Synthesis for Polynomial Lasso Programs*

Jan Leike[1] and Ashish Tiwari[2]**

[1] University of Freiburg, Germany. leike@informatik.uni-freiburg.de
[2] SRI International, Menlo Park, CA. ashish.tiwari@sri.com

Abstract. We present a method for the synthesis of polynomial lasso programs. These programs consist of a program stem, a set of transitions, and an exit condition, all in the form of algebraic assertions (conjunctions of polynomial equalities). Central to this approach is the discovery of non-linear (algebraic) loop invariants. We extend Sankaranarayanan, Sipma, and Manna's template-based approach and prove a completeness criterion. We perform program synthesis by generating a constraint whose solution is a synthesized program together with a loop invariant that proves the program's correctness. This constraint is non-linear and is passed to an SMT solver. Moreover, we can enforce the termination of the synthesized program with the support of test cases.

1 Introduction

There have been significant advances in automating program verification, and even extending the verification techniques to perform automated synthesis of correct programs. Often, automation is achieved using appropriate abstract domains for analysis. The choice of abstract domains is governed by the class of program fragments being analyzed. In this paper, we are interested in programs that perform some numerical computation. For reasoning about such programs, the theory of polynomial ideals has proven to be an excellent abstract domain because of two reasons. First, there is a nice correspondence between subsets of the program state space and polynomial ideals (as established in the field of algebraic geometry), and second, there are effective algorithms for computing with polynomial ideals. In this paper, we will use the abstract domain of polynomial ideals for reasoning about polynomial lasso programs.

In our terminology, a polynomial lasso program consists of an assertion describing program states before loop entry, an assertion describing program states after loop termination and a set of transitions corresponding to the branches in the loop body. All involved assertions are algebraic; that is, conjunctions of polynomial equalities.

Our approach for analysis of such polynomial lasso programs is not based on iterative fixpoint computation. Instead, we use the constraint-based approach,

* The final publication is available at link.springer.com.
** Supported in part by DARPA under contract FA8750-12-C-0284 and by the NSF grant SHF:CSR-1017483.

also known as template-based approach, for directly finding fixpoints using constraint solving. This way we avoid convergence issues of iterative fixpoint methods. Our starting point is a method presented by Sankaranarayanan, Sipma and Manna [19]. Despite its obvious incompleteness, the method is often successful in verifying programs. Why is this method "complete in practice"? We answer the question here by presenting a first completeness criterion for this method. For this purpose, we have to extend the original invariance criteria in [19] and generate a new and refined *invariance condition.*

Our interest here is not just on the verification problem, but also on the synthesis problem. Specifically, taking inspiration from recent work on synthesis of programs by completing partial program "sketches" [21,10], we start with a polynomial lasso program that contains parameters (variables to be synthesized) and a post condition. The goal is to find values for the parameters that result in a correct program. We solve the synthesis problem by generating a *synthesis constraint*—a constraint whose solution provides a valuation for the parameters. Additionally, the constraint's solution also supplies values that define an inductive loop invariant for the synthesized polynomial lasso program. This invariant constitutes as proof that the synthesized program is in fact correct with respect to the given post condition. Thus, we simultaneously synthesize the program and its proof of correctness. There is one caveat though: if variables that are critical to termination have parameterized updates, then the synthesized lasso program might not be terminating. To solve this problem, we use a finite number of test cases that specify input variable assignment, output variable assignment and a sequence of loop transitions. These test cases are used to strengthen the synthesis constraint so that the undesirable solutions are eliminated.

The template-based approach reduces the synthesis problem and the loop invariant discovery problem into an $\exists\forall$ constraint: the template variables and the synthesis variables are existentially ($\exists$) quantified, whereas the program variables are universally ($\forall$) quantified [10]. We use the theory of polynomial ideals to (conservatively) eliminate the inner $\forall$ quantifier. The resulting formula is our synthesis constraint – an (existentially quantified) conjunction of non-linear algebraic equalities – which is solved by an off-the-shelf non-linear SMT solver.

We demonstrate that the template-based approach on polynomial ideals abstract domain can be used to successfully synthesize polynomial lasso programs. However, the approach has certain limitations. First, it cannot handle inequalities. Polynomial ideals logically correspond to conjunctions of polynomial equalities. Now, inequalities can be encoded as equalities, but algorithms on polynomial ideals (that compute canonical Gröbner basis) do not lift easily to reasoning about the encoded inequalities [24]. For handling inequalities, one could use semialgebraic sets as the abstract domain, and then use algorithms based on either *cylindric algebraic decomposition* [5] or the *Positivstellensatz* [23,15,24], but we leave that for future work.

A second issue is the size of the synthesis constraint. Non-linear solvers scale very poorly with increasing number of variables and the synthesis constraint (generated by the synthesis process) can be large and tends to be non-linear.

The final issue is related to the completeness of our approach. Incompleteness arises due to the use of templates, and also due to the use of polynomial ideal theory rather than the theory of reals. We address the latter issue in section 5. For the former issue, we just have to use polynomial templates with sufficiently large degree bounds. In our examples, a general template of degree two or three was sufficient, but the size of generic template polynomials grows exponentially with their degree.

2 Related Work

The automatic discovery of polynomial invariants for imperative programs has received a lot of attention in recent years. Müller-Olm and Seidl generate invariant polynomial equalities of bounded degree by backwards propagation [13]. This can be seen as an extension to Karr's algorithm [12], which uses only linear arithmetic. Seidl, Flexeder and Petter apply the backwards-propagation method to programs over machine integers, i.e., programs whose variables range over the domain $\mathbb{Z}_{2^w}$ [20].

Rodrígues-Carbonell and Kapur use an iterative approach based on forward propagation and fixed point computation on Gröbner bases over the lattice of ideals to generate the ideal of all loop invariants [17,18].

Colón combines the two aforementioned approaches by doing the fixed point computation on ideals with linear algebra [7]. He introduces the notion of pseudo-ideals to ensure termination of the fixed point computation while retaining the expressiveness of generated invariants.

Polynomial program invariants can also be derived without using Gröbner basis computations [4]. Cachera et al. use backwards analysis and variable substitution on template polynomials for an incomplete approach.

The constraint solving approach that generates invariant polynomial equalities using templates was proposed by Sankaranarayanan, Sipma and Manna [19]. Invariant generation is a central ingredient to our synthesis method, so we want the invariant generation process to be as complete as possible. Therefore we extend their approach by using a more general condition for the invariant (see also Remark 2) that enables us to state a completeness criterion.

Polynomial lasso programs have also received some attention regarding the analysis of their termination properties. Bradley, Manna and Sipma use finite difference arithmetic to compute lexicographic polynomial ranking functions for polynomial lasso programs [3].

All the aforementioned papers consider the verification (or the invariant generation) problem. In this paper, inspired by recent work on program synthesis [21], we also consider the synthesis problem. Our work can be considered a more formal approach to Colón's method [6] that uses non-linear constraint solving to instantiate program schemata (parameterized programs augmented with constraints). Our approach relies on algebraic methods instead of heuristics.

Finally, Srivastava et al. [22] describe a big-picture program synthesis algorithm from scaffolds. These scaffolds consist of pre- and postconditions, a program flow template, and bounds on the number of variables and the number of local branches. For the synthesis condition, all control flows of the template program are unfolded and constraints are generated with respect to invariants and ranking functions ensuring the program's correctness and termination. This constraint is then proven by a specialized external method and our algorithm can be used as one of these external methods.

3 Preliminaries

Let V be a set of variables, $V = \{x_1, \ldots, x_n\}$. The variables of the 'next state' are denoted by the corresponding primed variables $V' = \{x'_1, \ldots, x'_n\}$. Having both primed and unprimed variables in an expression enables stating a relationship between two states.

For the set of real numbers $\mathbb{R}$, let $\mathbb{R}[V]$ denote the ring of polynomials in the variables V with coefficients from $\mathbb{R}$. A subset $I \subseteq \mathbb{R}[V]$ is an *ideal* if (a) $0 \in I$, (b) $f + g \in I$ for all $f, g \in I$, and (c) $h \cdot f \in I$ for all $f \in I$ and $h \in \mathbb{R}[V]$. For a set of polynomials $P = \{p_1(V), \ldots, p_k(V)\}$, *the ideal* $\langle P \rangle$ *generated by* P is

$$\langle P \rangle = \langle p_1, \ldots, p_k \rangle = \Big\{ \sum_{i=1}^{k} q_i(V) p_i(V) \,\Big|\, q_1, \ldots, q_k \in \mathbb{R}[V] \Big\}.$$

Note that if all polynomials in P evaluate to 0 at any point in $\mathbb{R}^n$, then all polynomials in $\langle P \rangle$ will also evaluate to 0 at that point.

By the Hilbert Basis Theorem, every ideal I has a finite set of generators. Moreover, for a fixed ordering on the monomials (such as total degree lexicographic ordering induced by any precedence relation on the variables), there is a finite "canonical" set of generators of I called a *Gröbner basis*. A Gröbner basis $G = \{g_1, \ldots, g_k\}$ for I has the following properties [8].

1. G is computable in DOUBLE-EXPSPACE from a set of generators of I (Buchberger's Algorithm).
2. For all $p \in \mathbb{R}[V]$, the result of division of p on G, denoted $\mathrm{NF}_G(p)$, is unique and does not depend on the order in which the division steps are performed.
3. For all $p \in \mathbb{R}[V]$, $\mathrm{NF}_G(p) = 0$ iff $p \in I$.

For example, if $P = \{xy - 2, x^2 - 4\}$ and we use the precedence $x \succ y$, then $G = \{x - 2y, y^2 - 1\}$ is a Gröbner basis for the ideal $\langle P \rangle$. Division of p on G can be performed by replacing x by $2y$ and replacing y^2 by 1 in p repeatedly. The result $\mathrm{NF}_G(x^2 + y^2 - 5)$ of division of $x^2 + y^2 - 5$ on G is 0, and hence we can conclude that $x^2 + y^2 - 5 \in \langle P \rangle$.

Definition 1 (Radical Ideal). *An ideal I is a* radical ideal *if $f^m \in I$ implies $f \in I$ for every $m \in \mathbb{N}$.*

Given an ideal I, note that the set $\{f \mid \exists m \in \mathbb{N} : f^m \in I\}$ is a (radical) ideal.

Definition 2 (Algebraic Assertion). *An* algebraic assertion $\varphi(V)$ *(or just* φ*) over the set of variables* V *is a formula of the form* $\bigwedge_{i=1}^{m} p_i(V) = 0$ *where each* $p_i \in \mathbb{R}[V]$ *for* $1 \leq i \leq m$*.*

An algebraic assertion $\bigwedge_{i=1}^{m} p_i(V) = 0$ generates an ideal $\langle\varphi\rangle = \langle p_1, \ldots, p_m\rangle$. We will use φ to denote the formula as well as the set of polynomials $\{p_1, \ldots, p_m\}$ in the formula. An assertion φ can be interpreted in the theory $\mathbb{R}$ of reals or in the theory $\mathbb{C}$ of complex numbers. A *valuation* is a mapping from variables to values (in the set of real numbers or the set of complex numbers). A polynomial in $\mathbb{R}[V]$ evaluates to a value (in $\mathbb{R}$ or $\mathbb{C}$) for a given valuation for V.

Theorem 1 (Zero Polynomial Theorem). *A polynomial* $p \in \mathbb{R}[V]$ *is zero for all possible valuations* $\nu : V \to \mathbb{R}$ *if and only if all of its coefficients are zero.*

Lemma 1. *Let* φ *be an algebraic assertion over* V *and* $p \in \mathbb{R}[V]$ *a polynomial. If* $p \in \langle\varphi\rangle$*, then* $\mathbb{R} \models \varphi(V) \to p(V) = 0$*.*

Theorem 2 (Hilbert's Nullstellensatz [8]). *Let* φ *be an algebraic assertion and* $p \in \mathbb{C}[V]$ *a polynomial. If* $\langle\varphi\rangle$ *is a radical ideal and* $\mathbb{C} \models \varphi(V) \to p(V) = 0$*, then* $p \in \langle\varphi\rangle$*.*

Lemma 2. *Let* $p, s \in \mathbb{R}[V]$ *and* $\langle\varphi\rangle \subseteq \mathbb{R}[V]$ *be an ideal. Then,* $p \in \langle s, \varphi\rangle$ *if and only if there is a polynomial* $t \in \mathbb{R}[V]$ *such that* $p - t \cdot s \in \langle\varphi\rangle$*.*

Proof. Let $\langle\varphi\rangle = \langle p_1, \ldots, p_k\rangle$. By definition, $p \in \langle s, \varphi\rangle$ iff there are $t, t_1, \ldots, t_k \in \mathbb{R}[V]$ such that $p = ts + \sum_i t_i p_i$. This is equivalent to $p - ts = \sum_i t_i p_i$, which holds iff $p - ts \in \langle\varphi\rangle$. □

Similar to the definition by Sankaranarayanan et al., we introduce template polynomials as a means for finding polynomials with certain properties. In our definition the template coefficients can be non-linear polynomials. For the mathematical details regarding template polynomials, see [19].

Definition 3 (Template Polynomial). *Let* A *and* V *be two disjoint sets of variables. A* template polynomial *or* template *over* (A, V) *is a polynomial with variables* V *and coefficients from* $\mathbb{R}[A]$*. A template is said to be a* linear template *if all of its coefficient polynomials are linear.*

Template polynomials will be denoted by upper case Greek letters. Given a degree bound d, the *generic template polynomial* Ψ over (A, V) of total degree d is given by

$$\Psi(V) = \sum_{|\gamma| \leq d} a_\gamma V^\gamma$$

where $\gamma \in \mathbb{N}^{\#V}$ is a multi-index and $A = \{a_\gamma \mid \gamma \in \mathbb{N}^{\#V}\}$ are template variables.

Definition 4 (Semantics of Templates). *For a set of template variables* A*, an* A-valuation *is a map* $\alpha : A \to \mathbb{R}$*. This map can be naturally extended to a map* $\tilde{\alpha} : \mathbb{R}[A][V] \to \mathbb{R}[V]$ *that replaces every occurrence of an* $a \in A$ *by* $\alpha(a)$*.*

4 Polynomial Lasso Programs

We define the syntax and semantics of polynomial lasso programs. We also define inductive invariants for such programs. Henceforth, semantic entailment, $\models$, should always be interpreted as in the theory $\mathbb{R}$ of reals.

Definition 5 (Polynomial Lasso Program). *A* polynomial lasso program $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$ *consists of*

- *a set of variables V,*
- *an algebraic assertion* stem *over V called the* program stem*,*
- *a* set of transitions $\mathcal{T}$*, where each transition $\tau \in \mathcal{T}$ is an algebraic assertion over $V \cup V'$,*
- *and an algebraic assertion* exit *over V, called the* exit condition*.*

A transition τ is said to be deterministic *if it can be written in the form*

$$\bigwedge_j h_j(V) = 0 \ \wedge \ \bigwedge_i x_i' g_i(V) - f_i(V) = 0,$$

where every $x_i' \in V'$ occurs exactly once and $\neg\mathsf{exit} \models g_i(V) \neq 0$. For every i and j, the polynomial h_j is called guard *and the polynomial $x_i' g_i(V) - f_i(V)$ is called* update*: f_i is its* numerator *and g_i its* denominator*. The polynomial lasso program L is called* pseudo-deterministic *if all its transitions $\tau \in \mathcal{T}$ are deterministic.*

Lassos with solely deterministic transitions can have overlapping guards, hence the choice of transitions may be non-deterministic even in a pseudo-deterministic polynomial lasso program. Due to the nature of imperative languages, pseudo-deterministic lassos possess a specific interest to us.

Definition 6 (Semantics of a Lasso Program). *Let $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$ be a polynomial lasso program. An* execution *of L is a (potentially infinite) sequence $\sigma = \nu_0 \nu_1 \ldots$ where $\nu_i : V \to \mathbb{R}$ is a valuation on the variables V such that*

1. $\nu_0 \models \mathsf{stem}$
2. *For all $i \geq 0$ there is a $\tau \in \mathcal{T}$ such that $\tau(\nu_i, \nu_{i+1})$.*
3. *$\nu_i \models \mathsf{exit}$ iff it is the last element in σ.*

Example 1 (Running example). Consider the imperative program and its lasso representation L shown in Figure 1. L is a pseudo-deterministic lasso program since τ is a deterministic transition with the two update polynomials $y' - y + 1$ and $s' - s - x_0$ and no guards. An execution of L is $\sigma = \nu_0 \nu_1$ where

$$\begin{aligned} \nu_0 &: \ x_0 \mapsto 3 \ y_0 \mapsto 1 \ y \mapsto 1 \ s \mapsto 0, \\ \nu_1 &: \ x_0 \mapsto 3 \ y_0 \mapsto 1 \ y \mapsto 0 \ s \mapsto 3. \end{aligned}$$

```
procedure product(x0, y0):
    s := 0;
    y := y0;
    while (y ≠ 0):
        s := s + x0;
        y := y - 1;
    return s;
```

Lasso program $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$:

$$\begin{aligned} V &= \{x_0, y_0, y, s\}, \\ \mathsf{stem} &\equiv s = 0 \land y = y_0, \\ \tau &\equiv y' = y - 1 \land s' = s + x_0, \\ \mathsf{exit} &\equiv y = 0, \\ \mathcal{T} &= \{\tau\} \end{aligned}$$

Fig. 1. An example imperative code and its representation as a polynomial lasso program (see Example 1). The program performs a multiplication by repeated addition.

Definition 7 (Correctness). *Let $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$ be a polynomial lasso program and let* post *be an algebraic assertion over V. The lasso L is said to be* (partially) correct with respect to the post condition post *if for every finite execution σ of L, the last valuation in σ is a model of* post. *L is* totally correct with respect to post *if it is partially correct with respect to* post *and it is terminating, i.e., there are no infinite executions of L.*

Definition 8 (Invariant). *Let $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$ be a polynomial lasso program. A polynomial $p \in \mathbb{R}[V]$ is called an* (inductive) invariant *of a transition $\tau \in \mathcal{T}$ if*

1. $\mathsf{stem} \models p(V) = 0$ *and*
2. $p(V) = 0 \land \tau(V, V') \land \neg\mathsf{exit} \models p(V') = 0$.

The polynomial p is called an (inductive) invariant of L *if it is an invariant of all transitions $\tau \in \mathcal{T}$.*

It is easily shown by means of induction that if p is an invariant of a lasso L, then for every execution σ of L and every valuation $\nu \in \sigma$, we have $\nu \models p = 0$.

Example 2. Example 1 calculates the product s of the two input values x_0 and y_0 by repeated addition. The polynomial lasso program L is partially correct with respect to the post condition $s = x_0 y_0$ and it is easy to check that $s + x_0 y - x_0 y_0 = 0$ is an invariant of L.

5 Polynomial Loop Invariants

In this section, we extend the approach for discovering loop invariants for polynomial lasso programs introduced by Sankaranarayanan, Sipma and Manna [19]. We define a weakened form of what they call *polynomial consecution*. We prove that under some restrictions, this is a complete approach for invariants over the complex numbers. The results established in this section will then be applied to program synthesis in section 6.

The first lemma relieves us in certain cases from the potentially very expensive computation of a Gröbner basis for the loop transitions. Specifically, for a deterministic transition τ, division by the Gröbner basis of τ is equivalent to substitution of the primed variables according to the update statements.

Lemma 3. *Let τ be a deterministic transition with at most one guard polynomial h and updates $x_i' - f_i(V)$ that have denominator 1. If $x_i' \succ x_j$ in the monomial ordering for all i and j, then the set $G = \{h(V)\} \cup \{x_i' - f_i(V) \mid 1 \leq i \leq n\}$ is a Gröbner basis of the ideal $\langle \tau \rangle$.*

For the remainder of this paper, let $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$ be a fixed pseudo-deterministic polynomial lasso program. We will now define a sufficient, and under some assumptions also necessary, condition for a template polynomial to be an invariant of L.

Definition 9 (Invariance Condition). *For each transition $\tau \in \mathcal{T}$, let q_τ be any common multiple of the denominators of the update statements of τ. (In particular, q_τ can be the product of all denominators.) Let Ψ be a template polynomial over (A, V) of total degree d. Let $s(V)$ be the generator of* exit *if it has only one generator and* 1 *otherwise. The* invariance condition $\mathrm{IC}(L, \Psi)$ of L for Ψ *is the conjunction of*

$$\mathrm{NF}_{\mathsf{stem}}(\Psi(V)) = 0,$$
$$\mathrm{NF}_\tau(q_\tau(V)^d \cdot s(V) \cdot \Psi(V')) = \Phi_\tau(V) \cdot \Psi(V), \text{ for all } \tau \in \mathcal{T},$$

where the polynomials Φ_τ are generic template polynomials over (B_τ, V) whose degrees are bounded by the result of the division $\mathrm{NF}_\tau(q_\tau(V)^d \cdot s(V) \cdot \Psi(V'))$ and B_τ are new disjoint sets of template variables.

The variables V and V' are universally quantified in the invariance condition, whereas the variables A and $(B_\tau)_{\tau \in \mathcal{T}}$ are existentially quantified. By the Zero Polynomial Theorem 1, the equations in the invariance condition hold for all valuations on $V \cup V'$ if and only if all the coefficients of the polynomials are identical to zero. Therefore the variables V and V' can be removed from the invariance condition yielding a constraint on the variables A and $(B_\tau)_{\tau \in \mathcal{T}}$.

Remark 1. The invariance condition is designed to allow completeness in a wide variety of cases. We provide some intuition for its components below, but for details the reader is referred to the proof of Theorem 4.

- The result of the division $\mathrm{NF}_\tau(q_\tau(V)^d \cdot s(V) \cdot \Psi(V'))$ may not yield $\Psi(V)$, but rather some multiple of $\Psi(V)$. Hence, we have the generic template polynomial Φ_τ in the invariance condition.
- If an update statement, say $x_i' g_i - f_i$, in τ contains a nontrivial denominator g_i, then we may not be able to remove x_i' from $\Psi(V')$ by division on τ. Since every monomial in $\Psi(V')$ contains at most d primed variables, therefore multiplying $\Psi(V')$ with the polynomial $q_\tau(V)^d$ guarantees that division by τ will eliminate all primed variables.
- When the exit condition $s(V) = 0$ holds, we do not need Ψ to be inductive. Hence, we use the product $\Psi(V') \cdot s(V)$, which encodes that Ψ holds in the next state *or* the exit condition is satisfied.

– If the exit condition is generated by more than one polynomial, we cannot use this trick for all generators, thus loosing completeness. For simplicity, we set $s = 1$ in those cases, but selecting one of the exit condition's generators as s will make the condition more complete (but also more complex).

Remark 2. The invariance condition in Definition 9 is more general than the condition used by Sankaranarayanan et al. [19]. They use the following inductiveness property:

$$\mathrm{NF}_\tau(\Psi(V')) - \lambda \cdot \mathrm{NF}_\tau(\Psi(V)) = 0,$$

where λ is a real-valued variable. This not only restricts Φ_τ to a template of degree 0, it also omits the additions we have discussed in Remark 1.

Example 3. In order to state the invariance condition for Example 1, we first fix a template polynomial Ψ over V. The general second-degree template polynomial over V is the following.

$$\begin{aligned}\Psi(V) = {} & a_0 x_0^2 + a_1 y_0^2 + a_2 y^2 + a_3 s^2 + a_4 x_0 y_0 + a_5 x_0 y + a_6 x_0 s \\ & + a_7 y_0 y + a_8 y_0 s + a_9 y s + a_{10} x_0 + a_{11} y_0 + a_{12} y + a_{13} s + a_{14}\end{aligned}$$

The invariance condition $\mathrm{IC}(L, \Psi)$ is given by the following equations.

$$\begin{aligned}0 = {} & a_0 x_0^2 + (a_1 + a_2 + a_7) y^2 + (a_4 + a_5) x_0 y + a_{10} x_0 + (a_{11} + a_{12}) y + a_{14} \\ 0 = {} & (a_0 + a_3 + a_6 - b a_0) x_0^2 y + (a_1 - b a_1) y_0^2 y + (a_2 - b a_2) y^3 + (a_3 - b a_3) y s^2 \\ & + (a_4 + a_8 - b a_4) x_0 y_0 y + (a_5 + a_9 - b a_5) x_0 y^2 + (a_6 + 2a_3 - b a_6) x_0 s y \\ & + (a_7 - b a_7) y_0 y^2 + (a_8 - b a_8) y_0 y s + (a_9 - b a_9) y^2 s \\ & + (a_{10} + a_{13} - a_9 - a_5 - b a_{10}) x_0 y + (a_{11} - a_7 - b a_{11}) y_0 y \\ & + (a_{12} - 2a_2 - b a_{12}) y^2 + (a_{13} - a_9 - b a_{13}) y s + (a_{14} - b a_{14}) y\end{aligned}$$

Here, $\Phi_\tau(V) = b \cdot y$ is the generic template polynomial over $B_\tau = \{b\}$ of degree 0 multiplied with y, the generator of exit (for simplicity of presentation, we abstained from using a generic template polynomial for Φ_τ). By Theorem 1, these two equalities yield 21 equations which are linear after assigning a value to b. The assignment $\alpha : A \cup B_\tau \to \mathbb{R}$ given by the following table is a solution to the invariance condition $\mathrm{IC}(L, \Psi)$.

	b	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}
α	1	0	0	0	0	−1	1	0	0	0	0	0	0	0	1	0

This yields the loop invariant $\tilde{\alpha}(\Psi) = s + x_0 y - x_0 y_0$ from Example 2.

Theorem 3 (Soundness). *If $\alpha : A \cup \bigcup_{\tau \in \mathcal{T}} B \to \mathbb{R}$ is an assignment for the template variables that is a solution to the invariance condition* $\mathrm{IC}(L, \Psi)$*, then $\tilde{\alpha}(\Psi)$ is an invariant of L.*

Proof. $\mathrm{NF}_{\mathsf{stem}}(\tilde{\alpha}(\Psi)) = 0$, hence $\tilde{\alpha}(\Psi) \in \langle \mathsf{stem} \rangle$, and therefore $\mathsf{stem} \models \tilde{\alpha}(\Psi) = 0$ according to Lemma 1. By the premise,

$$q_\tau(V)^d \, s(V) \, \tilde{\alpha}(\Psi)(V') - \tilde{\alpha}(\Phi_\tau)(V) \, \tilde{\alpha}(\Psi)(V) \in \langle \tau \rangle$$

for all $\tau \in \mathcal{T}$, therefore $q_\tau(V)^d s(V)\tilde{\alpha}(\Psi)(V') \in \langle \tau, \tilde{\alpha}(\Psi)(V)\rangle$ by Lemma 2, and from Lemma 1 follows

$$\tau(V, V') \wedge \tilde{\alpha}(\Psi)(V) = 0 \models q_\tau(V)^d \cdot s(V) \cdot \tilde{\alpha}(\Psi)(V') = 0.$$

Since q_τ is a common multiple of denominators of updates in τ and $\neg\mathsf{exit}$ holds before any transition τ, it follows that $\neg\mathsf{exit} \models q_\tau(V) \neq 0$ by Definition 5. With $\neg\mathsf{exit} \models s(V) \neq 0$ we conclude that

$$\tau(V, V') \wedge \tilde{\alpha}(\Psi)(V) = 0 \wedge \neg\mathsf{exit} \models \tilde{\alpha}(\Psi)(V') = 0. \quad \square$$

A criterion for the method's completeness is given by the following theorem. The Nullstellensatz is applicable only when one considers the theory of complex numbers, which in general admits a proper subset of loop invariants. Furthermore, the Nullstellensatz demands all involved ideals be radical ideals [8].

Theorem 4 (Completeness in $\mathbb{C}$). *Let $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$ be a polynomial lasso program with the complex loop invariant[3] $p \in \mathbb{R}[V]$. If $\alpha : A \to \mathbb{R}$ is a valuation such that $\tilde{\alpha}(\Psi) = p$, then α can be extended to a solution to the invariance condition if the following additional premises are met.*

1. *The lasso L is pseudo-deterministic.*
2. *The ideal $\langle \mathsf{stem} \rangle$ and the ideal $\langle p \rangle$ are both radical ideals.*
3. *The ideal $\langle \mathsf{exit} \rangle$ is generated by a single polynomial $s \in \mathbb{R}[V]$.*
4. *The guard $h = 0$ of each transition $\tau \in \mathcal{T}$ is equivalent to True (i.e., h is 0).*
5. *The monomial ordering $\succ$ is lexicographic and $x_i' \succ x_j$ for all i, j.*

Proof. The polynomial p is a loop invariant of L, so by Definition 8,

$$\mathsf{stem} \models_{\mathbb{C}} p(V) = 0 \text{ and} \tag{1}$$
$$p(V) = 0, \tau(V, V'), \neg\mathsf{exit} \models_{\mathbb{C}} p(V') = 0 \text{ for all } \tau \in \mathcal{T}. \tag{2}$$

The ideal $\langle \mathsf{stem} \rangle$ is a radical ideal by Premise 2, so according to Hilbert's Nullstellensatz, Equation (1) implies $p \in \langle \mathsf{stem} \rangle$; and hence, α satisfies the first part of the invariance condition (IC).

To prove that α can be extended to satisfy the second part of IC, note that Equation (2), combined with Premise 3, yields

$$p(V) = 0, \tau(V, V') \models_{\mathbb{C}} s(V)p(V') = 0.$$

Using the Nullstellensatz, for some positive number k, we have

$$\big(q_\tau(V)^d s(V) p(V')\big)^k \in \langle p, \tau \rangle.$$

Since h is 0, normalizing by τ is equivalent to replacing primed variables using the update expressions in τ, and hence,

$$s(V)^k r(V)^k \in \langle p, \tau \rangle, \quad \text{where } r(V) := \mathrm{NF}_\tau(q_\tau(V)^d p(V'))$$

[3] The assertions of Definition 8 hold in the theory of the complex numbers.

Note that $r(V)$ has no prime variables since Premise 5 ensures all prime variables are greater with respect to the monomial ordering $\succ$ than the unprimed variables. Therefore, $s(V)^k r(V)^k \in \langle p, \tau\rangle \cap \mathbb{R}[V]$. Now, there are two cases.
(Case 1): $\langle p, \tau\rangle \cap \mathbb{R}[V] = \langle p\rangle$. Then, it follows that $s(V)^k r(V)^k \in \langle p\rangle$. Since $\langle p\rangle$ is a radical ideal, we can infer $s(V)r(V) \in \langle p\rangle$ and hence $\mathrm{NF}_\tau(q_\tau(V)^d s(V)p(V'))$ is a multiple of p. Hence, second part of IC is satisfied.
(Case 2): $\langle p, \tau\rangle \cap \mathbb{R}[V] \neq \langle p\rangle$. This is possible only if some multiple of the denominators rewrites to 0 by p. Hence, $p = 0$ implies $s(V) = 0$ (since $s \neq 0$ implies that denominators are nonzero). Since $\langle p\rangle$ is a radical ideal, it follows $s \in \langle p\rangle$, and hence $s(V)r(V) \in \langle p\rangle$ — as in (Case 1) above. □

It is important to emphasize that the generic template polynomial for the invariant must have a sufficiently large degree to be able to specialize to the loop invariant. This is presumed in the completeness statement. We will now discuss the other premises of Theorem 4.

Premise 1 ensures that the division of $\Psi(V')$ on a transition τ removes all primed variables, since we multiplied with $q_\tau(V)^d$ in the invariance condition. Premise 2 is a requirement by Hilbert's Nullstellensatz. In order to write a disjunction of exit and a polynomial equality as a product, exit must have a single generator; this is stated in Premise 3. We will discuss relaxing Premise 4 below. Finally, Premise 5 assures that primed variables are eliminated first, leaving only unprimed variables in appropriate cases. This is relevant because the right hand side $\Phi_\tau(V) \cdot \Psi(V)$ in the invariant condition contains only unprimed variables.

Remark 3. We can generalize the completeness result to also include the case when guards of transitions are nontrivial and when a conjunction $p_1 = 0 \wedge p_2 = 0$ is an inductive invariant, but neither $p_1 = 0$ nor $p_2 = 0$ by itself is an inductive invariant. This requires generalizing the second part of the invariance condition. Let Ψ_1 and Ψ_2 be the templates whose instantiation gives p_1 and p_2 respectively. Then, for all τ in $\mathcal{T}$, and for $i = 1, 2$,

$$\mathrm{NF}_\tau(q_\tau(V)^d \cdot s(V) \cdot \Psi_i(V')) = \Phi_1(V) \cdot \Psi_1(V) + \Phi_2(V) \cdot \Psi_2(V) + \Phi_3(V) \cdot h_\tau(V)$$

Note that Φ_1, Φ_2, Φ_3 are different templates for different τ's and different i's. As before, the degrees of the templates are bounded by the degree of the left-hand side, and d is the total degree of Ψ_i. In the completeness theorem, we can now drop Premise 4, but replace Premise 2 by the following generalization:

2 The ideal stem, and for all τ, the ideals $\langle p_1, p_2, h_\tau\rangle$, where $h_\tau = 0$ is the guard of τ, are radical ideals. Moreover, $\{p_1, p_2, h_\tau\}$ is a GB of $\langle p_1, p_2, h_\tau\rangle$.

The proof of the new completeness claim is a natural generalization of the proof of Theorem 4 above. □

Besides the five restrictions of Theorem 4, completeness does not extend to the field of real numbers due to the requirements of Hilbert's Nullstellensatz. The underlying problem is illustrated by the following example.

Example 4. The formula $\varphi \equiv x_1^2 + x_2^2 = 0$ has $x_1 = x_2 = 0$ as its only solution over the reals. However, $x_1, x_2 \notin \langle x_1^2 + x_2^2 \rangle$, although $\langle x_1^2 + x_2^2 \rangle$ is a radical ideal and $\varphi \models_{\mathbb{R}} x_1 = 0, x_2 = 0$.

Alternatively, we could formulate our results using *real radical ideals* [14].

Because the invariance condition in general is a non-linear constraint, solving it might be very difficult. General approaches for solving non-linear constraints have worst case space requirements that are doubly exponential in the size of the input. However, non-linear constraint solving is an active field of research and recently there have been some promising efforts to take the practical cases away from their DOUBLE-EXPSPACE worst-case complexity bound [11].

Another approach for solving the invariance condition stems from the observation that the invarianc condition becomes linear if an assignment for the template variables $(B_\tau)_{\tau \in \mathcal{T}}$ is given. One could use heuristics to find this assignment. For instance, practical experience suggests that if a solution to a variable $b \in B_\tau$ is $\lambda_b \in \mathbb{R}$, then the factor $(b - \lambda_b)$ occurs somewhere in the invariance condition. Using factors in the former form as an initial guess for the variables $(B_\tau)_{\tau \in \mathcal{T}}$ linearizes the equations and thus enables quick discovery of a solution in some cases.

In the special case that $\Phi_\tau(V) := \lambda$ is degree 0 (also called constant consecution), λ can be found as an eigenvalue of an appropriate transformer constructed by interpreting bounded degree polynomials as finite-dimensional vector spaces [16].

6 Synthesis

The technique for finding a loop invariant using the invariance condition established in the previous section will now be used for program synthesis. Given a polynomial lasso program, some transition updates can be parameterized by replacing them with template polynomials. The synthesis process will try to find a valuation of these template variables while respecting some post condition. The following definition formalizes this concept.

Definition 10 (Synthesis Problem). *A* synthesis problem $S = (C, L, \mathsf{post})$ *consists of*

- *a set of* synthesis variables C*,*
- *a polynomial lasso program* $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$ *where* stem *and* $\tau \in \mathcal{T}$ *contain template polynomials over* (C, V)*, and*
- *a* post condition *in form of an algebraic assertion* post *over* V*.*

A solution to the synthesis problem S *is a valuation* $\alpha : C \to \mathbb{R}$ *such that the lasso* $L_\alpha = (V, \tilde{\alpha}(\mathsf{stem}), \tilde{\alpha}(\mathcal{T}), \mathsf{exit})$ *is partially correct with respect to the post condition* post*.*

Example 5. Transforming L from Example 1 to L' by changing the transition τ to

$$y' = y - 1 \land s' = c_1 x_0 + c_2 y_0 + c_3 y + c_4 s + c_5$$

gives rise to a synthesis problem $S = (C, L', \mathsf{post})$ for $C = \{c_1, c_2, c_3, c_4, c_5\}$ and $\mathsf{post} \equiv s = x_0y_0$. A solution to S is $\alpha : c_1 \mapsto 1, c_2 \mapsto 0, c_3 \mapsto 0, c_4 \mapsto 1, c_5 \mapsto 0$ since $L_\alpha = L$ and L is partially correct with respect to post according to Example 2.

Our approach for solving the synthesis problem is based on the technique from the previous section. We will prove the partial correctness of the synthesized lasso program. The following lemma states that synthesized polynomial lasso program will be partially correct.

Lemma 4 (Synthesis Solution). *Let $S = (C, L, \mathsf{post})$ be a synthesis problem, $\alpha : C \to \mathbb{R}$ be a valuation on the synthesis variables, and let p be an invariant for L_α. If $p = 0 \wedge \mathsf{exit} \models \mathsf{post}$, then L_α is partially correct with respect to* post*, i.e., α is a solution to S.*

Proof. Let $\sigma = \nu_0 \dots \nu_k$ be a finite execution of L_α. According to the assumption, p is an invariant of L_α, so by Definition 8, $\nu_i \models p = 0$ for all $0 \leq i \leq k$. By Definition 6, $\nu_k \models \mathsf{exit}$, therefore $\nu_k \models p = 0 \wedge \mathsf{exit}$. According to the assumption, this implies $\nu_k \models \mathsf{post}$, which proves the correctness of L_α. ☐

To find a valuation for the synthesis variables, we define a *synthesis condition*. The synthesis condition will constrain the synthesis variables so that existence of a loop invariant p that implies the post condition is guaranteed; that is,

$$p = 0 \wedge \mathsf{exit} \models \mathsf{post}. \tag{3}$$

If $\mathsf{post} = \bigwedge_i \mathsf{post}_i = 0$, then the above is implied by $\mathsf{post}_i \in \langle p, \mathsf{exit} \rangle$ by Lemma 1. However, computing the Gröbner basis with respect to a template polynomial for p is extremely inefficient and potentially involves a huge number of case splits. But according to Lemma 2, we can equivalently write

$$\mathsf{post}_i - tp \in \mathsf{exit}, \tag{4}$$

for some unknown $t \in \mathbb{R}[V]$. This enables us to rewrite (3) in a way that only involves computing the Gröbner basis for non-template polynomials.

Example 6. Let S be the synthesis problem from Example 5. We use the loop invariant $p = s + x_0y - x_0y_0$ from Example 2 in Lemma 4 to show that α is a solution to S by checking

$$s + x_0y - x_0y_0 = 0 \wedge y = 0 \models s = x_0y_0,$$

or instead that for $t = 1$,

$$(s - x_0y_0) - t(s + x_0y - x_0y_0) \in \mathsf{exit}.$$

Definition 11 (Synthesis Condition). *Let $S = (C, L, \bigwedge_{i=1}^m \mathsf{post}_i(V)=0)$ be a synthesis problem, let Ψ be a template polynomial over (A, V) and for all $0 \leq i \leq m$, let Ω_i be a template polynomial over (D_i, V). The* synthesis condition*, $\mathrm{SC}(S, \Psi, \{\Omega_i \mid 0 \leq i \leq m\})$, of S is the formula*

$$\mathrm{IC}(L, \Psi) \wedge \bigwedge_i \mathrm{NF}_{\mathsf{exit}}(\mathsf{post}_i(V) - \Omega_i(V)\Psi(V)) = 0$$

Following the same argument as for the invariant condition, the synthesis condition simplifies to a conjunction of non-linear equations in the variables $A \cup C \cup (\bigcup_{\tau \in \mathcal{T}} B_\tau) \cup (\bigcup_{i=1}^m D_i)$. Utilizing an SMT solver, a solution to this constraint can be obtained that is then used to instantiate the template polynomials in the loop invariant and polynomial lasso program. According to the next theorem, this yields a correct program instance.

Motivated by Example 6, we may set $\Omega_i = 1$ in the synthesis condition. In this case the constraint $\mathrm{NF}_{\mathsf{exit}}(\mathsf{post}_i(V) - \Psi(V)) = 0$, the synthesis condition's constraint corresponding to the post condition, is linear. Using this observation we can use linear methods to eliminate some variables from the constraint system, thus simplifying it. The same trick also applies to the constraint $\mathrm{NF}_{\mathsf{stem}}(\Psi(V)) = 0$ in the invariance condition if the program stem does not contain any synthesis variables C.

Because the coefficients of some of the polynomials in L contain template variables, special care must be taken when computing a Gröbner basis for stem or $\tau \in \mathcal{T}$. Every division by some term containing a variable demands a case split on whether this term evaluates to zero. One way of circumventing this problem is to compute a Gröbner basis where the underlying algebraic structure for polynomial coefficients is the ring of parameter polynomials $\mathbb{R}[A]$. This requires a slightly modified division algorithm [1,2].

Theorem 5 (Synthesis Soundness). *If $\alpha : A \cup (\bigcup_{\tau \in \mathcal{T}} B_\tau) \cup C \cup (\bigcup_i D_i) \to \mathbb{R}$ is an assignment for the template variables that models the synthesis condition, then L_α is partially correct with respect to the post condition* post *and $\tilde{\alpha}(\Psi)$ is an invariant of L_α.*

Proof. By Theorem 3, $\tilde{\alpha}(\Psi)$ is an invariant of L_α. By definition, $\mathsf{post}_i(V) - \tilde{\alpha}(\Omega_i)(V)\tilde{\alpha}(\Psi)(V) \in \langle \mathsf{exit} \rangle$, therefore $\mathsf{post}_i \in \langle \tilde{\alpha}(\Psi), \mathsf{exit} \rangle$ for all i according to Lemma 2. By Lemma 1, $\tilde{\alpha}(\Psi) \wedge \mathsf{exit} \models \mathsf{post}_i$ for all i, hence $\tilde{\alpha}(\Psi) \wedge \mathsf{exit} \models \mathsf{post}$. Lemma 4 ensures that this implies that L_α is partially correct. $\square$

The synthesis process is not complete, even when the restrictions of Theorem 4 hold. The reason for this is the polynomial t in (4): we are using templates Ω_i for t, but a priori we have no upper bound on the degree of t. In practice, a template of degree 0 might be sufficient, as in our examples (see section 8).

7 Termination

A solution to the synthesis problem guarantees partial correctness of the synthesized program; however, termination is not guaranteed. Even if the synthesized program terminates, it might be highly inefficient, going through unnecessarily many loop iterations.

Example 7. If one extends L' from Example 5 to L'' by changing τ to

$$y' = c_6 y + c_7 \wedge s' = c_1 x_0 + c_2 y_0 + c_3 y + c_4 s + c_5$$

this yields a synthesis problem $S' = (C', L'', \mathsf{post})$ with $C' = C \cup \{c_6, c_7\}$. Possible solutions to S' include the valuations $\alpha_\lambda : c_1 \mapsto \lambda, c_2 \mapsto 0, c_3 \mapsto 0, c_4 \mapsto 1, c_5 \mapsto 0, c_6 \mapsto 1, c_7 \mapsto -\lambda$ for all $\lambda \in \mathbb{R}$.

If λ is small, the program needs more iterations for the same input, and if λ is zero, L_α will not terminate at all.

In order to address this, the synthesis condition can be augmented with a series of test cases, predefined input-output pairs that explicitly state the transitions required to compute them.

Definition 12 (Test Case). *Let (C, L, post) be a synthesis problem where $L = (V, \mathsf{stem}, \mathcal{T}, \mathsf{exit})$ is a pseudo-deterministic polynomial lasso program containing template variables C. A* test case $t = (\nu_0, \nu, \tau_1 \dots \tau_k)$ *consists of two V-valuations ν_0 and ν corresponding to the initial and final state respectively such that $\nu \models \mathsf{exit}$, as well as a finite sequence of transitions $\tau_1, \dots, \tau_k \in \mathcal{T}$. A solution α to a synthesis problem S is said to* adhere to the test case t *if, for $\nu_i = \tau_i \circ \dots \circ \tau_1(\nu_0)$, the sequence $\sigma = \nu_0 \nu_1 \dots \nu_k$ is an execution of L_α and $\nu_k = \nu$.*

Lemma 5. *Let $S = (C, L, \mathsf{post})$ be a synthesis problem with solution α and let $t = (\nu_0, \nu, \tau_1 \dots \tau_k)$ be a test case. If*

$$\nu_0 \models \alpha(\mathsf{stem}), \tag{5}$$

$$\nu = \alpha(\tau_k) \circ \dots \circ \alpha(\tau_1)(\nu_0), \tag{6}$$

$$\nu_i \not\models \mathsf{exit} \textit{ for } 0 \le i \le k-1, \textit{ and} \tag{7}$$

$$\nu_k \models \mathsf{exit} \tag{8}$$

then L_α adheres to the test case t.

Proof. $\sigma = \nu_0 \nu_1 \dots \nu_k$ for $\nu_i = \alpha(\tau_i) \circ \dots \circ \alpha(\tau_1)(\nu_0)$ is by construction an execution of L_α according to Definition 6. From (6) follows that $\nu_k = \nu$. □

If we add the equations (5), (6), (7) and (8) to the synthesis condition for every given test case, then by Lemma 5 any solution to these constraints will yield a solution to S that adheres to the test cases.

Example 8. Consider the synthesis problem S' from Example 7. The execution σ from Example 1 gives rise to the test case $t = (\nu_0, \nu_1, \tau)$, which by Lemma 5 adds the following additional constraints on the synthesis condition.

$$\begin{array}{lll} 1 = 1 & 0 = 1c_6 + c_7 & 1 \neq 0 \\ 0 = 0 & 3 = 3c_1 + 1c_2 + 1c_3 + 0c_4 + c_5 & 0 = 0 \end{array}$$

The valuation α_1 is the only one of the valuations α_λ given in Example 7 that models these two equations (however, it is not the only possible solution). L_{α_1} is a terminating lasso program for positive integers y_0.

In theory, if it is possible to synthesize a terminating program, then there exists a finite set of test cases that will guarantee that a terminating lasso is synthesized.

Theorem 6. *Let $S = (C, L, \mathsf{post})$ be a synthesis problem. If there is a solution α to S such that L_α is terminating then there is a finite set of test cases Σ such that any solution β of S which adheres to all test cases $t \in \Sigma$ is terminating.*

Proof. Let $\Sigma = \{t_0, t_1, \ldots\}$ be the test cases to all possible executions of L_α, and assume Σ is infinite (otherwise there is nothing to show). Each test case $t \in \Sigma$ corresponds to a polynomial assertion over the variables C by (5) and (6). This assertion constrains possible assignments of C. For every $i \geq 0$, let $\Sigma_i = \{t_0, \ldots, t_i\} \subset \Sigma$ be an ascending chain of finite subsets of Σ and let I_i be the ideal generated by the assertions from the test cases of Σ_i. It is clear that $\Sigma_i \subset \Sigma_{i+1}$, and hence $I_i \subseteq I_{i+1}$. By the *Ascending Chain Condition* [8], the ascending chain of ideals $I_0 \subseteq I_1 \subseteq \ldots$ must become stationary for some integer k, meaning $I_k = I_i$ for all $i \geq k$. This implies that the finite set of test cases Σ_k corresponds to the same ideal as Σ_i for $i \geq k$ and hence they have the same solution (set of assignments) for C. As a consequence, any solution β to S adhering to the test cases from Σ_k will enforce that σ is an execution of L_β iff it is an execution of L_α. $\square$

While Theorem 6 assures that under any circumstances, a finite set of test cases Σ suffices to force a useful solution from the synthesis problem, no upper bound to the cardinality of Σ is given.

In theory, this provides us with two powerful approaches of generating polynomial lasso programs, given an a priori bound on the number of program variables V. Both involve creating a polynomial lasso program with generic template polynomials as updates and guards.

1. Specify a (large) number of test cases. Ideally, these test cases can be automatically generated in some sophisticated way that ensures that they are not too redundant.
2. Provide a post condition and a complexity guess. Using the complexity guess, a terminating skeleton of the synthesis problem is generated using counter variables. The post condition provides a statement regarding the program's purpose.

Needless to say, both approaches create very large synthesis conditions that are unlikely to be handled automatically by present-day non-linear solvers, but this can change, especially for small program fragments, as technology develops.

8 Experimental Evaluation

We implemented our method in Haskell and used `nlsat` [11][4] to solve the non-linear constraints. To evaluate the practicability and scalability of our method, we ran it on a few selected examples which are listed in Table 1 together with a short description. Each example translates to a pseudo-deterministic polynomial lasso program. See the appendix for the source code to the examples as well as the discovered solutions to the constraints.

[4] As implemented in z3 version 4.3.1. `http://z3.codeplex.com/`

name	*description*
`product`	multiplication of two integers by repeated addition (see Figure 1 and Example 1)
`productS`	`product` with synthesis of one update statement (see Example 5)
`productSY`	`product` with synthesis of the loop body, including the termination-critical variable y (see Example 7)
`product2`	`product` with reciprocal y
`product2S`	`product2` with synthesis of one update statement
`gcd_lcm`	greatest common denominator and least common multiple of two integers [19]
`gcd_lcmS`	`gcd_lcm` with synthesis of two update statements
`div_mod`	integer division with remainder [9]
`div_modS`	`div_mod` with synthesis of the complete loop body with linear updates
`root2`	integer square root [9]
`root2S`	`root2` with synthesis of the stem and one update statement
`squareS`	square of an integer synthesized from a terminating skeleton with linear assignments
`cubeS`	cube of an integer synthesized from a terminating skeleton with linear assignments

Table 1. Example programs used to perform verification and synthesis experiments described in Table 2. For the source code to the examples see the appendix.

Table 2 contains the experiment's results. We list the program name together with the number of synthesis variables (#C), the degree of the loop invariant's generic template polynomial (deg), the number of its template variables (#A), the total number of variables in the generated constraint (#vars), the number of test cases used (#tc), the time to generate the constraint in seconds (constraints time) and the running time of the SMT solver in seconds (solver time). Our test system was a computer with eight AMD Opteron 8220 2.80GHz CPUs and 32GB RAM.

While the synthesis process is very fast for small examples, the non-linear constraint solver becomes the bottleneck in medium-sized problems (`product2`, `product2S` and `cubeS` use generic templates of degree 3): solving non-linear constraints scales poorly with the number of variables involved. Test cases might help mitigate this issue by significantly reducing the solution space.

9 Conclusion

We presented a method for synthesizing polynomial programs. This method is based on the discovery of non-linear loop invariants that prove the program's correctness. We generate a *synthesis condition*, a non-linear constraint whose solution is the synthesized polynomial lasso program and a loop invariant. We extended existing methods for non-linear invariant generation and provided a completeness criterion (Theorem 4). If we synthesize update statements of variables that occur in the exit condition, termination becomes a concern. We showed

name	$\#C$	deg	$\#A$	#vars	#tc	*constraints time (s)*	*solver time (s)*
`product`	0	2	15	20	0	0.55	0.02
`productS`	5	2	15	25	0	1.47	0.01
`productSY`	7	2	15	27	2	3.39	0.02
`product2`	0	3	35	50	0	39.23	128.24
`product2S`	5	3	35	55	0	200.20	24.46
`gcd_lcm`	0	2	28	42	0	11.85	0.02
`gcd_lcmS`	10	2	28	52	0	17.01	0.01
`div_mod`	0	2	15	16	0	0.62	0.01
`div_modS`	10	2	15	26	5	10.03	0.03
`root2`	0	2	15	25	0	2.80	4.52
`root2S`	9	2	15	34	0	3.80	0.02
`squareS`	6	2	10	20	0	0.56	0.00
`cubeS`	14	3	35	54	0	90.88	41.05

Table 2. Experimental results showing the time required to verify/synthesize various example programs, along with the size of the non-linear constraints solved in the process.

that we can utilize a finite set of test cases to restrict the solution space to terminating lassos (Theorem 6).

Using a benchmark of small examples, we showed that our method is applicable for the synthesis of small programs, as well as parts of medium-sized ones. A resource bottleneck is the non-linear constraint solver. As the solving of non-linear constraints is an active area of research, we expect that our technique will become more effective as non-linear solvers improve.

We assumed that the programs' variables take values in the set of reals $\mathbb{R}$, but since Gröbner bases are computable over rings [1,2], our method can also be applied to the integers $\mathbb{Z}$ or the finite ring of machine integers $\mathbb{Z}_{2^w}$ (see also [20]). Future work could also consider the question of how this method can be improved to handle inequalities.

Acknowledgements We would like to thank the anonymous reviewers for their valuable feedback.

References

1. W. W. Adams and P. Loustaunau. *An Introduction to Gröbner Bases*. American Mathematical Society, 1994.
2. L. Bachmair and A. Tiwari. D-bases for polynomial ideals over commutative noetherian rings. In *RTA*, 1997.
3. A. R. Bradley, Z. Manna, and H. B. Sipma. Polyranking for polynomial loops, 2005.
4. D. Cachera, T. Jensen, A. Jobin, and F. Kirchner. Inference of polynomial invariants for imperative programs: A farewell to Gröbner bases. In *SAS*, pages 58–74, 2012.

5. G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *2nd GI Conf. Automata Theory and Formal Languages*, 1975.
6. M. A. Colón. Schema-guided synthesis of imperative programs by constraint solving. In *LOPSTR*, pages 166–181, 2004.
7. M. A. Colón. Polynomial approximations of the relational semantics of imperative programs. *Sci. Comput. Program.*, 64(1), 2007.
8. D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra.* Springer, 1991.
9. E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.
10. S. Gulwani, S. Jha, A. Tiwari, and R. Venkatesan. Synthesis of loop-free programs. In *PLDI*, 2011.
11. D. Jovanović and L. de Moura. Solving non-linear arithmetic. *IJCAR*, 2012.
12. M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6:133–151, 1976.
13. M. Müller-Olm and H. Seidl. Computing polynomial program invariants. *Inf. Process. Lett.*, 91(5), 2004.
14. R. Neuhaus. Computation of real radicals of polynomial ideals — ii. *Journal of Pure and Applied Algebra*, 124(1–3):261 – 280, 1998.
15. P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming Ser. B*, 96(2), 2003.
16. R. Rebiha, N. Matringe, and A. V. Moura. Endomorphisms for non-trivial non-linear loop invariant generation. In *ICTAC*, LNCS 5160, pages 425–439, 2008.
17. E. Rodríguez-Carbonell and D. Kapur. Automatic generation of polynomial loop invariants: Algebraic foundations. In *ISSAC*. ACM, 2004.
18. E. Rodríguez-Carbonell and D. Kapur. Program verification using automatic generation of invariants. In *ICTAC*, 2005.
19. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Non-linear loop invariant generation using Gröbner bases. *POPL*, 2004.
20. H. Seidl, A. Flexeder, and M. Petter. Analysing all polynomial equations in Z_{2^w}. *SAS*, 2008.
21. A. Solar-Lezama, L. Tancau, R. Bodík, V. Saraswat, and S. Seshia. Combinatorial sketching for finite programs. In *ASPLOS*, 2006.
22. S. Srivastava, S. Gulwani, and J. S. Foster. From program verification to program synthesis. In *POPL '10*, pages 313–326, 2010.
23. G. Stengle. A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Math. Ann.*, 207, 1974.
24. A. Tiwari. An algebraic approach for the unsatisfiability of nonlinear constraints. In *CSL*, 2005.

Appendix: Source Code to the Experiments

This appendix lists the source code for the example programs used in section 8 (compare Table 1 for a short description). The programs are given in pseudo-code rather than polynomial lasso programs for improved readability and for completeness: some parts of the program code have to be omitted in the translation to polynomial lasso programs (e.g. the exit condition in `div_mod` and `div_modS` is an inequality). However, this translation is straightforward. The result is a *pseudo-deterministic* polynomial lasso program in each example. We provide the assignment α to the generated constraints as found by the SMT solver. In our examples, we set $\Omega_i = 1$ in the synthesis condition.

For the source code to the program `product` see Figure 1. Also compare Example 1 and Example 3. We use the generic template of degree 2 over the variables $V = \{x_0, y_0, y, s\}$.

$$\tilde{\alpha}(\Psi) = \frac{1}{4}(x_0 y_0 - x_0 y - s)$$
$$\tilde{\alpha}(\Phi) = y$$

```
procedure productS(x₀, y₀):
    s := 0;
    y := y₀;
    while (y ≠ 0):
        s := c₀x₀ + c₁y₀ + c₂y + c₃s + c₄;
        y := y − 1;
    assert(s = x₀y₀);
    return s;
```

From Example 5. $C = \{c_0, c_1, c_2, c_3, c_4\}$; we use the generic template of degree 2 over the variables $V = \{x_0, y_0, y, s\}$.

$$\alpha(c_0) = 1, \quad \alpha(c_1) = \alpha(c_2) = 0, \quad \alpha(c_3) = 1, \quad \alpha(c_4) = 0$$
$$\tilde{\alpha}(\Psi) = s - x_0 y_0 + x_0 y$$
$$\tilde{\alpha}(\Phi) = y$$

```
procedure productSY(x₀, y₀):
    s := 0;
    y := y₀;
    while (y ≠ 0):
        s := c₀x₀ + c₁y₀ + c₂y + c₃s + c₄;
        y := c₅y + c₆;
    assert(s = x₀y₀);
    return s;
```

Test cases:

- `productSY(3, 1) == 1` (1 loop iteration)
- `productSY(3, 2) == 6` (2 loop iterations)

From Example 7. $C = \{c_0, c_1, c_2, c_3, c_4\}$; we use generic template of degree 2 over the variables $V = \{x_0, y_0, y, s\}$.

$$\alpha(c_0) = 1, \quad \alpha(c_1) = -\frac{1}{2}, \quad \alpha(c_2) = 1, \quad \alpha(c_3) = 1, \quad \alpha(c_4) = -\frac{1}{2},$$
$$\alpha(c_5) = 1, \quad \alpha(c_6) = -1$$
$$\tilde{\alpha}(\Psi) = s - x_0 y_0 + x_0 y + \frac{1}{2} y^2 - \frac{1}{2} y_0 y$$
$$\tilde{\alpha}(\Phi) = y$$

```
procedure product2(x0, y0):
    s := x0;
    y := 1/y0
    while (y ≠ 1):
        s := s + x0;
        y := y/(1−y);
    return s;
```

This example differs from `product` by the use of the reciprocal value of y. The assignments `y := 1/y0;` and `y := y/(1−y);` are translated to the polynomials $y'y_0 - 1$ and $y'(1-y) - y$ respectively. We use the generic template of degree 3 over the variables $V = \{x_0, y_0, y, s\}$.

$$\tilde{\alpha}(\Psi) = \sqrt{2}(x_0 y - x_0 + x_0 y_0 y - ys)$$
$$\tilde{\alpha}(\Phi) = y - 1$$

```
procedure product2S(x0, y0):
    s := x0;
    y := 1/y0
    while (y ≠ 1):
        s := c0x0 + c1y0 + c2y + c3s + c4;
        y := y/(1−y);
    assert(s = x0y0);
    return s;
```

We use the generic template of degree 3 over the variables $V = \{x_0, y_0, y, s\}$.

$$\alpha(c_0) = 1, \quad \alpha(c_1) = \alpha(c_2) = 0, \quad \alpha(c_3) = 1, \quad \alpha(c_4) = 0$$
$$\tilde{\alpha}(\Psi) = x_0 - x_0 y - x_0 y_0 y + ys$$
$$\tilde{\alpha}(\Phi) = y - 1$$

```
procedure gcd_lcm(x1, x2):
    y1 := x1;
    y2 := x2;
    y3 := x2;
    y4 := 0;
    while (y1 ≠ y2):
        if (y1 > y2):
            y1 := y1 − y2;
            y4 := y4 + y3;
        else:
            y2 := y2 − y1;
            y3 := y3 + y4;
    assert(y1(y3 + y4) − x1x2);
    return (y1, y3 + y4);
```

The inequality $y_1 > y_2$ cannot be translated into a polynomial lasso program syntax and is thus omitted. We use the generic template of degree 2 over the variables $V = \{x_1, x_2, y_1, y_2, y_3, y_4\}$.

$$\tilde{\alpha}(\Psi) = x_1x_2 - y_1y_3 - y_2y_4$$
$$\tilde{\alpha}(\Phi_1) = \tilde{\alpha}(\Phi_2) = y_1 - y_2$$

```
procedure gcd_lcmS(x1, x2):
    y1 := x1;
    y2 := x2;
    y3 := x2;
    y4 := 0;
    while (y1 ≠ y2):
        if (y1 > y2):
            y4 := c0y1 + c1y2 + c2y3 + c3y4 + c4;
            y1 := y1 − y2;
        else:
            y3 := c5y1 + c6y2 + c7y3 + c8y4 + c9;
            y2 := y2 − y1;
    assert(y1(y3 + y4) − x1x2);
    return (y1, y3 + y4);
```

The inequality $y_1 > y_2$ cannot be translated into a polynomial lasso program syntax and is thus omitted. $C = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\}$; we use the generic template of degree 2 over the variables $V = \{x_1, x_2, y_1, y_2, y_3, y_4\}$.

$$\alpha(c_0) = \alpha(c_1) = 0, \quad \alpha(c_2) = \alpha(c_3) = 1, \quad \alpha(c_4) = 0,$$
$$\alpha(c_5) = \alpha(c_6) = 0, \quad \alpha(c_7) = \alpha(c_8) = 1, \quad \alpha(c_9) = 0,$$
$$\tilde{\alpha}(\Psi) = x_1x_2 - y_1y_3 - y_2y_4$$
$$\tilde{\alpha}(\Phi_1) = \tilde{\alpha}(\Phi_2) = y_1 - y_2$$

```
procedure div_mod(a, d):
    q := 0;
    r := a;
    while (r ≥ d):
        r := r − d;
        q := q + 1;
    return (q, r);
```

The while-condition $r \geq d$ is translated to *true*. We use the generic template of degree 2 over the variables $V = \{a, d, q, r\}$.

$$\tilde{\alpha}(\Psi) = \frac{1}{2}(r + qd - a)$$
$$\tilde{\alpha}(\Phi) = 1$$

```
procedure div_modS(a, d):
    q := 0;
    r := a;
    while (r ≥ d):
        r := c0 a + c1 d + c2 q + c3 r + c4;
        q := c5 a + c6 d + c7 q + c8 r_old + c9;
    return (q, r);
```

Test cases:

- `div_modS(4, 3) == (1, 1)` (1 loop iteration)
- `div_modS(5, 2) == (2, 1)` (2 loop iterations)
- `div_modS(1, 1) == (1, 0)` (1 loop iteration)
- `div_modS(15, 6) == (2, 3)` (2 loop iterations)
- `div_modS(17, 17) == (1, 0)` (1 loop iterations)

The while-condition $r \geq d$ is translated to *true*. The synthesis relies purely on test cases for correctness and termination as there are no exit or post condition supplied. $C = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9\}$; we use the generic template of degree 2 over the variables $V = \{a, d, q, r\}$.

$$\alpha(c_0) = 0, \quad \alpha(c_1) = -1, \quad \alpha(c_2) = 0, \quad \alpha(c_3) = 1, \quad \alpha(c_4) = 0,$$
$$\alpha(c_5) = 0, \quad \alpha(c_6) = 0, \quad \alpha(c_7) = 1, \quad \alpha(c_8) = 0, \quad \alpha(c_9) = 1$$
$$\tilde{\alpha}(\Psi) = r + dq - a$$
$$\tilde{\alpha}(\Phi) = 1$$

```
procedure root2(n):
    p := 0;
```

```
    q := 1;
    r := n;
    while (q ≤ n):
        q := 4q;
    while (q ≠ 1):
        q := q/4;
        h := p + q;
        p := p/2;
        if (r ≥ h):
            p := p + q;
            r := r − h;
    assert(n = p^2 + r);
    return (p, r);
```

The first while loop is translated to an arbitrary assignment to q, the `if`-condition is omitted. We use the generic template of degree 2 over the variables $V = \{p, q, r, n\}$.

$$\tilde{\alpha}(\Psi) = \frac{1}{8}(nq - qr - p^2)$$
$$\tilde{\alpha}(\Phi_1) = \tilde{\alpha}(\Phi_2) = \frac{1}{4}(q - 1)$$

```
procedure root2S(n):
    p := c_0 n + c_1;
    q := 1;
    r := c_2 n + c_3;
    while (q ≤ n):
        q := 4q;
    while (q ≠ 1):
        q := q/4;
        h := p + q;
        p := p/2;
        if (r ≥ h):
            p := p + q;
            r := c_4 r + c_5 p_old + c_6 q_old + c_7 n + c_8;
    assert(n = p^2 + r);
    return (p, r);
```

$C = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$. The first `while` loop is translated to an arbitrary assignment to q, the `if`-condition is omitted. This example has a parameterized program stem. We use the generic template of degree 2 over the variables

$V = \{p, q, r, n\}$.

$$\alpha(c_0) = \alpha(c_1) = 0, \quad \alpha(c_2) = 1, \alpha(c_3) = 0,$$
$$\alpha(c_4) = 1, \quad \alpha(c_5) = -1, \quad \alpha(c_6) = \frac{1}{4}, \quad \alpha(c_7) = \alpha(c_8) = 0$$
$$\tilde{\alpha}(\Psi) = nq - qr - p^2$$
$$\tilde{\alpha}(\Phi_1) = \tilde{\alpha}(\Phi_2) = \frac{1}{4}(q - 1)$$

```
procedure squareS(n):
    a := n;
    b := c0 n + c1;
    while (a ≠ 0):
        b := c2 a + c3 b + c4 n + c5;
        a := a − 1;
    assert(b = n^2);
    return b;
```

$C = \{c_0, c_1, c_2, c_3, c_4, c_5\}$. This is an example for a synthesis from terminating lasso program skeleton. The lasso is constructed from a complexity guess (n steps to termination) and an auxiliary variable b (see also the program `cubeS`). We use the generic template of degree 2 over the variables $V = \{a, b, n\}$.

$$\alpha(c_0) = -\frac{1}{2}, \quad \alpha(c_1) = 0,$$
$$\alpha(c_2) = 2, \quad \alpha(c_3) = 1, \quad \alpha(c_4) = 0, \quad \alpha(c_5) = -\frac{1}{2}$$
$$\tilde{\alpha}(\Psi) = b - n^2 + \frac{1}{2}a + a^2$$
$$\tilde{\alpha}(\Phi) = a$$

```
procedure cubeS(n):
    a := n;
    b := c0 n + c1;
    c := c2 n + c3;
    while (a ≠ 0):
        b := c4 a + c5 b + c6 c + c7 n + c8;
        c := c9 a + c10 b_old + c11 c + c12 n + c13;
        a := a − 1;
    assert(b = n^3);
    return b;
```

$C = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}, c_{13}\}$. This is an example for a synthesis from terminating lasso program skeleton. The lasso is constructed from a complexity guess (n steps to termination) and an auxiliary variables b and c

(see also the program `squareS`). We use the generic template of degree 3 over the variables $V = \{a, b, c, n\}$.

$$\alpha(c_0) = -\sqrt{8} + \frac{1}{2}, \quad \alpha(c_1) = 0, \quad \alpha(c_2) = \frac{1}{2}(\sqrt{2} - 1), \quad \alpha(c_3) = \sqrt{2},$$

$$\alpha(c_4) = 0, \quad \alpha(c_5) = 1, \quad \alpha(c_6) = 1, \quad \alpha(c_7) = \frac{1}{\sqrt{2}} + 1, \quad \alpha(c_8) = 0,$$

$$\alpha(c_9) = \frac{3}{2}, \quad \alpha(c_{10}) = 0, \quad \alpha(c_{11}) = 1, \quad \alpha(c_{12}) = 1, \quad \alpha(c_{13}) = -\sqrt{8}$$

$$\tilde{\alpha}(\Psi) = b - n^3 + \frac{1}{2}a^2 n + \frac{1}{2}a^3 + ac + \left(\sqrt{2} - \frac{1}{2}\right)a + \frac{1}{2}\left(\sqrt{2} + 1\right)an - \sqrt{2}a^2$$

$$\tilde{\alpha}(\Phi) = a$$