Available online at www.sciencedirect.com
ScienceDirect
Science of Computer Programming 64 (2007) 76–96
Science of Computer Programming
www.elsevier.com/locate/scico

Polynomial approximations of the relational semantics of imperative programs☆

Michael A. Colón

Center for High Assurance Computer Systems, Naval Research Laboratory, Washington, DC, USA

Received 31 January 2005; received in revised form 15 October 2005; accepted 15 March 2006
Available online 23 October 2006

Abstract

We present a static analysis that approximates the relational semantics of imperative programs by systems of low-degree polynomial equalities. Our method is based on Abstract Interpretation in a lattice of polynomial pseudo ideals — finite-dimensional vector spaces of degree-bounded polynomials that are closed under degree-bounded products. For a fixed degree bound, the sizes of bases of pseudo ideals and the lengths of chains in the lattice of pseudo ideals are bounded by polynomials in the number of program variables. Despite the approximate nature of our analysis, for several programs taken from the literature on non-linear polynomial invariant generation our method produces results that are as precise as those produced by methods based on polynomial ideals and Gröbner bases.

Keywords: Program analysis; Relational semantics; Abstract Interpretation; Polynomial invariants; Non-linear invariants; Polynomial ideals

1. Introduction

The relational semantics of a program characterize its input–output behavior as a relation on states [23,22]. A pair of states is included in the relation if the program, when started in the first state, can halt in the second. The relational semantics are often expressed by a formula in the variables X, denoting the values of the program variables in the initial state, and X', denoting the values in the final state. Such logical characterizations of the relational semantics are often easier to comprehend than the program itself.

Consider the program presented in Fig. 1(a), which computes the square of a non-negative integer x by summing the first x odd numbers. The relational semantics of this program are approximated by the relation

$$[\![x' = x \ \wedge \ y' = x \ \wedge \ z' = x^2]\!],$$

where the variables x, y, and z denote the initial values of the corresponding program variables, while the variables x', y', and z' denote their final values. Since the program never terminates when x is negative initially, this relation is but an approximation of the relational semantics of the program. Nevertheless, the approximation is sound, and for applications in which only the terminating behavior of the program is relevant, such an approximation proves quite

☆ This research was supported by the Office of Naval Research.
E-mail address: colon@itd.nrl.navy.mil.

0167-6423/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2006.03.004

```
var x, y, z : integer;
ℓ1 : ⟨y, z⟩ := ⟨0, 0⟩;
ℓ2 : while y ≠ x do
        ⟨y, z⟩ := ⟨y + 1, z + 2y + 1⟩;
ℓ3 : halt
```

(a)

```
var x, y, z : integer;
ℓ1 : ⟨y, z⟩ := ⟨x, x^2⟩;
ℓ2 : halt
```

(b)

Fig. 1. Computing the square.

valuable. A program verification system might view it as an abstraction of the program suitable for compositional reasoning when establishing partial correctness. A re-engineering system might annotate the program with the relation to document the program's behavior. A compiler, noting that the relation defines a function, might produce the more efficient – and terminating – version shown in Fig. 1(b).

The program of Fig. 1(a) contains only linear assignments, yet any approximation of its relational semantics by a system of linear equalities fails to capture the essence of the program — that it computes the square of x. To adequately describe the input–output behavior of this program, we must consider systems of quadratic polynomial equalities. The expressiveness offered by non-linear polynomial equalities allows for more precise approximations of program behavior. In the presence of loops, even programs with purely linear assignments and conditions can have input–output relations that cannot be adequately approximated linearly.

The relational semantics can be approximated directly by Abstract Interpretation [5,4]. Alternatively, traditional invariant generation methods [12,30,16,6] can be applied by augmenting the program with auxiliary variables that preserve the initial values of program variables and approximating the set of reachable states. This reduction, however, may cause some analyses to become infeasible, i.e., those with high complexity in the number of program variables. For example, the set of extreme points and rays of a polyhedron may exceed the available memory when computing convex hulls in linear relation analysis [8,14]. When approximating the relational semantics of a program, we must be attentive to the complexity of our methods in terms of the number of variables, since there are twice as many — representing the initial and final values.

We present a static analysis that approximates the relational semantics of imperative programs by systems of low-degree polynomial equalities. Our method is based on Abstract Interpretation. Rather than interpret the program in the concrete domain of binary relations on states, we interpret it in an abstract domain of *polynomial pseudo ideals* — vector spaces of degree-bounded polynomials that are closed under degree-bounded products.

The principal advantage of our method is that it generates useful low-degree approximations of the relational semantics with reasonable efficiency. For a fixed degree bound, the number of polynomials in a minimal basis of a pseudo ideal and the time required to compute operators on pseudo ideals are bounded by polynomials in the number of program variables. Furthermore, the lengths of chains in the lattice of pseudo ideals are bounded by a polynomial in the number of program variables. As a result, while our choice of abstract domain leads to some loss of precision in the analysis, there is no additional loss of precision due to the introduction of a widening to extrapolate the limits of infinite chains. Another advantage of our approach is that, by a suitable choice of representation, it is easily implemented using well-known algorithms from linear algebra.

The primary disadvantage of our method is that both the space and time complexity are exponential in the bound d on the degrees of polynomials. For programs with many variables, our method is feasible only for small values of d. In addition, the degree bound must be chosen *a priori*. Of course, it is possible to repeat the analysis for increasing values of d until it yields sufficiently accurate results or becomes infeasible. Finally, the representation of pseudo ideals we have adopted, while simplifying the implementation of the analysis, is often highly redundant. As a result, our method often exhibits its polynomial worst-case behavior.

We have implemented our method and have applied it to several programs taken from the literature on non-linear polynomial invariant generation [28,24,27]. Despite the approximate nature of program analysis using pseudo ideals, for these programs our method produces results as precise as those produced by more heavyweight methods based on polynomial ideals and Gröbner bases.

This article is organized as follows: In Section 2, we cover preliminary material, including transition systems, lattices, vector spaces, polynomial ideals, and algebraic transition systems. Our method is developed in Section 3.

We first present a brief introduction to Abstract Interpretation. Then, to motivate our choice of abstract domain, we demonstrate the difficulties that arise in the domains of polynomial ideals and *polynomial spaces* — vector spaces of degree-bounded polynomials. We then present our approach, describe the representations and algorithms used in its implementation, and prove it sound. In Section 4, we demonstrate the utility of our method on a number of small but subtle programs. We address related work in Section 5 and conclude with Section 6.

2. Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a finite set of *variables*. A *state* $\sigma : X \to \mathbb{Q}$ maps each variable to a rational. The set of all states is denoted by Σ. A *state formula* φ is a first-order expression whose free variables belong to X. A state σ *satisfies* φ, written $\sigma \models \varphi$, precisely when φ holds in the model that interprets the variables as in σ.

A *(binary) relation* ρ on Σ is a subset of $\Sigma \times \Sigma$. Each pair $\langle \sigma, \sigma' \rangle$ of ρ consists of a *prestate* σ and a *poststate* σ'. The *composition* $\rho_1 \circ \rho_2$ of two relations is the set of pairs $\langle \sigma, \sigma' \rangle$ such that $\langle \sigma, \bar{\sigma} \rangle \in \rho_1$ and $\langle \bar{\sigma}, \sigma' \rangle \in \rho_2$ for some *intermediate state* $\bar{\sigma}$. The set of all relations on Σ is denoted by $\mathcal{R}$. Let $X' = \{x'_1, \ldots, x'_n\}$ be a set of *primed variables* corresponding to the variables in X. A *relation formula* ψ is a first-order expression whose free variables belong to $X \cup X'$. A pair $\langle \sigma, \sigma' \rangle$ *satisfies* ψ, written $\langle \sigma, \sigma' \rangle \models \psi$, if ψ holds in the model that interprets X as in the prestate and X' as in the poststate. A relation ρ *satisfies* ψ if $\langle \sigma, \sigma' \rangle \models \psi$ for every pair $\langle \sigma, \sigma' \rangle$ of ρ. The largest relation satisfying the formula ψ is denoted by $[\![\psi]\!]$.

2.1. Transition systems

We model imperative programs by transition systems. A *transition system* $\mathcal{P} = \langle X, L, \mathcal{T}, \ell_{\text{init}}, \ell_{\text{fin}} \rangle$ consists of a finite set of variables X, a finite set of *locations* L, a finite set of *transitions* $\mathcal{T}$, an *initial location* $\ell_{\text{init}} \in L$, and a *final location* $\ell_{\text{fin}} \in L$. A transition $\tau \in \mathcal{T}$ is a tuple $\langle \ell, m, \rho \rangle$ consisting of a *prelocation* $\ell \in L$, a *postlocation* $m \in L$, and a relation ρ on Σ, known as the *transition relation*. Transitions represent single execution steps, and locations serve to sequence transitions. We assume there are no transitions to the initial location or from the final location.

A *trace* $\langle \ell_0, \sigma_0 \rangle \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_n} \langle \ell_n, \sigma_n \rangle$ is a finite sequence of interleaved *configurations*, i.e., locations paired with states, and transitions that begins with the initial location ℓ_{init} and such that, for each $i \in \{1, \ldots, n\}$, τ_i is $\langle \ell_{i-1}, \ell_i, \rho_i \rangle$ for some relation ρ_i with $\langle \sigma_{i-1}, \sigma_i \rangle \in \rho_i$. A state formula φ is *invariant* at location ℓ if $\sigma \models \varphi$ whenever $\langle \ell, \sigma \rangle$ appears in a trace. That is, φ is invariant at ℓ if φ holds whenever ℓ is reached. A *computation* is a trace that ends with the final location ℓ_{fin}.

Example 1. The program in Fig. 1(a) can be modeled by the transition system $\mathcal{P} = \langle X, L, \mathcal{T}, \ell_1, \ell_3 \rangle$, where $X = \{x, y, z\}$, $L = \{\ell_1, \ell_2, \ell_3\}$, and $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, with

$$\tau_1 = \langle \ell_1, \ell_2, [\![x' = x \ \wedge \ y' = 0 \ \wedge \ z' = 0]\!] \rangle,$$
$$\tau_2 = \langle \ell_2, \ell_2, [\![y \neq x \ \wedge \ x' = x \ \wedge \ y' = y + 1 \ \wedge \ z' = z + 2y + 1]\!] \rangle, \text{ and}$$
$$\tau_3 = \langle \ell_2, \ell_3, [\![y = x \ \wedge \ x' = x \ \wedge \ y' = y \ \wedge \ z' = z]\!] \rangle.$$

Transition τ_1 models the initial assignment statement, transition τ_2 models the loop body, and transition τ_3 models the loop exit. In modeling assignment statements, the preservation of unassigned variables is made explicit. Also, the loop condition is modeled twice — once, in positive form, for the loop body, and once, in negative form, for the loop exit.

A *path* $\pi = \ell_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_n} \ell_n$ from location ℓ_0 to location ℓ_n of a transition system is a sequence of interleaved locations and transitions such that, for each $i \in \{1, \ldots, n\}$, τ_i is $\langle \ell_{i-1}, \ell_i, \rho_i \rangle$ for some relation ρ_i. A path from ℓ_{init} to ℓ_{fin} is said to be *proper*. Every computation induces a proper path. The *path relation* $[\![\pi]\!]$ of path $\pi = \ell_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_n} \ell_n$ is the composition $\rho_1 \circ \cdots \circ \rho_n$ of the transition relations along π. The path relation $[\![\Pi]\!]$ of a set of paths is the union of the path relations of the paths of Π.

Definition 2 (*Relational Semantics*). The *relational semantics* $[\![\mathcal{P}]\!]$ of a transition system $\mathcal{P}$ is the union of the path relations of its proper paths.

The relational semantics are also known as the *input–output relation*.

A *cycle* is a finite path $\ell_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_n} \ell_0$ that begins and ends with the same location. A transition system is *acyclic* if it contains no cycles. Since acyclic systems contain only finitely many paths, their relational semantics can be computed as finite unions of path relations. The difficulty in computing the relational semantics lies with transition systems containing cycles.

Let $\mathcal{P}$ be a transition system containing cycles, and suppose C is a *cutset* for $\mathcal{P}$ — a subset of locations with the property that every cycle of $\mathcal{P}$ contains at least one *cutpoint* of C. Call a path π *simple* relative to C if π never passes through a cutpoint of C. That is, π can begin and end with a cutpoint, but cannot contain a cutpoint as an intermediate location. Let $\mathrm{Simp}_C(l, m)$ denote the paths from l to m that are simple relative to C.

Suppose the cutset C consists of a single cutpoint c. Then the proper paths of $\mathcal{P}$ can be partitioned into those that never pass through c and those that move from ℓ_{init} to c, cycle back to c a finite number of times, then move from c to ℓ_{fin}. Based on this observation, we define the following four relations:

$$\rho_{\text{simp}} = [\![\mathrm{Simp}_C(\ell_{\text{init}}, \ell_{\text{fin}})]\!] \qquad \rho_{\text{cyc}} = [\![\mathrm{Simp}_C(c, c)]\!]$$
$$\rho_{\text{init}} = [\![\mathrm{Simp}_C(\ell_{\text{init}}, c)]\!] \qquad \rho_{\text{fin}} = [\![\mathrm{Simp}_C(c, \ell_{\text{fin}})]\!].$$

Each of these relations is a finite union of path relations. The relational semantics of $\mathcal{P}$ can be computed as follows:

$$[\![\mathcal{P}]\!] = \rho_{\text{simp}} \cup (\rho_* \circ \rho_{\text{fin}}),$$

where ρ_* is the least fixed point of the *semantic function* $f_{\mathcal{P}}$ associated with $\mathcal{P}$:

$$f_{\mathcal{P}}(\rho) = \rho_{\text{init}} \cup (\rho \circ \rho_{\text{cyc}}).$$

The fixed point ρ_* gives the relational semantics to the cutpoint c.

When C contains multiple cutpoints, we compute the relational semantics to all cutpoints simultaneously by defining the semantic function $f_{\mathcal{P}}$ on maps from C to $\mathcal{R}$. Let

$$f_{\mathcal{P}}(\eta)(c) = [\![\mathrm{Simp}_C(\ell_{\text{init}}, c)]\!] \cup \left(\bigcup_{b \in C} \eta(b) \circ [\![\mathrm{Simp}_C(b, c)]\!] \right),$$

for all maps η from C to $\mathcal{R}$ and cutpoints $c \in C$. The least fixed point η_* of $f_{\mathcal{P}}$, then, is the map that yields, for every cutpoint c, the relational semantics of all paths from ℓ_{init} to c. The relational semantics of $\mathcal{P}$ are then defined in terms of η_*:

$$[\![\mathcal{P}]\!] = [\![\mathrm{Simp}_C(\ell_{\text{init}}, \ell_{\text{fin}})]\!] \cup \left(\bigcup_{c \in C} \eta_*(c) \circ [\![\mathrm{Simp}_C(c, \ell_{\text{fin}})]\!] \right).$$

This approach to defining the relational semantics at cutpoints is similar to the cutpoint-based approach to establishing inductive assertions of flow chart programs [19].

Example 3. All cycles of the transition system $\mathcal{P}$ of Example 1 are cut by the single cutpoint ℓ_2. The simple paths relative to this cutpoint consist of single transitions and yield the following relations:

$$\begin{aligned}
\rho_{\text{simp}} &= [\![\text{false}]\!] = \emptyset, \\
\rho_{\text{init}} &= [\![x' = x \ \wedge \ y' = 0 \ \wedge \ z' = 0]\!], \\
\rho_{\text{cyc}} &= [\![y \neq x \ \wedge \ x' = x \ \wedge \ y' = y + 1 \ \wedge \ z' = z + 2y + 1]\!], \text{ and} \\
\rho_{\text{fin}} &= [\![y = x \ \wedge \ x' = x \ \wedge \ y' = y \ \wedge \ z' = z]\!].
\end{aligned}$$

To compute the relational semantics, we first compute the least fixed point ρ_* of the semantic function $f_{\mathcal{P}}$, where

$$f_{\mathcal{P}}(\rho) = \rho_{\text{init}} \cup (\rho \circ \rho_{\text{cyc}}).$$

Consider the sequence of relations $\rho_1 \subseteq \rho_2 \subseteq \rho_3 \ldots$, where

$$\begin{aligned}
\rho_1 &= f_{\mathcal{P}}(\emptyset) \\
&= [\![x' = x \ \wedge \ y' = 0 \ \wedge \ z' = 0]\!] \\
\rho_2 &= f_{\mathcal{P}}(\rho_1) \\
&= [\![x' = x \ \wedge \ ((y' = 0 \ \wedge \ z' = 0) \ \vee \ (x \neq 0 \ \wedge \ y' = 1 \ \wedge \ z' = 1))]\!] \\
\rho_3 &= f_{\mathcal{P}}(\rho_2) \\
&= \left[\!\!\left[x' = x \ \wedge \ \begin{pmatrix} (y' = 0 \ \wedge \ z' = 0) \ \vee \ (x \neq 0 \ \wedge \ y' = 1 \ \wedge \ z' = 1) \ \vee \\ (x \neq 0 \ \wedge \ x \neq 1 \ \wedge \ y' = 2 \ \wedge \ z' = 4) \end{pmatrix} \right]\!\!\right] \\
\rho_4 &= f_{\mathcal{P}}(\rho_3) \\
&= \left[\!\!\left[x' = x \ \wedge \ \begin{pmatrix} (y' = 0 \ \wedge \ z' = 0) \ \vee \ (x \neq 0 \ \wedge \ y' = 1 \ \wedge \ z' = 1) \ \vee \\ (x \neq 0 \ \wedge \ x \neq 1 \ \wedge \ y' = 2 \ \wedge \ z' = 4) \ \vee \\ (x \neq 0 \ \wedge \ x \neq 1 \ \wedge \ x \neq 2 \ \wedge \ y' = 3 \ \wedge \ z' = 9) \end{pmatrix} \right]\!\!\right]
\end{aligned}$$

Generalizing the growing disjunction, the least fixed point is

$$\rho_* = [\![x' = x \ \wedge \ x \notin \{0, \ldots, y' - 1\} \ \wedge \ y' \in \mathbb{N} \ \wedge \ z' = (y')^2]\!].$$

Given this fixed point, we then compute the relational semantics as

$$\begin{aligned}
[\![\mathcal{P}]\!] &= \rho_{\text{simp}} \cup (\rho_* \circ \rho_{\text{fin}}) \\
&= [\![x \in \mathbb{N} \ \wedge \ x' = x \ \wedge \ y' = x \ \wedge \ z' = x^2]\!].
\end{aligned}$$

Thus, provided x is a non-negative integer, the final values of x and y equal the initial value of x, and the final value of z equals the square of the initial value of x. Unlike the approximation presented in Section 1, these semantics show that $\mathcal{P}$ computes the square of x only when x is a non-negative integer.

2.2. Lattices

A *lattice* $\langle L, \sqsubseteq, \sqcap, \sqcup \rangle$ is a non-empty partially ordered set in which every pair of *points* $l_1, l_2 \in L$ has a *greatest lower bound* $l_1 \sqcap l_2$ (*meet*) and a *least upper bound* $l_1 \sqcup l_2$ (*join*) in L. We will often denote the lattice $\langle L, \sqsubseteq, \sqcap, \sqcup \rangle$ by its *carrier* L. A lattice L is *complete* if every subset of L has a greatest lower bound and a least upper bound in L. Every finite lattice is complete, and every complete lattice possesses a *least element* $\bot$ and a *greatest element* $\top$. The *dual* of the lattice $\langle L, \sqsubseteq, \sqcap, \sqcup \rangle$ is the lattice $\langle L, \sqsupseteq, \sqcup, \sqcap \rangle$ [10].

An *ascending chain* $l_1 \sqsubset l_2 \sqsubset \ldots \sqsubset l_k \sqsubset \ldots$ of the lattice $\langle L, \sqsubseteq, \sqcap, \sqcup \rangle$ is a strictly increasing sequence of points of L, where $\sqsubset$ denotes the irreflexive restriction of $\sqsubseteq$. Dually, a *descending chain* $l_1 \sqsupset l_2 \sqsupset \ldots \sqsupset l_k \sqsupset \ldots$ is a strictly decreasing sequence of points. A lattice satisfies the *ascending (descending) chain condition* if it contains no infinite ascending (descending) chains. Every lattice satisfying both chain conditions is complete.

A function $f : L \to M$ is *monotone* from $\langle L, \sqsubseteq_L, \sqcap_L, \sqcup_L \rangle$ to $\langle M, \sqsubseteq_M, \sqcap_M, \sqcup_M \rangle$ iff $l_1 \sqsubseteq_L l_2$ implies $f(l_1) \sqsubseteq_M f(l_2)$ for all $l_1, l_2 \in L$, and monotone on $\langle L, \sqsubseteq, \sqcap, \sqcup \rangle$ if monotone from the lattice to itself. A *fixed point* of a monotone function f on L is any point $l \in L$ satisfying $f(l) = l$. Every monotone function on a complete lattice possesses a *least fixed point* $\text{lfp}(f)$. In any lattice satisfying the ascending chain condition, the least fixed point of a monotone function f can be computed iteratively:

$$\text{lfp}(f) = \bigsqcup_{i \geq 0} f^i(\bot)$$

After finitely many iterations, the computation will terminate.

A *Galois connection* $\langle \alpha, \gamma \rangle$ from a complete lattice L to a complete lattice M is a pair of monotone functions $\alpha : L \to M$ and $\gamma : M \to L$ satisfying $l \sqsubseteq_L \gamma(\alpha(l))$ for all $l \in L$ and $\alpha(\gamma(m)) \sqsubseteq_M m$ for all $m \in M$. When $\langle \alpha, \gamma \rangle$ is a Galois connection from L to M, $\langle \alpha, \gamma \rangle$ is also an *adjunction*, i.e., $\alpha(l) \sqsubseteq_M m$ iff $l \sqsubseteq_L \gamma(m)$ for all $l \in L$ and $m \in M$. A Galois connection from L to M enables the least fixed point of a monotone function f on L to be approximated by the least fixed point of a suitably-chosen function g on M [5,7,25].

Theorem 4. *Let $\langle \alpha, \gamma \rangle$ be a Galois connection from the lattice $\langle L, \sqsubseteq_L, \sqcap_L, \sqcup_L \rangle$ to the lattice $\langle M, \sqsubseteq_M, \sqcap_M, \sqcup_M \rangle$, where both lattices are complete. Let f and g be monotone functions on L and M, respectively, satisfying*

$$\alpha(f(\gamma(m))) \sqsubseteq_M g(m),$$

for all $m \in M$. Then

$$\mathrm{lfp}(f) \sqsubseteq_L \gamma(\mathrm{lfp}(g)) \quad \textit{and} \quad \alpha(\mathrm{lfp}(f)) \sqsubseteq_M \mathrm{lfp}(g).$$

Proof. See [25].

The approximation of f *induced* by the Galois connection $\langle \alpha, \gamma \rangle$ from L to M is defined as $g(m) = \alpha(f(\gamma(m)))$, for all $m \in M$.

2.3. Vector spaces

A *(rational) vector* v in $X = \{x_1, \ldots, x_n\}$ is a map from X to $\mathbb{Q}$. The set of all vectors is denoted by $\mathbb{Q}^X$. The *sum* $v_1 + v_2$ of two vectors satisfies $(v_1 + v_2)(x) = v_1(x) + v_2(x)$ for all $x \in X$. The *scalar product* λv of a rational λ and a vector v satisfies $(\lambda v)(x) = \lambda v(x)$ for all $x \in X$.

Definition 5 (*Vector Space*). A set of vectors S forms a *vector space* if

- $0 \in S$,
- $v_1 + v_2 \in S$ for all $v_1, v_2 \in S$, and
- $\lambda v \in S$ for all $v \in S$ and $\lambda \in \mathbb{Q}$.

The least space is the trivial space $\{0\}$ consisting of the *zero vector*. The greatest space is $\mathbb{Q}^X$. Vector spaces are closed under intersection, but not union. Instead, the *sum* $S_1 + S_2$ is the least space containing S_1 and S_2, where $S_1 + S_2$ denotes the set of vectors of the form $v_1 + v_2$, with $v_1 \in S_1$ and $v_2 \in S_2$. The family $\mathcal{S}$ of vector spaces in X forms a complete lattice $\langle \mathcal{S}, \subseteq, \cap, + \rangle$.

The space $\mathcal{S}p(V)$ *generated by* a set of vectors V is the intersection of all spaces containing V. Equivalently, $\mathcal{S}p(V)$ is the set of *linear combinations* $\lambda_1 v_1 + \cdots + \lambda_k v_k$ of vectors of V with rational coefficients. A *basis* of a vector space S is any set of vectors that generates S. Every vector space in $\mathbb{Q}^X$ has a finite basis. A basis V of a vector space S is *minimal* if no proper subset of V generates S.[1]

All minimal bases of a vector space S have the same number of vectors, known as the *dimension* of S and denoted by $\dim(S)$. The space $\{0\}$ has dimension 0, while the space $\mathbb{Q}^X$ has dimension n, where $X = \{x_1, \ldots, x_n\}$. For any two vector spaces S_1 and S_2, if $S_1 \subseteq S_2$, then $\dim(S_1) \leq \dim(S_2)$. If $S_1 \subset S_2$, then $\dim(S_1) < \dim(S_2)$. As a result, the lattice of vector spaces $\langle \mathcal{S}, \subseteq, \cap, + \rangle$ satisfies both chain conditions. No ascending nor descending chain in this lattice contains more than $n + 1$ points, where $X = \{x_1, \ldots, x_n\}$ [17].

Let $\succ$ be a total order on the set of variables X. The *leading variable* $\mathrm{LV}(v)$ of a non-zero vector v is the greatest variable that is mapped to a non-zero value, i.e., $\mathrm{LV}(v) = \max_{\succ} \{x \in X \mid v(x) \neq 0\}$. A *standard basis* of the space S is a basis $V = \{v_1, \ldots, v_k\}$ of non-zero vectors satisfying (i) $v_i(\mathrm{LV}(v_i)) = 1$ for all $v_i \in V$ and (ii) $v_j(\mathrm{LV}(v_i)) = 0$ for all $v_i, v_j \in V$ with $i \neq j$. Standard bases provide canonical representations of vector spaces.

Most operations on vector spaces can be implemented using *Gauss–Jordan elimination* (or *reduction*) — an algorithm that converts any finite basis of a space S into a standard basis in polynomial time [29]. The space S_1 is included in the space S_2, where S_1 and S_2 are generated by standard bases V_1 and V_2, respectively, iff the standard basis of $\mathcal{S}p(V_1 \cup V_2)$ is V_2. The standard basis of the sum $S_1 + S_2$ can be computed by applying Gauss–Jordan reduction to the union of their bases. To compute the intersection of spaces S_1 and S_2, we apply Karr's algorithm [15] to compute a basis of $S_1 \cap S_2$, followed by Gauss–Jordan reduction to compute a standard basis. Although Karr presents his algorithm as computing the sum of two spaces represented by systems of linear equalities, the algorithm also computes the intersection of spaces represented by bases.

[1] The traditional definition of *basis* assumes that it is minimal. Our definition does not make this assumption to remain consistent with bases of polynomial ideals.

2.4. Polynomial ideals

A *monomial* m in $X = \{x_1, \ldots, x_n\}$ is a product of powers $x_1^{e_1} \cdots x_n^{e_n}$, where the exponents $e_1, \ldots, e_n$ are non-negative integers. A *(rational) polynomial* p in X is a linear combination $c_1 m_1 + \cdots + c_k m_k$ of monomials with rational coefficients. The ring of polynomials in X is denoted by $\mathbb{Q}[X]$.

The *degree* $\deg(m)$ of a monomial $m = x_1^{e_1} \cdots x_n^{e_n}$ is the sum $e_1 + \cdots + e_n$ of its exponents. The degree $\deg(p)$ of a non-zero polynomial is the maximal degree of its monomials with non-zero coefficients. The degree of the zero polynomial is $-\infty$. A polynomial is *linear* if its degree is one, *quadratic* if two, and *cubic* if three.

Definition 6 (*Polynomial Ideal*). A set I of polynomials forms an *ideal* if

- $0 \in I$,
- $p_1 + p_2 \in I$ for all $p_1, p_2 \in I$, and
- $qp \in I$ for all $p \in I$ and $q \in \mathbb{Q}[X]$.

The least ideal is the trivial ideal $\{0\}$, and the greatest is $\mathbb{Q}[X]$. Polynomial ideals are closed under intersection, but not union. As is the case with spaces, the sum $I_1 + I_2$ of two ideals is the least ideal that contains I_1 and I_2. The family $\mathcal{I}$ of ideals of polynomials in X forms a complete lattice $\langle \mathcal{I}, \subseteq, \cap, + \rangle$. This lattice satisfies the ascending chain condition, but not the descending chain condition.

The ideal $\mathcal{I}d(P)$ *generated by* a set of polynomials P is the intersection of all ideals containing P. Alternatively, $\mathcal{I}d(P)$ is the set of all linear combinations $q_1 p_1 + \cdots + q_k p_k$ of polynomials of P with polynomial coefficients. A *basis* of a polynomial ideal I is a set of polynomials that generates I. A basis P of the ideal I is *minimal* if no proper subset of P generates I. As is the case with vector spaces, every polynomial ideal has a finite basis. This result is known as the *Hilbert Basis Theorem*. However, unlike vector spaces, minimal bases of polynomial ideals need not have the same cardinality.

A *(reduced) Gröbner basis* is the analog for polynomial ideals of a standard basis for vector spaces. Gröbner bases provide canonical representations of polynomial ideals that are suitable for computing intersections and sums of ideals and deciding inclusion between ideals. Any finite basis of a non-trivial ideal can be transformed into a Gröbner basis using Buchberger's algorithm [9,1]. However, the size of a Gröbner basis can be exponential in the number of variables [31].

2.5. Algebraic transition systems

Consider the ring $\mathbb{Q}[X, X']$ of polynomials in X and X'. The polynomial p *vanishes* on the relation ρ if ρ satisfies the equality $p = 0$. The *(polynomial) theory* $\mathcal{T}h(\rho)$ is the set of polynomials in $\mathbb{Q}[X, X']$ that vanish on ρ. The relation $[\![P]\!]$ *defined* by a set of polynomials in $\mathbb{Q}[X, X']$ is the largest relation on which every polynomial of P vanishes. The relation ρ is *algebraic* if $\rho = [\![P]\!]$ for some set of polynomials P. The pair $\langle \mathcal{T}h, [\![\cdot]\!] \rangle$ is a Galois connection from the lattice $\langle \mathcal{R}, \subseteq, \cap, \cup \rangle$ of relations to the lattice $\langle \mathcal{I}, \supseteq, +, \cap \rangle$, the dual of the lattice of ideals of polynomials in $\mathbb{Q}[X, X']$.

Our method approximates the relational semantics of algebraic transition systems, where a transition system $\mathcal{P} = \langle X, L, \mathcal{T}, \ell_{\text{init}}, \ell_{\text{fin}} \rangle$ is *algebraic* if the transition relation of every transition $\tau_i \in \mathcal{T}$ is represented by a finite set of polynomials P_i in $\mathbb{Q}[X, X']$. The *degree* $\deg(\mathcal{P})$ of an algebraic transition system is the maximal degree of any polynomial appearing in the representation of a transition relation.

Imperative programs over variables taking on rational and integer values can be modeled by algebraic transition systems, albeit with some potential loss of precision. For example, an assignment with a polynomial right-hand side can be modeled precisely, while non-polynomial assignments are modeled as non-deterministic updates. A condition expressed as a conjunction of polynomial equalities can be modeled precisely, while others must be approximated. For example, a conditional statement involving an inequality can be modeled as a non-deterministic choice between its two branches.

Example 7. The program of Fig. 1(a) contains only polynomial assignments, but the condition guarding the loop is a disequality. The program can be modeled, with some loss of precision, by the algebraic transition system

$\mathcal{P} = \langle X, L, \mathcal{T}, \ell_1, \ell_3\rangle$, where $X = \{x, y, z\}$, $L = \{\ell_1, \ell_2, \ell_3\}$, and $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$, with

$$\tau_1 = \langle \ell_1, \ell_2, [\![x' = x \ \wedge\ y' = 0 \ \wedge\ z' = 0]\!]\rangle,$$
$$\tau_2 = \langle \ell_2, \ell_2, [\![x' = x \ \wedge\ y' = y + 1 \ \wedge\ z' = z + 2y + 1]\!]\rangle, \text{ and}$$
$$\tau_3 = \langle \ell_2, \ell_3, [\![y = x \ \wedge\ x' = x \ \wedge\ y' = y \ \wedge\ z' = z]\!]\rangle.$$

The transition relation of transition τ_2, which models the body of the loop, is imprecise. It does not model the disequality $y \neq x$. The loop exit, on the other hand, is modeled precisely since the negation of $y \neq x$ is a polynomial equality.

The imprecision caused by modeling imperative programs as algebraic transition systems can often be ameliorated through judicious use of invariants. For example, the assignment $x := x$ div 2 is not a polynomial assignment, where the div operator denotes integer division, i.e., x div $2 = \lfloor \frac{1}{2}x \rfloor$. Under normal circumstances, such an assignment must be modeled as a non-deterministic update of x. However, if x is guaranteed to be even, the assignment is equivalent to the polynomial assignment $x := \frac{1}{2}x$. Techniques such as congruence analysis [13] and numerical power analysis [20] can often be applied to derive invariants of the program that improve the accuracy of its algebraic model. As there are many such analyses that can improve the accuracy of the modeling, the exact process used to model an imperative program by an algebraic transition system lies outside the scope of this paper.

3. Approximating the relational semantics

We now present a static analysis that approximates the relational semantics of imperative programs by systems of low-degree polynomial equalities. Our method assumes that the program is modeled as an algebraic transition system and executes the transition system symbolically in the domain of polynomial pseudo ideals — vector spaces of degree-bounded polynomials that are closed under degree-bounded products.

To motivate our choice of abstract domain and illustrate its design, we develop the method in stages. We first present a summary of Abstract Interpretation — a framework for the design of program analyses based on symbolic execution in abstract domains. We then consider the candidate abstract domains of polynomial ideals and polynomial spaces, i.e., vector spaces of degree-bounded polynomials. In the first domain, the presence of infinite ascending chains leads to non-termination. In the second, the results obtained are often weak. Next, we consider the domain of polynomial pseudo ideals, which offers improved precision over polynomial spaces and satisfies the ascending chain condition. We then describe the representations and algorithms used by our approach and prove the method sound.

3.1. Abstract Interpretation

Abstract Interpretation [5] provides a framework for the design of computable approximations of the semantics of programs. In the traditional Galois connection approach, the *standard* (or *concrete*) *semantics* $[\![\mathcal{P}]\!]$ of a program are defined in terms of the least fixed point of a *concrete semantic function* $f_{\mathcal{P}}$ on a complete lattice of concrete semantic values, known as the *concrete domain*. The *abstract semantics* $[\![\mathcal{P}]\!]_{\mathcal{A}}$ are defined in terms of the least fixed point of an *abstract semantic function* $g_{\mathcal{P}}$ on a complete lattice $\mathcal{A}$ of abstract semantic values, known as the *abstract domain*. The concrete semantics are normally not computable. Instead, they provide an idealized semantics by which the soundness and precision of the abstract semantics can be gauged. The abstract semantics, on the other hand, are intended to be computable.

The key challenge in applying Abstract Interpretation is identifying a suitable abstract domain $\mathcal{A}$. First, to guarantee the soundness of the analysis, the concrete and abstract domains should be related by a Galois connection. Second, the points of $\mathcal{A}$ should be representable in finite space. Third, the meet and join operators of $\mathcal{A}$ and the abstract semantic function $g_{\mathcal{P}}$ should all be computable. Fourth, to detect convergence of iterative fixed point computations, the partial order on $\mathcal{A}$ should be decidable. Finally, to ensure termination of the analysis, the abstract domain should either satisfy the ascending chain condition or possess a *widening* to extrapolate the limits of infinite chains [7].

For our method, the concrete semantics are the relational semantics and are defined in the concrete domain $\langle \mathcal{R}, \subseteq, \cap, \cup\rangle$ of relations on states. One potential abstract domain is the dual $\langle \mathcal{I}, \supseteq, +, \cap\rangle$ of the lattice of ideals of polynomials in $\mathbb{Q}[X, X']$. This domain is suggested by the Galois connection $\langle \mathcal{I}h, [\![\cdot]\!]\rangle$ from $\langle \mathcal{R}, \subseteq, \cap, \cup\rangle$ to $\langle \mathcal{I}, \supseteq, +, \cap\rangle$. Using Gröbner bases and Buchberger's algorithm, ideals are representable in finite space, intersections

and sums of ideals are computable, and inclusion between ideals is decidable. However, the worst-case exponential space complexity of Gröbner basis methods will cause this approach to exhaust available memory for some programs. More importantly, the infinite ascending chains of the abstract domain will frequently lead to non-termination.

Example 8. Consider the transition system $\mathcal{P}$ of Example 7. Let $g_{\mathcal{P}}$ be the abstract semantic function induced by $\langle \mathcal{T}h, [\![\cdot]\!] \rangle$. That is, $g_{\mathcal{P}}(I) = \mathcal{T}h(f_{\mathcal{P}}([\![I]\!]))$, for any ideal I of $\mathbb{Q}[X, X']$.

Iterative computation of the least fixed point of $g_{\mathcal{P}}$ yields the infinite chain $I_1 \supset I_2 \supset I_3 \supset \cdots \supset I_k \supset \ldots$, where

$$
\begin{aligned}
I_1 &= g_{\mathcal{P}}(\mathbb{Q}[X, X']) \\
&= \mathcal{I}d(\{x' - x, y', z'\}), \\
I_2 &= g_{\mathcal{P}}(I_1) \\
&= \mathcal{I}d(\{x' - x, y'(y' - 1), z' - y'\}), \\
I_3 &= g_{\mathcal{P}}(I_2) \\
&= \mathcal{I}d(\{x' - x, y'(y' - 1)(y' - 2), z' - (y')^2\}), \text{ and} \\
I_k &= \mathcal{I}d\left(\left\{x' - x, \prod_{i=0}^{k-1}(y' - i), z' - (y')^2\right\}\right).
\end{aligned}
$$

Within three iterations, the analysis discovers that $x' = x$ and $z' = (y')^2$. However, the analysis never terminates. Instead, it visits an infinite chain that encodes disjunctions as products of polynomials. Note that $I_1 \supset I_2 \supset I_3 \supset \ldots$ is a descending chain of the lattice of polynomial ideals, but an ascending chain of the abstract domain, which is the dual of the lattice of ideals.

When faced with an abstract domain with infinite ascending chains, there are two options. We can devise a *widening* to extrapolate the limits of infinite chains. These extrapolated limits can be improved by use of a *narrowing* [7]. Alternatively, we can modify the domain to eliminate the infinite ascending chains. We adopt the latter strategy.

3.2. Polynomial spaces

By restricting the abstract domain to degree-bounded sets of polynomials, it is possible to eliminate all infinite ascending chains. The points of such a domain cannot be closed under polynomial products, and thus they are not polynomial ideals. They can, however, be closed under polynomial sums and scalar products.

Definition 9 (*Polynomial Space*). A set of polynomials S forms a *polynomial space of degree d* if

- $\deg(p) \leq d$ for all $p \in S$,
- $0 \in S$,
- $p_1 + p_2 \in S$ for all $p_1, p_2 \in S$, and
- $\lambda p \in S$ for all $p \in S$ and $\lambda \in \mathbb{Q}$.

A polynomial space of degree d is simply a vector space of polynomials, where the polynomials are treated as vectors indexed by monomials. The least such space is $\{0\}$, and the greatest is $\mathbb{Q}_d[X]$, the set of all polynomials in X of degree d or less. The family of polynomial spaces of degree d forms a complete lattice that satisfies both chain conditions.

In addition, for a fixed bound d, the sizes of minimal bases of polynomial spaces and the lengths of chains are bounded by polynomials in the number of variables. The number of monomials in $X = \{x_1, \ldots, x_n\}$ of degree d or less is $\left(\binom{n+1}{d}\right)$ — the number of multisets of size d chosen from $n + 1$ elements, where the additional element, i.e., 1, allows for monomials of degree less than d. Now, $\left(\binom{n+1}{d}\right) \leq (n + 1)^d$, which is polynomial in n for a fixed d.

Let $d \geq 1$ be a degree bound. For any relation ρ, the *theory* $\mathcal{T}h_d(\rho)$ of degree d is the set of polynomials of degree d or less that vanish on ρ. $\mathcal{T}h_d(\rho)$ is a polynomial space of degree d for any relation ρ. Furthermore, $\langle \mathcal{T}h_d, [\![\cdot]\!] \rangle$ is a Galois connection from $\langle \mathcal{R}, \subseteq, \cap, \cup \rangle$ to $\langle \mathcal{S}_d, \supseteq, +, \cap \rangle$, the dual of the lattice of spaces of $\mathbb{Q}_d[X, X']$. This Galois connection suggests an abstract domain for approximating the relational semantics. However, this domain often yields weak results.

Example 10. Consider the transition system $\mathcal{P}$ of Example 7. Let $g_{\mathcal{P}}$ be the abstract semantic function on quadratic spaces induced by $\langle \mathcal{T}h_2, [\![\cdot]\!] \rangle$. That is, $g_{\mathcal{P}}(S) = \mathcal{T}h_2(f_{\mathcal{P}}([\![S]\!]))$, for any quadratic space S of $\mathbb{Q}_2[X, X']$.

Iterative computation of the least fixed point of $g_{\mathcal{P}}$ yields the chain $S_1 \supset S_2 \supset S_3$, where

$$\begin{aligned} S_1 &= g_{\mathcal{P}}(\mathbb{Q}_2[X, X']) \\ &= \mathcal{S}p(\{x' - x, y', z'\}), \\ S_2 &= g_{\mathcal{P}}(S_1) \\ &= \mathcal{S}p(\{x' - x, y' - z'\}), \text{ and} \\ S_3 &= g_{\mathcal{P}}(S_2) \\ &= \mathcal{S}p(\{x' - x\}). \end{aligned}$$

Taking the least fixed point S_* to be S_3, we compute the abstract semantics as

$$\begin{aligned} [\![\mathcal{P}]\!]_{\mathcal{S}_2} &= \mathcal{T}h_2(\rho_{\text{simp}} \cup [\![S_*]\!] \circ \rho_{\text{fin}}) \\ &= Sp(\{x' - x, y' - x\}). \end{aligned}$$

In this abstract domain, the analysis terminates quickly, but the result is weak. It infers the equalities $x' = x$ and $y' = x$, but not $z' = x^2$.

3.3. Polynomial pseudo ideals

The weakness of the analysis in the domain of quadratic spaces is attributable to the fact that polynomial spaces are not closed under polynomial products. To improve the accuracy of the analysis, we move to the domain of polynomial spaces that are closed under degree-bounded products.

Definition 11 (*Polynomial Pseudo Ideal*). A set J of polynomials forms a *polynomial pseudo ideal of degree d* if

- $\deg(p) \leq d$ for all $p \in J$,
- $0 \in J$,
- $p_1 + p_2 \in J$ for all $p_1, p_2 \in J$, and
- $qp \in J$ for all $p \in J$ and $q \in \mathbb{Q}[X]$ with $\deg(qp) \leq d$.

The least pseudo ideal of degree d is $\{0\}$, and the greatest is $\mathbb{Q}_d[X]$. While pseudo ideals are closed under intersection, unlike polynomial ideals and polynomial spaces, they are not closed under sum.

Example 12. Consider the following quadratic pseudo ideals in $X = \{x, y, z\}$:

$$J_1 = \{\lambda z^2 + \lambda x \mid \lambda \in \mathbb{Q}\} \quad \text{and} \quad J_2 = \{\lambda z^2 + \lambda y \mid \lambda \in \mathbb{Q}\}.$$

The sum $J_1 + J_2$ does not form a pseudo ideal. For example, the polynomial $x - y = (z^2 + x) - (z^2 + y)$ is a member of $J_1 + J_2$, but $z(x - y) \notin J_1 + J_2$.

The pseudo ideal $\mathcal{P}s_d(P)$ of degree d *generated* by a set P of polynomials of degree no greater than d is the intersection of all pseudo ideals of degree d that contain P. A *basis* of a pseudo ideal J of degree d is any set of polynomials P that generates J. The *join* $J_1 \uplus_d J_2$ of two pseudo ideals of degree d is simply $\mathcal{P}s_d(J_1 \cup J_2)$. The family of pseudo ideals of degree d forms a complete lattice that satisfies both chain conditions.

Proposition 13. *For any $d \geq 1$, $\langle \mathcal{T}h_d, [\![\cdot]\!] \rangle$ is a Galois connection from the lattice of relations $\langle \mathcal{R}, \subseteq, \cap, \cup \rangle$ to $\langle \mathcal{J}_d, \supseteq, \uplus_d, \cap \rangle$, the dual of the lattice of pseudo ideals of $\mathbb{Q}_d[X, X']$.*

Proof. Let $d \geq 1$ be a degree bound. For any relation ρ, $\mathcal{T}h_d(\rho)$ is a subset of $\mathbb{Q}_d[X, X']$, contains the zero polynomial, and is closed under polynomial sums and degree-bounded products. Thus $\mathcal{T}h_d(\rho)$ is a pseudo ideal of degree d. If $\rho_1 \subseteq \rho_2$, then $\mathcal{T}h_d(\rho_1) \supseteq \mathcal{T}h_d(\rho_2)$. Furthermore, since $\mathcal{T}h_d(\rho)$ consists of polynomials that vanish on ρ, $\rho \subseteq [\![\mathcal{T}h_d(\rho)]\!]$. For any pseudo ideals J_1 and J_2 of degree d, $J_1 \supseteq J_2$ implies $[\![J_1]\!] \subseteq [\![J_2]\!]$. For any pseudo ideal J of degree d, $[\![J]\!]$ causes every polynomial in J to vanish. Thus, $\mathcal{T}h_d([\![J]\!]) \supseteq J$. □

3.4. Representing pseudo ideals

A basis P of a pseudo ideal J of degree d is *closed* if $J = \mathcal{S}p(P)$. That is, P is closed if the polynomial space it generates is closed under degree-bounded products and is therefore a pseudo ideal. We represent pseudo ideals by standard closed bases. This representation, while frequently redundant, simplifies the implementation of operators on pseudo ideals.

Essential to our representation of pseudo ideals is an algorithm to compute the closure of a basis. Given a basis P of a pseudo ideal J, the algorithm computes a closed basis P_* of J. The basis P_* is computed iteratively, as the limit of a finite sequence of bases $P_1, P_2, \ldots, P_k$. Initially $P_1 = P$, and for each $i \geq 1$, P_{i+1} is computed from P_i as follows: We first compute a basis Q_i of the subspace of polynomials of P_i of degree no greater than $d-1$. This basis is computed by applying Gauss–Jordan reduction to eliminate all monomials in P_i of degree d. We then take P_{i+1} to be the standard basis of the space $\mathcal{S}p(P_i \cup \{xq \mid q \in Q_i, x \in X\})$. The algorithm terminates when $P_{i+1} = P_i$. Since the lattice of polynomial spaces satisfies the ascending chain condition, termination is guaranteed.

Example 14. Let J be the quadratic pseudo ideal generated by P, where

$$P = \{x' - x, y', z'\}.$$

In one iteration, we compute the closed basis of J as

$$P_* = \left\{ \begin{array}{cccccc} x'-x, & x'y-xy, & y', & y'x, & z', & (z')^2, \\ (x')^2-x^2, & x'z-xz, & x'y', & y'y, & x'z', & z'x, \\ x'x-x^2, & & (y')^2, & y'z, & y'z', & z'y, \\ & & & & & z'z \end{array} \right\}.$$

The polynomial $(y')^2 - z'$, which was lost in the analysis using polynomial spaces presented in Example 10, is in the space generated by P_*.

The correctness of our algorithm is based on the following result:

Proposition 15. *A polynomial space S of degree d is a pseudo ideal of degree d iff $xp \in S$ for every polynomial $p \in S$ with* $\deg(p) \leq d-1$ *and variable $x \in X$.*

Proof. Let S be a polynomial space of degree d. Suppose S is also a pseudo ideal of degree d. Then for every $p \in S$ with $\deg(p) \leq d-1$ and $x \in X$, we have $xp \in S$. Conversely, suppose $xp \in S$ for every $p \in S$ with $\deg(p) \leq d-1$ and $x \in X$. We will first show that $mp \in S$ for every $p \in S$ and monomial m with $\deg(mp) \leq d$ by induction on $\deg(m)$. The base case is immediate, since 1 is the only monomial of degree 0.

To establish the induction step, suppose that for every monomial l of degree k and every polynomial $p \in S$, if $\deg(lp) \leq d$ then $lp \in S$. Let m be a monomial of degree $k+1$, p be any polynomial of S, and suppose $\deg(mp) \leq d$. For some variable x and some monomial l, we have $m = xl$ and $\deg(l) = k$. Thus, by the induction hypothesis, $lp \in S$. Furthermore, $\deg(lp) \leq d-1$. Since it is assumed that S is closed under variable products, we have $x(lp) = mp \in S$.

Now, suppose $p \in S$ and $q = c_1 m_1 + \cdots + c_k m_k$ with $\deg(qp) \leq d$. Since S is closed under degree-bounded products with monomials, $m_1 p, \ldots, m_k p \in S$. Since S is a space, $qp \in S$. Thus, S is a pseudo ideal of degree d. □

3.5. Computing operators on pseudo ideals

Since we represent pseudo ideals by standard closed bases, operators on pseudo ideals can be implemented using algorithms for vector spaces. The intersection $J_1 \cap J_2$ of pseudo ideals $J_1 = \mathcal{S}p(P_1)$ and $J_2 = \mathcal{S}p(P_2)$ is computed using Karr's algorithm. The join $J_1 \uplus_d J_2$ is computed by Gauss–Jordan reduction applied to the closure of $P_1 \cup P_2$. The inclusion $J_1 \subseteq J_2$ is decided by verifying that the standard basis of $\mathcal{S}p(P_1 \cup P_2)$ is P_2.

For our analysis, we will also need an abstract composition operator $J_1 \bullet_d J_2$ that approximates the concrete composition $[\![J_1]\!] \circ [\![J_2]\!]$. The abstract operator is computed using the following algorithm: Let $J_1 = \mathcal{S}p(P_1)$ and $J_2 = \mathcal{S}p(P_2)$ be pseudo ideals of degree d represented by closed bases. We switch temporarily to the polynomial ring $\mathbb{Q}[X, \bar{X}, X']$, where $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$ is a set of *intermediate variables* corresponding to the variables of X. In this ring, we compute the closure Q_* of $Q = P_1[X' \to \bar{X}] \cup P_2[X \to \bar{X}]$, where $P_i[Y \to Z]$ denotes substitution in

P_i of the variables in Y by the corresponding variables in Z. We then apply Gauss–Jordan reduction to eliminate all monomials of Q_* that contain an intermediate variable. The resulting set P of polynomials in X and X' is the basis of $J_1 \bullet_d J_2$.

Proposition 16. *For every $d \geq 1$ and every pair of pseudo ideals J_1 and J_2 of degree d, $\mathcal{T}h_d([\![J_1]\!] \circ [\![J_2]\!]) \supseteq J_1 \bullet_d J_2$.*

Proof. Let $d \geq 1$ be a degree bound, and let J_1 and J_2 be polynomial pseudo ideals of degree d represented by standard closed bases P_1 and P_2, respectively. Consider the ternary relation r consisting of all triples $\langle \sigma, \bar{\sigma}, \sigma' \rangle$ with $\langle \sigma, \bar{\sigma} \rangle \in [\![J_1]\!]$ and $\langle \bar{\sigma}, \sigma' \rangle \in [\![J_2]\!]$. This relation causes every polynomial in $Q = P_1[X' \to \bar{X}] \cup P_2[X \to \bar{X}]$ to vanish, where the intermediate variables $\bar{X}$ are interpreted as in the intermediate states. Thus, every polynomial in Q_*, the closure of Q, also vanishes on r.

Let P be the result of eliminating all monomials containing an intermediate variable from Q_*. Since $P \subseteq \mathcal{S}p(Q_*) \cap \mathbb{Q}_d[X, X']$, every polynomial in P vanishes on the relation ρ, where $\langle \sigma, \sigma' \rangle \in \rho$ iff $\langle \sigma, \bar{\sigma}, \sigma' \rangle \in r$ for some $\bar{\sigma}$. But ρ is simply $[\![J_1]\!] \circ [\![J_2]\!]$. So, every polynomial in P vanishes on $[\![J_1]\!] \circ [\![J_2]\!]$. Since $J_1 \bullet_d J_2 = \mathcal{S}p(P)$, we have $\mathcal{T}h_d([\![J_1]\!] \circ [\![J_2]\!]) \supseteq J_1 \bullet_d J_2$. □

3.6. Analysis with pseudo ideals

Let $\mathcal{P} = \langle X, L, \mathcal{T}, \ell_{\text{init}}, \ell_{\text{fin}} \rangle$ be an algebraic transition system, and let d be a degree bound, where $d \geq \deg(\mathcal{P})$. That is, the transition relation of each transition $\tau_i \in \mathcal{T}$ is represented by a finite set of polynomials P_i of degree no greater than d.

For any path $\pi = \ell_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_n} \ell_n$ of $\mathcal{P}$, define the *abstract path relation* $[\![\pi]\!]_{\mathcal{J}_d}$ to be the pseudo ideal $(J_1 \bullet_d J_2) \bullet_d \cdots \bullet_d J_n$, where $J_i = \mathcal{P}s_d(P_i)$ for each $i \in \{1, \ldots, n\}$. The abstract path relation $[\![\Pi]\!]_{\mathcal{J}_d}$ of a set of paths is the intersection of the abstract path relations of the paths in Π.

Proposition 17. *For every $d \geq \deg(\mathcal{P})$ and path π of $\mathcal{P}$, $\mathcal{T}h_d([\![\pi]\!]) \supseteq [\![\pi]\!]_{\mathcal{J}_d}$. For every set of paths Π, $\mathcal{T}h_d([\![\Pi]\!]) \supseteq [\![\Pi]\!]_{\mathcal{J}_d}$.*

Proof. Let $\pi = \ell_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_n} \ell_n$ be an arbitrary path of $\mathcal{P}$. For every $i \in \{1, \ldots, n\}$, let $J_i = \mathcal{P}s_d(P_i)$, where τ_i has relation $[\![P_i]\!]$. Since $P_i \subseteq J_i$, $[\![P_i]\!] \supseteq [\![J_i]\!]$. Since $\mathcal{T}h_d([\![P_i]\!]) \supseteq J_i$ and $\langle \mathcal{T}h_d, [\![\cdot]\!] \rangle$ is an adjunction, $[\![P_i]\!] \subseteq [\![J_i]\!]$. Thus, $[\![P_i]\!] = [\![J_i]\!]$.

Now,

$$\begin{aligned}
\mathcal{T}h_d([\![\pi]\!]) &= \mathcal{T}h_d([\![P_1]\!] \circ \cdots \circ [\![P_n]\!]) \\
&= \mathcal{T}h_d([\![J_1]\!] \circ \cdots \circ [\![J_n]\!]) \\
&\supseteq (J_1 \bullet_d J_2) \bullet_d \cdots \bullet_d J_n \quad \text{(by Proposition 16)} \\
&= [\![\pi]\!]_{J_d}.
\end{aligned}$$

For a set of paths Π, $\mathcal{T}h_d([\![\Pi]\!]) = \bigcap_{\pi \in \Pi} \mathcal{T}h_d([\![\pi]\!]) \supseteq \bigcap_{\pi \in \Pi} [\![\pi]\!]_{\mathcal{J}_d} = [\![\Pi]\!]_{\mathcal{J}_d}$. □

Let C be a cutset for $\mathcal{P}$, and suppose C consists of a single cutpoint c. Consider the following abstract path relations of $\mathcal{P}$:

$$J_{\text{simp}} = [\![\text{Simp}_C(\ell_{\text{init}}, \ell_{\text{fin}})]\!]_{\mathcal{J}_d} \qquad J_{\text{cyc}} = [\![\text{Simp}_C(c, c)]\!]_{\mathcal{J}_d}$$
$$J_{\text{init}} = [\![\text{Simp}_C(\ell_{\text{init}}, c)]\!]_{\mathcal{J}_d} \qquad J_{\text{fin}} = [\![\text{Simp}_C(c, \ell_{\text{fin}})]\!]_{\mathcal{J}_d}$$

Let the abstract semantic function $g_{\mathcal{P}}$ on $\langle \mathcal{J}_d, \supseteq, \uplus_d, \cap \rangle$ satisfy

$$g_{\mathcal{P}}(J) = J_{\text{init}} \cap (J \bullet_d J_{\text{cyc}}),$$

for every $J \in \mathcal{J}_d$. Then the abstract semantics of $\mathcal{P}$ are defined as

$$[\![\mathcal{P}]\!]_{\mathcal{J}_d} = J_{\text{simp}} \cap (J_* \bullet_d J_{\text{fin}}),$$

where J_* is the least fixed point of $g_{\mathcal{P}}$. Since each of the operators used in defining $[\![\mathcal{P}]\!]_{\mathcal{J}_d}$ is computable, and since the abstract domain has no ascending chains, the abstract semantics are computable.

When C contains more than one cutpoint, the abstract semantic function $g_{\mathcal{P}}$ is defined on maps from C to $\mathcal{J}_d$:

$$g_{\mathcal{P}}(\zeta)(c) = [\![\mathrm{Simp}_C(\ell_{\mathrm{init}}, c)]\!]_{\mathcal{J}_d} \cap \left(\bigcap_{b \in C} \zeta(b) \bullet_d [\![\mathrm{Simp}_C(b, c)]\!]_{\mathcal{J}_d} \right),$$

for every map ζ from C to $\mathcal{J}_d$ and every cutpoint $c \in C$. The abstract semantics are then defined as

$$[\![\mathcal{P}]\!]_{\mathcal{J}_d} = [\![\mathrm{Simp}_C(\ell_{\mathrm{init}}, \ell_{\mathrm{fin}})]\!]_{\mathcal{J}_d} \cap \left(\bigcap_{c \in C} \zeta_*(c) \bullet_d [\![\mathrm{Simp}_C(c, \ell_{\mathrm{fin}})]\!]_{\mathcal{J}_d} \right),$$

where ζ_* is the least fixed point of $g_{\mathcal{P}}$.

Example 18. Consider the algebraic transition system $\mathcal{P}$ of Example 7, where all cycles are cut by the single cutpoint ℓ_2. We approximate its relational semantics in the domain of quadratic pseudo ideals as follows:

$$J_{\mathrm{simp}} = \mathcal{P}s_2(\{1\}) \qquad J_{\mathrm{cyc}} = \mathcal{P}s_2(\{x' - x,\ y' - y - 1,\ z' - z - 2y - 1\})$$
$$J_{\mathrm{init}} = \mathcal{P}s_2(\{x' - x,\ y',\ z'\}) \qquad J_{\mathrm{fin}} = \mathcal{P}s_2(\{y - x,\ x' - x,\ y' - y,\ z' - z\})$$

The abstract semantic function is $g_{\mathcal{P}}$, where

$$g_{\mathcal{P}}(J) = J_{\mathrm{init}} \cap (J \bullet_2 J_{\mathrm{cyc}}),$$

for all $J \in \mathcal{J}_2$. Computing the least fixed point of $g_{\mathcal{P}}$ iteratively yields the sequence $J_1 \supset J_2 \supset J_3 \supset J_4 \supset J_5$, where

$$\begin{aligned}
J_1 &= g_{\mathcal{P}}(\mathbb{Q}_2[X, X']) \\
&= \mathcal{P}s_2(\{x' - x,\ y',\ z'\}), \\
J_2 &= g_{\mathcal{P}}(J_1) \\
&= \mathcal{P}s_2(\{x' - x,\ y' - z',\ (y')^2 - z'\}), \\
J_3 &= g_{\mathcal{P}}(J_2) \\
&= \mathcal{P}s_2(\{x' - x,\ (y')^2 - z',\ y'z' + 2y' - 3z',\ (z')^2 + 6y' - 7z'\}), \\
J_4 &= g_{\mathcal{P}}(J_3) \\
&= \mathcal{P}s_2\left(\left\{x' - x,\ (y')^2 - z',\ y'z' - \frac{1}{6}(z')^2 + y' - \frac{11}{6}z'\right\}\right), \text{ and} \\
J_5 &= g_{\mathcal{P}}(J_4) \\
&= \mathcal{P}s_2(\{x' - x,\ (y')^2 - z'\}).
\end{aligned}$$

The crucial step is the second, which discovers that $z' = (y')^2$:

$$\begin{aligned}
J_2 &= g_{\mathcal{P}}(J_1) \\
&= J_{\mathrm{init}} \cap (J_1 \bullet_2 J_{\mathrm{cyc}}) \\
&= J_{\mathrm{init}} \cap (\mathcal{P}s_2(\{x' - x,\ y',\ z'\}) \bullet_2 \mathcal{P}s_2(\{x' - x,\ y' - y - 1,\ z' - z - 2y - 1\})) \\
&= J_{\mathrm{init}} \cap \mathcal{P}s_2(\{x' - x,\ y' - 1,\ z' - 1\}) \\
&= \mathcal{P}s_2(\{x' - x,\ y',\ z'\}) \cap \mathcal{P}s_2(\{x' - x,\ y' - 1,\ z' - 1\}) \\
&= \mathcal{P}s_2(\{x' - x,\ y' - z',\ (y')^2 - z'\}).
\end{aligned}$$

Taking the least fixed point J_* to be J_5, we compute the abstract semantics as

$$\begin{aligned}
[\![\mathcal{P}]\!]_{\mathcal{J}_2} &= J_{\mathrm{simp}} \cap (J_* \bullet_2 J_{\mathrm{fin}}) \\
&= \mathcal{P}s_2(\{x' - x,\ y' - x,\ (x)^2 - z'\})
\end{aligned}$$

Thus, the relational semantics of $\mathcal{P}$ are approximated by the relation

$$[\![x' = x \ \wedge \ y' = x \ \wedge \ z' = x^2]\!].$$

3.7. Soundness of the analysis

Let $\mathcal{P} = \langle X, L, \mathcal{T}, \ell_{\text{init}}, \ell_{\text{fin}}\rangle$ be an algebraic transition system, and let d be a degree bound, where $d \geq \deg(\mathcal{P})$. Suppose C is a cutset for $\mathcal{P}$. If C consists of a single cutpoint c, the concrete semantic function $f_{\mathcal{P}}$ is defined on the lattice $\langle \mathcal{R}, \subseteq, \cap, \cup\rangle$, and the abstract semantic function $g_{\mathcal{P}}$ is defined on $\langle \mathcal{J}_d, \supseteq, \uplus_d, \cap\rangle$.

Lemma 19. *$\mathcal{T}h_d([\![\mathcal{P}]\!]) \supseteq [\![\mathcal{P}]\!]_{\mathcal{J}_d}$ when C consists of a single cutpoint c.*

Proof. Let ρ_* be the least fixed point of $f_{\mathcal{P}}$, and let J_* be the least fixed point of $g_{\mathcal{P}}$. We will first demonstrate that $\mathcal{T}h_d(\rho_*) \supseteq J_*$. By Proposition 17, $\mathcal{T}h_d(\rho_{\text{init}}) \supseteq J_{\text{init}}$ and $\mathcal{T}h_d(\rho_{\text{cyc}}) \supseteq J_{\text{cyc}}$. Since $\langle \mathcal{T}h_d, [\![\cdot]\!]\rangle$ is a Galois connection and, thus, an adjunction, $\rho_{\text{cyc}} \subseteq [\![J_{\text{cyc}}]\!]$. So, for any $J \in \mathcal{J}_d$,

$$\begin{aligned}
\mathcal{T}h_d(f_{\mathcal{P}}([\![J]\!])) &= \mathcal{T}h_d(\rho_{\text{init}} \cup ([\![J]\!] \circ \rho_{\text{cyc}})) \\
&= \mathcal{T}h_d(\rho_{\text{init}}) \cap \mathcal{T}h_d([\![J]\!] \circ \rho_{\text{cyc}}) \\
&\supseteq J_{\text{init}} \cap \mathcal{T}h_d([\![J]\!] \circ \rho_{\text{cyc}}) \quad (\text{since } \mathcal{T}h_d(\rho_{\text{init}}) \supseteq J_{\text{init}}) \\
&\supseteq J_{\text{init}} \cap \mathcal{T}h_d([\![J]\!] \circ [\![J_{\text{cyc}}]\!]) \quad (\text{since } \rho_{\text{cyc}} \subseteq [\![J_{\text{cyc}}]\!]) \\
&\supseteq J_{\text{init}} \cap (J \bullet_d J_{\text{cyc}}) \quad (\text{by Proposition 16}) \\
&= g_{\mathcal{P}}(J).
\end{aligned}$$

Therefore, by Theorem 4, $\rho_* \subseteq [\![J_*]\!]$ and $\mathcal{T}h_d(\rho_*) \supseteq J_*$.

Next, we show that $\mathcal{T}h_d([\![\mathcal{P}]\!]) \supseteq [\![\mathcal{P}]\!]_{\mathcal{J}_d}$. By Proposition 17, $\mathcal{T}h_d(\rho_{\text{simp}}) \supseteq J_{\text{simp}}$ and $\mathcal{T}h_d(\rho_{\text{fin}}) \supseteq J_{\text{fin}}$. Again, since $\langle \mathcal{T}h_d, [\![\cdot]\!]\rangle$ is an adjunction, $\rho_{\text{fin}} \subseteq [\![J_{\text{fin}}]\!]$. Thus,

$$\begin{aligned}
\mathcal{T}h_d([\![\mathcal{P}]\!]) &= \mathcal{T}h_d(\rho_{\text{simp}} \cup (\rho_* \circ \rho_{\text{fin}})) \\
&= \mathcal{T}h_d(\rho_{\text{simp}}) \cap \mathcal{T}h_d(\rho_* \circ \rho_{\text{fin}}) \\
&\supseteq \mathcal{T}h_d(\rho_{\text{simp}}) \cap \mathcal{T}h_d([\![J_*]\!] \circ \rho_{\text{fin}}) \quad (\text{since } \rho_* \subseteq [\![J_*]\!]) \\
&\supseteq J_{\text{simp}} \cap \mathcal{T}h_d([\![J_*]\!] \circ \rho_{\text{fin}}) \quad (\text{since } \mathcal{T}h_d(\rho_{\text{simp}}) \supseteq J_{\text{simp}}) \\
&\supseteq J_{\text{simp}} \cap \mathcal{T}h_d([\![J_*]\!] \circ [\![J_{\text{fin}}]\!]) \quad (\text{since } \rho_{\text{fin}} \subseteq [\![J_{\text{fin}}]\!]) \\
&\supseteq J_{\text{simp}} \cap (J_* \bullet_d J_{\text{fin}}) \quad (\text{by Proposition 16}) \\
&= [\![\mathcal{P}]\!]_{\mathcal{J}_d}. \quad \square
\end{aligned}$$

If the cutset C contains multiple cutpoints, $f_{\mathcal{P}}$ is defined on $\langle \mathcal{R}^C, \sqsubseteq_{\mathcal{R}}, \sqcap_{\mathcal{R}}, \sqcup_{\mathcal{R}}\rangle$, the complete lattice of total maps from C to $\mathcal{R}$, where $\sqsubseteq_{\mathcal{R}}, \sqcap_{\mathcal{R}}$, and $\sqcup_{\mathcal{R}}$ are pointwise extensions of $\subseteq, \cap$, and $\cup$, respectively. For example, $\eta_1 \sqsubseteq_{\mathcal{R}} \eta_2$ iff $\eta_1(c) \subseteq \eta_2(c)$ for all $c \in C$. Similarly, $g_{\mathcal{P}}$ is defined on $\langle \mathcal{J}_d^C, \sqsupseteq_{\mathcal{J}_d}, \sqcup_{\mathcal{J}_d}, \sqcap_{\mathcal{J}_d}\rangle$, the complete lattice of maps from C to $\mathcal{J}_d$. These two lattices are related by the Galois connection $\langle \alpha, \gamma\rangle$, where $\alpha(\eta)(c) = \mathcal{T}h_d(\eta(c))$ and $\gamma(\zeta)(c) = [\![\zeta(c)]\!]$ for all $\eta \in \mathcal{R}^C$, $\zeta \in \mathcal{J}_d^C$, and $c \in C$ [25].

Lemma 20. *$\mathcal{T}h_d([\![\mathcal{P}]\!]) \supseteq [\![\mathcal{P}]\!]_{\mathcal{J}_d}$ when the cutset C contains multiple cutpoints.*

Proof. The proof is similar to the proof of Lemma 19, and we omit many details. Let η_* be the least fixed point of $f_{\mathcal{P}}$, and let ζ_* be the least fixed point of $g_{\mathcal{P}}$. For every $\zeta \in \mathcal{J}_d^C$ and cutpoint $c \in C$,

$$\begin{aligned}
\alpha(f_{\mathcal{P}}(\gamma(\zeta)))(c) &= \mathcal{T}h_d([\![\text{Simp}_C(\ell_{\text{init}}, c)]\!]) \cap \left(\bigcap_{b \in C} \mathcal{T}h_d([\![\zeta(b)]\!] \circ [\![\text{Simp}_C(b, c)]\!])\right) \\
&\supseteq [\![\text{Simp}_C(\ell_{\text{init}}, c)]\!]_{\mathcal{J}_d} \cap \left(\bigcap_{b \in C} \mathcal{T}h_d([\![\zeta(b)]\!] \circ [\![\, [\![\text{Simp}_C(b, c)]\!]_{\mathcal{J}_d}\,]\!])\right) \\
&\supseteq [\![\text{Simp}_C(\ell_{\text{init}}, c)]\!]_{\mathcal{J}_d} \cap \left(\bigcap_{b \in C} \zeta(b) \bullet_d [\![\text{Simp}_C(b, c)]\!]_{\mathcal{J}_d}\right) \\
&= g_{\mathcal{P}}(\zeta)(c).
\end{aligned}$$

```
var x, y, z : integer;
ℓ1 : z := 0
ℓ2 : while y ≠ 0 do
       if y mod 2 = 1 then
         ⟨x, y, z⟩ := ⟨2x, (y − 1) div 2, z + x⟩
       else
         ⟨x, y, z⟩ := ⟨2x, y div 2, z⟩;
ℓ3 : halt
```

Fig. 2. Russian peasant multiplication.

The first inclusion is justified by Proposition 17 and the fact that for any set of paths Π, $\mathcal{T}h_d(\llbracket \Pi \rrbracket) \supseteq \llbracket \Pi \rrbracket_{\mathcal{J}_d}$ iff $\llbracket \Pi \rrbracket \subseteq \llbracket\, \llbracket \Pi \rrbracket_{\mathcal{J}_d}\, \rrbracket$. The second is justified by Proposition 16. Invoking Theorem 4, $\mathcal{T}h_d(\eta_*(c)) \supseteq \zeta_*(c)$ and $\eta_*(c) \subseteq \llbracket \zeta_*(c) \rrbracket$ for every cutpoint $c \in C$.

Next, we show that $\mathcal{T}h_d(\llbracket \mathcal{P} \rrbracket) \supseteq \llbracket \mathcal{P} \rrbracket_{\mathcal{J}_d}$.

$$
\begin{aligned}
\mathcal{T}h_d(\llbracket \mathcal{P} \rrbracket) &= \mathcal{T}h_d(\llbracket \mathrm{Simp}_C(\ell_{\mathrm{init}}, \ell_{\mathrm{fin}}) \rrbracket) \cap \left(\bigcap_{c \in C} \mathcal{T}h_d(\eta_*(c) \circ \llbracket \mathrm{Simp}_C(c, \ell_{\mathrm{fin}}) \rrbracket) \right) \\
&\supseteq \mathcal{T}h_d(\llbracket \mathrm{Simp}_C(\ell_{\mathrm{init}}, \ell_{\mathrm{fin}}) \rrbracket) \cap \left(\bigcap_{c \in C} \mathcal{T}h_d(\llbracket \zeta_*(c) \rrbracket \circ \llbracket \mathrm{Simp}_C(c, \ell_{\mathrm{fin}}) \rrbracket) \right) \\
&\supseteq \llbracket \mathrm{Simp}_C(\ell_{\mathrm{init}}, \ell_{\mathrm{fin}}) \rrbracket_{\mathcal{J}_d} \cap \left(\bigcap_{c \in C} \mathcal{T}h_d(\llbracket \zeta_*(c) \rrbracket \circ \llbracket\, \llbracket \mathrm{Simp}_C(c, \ell_{\mathrm{fin}}) \rrbracket_{\mathcal{J}_d}\, \rrbracket) \right) \\
&\supseteq \llbracket \mathrm{Simp}_C(\ell_{\mathrm{init}}, \ell_{\mathrm{fin}}) \rrbracket_{\mathcal{J}_d} \cap \left(\bigcap_{c \in C} \zeta_*(c) \bullet_d \llbracket \mathrm{Simp}_C(c, \ell_{\mathrm{fin}}) \rrbracket_{\mathcal{J}_d} \right) \\
&= \llbracket \mathcal{P} \rrbracket_{\mathcal{J}_d},
\end{aligned}
$$

where the first inclusion is justified by $\eta_*(c) \subseteq \llbracket \zeta_*(c) \rrbracket$, and the last two are justified by Propositions 17 and 16, respectively. □

Combining Lemmas 19 and 20, we conclude that our method is sound.

Theorem 21 (*Soundness*). *For any algebraic transition system $\mathcal{P}$, degree bound $d \geq \deg(\mathcal{P})$, and non-empty cut-set C, $\mathcal{T}h_d(\llbracket \mathcal{P} \rrbracket) \supseteq \llbracket \mathcal{P} \rrbracket_{\mathcal{J}_d}$.*

4. Applications

To demonstrate the utility of our method, we have implemented it in Java and applied it to several programs taken from the literature on non-linear invariant generation. For the programs we have considered, our analysis produces results as precise as those produced by other methods [28,26].

4.1. Russian peasant multiplication

The program shown in Fig. 2 is an implementation of a folk algorithm for multiplication. It computes the product of non-negative integers x and y by decomposing y into its binary representation. We model the body of the loop by two transitions, one for each branch of the conditional. The assignments to x and z in the loop body are polynomial, but the assignments to y are not. The conditional statement, however, allows the assignments to y to be modeled precisely: $y := (y - 1)$ div 2 is equivalent to $y := \frac{1}{2}y - \frac{1}{2}$ when y is odd, and $y := y$ div 2 is equivalent to $y := \frac{1}{2}y$ when y is even. The loop condition is a disequality. Thus, it is ignored when modeling the loop body. However, its negation, $y = 0$, is modeled precisely for the loop exit.

```
var a, b, p, q : integer;
ℓ1 : ⟨p, q⟩ := ⟨1, 0⟩;
ℓ2 : while a ≠ 0 ∧ b ≠ 0 do
        if a mod 2 = 0 ∧ b mod 2 = 0 then
          ⟨a, b, p, q⟩ := ⟨a div 2, b div 2, 4p, q⟩
        else if a mod 2 = 1 ∧ b mod 2 = 0 then
          ⟨a, b, p, q⟩ := ⟨a − 1, b, p, q + bp⟩
        else if a mod 2 = 0 ∧ b mod 2 = 1 then
          ⟨a, b, p, q⟩ := ⟨a, b − 1, p, q + ap⟩
        else
          ⟨a, b, p, q⟩ := ⟨a − 1, b − 1, p, q + (a + b − 1)p⟩;
ℓ3 : halt
```

Fig. 3. Alternate product.

All cycles of this program are cut by the single cutpoint ℓ_2. Applying our analysis in the domain of quadratic pseudo ideals yields the following approximate semantics to cutpoint ℓ_2:

$$[\![z' = xy - x'y']\!].$$

The relational semantics of the program are then approximated as

$$[\![y' = 0 \ \wedge \ z' = xy]\!].$$

The analysis takes 90 ms to complete.[2] Thus, the program halts with the final value of z equal to the product of the initial values of x and y.

4.2. Alternate product

The program shown in Fig. 3 is taken from Rodríguez-Carbonell and Kapur [26]. It computes the product of non-negative integers a and b. The body of the loop is modeled by four transitions, one for each branch of the conditional. The first assignment in the loop body, while not polynomial, can be modeled precisely since the conditional statement guarantees that a and b are both even. All remaining assignments are polynomial. The loop condition is a conjunction of disequalities and is ignored in modeling the loop body. The loop exit, on the other hand, is modeled precisely since the disjunction $a = 0 \vee b = 0$ is equivalent to the polynomial equality $ab = 0$.

All cycles are cut by the single cutpoint ℓ_2. Noting that the program contains quadratic assignments, we apply our analysis in the domain of cubic pseudo ideals. Our method approximates the relational semantics to ℓ_2 as

$$[\![q' = ab - a'b'p']\!],$$

and approximates the semantics of the program as

$$[\![a'b' = 0 \ \wedge \ q' = ab]\!].$$

While the result of the analysis is a quadratic relation, the fixed point used to compute it is cubic. Had we analyzed the program in the domain of quadratic pseudo ideals, we would have derived the weaker approximation

$$[\![a'b' = 0]\!].$$

By working with cubic rather than quadratic pseudo ideals, we increase the accuracy of the analysis, along with the space used and the time taken (2.1 s).

[2] Reported times are for a 1.7 GHz Pentium with 1 GB of RAM.

$$
\begin{array}{l}
\textbf{var } x_1, x_2, y_1, y_2 : \textbf{integer};\\
\ell_1 : \langle y_1, y_2\rangle := \langle 0, x_1\rangle;\\
\ell_2 : \textbf{while } y_2 \geq x_2 \textbf{ do}\\
\qquad \langle y_1, y_2\rangle := \langle y_1 + 1, y_2 - x_2\rangle;\\
\ell_3 : \textbf{halt}
\end{array}
$$

Fig. 4. Division by subtraction.

$$
\begin{array}{l}
\textbf{var } x_1, x_2, y_1, y_2, y_3, y_4 : \textbf{integer};\\
\ell_1 : \langle y_1, y_2, y_3, y_4\rangle := \langle x_1, x_2, 1, 0\rangle;\\
\ell_2 : \textbf{while } y_1 > y_2 \textbf{ do}\\
\qquad \langle y_2, y_3\rangle := \langle 2y_2, 2y_3\rangle;\\
\ell_3 : \textbf{while } \text{true } \textbf{do}\\
\qquad \left[\begin{array}{l}
\textbf{if } y_1 \geq y_2 \textbf{ then}\\
\quad \langle y_1, y_4\rangle := \langle y_1 - y_2, y_4 + y_3\rangle;\\
\textbf{if } y_3 = 1 \textbf{ then}\\
\quad \ell_4 : \textbf{halt};\\
\langle y_2, y_3\rangle := \langle y_2 \text{ div } 2, y_3 \text{ div } 2\rangle
\end{array}\right]
\end{array}
$$

Fig. 5. Hardware integer division.

4.3. Division by subtraction

The program shown in Fig. 4 performs division by repeated subtraction. Given a non-negative integer x_1 and a positive integer x_2, it computes their quotient and remainder into y_1 and y_2, respectively. All assignments in this program are polynomial and are modeled precisely. The loop condition is an inequality, and neither it nor its negation can be modeled by polynomial equalities. Thus, we treat the decision to iterate or terminate the loop as a non-deterministic choice.

All cycles of the program are cut by the single cutpoint ℓ_2, and an analysis in the domain of quadratic pseudo ideals yields the following relation in 190 ms:

$$[\![x_1' = x_1 \ \wedge\ x_2' = x_2 \ \wedge\ x_1 = x_2 y_1' + y_2']\!].$$

This approximation of the relational semantics is imprecise. It does not guarantee that the final values of y_1 and y_2 are the quotient and remainder, respectively, of x_1 and x_2. To improve the result, linear relation analysis [8,14] can be applied to discover the invariant $0 \leq y_2 < x_2$ at ℓ_3, assuming $x_1 \geq 0$ and $x_2 > 0$ initially. Combining the results of both analyses yields a more precise approximation of the relational semantics of the program:

$$\left[\!\!\left[\begin{array}{l} x_1' = x_1 \ \wedge\ x_2' = x_2 \ \wedge\ x_1 = x_2 y_1' + y_2' \ \wedge \\ \quad (x_1 \geq 0 \ \wedge\ x_2 > 0 \ \rightarrow\ 0 \leq y_2' < x_2) \end{array}\right]\!\!\right]$$

Thus, upon termination, y_1 holds the quotient of x_1 and x_2, and y_2 holds the remainder.

4.4. Hardware integer division

The program shown in Fig. 5, taken from Manna [19], also computes the quotient and remainder of a non-negative integer x_1 and a positive integer x_2. The program contains two loops. The first initializes y_3 to the smallest power of 2 for which $y_2 = x_2 y_3$ is greater than or equal to x_1. The second computes the quotient of x_1 and x_2 into y_4 and the remainder into y_1. In modeling the program, we ignore the condition of the first loop and the first conditional of the second loop, both of which are inequalities. For the second loop, the fact that y_3 is a power of 2 combined with the invariant $y_2 = x_2 y_3$ at ℓ_3 permits us to model the assignments to y_2 and y_3 accurately.

```
var n, p, q, r, h : integer;
ℓ1 : ⟨p, q, r⟩ := ⟨0, 1, n⟩;
ℓ2 : while q ≤ n do
        q := 4q;
ℓ3 : while q ≠ 1 do
     [ q := q div 4;
     | ⟨h, p⟩ := ⟨p + q, p div 2⟩;
     | if r ≥ h then
     |    ⟨p, r⟩ := ⟨p + q, r − h⟩ ]
ℓ4 : halt
```

Fig. 6. Dijkstra's square root.

Taking the cutset to be $\{\ell_2, \ell_3\}$, we analyze the program in the domain of quadratic pseudo ideals, deriving the following relation to ℓ_2:

$$[\![x_1' = x_1 \ \wedge\ x_2' = x_2 \ \wedge\ y_1' = x_1 \ \wedge\ y_4' = 0 \ \wedge\ y_2' = x_2 y_3']\!],$$

and the following relation to ℓ_3:

$$[\![x_1' = x_1 \ \wedge\ x_2' = x_2 \ \wedge\ y_2' = x_2 y_3' \ \wedge\ x_1 y_3' = y_1' y_3' + y_2' y_4']\!].$$

The relational semantics of the program are then approximated as

$$[\![x_1' = x_1 \ \wedge\ x_2' = x_2 \ \wedge\ y_2' = x_2 \ \wedge\ y_3' = 1 \ \wedge\ x_1 = x_2 y_4' + y_1']\!].$$

The entire analysis takes 1.3 s.

Again, this approximation does not guarantee that y_4 holds the quotient and y_1 holds the remainder at exit. To improve the accuracy of the analysis, linear invariants of the program are needed.

4.5. Dijkstra's square root

The program in Fig. 6, taken from Dijkstra [11], computes the integer square root of a non-negative integer n. The program contains two loops, and the second loop contains non-polynomial assignments to the variables p and q. However, Dijkstra's analysis of the program reveals that, at location ℓ_3, p is divisible by 2^k and $q = 4^k$ for some integer $k \geq 0$. Thus, the assignments to p and q can be modeled precisely.

Taking the cutset to be $\{\ell_2, \ell_3\}$, we analyze the program in the domain of quadratic pseudo ideals, deriving the following relation to ℓ_2:

$$[\![h' = h \ \wedge\ n' = n \ \wedge\ p' = 0 \ \wedge\ r' = n]\!],$$

and the following relation to ℓ_3:

$$[\![n' = n \ \wedge\ (p')^2 = q'n - q'r']\!].$$

The relational semantics are then approximated as

$$[\![n' = n \ \wedge\ q' = 1 \ \wedge\ n = (p')^2 + r']\!].$$

The analysis takes 800 ms.

At location ℓ_3, the linear inequalities $0 \leq r < 2p + q$ are invariant, assuming $n \geq 0$. Combining this invariant with the approximation computed by our method yields a better characterization of the relational semantics:

$$[\![n' = n \ \wedge\ q' = 1 \ \wedge\ r' = n - (p')^2 \ \wedge\ (n \geq 0 \rightarrow (p')^2 \leq n < (p' + 1)^2)]\!].$$

Thus, p' is the integer square root of n, provided n is non-negative.

5. Related work

Recently, a number of proposals have been advanced for generating non-linear invariants of imperative programs.

Sankaranarayanan et al. propose a constraint solving approach for generating invariant non-linear equalities [28]. This work extends our prior joint work on constraint-based generation of invariant linear inequalities [3]. Given a template equality – a polynomial equality whose coefficients are linear expressions in a set of parameters – their approach uses Gröbner basis methods to construct a system of non-linear constraints whose solutions correspond to invariant instances of the template. They then propose heuristics for solving the generated constraints. The principal advantage of the approach is that it eliminates the need for a widening to extrapolate the limits of infinite chains of ideals. The principal disadvantage is its incompleteness. To avoid producing intractable systems of non-linear constraints, the method generates constraints that are sufficient to ensure correctness of the invariant, but not necessary.

Rodríguez-Carbonell and Kapur propose non-linear invariant generation by Abstract Interpretation in the lattice of polynomial ideals, with ideals represented by Gröbner bases [27]. To ensure termination of the analysis, they propose a widening on ideals that computes a Gröbner basis with respect to a graded order on monomials and retains only those polynomials in the basis of degree no greater than a degree bound d. The primary advantage of this approach is the increased precision of working in the lattice of ideals. In addition, the method is able to handle disequalities in the conditions of *if* and *while* statements using quotients of ideals. The primary disadvantage is the potential for encountering the worst-case exponential complexity of Gröbner methods during the analysis. However, the empirical results presented by Rodríguez-Carbonell and Kapur demonstrate that this worst-case complexity does not arise for the programs they consider.

Müller-Olm and Seidl generate invariant polynomial equalities of bounded degree by backward propagation [24]. Their approach is limited to programs with linear assignments and ignores the conditions of *if* and *while* statements. However, their analysis is based on methods from linear algebra and can be performed in both space and time that is polynomial in the number of program variables. The approaches based on Gröbner bases, on the other hand, both have space and time complexity that is exponential in the number of variables in the worst case.

Our method can be seen as a hybrid of the Gröbner basis methods and the method of Müller-Olm and Seidl. Like the Gröbner methods, our method achieves increased precision due to its firm grounding in the theory of polynomial ideals. Like the method of Müller-Olm and Seidl, our method gains space and time efficiency by using representations and algorithms from linear algebra.

6. Conclusion

We have presented a method that computes algebraic approximations of the relational semantics of imperative programs by Abstract Interpretation in a lattice of polynomial pseudo ideals of bounded degree. Our method can be seen as extending Karr's method for generating invariant linear equalities [15] to produce non-linear equalities by exploiting the strong connection between polynomial ideals and infinite dimensional vector spaces [18]. Pseudo ideals provide finite dimensional approximations of polynomial ideals that are fine enough to produce useful results, yet coarse enough to remain tractable. For a fixed degree bound, the number of polynomials required to represent a pseudo ideal, the time required to compute operators on pseudo ideals, and the lengths of chains of pseudo ideals are all bounded by polynomials in the number of program variables. Our approach is incomplete, and it can be less precise than Abstract Interpretation in the domain of polynomial ideals. However, for several programs drawn from the literature on non-linear polynomial invariant generation, our method produces results as precise as those produced by Gröbner basis methods.

In any event, while Abstract Interpretation using polynomial ideals provides a reasonable measure of relative completeness, it too is incomplete. All algebraic theories are radical, but the lattice of polynomial ideals contains non-radical ideals.[3] As a result, an analysis of the program in Fig. 7(a) using polynomial ideals fails to deduce that the program halts with x equal to zero. Moving to the lattice of radical ideals, however, will not yield a complete method. For example, an analysis of the program in Fig. 7(b) using radical ideals will fail to deduce that the program halts with both x and y equal to zero. The appropriate abstract domain for analyzing this program is the lattice of real ideals.

[3] An ideal I is *radical* if $p^d \in I$ implies $p \in I$, and *real* if $p_1^2 + \cdots + p_k^2 \in I$ implies $p_1, \ldots, p_k \in I$.

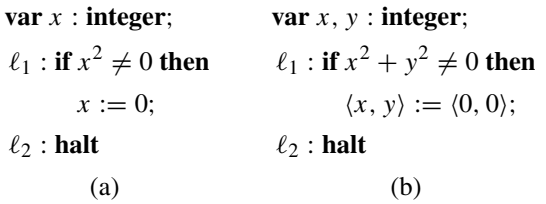

Fig. 7. Incompleteness of ideals.

Our decision to approximate the relational semantics in a lattice of polynomial pseudo ideals rather than devise a widening for the lattice of ideals is a pragmatic one, driven by the worst-case complexity of operations on ideals [21]. For some applications, however, the increased precision afforded by Abstract Interpretation in the lattice of ideals justifies the cost. In these cases, the family of lattices of polynomial pseudo ideals can serve as the basis for a widening for the lattice of ideals. For example, we can iterate for a finite number of steps using polynomial ideals, then move to pseudo ideals of degree d, where d is the maximal degree of any polynomial appearing in a Gröbner basis. Better yet, we can compute a set of monomials based on the Gröbner bases encountered and move to the corresponding lattice of polynomial spaces which are closed under a suitable monomial-restricted product. That is, we can tailor the choice of abstract domain to the set of monomials that appear important to the analysis, thereby permitting relevant high-degree monomials while potentially avoiding the complexity of polynomial pseudo ideals of high degree.

Acknowledgements

We thank Ramesh Bharadwaj, Ralph Jeffords, Elizabeth Leonard, and the anonymous reviewers for their helpful suggestions on an earlier version of this article and on the conference version [2].

References

[1] T. Becker, V. Weispfenning, Gröbner Bases: A Computational Approach to Commutative Algebra, Springer-Verlag, New York, 1993.
[2] M.A. Colón, Approximating the algebraic relational semantics of imperative programs, in: R. Giacobazzi (Ed.), Proceedings of the 11th International Static Analysis Symposium, Springer, 2004, pp. 296–311.
[3] M. Colón, S. Sankaranarayanan, H. Sipma, Linear invariant generation using non-linear constraint solving, in: F. Somenzi, W.H. Jr. (Eds.), Proceedings of the 15th International Conference on Computer Aided Verification, Springer-Verlag, 2003, pp. 420–432.
[4] P. Cousot, Constructive design of a hierarchy of semantics of a transition system by Abstract Interpretation, Electronic Notes in Theoretical Computer Science 6 (1997) 77–102.
[5] P. Cousot, R. Cousot, Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Conference Record of the 4th ACM Symposium on Principles of Programming Languages, 1977, pp. 238–252.
[6] P. Cousot, R. Cousot, Automatic synthesis of optimal invariant assertions: Mathematical foundations, in: Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages, 1977, pp. 1–12.
[7] P. Cousot, R. Cousot, Comparing the Galois connection and widening/narrowing approaches to Abstract Interpretation, in: M. Bruynooghe, M. Wirsing (Eds.), Proceedings of the 4th International Symposium on Programming Language Implementation and Logic Programming, Springer-Verlag, 1992, pp. 269–295.
[8] P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in: Conference Record of the 5th ACM Symposium on Principles of Programming Languages, 1978, pp. 84–96.
[9] D. Cox, J. Little, D. O'Shea, Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra, Springer-Verlag, New York, 1992.
[10] B.A. Davey, H.A. Priestley, Introduction to Lattices and Order, Cambridge University Press, Cambridge, 1990.
[11] E.W. Dijkstra, A Discipline of Programming, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1976.
[12] S.M. German, B. Wegbreit, A synthesizer of inductive assertions, IEEE Transactions on Software Engineering SE-1 (1) (1975) 68–75.
[13] P. Granger, Static analysis of linear congruence equalities among variables of a program, in: S. Abramsky, T.S.E. Maibaum (Eds.), Proceedings of the International Joint Conference on Theory and Practice of Software Development, Springer-Verlag, 1991, pp. 169–192.
[14] N. Halbwachs, Y.-E. Proy, P. Roumanoff, Verification of real-time systems using linear relation analysis, Formal Methods in System Design 11 (2) (1997) 157–185.
[15] M. Karr, Affine relationships among variables of a program, Acta Informatica 6 (1976) 133–151.
[16] S. Katz, Z. Manna, Logical analysis of programs, Communications of the ACM 19 (4) (1976) 188–206.
[17] R.W. Kaye, R. Wilson, Linear Algebra, Oxford University Press, 1998.
[18] D. Lazard, Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations, in: Proceedings of the European Computer Algebra Conference, Springer, 1983, pp. 146–156.
[19] Z. Manna, Mathematical Theory of Computation, McGraw-Hill, New York, 1974.

[20] I. Mastroeni, Numerical power analysis, in: Proceedings of the 2nd Symposium on Programs as Data Objects, Springer-Verlag, 2001, pp. 117–137.
[21] E.W. Mayr, A.R. Meyer, The complexity of the word problems for commutative semigroups and polynomial ideals, Advances in Mathematics 46 (1982) 305–329.
[22] A. Mili, A relational approach to the design of deterministic programs, Acta Informatica 20 (1983) 315–328.
[23] H. Mills, The new math of computer programming, Communications of the ACM 18 (1) (1975) 43–48.
[24] M. Müller-Olm, H. Seidl, Precise interprocedural analysis through linear algebra, in: N.D. Jones, X. Leroy (Eds.), 31st ACM Symposium on Principles of Programming Languages, 2004, pp. 330–341.
[25] F. Nielson, H.R. Nielson, C. Hankin, Principles of Program Analysis, Springer, 1999.
[26] E. Rodríguez-Carbonell, D. Kapur, Automatic generation of polynomial loop invariants for imperative programs, Tech. Rep. TR-CS-2003-39, University of New Mexico, 2003.
[27] E. Rodríguez-Carbonell, D. Kapur, An Abstract Interpretation approach for automatic generation of polynomial invariants, in: R. Giacobazzi (Ed.), Proceedings of the 11th International Static Analysis Symposium, Springer, 2004, pp. 280–295.
[28] S. Sankaranarayanan, H.B. Sipma, Z. Manna, Non-linear loop invariant generation using Gröbner bases, in: N.D. Jones, X. Leroy (Eds.), 31st ACM Symposium on Principles of Programming Languages, 2004, pp. 318–329.
[29] A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons, Chichester, 1986.
[30] B. Wegbreit, Property extraction in well-founded property sets, IEEE Transactions on Software Engineering SE-1 (3) (1975) 270–285.
[31] C.K. Yap, A new lower bound construction for the word problem for commutative Thue systems, Journal of Symbolic Computation 12 (1) (1991) 1–28.