (2000), 796–817. Retrieved from https://api.semanticscholar.org/CorpusID:16740871

[50] Jean Bernard Lasserre. 2009. *Moments, Positive Polynomials and Their Applications*, Vol. 1. World Scientific.

[51] Jean B. Lasserre. 2015. Tractable approximations of sets defined with quantifiers. *Mathematical Programming* 151, 2 (2015), 507–527.

[52] Jean B. Lasserre and Mihai Putinar. 2010. Positivity and optimization for semi-algebraic functions. *SIAM Journal on Optimization* 20, 6 (2010), 3364–3383. DOI: https://doi.org/10.1137/090775221
[53] Jean B. Lasserre and Mihai Putinar. 2012. Positivity and optimization: Beyond polynomials. In *Handbook on Semidefinite, Conic and Polynomial Optimization*. Miguel F. Anjos and Jean B. Lasserre (Eds.), Springer, 407–434.
[54] Shang-Wei Lin, Jun Sun, Hao Xiao, Yang Liu, David Sanán, and Henri Hansen. 2017. FiB: Squeezing loop invariants by interpolation between forward/backward predicate transformers. In *32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 793–803. DOI: https://doi.org/10.1109/ASE.2017.8115690
[55] Wang Lin, Min Wu, Zhengfeng Yang, and Zhenbing Zeng. 2014. Proving total correctness and generating preconditions for loop programs via symbolic-numeric computation methods. *Frontiers of Computer Science* 8, 2 (2014), 192–202. DOI: https://doi.org/10.1007/s11704-014-3150-6
[56] Hongming Liu, Hongfei Fu, Zhiyong Yu, Jiaxin Song, and Guoqiang Li. 2022. Scalable linear invariant generation with Farkas' lemma. *Proceedings of the ACM on Programming Languages* 6, OOPSLA2 (2022), 204–232. DOI: https://doi.org/10.1145/3563295
[57] J. Löfberg. 2004. YALMIP: A toolbox for modeling and optimization in MATLAB. In *2004 IEEE International Symposium on Computer Aided Control Systems Design (CACSD '04)*, 284–289.
[58] Ngoc Hoang Anh Mai, Jean-Bernard Lasserre, and Victor Magron. 2022. Positivity certificates and polynomial optimization on non-compact semialgebraic sets. *Mathematical Programming* 194, 1 (2022), 443–485. DOI: https://doi.org/10.1007/s10107-021-01634-1
[59] Murray Marshall. 2008. *Positive Polynomials and Sums of Squares*. Number 146. American Mathematical Soc.
[60] Kenneth L. McMillan. 2003. Interpolation and SAT-based model checking. In *15th International Conference on Computer Aided Verification (CAV 2003)*, Lecture Notes in Computer Science, Vol. 2725. Springer, 1–13. DOI: https://doi.org/10.1007/978-3-540-45069-6_1
[61] Kenneth L. McMillan. 2006. Lazy abstraction with interpolants. In *18th International Conference on Computer Aided Verification (CAV '06)*, Lecture Notes in Computer Science, Vol. 4144. Springer, 123–136. DOI: https://doi.org/10.1007/11817963_14
[62] Markus Müller-Olm and Helmut Seidl. 2004. Computing polynomial program invariants. *Information Processing Letters* 91, 5 (2004), 233–244. DOI: https://doi.org/10.1016/j.ipl.2004.05.004
[63] Markus Müller-Olm and Helmut Seidl. 2004. A note on Karr's algorithm. In *31st International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 3142. Springer, 1016–1028. DOI: https://doi.org/10.1007/978-3-540-27836-8_85
[64] Julian Müllner, Marcel Moosbrugger, and Laura Kovács. 2024. Strong invariants are hard: On the hardness of strongest polynomial invariants for (probabilistic) programs. *Proceedings of the ACM on Programming Languages* 8, POPL (2024), 882–910. DOI: https://doi.org/10.1145/3632872
[65] Peter Naur. 1966. Proof of algorithms by general snapshots. *BIT Numerical Mathematics* 6, 4 (1966), 310–316.
[66] Jiawang Nie. 2014. Optimality conditions and finite convergence of Lasserre's hierarchy. *Mathematical Programming* 146 (2014), 97–121.
[67] Oded Padon, James R. Wilcox, Jason R. Koenig, Kenneth L. McMillan, and Alex Aiken. 2022. Induction duality: Primal-dual search for invariants. *Proceedings of the ACM on Programming Languages* 6, POPL (2022), 1–29. DOI: https://doi.org/10.1145/3498712
[68] Pablo A. Parrilo. 2000. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. California Institute of Technology.
[69] Mihai Putinar. 1993. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal* 42, 3 (1993), 969–984.
[70] Enric Rodríguez-Carbonell. 2016. Some Programs That Need Polynomial Invariants in Order to Be Verified. Retrieved from https://www.cs.upc.edu/erodri/webpage/polynomial_invariants/list.html
[71] Enric Rodríguez-Carbonell and Deepak Kapur. 2004. An abstract interpretation approach for automatic generation of polynomial invariants. In *11th International Symposium on Static Analysis*, Lecture Notes in Computer Science, Vol. 3148. Springer, 280–295. DOI: https://doi.org/10.1007/978-3-540-27864-1_21
[72] Enric Rodríguez-Carbonell and Deepak Kapur. 2004. Automatic generation of polynomial loop invariants: Algebraic foundations. In *2004 International Symposium on Symbolic and Algebraic Computation*, 266–273.
[73] Enric Rodríguez-Carbonell and Deepak Kapur. 2007. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. *Science of Computer Programming* 64, 1 (2007), 54–75.
[74] Enric Rodríguez-Carbonell and Deepak Kapur. 2007. Generating all polynomial invariants in simple loops. *Journal of Symbolic Computation* 42, 4 (2007), 443–476.
[75] Pierre Roux, Yuen-Lam Voronin, and Sriram Sankaranarayanan. 2018. Validating numerical semidefinite programming solvers for polynomial invariants. *Formal Methods in System Design* 53, 2 (2018), 286–312.

[76] Sartaj Sahni. 1974. Computationally related problems. *SIAM Journal on Computing* 3, 4 (1974), 262–279. DOI: https://doi.org/10.1137/0203021

[77] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. 2004. Constructing invariants for hybrid systems. In *7th International Workshop on Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science, Vol. 2993. Springer, 539–554. DOI: https://doi.org/10.1007/978-3-540-24743-2_36

[78] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2004. Constraint-based linear-relations analysis. In *11th International Symposium on Static Analysis*, Lecture Notes in Computer Science, Vol. 3148. Springer, 53–68. DOI: https://doi.org/10.1007/978-3-540-27864-1_7

[79] Mohamed Amin Ben Sassi and Antoine Girard. 2012. Controller synthesis for robust invariance of polynomial dynamical systems using linear programming. *Systems & Control Letters* 61, 4 (2012), 506–512.

[80] Rahul Sharma and Alex Aiken. 2014. From invariant checking to invariant inference using randomized search. In *26th International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 8559. Springer, 88–105. DOI: https://doi.org/10.1007/978-3-319-08867-9_6

[81] Xujie Si, Aaditya Naik, Hanjun Dai, Mayur Naik, and Le Song. 2020. Code2Inv: A deep learning framework for program verification. In *32nd International Conference on Computer Aided Verification*, Lecture Notes in Computer Science, Vol. 12225. Springer, 151–164. DOI: https://doi.org/10.1007/978-3-030-53291-8_9

[82] Alfred Tarski. 1951. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley.

[83] Onur Toker and Hitay Özbay. 1995. On the NP-hardness of solving bilinear matrix inequalities and simultaneous stabilization with static output feedback. In *the American Control Conference (ACC)*, Vol. 4, 2525–2526. DOI: https://doi.org/10.1109/ACC.1995.532300

[84] Chenglin Wang and Fangzhen Lin. 2024. On polynomial expressions with C-finite recurrences in loops with nested nondeterministic branches. In *36th International Conference on Computer Aided Verification (CAV '24)*, Lecture Notes in Computer Science, Vol. 14681. Springer, 409–430. DOI: https://doi.org/10.1007/978-3-031-65627-9_20

[85] Qiuye Wang, Mingshuai Chen, Bai Xue, Naijun Zhan, and Joost-Pieter Katoen. 2021. Synthesizing invariant barrier certificates via difference-of-convex programming. In *33rd International Conference on Computer Aided Verification (CAV '21)*, Lecture Notes in Computer Science, Vol. 12759. Springer, 443–466.

[86] Qiuye Wang, Mingshuai Chen, Bai Xue, Naijun Zhan, and Joost-Pieter Katoen. 2022. Encoding inductive invariants as barrier certificates: Synthesis via difference-of-convex programming. *Information and Computation* 289, Part (2022), 104965. DOI: https://doi.org/10.1016/j.ic.2022.104965

[87] Hao Wu, Shenghua Feng, Ting Gan, Jie Wang, Bican Xia, and Naijun Zhan. 2024. On completeness of SDP-based barrier certificate synthesis over unbounded domains. In *26th International Symposium on Formal Methods (FM (2))*, Lecture Notes in Computer Science, Vol. 14934. Springer, 248–266. DOI: https://doi.org/10.1007/978-3-031-71177-0_16

[88] Hao Wu, Jie Wang, Bican Xia, Xiakun Li, Naijun Zhan, and Ting Gan. 2024. Nonlinear Craig interpolant generation over unbounded domains by separating semialgebraic sets. In *26th International Symposium on Formal Methods (FM (1))*, Lecture Notes in Computer Science, Vol. 14933. Springer, 92–110. DOI: https://doi.org/10.1007/978-3-031-71162-6_5

[89] Lu Yang, Chaochen Zhou, Naijun Zhan, and Bican Xia. 2010. Recent advances in program verification through computer algebra. *Frontiers of Computer Science in China* 4, 1 (2010), 1–16. DOI: https://doi.org/10.1007/s11704-009-0074-7

[90] Jianan Yao, Gabriel Ryan, Justin Wong, Suman Jana, and Ronghui Gu. 2020. Learning nonlinear loop invariants with gated continuous logic networks. In *41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 106–120.

[91] Shiwen Yu, Ting Wang, and Ji Wang. 2023. Loop invariant inference through SMT solving enhanced reinforcement learning. In *32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 175–187. DOI: https://doi.org/10.1145/3597926.3598047

Received 23 August 2023; revised 7 September 2024; accepted 30 November 2024