Solvable Polynomial Ideals: The Ideal Reflection for Program Analysis

JOHN CYPHERT, University of Wisconsin-Madison, USA
ZACHARY KINCAID, Princeton University, USA

This paper presents a program analysis method that generates program summaries involving polynomial arithmetic. Our approach builds on prior techniques that use solvable polynomial maps for summarizing loops. These techniques are able to generate *all* polynomial invariants for a restricted class of programs, but cannot be applied to programs outside of this class—for instance, programs with nested loops, conditional branching, unstructured control flow, etc. There currently lacks approaches to apply these prior methods to the case of general programs. This paper bridges that gap. Instead of restricting the kinds of programs we can handle, our method *abstracts* every loop into a model that can be solved with prior techniques, bringing to bear prior work on solvable polynomial maps to general programs. While no method can generate all polynomial invariants for arbitrary programs, our method establishes its merit through a *monotonicty* result. We have implemented our techniques, and tested them on a suite of benchmarks from the literature. Our experiments indicate our techniques show promise on challenging verification tasks requiring non-linear reasoning.

CCS Concepts: • **Theory of computation** → *Logic and verification*; **Automated reasoning**; **Abstraction**; • **Mathematics of computing** → *Gröbner bases and other special bases.*

Additional Key Words and Phrases: Algebraic program analysis, polynomial invariants, monotone program analysis

ACM Reference Format:
John Cyphert and Zachary Kincaid. 2024. Solvable Polynomial Ideals: The Ideal Reflection for Program Analysis. *Proc. ACM Program. Lang.* 8, POPL, Article 25 (January 2024), 29 pages. https://doi.org/10.1145/3632867

1 INTRODUCTION

There has been a long history of prior work that automatically generates polynomial invariants. One line of work in this direction seeks to generate *all* possible polynomial invariants for a restricted class of programs [Hrushovski et al. 2018, 2023; Humenberger et al. 2018; Kovács 2008; Rodríguez-Carbonell and Kapur 2004]. These complete methods give strong, predictable results; however, there is no obvious way to use such techniques for general programs, which may contain nested loops, branching, and unstructured control flow. Another line of research into the automatic generation of polynomial invariants looks to apply to general programs; however, such techniques are often heuristic in nature [Farzan and Kincaid 2015; Kincaid et al. 2018] or are limited on what kind of invariants they can produce [Cachera et al. 2012; Müller-Olm and Seidl 2004; Oliveira et al. 2016; Sankaranarayanan et al. 2004], e.g. returning only polynomials up to some degree.

It is impossible to fully bridge the gap between these two lines of research. No method can generate all polynomial invariants for general programs. No method can even generate all *linear* invariants for general programs [Müller-Olm and Seidl 2004]. However, the two lines of research

Authors' addresses: John Cyphert, University of Wisconsin-Madison, Madison, WI, USA, jcyphert@wisc.edu; Zachary Kincaid, Princeton University, Princeton, NJ, USA, zkincaid@cs.princeton.edu.

ACM 2475-1421/2024/1-ART25
https://doi.org/10.1145/3632867

raises the question: Can a method that generates polynomial invariants and works for general programs provide *some* guarantee on predictability?

In this paper we present techniques to give a positive answer to the previous question. Our method builds on the *algebraic program analysis* framework [Kincaid et al. 2021]. Within this framework, summaries are created for larger and larger subprograms in a bottom-up manner. The essential challenge is the summarization of loops. In short, summarizing a loop amounts to over-approximating the reflexive transitive closure of a transition formula that describes the loop body. Once an appropriate loop summarization technique is constructed, the algebraic framework can then employ the technique as a subroutine in the analysis of whole programs.

The technique for summarizing loops described in this paper works by *abstracting* a transition formula describing an arbitrary loop body to an object, which we call a *transition ideal*. Informally, a transition ideal is a set of polynomial equations describing the transition relation of a loop body. Checking whether a non-linear transition formula (with an integrality predicate) implies a polynomial equation is undecidable for the standard model; however, Kincaid et al. [2023] developed a theory, **LIRR**, for which it is possible to compute *all* implied polynomial equations. Our work utilizes **LIRR** to extract transition ideals from loop body transition formulas. The extraction of transition ideals is complete for **LIRR** and sound for the standard interpretation. A transition ideal can be generated as a summary of an inner loop, a summary of a program with branches, etc. To summarize the loop, we would like to compute the transitive closure of the extracted transition ideal; however, the dynamics of a transition ideal can be chaotic and difficult to capture. Thus, our key insight is that we need to again abstract the transition ideal to some other object for which we know how to compute invariants. In Section 4 we show how given an arbitrary transition ideal one can compute a *best abstraction* as a *solvable transition ideal*, which we call its *solvable reflection*. A solvable transition ideal is a transition ideal that contains all of the defining polynomials of at least one *solvable polynomial map*, a class of polynomial maps that have been utilized in prior work on complete polynomial invariant generation [Amrollahi et al. 2022; Humenberger et al. 2018; Kovács 2008; Rodríguez-Carbonell and Kapur 2004]. In Section 5 we show that the method of Kauers and Zimmermann [2008] can be generalized to compute the polynomial invariants of a solvable transition ideal. These resulting polynomial invariants can be translated back to a transition formula, which gives a method for summarizing arbitrary loops. Hence, via the algebraic framework we achieve a method to generate polynomial invariants to programs with arbitrary control-flow.

While our method is not complete for arbitrary programs, we can guarantee our method is *monotone*. The exact definition of monotonicity is given in Section 6; informally though, a program analysis is monotone if "more information in yields more information out". That is, improving the precision of a code-fragment, e.g. by strengthening the precondition or adding assumptions, necessarily improves the overall analysis result. Our method is monotone because we do not extract just *some* solvable transition ideal from a loop body, but our method extracts the *best* solvable transition ideal. Another way to understand this result is that while our method is not complete for general programs, it is complete, in a sense, at every loop. Given a summary for a loop body, we will always compute the solvable transition ideal that most closely approximates it. Thus, in the restricted case of a simple loop whose body is described by a solvable polynomial map, our method is complete; and, in general our method is monotone.

Summarizing, this paper presents a program analyzer that (1) produces *non-linear summaries*, (2) works for polynomial programs with *arbitrary control-flow*, and (3) is *monotone*. We have implemented our method and our experiments show that it performs comparably to the top performers on the `c/ReachSafety-Loops` subcategory of the Software Verification Competition.

Our paper makes the following contributions:

(1) We introduce transition ideals and solvable transition ideals. We further generalize solvable transition ideals with *ultimately* solvable transition ideals.
(2) We show that transition ideals admit (ultimately) solvable *reflections*.
   - We present algorithms for computing solvable linear reflections (Section 4.1) and ultimately solvable reflections (Section 4.2) of transition ideals. Linear reflections correspond to *best abstractions* with respect to linear simulations.
   - We generalize this method to compute best abstractions with respect to *polynomial simulations* of bounded degree (Section 4.3).
(3) We present a complete algorithm for computing *all* the polynomial invariants of (ultimately) solvable transition ideals.
   - Our summarization algorithm utilizes a sub-algorithm that might be of independent interest. Our sub-algorithm generalizes the technique of Kauers and Zimmermann [2008], which computes the algebraic relations of c-finite sequences of rational numbers, to compute algebraic relations of c-finite sequences over an arbitrary $\mathbb{Q}$-algebra (Problem 5.1).
(4) An implementation of the combination of the abstraction and transitive closure results yields a *monotone* program analysis that produces polynomial invariants for polynomial programs.

The rest of the paper is organized as follows. Section 2 illustrates the main features of our method on a challenging example. Section 3 gives background on commutative algebra, polynomial ideals, and solvable polynomial maps. Section 4 describes the method of extracting (ultimately) solvable transition ideals from arbitrary transition ideals. Section 5 describes the method of summarizing (ultimately) solvable transition ideals. Section 6 connects these ideas to transition formulas, and shows how the methods can be integrated into a program analyzer. Section 7 presents the experimental evaluation. Section 8 discusses related work. Omitted proofs can be found in extended version of this document [Cyphert and Kincaid 2023a].

2 OVERVIEW

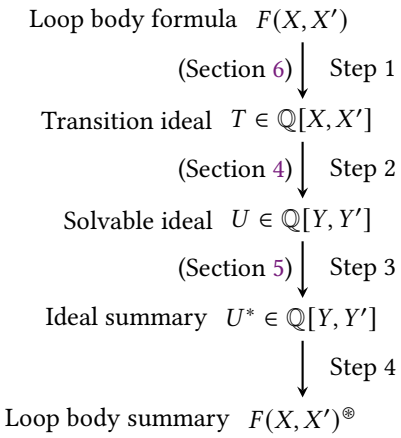
In this section, we present our technique for program verification on the motivating example found in Fig. 1a. Two relevant features of this verification task is that (1) the program has a *nested loop*; and (2) to verify the assertions at the end of the program, an invariant involving *non-linear arithmetic* is required. This combination of nested loops and non-linear arithmetic presents a significant challenge for existing methods.

Our approach to analyzing programs builds on the algebraic program analysis framework [Kincaid et al. 2021]. Within this framework, analysis proceeds by producing *transition formulas* for each program substructure. A transition formula, $F(X, X')$, is a formula over the program variables X as well as their primed counterparts X'. Such a formula represents a relation over program states, where the unprimed variables correspond to the pre-state and the primed variables correspond to the post-state. Summaries for the sequencing and branching of program substructures corresponds to the transition formulas operations $F(X,X') \circ G(X,X') \triangleq \exists X''.F(X,X'') \wedge G(X'',X')$ and $F(X,X') \oplus G(X,X') \triangleq F(X,X') \vee G(X,X')$ respectively. From these two operations, one can accurately summarize non-looping code. For example, a transition formula, F_i for the inner-loop of Fig. 1a would look like $F_i \triangleq c < b \wedge c' = c - b \wedge k' = k + 1 \wedge b' = b$. Of course, we are interested in analyzing programs that do have loops, so an algebraic analysis for looping code must have an iteration operator $F^{\circledast}$. The benefit is that once an iteration operator is created, the analysis can work for any loop, regardless of the underlying program structure.

Figure 1b gives an overview of our iteration operator. We illustrate the method by discussing the analysis of the inner-loop of Fig. 1a. The goal of Step 1 (discussed in Section 6) is to extract

```
1   a = x; b = y; p = 1; q = 0;
2   r = 0; s = 1; c = 0; k = 0;
3   while (b != 0) {
4    c = a; k = 0;
5    while (c >= b) {
6     c = c − b;
7     k = k + 1;
8    }
9    a = b;
10   b = c;
11   p, q = q, p − q ∗ k;
12   r, s = s, r − s ∗ k;
13  }
14  assert(q∗x + s∗y == 0);
15  assert(p∗x + r∗y == a);
16
```

(a) This program implements the extended Euclidean algorithm. The program is a modified version of the verification task egcd2-ll.c (https://github.com/sosy-lab/sv-benchmarks/blob/master/c/nla-digbench/egcd2-ll.c). The assignments on lines 11 and 12 are parallel assignments.



(b) Overview of the method

Fig. 1. Motivating example and overview

a *transition ideal* from the loop body. Informally, transition ideal T corresponds to a transition formula that can be expressed as a conjunction of polynomial equations. That is, the transition ideal of a transition formula $F(X, X')$ is the set $\mathbf{I}(F) = \{p \in \mathbb{Q}[X, X'] : F \models p = 0\}$ of polynomials that vanish on all models of F. For example, for the transition formula for the inner-loop body, we have $T_i = \mathbf{I}(F_i) = \mathbf{I}(c < b \wedge c' = c - b \wedge k' = k + 1 \wedge b' = b) = \langle c' - c + b, k' - k - 1, b' - b \rangle$.

The objective of Step 2 (presented in Section 4) of our method is to extract a *solvable transition ideal* from T_i. We say that a transition ideal U is solvable if it contains a solvable polynomial map p (a homomorphism $p : \mathbb{Q}[X] \to \mathbb{Q}[X]$ of a particular form, defined in Section 3.3), in the sense that $x' - p(x)$ belongs to U for all variables x. Such a p is called a *solvability witness* for U. For the transition ideal T_i, the extraction step is trivial because T_i itself is solvable: the function p_i mapping $\{c \mapsto c - b, k \mapsto k + 1, b \mapsto b\}$ (which is affine, a special case of solvable) is a witness. Therefore, the result of the second step of our method is the solvable transition ideal $U_i = T_i$.

The task of Step 3 (presented in Section 5) of our method is to "summarize" the ideal U_i, as the transition ideal $U_i^* = \bigcap_{n=0}^{\infty} U_i^n$. Thinking of U_i^n as a set of polynomial constraints that hold after n iterations of U_i, U_i^* represents the constraint that hold after *any* number of iterations. The process of computing U^* from a solvable transition ideal U is the subject of Section 5. The basic idea is that we can "solve" the solvable witness p by deriving a closed-form $p^n(x)$ for each $x \in X$. In the case of the inner-loop of Fig. 1a, we have $p_i^n(c) = c - bn, p_i^n(k) = k + n, p_i^n(b) = b$. This solution represents the value of the program variables c, k, and b after n iterations of the loop. We can then obtain polynomial invariants by eliminating n. For our running example, we have $U_i^* = \langle c' - c + b(k' - k), b' - b \rangle$. Since U_i happens to coincide with $\langle c - p_i(c), k - p_i(k), b - p_i(b) \rangle$, computing U_i^* is essentially the same process as Kauers and Zimmermann [2008]; Kovács [2008]'s

complete invariant generation for solvable polynomial maps; Section 5 shows how these ideas can be extended to solvable transition ideals in general.

The final step of our iteration operator (Step 4) is to translate U_i^* back to a transition formula, $F(X, X')^{\circledast}$. For the inner-loop of Fig. 1a, the transition ideal U_i^* translates to the transition formula $F_i^{\circledast} \triangleq c' - c - b(k' - k) = 0 \wedge b' - b = 0$. $F_i^{\circledast}$ is our summary for the inner loop.

The example analysis of the inner loop of Fig. 1a gives the basic outline of how our method analyzes a loop. However, because the body of the inner loop implements a solvable polynomial map, Step 2 of Fig. 1b was trivial. To understand the general case when a loop's body is *not* described by a solvable polynomial map, consider the outer loop of Fig. 1b. Let F_o be a transition formula describing the outer-loop body, and let T_o be a transition ideal obtained from F_o. T_o contains many polynomials that do *not* represent solvable assignments. For example, because the result of analysis of the inner loop $F_i^{\circledast}$ is an approximation of the inner loop, the variable k is updated non-deterministically in T_o; i.e., there is no $k' - p \in T_o$ for any polynomial p. Furthermore, q has a non-linear self-dependence, i.e. $q' - p + qk' \in T_o$, which is not solvable. These complications mean that we cannot capture the dynamics of the variables of the outer loop using solvable polynomial maps.

However, we can find some *terms* that evolve predictably. For example, b and c are always equal in the post-state, i.e. $b' - c' \in T_o$, and the sign of $qr - ps$ flips between the pre-state and post-state, i.e. $(q'r' - p's') + (qr - ps) \in T_o$. The evolution of these terms can be represented with a solvable transition ideal. Figure 2a illustrates how the evolution of these terms for a single loop iteration can be represented by the solvable transition ideal U_o'. Because U_o' represents terms that are in T_o,

$$\left\langle \left\{ \begin{array}{ll} d & \mapsto qr - ps \\ e & \mapsto b - c \end{array} \right\}, \left(\begin{array}{l} d' + d \\ e' \end{array} \right) \right\rangle \qquad \left(\begin{array}{l} (q'r' - p's') + (qr - ps) \\ (b' - c') \end{array} \right)$$

(a) A solvable abstraction, $\langle u, U_o' \rangle$, of T_o (b) Image of U_o' under $\overline{u}$

Fig. 2. An abstraction of the outer-loop transition ideal

the pair $\langle u, U_o' \rangle$ *abstracts* T_o. Let Y be the set of variables $\{d, e\}$ and recall X is used for the set of program variables. The variable d in U_o' represents the polynomial term $qr - ps$ and the variable e represents the linear term $b - c$. This connection between the variables of U_o' and the terms of T_o is captured by the *simulation* $u : \mathbb{Q}[Y] \to \mathbb{Q}[X]$, the polynomial homomorphism defined by $u(d) = qr - ps$ and $u(e) = b - c$. $\langle u, U_o' \rangle$ is a sound abstraction of T_o in the sense that $\overline{u}[U_o']$ is contained in T_o, where $\overline{u}[U_o']$ denotes the image of U_o' under the homomorphism u extended to "primed" vocabulary by defining $\overline{u}(d) = u(d)'$ and $\overline{u}(e) = u(e)'$.

While $\langle u, U_o' \rangle$ is *an* abstraction of T_o, there could be other abstractions of T_o that are better. For example, there are other polynomial terms that behave predictably that are not in $\overline{u}[U_o']$, e.g. $(a's' - c'r') + (as - cr) - (br - cr) \in T_o$. Other abstractions may consider terms that are not captured by $\langle u, U_o' \rangle$. However, the techniques of Section 4 does not just extract *a* sound abstraction, but actually extracts a *best abstraction*, with respect to a class of simulations. We call such a best abstraction a *solvable reflection*[1]. Informally, a solvable reflection is best in that any other abstraction also abstracts the solvable reflection. In Section 4.1 we give an algorithm for producing a solvable reflection with respect to linear simulations. For the case of linear simulations, $\langle v, V \rangle$ is a solvable reflection, with $V = \langle e' \rangle \subseteq \mathbb{Q}[e, e']$, $v(e) = b - c$. In other words, capturing the dynamics of the linear term $b - c$ is the best among all possible abstractions of linear terms with solvable transition ideals.

[1]The name solvable reflection is derived from the notion of a reflective subcategory in category theory.

In Section 4.3, we extend our algorithm for finding linear simulations and give a method for producing solvable reflections with respect to polynomial simulations of a bounded degree. The simulation u from Fig. 2 is an example of a degree-2 simulation, i.e. the mapping for the variable d is a degree-2 polynomial. Our extended method is able to produce the solvable reflection, $\langle t, U_o \rangle$, with respect to degree-2 simulations, of T_o. $\langle t, U_o \rangle$ is too big to be presented here; however, it necessarily captures more dynamics of the outer loop compared to $\left\langle u, U_o' \right\rangle$. Furthermore, for this example, the closure, U_o^*, when combined with the program's initial conditions is strong enough to prove the two assertions at the end of the program, verifying the program in Fig. 1a.

The key that makes the overall process *monotone* is the combination of best abstractions with complete invariant generation for solvable transition ideals. In other words, at every loop we are finding the strongest loop-body invariant that we know how to completely solve. This leads to the result that our iteration operator is monotone (Section 6). Moreover, in the case when the loop body is described by a solvable polynomial map, similar to the case of the analysis of the inner-loop, our method essentially reduces to prior methods. Consequently our method is complete in such a case.

3 BACKGROUND

3.1 Polynomials, Ideals, and Gröbner Bases

We use $\mathbb{Q}[z_1, \ldots, z_n]$ and $\mathbb{Q}[Z]$ to denote the ring of polynomials with rational coefficients over the variables $\{z_1, \ldots, z_n\} = Z$. A **polynomial homomorphism** is a ring homomorphism $f : \mathbb{Q}[X] \to \mathbb{Q}[Y]$ between two polynomial rings. Provided that X is finite, a polynomial homomorphism can be represented by its action on the variables X. We say that f is **linear** if for each $x \in X$, $f(x)$ is either 0 or a homogeneous polynomial of degree 1. We say that f is a **polynomial endomorphism** if $X = Y$. In this paper, every polynomial ring we consider is over a finite set of variables.

Next, we highlight standard definitions for polynomial ideals. For a more in depth presentation of these topics, Cox et al. [2015] provides a good introduction. A polynomial **ideal** $I \subseteq \mathbb{Q}[Z]$ is a set that contains 0, is closed under addition, and for any $p \in I$ and $q \in \mathbb{Q}[Z]$, $pq \in I$. Intuitively, one can consider an ideal I a collection of polynomial equations $\{p = 0 : p \in I\}$. The conditions of an ideal can be read as inference rules: $0 = 0$, if $p = 0$ and $q = 0$ then $p + q = 0$, and if $p = 0$ then $pq = 0$. For any collection of polynomials $P \subseteq \mathbb{Q}[Z]$, we use $\langle P \rangle \triangleq \{g_1 p_1 + \cdots + g_l p_l : p_i \in P, g_i \in \mathbb{Q}[Z]\}$ to denote the **ideal generated by** P.

A **monomial** m is a product of variables of the form $m = z_1^{d_1} \ldots z_n^{d_n}$. The **total degree** of m is $d_1 + \cdots + d_n$. A **monomial order**, $\ll$, is a total ordering on monomials, such that for any monomial v, $1 \ll v$ and if $m \ll n$ then $mv \ll nv$. The **leading monomial**, $\mathrm{LM}(p)$, with respect to a given monomial order, of a polynomial $p = a_1 m_1 + \cdots + a_n m_n$ is the greatest monomial among $m_1, \ldots, m_n$. In this paper, we make use of two different types of monomial orders: *graded orders* and *elimination orders*. Graded orders first compare monomials by total degree, with larger degree corresponding to a larger monomial; ties in total degree are broken by some other monomial order. For example, a graded order that breaks ties using a lexicographic ordering on monomials is the **graded lexicographic order**. Let $X \cup Y$ be a partition of the variables Z. Let $m = m_x m_y$ and $n = n_x n_y$ be monomials with m_x and n_x containing only X variables, and m_y and n_y only containing Y variables. Let $\ll$ be some monomial order. The **elimination order** $\ll_X$ defines $m \ll_X n$ as either (1) $m_x \ll n_x$ or (2) $m_x = n_x$ and $m_y \ll n_y$.

Example 3.1. Consider monomials over the variables x, y, and z with x lexicographically greater than y and y lexicographically greater than z. Let $\ll_{\text{grlex}}$ be the graded lexicographic order, and let $\ll_{\{z\}}$ be the elimination order that eliminates z and uses $\ll_{\text{grlex}}$ for remaining comparisons.

- $z^2 \ll_{\text{grlex}} x^2 \ll_{\text{grlex}} x^2 z^2 \ll_{\text{grlex}} xy^2 z$
- $x^2 \ll_{\{z\}} xy^2 z \ll_{\{z\}} z^2 \ll_{\{z\}} x^2 z^2$

Fixing a monomial ordering, $\ll$, every polynomial ideal $I \subseteq \mathbb{Q}[X]$ admits a finite **Gröbner basis** $G \subseteq \mathbb{Q}[X]$, with $\langle G \rangle = I$. The exact definition of a Gröbner basis is unimportant for this paper. Instead we note the relevant properties of Gröbner bases that we need. Given a Gröbner basis, $G = \{g_1, \ldots, g_k\}$ (w.r.t $\ll$) for an ideal I, every polynomial $p \in \mathbb{Q}[X]$ can be written with respect to G as $p = c_1 g_1 + \cdots + c_k g_k + r$, with $c_1, \ldots, c_k, r \in \mathbb{Q}[X]$ such that:

- r is the unique polynomial with: (1) no term of r is divisible by any $\mathrm{LM}(g_1), \ldots, \mathrm{LM}(g_k)$, and (2) there is a $g \in I$ such that $p = g + r$. Consequently, r has the property that $\mathrm{LM}(r) \ll \mathrm{LM}(r')$ for any $r' \in \mathbb{Q}[X]$ and $g' \in I$ with $p = g' + r'$.
- $\mathrm{LM}(c_i g_i) \ll \mathrm{LM}(p)$ for $i = 1, \ldots, k$ and $\mathrm{LM}(r) \ll \mathrm{LM}(p)$.

Given a finite set of polynomials P there are algorithms [Buchberger 1976; Faugère 1999] for computing a Gröbner basis of $\langle P \rangle$. Furthermore, given a Gröbner basis, G, there are algorithms for rewriting a polynomial by G.

Example 3.2. Consider the graded lexicographic order over the variables x, y, and z. We will not go through the steps to calculate it but $\{z^2 - 1, x - 2, y + z\}$ is a Gröbner basis for the ideal $\left\langle y^2 - 1, yz + 1, xz^2 - 2 \right\rangle$. Thus, $\left\langle z^2 - 1, x - 2, y + z \right\rangle = \left\langle y^2 - 1, yz + 1, xz^2 - 2 \right\rangle$.

$xz^2 + x^2 - y$ can be written with respect to the Gröbner basis $\left\langle y^2 - 1, yz + 1, xz^2 - 2 \right\rangle$ as

$$xz^2 + x^2 - y = x \underbrace{(z^2 - 1)}_{g_1} + (x + 3) \underbrace{(x - 2)}_{g_2} - \underbrace{(y + z)}_{g_3} + \underbrace{z + 6}_{r} .$$

The combination of Gröbner bases and elimination orderings result in the **key property of elimination orderings**: Let X and Y be disjoint sets of variables, and let $G \subseteq \mathbb{Q}[X, Y]$ be a Gröbner basis for $\langle G \rangle$ w.r.t. $\ll_X$. Then $\langle G \cap \mathbb{Q}[Y] \rangle = \langle G \rangle \cap \mathbb{Q}[Y]$. This key property is critical to many of our algorithms and arguments.

Example 3.3. With respect to the graded lexicographic order over the variables x and y, $G = \{x^3 - y^2 - 1, y^3 - 4x^2 + y, xy - 4\}$ is a Gröbner basis for the ideal $\langle G \rangle$. However, with respect to the elimination order that eliminates x, $\ll_{\{x\}}$, $G' = \{16x - y^4 - y^2, y^5 + y^3 - 64\}$ is a Gröbner basis for the ideal $\langle G \rangle$. By the key property of elimination orderings $\langle G \rangle \cap \mathbb{Q}[y] = \langle G' \rangle \cap \mathbb{Q}[y] = \langle G' \cap \mathbb{Q}[y] \rangle = \left\langle y^5 + y^3 - 64 \right\rangle$. Informally, $\left\langle y^5 + y^3 - 64 \right\rangle$ is the ideal $\langle G \rangle$ with the variable x "projected out".

Let X and Y be finite set of variables, let $P \subseteq \mathbb{Q}[X]$ be a set of polynomials, and let $f : \mathbb{Q}[Y] \to \mathbb{Q}[X]$ be a polynomial homomorphism. Then the inverse image $f^{-1}[\langle P \rangle] \triangleq \{q \in \mathbb{Q}[Y] : f(q) \in \langle P \rangle\}$ of $\langle P \rangle$ under f is an ideal of $\mathbb{Q}[Y]$, and it can be computed as follows. Without loss of generality, we assume X and Y are disjoint. Let G be a Gröbner basis for the ideal generated by $P \cup \{y - f(y) : y \in Y\}$, with respect to an elimination order $\ll_X$. Define $\mathit{inv.image}(f, P) \triangleq G \cap \mathbb{Q}[Y]$.

Lemma 3.1 (Inverse image)**.** Let X and Y be finite sets of variables, let $P \subseteq \mathbb{Q}[X]$ be a set of polynomials, and let $f : \mathbb{Q}[Y] \to \mathbb{Q}[X]$ be a polynomial homomorphism. Then we have

$$\langle \mathit{inv.image}(f, P) \rangle = f^{-1}[\langle P \rangle] .$$

Example 3.4. Let $X = \{x, y\}$ and $Y = \{a, b\}$. Let $f : \mathbb{Q}[Y] \to \mathbb{Q}[X]$ be the polynomial homomorphism defined by $f(a) = x^2 + y^2$ and $f(b) = xy$, and let $P \subseteq \mathbb{Q}[X]$ be the set $\{x + y + 1\}$. A Gröbner basis for the ideal generated by $\{x + y + 1\} \cup \{a - (x^2 + y^2), b - xy\}$, with respect to an elimination order $\ll_X$ is $G = \{a + 2b - 1, b + y^2 + y, x + y + 1\}$. $G \cap \mathbb{Q}[Y] = \{a + 2b - 1\}$. Thus, by Lemma 3.1, $\langle a + 2b - 1 \rangle = f^{-1}[\langle x + y + 1 \rangle]$. We can easily verify one direction of the equality by observing that $f(a + 2b - 1) = x^2 + y^2 + 2xy - 1 = (x + y - 1)(x + y + 1) \in \langle x + y + 1 \rangle$.

Given two ideals $\langle I \rangle = A \subseteq \mathbb{Q}[X]$ and $\langle J \rangle = B \subseteq \mathbb{Q}[X]$, $A \cap B$ is also an ideal of $\mathbb{Q}[X]$. A basis for $A \cap B$ can be computed from I and J using Gröbner basis techniques. $A + B \subseteq \mathbb{Q}[X]$

is an ideal and represents the set $\{a + b : a \in A, b \in B\}$. $A + B$ is the smallest ideal containing A and B, and $A + B = \langle I \cup J \rangle$. For any ideal $I \subseteq \mathbb{Q}[X]$ and polynomial $p \in \mathbb{Q}[X]$, we denote the set $\{q : p - q \in I\}$ (equivalently, $\{p + q : q \in I\}$) as $p + I$. We use $\mathbb{Q}[X]/I$ to denote the ring with carrier $\{p + I : p \in \mathbb{Q}[X]\}$, with addition and multiplication lifted to sets.

3.2 Commutative Algebra

Define a $\mathbb{Q}$-**algebra** to be a commutative algebra over $\mathbb{Q}$; that is, an algebraic structure that is both a commutative ring and a linear space over $\mathbb{Q}$. Examples of $\mathbb{Q}$-algebras include $\mathbb{Q}$ itself, the field of algebraic numbers $\bar{\mathbb{Q}}$, $\mathbb{Q}[X]$, and $\mathbb{Q}[X]/I$ for any set of variables X and ideal $I \subseteq \mathbb{Q}[X]$. For any set of variables X, a $\mathbb{Q}$-algebra A defines an algebra homomorphism $(-)^A : \mathbb{Q}[X] \to (A^X \to A)$, where $x^A(v) = v(x)$.[2] For any set $S \subseteq A^X$, define the vanishing ideal of S to be

$$\mathbf{I}_A(S) \triangleq \left\{ p \in \mathbb{Q}[X] : p^A(v) = 0 \text{ for every } v \in S \right\} .$$

Observe that for any polynomial endomorphism $f : \mathbb{Q}[X] \to \mathbb{Q}[X]$ and any $\mathbb{Q}$-algebra A, f defines a function $f_A : A^X \to A^X$ by $f_A(v) = \lambda x. f(x)^A(v)$. For example, let $A = \mathbb{Q}[w]$, $X = \{x, y\}$, and $v = \{x \mapsto 2, y \mapsto w\} \in A^X$. Then let $f : \mathbb{Q}[x, y] \to \mathbb{Q}[x, y]$ be the polynomial homomorphism defined by $f(x) = 2x$ and $f(y) = y + 1$. Then, $f_A(v) = \{x \mapsto 4, y \mapsto w + 1\}$.

Note that for any $\mathbb{Q}$-algebra A, the set of infinite sequences over A, A^ω, is also a $\mathbb{Q}$-algebra. The multiplication and addition operations of A^ω are defined pointwise. Let 0_A and 1_A be the additive and multiplicative unit of A, then $\{0_A\}_{n=0}^\infty$ and $\{1_A\}_{n=0}^\infty$ are the additive and multiplicative unit of A^ω. The scalar multiplication operation of A^ω is defined as applying the scalar multiplication of A to each element of the infinite sequence.

For a $\mathbb{Q}$-algebra A, and a set $G \subseteq A$, we use $span(G)$ to denote the smallest subspace of A that contains G, and $alg(G)$ to denote the smallest $\mathbb{Q}$-subalgebra of A that contains G.

3.3 C-finite Recurrences

Let A be a $\mathbb{Q}$-algebra. A sequence $\{a(n)\}_{n=0}^\infty \in A^\omega$ is **c-finite** if it satisfies a c-finite recurrence. A **c-finite recurrence** has the form:[3]

$$a(n) = c_1 a(n-1) + \cdots + c_d a(n-d) \tag{1}$$

for constants $c_i \in \mathbb{Q}$, for all $n \geq d$. Given a recurrence of the form from Eq. (1) the **order** of the recurrence is d. The **characteristic polynomial** of a c-finite recurrence of the form from Eq. (1) is $p(x) = x^d - c_1 x^{d-1} - \cdots - c_{d-1} x - c_d \in \mathbb{Q}[x]$. The Fibonacci sequence, $\{F(n)\}_{n=0}^\infty$, is a classical example of a c-finite sequence over the $\mathbb{Q}$-algebra $\mathbb{Q}$, which satisfies the order 2 recurrence $F(n) = F(n-1) + F(n-2)$. The characteristic polynomial of the Fibonacci recurrence is $p_{\text{Fib}}(x) = x^2 - x - 1$.

Every c-finite recurrence admits a closed-form as a polynomial-exponential [Everest et al. 2003]. More specifically, given a recurrence of the form from Eq. (1) with $c_i \in \mathbb{Q}$ and d initial values $a(0), \ldots, a(d-1)$ from some $\mathbb{Q}$-algebra A, then

$$a(n) = \sum_{i=1}^{d} \left(z_i(n) + \sum_{j=1}^{d} p_{ij}(n) \Theta_j^n \right) a(i-1) \tag{2}$$

[2] $\mathbb{Q}[X]$ is the free $\mathbb{Q}$-algebra generated by X; $(-)^A$ is the "evaluation" function that we get from freeness.

[3] Here we present c-finite recurrences in *homogeneous* form. *Inhomogeneous* c-finite recurrences can have an additional additive constant c_{d+1} at the end of Eq. (1). Any inhomogeneous c-finite recurrence can be transformed to a homogeneous c-finite recurrence of order 1 higher, and so no power is lost when only considering the homogenous form.

where each Θ_i is a complex root of the characteristic polynomial of the recurrence, each $p_{ij} \in \bar{\mathbb{Q}}[x]$[4], and $z_i(n) : \mathbb{N} \to \mathbb{Q}$ with $z_i(k) = 0$ for any $k \geq d$. More specifically, $z_i(k) = 0$ for any k greater than or equal to the multiplicity of 0 as a root of the characteristic polynomial. If 0 is not a root of the characteristic polynomial then the terms $z_i(k) = 0$ for all $k \in \mathbb{N}$ and can be omitted from the closed-form. Determining such a closed-form from a recurrence is referred to as "solving" the recurrence. The roots of the Fibonacci characteristic polynomial $x^2 - x - 1$ are $\phi = \frac{1+\sqrt{5}}{2}$ and $\psi = \frac{1-\sqrt{5}}{2}$. Assuming, $F(0) = 0$ and $F(1) = 1$, a solution to the Fibonacci recurrence in the form of Eq. (2) is Binet's formula $F(n) = \frac{1}{\sqrt{5}}\phi^n - \frac{1}{\sqrt{5}}\psi^n$.

A polynomial endomorphism $f : \mathbb{Q}[X] \to \mathbb{Q}[X]$ is **solvable** if there exists a partition $X = X_1 \cup \cdots \cup X_n$ of X (with $X_i \cap X_j = \emptyset$ for all $i \neq j$) such that for each X_i and each $x \in X_i$, $f(x)$ can be written as $g(X_i) + h(X_1, \ldots, X_{i-1})$, where g is a linear polynomial in the variables X_i and h is a polynomial (of arbitrary degree) in the variables $X_1 \cup \cdots \cup X_{i-1}$.

C-finite recurrences are equivalent to solvable polynomial maps in the sense that each solvable polynomial map $f : \mathbb{Q}[X] \to \mathbb{Q}[X]$ defines $|X|$ c-finite sequences $\{f^i(x_1)\}_{i=0}^{\infty}, \ldots, \{f^i(x_n)\}_{i=0}^{\infty}$, each of order $|X|$ [Kincaid et al. 2018, Section 8]. Conversely, each c-finite recurrence $a(n) = c_1 a(n-1) + \cdots + c_d a(n-d)$ can be transformed to a solvable map f over d variables, $a_n, \ldots, a_{n-d}$ as the homomorphism defined by $f(a_n) = c_1 a_{n-1} + \cdots + c_d a_{n-d}$ and $f(a_{n-i}) = a_{n-i+1}$ for $0 < i \leq d$. Due to this equivalence, solvable polynomial maps can effectively be "solved" in the form of Eq. (2) in the same way as c-finite recurrences. Hence the name *solvable* polynomial map.

3.4 Transition Formulas and Linear Integer/Real Rings

Fix a set of program variables X. We use $X' = \{x' : x \in X\}$ to denote a set of "primed" copies of variables in X (presumed disjoint from X). We use $(-)'$ to denote the homomorphism $\mathbb{Q}[X] \to \mathbb{Q}[X']$ that maps each x to its primed copy x'.

A **transition formula** is a formula F with free variables in X and X' (in some first-order language), with the unprimed variables representing the pre-state of some computation, and the primed variables representing the post-state. For transition formulas F_1 and F_2, we use $F_1 \circ F_2 \triangleq \exists X''. F_1[X' \mapsto X''] \wedge F_2[X \mapsto X'']$ to denote the sequential composition of F_1 and F_2. For any transition formula F and natural number n, we use $F^n \triangleq F \circ \cdots \circ F$ (n times) to denote the n-fold sequential composition of F with itself.

Within this paper, we shall assume that transition formulas are expressed in the existential fragment of the language of non-linear mixed integer/real arithmetic (that is, the language of rational constants, addition, multiplication, an order relation, and an integrality predicate). Although this language is undecidable over the *standard* model, Kincaid et al. [2023] showed that ground satisfiability is decidable if we allow more general interpretations, namely over linear integer/real rings (**LIRR**). For our purposes, we may think of linear integer/real rings as $\mathbb{Q}$-algebras that satisfy some additional axioms concerning the order relation and integrality predicate, which are not relevant to this paper. We will assume **LIRR** as a background theory in the remainder of the paper, and use $F \models_{\mathbf{LIRR}} G$ to denote that the formula F entails the formula G modulo **LIRR**.

In addition to satisfiability being decidable, there is a procedure [Kincaid et al. 2023] for computing the vanishing ideal $\mathbf{I}_{\mathbf{LIRR}}(F)$ of a formula F in the existential fragment of the language: the ideal of all polynomials p such that $F \models_{\mathbf{LIRR}} p = 0$. See Example 6.1 for an example of $\mathbf{I}_{\mathbf{LIRR}}(F)$ from a formula F. For any ideal I generated by polynomials $p_1, \cdots, p_n$, we use $\mathbf{F}(I)$ to denote the formula $p_1 = 0 \wedge \cdots \wedge p_n = 0$. The choice of generators for I is irrelevant in the sense that if two sets of polynomials P and Q generate the same ideal, then $\bigwedge_{p \in P} p = 0$ and $\bigwedge_{q \in Q} q = 0$ are equivalent

[4]More specifically, each polynomial p_{ij} has coefficients in the splitting field of the characteristic polynomial, $\mathbb{Q}(\Theta_1, \ldots, \Theta_d)$

modulo **LIRR**. Note that $\mathbf{I}_{\mathbf{LIRR}}$ and $\mathbf{F}$ form a Galois connection: for any transition formula F and ideal I over the free variables of F, we have $F \models_{\mathbf{LIRR}} \mathbf{F}(I)$ if and only if $\mathbf{I}_{\mathbf{LIRR}}(F) \supseteq I$. This implies that (1) for formulas F and G, if $F \models_{\mathbf{LIRR}} G$, then $\mathbf{I}_{\mathbf{LIRR}}(F) \supseteq \mathbf{I}_{\mathbf{LIRR}}(G)$, and (2) for ideals I and J if $I \supseteq J$, then $\mathbf{F}(I) \models_{\mathbf{LIRR}} \mathbf{F}(J)$.

3.5 Transition Ideals

The main results of this paper are concerned with *transition ideals*. A **transition ideal** is an ideal in the ring $\mathbb{Q}[X, X']$ for some set of variables X. Transition ideals are not tied to the theory **LIRR**, but can be seen as the vanishing ideals of transition formulas, and their operations can be understood in terms of corresponding operations on transition formulas.

For transition ideals T_1 and T_2, define

$$T_1 \cdot T_2 \triangleq (T_1[X' \mapsto X''] + T_2[X \mapsto X'']) \cap \mathbb{Q}[X, X'] \,.$$

Equivalently, $T_1 \cdot T_2$ is equal to $\mathbf{I}_{\mathbf{LIRR}}(\mathbf{F}(T_1) \circ \mathbf{F}(T_2))$. For any transition ideal T and natural number n, define

$$T^0 = \langle \{x' - x : x \in X\} \rangle \qquad T^n \triangleq \underbrace{T \cdot \ldots \cdot T}_{n \text{ times}} \qquad T^* \triangleq \bigcap_{i=0}^{\infty} T^i.$$

Note that, since the polynomials in a transition ideal are interpreted as *constraints*, T^* can be interpreted as the set of constraints that are common to all T^i. In particular, if F is a transition formula, then for any $n \in \mathbb{N}$, we have $F^n \models_{\mathbf{LIRR}} \mathbf{F}((\mathbf{I}_{\mathbf{LIRR}}(F))^*)$.

Example 3.5. $T = \left\langle w' - wy, x' - 2x - y^2, y' - y - z, z' - 3z, y - z - 1 \right\rangle$ is an example of a transition ideal. $T^0 = \langle w' - w, x' - x, y' - y, z' - z \rangle$, $T^1 = T$, and

$$\begin{aligned} T^2 &= \langle w' - wy(y+z), x' - 2(2x + y^2) - (y+z)^2, y' - (y+z) - 3z, z' - 3(3z), \\ &\qquad (y+z) - 3z - 1, y - z - 1 \rangle \\ &= \langle w' - wy^2 - wz, x' - 4x - 3y^2 - 2yz - z^2, y' - y - 4z, z' - 9z, y - 2z - 1, y - z - 1 \rangle \\ &= \langle w' - w, x' - 4x - 3, y' - 1, z', y - 1, z \rangle \end{aligned}$$

Define the **domain** of T to be $\mathrm{dom}(T) \triangleq T \cap \mathbb{Q}[X]$, and the **invariant domain** of T to be $\mathrm{dom}^*(T) = \sum_{n=0}^{\infty} \mathrm{dom}(T^n)$. Informally, if we think of a transition ideal as a set of polynomial equations constraining the transition between a pre-state X and post-state X', then the domain of T is the set of constraints that must be satisfied by a pre-state in order to have a successor. Pre-states that satisfy the invariant domain of T are those states which have arbitrarily long computations described by T. Given a transition ideal T, $\mathrm{dom}^*(T)$ can be calculated as follows. By inspection of the definition of $T^n \cdot T$, we see $\mathrm{dom}(T^n) \subseteq \mathrm{dom}(T^{n+1})$ for any $n \geq 1$. Therefore, we have the ascending chain of ideals:

$$\mathrm{dom}(T) \subseteq \mathrm{dom}(T^2) \subseteq \mathrm{dom}(T^3) \subseteq \ldots.$$

By Hilbert's basis theorem this chain must stabilize at some N. That is, there exists $N \geq 1$ such that $\mathrm{dom}(T^N) = \mathrm{dom}(T^{N+1}) = \mathrm{dom}^*(T)$.

We say that T is **solvable** if there is a solvable $p : \mathbb{Q}[X] \to \mathbb{Q}[X]$ such that $x' - p(x) \in T$ for all $x \in X$; call p a **solvability witness** for T. We say that T is **ultimately solvable** if $T + \mathrm{dom}^*(T)$ is solvable.

Example 3.6. Recall Example 3.5, with $T = \left\langle w' - wy, x' - 2x - y^2, y' - y - z, z' - 3z, y - z - 1 \right\rangle$. $\mathrm{dom}(T) = \langle y - z - 1 \rangle$. For this example, the invariant domain stabilizes at $N = 2$, and so $\mathrm{dom}(T^2) =$

$\mathrm{dom}^*(T) = \langle y-1, z\rangle$. T is *not* a solvable transition ideal. This is because the dynamics of w, $w \mapsto wy$ cannot be captured by a solvable polynomial map.

$T' = \left\langle x' - 2x - y^2, y' - y - z, z' - 3z, y - z - 1\right\rangle$ is a solvable transition ideal recognized by the solvability witness $p : \mathbb{Q}[x,y,z] \to \mathbb{Q}[x,y,z]$ defined by $p(x) = 2x + y^2$, $p(y) = y + z$, and $p(z) = 3z$. Note that even though there is a non-linear dependence on the variable y for the assignment $p(x)$, p is still solvable using the variable partition $\{y,z\} \cup \{x\}$.

T is an example of an ultimately solvable transition ideal. $T + \mathrm{dom}^*(T) = \langle w' - w, x' - 2x - 1, y' - 1, z', y - 1, z\rangle$ is solvable, and is recognized by the solvability witness $q : \mathbb{Q}[w,x,y,z] \to \mathbb{Q}[w,x,y,z]$ defined by $q(w) = w$, $q(x) = 2x + 1$, $q(y) = 1$, and $q(z) = 0$.

Let $T \subseteq \mathbb{Q}[X, X']$ and $U \subseteq \mathbb{Q}[Y, Y']$ be transition ideals (possibly over different variables). A polynomial homomorphism $s : \mathbb{Q}[Y] \to \mathbb{Q}[X]$ is a **simulation** from T to U (notice reversal of direction) if for every polynomial $p \in U$, we have $\overline{s}(p) \in T$, where $\overline{s} : \mathbb{Q}[Y, Y'] \to \mathbb{Q}[X, X']$ denotes the extension of s to the "doubled" vocabulary, which maps each $x \in X$ to $s(x)$ and each $x' \in X'$ to $s(x)'$. We say that s is a **linear simulation** if it is linear and a simulation. If $T \subseteq \mathbb{Q}[X, X']$, $U \subseteq \mathbb{Q}[Y, Y']$, and $V \subseteq \mathbb{Q}[Z, Z']$ are transition ideals and $s : T \to U$ and $t : U \to V$ are simulations, then their composition $t; s \triangleq s \circ t$ is a simulation from T to V (again noting that simulations go in the opposite direction of polynomial homomorphisms).

4 SOLVABLE REFLECTIONS

In this section, we show that every transition ideal $T \subseteq \mathbb{Q}[X, X']$ admits a *solvable reflection*: there is a solvable transition ideal U and a linear simulation $s : T \to U$ that is a closer approximation of T than any other simulating solvable transition ideal.

Example 4.1. Figure 3 illustrates a transition ideal T that will be used as a running example throughout this section. One may think of T as the polynomial map

$$f(w,x,y,z) = (z^2, 4x(1-x), -4x^2 + 3x + y - 1, z + x^2 - 2xy + y^2)$$

(corresponding to the first four generators of T) restricted to the domain $w - z^2 - 1$ (corresponding to the fifth). The map f is not solvable, since x exhibits a non-linear self-dependence (in fact, $x \mapsto 4x(1-x)$ is a logistic map, a famous example in chaos theory that Ulam and von Neumann [1947] suggested as the basis of a pseudo-random number generator). If we restrict the transition ideal to the variable w, the resulting transition ideal $T_w = \langle w' - w + 1\rangle$ *is* solvable, and we can compute a closed form for its ith iterate: $T_w^i = \langle w' - w + i\rangle$. This is an instance of a *solvable abstraction* of T: T_w is a solvable transition ideal that approximates the dynamics of T, and the nature of the approximation is given by the inclusion homomorphism $\mathbb{Q}[w] \to \mathbb{Q}[w,x,y,z]$ (mapping $w \mapsto w$).

There are other solvable abstractions of T which capture different aspects of T's dynamics. Observe that while it's challenging to understand the dynamics of x or y, their *difference* behaves predictably: T contains the polynomial $(x' - y') - (x - y) + 1$, indicating that the value of $(x - y)$ decreases by 1 at each step. Coincidentally, the dynamics of $(x - y)$ are identical to that of w (both decrease by 1), so this information can be represented as the solvable abstraction $\langle \{w \mapsto x - y\}, T_w\rangle$.

Yet another solvable abstraction $\langle s, U\rangle$ is pictured in Fig. 3. This abstraction is more desirable than either $\langle \{w \mapsto w\}, T_w\rangle$ or $\langle \{w \mapsto x - y\}, T_w\rangle$, in the sense that $\langle s, U\rangle$ captures the dynamics of not only $(x - y)$ and w, but also z. In fact, $\langle s, U\rangle$ is a *solvable reflection* of T, in the sense that any other solvable abstraction of T similarly factors through $\langle s, U\rangle$.

Formally, a **solvable reflection** of $T \subseteq \mathbb{Q}[X, X']$ (with respect to linear simulations) is a pair $\langle s, U\rangle$ consisting of a transition ideal $U \subseteq \mathbb{Q}[Y, Y']$ (for some set of variables Y) and a polynomial homomorphism $s : \mathbb{Q}[Y] \to \mathbb{Q}[X]$ such that:

$$\left(\begin{array}{l} w' - z^2 \\ x' - 4x(1-x) \\ y' + 4x^2 - 3x - y - 1 \\ z' - z - x^2 + 2xy - y^2 \\ w - z^2 - 1 \end{array}\right) \qquad \left\langle \left\{\begin{array}{ll} a & \mapsto (x-y) \\ b & \mapsto w \\ c & \mapsto z \end{array}\right\}, \left(\begin{array}{l} a' - a - 1 \\ b' - b + 1 \\ c' - c - a^2 \\ b - c^2 - 1 \end{array}\right) \right\rangle \qquad \left(\begin{array}{l} (x' - y') - (x - y) + 1 \\ w' - w + 1 \\ z' - z - (x-y)^2 \\ w - z^2 - 1 \end{array}\right)$$

(a) A transition ideal, T (b) A solvable reflection $\langle s, U\rangle$ of T (c) Image of U under $\overline{s}$

Fig. 3. A solvable reflection of a transition ideal

(1) s is a linear simulation from T to U
(2) U is solvable
(3) For any other pair $\langle v, V\rangle$ satisfying 1 and 2, there exists a unique linear simulation $\hat{v} : U \to V$ such that $v = \hat{v}; s$.

Section 4.1 describes a procedure for computing solvable reflections. We then extend this result in two ways: (1) Section 4.2 generalizes from solvable to *ultimately* solvable reflections, and (2) Section 4.3 generalizes from linear simulations to polynomial simulations.

4.1 Computing Solvable Reflections

To begin, we give an alternate characterization of solvable transition ideals, which will serve as the basis of our algorithm for computing solvable reflections. For a transition ideal $T \subseteq \mathbb{Q}[X, X']$ and a set of polynomials $P \subseteq \mathbb{Q}[X]$, define

$$\text{Det}\,(T, P) \triangleq \{p \in \mathit{span}\,(X) : \exists q \in P. p' - q \in T\}$$

Intuitively, $\text{Det}\,(T, P)$ represents the set of linear functionals that are "determined up to P"; i.e., T constrains the post-state value of $p \in \text{Det}\,(T, P)$ to be equal to some $q \in P$. Observe that T is solvable exactly when there is an ascending chain $X_1 \subset X_2 \cdots \subset X_n = X$ of sets of variables (called a *stratification* of X) such that for each $i \in \{1, \ldots, n-1\}$, we have $\mathit{span}\,(X_{i+1}) \subseteq \text{Det}\,(T, \mathit{span}\,(X_{i+1}) + \mathit{alg}\,(X_i))$. (i.e., for each $x \in X_{i+1}$, we have $x' - p - q \in T$ for some linear polynomial $p \in \mathit{span}\,(X_{i+1})$ over the variables X_{i+1} and some polynomial $q \in \mathit{alg}\,(X_i)$ over the variables $X_i = X_1 \cup \cdots \cup X_i$).

Example 4.2. Consider the solvable transition ideal U in Fig. 3b. A solvability witness for U is the polynomial homomorphism f_U that sends $a \mapsto a - 1$, $b \mapsto b + 1$, and $c \mapsto c + a^2$, and the corresponding partition of the variables is $\{a, b\} \cup \{c\}$. The fact that f_U is solvable corresponds precisely to the facts that both $f(a)$ and $f(b)$ belong to $\mathit{span}\,(\{a, b\}) + \mathit{alg}\,(\emptyset)$, and $f(c)$ belongs to $\mathit{span}\,(\{a, b, c\}) + \mathit{alg}\,(\{a, b\})$.

We may derive from this solvability witness a stratification $\{a, b\} \subset \{a, b, c\}$ (i.e., the ith stratum is the union of the first i cells in the ordered partition). Observe that $\text{Det}\,(U, \mathit{span}\,(\{a, b\}) + \mathit{alg}\,(\emptyset)) = \mathit{span}\,(\{a, b\})$, and $\text{Det}\,(U, \mathit{span}\,(\{a, b, c\}) + \mathit{alg}\,(a, b)) = \mathit{span}\,(\{a, b, c\})$.

Furthermore, observe that the simulation $u : T \to U$ induces a corresponding structure in T, namely $\text{Det}\,(T, \mathit{span}\,(\{(x-y), w\}) + \mathit{alg}\,(\emptyset)) = \mathit{span}\,(\{(x-y), w\})$ (where $\{(x-y), w)\} = \{u(a), u(b)\}$) and $\text{Det}\,(U, \mathit{span}\,(\{(x-y), w, z\}) + \mathit{alg}\,((x-y), w)) = \mathit{span}\,(\{(x-y), w, z\})$ (where $\{(x-y), w, z)\} = \{u(a), u(b), u(c)\}$. In fact this stratification structure determines the solvable reflection of T uniquely (up to isomorphism) in the sense that if $v : \mathbb{Q}[d_1, d_2, d_3] \to \mathbb{Q}[w, x, y, z]$ is any polynomial homomorphism such that $\{v(d_1), v(d_2)\}$ is a basis for $\mathit{span}\,((x-y), w)$ and $\{v(d_1), v(d_2), v(d_3)\}$ is a basis for $\mathit{span}\,((x-y), w, z)$, then $\langle v, \overline{v}^{-1}\,[T]\rangle$ is a solvable reflection of T. The essential idea behind the algorithm presented in this section is to calculate this structure by discovering each stratum in turn.

Theorem 4.1. Every transition ideal has a solvable reflection.

Proof. Let $T \subseteq \mathbb{Q}[X, X']$ be a transition ideal. We may calculate the solvable reflection of T as follows. For each natural number i, we define a linear subspace of polynomials $\mathcal{S}_i(T) \subseteq span(X)$ as follows:

- $\mathcal{S}_0(T) \triangleq \{0\}$.
- For each $i \geq 0$, $\mathcal{S}_{i+1}(T)$ is the greatest fixed point of $S \mapsto \text{Det}(T, S + alg(\mathcal{S}_i(T)))$. Such a fixed point always exists by the Knaster-Tarski fixed point theorem, noting that $S \mapsto \text{Det}(T, S + alg(\mathcal{S}_i(T)))$ is a monotone operator on the complete lattice of subspaces of $span(X)$.

Since $\mathcal{S}_0(T) \subseteq \mathcal{S}_1(T) \subseteq \ldots$ is an ascending chain of subspaces of a finite-dimensional space $span(X)$, it must stabilize (i.e., there is some n such that $\mathcal{S}_n(T) = \mathcal{S}_{n+1}(T)$). Call the resulting space $\mathcal{S}^*(T)$.

For any $i \in \mathbb{N}$, let d_i be the dimension of $\mathcal{S}_i(T)$. Choose an ordered basis $p_1, \ldots, p_n$ for $\mathcal{S}^*(T)$ such that for each i, $p_1, \ldots, p_{d_i}$ spans $\mathcal{S}_i(T)$ (such a basis may be obtained by choosing an arbitrary basis for $\mathcal{S}_0(T)$ and then extending it to $\mathcal{S}_1(T)$, then extending that basis to $\mathcal{S}_2(T)$, and so on). Let $Y = \{y_1, \ldots, y_n\}$ be a set of variables disjoint from X, and let $u : \mathbb{Q}[Y] \to \mathbb{Q}[X]$ be the homomorphism that maps $y_i \mapsto p_i$. Finally, define $R(T) \triangleq \langle u, \overline{u}^{-1}[T] \rangle$. We will show that $R(T)$ is a solvable reflection of T.

First, we show that $\overline{u}^{-1}[T]$ is solvable. We construct a solvability witness as follows. For any $i \leq n$, let $s(i)$ be the least number such that $i \leq d_{s(i)}$. For each $1 \leq k \leq n$, let $Y_k = \{y_i : s(i) = k\}$. For any i we have $p_i \in \mathcal{S}_{s(i)}(T)$, and so

$$p_i \in \text{Det}\left(T, \mathcal{S}_{s(i)}(T) + alg\left(\mathcal{S}_{s(i)-1}(T)\right)\right)$$

It follows that there is some $q_i \in \mathcal{S}_{s(i)}(T)$ and $\hat{q}_i \in alg\left(\mathcal{S}_{s(i)-1}(T)\right)$ such that $p_i' - q_i - \hat{q}_i \in T$. Since $u[span(Y_i)] = \mathcal{S}_{s(i)}(T)$ and $u[alg(Y_1 \cup \cdots \cup Y_{i-1})] = alg\left(\mathcal{S}_{s(i)-1}(T)\right)$, there is some $t_i \in span(Y_i)$ and $\hat{t}_i \in alg(Y_1 \cup \cdots \cup Y_{i-1})$ such that $u(t_i) = q_i$ and $u(\hat{t}_i) = \hat{q}_i$. Define $r : \mathbb{Q}[Y] \to \mathbb{Q}[Y]$ to be the polynomial homomorphism that sends $y_i \mapsto t_i + \hat{t}_i$. Observe that r is a solvability witness for $\overline{u}^{-1}[T]$: r is solvable by construction, and for each y_i we have

$$\overline{u}(y_i' - r(y_i)) = \overline{u}(y_i' - t_i - \hat{t}_i) = \overline{u}(y_i') - \overline{u}(t_i) - \overline{u}(\hat{t}_i) = p_i' - q_i - \hat{q}_i \in T$$

and so $y_i' - r(y_i) \in \overline{u}^{-1}[T]$.

Next, we must show that $R(T)$ is universal. Suppose that $V \subseteq \mathbb{Q}[Z, Z']$ is a solvable transition ideal, and that $v : \mathbb{Q}[Z] \to \mathbb{Q}[X]$ is a linear simulation from T to V. We must show that there is a homomorphism $\hat{v} : \mathbb{Q}[Z] \to \mathbb{Q}[Y]$ such that $u \circ \hat{v} = v$ and $\hat{v}$ is a simulation from $\overline{u}^{-1}[T]$ to V.

It is sufficient to show that for each $z \in Z$, we have $v(z) \in \mathcal{S}^*(T)$: under this assumption, for each z, $v(z)$ can be written (uniquely) as a linear combination $v(z) = a_1 p_1 + \cdots + a_n p_n = u(a_1 y_1 + \cdots + a_n y_n)$, and so we can define $\hat{v}$ by $\hat{v}(z) \triangleq a_1 y_1 + \cdots + a_n y_n$. It follows that

$$\overline{\hat{v}}^{-1}[\overline{u}^{-1}[T]] = \overline{(u \circ \hat{v})}^{-1}[T] = \overline{v}^{-1}[T] \supseteq V\,,$$

and so $\hat{v}$ is a simulation from $\overline{u}^{-1}[T]$ to V.

Since U is solvable, there is a partition $Z_1, \ldots, Z_k$ of Z and a polynomial homomorphism $g : \mathbb{Q}[Z] \to \mathbb{Q}[Z]$ such that for each i and each $z \in Z_i$, $g(z)$ can be written as the sum of a linear term in Z_i and a polynomial in $Z_1, ..., Z_{i-1}$. For each $i \in \{0, \ldots, n\}$, let $Z_{\leq i}$ denote $\bigcup_{j=1}^{i} Z_j$. We show by induction on k that for all $z \in Z_{\leq k}$, $v(z) \in \mathcal{S}_k(T)$. The base case $k = 0$ is trivial: $Z_{\leq 0}$ is empty. For the inductive step, suppose $s(Z_{\leq k}) \subseteq \mathcal{S}_k(T)$, and prove $s(Z_{\leq k+1}) \subseteq \mathcal{S}_{k+1}(T)$. Let $z \in Z_{\leq_{k+1}}$. Since g is a witness to solvability of V, we have $z' - g(z) \in V$, and since v is a simulation from T to V, we have $\overline{v}(z' - g(z)) \in T$. Since g is solvable, we have $g(z) \in span(Z_{k+1}) + alg(Z_{\leq k})$, and so

Algorithm 1: Computation of determined functionals

Input: Finite sets of polynomials $T \subseteq \mathbb{Q}[X, X']$, $V \subseteq \mathbb{Q}[X]$, and $Q \subseteq \mathbb{Q}[X]$
Output: Set of polynomials D such that $span(D) = \text{Det}(\langle T \rangle, span(V) + alg(Q))$

```
Y ← {y_v : v ∈ V} ;        /* Introduce a variable y_v for each generator v ∈ V */
Z ← {z_q : q ∈ Q} ;        /* Introduce a variable z_q for each generator q ∈ Q */
Let f : ℚ[X, Y, Z] → ℚ[X, X'] be the map that sends x ↦ x', y_v ↦ v, z_q ↦ q;
F ← inv.image(f, T);
G ← Gröbner basis for F w.r.t. ≪_{X∪Y} for some graded order ≪;
D ← ∅;
foreach g ∈ G do
  if g = p + q + r for some non-zero p ∈ span(X), q ∈ span(Y), and r ∈ ℚ[Z] then
    D ← D ∪ {p};
return D
```

$v(g(z)) \in span(v(Z_{k+1})) + alg(v(Z_{\leq k}))$. Since $\overline{v}(z' - g(z)) = v(z)' - v(g(z)) \in T$ and $v(g(z)) \in span(v(Z_{k+1})) + alg(v(Z_{\leq k}))$, we have $v(z) \in \text{Det}(T, span(v(Z_{k+1})) + alg(v(Z_{\leq k})))$, and so by the induction hypothesis and monotonicity of Det we have $v(z) \in \text{Det}(T, span(v(Z_{k+1})) + \mathcal{S}_k(T))$. Since this holds for all $z \in Z_{k+1}$, we have $span(v(Z_{k+1}) \subseteq \text{Det}(T, span(v(Z_{k+1})) + \mathcal{S}_k(T))$, and since $\mathcal{S}_{k+1}(T)$ is defined to be greatest fixedpoint of $S \mapsto \text{Det}(T, S + \mathcal{S}_k(T))$, we have $span(v(Z_{k+1})) \subseteq \mathcal{S}_{k+1}(T)$, and so for all $z \in Z_{k+1}$, $v(z) \in \mathcal{S}_{k+1}(T)$. □

The proof of Theorem 4.1 is constructive and gives rise to a procedure for computing solvable reflections of transition ideals, depicted in Algorithm 2. The procedure relies on a subroutine for computing Det, which is given in Algorithm 1 (the correctness of which is Lemma 4.2). Otherwise, the procedure follows the steps of the proof directly.

Lemma 4.2. Let $T \subseteq \mathbb{Q}[X, X']$, $V \subseteq \mathbb{Q}[X]$, and $Q \subseteq \mathbb{Q}[X]$ (for some set of variables X). Then Algorithm 1 computes a set of polynomials D such that $span(D) = \text{Det}(\langle T \rangle, span(V) + alg(Q))$.

4.2 Ultimately Solvable Reflections

The algorithm presented in Section 5 reveals that a weaker condition than solvability is sufficient in order to compute the Kleene closure of a transition ideal, namely the transition needs to be *ultimately solvable*. This raises the question of whether it's possible to compute ultimately solvable reflections of arbitrary transition ideals, and thereby obtain a more powerful algorithm for generating polynomial invariants for loops. In this section, we answer that question in the affirmative.

Let T be a transition ideal. Define a sequence $\langle t_0, T_0 \rangle, \langle t_1, T_1 \rangle, \ldots$ where t_0 is the identity function, $T_0 = T$, and for each $i \geq 0$, t_{i+1} is the simulation component of the solvable reflection of $T_i + \text{dom}^*(T_i)$, and T_{i+1} is $\overline{t_{i+1}}^{-1}[T_i]$. Since for all i, if t_i is not invertible then the dimension of T_i is strictly smaller than T_{i-1}, there must be some first index n such that t_n is invertible. Define $R^*(T) \triangleq \left\langle u^*, \overline{u^*}^{-1}[T] \right\rangle$, where $u^* = t_1 \circ \cdots \circ t_{n-1}$.

Lemma 4.3. Let T be a transition ideal. Then $R^*(T)$ is an ultimately solvable reflection of T.

4.3 Polynomial Simulations

Here, we consider a generalization of our definition of (ultimately) solvable reflections, in which the simulation from a transition ideal to its reflection is a polynomial map rather than a linear map.

Algorithm 2: Solvable reflection

Input: Finite set of polynomials $T \subseteq \mathbb{Q}[X, X']$
Output: Solvable reflection of T

$i \leftarrow 0$;
$n \leftarrow 0$;
$S_0 \leftarrow \emptyset$;
repeat
  $n' = n$;
  $S_{i+1} \leftarrow X$;
  /* $\mathsf{span}(S_{i+1})$ is the greatest fixpoint of $X \mapsto \mathsf{Det}(T, \mathsf{span}(X) + \mathsf{alg}(S_i))$ */
  repeat
    $dim \leftarrow |S_{i+1}|$;
    $S_{i+1} \leftarrow \mathsf{Det}(T, span(S_{i+1}) + alg(S_i))$; /* Algorithm 1 */
  until $dim = |S_{i+1}|$;
  foreach $p \in S_{i+1}$ **do**
    if $p \notin span(q_0, \ldots, q_{n-1})$ **then**
      $q_n \leftarrow p$;
      $n \leftarrow n + 1$;
until $n = n'$;
Let u be the map $\mathbb{Q}[y_0, \ldots, y_{n-1}] \to \mathbb{Q}[X]$ mapping $y_i \mapsto q_i$;
return $\langle u, inv.image(\overline{u}, T)\rangle$

Let X be a set of variables and let $d \in \mathbb{N}$ be a fixed degree bound. Let $X^{\leq d}$ be the set of monomials of degree at most d (excluding 1), let Y be a set of variables of cardinality equal to that of $X^{\leq d}$, and let $f_{X,d} : Y \to X^{\leq d}$ be a bijection. Observe that if Z is a set of variables and $g : \mathbb{Q}[Z] \to \mathbb{Q}[X]$ is a polynomial homomorphism of degree at most d, then there is unique *linear* polynomial homomorphism $\hat{g}$ such that $g = \hat{g} \circ f_{X,d}$. As a result, we can reduce the problem of computing reflections with respect to bounded-degree polynomial simulations to the problem of computing reflections with respect to linear simulations:

Lemma 4.4. Let $T \subseteq \mathbb{Q}[X, X']$ be a transition ideal, and let $d \in \mathbb{N}$ be a fixed degree bound. Suppose that $\langle u, U\rangle$ is an (ultimately) solvable reflection of $\overline{f_{X,d}}^{-1}[T]$. Then $\left\langle u \circ f_{X,d}, U\right\rangle$ is an (ultimately) solvable reflection of T with respect to degree-d simulations, in the sense that (1) $u \circ f_{X,d}$ has degree at most d, (2) U is solvable, and (3) for solvable transition ideal V and simulation v from T to V of degree at most d, there is a unique linear simulation $\hat{v}$ such that $v = \hat{v} \circ u \circ f_{X,d}$.

5 KLEENE CLOSURE OF SOLVABLE TRANSITION IDEALS

In this section we describe how to compute $T^* = \bigcap_{i=0}^{\infty} T^i$ when T is either a solvable and ultimately solvable transition ideal. In doing so we introduce a sub-problem of potential independent interest. The sub-problem asks how to find the set of rational polynomials that evaluate to 0 for every position in a $\mathbb{Q}$-algebra sequence defined by a solvable polynomial map.

5.1 Finding the Relations of a Solvable Map over a $\mathbb{Q}$-Algebra

Problem 5.1. Let A be a $\mathbb{Q}$-algebra, let X be a finite set of variables, and let $v \in A^X$. *Given a solvable map* $f : \mathbb{Q}[X] \to \mathbb{Q}[X]$ *and basis* I *such that* $\langle I\rangle = \mathbf{I}_A(\{v\})$, *find a basis for* $\mathbf{I}_A\left(\left\{f_A^i(v) : i \in \mathbb{N}\right\}\right) \subseteq \mathbb{Q}[X]$.

Intuitively, the solvable map f in Problem 5.1 defines a sequence, $(v, f_A^1(v), f_A^2(v), \dots)$. The goal of the problem is to find a basis for the set of polynomials $p \in \mathbb{Q}[X]$ such that

$$(p^A(v), p^A(f_A^1(v)), p^A(f_A^2(v)), \dots) = (0, 0, 0, \dots).$$

The purpose of the ideal $\langle I \rangle = \mathbf{I}_A(\{v\})$ is to give the set of polynomial relations of the first element of the sequence, v. In the case of Problem 5.1 we take $\mathbf{I}_A(\{v\})$ as a given to encode the relevant information of A and v.

Example 5.1. Let $X = \{x, y\}$, $A = \mathbb{Q}[w]/\langle w^2 - 3\rangle$, and $v = \{x \mapsto w + \langle w^2 - 3\rangle, y \mapsto 2w + 3 + \langle w^2 - 3\rangle\}$. For this case we have $\mathbf{I}_A(\{v\}) = \langle 2x - y + 3, x^2 - 3\rangle$. Let $f : \mathbb{Q}[x, y] \to \mathbb{Q}[x, y]$ be the polynomial homomorphism defined by $f(x) = 2y$ and $f(y) = 2x$. Then f is a solvable map that defines the following sequence over A^X:

$$\left(\begin{Bmatrix} x \mapsto w + \langle w^2 - 3\rangle \\ y \mapsto 2w + 3 + \langle w^2 - 3\rangle \end{Bmatrix}, \begin{Bmatrix} x \mapsto 4w + 6 + \langle w^2 - 3\rangle \\ y \mapsto 2w + \langle w^2 - 3\rangle \end{Bmatrix}, \begin{Bmatrix} x \mapsto 4w + \langle w^2 - 3\rangle \\ y \mapsto 8w + 12 + \langle w^2 - 3\rangle \end{Bmatrix}, \dots\right)$$

It can readily be verified that $p(x, y) = x^2 - 4xy + y^2 \in \mathbf{I}_A\left(\left\{f_A^i(v) : i \in \mathbb{N}\right\}\right)$. For instance, let $v_2 = \{x \mapsto 4w + \langle w^2 - 3\rangle, y \mapsto 8w + 12 + \langle w^2 - 3\rangle\} \in A^X$ denote the second (indexing from 0) valuation of the sequence. Then $p^A(v_2) = (4w + \langle w^2 - 3\rangle)^2 - 4(4w + \langle w^2 - 3\rangle)(8w + 12 + \langle w^2 - 3\rangle) + (8w + 12 + \langle w^2 - 3\rangle)^2 = 144 - 48w^2 + \langle w^2 - 3\rangle = 144 - 48(3) + \langle w^2 - 3\rangle = 0 + \langle w^2 - 3\rangle$.

We can also view Problem 5.1 as defining $|X|$ c-finite sequences of A elements, where each sequence is the trajectory of a particular variable. If we take the special case where $A = \mathbb{Q}$ then the goal is to find the *algebraic relations* [Kauers and Zimmermann 2008] of the $|X|$ sequences.

Definition 5.1. (Modification of Kauers and Zimmermann [2008]) Let k be a field and K a (commutative) k-algebra. An **algebraic relation over** k among $a_1, \dots, a_m \in K$ is an element of the kernel of the k-algebra homomorphism $\varphi : k[x_1, \dots, x_m] \to K$ that maps x_j to a_j.

Kauers and Zimmermann [2008, Algorithm 2] presents a method to find the set of algebraic relations over $\mathbb{Q}$ for the case of a set of c-finite sequences over the $\mathbb{Q}$-algebra, $\mathbb{Q}^\omega$; consequently, solving Problem 5.1 for the case where $A = \mathbb{Q}$. In this section we show how the method of Kauers and Zimmermann [2008] can be utilized to solve Problem 5.1 for the case of an arbitrary $\mathbb{Q}$-algebra A. First we briefly review the method of Kauers and Zimmermann [2008].

At a high-level, the method of Kauers and Zimmermann [2008] is, given c-finite sequences $\{a_1(n)\}_{n=0}^\infty, \dots, \{a_k(n)\}_{n=0}^\infty \in \mathbb{Q}^\omega$, perform the following:

(1) Compute closed-form solutions of each sequence as $a_i(n) = \sum_{j=1}^m p_{ij}(n)\Theta_j^n$ for polynomials $p_{ij} \in \bar{\mathbb{Q}}[n]$ and values $\Theta_1, \dots, \Theta_m \in \bar{\mathbb{Q}}$.
(2) Using the algorithm of Ge [1993] compute a basis $J \subseteq \mathbb{Q}[y_0, y_1, \dots, y_m]$ for the ideal of algebraic relations over $\mathbb{Q}$ of the sequences $\{n\}_{n=0}^\infty, \{\Theta_1^n\}_{n=0}^\infty, \dots, \{\Theta_m^n\}_{n=0}^\infty \in \bar{\mathbb{Q}}^\omega$. That is, $p(y_0, y_1, \dots, y_m) \in \langle J\rangle$ if and only if $p(n, \Theta_1^n, \dots, \Theta_m^n) = 0$ for $n \in \mathbb{N}$.
(3) Let $B = \left\{x_i - \sum_{j=1}^m p_{ij}(y_0)y_j : 1 \le i \le k\right\}$. Using Gröbner basis elimination techniques compute the ideal $\langle H\rangle = (\langle B \cup J\rangle) \cap \mathbb{Q}[x_1, \dots, x_k]$.

The resulting ideal, $\langle H\rangle$, is the ideal of algebraic relations over $\mathbb{Q}$ of the sequences $\{a_1(n)\}_{n=0}^\infty, \dots, \{a_k(n)\}_{n=0}^\infty \in \mathbb{Q}^\omega$. That is, $\langle H\rangle$ has the property that $p \in \langle H\rangle$ if and only if $p \in \mathbb{Q}[x_1, \dots, x_k]$ and $p(a_1(n), \dots, a_k(n)) = 0$ for all $n \in \mathbb{N}$.

Remark. It should be noted that the method presented above as well as in Kauers and Zimmermann [2008] only works when the characteristic polynomials of the input recurrences do not have 0 as a root. Kauers and Zimmermann [2008] notes this, and correctly states such a situation can be handled with a pre-processing step. In this paper, we are more explicit on how to handle 0 roots.

Algorithm 3: Solve solvable map

Input: $\mathbb{Q}$-algebra A, valuation $v \in A^X$, solvable map $f : \mathbb{Q}[X] \to \mathbb{Q}[X]$, and basis I for the ideal $\mathbf{I}_A(\{v\})$.

Output: A basis for the ideal $\mathbf{I}_A\left(\left\{f_A^n(v) : i \in \mathbb{N}\right\}\right)$

1 $m \leftarrow |X|$;

2 For each $x_j \in X$ solve the trajectory of x_j as $f_A^n(x_j) = \sum_{i=1}^m \left(z_{ij}(n) + \sum_{k=1}^m p_{ijk}(n)\Theta_k^n\right) v(x_i)$;

3 For $1 \le i \le m$ and $1 \le j \le m$ let $a_{ij}(n) = \sum_{k=1}^m p_{ijk}(n)\Theta_k^n \in \mathbb{Q}^\omega$;

4 Using Kauers and Zimmermann [2008, Algorithm 2] let $J' \subseteq \mathbb{Q}[y_{11}, \ldots, y_{1m}, \ldots, y_{m1}, \ldots, y_{mm}]$ be the ideal of algebraic relations of $\{a_{11}(n+m)\}_{n=0}^\infty, \ldots, \{a_{1m}(n+m)\}_{n=0}^\infty, \ldots, \{a_{m1}(n+m)\}_{n=0}^\infty, \ldots, \{a_{mm}(n+m)\}_{n=0}^\infty$;

5 $J \leftarrow$ Basis for the ideal $J' \cap \bigcap_{n=0}^{m-1} \left\langle \left\{ y_{ij} - (z_{ij}(n) + \sum_{k=1}^m p_{ijk}(n)\Theta_k^n) : 1 \le i \le m, 1 \le j \le m \right\} \right\rangle$;

6 **return** *inv.image*$(f, I \cup J)$, *where* f *is the map that sends* $x_j \mapsto x_1 y_{1j} + \cdots + x_m y_{mj}$

The main observation that leads to our method for a general $\mathbb{Q}$-algebra A is that the only "new" algebraic relations over $\mathbb{Q}$ for the A sequences must come from the relations of the initial values of the sequence (the ideal $\langle I \rangle = \mathbf{I}_A(\{v\})$ in the statement of Problem 5.1). This observation leads to our method for solving Problem 5.1, which we present as Algorithm 3.

Algorithm 3 begins by writing the trajectory of each variable x_j as the closed-form solution $\sum_{i=1}^m \left(z_{ij}(n) + \sum_{k=1}^m p_{ijk}(n)\Theta_k^n\right) v(x_i)$, which is a sum-of-products of the form $a'_{1j}(n)v(x_1) + \cdots + a'_{mj}(n)v(x_m)$ with each $\left\{a'_{ij}(n)\right\}_{n=0}^\infty \in \mathbb{Q}^\omega$. Moreover, each a'_{ij} is a rational c-finite sequence. However, in general, the sequences $\left\{a'_{ij}(n)\right\}_{n=0}^\infty$ might have 0 as a root of their characteristic polynomials—hence the presence of the term $z_{ij}(n)$. These $z_{ij}(n)$ terms mean that the method of [Kauers and Zimmermann 2008] cannot be directly applied. The following lemma shows how we can handle this general case.

Lemma 5.2. Let $\left\{a'_1(n)\right\}_{n=0}^\infty, \ldots, \left\{a'_m(n)\right\}_{n=0}^\infty \in \mathbb{Q}^\omega$ such that $a'_i(n) = z_i(n) + a_i(n)$ for $\{z_i(n)\}_{n=0}^\infty$, $\{a_i(n)\}_{n=0}^\infty \in \mathbb{Q}^\omega$. Furthermore, suppose that there exists some $d \in \mathbb{N}$ such that for all $1 \le i \le m$, $z_i(n) = 0$ for $n \ge d$. Let $J' \subseteq \mathbb{Q}[y_1, \ldots, y_m]$ be the ideal of algebraic relations over $\mathbb{Q}$ of $\{a_1(n+d)\}_{n=0}^\infty, \ldots, \{a_m(n+d)\}_{n=0}^\infty$. Then $J = J' \cap \bigcap_{n=0}^{d-1} \langle \{y_i - (z_i(n) + a_i(n)) : 1 \le i \le m\} \rangle$ is the ideal of algebraic relations of $\left\{a'_1(n)\right\}_{n=0}^\infty, \ldots, \left\{a'_m(n)\right\}_{n=0}^\infty$.

Because each $\left\{a'_{ij}(n)\right\}_{n=0}^\infty$ is c-finite, each $z_{ij}(n)$ has the property that $z_{ij}(n) = 0$ for $n \ge d$, where d is the maximal order of the sequences $\left\{a'_{ij}(n)\right\}_{n=0}^\infty$. In the case of Algorithm 3 we have $d \le m$. Thus, each $z_{ij}(n) = 0$ for $n \ge m$. Lemma 5.2 shows that J in Algorithm 3 is a basis for the ideal of algebraic relations over $\mathbb{Q}$ of the $\left\{a'_{ij}(n)\right\}_{n=0}^\infty$ sequences.

We use the next two lemmas to establish the needed property of the return value of Algorithm 3 and thus show that Algorithm 3 is correct. The desired result of Algorithm 3 is the ideal of algebraic relations over $\mathbb{Q}$ of the A sequences, $f_A^n(x_j)$ defined in Line 2 of Algorithm 3. These sequences are defined as a sum-of-products of rational sequences and constant sequences of the valuations $v(x_i)$. The next lemma (Lemma 5.3) shows that if we want to find the algebraic relations over $\mathbb{Q}$ among arbitrary rational sequences lifted to A^ω and constant valuations $\left\{\left\{v(x_j)\right\}_{i=0}^\infty : x_j \in X\right\}$, it

is sufficient to consider the ideals of algebraic relations over $\mathbb{Q}$ among $\left\{\left\{v(x_j)\right\}_{i=0}^{\infty} : x_j \in X\right\}$ and the lifted rational sequences separately. This is what makes Algorithm 3 possible: we are given the algebraic relations over $\mathbb{Q}$ among $\left\{\left\{v(x_j)\right\}_{i=0}^{\infty} : x_j \in X\right\}$ as input, and we can calculate the algebraic relations over $\mathbb{Q}$ among the c-finite sequences defined in Line 2 using the algorithm of Kauers and Zimmermann [2008]. The second lemma (Lemma 5.4) then applies Lemma 5.3 to the specific form of the $\left\{f_A^n(x_j)\right\}_{n=0}^{\infty}$ sequences defined in Algorithm 3 to establish the correctness of the algorithm.

Lemma 5.3. Let $\{a_1(i)\}_{i=0}^{\infty}, \ldots, \{a_m(i)\}_{i=0}^{\infty} \in \mathbb{Q}^{\omega}$ and let $J \subseteq \mathbb{Q}[y_1, \ldots, y_m]$ be the ideal of algebraic relations over $\mathbb{Q}$ among $\{a_1(i)\}_{i=0}^{\infty}, \ldots, \{a_m(i)\}_{i=0}^{\infty}$. Let A be a $\mathbb{Q}$-algebra with additive unit 0_A and multiplicative unit 1_A. Let $X = \{x_1, \ldots, x_n\}$. Let $v \in A^X$ and let $I = \mathbf{I}_A(\{v\}) \subseteq \mathbb{Q}[X]$. Let $\varphi : \mathbb{Q}[X, Y] \to A^{\omega}$ be the $\mathbb{Q}$-algebra homomorphism defined by

$$\varphi(x_j) = \left\{v(x_j)\right\}_{i=0}^{\infty} \qquad \varphi(y_j) = \left\{a_j(i)(1_A)\right\}_{i=0}^{\infty}$$

Then $\varphi(p) = \{0_A\}_{i=0}^{\infty}$ if and only if $p \in \langle I \cup J\rangle \subseteq \mathbb{Q}[X, Y]$.

Proof. ($\Longleftarrow$) Suppose $p \in \langle I \cup J\rangle$. Thus, $p = g + h$ for some $g \in \langle I\rangle \subseteq \mathbb{Q}[X, Y]$ and $h \in \langle J\rangle \subseteq \mathbb{Q}[X, Y]$.

Because $g \in \langle I\rangle$, $g = \sum_{j=1}^{k_1} f_j g_j$ for $f_j \in \mathbb{Q}[X, Y]$ and $g_j \in I$. Because $g_j \in I = \mathbf{I}_A(\{v\})$, $g_j^A(v) = 0_A$ for each j. Therefore, $\varphi(g) = \sum_j^{k_1} \varphi(f_j) \left\{g_j^A(v)\right\}_{i=0}^{\infty} = \sum_j^{k_1} \varphi(f_j) \{0_A\}_{i=0}^{\infty} = \{0_A\}_{i=0}^{\infty}$.

Now we show the same for h. Because $h \in \langle J\rangle$, $h = \sum_{j=1}^{k_2} f_j h_j$ for $f_j \in \mathbb{Q}[X, Y]$ and $h_j \in J \subseteq \mathbb{Q}[Y]$. Because $h_j \in J$, $\varphi(h_j) = \left\{h_j(a_1(i), \ldots, a_m(i))(1_A)\right\}_{i=0}^{\infty} = \{0\}_{i=0}^{\infty}$ for each j. Therefore, $\varphi(h) = \sum_{j=1}^{k_2} \varphi(f_j)\varphi(h_j) = \sum_{j=1}^{k_2} \varphi(f_j) \{0_A\}_{i=0}^{\infty} = \{0_A\}_{i=0}^{\infty}$.

Combining the previous paragraphs we have that, because φ is a homomorphism, $\varphi(p) = \varphi(g) + \varphi(h) = \{0_A\}_{i=0}^{\infty} + \{0_A\}_{i=0}^{\infty} = \{0_A\}_{i=0}^{\infty}$.

($\Longrightarrow$) Suppose $\varphi(p) = \{0_A\}_{i=0}^{\infty}$. Let r be p reduced by a Gröbner basis for $\langle I \cup J\rangle$ under the elimination order $\ll_X$. That is $p = g_I + g_J + r$ for some $g_I \in I$, $g_J \in J$ and for any other $g' \in \langle I \cup J\rangle$ and $r' \in \mathbb{Q}[X, Y]$ with $p = g' + r'$, $\mathrm{LM}(r') \gg_X \mathrm{LM}(r)$. Because φ is a homomorphism we have $\{0_A\}_{i=0}^{\infty} = \varphi(p) = \varphi(g_I + g_J + r) = \varphi(g_I) + \varphi(g_J) + \varphi(r) = \{0_A\}_{i=0}^{\infty} + \varphi(r) = \varphi(r)$. We can write r by collecting X monomials with respect to the order $\gg_X$ as follows

$$\begin{aligned} r =& m_1^X p_1^Y(y_1, \ldots, y_m) + \cdots + m_k^X p_k^Y(y_1, \ldots, y_m) + \\ & p_{k+1}^Y(y_1, \ldots, y_m) \end{aligned}$$

where each m_s^X is a distinct monomial of X variables with $m_1^X \gg_X m_j^X$ for $j = 2, \ldots, k$, and each p_s^Y is a polynomial in $\mathbb{Q}[Y]$.

Observe that there exists some i such that $p_1^Y(a_1(i), \ldots, a_m(i)) \neq 0$. If not, $\varphi(p_1^Y(y_1, \ldots, y_m)) = \left\{p_1^Y(a_1(i), \ldots, a_m(i))(1_A)\right\}_{i=0}^{\infty} = \{0_A\}_{i=0}^{\infty}$. Thus, $p_1^Y(a_1(i), \ldots, a_m(i)) = 0$ for $i \in \mathbb{N}$, and therefore p_1^Y is an algebraic relation over $\mathbb{Q}$ among $\{a_1(i)\}_{i=0}^{\infty}, \ldots, \{a_m(i)\}_{i=0}^{\infty}$. Then by definition $p_1^Y \in J$. But this contradicts the property that r is reduced with respect to J. That is, if $p_1^Y(y_1, \ldots, y_m) \in \langle J\rangle$, then there exists a better r' that does not contain the monomial m_1^X. Therefore, there is some i such that $p_1^Y(a_1(i), \ldots, a_m(i)) \neq 0$.

Let i be such that $p_1^Y(a_1(i), \ldots, a_m(i)) \neq 0$. We have $\varphi(r) = \{0_A\}_{i=0}^{\infty}$ by assumption, and so $\varphi(r)_i = 0_A$, where

$$\begin{aligned} \varphi(r)_i =& (m_1^X)^A(v) p_1^Y(a_1(i), \ldots, a_m(i)) + \cdots + (m_k^X)^A(v) p_k^Y(a_1(i), \ldots, a_m(i)) + \\ & p_{k+1}^Y(a_1(i), \ldots, a_m(i)) \end{aligned}$$

denotes the ith element of $\varphi(r)$. Because $\varphi(r)_i = 0_A$, we must have

$$\begin{aligned} &m_1^X p_1^Y(a_1(i),\ldots,a_m(i)) + \cdots + m_k^X p_k^Y(a_1(i),\ldots,a_m(i)) + \\ &p_{k+1}^Y(a_1(i),\ldots,a_m(i)) \in I \end{aligned} \tag{3}$$

Denote the polynomial in (3) as h. We can rewrite r as follows

$$r = \frac{p_1^Y(y_1,\ldots,y_m)}{p_1^Y(a_1(i),\ldots,a_m(i))}h + r'$$

For some r' containing x monomials $m_2^X,\ldots,m_k^X$. However, this (nearly) contradicts the property that p is reduced with respect to I. That is, $p = (g_I + g_J + \frac{p_1^Y(y_1,\ldots,y_m)}{p_1^Y(a_1(i),\ldots,a_m(i))}h) + r'$ with $g_I + g_J + \frac{p_1^Y(y_1,\ldots,y_m)}{p_1^Y(a_1(i),\ldots,a_m(i))}h \in \langle I \cup J\rangle$. Moreover, $\mathrm{LM}(r) \gg_X \mathrm{LM}(r')$, because r' does not contain the monomial m_1^X and $m_1^X \gg_X m_j^X$ for $j = 2,\ldots,k$. The only way to avoid the contradiction is to have m_1^X be a constant. Therefore, because r is reduced with respect to an order that eliminates X variables and $\mathrm{LM}(r) \in \mathbb{Q}[Y]$, we have $r \in \mathbb{Q}[Y]$.

Finally, because $r \in \mathbb{Q}[Y]$ and $\varphi(r) = \{0_A\}_{i=0}^{\infty}$, r must be an algebraic relation over $\mathbb{Q}$ among $\{a_1(i)\}_{i=0}^{\infty},\ldots,\{a_m(i)\}_{i=0}^{\infty}$, and therefore $r \in J$. However, because r must be reduced with respect to J, $r = 0$. Thus, $p = g_I + g_J$ with $g_I + g_J \in I + J$. Therefore, $p \in \langle I \cup J\rangle$. □

Lemma 5.4. Let $\{a_{1m}(i)\}_{i=0}^{\infty},\ldots,\{a_{1m}(i)\}_{i=0}^{\infty},\ldots,\{a_{m1}(i)\}_{i=0}^{\infty},\ldots,\{a_{mm}(i)\}_{i=0}^{\infty} \in \mathbb{Q}^{\omega}$ and let $J \subseteq \mathbb{Q}[y_{11},\ldots,y_{1m},\ldots,y_{m1},\ldots,y_{mm}]$ be the ideal of algebraic relations over $\mathbb{Q}$ among these sequences. Let A be a $\mathbb{Q}$-algebra, $X = \{x_1,\ldots,x_m\}$, $v \in A^X$, and $I = \mathbf{I}_A(\{v\}) \subseteq \mathbb{Q}[X]$. For every $i \in \mathbb{N}$ let $w^i \in A^X$ be defined as $w^i(x_j) = a_{1j}(i)v(x_1) + \cdots + a_{mj}(i)v(x_m)$. Let $f : \mathbb{Q}[X] \to \mathbb{Q}[X,Y]$ be the polynomial homomorphism that maps $x_j \mapsto x_1y_{1j} + \cdots + x_my_{mj}$ for all j. Then $\mathbf{I}_A\left(\left\{w^i : i \in \mathbb{N}\right\}\right) = f^{-1}[\langle I \cup J\rangle]$.

Combining Lemmas 3.1, 5.2 and 5.4 establishes the correctness of Algorithm 3.

Theorem 5.5. Algorithm 3 solves Problem 5.1.

5.2 Closing Transition Ideals

The result of Algorithm 3 produces a polynomial ideal that summarizes the algebraic relations over $\mathbb{Q}$ of a solvable polynomial map. In this subsection, we show how the result of Algorithm 3 can be used to compute T^* for a (ultimately) solvable transition ideal T. Recall that every solvable transition ideal T comes with a solvability witness p. The basic idea to compute T^* is to use Algorithm 3 to summarize the algebraic relations over $\mathbb{Q}$ among the sequences defined by p. However, Problem 5.1, which is solved by Algorithm 3, is defined over a $\mathbb{Q}$-algebra A. Hence, in this subsection we work to motivate and explain that in order to calculate T^*, the $\mathbb{Q}$-algebra we want to instantiate Problem 5.1 with is $A = \mathbb{Q}[X,X']/\mathrm{dom}^*(T)$.

Before we talk of $\mathbb{Q}$-algebras, we make a brief observation of the structure of solvable transition ideal. Intuitively, a solvable transition ideal can be broken into a domain part, containing only unprimed variables, and a transition part. The next lemma formalizes this point.

Lemma 5.6. Let $T \subseteq \mathbb{Q}[X,X']$ be a solvable transition ideal, and let $p : \mathbb{Q}[X] \to \mathbb{Q}[X]$ be a solvable witness for T. Then $T^n = \mathrm{dom}(T^n) + \left\langle \left\{x_i' - p^n(x_i) : 1 \le i \le m\right\}\right\rangle$ for $1 \le n$.

From Lemma 5.6, we see that the iterated behavior of the transition ideal is *mostly* captured by the iterated behavior of the polynomial witness; what is missing is the $\mathrm{dom}(T^n)$ part. Note that $\mathrm{dom}(T^n)$ is an ideal for each n. If we let A be the $\mathbb{Q}$-algebra $\mathbb{Q}[X,X']/I$ for some ideal I, we

can define a sequence like the one in Problem 5.1 that uses a solvable witness p to transition not only variables but sets of polynomials with respect to I. This can be formalized in the language of Problem 5.1 for a solvable transition ideal $T \subseteq \mathbb{Q}[X, X']$ with solvability witness $p : \mathbb{Q}[X] \to \mathbb{Q}[X]$ as follows:

- Let $A = \mathbb{Q}[X, X']/I$.
- Let $\hat{p} : \mathbb{Q}[X, X'] \to \mathbb{Q}[X, X']$ extend p as $\hat{p}(x_i) = x_i$ if $x_i \in X$, and $\hat{p}(x'_i) = (p(x_i))'$ if $x'_i \in X'$.
- Let $v \in A^{X \cup X'}$ be defined as $v(x_i) = x_i + I$ if $x_i \in X$ and $v(x'_i) = p(x_i) + I$ if $x'_i \in X'$.

The question then is what should we take for I. We want I to be the polynomials not captured by the iteration of the solvable witness. Thus, from Lemma 5.6 these are the polynomials in $\mathrm{dom}(T^n)$. For Problem 5.1 I needs to be fixed, so we have two obvious options for I, the domain of T, $\mathrm{dom}(T)$, or the invariant domain of T, $\mathrm{dom}^*(T)$. The next example shows what happens if we use the domain of T and why that gives us a mismatch for computing T^*.

Example 5.2. Consider the solvable transition ideal $T = \langle y' - y - z, z' - 3z, y - z - 1 \rangle$, with $\mathrm{dom}(T) = \langle y - z - 1 \rangle = I$. A solvable witness for T is $p(y) = y+z$ and $p(z) = 3z$. Let $A = \mathbb{Q}[X, X']/I$, $\hat{p} : \mathbb{Q}[X, X'] \to \mathbb{Q}[X, X']$ extend p as above, and let $v \in A^{X \cup X'}$ with $v(y) = y + I$, $v(z) = z + I$, $v(y') = y+z+I$, and $v(z') = 3z+I$. In the language of Problem 5.1, we have $\mathbf{I}_A(\{v\}) = T$. $\hat{p}_A$ defines the following sequence[5]:

$$\left(\begin{Bmatrix} y \mapsto y + I \\ z \mapsto z + I \\ y' \mapsto 2z + 1 + I \\ z' \mapsto 3z + I \end{Bmatrix}, \begin{Bmatrix} y \mapsto y + I \\ z \mapsto z + I \\ y' \mapsto 5z + 1 + I \\ z' \mapsto 9z + I \end{Bmatrix}, \begin{Bmatrix} y \mapsto y + I \\ z \mapsto z + I \\ y' \mapsto 14z + 1 + I \\ z' \mapsto 27z + I \end{Bmatrix}, \ldots \right).$$

Taking $\mathbf{I}_A(\{v^i\})$ for each i of the above sequence is nearly $T, T^2, T^3, \ldots$. However, $T^2 = \langle y' - y - z, z' - 3z, y - 1, z \rangle = \langle y' - 1, z', y - 1, z \rangle$, but $y' - 1 \notin \mathbf{I}_A(\{v^2\})$. Moreover, for *every* $i \geq 2$, $\mathbf{I}_A(\{v^i\}) \neq T^i$.

The essential problem with the previous example is that the domain of T is not stable for higher iterations of T^i. If instead of $I = \mathrm{dom}(T)$ in the previous example we used $I = \mathrm{dom}^*(T)$ then we would have the equality $\mathbf{I}_A(\{v^i\}) = T^i$ for $i \geq 2$. This observation that equality can be recovered for some i by using the *invariant domain* is our key insight for computing T^*. The issue of Example 5.2 is fixed using the reasoning in the following lemma.

Lemma 5.7. Let $T \subseteq \mathbb{Q}[X, X']$ be a solvable transition ideal with solvability witness p. Let $N \geq 1$ be such that $\mathrm{dom}(T^N) = \mathrm{dom}^*(T)$, and define $I \triangleq \mathrm{dom}^*(T)$. Let A be the $\mathbb{Q}$-algebra $\mathbb{Q}[X, X']/I$. Let $\hat{p} : \mathbb{Q}[X, X'] \to \mathbb{Q}[X, X']$ be the homomorphism defined by $\hat{p}(x_i) = x_i$ and $\hat{p}(x'_i) = p(x_i)'$. Let $v \in A^{X \cup X'}$ be the valuation defined by $v(x_i) = x_i + I$ and $v(x'_i) = p(x_i) + I$. Then for $1 \leq N \leq n$, $T^n = \mathbf{I}_A(\{\hat{p}_A^{n-1}(v)\})$. Furthermore, if $1 \leq n < N$ then $T^n \subseteq \mathbf{I}_A(\{\hat{p}_A^{n-1}(v)\})$

Corollary 5.7.1. Lemma 5.7 also holds for ultimately solvable transition ideals.

Example 5.3. Recall Example 5.2, but with $I = \mathrm{dom}(T^2) = \mathrm{dom}(T^3) = \mathrm{dom}^*(T) = \langle y - 1, z \rangle$. $\hat{p}_A$ defines the following sequence:

$$\left(\begin{Bmatrix} y \mapsto 1 + I \\ z \mapsto 0 + I \\ y' \mapsto 1 + I \\ z' \mapsto 0 + I \end{Bmatrix}, \begin{Bmatrix} y \mapsto 1 + I \\ z \mapsto 0 + I \\ y' \mapsto 1 + I \\ z' \mapsto 0 + I \end{Bmatrix}, \begin{Bmatrix} y \mapsto 1 + I \\ z \mapsto 0 + I \\ y' \mapsto 1 + I \\ z' \mapsto 0 + I \end{Bmatrix}, \ldots \right).$$

[5]Note that $y + z + \langle y - z - 1 \rangle = 2z + 1 + \langle y - z - 1 \rangle$

Informally, Lemma 5.7 states that the long-running relations of a solvable transition ideal T is *exactly* captured by $\mathbf{I}_A\left(\left\{\hat{p}^i(v) : i \in \mathbb{N}\right\}\right)$. This is what Example 5.3 shows. However, we cannot just take the long-running relations of T as our summary T^*. This is because for iterations before the invariant domain has stabilized we do not have equality. However, the invariant domain must stabilize in a finite number of iterations, and can then be recovered via ideal intersection. This leads to the following theorem.

Theorem 5.8. Let T be a (ultimately) solvable transition ideal $T \subseteq \mathbb{Q}[X, X']$ with solvability witness p. Let $N \geq 1$ be such that $\mathrm{dom}(T^N) = \mathrm{dom}^*(T) = \mathrm{I}$. Let $\hat{p} : \mathbb{Q}[X, X'] \to \mathbb{Q}[X, X']$ be the homomorphism defined by $\hat{p}(x_i) = x_i$ and $\hat{p}(x_i') = p(x_i)'$. Let $v \in A^{X \cup X'}$ be the valuation defined by $v(x_i) = x_i + I$ and $v(x_i') = p(x_i) + I$. Then

$$\bigcap_{i=0}^{\infty} T^i = \left(\bigcap_{i=0}^{N-1} T^i\right) \cap \left(\mathbf{I}_A\left(\left\{\hat{p}^i(v) : i \in \mathbb{N}\right\}\right)\right).$$

The right-hand-side of the equation in Theorem 5.8 is computable. The term $\mathbf{I}_A\left(\left\{\hat{p}^i(v) : i \in \mathbb{N}\right\}\right)$ can be computed by Algorithm 3 with $\mathbf{I}_A(\{v\}) = \mathrm{dom}^*(T) + T$. The term $(\bigcap_{i=0}^{N-1} T^i)$ is a finite intersection of polynomial ideals which can be computed via Gröbner basis techniques. Asymptotically, the Gröbner basis calculations for computing intersections as well as the Gröbner basis calculations in Algorithm 3 dominate the running time, making the overall computation exponential.

Example 5.4. Recall $T = \langle y' - y - z, z' - 3z, y - z - 1\rangle$ from Example 5.2.

$$T^* = \left\langle 3z^2 - 4zz' + (z')^2, yz' - yz + z^2 - zz' - z' + z, 2y' - 2y - z' + z\right\rangle$$

6 LOOP SUMMARIZATION MODULO LIRR

Loop summarization is the problem of computing, for a given transition formula F representing the body of some loop, an over-approximation of the reflexive transitive closure of F. This section describes how to combine the components introduced in the previous two sections to accomplish this task. We prove the key property that our loop summarization procedure is monotone. Finally, we discuss how this procedure can be combined with other summarization techniques to enhance the ability of an algebraic program analyzer to generate non-linear loop summaries.

Our iteration operator takes a four-step approach (pictured Fig. 1b). Given an input transition formula F,

(1) Compute the transition ideal $\mathbf{I}_{\mathbf{LIRR}}(F)$ of F (using the algorithm of Kincaid et al. [2023])
(2) Compute a solvable reflection $\langle t, T\rangle$ of $\mathbf{I}_{\mathbf{LIRR}}(F)$ (Section 4)
(3) Compute T^* (Section 5)
(4) Calculate the formula corresponding to the image of T^* under t.

More succinctly, we define an operator $(-)^{\circledast} : \mathbf{TF} \to \mathbf{TF}$ to be

$$F^{\circledast} \triangleq \mathbf{F}(\bar{t}\,[T^*])$$

where $\langle t, T\rangle$ is a solvable reflection of $\mathbf{I}_{\mathbf{LIRR}}(F)$. Naturally, one may repeat this recipe for defining a loop summarization operator by using ultimately solvable transition ideals (Section 4.2) and/or polynomial simulations (Section 4.3), and the soundness and monotonicity results that we prove below hold also for these variants.

Example 6.1. Consider the transition formula F below, and its associated transition ideal:

$$F = \left(\begin{array}{c} (k' = k+1) \wedge (x' = y) \wedge (y' = x) \\ \wedge \left(\begin{array}{c} (z \geq 0 \wedge z' = w + z \wedge w' = x^2) \\ \vee (\neg(z \geq 0) \wedge w' = w + z \wedge z' = x^2) \end{array} \right) \end{array} \right) \quad \mathbf{I_{LIRR}}\,(F) = \left\langle \begin{array}{l} w'x^2 - w'z' + x^2z' - x^2, \\ -w' + w + x^2 - z' + z, \\ x' - y, \\ y' - x, \\ k' - k - 1 \end{array} \right\rangle$$

Notice that, while F employs a rich logical language involving disjunction, negation, and inequalities, its ideal $\mathbf{I_{LIRR}}\,(F)$ is defined by the set of polynomials p such that F entails $p = 0$. A solvable reflection of $\mathbf{I_{LIRR}}\,(F)$ is $\langle t, T\rangle$ where t is the map that sends $a \mapsto x$, $b \mapsto y$, $c \mapsto (w + z)$, and $d \mapsto k$, and T is the ideal $\left\langle a' - b, b' - a, c' - c - a^2, d' - d - 1\right\rangle$. The closure of T is $T^* = \left\langle ab - bb' + (b')^2 - ab', a' + b' - a - b, b^2d' + a^2d' - a^2d - b^2d - b^2 + ab' + bb' - ab - 2c' + 2c\right\rangle$. The (ideal generated by) the image of T^* under $\overline{t}$ is

$$\left\langle \begin{array}{c} xy - yy' + (y')^2 - xy', x' + y' - x - y, \\ y^2k' + x^2k' - x^2k - y^2k - y^2 + xy' + yy' - xy - 2(w' + z') + 2(w + z) \end{array} \right\rangle.$$

Finally, we have $F^{\circledast} \triangleq xy + (y')^2 = yy' + xy' \wedge x' + y' = x + y \wedge y^2k' + x^2k' + xy' + yy' - 2(w' + z') = x^2k + y^2k + y^2 + xy - 2(w + z)$.

Theorem 6.1 (Soundness)**.** Let F be a transition formula. For any $n \in \mathbb{N}$, we have $F^n \models_{\mathbf{LIRR}} F^{\circledast}$

Theorem 6.2 (Monotonicity)**.** Let F and G be transition formulas. If $F \models_{\mathbf{LIRR}} G$, then $F^{\circledast} \models_{\mathbf{LIRR}} G^{\circledast}$.

Proof. Suppose $F \models_{\mathbf{LIRR}} G$. Let $\langle t, T\rangle$ be a solvable reflection of $\mathbf{I_{LIRR}}\,(F)$, and let $\langle u, U\rangle$ be a solvable reflection of $\mathbf{I_{LIRR}}\,(G)$. Since $F \models_{\mathbf{LIRR}} G$, we must have $\mathbf{I_{LIRR}}\,(F) \supseteq \mathbf{I_{LIRR}}\,(G)$, and thus u is a simulation from $\mathbf{I_{LIRR}}\,(F)$ to U. Since U is solvable and $\langle t, T\rangle$ is a solvable reflection of $\mathbf{I_{LIRR}}\,(F)$, there is a (unique) simulation $v : T \to U$ such that $u = v; t = t \circ v$. Since v is a simulation, we have $T^* \supseteq \overline{v}\,[U^*]$, and so

$$\overline{t}\,[T^*] \supseteq \overline{t}\,[\overline{v}\,[U^*]] = \overline{(t \circ v)}\,[U^*] = \overline{u}\,[U^*]$$

Since $F^{\circledast} = \mathbf{F}(\overline{t}\,[T^*])$ and $G^{\circledast} = \mathbf{F}(\overline{u}\,[U^*])$, we have $F^{\circledast} \models_{\mathbf{LIRR}} G^{\circledast}$. □

6.1 Modular Design of Loop Summarization Operators

The loop summarization operator that is defined in this paper is designed to compute polynomial invariants. Such invariants are often just *a* component of a correctness argument for a program—for example, a correctness argument may rely upon reasoning about inequalities, or may require a disjunctive invariant. Our loop summarization operator can be incorporated in a broader invariant generation scheme by using various combinators to combine summarization operators. For instance, the simplest such combinator is a *product*, which combines two loop summarization $\circledast_1$ and $\circledast_2$ into one $\circledast_1 \times \circledast_2$ by taking their conjunction:

$$F^{\circledast_1 \times \circledast_2} \triangleq F^{\circledast_1} \wedge F^{\circledast_2}$$

Provided that both the $\circledast_1$ and $\circledast_2$ operators are monotone, then (1) so is their product, and (2) the resulting analysis is at least as precise as either component analysis.

Another kind of summarization combinator is the refinement technique proposed in Cyphert et al. [2019]. This combinator exposes phase structure in loops, and in particular enables a "base" summarization operator that may only generate conjunctive invariants to produce disjunctive invariants. In addition to monotonicity, this combinator requires four additional axioms in order to guarantee that it improves analysis precision. The following proposition states that indeed our summarization operator satisfies these conditions.

Proposition 6.3. Let F be a transition formula. Then the following hold

- (Reflexivity) $1 \models_{\mathbf{LIRR}} F^{\circledast}$
- (Extensivity) $F \models_{\mathbf{LIRR}} F^{\circledast}$
- (Transitivity) $F^{\circledast} \circ F^{\circledast} \equiv_{\mathbf{LIRR}} F^{\circledast}$
- (Unrolling) For any natural number n, $(F^n)^* \models_{\mathbf{LIRR}} F^*$.

7 EXPERIMENTAL EVALUATION

We consider two experimental questions concerning our methods for synthesizing loop invariants for general programs:

(1) (Section 7.2) How do our techniques apply to the task of verifying general programs?
(2) (Section 7.3) How do our techniques for generating polynomial invariants perform on programs for which other tools guarantee completeness?

In relation to each of these questions we also want to understand the performance, both in terms of accuracy and running time, of using linear simulations as well as polynomial simulations of bounded degree for extracting solvable transition ideals from transition ideals.

7.1 Experimental Setup

Implementation. We implemented the techniques described in this paper in a tool called Abstractionator. Our implementation relies on

- Chilon and ChilonInv [Kincaid et al. 2023], for **LIRR** operations and generating invariant inequalities, respectively.
- The FGb library [Faugère 2010] for an implementation of the F4 algorithm [Faugère 1999], which we use for computing of Gröbner bases.
- Flint [The FLINT team 2023] for integer lattice computations and Arb [Johansson 2017] for numerical polynomial root finding. These operations are required to implement the algorithm of Ge [1993] used in Algorithm 3.

Abstractionator can be configured to use either linear or quadratic simulations, and either solvable or ultimately solvable transition ideals. Our testing revealed that (1) the difference between using solvable and ultimately solvable is negligible (both in success rate and runtime performance), and (2) the cost of naïve computation of the full inverse image $\overline{f_{X,2}}^{-1}[-]$ for quadratic simulations is prohibitively high. In the following, we report on two configurations of Abstractionator: *USP-Lin* is the product of the ChilonInv domain and iteration operator induced by ultimately solvable linear reflections, *USP-Quad* is the product of USP-Lin and the iteration operator induced by solvable quadratic simulations with a single stratum (which necessitates only computing the affine polynomials in $\overline{f_{X,2}}^{-1}[-]$, and is therefore more tractable).

Environment. We ran all experiments on a virtual machine (using Oracle VirtualBox), with a guest OS of Ubuntu 22.04 allocated with 8 GB of RAM, using a 4-core Intel Core i7-4790K CPU @ 4.00 GHz. All tools were run with the BenchExec [Wendler and Beyer 2023] tool using a time limit of 300 seconds on all benchmarks.

Benchmarks. Our 202 benchmarks programs are sourced from the set of safe[6] benchmarks from the `c/ReachSafety-Loops` subcategory of the Software Verification Competition (SV-COMP) [Beyer 2023]. We divided our 202 benchmarks into a loops category consisting of 176 programs, and an NLA category consisting of 26 benchmarks. The NLA benchmarks are modified versions of the programs in the `nla-digbench` set from SV-COMP, intended to evaluate the strength of

[6]That is, the error location is truly unreachable.

Table 1. Comparison of tools on the loops and NLA benchmarks. T represents the amount of time, in seconds, take by each tool not including timeouts nor out of memory exceptions. The number of timeouts is reported in parentheses. We also experienced out of memory exceptions with VeriAbs which are noted in parentheses. #P represents the number of benchmarks proved correct. The best results in each category is bolded.

		loops	NLA	Total
	#B	176	26	202
ChilonInv	#P	144	1	145
	T	974 (5)	**38 (1)**	1010 (6)
USP-Lin	#P	149	8	157
	T	1130 (5)	97 (1)	1230 (6)
USP-Quad	#P	122	**15**	132
	T	1750 (31)	93 (6)	1840 (37)
CRA	#P	**154**	8	**162**
	T	**669 (5)**	133 (8)	**802 (13)**
VeriAbs	#P	116	2	118
	T	3430 (55, 2 OOM)	42 (23, 1 OOM)	3470 (78, 3 OOM)
ULTIMATE Automizer	#P	125	9	134
	T	2270 (51)	247 (17)	2520 (68)

Abstractionator's ability to generate non-linear invariants. The `nla-benchmark` programs from SV-COMP have "proposed invariants" at each loop header, as well as assertions at the end of the programs as post conditions; we obtained the NLA suite by removing these "proposed invariants". As a result, non-linear invariants must be *synthesized* in order to prove the post-condition (rather than simply *verifying* that the proposed invariant is an invariant, and implies the post-condition). The program from Fig. 1a is an example of a program in the NLA suite.

Comparison Tools. We have compared our techniques with ChilonInv [Kincaid et al. 2023], CRA [Kincaid et al. 2018], VeriAbs 1.5.1-2 [Afzal et al. 2019], and ULTIMATE Automizer 0.2.3 [Heizmann et al. 2009]. ChilonInv and CRA use a similar verification strategy of extracting implied solvable invariants of loop bodies to generate invariants of loops. VeriAbs and ULTIMATE Automizer are high performers at SV-COMP and provide context to the overall results. The strategies of ChilonInv, USP-Lin, and USP-Quad are all monotone algebraic analyses and the refinement technique of Cyphert et al. [2019] applies. Refinement is guaranteed to improve the precision of these three techniques, and so we have employed refinement in the comparison of these three strategies.

7.2 How Do Our Techniques Perform on a Suite of General Verification Tasks?

Table 1 gives the results of running each tool on the program verification benchmarks. Theoretically, in terms of precision, ChilonInv $\preceq$ USP-Lin $\preceq$ USP-Quad. However, this does not consider timeouts. Due to the increased power of USP-Lin and USP-Quad we would expect in terms of time taken ChilonInv $\preceq$ USP-Lin $\preceq$ USP-Quad, and this is what we see reflected in Table 1. In our experiments we found USP-Lin to outperform ChilonInv in both the loops category and the NLA category in terms of programs verified, at the expensive of additional running time. Theoretically, USP-Quad is stronger than ChilonInv and USP-Lin; however, the extra power comes at a price of running time. As can be seen from Table 1 USP-Quad performed *worse* on the loops category compared with ChilonInv and USP-Lin because of the number of timeouts. However, due to its strong non-linear reasoning capability, USP-Quad outperformed all the other tools on the difficult NLA benchmarks.

Theoretically, USP-Lin and USP-Quad are incomparable with the other tools. On one hand CRA's recurrence extraction procedure is weaker than the methods in this paper. However, CRA is also able

to produce invariants involving exponential and polynomial terms, whereas the techniques in this paper are only able to produce invariants involving polynomial terms. VeriAbs is a portfolio of many different techniques, such as bounded model checking and k-induction. ULTIMATE Automizer implements a trace abstraction algorithm. We note that while USP-Lin outperformed VeriAbs and ULTIMATE Automizer on the loops category, VeriAbs and ULTIMATE Automizer have additional capabilities such as the ability to produce counterexamples in the case when an assertion does not hold. This capability is outside the scope of USP-Lin and USP-Quad. Nevertheless, we find USP-Lin to be quite competitive on our benchmark suite. It outperformed all other tools on the loops category except for CRA, where it is behind by only 5 examples. Moreover, because of the success of USP-Quad on the NLA suite we find that powerful techniques that generate polynomial invariants are required to verify interesting programs found in the literature.

7.3 How Do Our Techniques Compare with Prior Methods for Complete Generation of Polynomial Invariants?

In this subsection, we consider how our method for generating polynomial invariants (which works on general programs) compares with the method presented by Humenberger et al. [2018] (which is complete, but applies to a more limited class of programs). The method of Humenberger et al. [2018] is implemented in a tool called Aligator. Both our methods of linear simulations as well as polynomial simulations are complete for loops whose bodies are described by a solvable polynomial map. Aligator is also complete for such loops. However, the completeness result of Humenberger et al. [2018] also extends to multi-path loops, where each branch is described by a solvable polynomial map (e.g., a loop of the form `while(*){ if (*) A else B }`, where `A` and `B` are described by solvable polynomial maps). On such an example, Aligator will produce *all* polynomial invariants of the loop, but Abstractionator cannot make the same guarantee. At the level of a loop we *abstract* the loop body to a solvable transition ideal. In the case of `while(*){ if (*) A else B }`, we create a solvable transition ideal that abstracts *both* `A` and `B`, which is strictly weaker than considering `A` and `B` separately as in Humenberger et al. [2018].

Table 2. USP-Lin and USP-Quad on the multi-path Aligator benchmarks.

	USP-Lin	USP-Quad
egcd.c	✗	✓
fermat2.c	✗	✓
lcm2.c	✗	✓
divbin.c	✗	✗
prodbin.c	✗	✓
dijkstra.c	✗	✗

We investigated how USP-Lin and USP-Quad perform on multi-path loops for which Aligator is complete, but our techniques are incomplete. A direct practical comparison between USP-Lin, USP-Quad, and Aligator is challenging because they take different formats as input. However, a subset of 6 programs from the multi-path benchmark suite of Aligator are applicable for our tool[7]. All of these 6 programs are found in the NLA suite discussed in Section 7.2. More detailed results of running USP-Lin and USP-Quad on these six examples can be found in Table 2. The completeness result of Humenberger et al. [2018] applies to these 6 programs, so given enough time Aligator would be able to verify all 6 of them. As can be seen from Table 2 USP-Lin, is unable to verify any of the 6 programs; however, USP-Quad is able to verify 4 of the 6. For the other 2 programs, the reason USP-Quad is unable to succeed is because those examples perform integer division in a situation in which no round-off occurs. In these programs this property is essentially encoded with an exponential invariant, which is outside the capabilities of USP-Quad. From the results of Table 2 we conclude that while the class of loops for which our technique is complete is a subset of Aligator's, we can still generate most of the invariants needed to prove correctness.

[7] Aligator and Abstractionator treat integer division differently

8 RELATED WORK

Polynomial abstractions of loops. The algorithm in Section 4 for computing the solvable reflection of a transition ideal can be seen as both a refinement of Kincaid et al. [2018]'s algorithm for extracting a solvable polynomial map from a transition formula and a generalization of Zhu and Kincaid [2021a]'s algorithm for computing deterministic affine reflections. Contrasting with [Kincaid et al. 2018], our algorithm is guaranteed to find a *best* abstraction as a solvable transition ideal, which is essential to prove monotonicity of our analysis. Contrasting with [Zhu and Kincaid 2021a], our algorithm consumes and produces transition ideals, which generalize affine relations.

Amrollahi et al. [2022] considers the problem of abstracting polynomial endomorphisms by solvable polynomial maps. The technique presented in Section 4 is more general in the sense that it operates on *transition ideals* rather than polynomial endomorphisms. A polynomial endomorphism $p : \mathbb{Q}[X] \to \mathbb{Q}[X]$ can be encoded as a transition ideal, generated by the polynomials $\{x' - p(x) : x \in X\}$, in which case the algorithm in Section 4.1 computes a solvable transition ideal (from which we may recover a solvable polynomial map—that is, our procedure serves the same purpose as of Amrollahi et al. [2022] for the inputs considered in that work). Moreover, our procedure provides a precision guarantee: it finds *solvable reflections* of transition ideals.

For example, consider the loop below (left) along with its solvable reflection (right)

$$\textbf{while} * \textbf{ do } \left\{ \begin{array}{l} x := x + z^2 + 1; \\ y := y - z^2; \\ z := z + (x + y)^2 \end{array} \right\} \qquad \underbrace{\langle \{a \mapsto x + y, b \mapsto z\}, \langle a' - a - 1, b' - b - a \rangle \rangle}_{\text{Solvable reflection}}$$

While the technique in [Amrollahi et al. 2022] is able to identify the first polynomial in the reflection (corresponding to the update $(x + y) := (x + y) + 1$) it cannot find the second ($z' := z + (x + y)^2$), since there is a non-linear dependence of z upon the "defective" variables x and y whose dynamics cannot be described by a solvable polynomial map.

Frohn et al. [2020] considers another related problem: *given a polynomial endomorphism p, is there a polynomial automorphism f such that $f^{-1} \circ p \circ f$ is solvable?* The procedure in Section 4 can also be used to solve this problem: if $\langle t, T \rangle$ is the solvable reflection of p, then such an f exists (namely, t) exactly when the ambient dimension of T is equal to that of p (and T has real eigenvalues). Section 4 generalizes this result in the sense that, (1) we operate on transition ideals rather than polynomial endomorphisms and (2) should the answer to the decision problem be "no", we may still compute an *abstraction* of p.

Complete polynomial invariant generation. Hrushovski et al. [2018, 2023]; Humenberger et al. [2018]; Kovács [2008]; Rodríguez-Carbonell and Kapur [2004] are complete methods for generating polynomial invariants on limited program structures. Our method matches the completeness results of these works on single loops whose bodies are described by solvable polynomial maps; however, the completeness result of each of these works cover additional situations.

Hrushovski et al. [2018, 2023] present a method that is complete for generating polynomial invariants for affine programs where all branching represents non-deterministic choice. Our methods have no issue analyzing such programs. Moreover, our method can also reason about programs with polynomial assignments as well as branching with conditionals. However, even though our method can reason about general affine programs, we can only guarantee completeness in the case of a loop whose body is described by a solvable polynomial map.

Kovács [2008] presents complete polynomial invariant generation for *P-solvable* loops. These are loops, with no branching, whose bodies have either Gosper-summable or c-finite assignments. As stated in Section 3.3 c-finite sequences are equivalent to solvable polynomial maps, and so our technique matches Kovács [2008] in that regard. However, while we always extract a solvable

transition ideal from a loop, solvable transition ideals are not powerful enough to capture certain Gosper-summable examples. Thus, while our method is monotone on such examples, it does not guarantee completeness. Humenberger et al. [2018] extends Kovács [2008] to the case of multi-path loops where each branch has a body with Gosper-summable or c-finite recurrence assignments. In the Gosper-summable case the comparison is the same as Kovács [2008]. In the multi-path c-finite case we are also not complete; however, we experimentally compare with Humenberger et al. [2018] in Section 7.3. In either the case of Kovács [2008] or Humenberger et al. [2018] they cannot make a completeness guarantee for programs having branching with conditionals or programs with arbitrary loop nesting.

Rodríguez-Carbonell and Kapur [2004]; Rodríguez-Carbonell and Kapur [2007] present a complete method for the case of a single multi-path loop where each branch has a body described by a c-finite recurrence. This matches the c-finite case of Humenberger et al. [2018], except Rodríguez-Carbonell and Kapur [2004]; Rodríguez-Carbonell and Kapur [2007] have an additional restriction on the eigenvalues of the c-finite recurrences (corresponding to the Θ_i variables of Eq. (2)). For Rodríguez-Carbonell and Kapur [2004]; Rodríguez-Carbonell and Kapur [2007] the eigenvalues are required to be positive and rational. We have no such restriction and so our method generalizes Rodríguez-Carbonell and Kapur [2004]; Rodríguez-Carbonell and Kapur [2007] in the case of a simple loop where the body is described by a c-finite recurrence. However, their completeness result goes beyond our capability in the case of a multi-path loop with positive rational eigenvalues.

Template Based Methods. Another method for generating polynomial invariants is to reduce the problem to constraint solving by supposing that the invariant takes the form of some parameterized template, and solving for the parameters [Cachera et al. 2012; Chatterjee et al. 2020; Goharshady et al. 2023; Kojima et al. 2018; Müller-Olm and Seidl 2004; Oliveira et al. 2016; Sankaranarayanan et al. 2004]. These methods have the benefit of being able to handle problems with arbitrary control flow. Furthermore, they are often complete for generating invariants that fit the given template. Many template methods consider all polynomials up to some bounded degree. In such cases when the desired polynomial is within the degree bound, template based methods have the potential to generate invariants for general programs that our method would theoretically miss. In contrast, our method does not require a degree bound. Even for linear simulations, there is no bound on the degree of the invariant our method calculates.

Monotone algebraic program analysis. A recent line of work has used the framework of algebraic program analysis to develop program analyses with monotonicity guarantees [Kincaid et al. 2023; Silverman and Kincaid 2019; Zhu and Kincaid 2021a,b]. In particular, Kincaid et al. [2023] proposes a monotone loop summarization algorithm based on the theory of linear integer/real rings. Our technique is complementary in the sense that our method computes stronger invariant polynomial equations than [Kincaid et al. 2023], but cannot synthesize invariant polynomial inequalities.

DATA-AVAILABILITY STATEMENT

An implementation of Abstractionator and experimental scripts are available on Zenodo [Cyphert and Kincaid 2023b]. Omitted proofs can be found in Cyphert and Kincaid [2023a].

ACKNOWLEDGMENTS

We would like to thank Tom Reps for his contributions to the discussions that lead to this paper. This work was supported, in part, by a gift from Rajiv and Ritu Batra, a Google PhD fellowship, and by the NSF under grant number 1942537. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the sponsoring entities.

REFERENCES

Mohammad Afzal, A. Asia, Avriti Chauhan, Bharti Chimdyalwar, Priyanka Darke, Advaita Datar, Shrawan Kumar, and R. Venkatesh. 2019. VeriAbs : Verification by Abstraction and Test Generation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1138–1141. https://doi.org/10.1109/ASE.2019.00121

Daneshvar Amrollahi, Ezio Bartocci, George Kenison, Laura Kovács, Marcel Moosbrugger, and Miroslav Stankovič. 2022. Solving Invariant Generation for Unsolvable Loops. In *Static Analysis*, Gagandeep Singh and Caterina Urban (Eds.). Springer Nature Switzerland, Cham, 19–43. https://doi.org/10.1007/978-3-031-22308-2_3

Dirk Beyer. 2023. Competition on Software Verification and Witness Validation: SV-COMP 2023. In *Tools and Algorithms for the Construction and Analysis of Systems*, Sriram Sankaranarayanan and Natasha Sharygina (Eds.). Springer Nature Switzerland, Cham. https://doi.org/10.1007/978-3-031-30820-8_29

Bruno Buchberger. 1976. A Theoretical Basis for the Reduction of Polynomials to Canonical Forms. *SIGSAM Bull.* 10, 3 (Aug. 1976), 19–29. https://doi.org/10.1145/1088216.1088219

David Cachera, Thomas Jensen, Arnaud Jobin, and Florent Kirchner. 2012. Inference of Polynomial Invariants for Imperative Programs: A Farewell to Gröbner Bases. In *Static Analysis*, Antoine Miné and David Schmidt (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 58–74. https://doi.org/10.1007/978-3-642-33125-1_7

Krishnendu Chatterjee, Hongfei Fu, Amir Kafshdar Goharshady, and Ehsan Kafshdar Goharshady. 2020. Polynomial Invariant Generation for Non-Deterministic Recursive Programs. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 672–687. https://doi.org/10.1145/3385412.3385969

David A. Cox, John Little, and Donal O'Shea. 2015. *Ideals, Varieties, and Algorithms* (4th ed.). Springer. https://doi.org/10.1007/978-3-319-16721-3

John Cyphert, Jason Breck, Zachary Kincaid, and Thomas W. Reps. 2019. Refinement of path expressions for static analysis. *Proc. ACM Program. Lang.* 3, POPL (2019), 45:1–45:29. https://doi.org/10.1145/3290358

John Cyphert and Zachary Kincaid. 2023a. Solvable Polynomial Ideals: The Ideal Reflection for Program Analysis. arXiv:2311.04092

John Cyphert and Zachary Kincaid. 2023b. *Solvable Polynomial Ideals: The Ideal Reflection for Program Analysis Artifact*. https://doi.org/10.5281/zenodo.10069757

Graham Everest, Alfred J van der Poorten, Igor Shparlinski, and Thomas Ward. 2003. *Recurrence Sequences*. Vol. 104. American Mathematical Society Providence, RI.

Azadeh Farzan and Zachary Kincaid. 2015. Compositional Recurrence Analysis. In *Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design* (Austin, Texas) *(FMCAD '15)*. FMCAD Inc, Austin, Texas, 57–64. https://doi.org/10.1109/FMCAD.2015.7542253

Jean-Charles Faugère. 1999. A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra* 139, 1 (1999), 61–88. https://doi.org/10.1016/S0022-4049(99)00005-5

Jean-Charles Faugère. 2010. FGb: A Library for Computing Gröbner Bases. In *Mathematical Software – ICMS 2010*. https://www-polsys.lip6.fr/~jcf/FGb/index.html

Florian Frohn, Marcel Hark, and Jürgen Giesl. 2020. Termination of Polynomial Loops. In *Static Analysis*, David Pichardie and Mihaela Sighireanu (Eds.). Springer International Publishing, Cham, 89–112. https://doi.org/10.1007/978-3-030-65474-0_5

Guoqiang Ge. 1993. *Algorithms Related to Multiplicative Representations of Algebraic Numbers*. Ph. D. Dissertation. Mathematics Department, University of California at Berkeley, Berkeley, CA.

Amir Kafshdar Goharshady, S. Hitarth, Fatemeh Mohammadi, and Harshit Jitendra Motwani. 2023. Algebro-Geometric Algorithms for Template-Based Synthesis of Polynomial Programs. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 100 (apr 2023), 30 pages. https://doi.org/10.1145/3586052

Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. 2009. Refinement of Trace Abstraction. In *Static Analysis, 16th International Symposium, SAS 2009, Los Angeles, CA, USA, August 9-11, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5673)*, Jens Palsberg and Zhendong Su (Eds.). Springer, 69–85. https://doi.org/10.1007/978-3-642-03237-0_7

Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. 2018. Polynomial Invariants for Affine Programs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (Oxford, United Kingdom) *(LICS '18)*. Association for Computing Machinery, New York, NY, USA, 530–539. https://doi.org/10.1145/3209108.3209142

Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. 2023. On Strongest Algebraic Program Invariants. *J. ACM* 70, 5, Article 29 (oct 2023), 22 pages. https://doi.org/10.1145/3614319

Andreas Humenberger, Maximilian Jaroschek, and Laura Kovács. 2018. Invariant Generation for Multi-Path Loops with Polynomial Assignments. In *Verification, Model Checking, and Abstract Interpretation*, Isil Dillig and Jens Palsberg (Eds.). Springer International Publishing, Cham, 226–246. https://doi.org/10.1007/978-3-319-73721-8_11

Fredrik Johansson. 2017. Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. *IEEE Trans. Comput.* 66 (2017), 1281–1292. Issue 8. https://doi.org/10.1109/TC.2017.2690633

Manuel Kauers and Burkhard Zimmermann. 2008. Computing the algebraic relations of C-finite sequences and multisequences. *Journal of Symbolic Computation* 43, 11 (2008), 787–803. https://doi.org/10.1016/j.jsc.2008.03.002

Zachary Kincaid, John Cyphert, Jason Breck, and Thomas Reps. 2018. Non-Linear Reasoning for Invariant Synthesis. *Proc. ACM Program. Lang.* 2, POPL, Article 54 (dec 2018), 33 pages. https://doi.org/10.1145/3158142

Zachary Kincaid, Nicolas Koh, and Shaowei Zhu. 2023. When Less Is More: Consequence-Finding in a Weak Theory of Arithmetic. *Proc. ACM Program. Lang.* 7, POPL, Article 44 (jan 2023), 33 pages. https://doi.org/10.1145/3571237

Zachary Kincaid, Thomas Reps, and John Cyphert. 2021. Algebraic Program Analysis. In *Computer Aided Verification*, Alexandra Silva and K. Rustan M. Leino (Eds.). Springer International Publishing, Cham, 46–83. https://doi.org/10.1007/978-3-030-81685-8_3

Kensuke Kojima, Minoru Kinoshita, and Kohei Suenaga. 2018. Generalized homogeneous polynomials for efficient template-based nonlinear invariant synthesis. *Theoretical Computer Science* 747 (2018), 33–47. https://doi.org/10.1016/j.tcs.2018.06.005

Laura Kovács. 2008. Reasoning Algebraically About P-Solvable Loops. In *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 249–264. https://doi.org/10.1007/978-3-540-78800-3_18

Markus Müller-Olm and Helmut Seidl. 2004. A Note on Karr's Algorithm. In *Automata, Languages and Programming*, Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1016–1028. https://doi.org/10.1007/978-3-540-27836-8_85

Markus Müller-Olm and Helmut Seidl. 2004. Computing polynomial program invariants. *Inform. Process. Lett.* 91, 5 (2004), 233–244. https://doi.org/10.1016/j.ipl.2004.05.004

Steven De Oliveira, Saddek Bensalem, and Virgile Prevosto. 2016. Polynomial invariants by linear algebra. *Lecture Notes in Computer Science* 9938 LNCS (2016), 479–494. https://doi.org/10.1007/978-3-319-46520-3_30 Conference of 14th International Symposium on Automated Technology for Verification and Analysis, ATVA 2016 ; Conference Date: 17 October 2016 Through 20 October 2016; Conference Code:185289.

Enric Rodríguez-Carbonell and Deepak Kapur. 2004. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation* (Santander, Spain) *(ISSAC '04)*. Association for Computing Machinery, New York, NY, USA, 266–273. https://doi.org/10.1145/1005285.1005324

Enric Rodríguez-Carbonell and Deepak Kapur. 2007. Generating all polynomial invariants in simple loops. *Journal of Symbolic Computation* 42, 4 (2007), 443–476. https://doi.org/10.1016/j.jsc.2007.01.002

Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2004. Non-Linear Loop Invariant Generation Using GröBner Bases. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Venice, Italy) *(POPL '04)*. Association for Computing Machinery, New York, NY, USA, 318–329. https://doi.org/10.1145/964001.964028

Jake Silverman and Zachary Kincaid. 2019. Loop Summarization with Rational Vector Addition Systems. In *Computer Aided Verification*, Isil Dillig and Serdar Tasiran (Eds.). Springer International Publishing, Cham, 97–115. https://doi.org/10.1007/978-3-030-25543-5_7

The FLINT team. 2023. *FLINT: Fast Library for Number Theory*. Version 2.9.0, https://flintlib.org.

S. M. Ulam and John von Neumann. 1947. On combination of stochastic and deterministic processes. Summer meeting of the American Mathematical Society.

Philipp Wendler and Dirk Beyer. 2023. Bench Exec 3.16. https://github.com/sosy-lab/benchexec.

Shaowei Zhu and Zachary Kincaid. 2021a. Reflections on Termination of Linear Loops. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part II*. Springer-Verlag, Berlin, Heidelberg, 51–74. https://doi.org/10.1007/978-3-030-81688-9_3

Shaowei Zhu and Zachary Kincaid. 2021b. Termination Analysis without the Tears. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 1296–1311. https://doi.org/10.1145/3453483.3454110

Received 2023-07-11; accepted 2023-11-07