Finding Polynomial Loop Invariants for Probabilistic Programs

Yijun Feng[1(✉)], Lijun Zhang[2,3], David N. Jansen[2(✉)], Naijun Zhan[2], and Bican Xia[1]

[1] LMAM and School of Mathematical Sciences, Peking University, Beijing, China
fyj_jyf9225@pku.edu.cn, xbc@math.pku.edu.cn

[2] State Key Laboratory of Computer Science, Institute of Software, CAS, Beijing, China
{zhanglj,dnjansen,znj}@ios.ac.cn

[3] University of Chinese Academy of Sciences, Beijing, China

Abstract. Quantitative loop invariants are an essential element in the verification of probabilistic programs. Recently, multivariate Lagrange interpolation has been applied to synthesizing polynomial invariants. In this paper, we propose an alternative approach. First, we fix a polynomial template as a candidate of a loop invariant. Using Stengle's Positivstellensatz and a transformation to a sum-of-squares problem, we find sufficient conditions on the coefficients. Then, we solve a semidefinite programming feasibility problem to synthesize the loop invariants. If the semidefinite program is unfeasible, we backtrack after increasing the degree of the template. Our approach is semi-complete in the sense that it will always lead us to a feasible solution if one exists and numerical errors are small. Experimental results show the efficiency of our approach.

1 Introduction

Probabilistic programs extend standard programs with probabilistic choices and are widely used in protocols, randomized algorithms, stochastic games, etc. In such situations, the program may report incorrect results with a certain probability, rendering classical program specification methods [10,18] inadequate. As a result, formal reasoning about the correctness needs to be based on quantitative specifications. Typically, a probabilistic program consists of steps that choose probabilistically between several states, and the specification of a probabilistic program contains constraints on the probability distribution of final states, e.g. through the expected value of a random variable. Therefore the expected value is often the object of correctness verification [14,21,23].

To reason about correctness for probabilistic programs, quantitative annotations are needed. Most importantly, correctness of while loops can be proved by inferring special bounds on expectations, usually called *quantitative loop invariants* [23]. As in the classical setting, finding such invariants is the bottleneck of proving program correctness. For some restricted classes, such as linear loop invariants, some techniques have been established [4,21,24]. To use them to synthesize polynomial loop invariants, so-called linearization can be used [1],

D. D'Souza and K. Narayan Kumar (Eds.): ATVA 2017, LNCS 10482, pp. 400–416, 2017.
DOI: 10.1007/978-3-319-68167-2_26

a technique widely applied in linear algebra. It views higher-degree monomials as new variables, establishes their relation with existing variables, and then exploits linear loop invariant generation techniques. However, the number of monomials is exponential in the degree. Rodríguez-Carbonell and Kapur [28] introduce solvable mappings, which are a generalization of affine mappings, to avoid non-polynomial effects generated by polynomial programs. Recently, Chen et al. [7] applied multivariate Lagrange interpolation to synthesize polynomial loop invariants directly.

Another important problem for probabilistic programs is the *almost-sure termination problem,* answering whether the program terminates almost surely. Fioriti and Hermanns [13] argued that Lyapunov ranking functions, used in non-probabilistic termination analysis, cannot be extended to probabilistic programs. Instead, they extended ranking supermartingales [3] to the bounded probabilistic case and provided a compositional and sound proof method for the almost-sure termination problem. Kaminski and Katoen [20] investigated the computational complexity of computing expected outcomes (including lower bounds, upper bounds and exact expected outcomes) and of deciding almost-sure termination of probabilistic programs. Further, Chatterjee et al. [6] investigated termination problems for affine probabilistic programs. Recently, they also presented a method [5] to efficiently synthesize ranking supermartingales by Putinar's Positivstellensatz [27] and used it to prove the termination of probabilistic programs. Their method is sound and semi-complete over a large class of programs.

In this paper, we develop a technique exploiting *semidefinite programming* through another Positivstellensatz to synthesize the quantitative loop invariants. Positivstellensätze are essential theorems in real algebra to describe the structure of polynomials that are positive (or non-negative) on a *semialgebraic set.* While our approach shares some similarities with the one in [5], the difference to the termination problem requires a variation of the theorem. In detail, Putinar's Positivstellensatz deals with the situation when the polynomial is strictly positive on a quadratic module, which is not enough for quantitative loop invariants. In the program correctness problem, equality constraints are taken into consideration as well as inequalities. Therefore in our method, Stengle's Positivstellensatz [29] dealing with general real semi-algebraic sets is being used.

As previous results [7,15,21], our approach is *constraint-based* [8]. We fix a polynomial template for the invariants with a fixed degree and generate constraints from the program. The constraints can be transformed into an emptiness problem of a semialgebraic set. By Stengle's Positivstellensatz [29], it suffices to solve a semidefinite programming feasibility problem, for which efficient solvers exist. From a feasible solution (which need not be tight) we can then obtain the corresponding template coefficients. If the solver does not provide a feasible solution or a verification shows that the coefficients are not correct, we refine the analysis by adding constraints to block the undesired solutions or by increasing the template degree, which will always lead us to a feasible solution if one exists.

The method is applied to several case studies taken from [7]. Our technique usually solves the problem within one second, which is about one tenth of the time taken by the tool of [7]. Our tool supports real variables rather than discrete ones and can generate polynomial invariants. We illustrate these features by analyzing a non-linear perceptron program and a model for airplane delay with continuous distributions. Moreover, we conduct a sequence of trials on parameterized probabilistic programs to show that the main influence factor on the running time of our method is the degree of the invariant template. We compare our results on these examples with the Lagrange Interpolation Prototype (LIP) in [7], Prinsys [15] and the tool for super-martingales (TM) [3].

2 Preliminaries

In this section we introduce some notations. We use $\mathbf{X}_n$ to denote an n-tuple of variables $(X_1, \ldots, X_n)$. For a vector $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{N}^n$, $\mathbf{X}_n^\alpha$ denotes the monomial $X_1^{\alpha_1} \cdots X_n^{\alpha_n}$, and $d = \sum_i \alpha_i$ is its degree.

Definition 1. *A polynomial f in variables $X_1, \ldots, X_n$ is a finite linear combination of monomials: $f = \sum_\alpha c_\alpha \mathbf{X}_n^\alpha$ where finitely many $c_\alpha \in \mathbb{R}$ are non-zero.*

The degree of a polynomial is the highest degree of its component monomials. Extending the notation, for a sequence of polynomials $\mathcal{F} = (f_1, \ldots, f_s)$ and a vector $\alpha = (\alpha_1, \ldots, \alpha_s) \in \mathbb{R}^s$, we let $\mathcal{F}^\alpha$ denote $\prod_{i=1}^s f_i^{\alpha_i}$. The polynomial ring with n variables is denoted with $\mathbb{R}[\mathbf{X}_n]$, and the set of polynomials of degree at most d is denoted with $\mathbb{R}^{\leq d}[\mathbf{X}_n]$. For $f \in \mathbb{R}[\mathbf{X}_n]$ and $\mathbf{z}_n = (z_1, \ldots, z_n) \in \mathbb{R}^n$, $f(\mathbf{z}_n) \in \mathbb{R}$ is the value of f at $\mathbf{z}_n$.

A *constraint* is a quantifier-free formula constructed from (in)equalities of polynomials. It is *linear* if it contains only linear expressions. A semialgebraic set is a set described by a constraint:

Definition 2. *A semialgebraic set in $\mathbb{R}^k$ is a finite union of sets of the form $\{x \in \mathbb{R}^k | f(x) = 0 \wedge \bigwedge_{g \in \mathcal{G}} g > 0\}$, where f is a polynomial and $\mathcal{G}$ is a finite set of polynomials.*

A polynomial $p(\mathbf{X}_n) \in \mathbb{R}[\mathbf{X}_n]$ is a *sum of squares* (or SOS, for short), if there exist polynomials $f_1(x), \ldots, f_m(x) \in \mathbb{R}[\mathbf{X}_n]$ such that $p(\mathbf{X}_n) = \sum_{i=1}^m f_i^2(\mathbf{X}_n)$. Chapters 2 and 3 of [2] introduce a way to transform the problem whether a given polynomial is an SOS into a *semidefinite programming problem* (or SDP, for short), which is a generalization of linear programming problem. We introduce the transformation and SDP problems briefly in our technical report [12].

2.1 Probabilistic Programs

We use a simple *probabilistic guarded-command language* to construct *probabilistic programs* with the grammar:

$$P ::= \mathsf{skip} \mid \mathsf{abort} \mid x := E \mid P; P \mid P\ [p]\ P \mid \mathsf{if}\ (G)\ \mathsf{then}\ \{P\}\ \mathsf{else}\ \{P\} \mid \mathsf{while}(G)\{P\}$$

where E is a real-valued expression and G is a Boolean guard defined by the grammar:

$E ::= c \mid \mathbf{x}_n \mid r \mid$	constant/variable/random variable
$\quad E + E \mid E \cdot E \mid$	arithmetic
$G ::= E < E \mid G \wedge G \mid \neg G$	guards

Random variable r follows a given probability distribution, discrete or continuous. For $p \in [0, 1]$, the *probabilistic choice command* P_0 $[p]$ P_1 executes P_0 with probability p and P_1 with probability $1 - p$.

Example 3. The following probabilistic program P describes a simple game:

$$z := 0;\ \mathsf{while}(0 < x < y)\ \{x := x + 1[0.5]x := x - 1;\ z := z + 1\}.$$

The program models a game where a player has x dollars at the beginning and keeps tossing a coin with probability 0.5. The player wins one dollar if he tosses a head and loses one dollar for a tail. The game ends when the player loses all his money, or he wins $y - x$ dollars for a predetermined y. The variable z records the number of tosses made by the player during this game.

We assume that the reader is familiar with the basic concepts of probability theory and in particular expectations, see e.g. [11] for details. Expectations are typically functions from program states (i.e. the real-valued program variables) to $\mathbb{R}$. An expectation is called a *post-expectation* when it is to be evaluated on the final distribution, and it is called a *pre-expectation* when it is to be evaluated on the initial distribution. Let *preE*, *postE* be expectations and *prog* a probabilistic program. We say that the sentence $\langle preE\rangle$ *prog* $\langle postE\rangle$ *holds* if the expected value of *postE* after executing *prog* is equal to or greater than the expected value of *preE*. When *postE* and *preE* are functions, the comparison is executed pointwise.

Classical programs can be viewed as special probabilistic programs in the following sense. For classical precondition *pre* and postcondition *post*, let the characteristic function χ_{pre} equal 1 if the precondition is true and 0 otherwise, and define χ_{post} similarly. If one considers a Hoare triple $\{pre\}$ *prog* $\{post\}$ where *prog* is a classical program, then it holds if and only if $\langle \chi_{pre}\rangle$ *prog* $\langle \chi_{post}\rangle$ holds in the probabilistic sense.

2.2 Probabilistic Predicate Transformers

Let P_0, P_1 be probabilistic programs, E an expression, *post* a post-expectation, *pre* a pre-expectation, G a Boolean expression, and $p \in (0, 1)$. The probabilistic predicate transformer *wp* can be defined as follows [16]:

$$
\begin{aligned}
&wp(\mathsf{skip}, post) = post\\
&wp(\mathsf{abort}, post) = 0\\
&wp(x := E, post) = post[x/\mathbb{E}_S(E)]\\
&wp(P;\ Q, post) = wp(P, wp(Q, post))\\
&wp(\mathsf{if}(G)\ \mathsf{then}(P)\ \mathsf{else}(Q), post) = \chi_G \cdot wp(P, post) + (1 - \chi_G) \cdot wp(Q, post)\\
&wp(P\ [p]\ Q, post) = p \cdot wp(P, post) + (1 - p) \cdot wp(Q, post)\\
&wp(\mathsf{while}(G)\ \{P\}, post) = \mu X.(\chi_G \cdot wp(P, X) + (1 - \chi_G) \cdot post)
\end{aligned}
$$

Here $post[x/\mathbb{E}_S(E)]$ denotes the formula obtained by replacing free occurrences of x in $post$ by the expectation of E over state space S. The least fixed point operator μ is defined over the domain of expectations [23,24], and it can be shown that $\langle pre \rangle\ P\ \langle post \rangle$ holds if and only if $pre \leq wp(P, post)$. Thus, $wp(P, post)$ is the greatest lower bound of precondition expectation of P with respect to $post$, and we say $wp(P, post)$ is the *weakest pre-expectation of P w.r.t. post.*

2.3 Positivstellensatz

Hilbert's Nullstellensatz is very important in algebra, and its real version, known as Positivstellensatz, is crucial to our method. First, some concepts are needed to introduce the theorem.

- The set $P \subseteq \mathbb{R}[\mathbf{X}_n]$ is a *positive cone* if it satisfies: (i) If $a \in \mathbb{R}[\mathbf{X}_n]$, then $a^2 \in P$, and (ii) P is closed under addition and multiplication.
- The set $M \subseteq \mathbb{R}[\mathbf{X}_n]$ is a *multiplicative monoid with 0* if it satisfies: (i) $0, 1 \in M$, and (ii) M is closed under multiplication.
- The set $I \subseteq \mathbb{R}[\mathbf{X}_n]$ is an *ideal* if it satisfies: (i) $0 \in I$, (ii) I is closed under addition, and (iii) If $a \in I$ and $b \in \mathbb{R}[\mathbf{X}_n]$, then $a \cdot b \in I$.

We are interested in finitely generated positive cones, multiplicative monoids with 0, and ideals. Let $\mathcal{F} = \{f_1, \ldots, f_s\}$ be a finite set of polynomials. We recall that

- Any element in the positive cone generated by $\mathcal{F}$ (i. e., the smallest positive cone containing $\mathcal{F}$) is of the form
$$\sum_{\alpha \in \{0,1\}^s} k_\alpha \mathcal{F}^\alpha \quad \text{where } k_\alpha \text{ is a sum of squares for all } \alpha \in \{0,1\}^s$$
In the sum, α denotes an s-length vector with each element 0 or 1.
- Any nonzero element in the multiplicative monoid with 0 generated by $\mathcal{F}$ is of the form $\mathcal{F}^\alpha$, where $\alpha = (\alpha_1, \ldots, \alpha_s) \in \mathbb{N}^s$.
- Any element in the ideal generated by $\mathcal{F}$ is of the form $k_1 f_1 + k_2 f_2 + \cdots + k_s f_s$, where $k_1, \ldots, k_s \in \mathbb{R}[\mathbf{X}_n]$.

The Positivstellensatz due to Stengle states that for a system of real polynomial equalities and inequalities, either there exists a solution, or there exists a certain polynomial which guarantees that no solution exists.

Theorem 4 (Stengle's Positivstellensatz [29]). *Let $(f_j)_{j=1}^s, (g_k)_{k=1}^t, (h_l)_{l=1}^w$ be finite families of polynomials in $\mathbb{R}[\mathbf{X}_n]$. Denote by P the positive cone generated by $(f_j)_{j=1}^s$, by M the multiplicative monoid with 0 generated by $(g_k)_{k=1}^t$, and by I the ideal generated by $(h_l)_{l=1}^w$. Then the following are equivalent:*

1. *The set*
$$\left\{ \mathbf{z}_n \in \mathbb{R}^n \,\middle|\, \begin{array}{l} f_j(\mathbf{z}_n) \geq 0,\ j = 1, \ldots, s \\ g_k(\mathbf{z}_n) \neq 0,\ k = 1, \ldots, t \\ h_l(\mathbf{z}_n) = 0,\ l = 1, \ldots, w \end{array} \right\} \tag{1}$$
is empty.
2. *There exist $f \in P, g \in M, h \in I$ such that $f + g^2 + h = 0$.*

3 Problem Formulation

The question that concerns us here is to verify whether the loop sentence

$$\langle preE\rangle \ \mathsf{while}(G)\ \{body\}\ \langle postE\rangle$$

holds, when given the pre-expectation *preE*, post-expectation *postE*, a Boolean expression G, and a loop-free probabilistic program *body*. One way to solve this problem is to calculate the weakest pre-expectation $wp(\mathsf{while}(G, \{body\}), postE)$ and to check whether it is not smaller than *preE*. However, the weakest pre-expectation of a while statement requires a fixed-point computation, which is not trivial. To avoid the fixed point, the problem can be solved through a quantitative loop invariant.

Theorem 5 [15]**.** *Let preE be a pre-expectation, postE a post-expectation, G a Boolean expression, and body a loop-free probabilistic program. To show*

$$\langle preE\rangle \ \mathit{while}(G)\ \{body\}\ \langle postE\rangle,$$

it suffices to find a loop invariant I which is an expectation such that

1. *(boundary)* $preE \le I$ *and* $I\cdot(1-\chi_G) \le postE$*;*
2. *(invariant)* $I\cdot\chi_G \le wp(body, I)$*;*
3. *(soundness) the loop terminates with probability 1 from any state that satisfies G, and*
   (a) *the number of iterations is finite, or*
   (b) *I is bounded from above by some fixed constant, or*
   (c) *the expected value of* $I\cdot\chi_G$ *tends to zero as the number of iterations tends to infinity.*

Since soundness of a loop invariant is not related to pre- and postconditions and can be verified from its type before any specific invariants are found, we focus on the boundary and invariant conditions in Theorem 5. The soundness property is left to be verified manually in case studies.

For pre-expectation *preE* and post-expectation *postE*, the boundary and invariant conditions in Theorem 5 provide the following requirements for a loop invariant I:

$$\begin{aligned} preE &\le I \\ I\cdot(1-\chi_G) &\le postE \\ I\cdot\chi_G &\le wp(body, I). \end{aligned} \tag{2}$$

The inequalities induced by the boundary and invariant conditions contain indicator functions, which may be difficult to analyze if they appear multiple times. So we rewrite them to a standard form. For a Boolean expression F, we use $[F]$ to represent its integer value, i.e. $[F] = 1$ if F is true, and $[F] = 0$ otherwise. An expectation is in *disjoint normal form* (DNF) if it is of the form

$$f = [F_1]\cdot f_1 + \cdots + [F_k]\cdot f_k$$

where the F_i are disjoint expressions, which means any two of the expressions cannot be true simultaneously, and the f_i are polynomials.

Lemma 6 [21]**.** *Suppose* $f = [F_1]\cdot f_1 + \cdots + [F_k]\cdot f_k$ *and* $g = [G_1]\cdot g_1 + \cdots + [G_l]\cdot g_l$ *are expectations over* $\mathbf{X}_n$ *in DNF. Then,* $f \le g$ *if and only if (pointwise)*

$$\bigwedge_{i=1}^{k}\bigwedge_{j=1}^{l}\left[F_i \wedge G_j \Rightarrow f_i \le g_j\right] \wedge \bigwedge_{i=1}^{k}\left[F_i \wedge \left(\bigwedge_{j=1}^{l} \neg G_j\right) \Rightarrow f_i \le 0\right] \wedge \bigwedge_{j=1}^{l}\left[\left(\bigwedge_{i=1}^{k} \neg F_i\right) \wedge G_j \Rightarrow 0 \le g_j\right]. \quad (3)$$

Example 7. Consider the following loop sentence for our running example:

$\langle xy - x^2\rangle\ z := 0;\ \mathsf{while}(0 < x < y)\{x := x + 1\ [0.5]\ x := x - 1; z := z + 1;\}\ \langle z\rangle$

For this case, the following must hold for any loop invariant I.

$$xy - x^2 \le I$$
$$I \cdot [x \le 0 \vee y \le x] \le z$$
$$I \cdot [0 < x < y] \le 0.5 \cdot I(x+1, y, z+1) + 0.5 \cdot I(x-1, y, z+1)$$

By Lemma 6, these requirements can be written as

$$xy - x^2 \le I \wedge \quad (4)$$

$$x \le 0 \vee y \le x \Rightarrow I \le z \wedge \quad (5)$$

$$0 < x < y \Rightarrow 0 \le z \wedge \quad (6)$$

$$0 < x < y \Rightarrow I \le 0.5 \cdot I(x+1, y, z+1) + 0.5 \cdot I(x-1, y, z+1) \wedge \quad (7)$$

$$x \le 0 \vee y \le x \Rightarrow 0 \le 0.5 \cdot I(x+1, y, z+1) + 0.5 \cdot I(x-1, y, z+1) \quad (8)$$

The program in this example originally served as a running example in [7]. There, after transforming the constraints into the form above, Lagrange interpolation was applied to synthesize the template coefficients. Our approach asserts the correctness of each conjunct in (4–8) by checking the nonnegativity of the polynomial on the right side over a semialgebraic set related to polynomials on the left side. In this way, we use the Positivstellensatz to synthesize the coefficients.

4 Constraint Solving by Semidefinite Programming

Our aim is to synthesize coefficients for the fixed invariant template for simple (Subsect. 4.1) and nested (Subsect. 4.2) programs. Checking the validity of constraints can be transformed into checking the emptiness of a semialgebraic set. Then, we show that the emptiness problem can be turned into sum-of-squares constraints using Stengle's Positivstellensatz.

Our Approach in a Nutshell. For a given polynomial template as a candidate quantitative loop invariant, it needs to satisfy *boundary* and *invariant* conditions. Our goal is to synthesize the coefficients in the template. These conditions describe a semialgebraic set, and the satisfiability of the constraints is equivalent to the non-emptiness of the corresponding semialgebraic set. Applying the Positivstellensatz (see Sect. 2.3), we will transform the problem to an equivalent semidefinite programming problem using Lemma 8. Existing efficient solvers can be used to solve the problem. A more efficient yet sufficient way is to transform the problem into a *sum-of-squares problem* using Lemma 9 and then to solve it by semidefinite programming. After having synthesized the coefficients of the template, we verify whether they are valid. In case of a negative answer, which may happen due to numerical errors, some amendments can be made by adding further constraints, which is described in Sect. 4.3. If the problem is still unsolved, we try raising the degree of the template and reiterate the procedure.

4.1 Synthesis Algorithm for Simple Loop Programs

Now we are ready for the transformation method. Each conjunct obtained in Lemma 6 is of the form $F \Rightarrow G$, where F is a quantifier-free formula constructed from (in)equalities between polynomials in $\mathbb{R}^{\leq d}[\mathbf{X}_n]$, and G is of the form $f \leq g$, $f \leq 0$ or $0 \leq g$, with $f, g \in \mathbb{R}^{\leq d}[\mathbf{X}_n]$. If F contains negations, we use De Morgan's laws to eliminate them. If there is a disjunction in F, we split the constraints into sub-constraints as $\varphi \vee \psi \Rightarrow \chi$ is equivalent to $(\varphi \Rightarrow \chi) \wedge (\psi \Rightarrow \chi)$. After these simplifications, $F \Rightarrow G$ can be written in the form $\bigwedge_i (f_i \trianglerighteq_i 0) \Rightarrow g \geq 0$ where $\trianglerighteq_i \in \{\geq, =\}$. Observe that a constraint $\bigwedge_i (f_i \trianglerighteq_i 0) \Rightarrow g \geq 0$ is satisfied if and only if the set $\{x | f_i(x) \trianglerighteq_i 0 \text{ for all } i;\ -g(x) \geq 0; \text{ and } g(x) \neq 0\}$ is empty. In this way, we transform our constraint into the form required by Theorem 4.

Summarizing, Constraint (2) (the main condition of Theorem 5) is satisfied if and only if all semialgebraic sets created using the procedure above are empty. Now we are ready to transform this constraint to an SDP problem.

Lemma 8 [9,25]**.** *The emptiness of* (1) *is equivalent to the feasibility of an SDP problem.*

This and the following results are proven in the technical report [12] accompanying this publication. Although the transformation in Lemma 8 is effective, it is complicated in practice. In the following lemma we present a simpler yet sufficient procedure.

Lemma 9. *The following statements hold (with $\trianglerighteq_i \in \{\geq, =\}$):*

1. $f(\mathbf{X}_n) \geq 0 \Rightarrow g(\mathbf{X}_n) \geq 0$ *holds if* $g(\mathbf{X}_n) - u \cdot f(\mathbf{X}_n)$ *is a sum of squares for some* $u \in \mathbb{R}_{\geq 0}$.
2. $f(\mathbf{X}_n) = 0 \Rightarrow g(\mathbf{X}_n) \geq 0$ *holds if* $g(\mathbf{X}_n) - v \cdot f(\mathbf{X}_n)$ *is a sum of squares for some* $v \in \mathbb{R}$.
3. $f_1(\mathbf{X}_n) \trianglerighteq_1 0 \wedge f_2(\mathbf{X}_n) \trianglerighteq_2 0 \Rightarrow g(\mathbf{X}_n) \geq 0$ *holds if* $g(\mathbf{X}_n) - r_1 \cdot f_1(\mathbf{X}_n) - r_2 \cdot f_2(\mathbf{X}_n)$ *is a sum of squares for some* $r_1, r_2 \in \mathbb{R}$*; if* $\trianglerighteq_i$ *is* $\geq$*, it is additionally required that* $r_i \geq 0$.

Note that Item (3) can be strengthened slightly by adding a cross product $r_{12} f_1(\mathbf{X}_n) f_2(\mathbf{X}_n)$ and squares of the $f_i(\mathbf{X}_n)$.

Example 10. Applying the above procedure, Constraint (5) in Example 7 is split into $(x \leq 0 \Rightarrow I \leq z) \wedge (y \leq x \Rightarrow I \leq z)$ and then normalized to $(-x \geq 0 \Rightarrow z - I \geq 0) \wedge (x - y \geq 0 \Rightarrow z - I \geq 0)$. This holds if $z - I + u_1 x$ is a SOS for some $u_1 \in \mathbb{R}_{\geq 0}$ and $z - I + u_2(y - x)$ is a SOS for some $u_2 \in \mathbb{R}_{\geq 0}$. The other constraints can be handled in a similar way.

After applying the Positivstellensatz and Lemma 8, template coefficients for the loop invariant can be synthesized efficiently by semidefinite programming. See our technical report [12] for details on the corresponding technique.

Algorithm 1. Loop Invariant Generation with Refinement

Input: *sentence* := $\langle preE \rangle$ while$(G)\{body\}$ $\langle postE \rangle$ with program variables $\mathbf{X}_n$
Output: a loop invariant satisfying the boundary and invariant conditions

```
 1: loop
 2:    $d := 2$
 3:    Choose a template for $I \in \mathbb{R}^{\leq d}[\mathbf{X}_n]$
 4:    Let $f$ be Constraint (2), i.e. the boundary and invariant conditions from Theo-
       rem 5, for sentence
 5:    Let constraints be the SDP problem equivalent to $f$ according to Lemma 8
 6:    while constraints is feasible do
 7:       Set the coefficients in the template for $I$
 8:       Round the coefficients of $I$ into rational numbers
 9:       if $I$ satisfies the boundary and invariant conditions then
10:          Output $I$ and terminate
11:       end if
12:       Refine constraints
13:    end while
14:    $d := d + 2$
15: end loop
```

We summarize our approach in Algorithm 1. The aim is to synthesize the coefficients of template I. The terms in I are all terms with degree $\leq d$ from variables in $\mathbf{X}_n$. Algorithm 1 is semi-complete in the sense that it will generate an invariant if there exists one. Its termination is guaranteed in principle by Theorem 4 and the equivalence between SOS and SDP in Lemma 8, though due to numerical errors, the algorithm may fail to find I in practice.

For efficiency, Lemma 9 is often used instead of Lemma 8. Step 5 in Algorithm 1 is replaced by: "Let *constraints* be the relaxation of f to an SOS problem according to Lemma 9"; this can be translated to an equivalent SDP problem, which is simpler than the direct translation of Lemma 8.

Example 11. We extend Example 7 using Lemma 9. To illustrate our solution method, we choose Constraints (4), (5), and (7). The initial condition $z = 0$ is not included in these constraints, so (4) needs to be refined to $z = 0 \Rightarrow xy - x^2 \leq I$.

First, we set a template for I. Assume I as a quadratic polynomial with three variables x, y, z:

$$I = c_0 + c_1 x + c_2 y + c_3 z + c_{11} x^2 + c_{12} xy + c_{13} xz + c_{22} y^2 + c_{23} yz + c_{33} z^2$$

where $c_0, \ldots, c_{33} \in \mathbb{R}$ are coefficients that remains to be determined.

For Constraint (4) with initial constraint $z = 0$, we get the following corresponding constraint:

$$I - (xy - x^2) - v \cdot z \geq 0 \tag{4'}$$

For (5), the antecedens is a conjunction of two constraints. As in Example 10, (5) is split into two constraints and transformed into

$$\begin{gathered} z - I + u_1 \cdot x \geq 0 \quad \text{and} \\ z - I - u_2 \cdot (x - y) \geq 0 \end{gathered} \tag{5'}$$

For (7), the constraint $0 < x < y$ needs to be split into two inequalities $x > 0 \wedge y - x > 0$. Similarly to (5), we transform (7) to

$$0.5 \cdot I(x+1, y, z+1) + 0.5 \cdot I(x-1, y, z+1) - I - u_3 \cdot x - u_4 \cdot (y - x) \geq 0 \tag{7'}$$

In this way the example can be transformed into an SDP problem with constraints (4′), (5′), and (7′), and positivity constraints on the multipliers $u_1 \geq 0, \ldots, u_4 \geq 0$. (For v, an arbitrary real value is allowed.) Then the resulting SDP problem can be submitted to any SOS solver.

The result using solver SeDuMi [30] is shown below.

$$\begin{aligned} &I = -7.1097 \cdot 10^{-10} - 3.8818 \cdot 10^{-10} x - 0.4939 \cdot 10^{-10} y + z - x^2 + xy + \\ &2.7965 \cdot 10^{-10} xz + 0.97208 \cdot 10^{-10} y^2 + 4.4656 \cdot 10^{-10} yz - 0.28694 \cdot 10^{-10} z^2 \end{aligned}$$

If we ignore the amounts smaller than the order of magnitude of 10^{-6}, we get $I = z - x^2 + xy$. This I satisfies all constraints (including (6) and (8), which are similar to (5) and (7)), so it is correct.

4.2 Synthesis Algorithm for Nested Loop Programs

We are now turning to programs containing nested loops. To simplify our discussion, we assume the program only contains a single, terminating inner loop, i.e. it can be written as

$$\begin{aligned} P &= \mathsf{while}(G)\{\mathit{body}\} \\ &= \mathsf{while}(G)\{\mathit{body1};\ \mathsf{while}(G_{\mathrm{inn}})\{\mathit{body}_{\mathrm{inn}}\};\ \mathit{body2}\} \end{aligned}$$

where $\mathit{body1}$, $\mathit{body}_{\mathrm{inn}}$, and $\mathit{body2}$ are loop-free program fragments. (If the inner loop is placed within an if statement, one can transform it to the above form by strengthening G.) For a given preE and postE, we need to verify whether there exists an invariant I that satisfies Constraint (2) (the boundary and invariant conditions of Theorem 5). We denote the inner loop by $P_{\mathrm{inn}} = \mathsf{while}(G_{\mathrm{inn}})\{\mathit{body_{inn}}\}$.

For such a program, the main difficulty is how to deal with $\mathit{wp}(\mathit{body}, I)$ in Constraint (2). We propose a method here that takes the inner and outer iteration into consideration together and uses the verified pre-expectation of the inner loop to relax the constraint.

Fix templates for the polynomial invariants: I for the outer loop and I_{inn} for the inner loop P_{inn}, both with degree d. Since $\mathit{body2}$ is loop-free, it is easy to obtain $\tilde{I} := \mathit{wp}(\mathit{body2}, I)$. We use $\tilde{I}$ as post-expectation $\mathit{postE}_{\mathrm{inn}}$ for the inner loop. Note that (2) for the inner loop requires $\mathit{preE}_{\mathrm{inn}} \leq I_{\mathrm{inn}}$, so we can use the template I_{inn} also as template for $\mathit{preE}_{\mathrm{inn}}$. Then the constraints for loop invariants I and I_{inn} are

$$\begin{aligned} \mathit{preE} &\leq I \\ I \cdot [1 - \chi_G] &\leq \mathit{postE} \\ I \cdot \chi_G &\leq \widetilde{\mathit{wp}}(\mathit{body}, I) := \mathit{wp}(\mathit{body1}, \mathit{preE}_{\mathrm{inn}}) \\ \mathit{preE}_{\mathrm{inn}} &= I_{\mathrm{inn}} \\ I_{\mathrm{inn}} \cdot [1 - \chi_{G_{\mathrm{inn}}}] &\leq \mathit{postE}_{\mathrm{inn}} = \mathit{wp}(\mathit{body2}, I) \\ I_{\mathrm{inn}} \cdot \chi_{G_{\mathrm{inn}}} &\leq \mathit{wp}(\mathit{body}_{\mathrm{inn}}, I_{\mathrm{inn}}) \end{aligned} \tag{9}$$

The first three equations are almost Constraint (2) for the outer loop, except that $\widetilde{\mathit{wp}}$ is the strengthening of the weakest pre-expectation using $\mathit{preE}_{\mathrm{inn}} = I_{\mathrm{inn}}$ in the wp-calculation instead of $\mathit{wp}(P_{\mathrm{inn}}, \tilde{I})$. The last three equations are Constraint (2) for the inner loop, except that we require equality in $\mathit{preE}_{\mathrm{inn}} \leq I_{\mathrm{inn}}$.

Then we have the following lemma, proven in our technical report [12].

Lemma 12. *An invariant I that satisfies Constraint* (9) *also satisfies* (2)*, therefore it is a loop invariant for program P.*

4.3 Handling Numerical Error

In practice, it sometimes happens that numerical errors lead to wrong or trivial coefficients in the templates. We suggest several methods to refine the constraints and avoid these errors.

Due to the inaccuracy of floating-point calculations, it is hard for a software to check equations and inequalities like $x = 0$ or $x \neq 0$. A common trick to avoid this problem is to turn the equality constraint into $x \geq 0 \wedge x \leq 0$. As for inequalities, taking $x \neq 0$ as an example, a way to solve the problem is adding a new variable y to transform the constraint into $xy \geq 1$, since $xy \geq 1$ implies $x \neq 0$ for any value of y. The new constraints are in the form required by Theorem 4.

Numerical errors may also lead to an unsound invariant: we may get some coefficients with a small magnitude, which often result from floating-point inaccuracies. A common solution for this problem is to ignore those small numbers, usually smaller than 10^{-6} in practice, which means that if r is presented as $\frac{a_0}{a_1}$ with $a_1 > 0$, then it is the closest to r in all rational numbers having smaller denominator. In Example 11, eliminating the terms with a small order of magnitude was successful, but we cannot be sure whether the resulting invariant is correct if the remaining coefficients are approximate. We propose to check the soundness of such solutions symbolically as follows. Checking whether the generated invariant satisfies Constraint (2) is a special case of quantifier elimination $\forall \mathbf{x}_n \in \mathbb{R}^n,\ f(\mathbf{x}_n) \geq 0$. Such problem can be solved efficiently using an improved Cylindrical Algebraic Decomposition (CAD) algorithm implemented in [17]. In our experiments in Sect. 5, the found solutions are obtained by ignoring small numbers, and we verified they are correct by running CAD in a separate tool.

If the invariant still violates some of the constraints, we can try to strengthen the constraint (e. g., change $x \geq 0$ to $x \geq 0.1$) and repeat our method.

5 Experimental Results

We have implemented a prototype in Python to test our technique. We call the MATLAB toolbox YALMIP [22] with the SeDuMi solver [30] to solve the SDP feasibility problem. We use the math software Maple to verify the correctness of the constraints through CAD. The experiments were done on a computer with Intel(R) Core(TM) i7-4710HQ CPU and 16 GiB of RAM. The operating system is Window 7 (32bit). Constraint refinement cannot be handled automatically in the current version, but we plan to add it together with projection for rounding solutions in a future version.

Our prototype and the detailed experimental results can be found at http://iscasmc.ios.ac.cn/ppsdp. For each probabilistic program, we give the pre- and post-expectations in Table 1. The annotated pre-expectation serves as an exact estimate of the annotated post-expectation at the entrance of the loop. We apply the method to several different types of examples. A summary of the results is shown in Table 1. The first eleven probabilistic programs are benchmarks taken from paper [7]. We have further constructed three case studies to illustrate continuous distributions, polynomial probabilistic programs and nested loop programs. Detailed descriptions and the code of all examples are available from [12]. After generating an invariant, we ran CAD in Maple manually to verify its feasability.

Table 1. The column "Name" shows the name of each experiment. The annotated pre- and post-expectations are shown in the columns "*preE*" and "*postE*". The inferred quantitative loop invariant for each example is given in the column "Invariant". The column "Time" gives the running time needed of our tool: the first one is the total running time, and the second one is the time used in the SeDuMi solver.

Name	*preE*	*postE*	Invariant	Time (s)
ruin	$xy - x^2$	z	$z + xy - x^2$	0.4/0.3
bin1	$x + \frac{1}{4}ny$	x	$x + \frac{1}{4}ny$	0.4/0.2
bin2	$\frac{1}{8}n^2 - \frac{1}{8}n + \frac{3}{4}ny$	x	$x + \frac{1}{8}n^2 - \frac{1}{8}n + \frac{3}{4}ny$	0.7/0.3
bin3	$\frac{1}{8}n^2 - \frac{1}{8}n + \frac{3}{4}ny^2$	x	$x - 0.0057n - 0.0014x^2 + 0.1763xn + 712.909n^2 + 0.0014x^2n + 0.4114xn^2 + 0.4188ny^2 - 0.0178n^3$	0.7/0.3
geo	$x + 3zy$	x	$x + 3zy$	0.2/0.2
geo2	$x + \frac{15}{2}$	x	$x + 30.2312y + 3.4699z - 12.6648y^2 - 44.6591yz - 35.5112z^2 - 22.8807$	0.2/0.1
sum	$\frac{1}{4}n^2 + \frac{1}{4}n$	x	$x + \frac{1}{4}n^2 + \frac{1}{4}n$	0.3/0.1
prod	$\frac{1}{4}n^2 - \frac{1}{4}n$	xy	$-\frac{1}{4}n + xy + \frac{1}{2}xn + \frac{1}{2}yn + \frac{1}{4}n^2$	0.3/0.1
fair coin1	$\frac{1}{2} - \frac{1}{2}x$	$1 - x + xy$	$0.7130 - 0.5622x + 0.3364y + 0.8564n - 1.2740x^2 + 07610xy - 1.4572xn - 1.2208y^2 + 1.4572yn - 0.1366n^2$	0.2/0.1
fair coin2	$\frac{1}{2} - \frac{1}{2}y$	$x + xy$	$1.1941 + 1.6157x + 0.6387y + 7.9774n - 14.6705x^2 + 9.7904xy - 14.9948xn - 14.6457y^2 + 14.9948yn - 1.4058n^2$	0.3/0.1
fair coin3	$\frac{8}{3} - \frac{8}{3}x - \frac{8}{3}y + \frac{1}{3}n$	n	$6.0556 + 2.5964x + 3.2468y + 39.2052n - 69.9038x^2 + 44.0224xy - 72.1408xn - 69.8067y^2 + 72.1408yn - 6.7632n^2$	0.2/0.1
simple perceptron	$-2b$	n	$n - 2b$	0.3/0.1
airplane delay	$106.548x$	h	$106.548x - 106.548n + h$	0.4/0.2
airplane delay2	$282.507x$	h	$282.507(x - n) + h$	0.5/0.2
nested loop	$20(m - x)$	k	$k + 20(m - x)$	1.6/1.1

As we can see from Table 1, the running time of our method is within one second. There are some notes when calculating the examples. We relax the loop condition $z \neq 0$ in example geo2 into $z \geq 0.5$. Also in the fair coin example, we relax the loop condition $x \neq y$ into $x - y \geq 0.5 \vee y - x \geq 0.5$. Since variables in those two examples are integers, the relaxation is sound.

5.1 Evaluation

Other approaches to compute loop invariants in probabilistic programs are the Lagrange Interpolation Prototype (LIP) in [7], the tool for martingales (TM) in [3] and Prinsys in [15]. The tools are executed on the same computer, LIP and TM under Linux and the other two under Windows. In Table 2, we compare the features supported by the four tools.

Table 2. Comparison of the features supported by 4 tools

	Prinsys	LIP	TM	Our tool
Type of program	Linear	Cubic	Linear	Polynomial
Type of invariant	Linear	Polynomial	Linear	Polynomial
Computation method	Symbolic	Symbolic	Numerical	Numerical
Distribution of variable	Discrete	Discrete	Continuous	Continuous

We have tested the examples in Table 1 on these four tools. Prinsys takes the longest time and fails to verify any of non-linear examples presented. LIP fails to verify any examples that include a continuous variable or have a degree larger than 3; additionally it is always about 10 times slower than our tool. TM fails to verify examples ruin, bin3 and geo directly. We observe that it cannot treats constraints of the form $x = y$ or $x \neq y$ (where x and y might be variables or constants). However, by transforming $x = y$ into $x \geq y \wedge y \geq x$, TM can synthesize a supermartingale for the program. Also, it cannot verify the simple perceptron, as it is a non-linear program. Furthermore, TM cannot deal with nested loop programs.

A weakness of our approach is that it depends heavily on the number of variables. We have constructed an artificial example to expose this: a parametric linear program that repeats t times the probabilistic assignment

$$h := h + x_1 + \cdots + x_n \ [0.5] \ h := h + x_1 + \cdots + x_n + \text{UnifDist}(0, 2n)$$

This program has $n + 2$ variables. Table 3 compares the time consumption of the main technical step in our prototype. Adding five variables leads to a solver time that is about 2.5 times higher, showing that the measured solver time is exponential in the number of variables. The full description and code are, again, in our report [12].

Table 3. Comparison of running time (in seconds) of the parameterized linear example

Number of variables	$n = 15$	$n = 20$	$n = 25$	$n = 30$	$n = 35$	$n = 40$
Solver time of our tool	0.41	1.30	2.44	8.30	20.56	46.62

6 Conclusion

In this paper, we propose a method to synthesize polynomial quantitative invariants for recursive probabilistic programs by semialgebraic programming via a Positivstellensatz. First, a polynomial template is fixed whose coefficients remain to be determined. The loop and its pre- and post-expectation can be transformed into a semialgebraic set, of which the emptiness can be decided by finding a counterexample satisfying the condition of the Positivstellensatz. Semidefinite programming provides an efficient way to synthesize such a counterexample. The method can be applied to polynomial programs containing continuous or discrete variables, including those with nested loops. When numerical errors prevent finding a loop invariant polynomial right away, we currently can correct them *ad hoc* (by deleting terms with very small coefficients and sometimes strengthening the constraints), but we would like to develop a more systematic treatment.

As future improvement, we are working on the handling of numerical errors. A better approximation can be found by projecting $\tilde{I}(x)$ onto a rational subspace defined by SDP constraints [19,26]. There are also acceleration methods for different types of probabilistic programs: For *linear* programs, Handelman's Positivstellensatz describes a faster way to synthesize SOS constraints, and for *Archimedean* programs, [9] describes a faster way to apply Stengle's Positivstellensatz.

Acknowledgement. This work has been supported by the National Natural Science Foundation of China (Grants 61532019 and 61472473), the CAS/SAFEA International Partnership Program for Creative Research Teams, and the Sino-German CDZ project CAP (GZ 1023).

References

1. Barthe, G., Espitau, T., Ferrer Fioriti, L.M., Hsu, J.: Synthesizing probabilistic invariants via Doob's decomposition. arXiv preprint arXiv:1605.02765 (2016)
2. Blekherman, G., Parrilo, P.A., Thomas, R.R. (eds.): Semidefinite Optimization and Convex Algebraic Geometry. SIAM, Philadelphia (2012). doi:10.1137/1.9781611972290
3. Chakarov, A., Sankaranarayanan, S.: Probabilistic program analysis with martingales. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 511–526. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_34
4. Chakarov, A., Sankaranarayanan, S.: Expectation invariants for probabilistic program loops as fixed points. In: Müller-Olm, M., Seidl, H. (eds.) SAS 2014. LNCS, vol. 8723, pp. 85–100. Springer, Cham (2014). doi:10.1007/978-3-319-10936-7_6
5. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through positivstellensatz's. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 3–22. Springer, Cham (2016). doi:10.1007/978-3-319-41528-4_1

6. Chatterjee, K., Fu, H., Novotný, P., Hasheminezhad, R.: Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In: Bodik, R., Majumdar, R. (eds.) POPL'16, pp. 327–342. ACM, New York (2016). doi:10.1145/2837614.2837639
7. Chen, Y.-F., Hong, C.-D., Wang, B.-Y., Zhang, L.: Counterexample-guided polynomial loop invariant generation by lagrange interpolation. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 658–674. Springer, Cham (2015). doi:10.1007/978-3-319-21690-4_44
8. Colón, M.A., Sankaranarayanan, S., Sipma, H.B.: Linear invariant generation using non-linear constraint solving. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 420–432. Springer, Berlin (2003). doi:10.1007/978-3-540-45069-6_39
9. Dai, L., Xia, B., Zhan, N.: Generating non-linear interpolants by semidefinite programming. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 364–380. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39799-8_25
10. Dijkstra, E.: A Discipline of Programming, vol. 4. Prentice-Hall, Englewood Cliffs (1976)
11. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1. Wiley, Hoboken (1968)
12. Feng, Y., Zhang, L., Jansen, D.N., Zhan, N., Xia, B.: Finding polynomial loop invariants for probabilistic programs. arXiv:1707.02690 (2017)
13. Ferrer Fioriti, L.M., Hermanns, H.: Probabilistic termination: soundness, completeness, and compositionality. In: POPL 2015, Principles of Programming Languages, pp. 489–501. ACM, New York (2015). doi:10.1145/2775051.2677001
14. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: Dwyer, M.B., Herbsleb, J. (eds.) Future of Software Engineering (FOSE 2014), pp. 167–181. ACM, New York (2014). doi:10.1145/2593882.2593900
15. Gretz, F., Katoen, J.-P., McIver, A.: PRINSYS—on a quest for probabilistic loop invariants. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 193–208. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40196-1_17
16. Gretz, F., Katoen, J.-P., McIver, A.: Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. Perf. Eval. **73**, 110–132 (2014). doi:10.1016/j.peva.2013.11.004
17. Han, J., Jin, Z., Xia, B.: Proving inequalities and solving global optimization problems via simplified CAD projection. J. Symb. Comput. **72**, 206–230 (2016). doi:10.1016/j.jsc.2015.02.007
18. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969). doi:10.1145/363235.363259
19. Kaltofen, E.L., Li, B., Yang, Z., Zhi, L.: Exact certification in global polynomial optimization via sums-of-squares of rational functions with rational coefficients. J. Symb. Comput. **47**(1), 1–15 (2012). doi:10.1016/j.jsc.2011.08.002
20. Kaminski, B.L., Katoen, J.-P.: On the hardness of almost–sure termination. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9234, pp. 307–318. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48057-1_24
21. Katoen, J.-P., McIver, A.K., Meinicke, L.A., Morgan, C.C.: Linear-invariant generation for probabilistic programs: In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 390–406. Springer, Berlin (2010). doi:10.1007/978-3-642-15769-1_24

22. Löfberg, J.: YALMIP: a toolbox for modeling and optimization in MATLAB. In: 2004 IEEE International Symposium on Computer Aided Control Systems Design (CACSD), pp. 284–289. IEEE, Piscataway (2004). doi:10.1109/CACSD.2004.1393890
23. McIver, A., Morgan, C.C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer, New York (2005). doi:10.1007/b138392
24. Morgan, C., McIver, A., Seidel, K.: Probabilistic predicate transformers. ACM Trans. Progr. Lang. Syst. **18**(3), 325–353 (1996). doi:10.1145/229542.229547
25. Parrilo, P.A.: Semidefinite programming relaxations for semialgebraic problems. Math. Program. Ser. B **96**(2), 293–320 (2003). doi:10.1007/s10107-003-0387-5
26. Peyrl, H., Parrilo, P.A.: Computing sum of squares decompositions with rational coefficients. Theoret. Comput. Sci. **409**(2), 269–281 (2008). doi:10.1016/j.tcs.2008.09.025
27. Putinar, M.: Positive polynomials on compact semi-algebraic sets. Indiana Univ. Math. J. **42**(3), 969–984 (1993)
28. Rodríguez-Carbonell, E., Kapur, D.: Generating all polynomial invariants in simple loops. J. Symb. Comput. **42**(4), 443–476 (2007). doi:10.1016/j.jsc.2007.01.002
29. Stengle, G.: A nullstellensatz and a positivstellensatz in semialgebraic geometry. Math. Ann. **207**(2), 87–97 (1974). doi:10.1007/BF01362149
30. Sturm, J.F.: Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optim. Methods Softw. **11**(1–4), 625–653 (1999). doi:10.1080/10556789908805766