Automatic Generation of Polynomial Loop Invariants for Imperative Programs*

Enric Rodríguez Carbonell, Deepak Kapur
November 2003

Technical University of Catalonia, Barcelona
erodri@lsi.upc.es
University of New Mexico, Albuquerque
kapur@cs.unm.edu

Abstract. A general framework is presented for automating the discovery of loop invariants for imperative programs. Theoretical results about the correctness and completeness of the proposed method are given. More importantly, it is shown how this abstract approach can be used to automatically infer polynomial invariants. The method has been implemented in Maple. Evidence of its practical interest is shown by means of several non-trivial examples, for which the polynomial loop invariants generated are directly applicable for proving correctness by means of a simple verifier.

1 Introduction

The decade of the seventies saw considerable focus on research related to program verification based on Floyd-Hoare-Dijkstra's axiomatic semantics approach, using pre/postconditions and loop invariants. The framework has now become an integral part of CS curriculum, where students are typically taught to practise such techniques by hand. Mechanization of the approach did not catch on partly due to the lack of automated reasoning tools then available[1], and also partly due to the huge effort required in program annotation.

Regarding automated reasoning there has been considerable progress, which has led to variants of the approach in the form of static analysis of programs (type checking, type inference, extended static checking, other properties of variables based on abstract interpretations), model checking of computational structures (especially hardware) as well as software and hardware verification using automated theorem proving. For example, type inference (with the types of program variables being weaker forms of invariants) is now widely considered a useful feature supported in many functional and logical languages such as ML.

* This research was partially supported by an NSF ITR award CCR-0113611, the Prince of Asturias Endowed Chair in Information Science and Technology at the University of New Mexico and a FPU grant from the Spanish Secretaría de Estado de Educación y Universidades.

[1] See however some recent attempts to use automated theorem provers in teaching program verification in introductory programming classes [WS03].

However, the annotation burden remains. While input/output specifications also work as useful documentation, loop invariants do not provide comparable benefits. Completely specifying invariants in loops is tedious, due to the substantial amount of detail it requires; and also redundant, since the invariants repeat information that can be inferred from the program. In this paper we aim at reducing this annotation overhead, along the lines of the Houdini annotation assistant [FL01] or the Daikon tool [NE96].

We present a general framework for finding invariants of loops in imperative programs with conditional statements and assignments. A generic procedure based on forward propagation for computing loop invariants as fixed points is shown. We give conditions on the domains manipulated by the programs and on the language used to express the invariants so that the procedure can be implemented. Proofs of correctness and completeness are shown.

This abstract framework is then instantiated to consider programs in which loop invariants can be formulated as polynomial equations. It is shown that if assignments are invertible and their powers have polynomial structure, as is the case with invertible affine assignments, then polynomial loop invariants can be automatically generated using Gröbner bases (see [CLO98] or [AL94] as an introduction to Gröbner basis algorithm, commutative algebra and algebraic geometry, or to [BW93] for a more comprehensive treatment). It can be shown that the procedure for computing polynomial loop invariants terminates in $2m+1$ or fewer iterations, where m is the number of variables changing in the loop. This technique has been implemented in the mathematical tool Maple and has been applied to numerous programs, some of which are used for illustration. Moreover, the results obtained with our method have been used for automatically proving the correctness of the programs with a verifier.

The rest of the paper is organized as follows. In next subsection, related work is briefly reviewed. Section 2 introduces the general framework for computing loop invariants: the programming model is presented and necessary properties of the language for expressing invariants are studied. In Section 3 we instantiate this abstract framework to the case of polynomial equalities and obtain a procedure for finding polynomial invariants. Section 4 focuses on the use of algebraic geometry and commutative algebra in the implementation of the procedure. In Section 5 we give some illustrative examples. Section 6 is devoted to the application of polynomial invariant inference to program verification. Finally Section 7 concludes with a discussion of the advantatges and drawbacks of the approach and an overview on further research.

1.1 Related Work

The techniques presented here build upon the *difference equations method* ([EGLW72]), which proceeds in two steps: first, by means of recurrence equations (also called *difference equations*), an explicit expression is found for the value of each variable as a function of the number of loop iterations s, other variables that remain constant in the loop, and the input values; then the variable s is eliminated to obtain invariant predicates. Our work overcomes the difficulties in

dealing with loops with conditionals, for which traditionally only heuristics were available under this approach.

In a different direction, Karr showed in [Kar76] an algorithm for finding linear equalities between variables at any program point. Later on this work was extended by Cousot and Halbwachs ([CH78]), who applied the model of abstract interpretation (see [CC77]) to finding linear inequalities. Like our techniques, both are based in forward propagation and fixed point computation (see [Wei75]). But whereas Karr obtained termination directly, Cousot and Halbwachs had to introduce a *widening* operator ∇ which computes an upper approximation of the set of states. Our method can be regarded as computing also an upper approximation, which however is fine enough to guarantee completeness. Furthermore, the results that it yields complement the linear inequalities obtained by means of these other techniques.

More recently, in [CSS03] Colón et al. have used non-linear constraint solving and quantifier elimination to attack the same problem of linear inequalities. Although we also have to eliminate existentially quantified variables in our approach, since we are dealing with polynomial equalities we can apply Gröbner bases and elimination theory, which are not as costly as the general methods for quantifier elimination like cylindrical algebraic decomposition (CAD).

Another extension of Karr's ideas has been carried out by Müller-Olm and Seidl ([Mar03]), who find interprocedural polynomial equalities of bounded degree in programs with affine assigments using backward propagation and weakest preconditions, instead of forward propagation and strongest postconditions. When they can be applied, our techniques have the advantatge that no bound on the degree is necessary, and that polynomial assignments whose powers have polynomial structure are allowed.

2 General Framework

2.1 Programming Model

In this section we present our programming model and the way we abstract information from the original loops.

Let x_1, x_2, ..., x_m be the variables which change their value during the execution of the loop. We denote by $\bar{x}$ the tuple of these variables. Let Σ' be the set to which the variables belong according to their type declaration. For instance, if we had two integer variables and a rational variable it would be $\Sigma' = \mathbb{Z}^2 \times \mathbb{Q}$.

The only instructions we allow in loops are assignments and non-deterministic conditionals. As assignments can be composed and nested conditionals can be merged, using the notation of Dijkstra's guarded command language ([Dij76]) we can assume that loops have the form:

while $E'(\bar{x})$ **do**
  if $C'_1(\bar{x}) \rightarrow \bar{x} := f'_1(\bar{x})$;
  ...

$[]\ C'_i(\bar{x}) \rightarrow \bar{x} := f'_i(\bar{x});$
...
$[]\ C'_n(\bar{x}) \rightarrow \bar{x} := f'_n(\bar{x});$
end if
end while

where $E', C'_i : \Sigma' \rightarrow \{true, false\}$ and $f'_i : \Sigma' \rightarrow \Sigma'$ for $1 \leq i \leq n$.

Given such a loop, we will apply our techniques to an abstraction of it. Let $\Sigma \supseteq \Sigma'$ be a superset of the original state space Σ' and let $E, C_i : \Sigma \rightarrow \{true, false\}$ and $f_i : \Sigma \rightarrow \Sigma$ $(1 \leq i \leq n)$ be such that

i) $E'(\bar{x}) \Rightarrow E(\bar{x})$.
ii) $\forall i : 1 \leq i \leq n : E'(\bar{x}) \wedge C'_i(\bar{x}) \Rightarrow C_i(\bar{x}) \wedge f'_i(\bar{x}) = f_i(\bar{x})$.

Notice that if the E, C_i are just *true* and $f_i = f'_i$, then these conditions are immediately satisfied.

Our method will find invariants for the abstracted loop, which has the following simple structure:

while $E(\bar{x})$ **do**
if $C_1(\bar{x}) \rightarrow \bar{x} := f_1(\bar{x});$
...
$[]\ C_i(\bar{x}) \rightarrow \bar{x} := f_i(\bar{x});$
...
$[]\ C_n(\bar{x}) \rightarrow \bar{x} := f_n(\bar{x});$
end if
end while

In order to motivate this abstraction process, let us consider the next program, which is a product variant of a well-known algorithm for exponentiation:

function product (X, Y: integer) **returns** z: integer
{ Pre: $X \geq 0 \wedge Y \geq 0$}
var x, y: integer **end var**
$\langle x,\ y,\ z\rangle := \langle X,\ Y,\ 0\rangle$;
while $y \neq 0$ **do**
if y `mod` $2 = 1$ **then** $\langle x, y, z\rangle := \langle 2 * x, (y-1)$ `div` $2, x + z\rangle$;
$[]\ y$ `mod` $2 = 0$ **then** $\langle x, y, z\rangle := \langle 2 * x, y$ `div` $2, z\rangle$;
end if
end while
{ Post: $z = X \cdot Y$}
end function

As we will see later on, when looking for polynomial invariants we ignore the guards in the **if** and **while** statements, and require that the variables lie in a field, like $\mathbb{Q}$. If we have integer variables like in this case, we may consider the variables as rational numbers. However, all operations between variables must

be then operations in $\mathbb{Q}$. The only problem in this case is integer division `div`, which is not defined in $\mathbb{Q}$. The solution is to replace the integer divisions `div` by rational divisions /, which is correct provided no integer division truncation in the original program occurs. Therefore in the abstraction process we have to guarantee that whenever a division takes place in the original loop the division is exact, which is ensured by the conditions *i*) and *ii*). Thus, after taking $\mathbb{Q}$ as the type of the variables in this case our loop would be abstracted as follows:

var x, y: rational **end var**
$\langle x,\ y,\ z\rangle := \langle X,\ Y,\ 0\rangle$;
while *true* **do**
  if $true \rightarrow \langle x,\ y,\ z\rangle := \langle 2 * x,\ (y-1)/2,\ x+z\rangle$;
  [] $true \rightarrow \langle x,\ y,\ z\rangle := \langle 2 * x,\ y/2,\ z\rangle$;
  end if
end while

It is clear that conditions *i*) and *ii*) are indeed satisfied for this example.

2.2 Invariants of Abstracted Loops

In this section we study the invariants of the abstracted loops and their properties.

We start with the language used to express invariants, which we denote by $\mathcal{R}$. For simplicity, we use (many-sorted) first-order predicate calculus with equality as such a language. Later on when we instantiate this abstract framework, we will further restrict this language. Of course, our goal is to capture the semantics of the body of the loop using the strongest possible invariant expressible in $\mathcal{R}$. We characterize properties of such a strongest invariant as well as the languages expressive enough to admit such strong invariants.

Typically a formula serving as an invariant has two sets of free variables:

• those that represent program variables that change their value during the execution of the loop, denoted by $\bar{x}$.

• those that represent initial, usually unknown values of the variables before entering the loop, denoted by $\bar{x}^*$.

To make this explicit, in general formulas in the language $\mathcal{R}$ are written as $R(\bar{x}, \bar{x}^*)$. Then we say that $R(\bar{x}, \bar{x}^*)$ holds if $R(\bar{\alpha}, \bar{\alpha}^*)$ is valid $\forall \bar{\alpha}, \bar{\alpha}^* \in \Sigma$ (i.e., is true for every possible interpretation of the free variables).

We next define the class of invariants our method is aimed at:

Definition 1. *Let $R_0(\bar{x}^*) \in \mathcal{R}$ be a formula not depending on $\bar{x}$. A formula $R \in \mathcal{R}$ is* $\mathcal{R}$-invariant *with respect to R_0 if:*

i) $R_0(\bar{x}^*) \Rightarrow R(\bar{x}^*, \bar{x}^*)$

ii) $\forall i : 1 \leq i \leq n,\ R(\bar{x}, \bar{x}^*) \wedge E(\bar{x}) \wedge C_i(\bar{x}) \Rightarrow R(f_i(\bar{x}), \bar{x}^*)$.

In this definition R_0 represents a a precondition for the loop, i.e. a formula which is satisfied by the initial values of the variables before entering the loop:

$\forall \bar{\alpha}^* \in \Sigma$ initial value before entering the loop $R_0(\bar{\alpha}^*)$ is valid. For example, in the case of polynomial equalities and the product function in the previous section, $z^* = 0$ would be such a formula. If no formula is known to hold for the initial values, we can always take *true*.

It is easy to see that if R is an $\mathcal{R}$-invariant of the abstracted loop, then R is also an invariant of the original loop.

Lemma 1. *If $R(\bar{x}, \bar{x}^*)$ is $\mathcal{R}$-invariant with respect to R_0, then R is an invariant of the original loop.*

Proof. As $\forall \bar{\alpha}^* \in \Sigma' \subseteq \Sigma$ initial state before executing the loop we have $R_0(\bar{\alpha}^*)$ and $R_0(\bar{x}^*) \Rightarrow R(\bar{x}^*, \bar{x}^*)$ by $\mathcal{R}$-invariance, we have that $R(\bar{\alpha}^*, \bar{\alpha}^*)$ holds.

Now we have to prove that

$$\forall \bar{\alpha}, \bar{\alpha}^* \in \Sigma' \ \forall i : 1 \leq i \leq n \ R(\bar{\alpha}, \bar{\alpha}^*) \wedge E'(\bar{\alpha}) \wedge C'_i(\bar{\alpha}) \Rightarrow R(f'_i(\bar{\alpha}), \bar{\alpha}^*)$$

But this is a consequence of $\forall \bar{\alpha}, \bar{\alpha}^* \in \Sigma' \subseteq \Sigma$, $\forall i : 1 \leq i \leq n$:
$R(\bar{\alpha}, \bar{\alpha}^*) \wedge E(\bar{\alpha}) \wedge C_i(\bar{\alpha}) \Rightarrow R(f_i(\bar{\alpha}), \bar{\alpha}^*)$
$E'(\bar{\alpha}) \Rightarrow E(\bar{\alpha})$
$E'(\bar{\alpha}) \wedge C'_i(\bar{\alpha}) \Rightarrow C_i(\bar{\alpha}) \wedge f'_i(\bar{\alpha}) = f_i(\bar{\alpha})$.
□

To capture the semantics of the body of the loop, our goal is to compute the strongest possible invariant expressible in the language under consideration.

Definition 2. *The language $\mathcal{R}$ is* expressive *for a given loop if $\exists R_\infty \in \mathcal{R}$ such that*

1. *R_∞ is $\mathcal{R}$-invariant.*
2. *$\forall R$ $\mathcal{R}$-invariant, $R_\infty(\bar{x}, \bar{x}^*) \Rightarrow R(\bar{x}, \bar{x}^*)$.*

Notice that, if it exists, R_∞ is unique modulo equivalence.

In subsequent sections, we show examples of programs and languages which are expressive for writing strong invariants for these programs.

2.3 Fixed-Point Procedure for Computing Invariants

In this section we give a procedure for computing the strongest $\mathcal{R}$-invariant R_∞ of loops of the form described in Section 2.1 such that the language $\mathcal{R}$ is expressive for them.

Assume that E, the loop test, as well as the C_i, the tests in the conditional statements, are expressible in the language. Furthermore, assume that the assignments f_i are functions also expressible in the language. Let R_0 stand for the formula expressing a relation satisfied by the initial values of the variables.

The *General Procedure* computes successive approximations of the strongest $\mathcal{R}$-invariant R_∞ based on forward propagation until reaching a fixed point. The objective is to have computed R_∞ if the procedure terminates.

Forward Propagation Semantics If a formula $R(\bar{x}, \bar{x}^*)$ holds at the beginning of the body of the loop, the loop test $E(\bar{x})$ is true and the i-th conditional branch is executed, then the strongest postcondition at the end of the body of the loop is

$$\{\exists \bar{y}(\bar{x} = f_i(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge E(\bar{y}) \wedge C_i(\bar{y}))\}$$

Such a single step of forward propagation leads to a new approximation of the property held at the beginning of the loop.

$$\Big\{R(\bar{x}) \vee \Big(\bigvee_{i=1}^{n} \exists \bar{y}(\bar{x} = f_i(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge E(\bar{y}) \wedge C_i(\bar{y}))\Big)\Big\}$$

Now, assuming that the s-th power of an assignment function f_i is also expressible in the language, if the $i-$th branch is executed s-times, then the strongest postcondition is:

$$\Big\{\exists \bar{y}\Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y}))\big)\Big)\Big\}$$

Given that the number of iterations s is undetermined we have to take an infinite disjunction (which is not a formula anymore in the language unless existential quantifiers are used)

$$\Big\{\bigvee_{s=1}^{\infty} \Big(\exists \bar{y}\Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y}))\big)\Big)\Big)\Big)\Big\}$$

Finally, in a non-deterministic loop body, each of the branches can be executed and that too arbitrarily many times. This results in an "infinite" formula capturing the program state at the beginning of the loop after an undetermined branch has been executed arbitarily many times in the body of the loop.

$$\Big\{R(\bar{x}, \bar{x}^*) \vee \Big(\bigvee_{i=1}^{n} \bigvee_{s=1}^{\infty} \Big(\exists \bar{y}\Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y}))\big)\Big)\Big)\Big)\Big)\Big\}$$

And, furthermore, j-th assignment may be executed s_j number of times after i-th assignment has been executed s_i number of times. In general, there is intermixing. Thus, we keep on iterating until reaching a fixed point (or going forever). Formulas expressing the program state corresponding to such execution paths are computed using a procedure below.

In a highly expressive language, the above formula can perhaps be expressed using existential quantifiers. If the language does not permit existential quantifiers (as will be the case for concrete cases considered later), the language must have some desired properties in order to express sufficiently strong approximation of the above formulas. This is discussed below.

Definition 3. *The language* $\mathcal{R}$ *is* disjunctively closed *if*
$\forall R, S \in \mathcal{R}\ \exists T := R \sqcup S \in \mathcal{R}$ *such that*

$$R(\bar{x}, \bar{x}^*) \vee S(\bar{x}, \bar{x}^*) \Rightarrow T(\bar{x}, \bar{x}^*)$$

and $\forall T' \in \mathcal{R}$ *such that*

$$R(\bar{x}, \bar{x}^*) \vee S(\bar{x}, \bar{x}^*) \Rightarrow T'(\bar{x}, \bar{x}^*)$$

then $T(\bar{x}, \bar{x}^*) \Rightarrow T'(\bar{x}, \bar{x}^*)$.

Any first-order language is disjunctively closed, as we can take $\sqcup = \vee$. If the language consists of polynomial equations as atomic formulas and conjunction is the only boolean operation permitted, the language is disjunctively closed since if R and S are each a conjunction of polynomial equations, there is an equivalent conjunction of polynomial equations $R \sqcup S$ with the above property, as we will see. For the language of linear inequalities closed under conjunction (which is the language considered by Cousot and Halbwachs in [CH78]) we also have disjunctive closedness but not equivalence: given any R and S, which are conjunctions of linear inequalities, $R \sqcup S$ must be defined as the convex hull of R and S, although in this case $R \sqcup S$ is not equivalent to $R \vee S$ in general.

In the following, we impose additional requirements on the language so as to be able to eliminate existential quantifiers.

For an i-th conditional branch, let us assume that $\exists \varphi_i(R) \in \mathcal{R}$ the strongest formula in the language implied by

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y})) \big) \Big) \Big)$$

without any existential quantifiers or equivalent. As is obvious, two different kinds of existential variables need to be eliminated: the variables $\bar{y}$, which are related to the forward semantics of the assignment statement, and the loop counter s.

Definition 4. $\mathcal{R}$ allows quantifier elimination *if* $\forall R \in \mathcal{R}$, $\forall i : 1 \leq i \leq n$ $\exists T := \varphi_i(R) \in \mathcal{R}$ *such that*

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y})) \big) \Big) \Big) \Rightarrow T(\bar{x}, \bar{x}^*)$$

and $\forall T' \in \mathcal{R}$ *such that*

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y})) \big) \Big) \Big) \Rightarrow T'(\bar{x}, \bar{x}^*)$$

then $T(\bar{x}, \bar{x}^*) \Rightarrow T'(\bar{x}, \bar{x}^*)$.

For the case where $E = C_i = true$ for $1 \leq i \leq n$, $\mathcal{R}$ allows quantifier elimination if $\forall R \in \mathcal{R}$, $\forall i : 1 \leq i \leq n$ $\exists T := \varphi_i(R) \in \mathcal{R}$ such that

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \Big) \Rightarrow T(\bar{x}, \bar{x}^*)$$

and $\forall T' \in \mathcal{R}$ such that

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \Big) \Rightarrow T'(\bar{x}, \bar{x}^*)$$

then $T(\bar{x}, \bar{x}^*) \Rightarrow T'(\bar{x}, \bar{x}^*)$.

After replacing disjunction $\vee$ by $\sqcup$ and eliminating existential quantifiers by means of the φ_i's, we get the following procedure. It starts assigning to the variable R an initial formula satisfied by the initial values of the variables in the loop. This variable R stores the formula corresponding to the successive approximations of the invariant.

Input.
The mappings $f_1, ..., f_n$ of the assignments
A formula R_0 satisfied by the initial values
Output.
An $\mathcal{R}$-invariant formula R with respect to R_0

var R, R' : formulas in $\mathcal{R}$ **end var**

$R' := false$
$R := \Big(\bigwedge_{j=1}^{m} (x_j = x_j^*) \Big) \wedge R_0$
while $R' \not\Rightarrow R$ **do**
$R' := R$
$R := R \sqcup$ [2] $\Big(\bigsqcup_{i=1}^{n} \varphi_i(R) \Big)$
end while
return R

If the procedure terminates, then a fixed point of the formula mapping

$$R \mapsto R \sqcup \Big(\bigsqcup_{i=1}^{n} \varphi_i(R) \Big)$$

is computed.

The correctness and completeness of the above procedure are proved in Appendix A. In particular, we show that the formula R satisfies that :

[2] The use of $\sqcup$ approximating $\vee$ here may not be sufficient to guarantee the termination of the procedure computing the fixed point. Using a widening operator ∇ instead of $\sqcup$ (see Cousot and Halbwachs [CH78]) as a further approximation of $\vee$ can ensure the termination of the procedure, probably at the cost of completeness

– if the procedure terminates, R is an $\mathcal{R}$-invariant of the abstracted loop
– $R \Rightarrow R_\infty$ is an invariant of the procedure.

So if the procedure terminates $R \Rightarrow R_\infty$ and $R_\infty \Rightarrow R$, which finally leads to the result that the above procedure on termination indeed computes the strongest $\mathcal{R}$-invariant of the abstracted loop.

In summary, the language expressing loop invariants must satisfy the following properties:

– the language should be *expressive* enough for the class of simple loops under consideration in the sense that for each such loop, there should exist a formula in the language R_∞ that is invariant in the loop and furthemore, every invariant of the loop expressible in the language is implied by it.
– the language should be *disjunctively closed*, i.e., given any two formulas R, S in the language, it should be possible to compute a formula $R \sqcup S$ in it such that $R \sqcup S$ is the strongest formula implied by the disjunction of R and S.
– the language must *allow quantifier elimination* in the sense that given R and i there must exist a strongest formula $\varphi_i(R)$ implied by

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y})) \big) \Big) \Big)$$

For termination, it is further necessary that the fixed point can be computed in finitely many steps. For that, appropriate widening operators can be chosen to speed up the computation of the $\mathcal{R}$-invariant; literature on widening operations introduced in the context of abstract interpretation [CC77] can be extremely useful here.

3 Conjunctions of Polynomial Equations as Invariants

We consider the language of polynomial equations closed under conjunction and show that it satisfies all the requirements discussed in the previous section. Then we derive a procedure for finding polynomial invariants from the General Procedure which under restrictions over the assignment mappings is guaranteed to terminate.

3.1 Preliminaries

A mapping $g : \mathbb{Q}^r \to \mathbb{Q}^s$ is said to be affine if it is of the form $g(\bar{x}) = A\bar{x} + b$, where A is an $s \times r$ matrix with coefficients in $\mathbb{Q}$, and $b \in \mathbb{Q}^s$.

We denote by $\mathbb{Q}[\bar{z}] = \mathbb{Q}[z_1, ..., z_l]$ the set of polynomials with coefficients in $\mathbb{Q}$ in the variables $z_1, ..., z_l$. Given a set of polynomials $S \subseteq \mathbb{Q}[\bar{z}]$, the ideal spanned by S is written as $\langle S \rangle_{\mathbb{Q}[\bar{z}]}$ or simply as $\langle S \rangle$, and the *variety* of S is defined as its set of zeroes, $\mathbb{V}(S) = \{\bar{\alpha} \in \mathbb{Q}^l |\ p(\bar{\alpha}) = 0\ \forall p \in S\}$. Reciprocally, if $A \subseteq \mathbb{Q}^l$ the ideal $\mathbb{I}(A) = \{p \in \mathbb{Q}[\bar{z}] |\ p(\bar{\alpha}) = 0\ \forall \bar{\alpha} \in A\}$ is the *anihilator* of A. If $I \subseteq \mathbb{Q}[\bar{z}]$, we will write $\mathbb{IV}(I)$ instead of $\mathbb{I}(\mathbb{V}(I))$.

3.2 The Language

From now on we will restrict ourselves to the case $\mathcal{R} = \mathcal{P} = \{\bigwedge_{p\in P}\ p(\bar{x}, \bar{x}^*) = 0 \mid P \subset \mathbb{Q}[\bar{x}, \bar{x}^*] \text{ finite}\}$, that is to say, finite conjunctions of polynomial equations with rational coefficients. As we pointed out in Section 2.1, we require that the variables lie in a field so that we can use algebraic geometry. Assuming that the variables in our original program are of type integer or rational, we can take $\Sigma = \mathbb{Q}^m$, where m is the number of changing variables in the loop. Moreover, we ignore the guards from the original program, that is to say, for $1 \leq i \leq n$ we take $C_i = E = \mathit{true}$. For the time being we assume that the mappings of the assignments f_i are any polynomial mappings.

Now, let us denote by $\mathcal{I}$ the set of ideals of $\mathbb{Q}[\bar{x}, \bar{x}^*]$. Then we can establish the following correspondences between *formulas* in $\mathcal{P}$ and *ideals* in $\mathcal{I}$:

$$\phi : \mathcal{P} \longrightarrow \mathcal{I}, \qquad \bigwedge_{p\in P} (p = 0) \longmapsto \langle P \rangle$$

$$\psi : \mathcal{I} \longrightarrow \mathcal{P}, \qquad I \longmapsto \bigwedge_{p\in basis(I)} (p = 0)$$

where *basis* returns a finite basis of generators of a given ideal; by Hilbert's basis theorem, every ideal in $\mathbb{Q}[\bar{x}, \bar{x}^*]$ has a finite basis. It is clear then that both ϕ and ψ are well-defined. Moreover, $\forall R \in \mathcal{P}$ we have that

$$R(\bar{\alpha}, \bar{\alpha}^*) \text{ holds } \Leftrightarrow (\bar{\alpha}, \bar{\alpha}^*) \in \mathbb{V}(\phi(R))$$

and $\forall R, S \in \mathcal{P}$,

$$\Big(R(\bar{x}, \bar{x}^*) \Rightarrow S(\bar{x}, \bar{x}^*)\Big) \Leftrightarrow$$

$$\Leftrightarrow \mathbb{V}(\phi(R)) \subseteq \mathbb{V}(\phi(S)) \Leftrightarrow$$

$$\Leftrightarrow \mathbb{IV}(\phi(R)) \supseteq \mathbb{IV}(\phi(S))$$

3.3 Expressiveness

We must show that there exists for a given loop a polynomial invariant that implies all other polynomial invariants. The idea is to take the basis of the ideal generated by all polynomials appearing in the polynomial invariants of the loop. By Hilbert's basis theorem, such an infinite basis has an equivalent finite basis. Then we can take R_∞ as the conjunction of the polynomial equations corresponding to each polynomial in such a basis.

More formally:

Lemma 2. *$\mathcal{P}$ is expressive (for any given loop with polynomial assignments after ignoring the guards).*

Proof. We have to check that $\exists R_\infty \in \mathcal{P}$ such that

i) R_∞ is $\mathcal{P}$-invariant.

ii) $\forall R$ $\mathcal{P}$-invariant $R_\infty(\bar{x}, \bar{x}^*) \Rightarrow R(\bar{x}, \bar{x}^*)$

In our case a formula $R \in \mathcal{P}$ is $\mathcal{P}$-invariant if

- $R_0(\bar{x}^*) \Rightarrow R(\bar{x}^*, \bar{x}^*)$
- $\forall i : 1 \leq i \leq n$ $R(\bar{x}, \bar{x}^*) \Rightarrow R(f_i(\bar{x}), \bar{x}^*)$

First of all we define the ideal

$$P_\infty = \left\langle \bigcup \{\phi(R) |\ R \text{ is } \mathcal{P}\text{-invariant with respect to } R_0\} \right\rangle$$

By Hilbert's basis theorem, there exists a finite set of polynomials p_1, ..., $p_k \in \mathbb{Q}[\bar{x}, \bar{x}^*]$ such that $\langle p_1, ..., p_k \rangle = P_\infty$. Therefore for any R formula $\mathcal{P}$-invariant w.r.t. R_0 we have that $\phi(R) \subseteq \langle p_1, ..., p_k \rangle$. But clearly this implies that $\forall \bar{\alpha}, \bar{\alpha}^* \in \mathbb{Q}^m$

$$\bigwedge_{l=1}^{k} p_l(\bar{\alpha}, \bar{\alpha}^*) = 0 \Rightarrow R(\bar{\alpha}, \bar{\alpha}^*)$$

Therefore if we define $R_\infty := \left(\bigwedge_{l=1}^{k}\ p_l(\bar{x}, \bar{x}^*) = 0 \right)$ we satisfy *ii*).

Now we have to prove that *i*) also holds, that is to say, that R_∞ is $\mathcal{P}$-invariant. There exist $\mathcal{P}$-invariant formulas $Q_1, ..., Q_r$ such that $\forall l : 1 \leq l \leq k$ $\exists q_{1l} \in \phi(Q_1)$, ...,$q_{rl} \in \phi(Q_r)$ satisfying

$$p_l = \sum_{j=1}^{r} q_{jl}$$

Using the $\mathcal{P}$-invariance of the Q_j and that $\phi(Q_j) \subseteq \langle p_1, ..., p_k \rangle$, it is straightforward to prove that

$$R_0(\bar{x}^*) \Rightarrow \bigwedge_{l=1}^{k}\ p_l(\bar{x}^*, \bar{x}^*) = 0$$

and that $\forall i : 1 \leq i \leq n$ we have

$$\bigwedge_{l=1}^{k} p_l(\bar{x}, \bar{x}^*) = 0 \Rightarrow \bigwedge_{l=1}^{k} p_l(f_i(\bar{x}), \bar{x}^*)$$

□

3.4 Polynomial Equations Are Disjunctively Closed

Given two formulas R, S such that R as well as S is a conjunction of polynomial equations, the values that satisfy $R \vee S$ satisfy either R or S, and so are the union of the sets of zeroes of $\phi(R)$ and $\phi(S)$. But by duality the union of the varieties of ideals is the variety of their intersection. We have the following result:

Lemma 3. $\mathcal{P}$ *is* disjunctively closed.

Proof. Given $R, S \in \mathcal{P}$ we define $R \sqcup S$ as the formula obtained after equating to 0 a finite basis of $\phi(R) \cap \phi(S)$, $\psi(\phi(R) \cap \phi(S))$. Then $\mathbb{V}(\phi(R)) \cup \mathbb{V}(\phi(S)) = \mathbb{V}(\phi(R) \cap \phi(S)) = \mathbb{V}(\phi(R \sqcup S))$ implies that

$$R(\bar{x}, \bar{x}^*) \vee S(\bar{x}, \bar{x}^*) \Leftrightarrow (R \sqcup S)(\bar{x}, \bar{x}^*)$$

Since we have equivalence, $R \sqcup S$ thus defined is enough.
□

3.5 Existential Quantifier Elimination

In this section we show that in theory we can eliminate the quantifiers and the infinite disjunction in the following formula

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \Big)$$

using operations on polynomial ideals. Namely, for $i : 1 \leq i \leq n$ and R we can compute a basis B of the ideal

$$\mathbb{Q}[\bar{x}, \bar{x}^*] \bigcap \Big(\bigcap_{s=1}^{\infty} \Big\langle (-\bar{x} + f_i^s(\bar{y})) \bigcup \Big(\bigcup_{p \in \mathbb{IV}(\phi(R))} p(\bar{y}, \bar{x}^*) \Big) \Big\rangle \Big)$$

where $-\bar{x} + f_i^s(\bar{y})$ denotes a set of m polynomials, one for each of the m components. Then we can define $\varphi_i(R)(\bar{x}, \bar{x}^*) := (\bigwedge_{p \in B}\ p(\bar{x}, \bar{x}^*) = 0)$.

In Appendix B we show that the φ_i's defined this way satisfy all the required conditions. Unfortunately, the proof is not constructive. How to find such formulas (or how to approximate them) is another issue which will be dealt with later on in Section 4.

3.6 Procedure for Conjunctions of Polynomial Equalities

As we have seen in the previous sections, $\mathcal{P}$ satisfies all conditions to apply the abstract framework. Our goal now is to translate the General Procedure, which manipulates formulas, into a procedure manipulating ideals, because operations on the latter are easier to implement. In order to do that we will use the correspondences ϕ and ψ, defined in Section 3.2. Notice that $\forall R \in \mathcal{P}$, $\forall \bar{\alpha}, \bar{\alpha}^* \in \mathbb{Q}^m$ we have that

$$R(\bar{\alpha}^*, \bar{\alpha}^*) \Leftrightarrow (\bar{\alpha}, \bar{\alpha}^*) \in \mathbb{V}(\phi(R)) \Leftrightarrow$$

$$\Leftrightarrow (\bar{\alpha}, \bar{\alpha}^*) \in \mathbb{V}(basis(\phi(R)) \Leftrightarrow \psi \circ \phi(R)(\bar{\alpha}, \bar{\alpha}^*)$$

So if we apply ϕ to the formula variables R and R' in the General Procedure in order to compute with ideals and apply ψ to recover the output formula on termination, we still get the strongest polynomial invariant since equivalence is preserved.

Furthermore, it is not difficult to see that $\phi(R \wedge S) = \langle \phi(R) \cup \phi(S) \rangle$. Therefore the initial assignment

$$R := \Big(\bigwedge_{j=1}^{m} (x_j = x_j^*) \Big) \wedge R_0$$

is translated into

$$\phi(R) := \left\langle \Big(\bigcup_{j=1}^{m} \{x_j - x_j^*\} \Big) \cup \phi(R_0) \right\rangle = \langle \{\bar{x} - \bar{x}^*\} \cup \phi(R_0) \rangle$$

where $\phi(R_0)$ is an ideal of polynomials satisfied by the initial values, and $\bar{x} - \bar{x}^*$ denotes the m polynomials that correspond to the projections over each of the m coordinates.

It is also easy to prove that $\forall R, S \in \mathcal{P}$,

$$\Big(R(\bar{x}, \bar{x}^*) \Leftrightarrow S(\bar{x}, \bar{x}^*) \Big) \Leftrightarrow \mathbb{IV}(\phi(R)) = \mathbb{IV}(\phi(S))$$

Thus the test **while** $R' \not\Leftrightarrow R$ **do** in the General Procedure becomes **while** $\mathbb{IV}(\phi(R')) \neq \mathbb{IV}(\phi(R))$ **do** in the new one.

Before finally showing the procedure in terms of ideals we need to introduce new notation in order to represent

$$\bigcup_{p \in \mathbb{IV}(\phi(R))} p(\bar{y}, \bar{x}^*)$$

appearing in

$$\mathbb{Q}[\bar{x}, \bar{x}^*] \bigcap \Big(\bigcap_{s=1}^{\infty} \Big\langle (-\bar{x} + f_i^s(\bar{y})) \bigcup \Big(\bigcup_{p \in \mathbb{IV}(\phi(R))} p(\bar{y}, \bar{x}^*) \Big) \Big\rangle \Big)$$

Given an ideal $I \in \mathbb{Q}[\bar{x}, \bar{x}^*]$ and a polynomial mapping $g \in \mathbb{Q}[\bar{z}, \bar{x}]^m$, where the $\bar{z}$ are auxiliary variables (for example, $\bar{y}$) we define the ideal in $\mathbb{Q}[\bar{x}, \bar{x}^*, \bar{z}]$

$$subs(g, I) = \langle p(g(\bar{z}, \bar{x}), \bar{x}^*) \in \mathbb{Q}[\bar{x}, \bar{x}^*, \bar{z}] \mid p(\bar{x}, \bar{x}^*) \in I \rangle_{\mathbb{Q}[\bar{x}, \bar{x}^*, \bar{z}]}$$

that is to say the result of composing the $\bar{x}$ variables with g. In our case we have that

$$\Big\langle \bigcup_{p \in \mathbb{IV}(\phi(R))} p(\bar{y}, \bar{x}^*) \Big\rangle = subs(\bar{y}, \mathbb{IV}(\phi(R))$$

Then, taking into account the definition of $\sqcup$ and the φ_i's in the proofs of expressiveness and quantifier elimination and the remarks above, the procedure is

Input.

The polynomial mappings $f_1, ..., f_n$ of the assignments

An ideal I_0 of polynomials satisfied by the initial values

Output.

The strongest $\mathcal{P}$-invariant formula R_∞ with respect to $\psi(I_0)$

var I, I' : ideals in $\mathbb{Q}[\bar{x}, \bar{x}^*]$ **end var**

$I' := \mathbb{Q}[\bar{x}, \bar{x}^*]$
$I := \langle \{\bar{x} - \bar{x}^*\} \cup I_0 \rangle$
while $\mathbb{IV}(I') \neq \mathbb{IV}(I)$ **do**
  $I' := I$
  $I := I \cap \mathbb{Q}[\bar{x}, \bar{x}^*] \cap \left(\bigcap_{i=1}^{n} \bigcap_{s=1}^{\infty} \left\langle \{-\bar{x} + f_i^s(\bar{y})\} \bigcup subs(\bar{y}, \mathbb{IV}(I)) \right\rangle \right)$
end while
return $\psi(I)$

where the relation with the General Procedure is given by $I = \phi(R)$, $I' = \phi(R')$, $I_0 = \phi(R_0)$.

3.7 Eliminating Existential Quantifiers: Invertible Polynomial Mappings

Given a polynomial mapping g, we say that g is an invertible polynomial mapping if $\exists g^{-1}$ polynomial mapping. Now assume that $\forall i : 1 \leq i \leq n$ f_i is an invertible polynomial mapping. Then $\bar{x} = f_i^s(\bar{y})$ implies $\bar{y} = f_i^{-s}(\bar{x})$, and therefore we can eliminate $\bar{y}$ and replace the assignment

$$I := I \cap \mathbb{Q}[\bar{x}, \bar{x}^*] \cap \Big(\bigcap_{i=1}^{n} \bigcap_{s=1}^{\infty} \Big\langle \{-\bar{x} + f_i^s(\bar{y})\} \bigcup subs(\bar{y}, \mathbb{IV}(I)) \Big\rangle \Big)$$

by

$$I := I \cap \Big(\bigcap_{i=1}^{n} \bigcap_{s=1}^{\infty} subs(f_i^{-s}, \mathbb{IV}(I)) \Big)$$

Moreover, we can get rid of the $\mathbb{IV}(\cdot)$ operator in the procedure under certain hypotheses on the ideal I_0. Namely, if I_0 is such that $I_0 = \mathbb{IV}(I_0)$ then it can be proved that $I = \mathbb{IV}(I)$ is an invariant of the procedure for finding polynomial invariants, and therefore we can replace $\mathbb{IV}(I)$ by I under this hypothesis. The proof (see [CK]) is based on three facts:

- If $I_0 \in \mathbb{Q}[\bar{x}^*]$ is an ideal such that $I_0 = \mathbb{IV}(I_0)$, then

$$\langle \{\bar{x} - \bar{x}^*\} \cup I_0 \rangle = \mathbb{IV}(\langle \{\bar{x} - \bar{x}^*\} \cup I_0 \rangle)$$

- If I is an ideal such that $I = \mathbb{IV}(I)$ and g is an invertible polynomial mapping, then

$$\bigcap_{s=0}^{\infty} subs(g^{-s}, I) = \mathbb{IV}(\bigcap_{s=0}^{\infty} subs(g^{-s}, I))$$

- If I, J are ideals such that $I = \mathbb{IV}(I)$ and $J = \mathbb{IV}(J)$ then $I \cap J = \mathbb{IV}(I \cap J)$

The first result proves that the equality $I = \mathbb{IV}(I)$ holds at the beginning of the procedure, and the other two guarantee that at each step $I = \mathbb{IV}(I)$ is preserved.

So assuming that $I_0 = \mathbb{IV}(I_0)$ we can further simplify the procedure, which finally becomes

Input.
The invertible polynomial mappings $f_1, ..., f_n$ of the assignments
An ideal I_0 of polynomials satisfied by the initial values such that $I_0 = \mathbb{IV}(I_0)$
Output.
The strongest $\mathcal{P}$-invariant formula R_∞ with respect to $\psi(I_0)$

var I, I' : ideals in $\mathbb{Q}[\bar{x}, \bar{x}^*]$ **end var**

$I' := \mathbb{Q}[\bar{x}, \bar{x}^*]$
$I := \langle \{\bar{x} - \bar{x}^*\} \cup I_0 \rangle$
while $I' \neq I$ **do**
$I' := I$
$I := \bigcap_{i=1}^{n} \bigcap_{s=0}^{\infty} subs(f_i^{-s}, I)$
end while
return $\psi(I)$

Notice that the assignment $I := I \cap (\bigcap_{i=1}^{n} \bigcap_{s=1}^{\infty} subs(f_i^{-s}, I))$ is equivalent to $I := \bigcap_{i=1}^{n} \bigcap_{s=0}^{\infty} subs(f_i^{-s}, I)$. We will refer to this as the Polynomial Procedure.

3.8 Eliminating Existential Quantifiers: Solvable Mappings

Unfortunately, there exists no general formula for the powers f_i^{-s} when f_i is any invertible polynomial mapping. We have to restrict the set of treatable assignments to those for which we can compute the powers of their right-hand sides f_i. For this reason we introduce the concept of *solvable* mapping. Intuitively, a solvable mapping g is a polynomial mapping such that the recurrence $x_{s+1} = g(x_s)$ can be solved effectively and such that its solution (which is the general power g^s) has a polynomial structure.

Before giving the formal definition we need some notation. Given $U \subseteq \bar{x}$ a subset of the variables we denote by $g_U : \mathbb{Q}^m \to \mathbb{Q}^{|U|}$ the mapping consisting in the (sorted) tuple of the j-th components of g such that $x_j \in U$. For instance, for the mapping

$$g(a, b, p, q) = (a - 1, b, p, q + bp)$$

we would have

$$g_{\{a\}}(a, b, p, q) = g_{\{a\}}(a) = a - 1$$

$$g_{\{a,b,p\}}(a, b, p, q) = g_{\{a,b,p\}}(a, b, p) = (a - 1, b, p)$$

$$g_{\{q\}}(a, b, p, q) = q + bp$$

Definition 5. *Let $g \in \mathbb{Q}[\bar{x}]^m$ be a polynomial mapping. We say that g is solvable if there exists a partition of $\bar{x}$, $\bar{x} = U_1 \cup \cdots \cup U_k$, $U_i \cap U_j = \emptyset$ if $i \neq j$, such that $\forall j : 1 \leq j \leq k$ we have*

$$g_{U_j}(\bar{x}) = M_j U_j^T + P_j(U_1, ..., U_{j-1})$$

where $M_j \in \mathbb{Q}^{|U_j| \times |U_j|}$ is a matrix and P_j is a vector of $|U_j|$ polynomials with coefficients in $\mathbb{Q}$ and depending on the variables in $U_1, ..., U_{j-1}$ (P_1 must necessarily be a constant vector).

Moreover, we define the eigenvalues of g as the union of the eigenvalues of the matrices M_j, $1 \leq j \leq k$.

Finally, an assignment $\bar{x} := g(\bar{x})$ is said to be solvable *if the mapping g is solvable.*

Notice that any affine mapping $g(\bar{x}) = A\bar{x} + b$ is solvable, since we can take $U_1 = \bar{x}$, $M_1 = A$, $P_1 = b$, and then the eigenvalues of g are the eigenvalues of A. For example, the affine mappings

$$f_1(x, y, z) = (2x, y/2 - 1/2, x + z)$$

$$f_2(x, y, z) = (2x, y/2, z)$$

are solvable and in both cases the eigenvalues are $\{2, 1/2, 1\}$.

Let us consider an example of solvable mapping which is not affine. For instance the non-linear mapping

$$g(a, b, p, q) = (a - 1, b, p, q + bp)$$

is solvable. Indeed, we can take $U_1 = \{a, b, p\}$, $M_1 = diagonal(1, 1, 1)$, $P_1 = (-1, 0, 0)$, since

$$g_{\{a,b,p\}}(a, b, p, q) = g_{\{a,b,p\}}(a, b, p) = (a - 1, b, p);$$

and then $U_2 = \{q\}$, with $M_2 = (1)$ and $P_2 = bp$, as $g_{\{q\}}(a, b, p, q) = q + bp$. Moreover, in this case the eigenvalues of g are just $\{1\}$.

In order to motivate the name, let us compute g^s. Equivalently, we can solve the recurrence equation $(a_{s+1}, b_{s+1}, p_{s+1}, q_{s+1}) = g(a_s, b_s, p_s, q_s)$, whose solution is $g^s(a_0, b_0, p_0, q_0)$. We first solve the recurrence for a_s, b_s, p_s:

$$\begin{cases} a_{s+1} &= a_s - 1 \\ b_{s+1} &= b_s \\ p_{s+1} &= p_s \end{cases}$$

It is easy to see that

$$\begin{cases} a_s &= a_0 - s \\ b_s &= b_0 \\ p_s &= p_0 \end{cases}$$

Now as $q_{s+1} = q_s + b_s p_s$, plugging in the solution for the variables that have already been solved we get the recurrence

$$q_{s+1} = q_s + b_0 p_0$$

This equation can be easily solved, and finally we get that

$$q_s = q_0 + b_0 p_0 s$$

and

$$g^s(a, b, p, q) = \Big(a - s, b, p, q + bps\Big)$$

Notice that $g^s(\bar{x})$ is a vector of polynomials; this "polynomial structure" will be very important when eliminating the loop counter s in the implementation in Section 4.

Now, consider the following program, that computes the product of two integer numbers:

function product (A, B: integer) **returns** q: integer
{ Pre: $A \geq 0 \wedge B \geq 0$}
var a, b, p: integer **end var**
$\langle a,\ b,\ p,\ q\rangle$:=$\langle A,\ B,\ 1,\ 0\rangle$;
while $(a \neq 0) \wedge (b \neq 0)$ **do**
if $(a \texttt{ mod } 2 = 0) \wedge (b \texttt{ mod } 2 = 0)$
$\rightarrow \langle a, b, p, q\rangle := \langle a/2, b/2, 4p, q\rangle$;
[]$(a \texttt{ mod } 2 = 1) \wedge (b \texttt{ mod } 2 = 0)$
$\rightarrow \langle a, b, p, q\rangle := \langle a - 1, b, p, q + bp\rangle$;
[]$(a \texttt{ mod } 2 = 0) \wedge (b \texttt{ mod } 2 = 1)$
$\rightarrow \langle a, b, p, q\rangle := \langle a, b - 1, p, q + ap\rangle$;
[]$(a \texttt{ mod } 2 = 1) \wedge (b \texttt{ mod } 2 = 1)$
$\rightarrow \langle a, b, p, q\rangle := \langle a - 1, b - 1, p, q + (a + b - 1)p\rangle$;
end if
end while
{ Post: $q = A \cdot B$}

By applying the same construction as above, it is easy to see that all the assignments in this program are solvable.

3.9 Termination of the Procedure

The Polynomial Procedure is shown to be terminating under two cases:

- if the assignment mappings f_i's are solvable and the *associated eigenvalues are positive*. In that case, it can be shown that the procedure terminates in at most $2m+1$ iterations, where m is the number of variables changing value in the body of the loop.
- if the assignment mappings f_i's are polynomial and *commute*, i.e. $f_i \circ f_j = f_j \circ f_i$ for $1 \leq i, j \leq n$. In that case it is shown that the procedure terminates in at most $n + 1$ iterations, where n is the number of branches in the body of the loop.

Solvable Mappings with Positive Eigenvalues In [CK], it is proved using algebraic geometry arguments that for the case when the assignments f_i's are invertible solvable mappings and furthemore, the associated eigenvalues are positive, then the above procedure terminates in at most $2m+1$ steps, where m is the number of changing variables in the loop. Every iteration of the procedure is shown to increase at least by 1 the dimension of all those irreducible components of the variety of the computed ideal which are not invariant for the loop yet. In practice the dimension of the variety itself usually increases too, as we will see with some examples.

The formal statement of termination is the following:

Theorem 1. *If the mappings f_i are solvable and $\bigcup_{i=1}^{n}$ eigenvalues$(f_i) \subseteq \mathbb{R}^+$, then the Polynomial Procedure terminates in at most $2m+1$ iterations.*

The proof is given in [CK].

Commuting Polynomial Mappings We first prove a more general result, namely, that in the N-th iteration of the Polynomial Procedure, the effect of all possible compositions of assignment mappings of length $\leq N$ has been considered. Using this general result we show that if the assignment mappings commute, then a fixed point is reached in $n+1$ iterations, where n is the number of branches in the conditional statement of the body of the loop. In particular if $n=1$, i.e. there are no conditional statements, the procedure takes at most 2 iterations to generate the fixed point.

Let us consider the set of strings over the alphabet $[n] = \{1, ..., n\}$, which we denote by $[n]^*$. We also write $f = \{f_1, ..., f_n\}$. For every string $\sigma \in [n]^*$ we inductively define $[f]^\sigma$ as

$$[f]^\lambda(\bar{x}) = \bar{x}, \quad [f]^{\sigma.i}(\bar{x}) = f_i([f]^\sigma(\bar{x}))$$

where λ denotes the empty string. Moreover, given $\sigma \in [n]^*$ we also define $\nu(\sigma)$, the number of alternations of σ as:

- $\nu(\lambda) = -1$ (λ is the empty string)
- $\nu(i) = 0$
- $\nu(i.j.\sigma) = \nu(j.\sigma)$ if $i = j$
- $\nu(i.j.\sigma) = 1 + \nu(j.\sigma)$ if $i \neq j$

$(1 \leq i, j \leq n)$

Finally, for $N \in \mathbb{N}$ we also define $\Im_N$ as the ideal stored in the variable I in the Polynomial Procedure at the end of the N-th iteration. Then we have the following result:

Lemma 4. $\forall N \in \mathbb{N}$

$$\Im_N = \bigcap_{\substack{\sigma \in [n]^* \\ \nu(\sigma) \leq N-1}} subs(([f]^\sigma)^{-1}, \Im_0)$$

Proof. Let us prove it by induction on N. If $N = 0$, then the only $\sigma \in [n]^*$ such that $\nu(\sigma) \leq -1$ is $\sigma = \lambda$, the empty string. Since $subs(([f]^\lambda)^{-1}, \Im_0) = \Im_0$, our claim is true.

Now let us assume that $N > 0$. By definition of $\Im_N$ we have that

$$\Im_N = \bigcap_{i=1}^{n} \bigcap_{s=0}^{\infty} subs(f_i^{-s}, \Im_{N-1})$$

Applying the induction hypothesis,

$$\Im_N = \bigcap_{i=1}^{n} \bigcap_{s=0}^{\infty} subs(f_i^{-s}, \bigcap_{\substack{\sigma \in [n]^* \\ \nu(\sigma) \leq N-2}} subs(([f]^\sigma)^{-1}, \Im_0))$$

As the mappings f_i^{-s} are invertible, it can be proved that *subs* distributes with respect to $\cap$. Then

$$\Im_N = \bigcap_{i=1}^{n} \bigcap_{s=0}^{\infty} \bigcap_{\substack{\sigma \in [n]^* \\ \nu(\sigma) \leq N-2}} subs(f_i^{-s}, subs(([f]^\sigma)^{-1}, \Im_0))$$

Given $f, g \in \mathbb{Q}[\bar{x}]^m$ and an ideal $I \subseteq \mathbb{Q}[\bar{x}]$, $subs(f, subs(g, I)) = subs(g \circ f, I)$. So

$$\Im_N = \bigcap_{i=1}^{n} \bigcap_{s=0}^{\infty} \bigcap_{\substack{\sigma \in [n]^* \\ \nu(\sigma) \leq N-2}} subs(([f]^\sigma)^{-1} \circ (f_i^s)^{-1}, \Im_0) =$$

$$= \bigcap_{i=1}^{n} \bigcap_{s=0}^{\infty} \bigcap_{\substack{\sigma \in [n]^* \\ \nu(\sigma) \leq N-2}} subs((f_i^s \circ [f]^\sigma)^{-1}, \Im_0) =$$

$$= \bigcap_{i=1}^{n} \bigcap_{s=0}^{\infty} \bigcap_{\substack{\sigma \in [n]^* \\ \nu(\sigma) \leq N-2}} subs(([f]^{\sigma.\overbrace{i.\cdots.i}^{s}})^{-1}, \Im_0) =$$

$$= \bigcap_{\sigma \in [n]^*, \nu(\sigma) \leq N-1} subs(([f]^\sigma)^{-1}, \Im_0)$$

which is what we wanted to show.
□

Finally, we have the result of termination if the mappings f_i commute:

Theorem 2. *If the mappings f_i commute, i.e. $f_i \circ f_j = f_j \circ f_i \; \forall i, j : 1 \leq i, j \leq n$, then the procedure terminates in at most $n + 1$ iterations.*

Proof. If the f_i's commute then $\forall \sigma \in [n]^*$ we can build, by rearranging the mappings f_i and collapsing them in a single power, $\tau \in [n]^*$ such that $\nu(\tau) \leq n-1$ and $[f]^\sigma = [f]^\tau$. Then, by Lemma 4

$$\Im_n = \bigcap_{\substack{\sigma \in [n]^* \\ \nu(\sigma) \leq n-1}} \Big(subs(([f]^\sigma)^{-1}, \Im_0) \Big) =$$

$$= \bigcap_{\substack{\sigma \in [n]^* \\ \nu(\sigma) \leq n}} \Big(subs(([f]^\sigma)^{-1}, \Im_0) \Big) = \Im_{n+1}$$

Therefore the procedure terminates in at most $n+1$ iterations.
□

4 Implementation of the Polynomial Procedure

4.1 Approximating the Infinite Intersection: Powers of Solvable Mappings

For the sake of a clear presentation we have not shown yet the procedure that has actually been implemented. The problem with the Polynomial Procedure as given in Section 3 is that the assignment

$$I := \bigcap_{s=0}^{\infty} \bigcap_{i=1}^{n} subs(f_i^{-s}, I)$$

is not directly implementable, due to the infinite intersection. In this section we show how to approximate it by means of algebraic geometry methods without losing completeness.

The basic idea is to consider the parameter s as a new variable s and compute the general expression of the powers f_i^{-s} for $1 \leq i \leq n$; for that reason we need the mappings f_i to be invertible and solvable. For the time being let us assume that $f_i^{-s}(\bar{x}) \in \mathbb{Q}[s, \bar{x}]$. Then given a basis for $I \subseteq \mathbb{Q}[\bar{x}, \bar{x}^*]$ we get a basis for $subs(f_i^{-s}, I) \subseteq \mathbb{Q}[s, \bar{x}, \bar{x}^*]$ by substituting the $\bar{x}$ variables by $f_i^{-s}(\bar{x})$. By using standard Gröbner bases techniques, the finite intersection $\cap_{i=1}^n subs(f_i^{-s}, I)$ can be computed. What remains to be done is the infinite intersection. The approximation consists in taking an elimination monomial order for s and then eliminate this auxiliary variable from $\cap_{i=1}^n subs(f_i^{-s}, I)$.

Nevertheless, there is another problem with this approach. The hypothesis $f_i^{-s}(\bar{x}) \in \mathbb{Q}[s, \bar{x}]$ does not necessarily hold in general; exponential terms might appear, like in the example from Section 2.1:

$\langle x,\ y,\ z\rangle := \langle X,\ Y,\ 0\rangle$;
while *true* **do**
  if $true \rightarrow \langle x,\ y,\ z\rangle := \langle 2 * x,\ (y-1)/2,\ x + z\rangle$;
  [] $true \rightarrow \langle x,\ y,\ z\rangle := \langle 2 * x,\ y/2,\ z\rangle$;

end if
end while

$$f_1(x,y,z) = (2x, y/2 - 1/2, x + z)$$
$$f_2(x,y,z) = (2x, y/2, z)$$
$$f_1^s(x,y,z) = (2^s x, (1/2)^s y + (1/2)^s - 1, z + (2^s - 1)x)$$
$$f_2^s(x,y,z) = (2^s x, (1/2)^s y, z)$$

The eigenvalues of the f_i's in this case are $\{1/2, 2, 1\}$.

In general we have the following result concerning the powers of solvable mappings:

Theorem 3. *Let $g \in \mathbb{Q}[\bar{x}]^m$ be a solvable mapping with rational eigenvalues. Then for $1 \leq j \leq m$ $g_j^s(\bar{x})$, the j-th component of $g^s(\bar{x})$, can be expressed as*

$$g_j^s(\bar{x}) = \sum_{l=1}^{k_j} P_{jl}(s, \bar{x})(\gamma_{jl})^s, \ 1 \leq j \leq m, \ s \geq 0$$

where for $1 \leq j \leq m$, $1 \leq l \leq k_j$, $P_{jl} \in \mathbb{Q}[s, \bar{x}]$ and $\gamma_{jl} \in \mathbb{Q}$. Moreover, the γ_{jl} are products of the eigenvalues of g.

The proof (see Appendix C) is based on the fact that a matrix $M \in \mathbb{Q}^{r \times r}$ with rational eigenvalues can be decomposed as $M = S^{-1}JS$, with $S, J \in \mathbb{Q}^{r \times r}$, $\det(S) \neq 0$ and J the Jordan normal form of M ([Nom66]); and that a sequence $(w_s)_{s \in \mathbb{N}}$ is of the form

$$w_s = \sum_{j=1}^{k} P_j(s)\lambda_j^s, \qquad s \geq 0$$

with P_j polynomial for $1 \leq j \leq k$ if and only if its generating function $W(z) = \sum_{s=0}^{\infty} w_s z^s$ is a rational function (see [Sta97] for an introduction to generating functions).

A way to sort this problem out is to introduce more auxiliary variables to replace the exponential terms γ_{jl}^s and then eliminate them with a suitable elimination order. From now on let us assume that $\bigcup_{i=1}^n \text{eigenvalues}(f_i) \subseteq \mathbb{Q}^+$ (notice that by Theorem 1 termination is guaranteed in this case). It is clear that $\forall \gamma \in \mathbb{Q}^+$ there exists a unique prime decomposition of the form $\gamma = \prod_{i=1}^k \lambda_i^{\alpha_i}$ where the λ_i are primes for $1 \leq i \leq k$ and $\alpha_i \in \mathbb{Z}$. So we can compute a "base" $\Lambda = \{\lambda_1, ..., \lambda_k\} \subset \mathbb{N}$ of prime numbers such that $\forall \gamma \in \bigcup_{i=1}^n \text{eigenvalues}(f_i)$ we have $\gamma = \prod_{i=1}^k \lambda_i^{\alpha_k}$ for certain $\alpha_i \in \mathbb{Z}$. Then

$$\gamma^s = \prod_{i=1}^{k} (\lambda_i^{\alpha_k})^s = \prod_{i=1}^{k} (\lambda_i^{s \cdot \Sigma(\alpha_k)})^{|\alpha_k|}$$

where $\Sigma(\cdot)$ is the sign function.

Now we can introduce the variables u_i to replace λ_i^s and the variables v_i to replace λ_i^{-s}. By Theorem 3 for $1 \leq i \leq n$ there exists a polynomial mapping $F_i = F_i(s, \bar{u}, \bar{v}, \bar{x}) : \mathbb{Q}^{1+2k+m} \to \mathbb{Q}^m$ such that $\forall t \in \mathbb{N}$,

$$f_i^t(\bar{x}) = F_i(t, \bar{\lambda}^t, \bar{\lambda}^{-t}, \bar{x})$$

and

$$f_i^{-t}(\bar{x}) = F_i(-t, \bar{\lambda}^{-t}, \bar{\lambda}^t, \bar{x})$$

where $\bar{\lambda}^t = (\lambda_1^t, ..., \lambda_k^t)$.

For instance, in our previous example we can take $\Lambda = \{2\}$ and then we have:

$$f_1(x, y, z) = (2x, y/2 - 1/2, x + z)$$

$$f_2(x, y, z) = (2x, y/2, z)$$

$$f_1^s(x, y, z) = (2^s x, (1/2)^s y + (1/2)^s - 1, z + (2^s - 1)x)$$

$$f_2^s(x, y, z) = (2^s x, (1/2)^s y, z)$$

$$F_1(s, u, v, x, y, z) = (ux, vy + v - 1, z + (u - 1)x)$$

$$F_2(s, u, v, x, y, z) = (ux, vy, z)$$

where the variables u, v represent 2^s and $(1/2)^s$ respectively.

So far we have not considered the possible polynomial relations between t, the λ_i^t and the λ_i^{-t}, which may be important in order to eliminate the auxiliary variables. Let $\mathcal{L} = \{l \in \mathbb{Q}[s, \bar{u}, \bar{v}] \mid l(t, \bar{\lambda}^t, \bar{\lambda}^{-t}) = 0 \ \forall t \in \mathbb{N}\}$ be this set of polynomials. Let us also define $L = \{u_1 v_1 - 1, ..., u_k v_k - 1\}$. Then it can be shown that $\langle L \rangle_{\mathbb{Q}[s,\bar{u},\bar{v}]} = \mathcal{L}$ (see Appendix D for the proof), and therefore we have a precise characterization of $\mathcal{L}$.

For instance, since in the example we have $\Lambda = \{2\}$ and the variables u, v represent 2^s and $(1/2)^s$ respectively, in this case $L = \{uv - 1\}$.

4.2 Implementation Using Gröbner bases

In the implementation of the Polynomial Procedure the operations between ideals have been replaced by operations between sets of polynomials:

Input.

The solvable mappings with positive rational eigenvalues $f_1, ..., f_n$ of the assignments

A set S_0 of polynomials satisfied by the initial values such that $\langle S_0 \rangle = \mathbb{IV}(\langle S_0 \rangle)$

Output.

The strongest $\mathcal{P}$-invariant formula R_∞ with respect to $\left(\bigwedge_{p \in S_0} p = 0\right)$

var

S', S : sets of polynomials in $\mathbb{Q}[\bar{x}, \bar{x}^*]$

S_{aux} : set of polynomials in $\mathbb{Q}[s, \bar{u}, \bar{v}, \bar{x}, \bar{x}^*]$

end var

1: compute $f_1^s, ..., f_n^s, F_1, ..., F_n, L$
2: $S' := \{1\}$
3: $S := \{x_1^* - x_1, ..., x_m^* - x_m\} \cup S_0$
4: **while** $S' \neq S$ **do**
5: $S' := S$
6: S_{aux} :=`gröbner_basis`$(\bigcap_{i=1}^{n} \langle$`subs`$(F_i(-s, \bar{v}, \bar{u}, \cdot), S)\rangle_{\mathbb{Q}[s,\bar{u},\bar{v},\bar{x},\bar{x}^*]}, >)$
7: S_{aux} :=`gröbner_basis`$(S_{aux} \cup L, \succ)$
8: $S :=$ { polynomials in S_{aux} without $s, \bar{u}, \bar{v}$}
9: **end while**
10: **return** $\left(\bigwedge_{p \in S} p = 0\right)$

At line 1 the f_i^s's are computed by means of linear algebra or generating functions, using partial fraction decomposition. The command `rsolve` in Maple can be used for this purpose. For computing the F_i's from the f_i^s, the eigenvalues of the $f_i's$ are factorized and auxiliary variables $\bar{u}$ and $\bar{v}$ are introduced to substitute the exponentials. These can be done efficiently using commands supported in most mathematical packages.

The function `subs` in line 6 substitutes polynomials for variables in a given basis; this is the way the function *subs* applied to ideals is implemented. The intersection of ideals at the same line is performed by using Gröbner bases methods. The function `gröbner_basis` computes the unique reduced Gröbner basis of the ideal generated by a finite set of polynomials with respect to a specified monomial order. In this case $>$ is arbitrary

At line 7 the order $\succ$ on monomials for computing a Gröbner basis is chosen so that it eliminates the variables $s, \bar{u}, \bar{v}$ (typically using block-order or lexicographic ordering). At line 8 polynomials not containing any of the auxiliary variables as well as s are selected from the Gröbner basis, so that the result is a set of polynomials S such that

$$\langle S\rangle_{\mathbb{Q}[\bar{x},\bar{x}^*]} = \mathbb{Q}[\bar{x}, \bar{x}^*] \cap \left\langle L \cup \left(\bigcap_{i=1}^{n} subs(F_i(-s, \bar{v}, \bar{u}\cdot), \langle S\rangle)\right)\right\rangle_{\mathbb{Q}[s,\bar{u},\bar{v},\bar{x},\bar{x}^*]}$$

Moreover, S is the reduced Gröbner basis for this ideal with respect to the restriction of the order $\succ$ to the variables $\bar{x}, \bar{x}^*$.

The equality test on ideals is easily implemented by computing their reduced Gröbner basis using the same ordering on monomials since every ideal has a unique reduced Gröbner basis once the ordering is fixed. From the second iteration on, both S and S' are the reduced Gröbner bases of the respective ideals.

The following result ensures that the above procedure is still correct and complete:

Theorem 4. *If the Polynomial Procedure terminates with output I^*, the implementation terminates in at most the same number of iterations and with output S^* such that $\langle S^*\rangle_{\mathbb{Q}[\bar{x},\bar{x}^*]} = I^*$.*

See Appendix E. If we write $P_\infty = \phi(R_\infty)$ as we did in Section 3.3, the proof is based on the fact that $P_\infty \subseteq \langle S \rangle$ is kept invariant during all the execution, and so in particular on termination $P_\infty \subseteq \langle S^* \rangle$; and is based also on the fact that at any iteration the ideal I computed by the Polynomial Procedure includes the ideal $\langle S \rangle$ generated by the polynomials obtained in the implementation. So if the Polynomial Procedure terminates we have $I^* = P_\infty \subseteq \langle S^* \rangle \subseteq I^*$, and therefore all are equalities and the implementation terminates with a set of polynomials generating P_∞.

For the product function example discussed earlier, the algorithm works as follows:

`iteration` $0 \longrightarrow \quad \{z^*, x - x^*, y - y^*, z - z^*\}$
This states that x, y, z start with some unknown values x^*, y^*, z^*, respectively, except that z is initialized to be 0. That is why $z^* = 0$. The dimension of this ideal is 2.

`iteration` $1 \longrightarrow \quad \{z^*, xz - z - zx^*, -x^*y^* + z + xy, yz + zyx^* - zx^*y^* + z\}$
The dimension of this ideal is 3.
`iteration` $2 \longrightarrow \quad \{z^*, -x^*y^* + z + xy\}$
The dimension of this ideal is 4.
`iteration` $3 \longrightarrow \quad \{z^*, -x^*y^* + z + xy\}$
The fixed point is computed.

Thus, in only 3 iterations, the algorithm terminates. The polynomial equation $z^* = 0$ is the equation satisfied by the input. Substituting x^* and y^* in $-x^*y^* + z + xy$ by their initial values X and Y, an invariant $-X \cdot Y + z + x \cdot y = 0$, i.e. $z + x \cdot y = X \cdot Y$, is generated which suffices to prove the postcondition in the original program:

{ Pre: $X \geq 0 \wedge Y \geq 0$}
var x, y: integer **end var**
$\langle x,\ y,\ z \rangle := \langle X,\ Y,\ 0 \rangle$;
while $y \neq 0$ **do**
  if y `mod` $2 = 1$ **then** $\langle x, y, z \rangle := \langle 2 * x, (y - 1)$ `div` $2, x + z \rangle$;
  [] y `mod` $2 = 0$ **then** $\langle x, y, z \rangle := \langle 2 * x, y$ `div` $2, z \rangle$;
  end if
end while
{ Post: $z = X \cdot Y$}

Example 1. Consider the following function, which is a version of the program in [Knu69] that finds a factor of a positive odd number N with only addition and subtraction. Let us assume that it has been partially annotated with some invariants which however are not enough to prove partial correctness:

function fermat (N, R: integer) **returns** x, y: integer
  { Pre: $N \geq 1 \wedge N$ `mod` $2 = 1 \wedge (R + 1)^2 > N \geq R^2$}
  var r: integer **end var**

$\langle r,\ x,\ y\rangle$:=$\langle R^2 - N,\ 2 \cdot R + 1,\ 1\rangle$;
{ Inv: $N \geq 1 \wedge x$ `mod` $2\ = 1 \wedge y$ `mod` $2\ = 1$}
while $r \neq 0$ **do**
if $r > 0 \rightarrow \langle r,\ x,\ y\rangle$:=$\langle r - y,\ x,\ y + 2\rangle$;
[] $r < 0 \rightarrow \langle r,\ x,\ y\rangle$:=$\langle r + x,\ x + 2,\ y\rangle$;
end if
end while
{ Post: $x \neq y \wedge x$ `mod` $2 = y$ `mod` $2 \wedge N$ `mod` $((x - y)$ `div` $2) = 0$}
end function

Since there are no divisions in the algorithm, we may consider the variables as rational numbers and apply the algorithm.

$$f_1(r, x, y) = (r - y, x, y + 2)$$

$$f_2(r, x, y) = (r + x, x + 2, y)$$

$$f_1^s(r, x, y) = (r - sy - (s - 1)s, x, 2s + y)$$

$$f_2^s(r, x, y) = (r + sx + (s - 1)s, 2s + x, y)$$

Therefore we just have to add one new variable s:

$$F_1(s, r, x, y) = f_1^s(r, x, y) = (r - sy - (s - 1)s, x, 2s + y)$$

$$F_2(s, r, x, y) = f_2^s(r, x, y) = (r + sx + (s - 1)s, 2s + x, y)$$

Using $S_0 = \{y - 1\}$, we get the following trace of the computation.

`iteration` $0 \longrightarrow \{y^* - 1, r - r^*, x - x^*, y - y^*\}$
The dimension of this ideal is 2.
`iteration` $1 \longrightarrow \{y^* - 1, xy - yx^* - x + x^*, x^2 - (x^*)^2 - y^2 - 4r + 4r^* - 2x + 2x^* + 2y - 1, y^3 + 4ry - 4yr^* - 3y^2 - 4r + 4r^* + 3y - 1\}$
The dimension of this ideal is 3.
`iteration` $2 \longrightarrow \{y^* - 1, x^2 - (x^*)^2 - y^2 - 4r + 4r^* - 2x + 2x^* + 2y - 1\}$
The dimension of this ideal is 4.
`iteration` $3 \longrightarrow \{y^* - 1, x^2 - (x^*)^2 - y^2 - 4r + 4r^* - 2x + 2x^* + 2y - 1\}$

In this case, the algorithm terminates in 3 iterations as well. Substituting r and x by their initial values in the program $R^2 - N$ and $2R + 1$ respectively, we get that

$$\{-4r - 4N + x^2 - 2x - y^2 + 2y = 0\}$$

which is an invariant. This polynomial equation along with other nonpolynomial formulas used as annotations in the program is sufficient to prove the correctness of the program, using that $x^2 - 2x - y^2 + 2y = (x - y)(x + y - 2)$.

4.3 Further Improvements

Gröbner bases are well known for their bad behaviour as far as complexity is concerned (see [Huy86] for example). So an intelligent use of Gröbner bases computations may lead to a drastic reduction in time. A possible improvement when there are two or more branches is as follows.

Looking for polynomial invariants for all the branches together from the very beginning requires computing a lot of intersections of ideals at the same time. In order to avoid this, we can first find invariants for one branch; then find invariants for two branches, the previous and another one; and so on, until considering all possible branches. It is clear that the properties of correctness and completeness of the method are preserved. The following procedure is the formal description of what has been discussed above:

Input.

The solvable mappings with positive rational eigenvalues f_1,...,f_n of the assignments

A set S_0 of polynomials satisfied by the initial values such that $\langle S_0 \rangle = \mathbb{IV}(\langle S_0 \rangle)$

Output.

A set of polynomials S that generates the strongest $\mathcal{P}$-invariant ideal with respect to $\left(\bigwedge_{p \in S_0} p = 0\right)$

var
S', S : sets of polynomials in $\mathbb{Q}[\bar{x}^*, \bar{x}]$
S_{aux} : set of polynomials in $\mathbb{Q}[s, \bar{u}, \bar{v}, \bar{x}^*, \bar{x}]$
k : integer
end var

$S := \{x_1^* - x_1, ..., x_m^* - x_m\} \cup S_0$
for k **from** 1 **to** n **do**
compute F_k
$S' := \{1\}$
while $S' \neq S$ **do**
$S' := S$
$S_{aux} :=$ `gröbner_basis`$(\bigcap_{i=1}^{k} \langle$ `subs`$(F_i(-s, \bar{v}, \bar{u}, \cdot), S)\rangle_{\mathbb{Q}[s,\bar{u},\bar{v},\bar{x}^*,\bar{x}]}, >)$
$S_{aux} :=$ `gröbner_basis`$(S_{aux} \cup L, \succ)$
$S :=$ { polynomials in S_{aux} without $s, \bar{u}, \bar{v}$}
end while
end for
return S

Although it requires more iterations to terminate, in practice this algorithm is preferable because Gröbner bases computations are performed using less vari-

ables and less polynomials, and therefore more quickly than in the previous version.

5 Examples

Below we show some of the programs whose polynomial invariants have been successfully computed using the implementation described in Section 4.3.

The first of the programs has been introduced in Section 3.8

function product (A, B: integer) **returns** q: integer
{ Pre: $A \geq 0 \wedge B \geq 0$}
var a, b, p: integer **end var**
$\langle a,\ b,\ p,\ q\rangle$:=$\langle A,\ B,\ 1,\ 0\rangle$;
while $(a \neq 0) \wedge (b \neq 0)$ **do**
if $(a \texttt{ mod } 2 = 0) \wedge (b \texttt{ mod } 2 = 0)$
$\rightarrow \langle a, b, p, q\rangle := \langle a/2, b/2, 4 \cdot p, q\rangle$;
[]$(a \texttt{ mod } 2 = 1) \wedge (b \texttt{ mod } 2 = 0)$
$\rightarrow \langle a, b, p, q\rangle := \langle a - 1, b, p, q + b \cdot p\rangle$;
[]$(a \texttt{ mod } 2 = 0) \wedge (b \texttt{ mod } 2 = 1)$
$\rightarrow \langle a, b, p, q\rangle := \langle a, b - 1, p, q + a \cdot p\rangle$;
[]$(a \texttt{ mod } 2 = 1) \wedge (b \texttt{ mod } 2 = 1)$
$\rightarrow \langle a, b, p, q\rangle := \langle a - 1, b - 1, p, q + (a + b - 1) \cdot p\rangle$;
end if
end while
{ Post: $q = A \cdot B$}

In this case, $\{q + abp = AB\}$ is automatically generated as an invariant in 7 iterations.

The second program has been extracted from [CC77]:

var i, j: integer **end var**
$\langle i, j\rangle$:=$\langle 2, 0\rangle$;
while $true$ **do**
if $true \rightarrow$
$\langle i, j\rangle$:=$\langle i + 4, j\rangle$;
[] $true \rightarrow$
$\langle i, j\rangle$:=$\langle i + 2, j + 1\rangle$;
end if
end while

For this loop no invariant is generated. Since our technique is complete, i.e. if there is a polynomial equation which is invariant in the loop, the algorithm will find the strongest possible conjunction of invariant polynomial equations, it can be asserted that there are *no* invariant polynomial equations in the above loop. This is consistent with the results obtained by Cousot and Halbwachs, who do not find linear invariant equalities for this example.

The third example computes the floor of the square root of a natural number. It can be easily shown that the polynomial invariants automatically discovered by our procedure along with the partial annotations suffice to prove the correctness of the program.

function sqrt (N: integer) **returns** a: integer
  { Pre: $N \geq 0$}
  var s, t: integer **end var**
  $\langle a,\ s,\ t\rangle$:=$\langle 0,\ 1,\ 1\rangle$;
  { Inv: $s - t \leq N$}
  while $s \leq N$ **do**
    $\langle a,\ s,\ t\rangle$:=$\langle a+1,\ s+t+2,\ t+2\rangle$;
  end while
  { Post: $a^2 \leq N < (a+1)^2$}
end function

In this case, the conjunction $\{t = 2a + 1 \ \wedge \ s = (a+1)^2\}$ is automatically generated as an invariant in 2 iterations.

The fourth example, which has been extracted from [Dij76], also computes the floor of the square root:

function sqrt (N: integer) **returns** p: integer
  { Pre: $N \geq 0$}
  var q, r, h: integer **end var**

  p:=0; q:=1; r:=N;
  { Inv: $N \geq 0 \wedge \exists k\ (k \geq 0 \wedge q = 4^k)$}
  while $q \leq N$ **do**
    q:=$4 * q$;
  end while

  { Inv: $r \geq 0 \wedge r < 2p + q \wedge \exists k\ (k \geq 0 \wedge q = 4^k)$}
  while $q \neq 1$ **do**
    q:=q `div` 4; h:=$p + q$; p:=p `div` 2;
    if $r \geq h$ **then**
      p:=$p + q$; r:=$r - h$;
    end if
  end while
  { Post: $p^2 \leq N \wedge (p+1)^2 > N$}
end function

For the first loop, $\{p = 0 \wedge r = N\}$ is generated as an invariant. For the second loop, $\{p^2 + qr - qN = 0\}$ is generated as an invariant in 4 iterations.

The fifth example, also taken from [Dij76], is a version of Euclid's algorithm that computes the least common multiple of two natural numbers instead of its greatest common divisor. Since it is an extension of Euclid's algorithm, we know that $\{\text{gcd}(x, y) = \text{gcd}(a, b)\}$ is invariant in the loop. However, this is not enough to prove the (partial) correctness of the algorithm. The missing piece,

$\{xu + yv = 2ab\}$, crucial to show the correctness of the program, is obtained automatically using our procedure in 4 iterations.

function lcm $(a, b$: integer) **returns** z: integer
{ Pre: $a > 0 \wedge b > 0$}
var x, y, u, v: integer **end var**
$\langle x, y, u, v\rangle$:=$\langle a, b, b, a\rangle$;
{ Inv: gcd(x, y) =gcd(a, b)}
while $x \neq y$ **do**
if $x > y \rightarrow$
$\langle x, y, u, v\rangle$:=$\langle x - y, y, u, u + v\rangle$;
[] $x < y \rightarrow$
$\langle x, y, u, v\rangle$:=$\langle x, y - x, u + v, v\rangle$;
end if
end while
$z := (u + v)/2$;
$\{Post : z = \text{lcm}(a, b)\}$
end function

The last program is yet another version of Euclid's algorithm that computes the greatest common divisor of two natural numbers together with Bezout's coefficients:

function euclid_extended $(x, y$: integer) **returns** a, p, r: integer
{ Pre: $x > 0 \wedge y > 0$}
var b, q, s: integer **end var**
$\langle a, b, p, q, r, s\rangle$:=$\langle x, y, 1, 0, 0, 1\rangle$;
{ Inv: gcd(x, y) =gcd(a, b)}
while $a \neq b$ **do**
if $a > b \rightarrow$
$\langle a, b, p, q, r, s\rangle$:=$\langle a - b, b, p - q, q, r - s, s\rangle$;
[] $x < y \rightarrow$
$\langle a, b, p, q, r, s\rangle$:=$\langle a, b - a, p, q - p, r, s - r\rangle$;
end if
end while
{ Post: gcd$(x, y) = px + ry$}
end function

For this example our procedure yields the conjunction:

$$\{1+qr-sp = 0 \wedge rb-sa+x = 0 \wedge sy+qx-b = 0 \wedge bp-aq-y = 0 \wedge yr+xp-a = 0\}$$

as an invariant in 5 iterations.

6 Application of Polynomial Invariant Inference to Program Verification

We have used the polynomial invariants obtained with our algorithm to prove automatically the (partial) correctness of all the examples shown in this paper.

Our verifier is still at a very early stage of development. The programming language it accepts is a sub-language of `C`, featuring integer variables, the usual arithmetic operations ($+$, $*$, `div`, `mod`, etc.) and function calls. Programs are annotated with pre-postconditions and loop invariants. The polynomial invariants are obtained automatically with the implementation of the techniques here presented and for the time being have to be inserted manually; other kind of invariants, like linear inequalities or formulas involving exponentials and modulo operations, have to be supplied by the user. Additionally, given any program point, the user can ensure properties that hold at that point via `assume` statements, or check whether a certain property is true by means of the command `assert`.

Basically the verifier consists of three components. The verification condition generator creates a list of conditions from the code and the annotations that must be satisfied in order to ensure the partial correctness of the program. The conditions are generated according to Floyd-Hoare-Dijkstra's axiomatic semantics formalism, so that each verification condition is associated to a fragment of code.

The list of verification conditions is then given to a theorem prover, which tries to check that the conditions are met. So far we have tried two theorem provers: the first-order-logic general-purpose theorem prover SPASS (see [WBH+02]), and a simple prover for the integers that we have implemented in Prolog. Our first approach was to use SPASS taking a list of properties of the integers as axioms; although for simple examples it would succeed, SPASS had difficulties dealing with theorems that require algebraic manipulation and knowledge of the properties of the integer numbers. Because the list of axioms was growing exceedingly we decided to implement our own solver, which has given overall good results so far. However, for some programs like the following one, we are able to find polynomial invariants but not to prove correctness:

function divisor (N, D: integer) **returns** d, r: integer
  { Pre: $N > 0 \wedge N \texttt{ mod } 2 = 1 \wedge D \texttt{ mod } 2 = 1 \wedge D \geq 2\sqrt[3]{n} + 1$}
  var t, q: integer **end var**
  $\langle d, r, t, q\rangle$:=$\langle D, N \texttt{ mod } D, N \texttt{ mod } (D-2), 4 * (N \texttt{ div } (D-2) - N \texttt{ div } D)\rangle$;
  { Inv: $d \texttt{ mod } 2 = 1 \wedge d(dq - 4r + 4t - 2q) + 8r = 8N$}
  while $d \leq \lfloor\sqrt{(N)}\rfloor \wedge r \neq 0$ **do**
    $\langle d, r, t, q\rangle$:=$\langle d + 2, 2r - t + q, r, q\rangle$;
    if $r < 0$ **then**
      $\langle r, q\rangle$:=$\langle r + d, q + 4\rangle$;
    end if
    if $r \geq d$ **then**
      $\langle r, q\rangle$:=$\langle r - d, q - 4\rangle$;
    end if
    if $r \geq d$ **then**
      $\langle r, q\rangle$:=$\langle r - d, q - 4\rangle$;
    end if
  end while

{ Post: $d \neq 0 \wedge (r = 0 \Rightarrow N \texttt{ mod } d = 0)$}
end function

Finally a third component analizes the result of the theorem proving stage and reports whether the conditions have been satisfied or not. Each fragment of code is then tagged with "ok"(if the condition has been proved to hold), "error" (if the condition has been proved not to hold) or "don't know" (if the theorem prover has run out of time before deciding the validity of the condition).

Our plans in the near future are the following:

- enrich the set of data structures admitted by the verifier so that it includes records, arrays, lists and other recursive data structures.
- integrate completely in the verifier the algorithm presented in this paper and other techniques for mechanically inferring loop invariants (like linear inequalities, see [CH78] and [CSS03]), together with other theorem proving components.

7 Conclusions and Further Research

An abstract framework for automatically discovering invariants of loops with assignments and conditional statements is proposed. A general procedure for that is given if the language used to express invariants is expressive, disjunctively closed and allows quantifier elimination. It is shown that the procedure computes the strongest possible invariant expressible in the language.

This generic framework is then instantiated for the theory of polynomial equations closed under conjunction. It is proved that for this language, the strongest invariant of a loop can be automatically computed provided the assignment statements in the loop are solvable. Furthermore, the procedure for computing polynomial invariants is guaranteed to terminate if the eigenvalues of the assignment mappings are positive. This algorithm has been implemented in Maple using the Gröbner basis algorithm for computing intersection of ideals as well as for eliminating variables. The algorithm has been applied to many non-trivial programs and has successfully discovered invariants which turn out to be essential when proving correctness.

In the literature there are a number of methods that have been proposed to automate invariant generation. Due to the difficulty of the problem, most researchers agree that different methods have different strengths and weaknesses, and therefore a convenient combination of them is advisable. The aproach described here should be considered as a further useful tool in the repertoire of techniques that attempt to automate the inference of loop invariants. Nonetheless, our methods are promising when they can be applied. They do not depend on execution traces and adequate test suites, like [ECGN01] and other dynamic strategies. They do not need any pre-postconditions as in [Weg74] or [IS97]. However, if the precondition is available, it can be used to possibly speed up the computation and find invariants using that hypothesis. Furthermore, our techniques deal correctly with loops with conditionals, whereas in the difference

equations method ([EGLW72]) only heuristics could be applied. Moreover, while Cousot and Halbwachs ([CH78]) had to introduce the *widening* operator ∇ in order to get termination at the cost of completeness, our method is fine enough to guarantee completeness. As regards the work in [CSS03], Colón et al. use non-linear constraint solving and quantifier elimination, which is a much more difficult problem than elimination of variables in polynomial equalities. Finally, when they can be applied, our techniques have the advantatge over Müller-Olm and Seidl's ([Mar03]) that there is no bound on the degree of the polynomials, and that polynomial solvable assignments are allowed.

For future work, we are interested in exploring the proposed research along several directions:

- identify languages, possibly rich enough to specify properties of aggregate data structures such as arrays, records, other recursive data structures, pointers, etc, to which the generic framework applies.
- enrich the programming model to consider nested loops as well as procedure calls.
- explore the relationship between intermediate lemma speculation, needed to mechanize theorem proving by induction of tail recursive programs, and automatic inference of invariants of loops.
- extend our current prototype of verifier along the lines described in Section 6.

8 Acknowledgements

The authors would like to thank G.Godoy, R. Nieuwenhuis and A. Oliveras for their help and wise pieces of advice in previous versions of this paper.

References

[AL94] William W. Adams and Philippe Loustaunau. *An Introduction to Gröbner Bases*. American Mathematical Society, 1994.

[BW93] Thomas Becker and Volker Weispfenning. *Gröbner Bases. A Computational Approach to Commutative Algebra*. Springer-Verlag, 1993.

[CC77] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[CH78] P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints among Variables of a Program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press, New York, NY.

[CK] Enric Rodríguez Carbonell and Deepak Kapur. Algegraic geometry for finding polynomial loop invariants.

[CLO98] David Cox, John Little, and Donal O'Shea. *Ideals, Varieties and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer-Verlag, 1998.

[CSS03] Michael A. Colón, Sriram Sankaranarayanan, and Henny B. Sipma. Linear Invariant Generation Using Non-Linear Constraint Solving. In *Computer-Aided Verification (CAV 2003)*, volume 2725 of *Lecture Notes in Computer Science*, pages 420–432. Springer Verlag, 2003.

[Dij76] E. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[ECGN01] M.D. Ernst, J. Cockrell, W.G. Griswold, and D. Notkin. Dynamically Discovering Likely Program Invariants to Support Program Evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, Feb 2001.

[EGLW72] B. Elspas, M. W. Green, K. N. Levitt, and R. J. Waldinger. Research in Interactive Program-Proving Techniques. Technical report, Stanford Research Institute, Menlo Park, California, USA, May 1972.

[FL01] Cormac Flanagan and K. Rustan M. Leino. Houdini, an Annotation Assistant for ESC/Java. In *Proceedings of Formal Methods Europe (FME)*, volume 2021 of *LNCS*, pages 500–517, 2001.

[Huy86] D.T. Huynh. A superexponential lower bound for Gröbner bases and Church-Rosser commutative Thue systems. *Information and Control*, 68(1-3):196–206, 1986.

[IS97] A. Ireland and J. Stark. On the Automatic Discovery of Loop Invariants. In *Proceedings of the Fourth NASA Langley Formal Methods Workshop, NASA Conference Publication 3356*, 1997.

[Kar76] M. Karr. Affine Relationships Among Variables of a Program. *Acta Informatica*, 6:133–151, 1976.

[Knu69] Donald E. Knuth. *The Art of Computer Programming. Volume 2, Seminumerical Algorithms*. Addison-Wesley, 1969.

[Mar03] Markus Müller-Olm and Helmut Seidl. Computing Interprocedurally Valid Relations in Affine Programs. *Technical Report, University of Trier, to appear in POPL 2004*, 2003.

[NE96] Jeremy W. Nimmer and Michael D. Ernst. Static verification of dynamically detected program invariants: Integrating Daikon and ESC/Java. In *Proceedings of RV'01, First Workshop on Runtime Verification*, Paris, France, Jul 1996.

[Nom66] Katsumi Nomizu. *Fundamentals of Linear Algebra*. McGraw-Hill, 1966.

[Sta97] Richard P. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, 1997.

[WBH+02] Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobald, and Dalibor Topic. SPASS version 2.0. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, volume 2392 of *Lecture Notes in Artificial Intelligence*, pages 275–279, Kopenhagen, Denmark, 2002. Springer.

[Weg74] Ben Wegbreit. The Synthesis of Loop Predicates. *Communications of the ACM*, 17(2):102–112, Feb 1974.

[Wei75] Ben Weigbreit. Property extraction in well-founded property sets. *IEEE Transactions on Software Engineering*, 1(3):270–285, Sep 1975.

[WS03] Christoph Walther and Stephan Schweitzer. Verification in the classroom. *To appear in Journal of Automated Reasoning - Special Issue on Automated Reasoning and Theorem Proving in Education*, pages 1–43, 2003.

A Correctness and Completeness of the General Procedure

First of all we need the next lemma, which states that at any iteration of the General Procedure the invariant candidate stored in the formula variable R is implied by the initial formula:

Lemma 5. *$\Big(\bigwedge_{j=1}^{m}(x_j = x_j^*)\Big) \wedge R_0(\bar{x}^*) \Rightarrow R(\bar{x}, \bar{x}^*)$ is an invariant of the General Procedure.*

Proof. It trivially holds at the beginning of the General Procedure. And

$$\Big(\bigwedge_{j=1}^{m}(x_j = x_j^*)\Big) \wedge R_0(\bar{x}^*) \Rightarrow R(\bar{x}, \bar{x}^*)$$

implies

$$\Big(\bigwedge_{j=1}^{m}(x_j = x_j^*)\Big) \wedge R_0(\bar{x}^*) \Rightarrow R(\bar{x}, \bar{x}^*) \vee \Big(\bigsqcup_{i=1}^{n}\varphi(R,i)\Big)(\bar{x}, \bar{x}^*)$$

and then

$$\Big(\bigwedge_{j=1}^{m}(x_j = x_j^*)\Big) \wedge R_0(\bar{x}^*) \Rightarrow R(\bar{x}, \bar{x}^*) \sqcup \Big(\bigsqcup_{i=1}^{n}\varphi(R,i)\Big)(\bar{x}, \bar{x}^*)$$

□

The following proposition gives correctness, since the hypothesis is satisfied on termination of the General Procedure:

Proposition 1. *If the formula stored in the formula variable R (that we also denote by R) is such that*

$$R(\bar{x}, \bar{x}^*) \Leftrightarrow R \sqcup \Big(\bigsqcup_{i=1}^{n}\varphi(R,i)\Big)(\bar{x}, \bar{x}^*)$$

then R is $\mathcal{R}$-invariant with respect to R_0.

Proof. First let us see that $R_0(\bar{x}^*) \Rightarrow R(\bar{x}^*, \bar{x}^*)$. Let us take $\bar{\alpha}^* \in \Sigma$ such that $R_0(\bar{\alpha}^*)$ holds. Now as $\bigwedge_{j=1}^{m}(\alpha_j^* = \alpha_j^*)$ also holds, by Lemma 5 we have that $R(\bar{\alpha}^*, \bar{\alpha}^*)$ holds.

Now let us take $i : 1 \leq i \leq n$ and prove that $R(\bar{x}, \bar{x}^*) \wedge E(\bar{x}) \wedge C_i(\bar{x}) \Rightarrow R(f_i(\bar{x}), \bar{x}^*)$. Taking $s = 1$ and $\bar{y} = \bar{x}$:

$$R(\bar{x}, \bar{x}^*) \wedge E(\bar{x}) \wedge C_i(\bar{x}) \Rightarrow$$

$$\Rightarrow \bigvee_{s=1}^{\infty} \exists \bar{y}\Big(f_i(\bar{x}) = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1}\big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y}))\big)\Big)\Big) \Rightarrow$$

$$\Rightarrow \varphi(R,i)(f_i(\bar{x}),\bar{x}^*) \Rightarrow R(f_i(\bar{x}),\bar{x}^*) \vee \Big(\bigvee_{j=1}^{n} \varphi(R,j)(f_i(\bar{x}),\bar{x}^*)\Big) \Rightarrow$$

$$\Rightarrow R \sqcup \Big(\bigsqcup_{j=1}^{n} \varphi(R,j)\Big)(f_i(\bar{x}),\bar{x}^*) \Rightarrow R(f_i(\bar{x}),\bar{x}^*)$$

□

In order to show the completeness of the procedure we need the following lemma, which states that we always have a formula stored in R which is stronger than the $\mathcal{R}$-invariant R_∞ that we are looking for:

Lemma 6. *The implication $R(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$ is an invariant of the General Procedure.*

Proof. Let us see that it is true at the beginning of the loop. Indeed, $(\bigwedge_{j=1}^{m}(x_j = x_j^*)) \wedge R_0(\bar{x}^*) \Rightarrow \bar{x} = \bar{x}^* \wedge R_0(\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$ since R_∞ is $\mathcal{R}$-invariant.

Now we have to prove that $R(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$ implies

$$R \sqcup \Big(\bigsqcup_{i=1}^{n} \varphi(R,i)\Big)(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

In order to do that let us fix $i : 1 \leq i \leq n$ and prove by induction that $\forall s \in \mathbb{N}$

$$\exists \bar{y}\Big(\bar{x} = f_i^s(\bar{y}) \wedge R_\infty(\bar{y},\bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y}))\big)\Big)\Big) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

For $s = 0$ we have to see that

$$\exists \bar{y}(\bar{x} = \bar{y} \wedge R_\infty(\bar{y},\bar{x}^*)) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

which is obviously true. Now let us assume that $s \geq 1$. Given $\bar{\alpha}$, $\bar{\alpha}^* \in \Sigma$, let $\bar{\gamma} \in \Sigma$ be such that

$$\bar{\alpha} = f_i^s(\bar{\gamma}) \wedge R_\infty(\bar{\gamma},\bar{\alpha}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{\gamma})) \wedge C_i(f_i^t(\bar{\gamma}))\big)\Big)$$

We want to prove that then $R_\infty(\bar{\alpha},\bar{\alpha}^*)$ holds. By induction hypothesis we know that

$$\exists \bar{y}\Big(\bar{x} = f_i^{s-1}(\bar{y}) \wedge R_\infty(\bar{y},\bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-2} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y}))\big)\Big)\Big) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

Then

$$R_\infty(\bar{\gamma},\bar{\alpha}^*) \wedge \Big(\bigwedge_{t=0}^{s-2} \big(E(f_i^t(\bar{\gamma})) \wedge C_i(f_i^t(\bar{\gamma}))\big)\Big) \Rightarrow R_\infty(f_i^{s-1}(\bar{\gamma}),\bar{\alpha}^*)$$

Moreover, we notice that since R_∞ is $\mathcal{R}$-invariant,

$$R_\infty(\bar{x},\bar{x}^*) \wedge E(\bar{x}) \wedge C_i(\bar{x}) \Rightarrow R_\infty(f_i(\bar{x}),\bar{x}^*)$$

In particular,

$$R_\infty(f_i^{s-1}(\bar{\gamma}),\bar{\alpha}^*) \wedge E(f_i^{(s-1)}(\bar{\gamma})) \wedge C_i(f_i^{(s-1)}(\bar{\gamma})) \Rightarrow R_\infty(f_i^{s}(\bar{\gamma}),\bar{\alpha}^*)$$

Therefore

$$\bar{\alpha} = f_i^s(\bar{\gamma}) \wedge R_\infty(\bar{\gamma},\bar{\alpha}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{\gamma})) \wedge C_i(f_i^t(\bar{\gamma}))\big)\Big) \Rightarrow$$

$$\bar{\alpha} = f_i^s(\bar{\gamma}) \wedge R_\infty(\bar{\gamma},\bar{\alpha}^*) \wedge (\bigwedge_{t=0}^{s-2} (E(f_i^t(\bar{\gamma})) \wedge C_i(f_i^t(\bar{\gamma})))) \wedge E(f_i^{s-1}(\bar{\gamma})) \wedge C_i(f_i^{s-1}(\bar{\gamma})) \Rightarrow$$

$$\bar{\alpha} = f_i^s(\bar{\gamma}) \wedge R_\infty(f_i^{s-1}(\bar{\gamma}),\bar{\alpha}^*) \wedge E(f_i^{s-1}(\bar{\gamma})) \wedge C_i(f_i^{s-1}(\bar{\gamma})) \Rightarrow$$

$$\bar{\alpha} = f_i^s(\bar{\gamma}) \wedge R_\infty(f_i^{s}(\bar{\gamma}),\bar{\alpha}^*) \Rightarrow R_\infty(\bar{\alpha},\bar{\alpha}^*)$$

So $\forall s \in \mathbb{N}$

$$\exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R_\infty(\bar{y},\bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y}))\big)\Big)\Big) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

As $R(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$, we have

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y},\bar{x}^*) \wedge \Big(\bigwedge_{t=0}^{s-1} \big(E(f_i^t(\bar{y})) \wedge C_i(f_i^t(\bar{y}))\big)\Big)\Big) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

By the properties of φ, since $R_\infty \in \mathcal{R}$ we have that $\forall i : 1 \leq i \leq n$

$$\varphi(R,i)(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

Finally using induction, $R_\infty \in \mathcal{R}$ and the properties of $\sqcup$ it is easy to prove that

$$\Big(\bigsqcup_{i=1}^{n} \varphi(R,i)\Big)(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

and since $R(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$ we have that

$$R \sqcup \Big(\bigsqcup_{i=1}^{n} \varphi(R,i)\Big)(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$$

□

Finally, the following result summarizes both correctness and completeness:

Theorem 5. *If the procedure terminates, then* $R(\bar{x},\bar{x}^*) \Leftrightarrow R_\infty(\bar{x},\bar{x}^*)$

Proof. If the procedure terminates, then $R(\bar{x},\bar{x}^*) \Leftrightarrow R \sqcup (\sqcup_{i=1}^{n} \varphi(R,i))(\bar{x},\bar{x}^*)$. By Proposition 1 R is an $\mathcal{R}$-invariant and so $R_\infty(\bar{x},\bar{x}^*) \Rightarrow R(\bar{x},\bar{x}^*)$. But on the other hand, Lemma 6 implies that $R(\bar{x},\bar{x}^*) \Rightarrow R_\infty(\bar{x},\bar{x}^*)$.
□

B $\mathcal{P}$ allows quantifier elimination

Lemma 7. *$\mathcal{P}$ allows quantifier elimination.*

Proof. For $i : 1 \leq i \leq n$ and R we can compute a basis B of the ideal

$$\mathbb{Q}[\bar{x}, \bar{x}^*] \bigcap \Big(\bigcap_{s=1}^{\infty} \Big\langle (-\bar{x} + f_i^s(\bar{y})) \bigcup \Big(\bigcup_{p \in \mathbb{IV}(\phi(R))} p(\bar{y}, \bar{x}^*) \Big) \Big\rangle \Big)$$

where $-\bar{x} + f_i^s(\bar{y})$ denotes a set of m polynomials, one for each of the m components. Then we can define $\varphi_i(R)(\bar{x}, \bar{x}^*) := (\bigwedge_{p \in B} \ p(\bar{x}, \bar{x}^*) = 0)$.

Let us check that the φ_i's defined in this way satisfy that

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \Big) \Rightarrow \varphi_i(R)(\bar{x}, \bar{x}^*)$$

and that $\forall T \in \mathcal{P}$ such that

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \Big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \Big) \Rightarrow T(\bar{x}, \bar{x}^*)$$

then $\varphi_i(R)(\bar{x}, \bar{x}^*) \Rightarrow T(\bar{x}, \bar{x}^*)$.

First of all, we have to prove that

$$\bigvee_{s=1}^{\infty} \exists \bar{y} \big(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*) \big) \Rightarrow \varphi_i(R)(\bar{x}, \bar{x}^*)$$

Indeed, let $\bar{\alpha}, \bar{\alpha}^* \in \mathbb{Q}^m$ be such that the above formula is true; that is to say, $\exists s_0 \in \mathbb{N}, s_0 \ \geq 1 \ \exists \bar{\gamma} \in \mathbb{Q}^m$ satisfying $\bar{\alpha} = f_i^{s_0}(\bar{\gamma})$ and $R(\bar{\gamma}, \bar{\alpha}^*)$, i.e. $(\bar{\gamma}, \bar{\alpha}^*) \in \mathbb{V}(\phi(R))$. We have to see that then $\varphi_i(R)(\bar{\alpha}, \bar{\alpha}^*)$ holds, or equivalently, that $(\bar{\alpha}, \bar{\alpha}^*) \in \mathbb{V}(\phi(\varphi_i(R))) = \mathbb{V}(B)$. So given any polynomial $p(\bar{x}, \bar{x}^*) \in B$ let us see that $p(\bar{\alpha}, \bar{\alpha}^*) = 0$. If $P_1, ..., P_k$ is a basis for $\mathbb{IV}(\phi(R))$, as

$$p \in \langle B \rangle \subseteq \Big\langle (-\bar{x} + f_i^{s_0}(\bar{y})) \bigcup \Big(\bigcup_{p \in \mathbb{IV}(\phi(R))} p(\bar{y}, \bar{x}^*) \Big) \Big\rangle$$

we can write for certain polynomials $q_1, ..., q_m, p_1, ..., p_k \in \mathbb{Q}[\bar{x}, \bar{x}^*, \bar{y}]$

$$p(\bar{x}, \bar{x}^*) = \sum_{j=1}^{m} q_j(\bar{x}, \bar{x}^*, \bar{y})(-x_j + \pi_j(f_i^{s_0}(\bar{y}))) + \sum_{l=1}^{k} p_l(\bar{x}, \bar{x}^*, \bar{y}) P_l(\bar{y}, \bar{x}^*)$$

where $\pi_j : \mathbb{Q}^m \longrightarrow \mathbb{Q}$ is the projection over the j-th coordinate. Then evaluating at $\bar{x} = \bar{\alpha}, \bar{x}^* = \bar{\alpha}^*, \bar{y} = \bar{\gamma}$

$$p(\bar{\alpha}, \bar{\alpha}^*) = \sum_{j=1}^{m} q_j(\bar{\alpha}, \bar{\alpha}^*, \bar{\gamma})(-\bar{\alpha}_j + \pi_j(f_i^{s_0}(\bar{\gamma}))) + \sum_{l=1}^{k} p_l(\bar{\alpha}, \bar{\alpha}^*, \bar{\gamma}) P_l(\bar{\gamma}, \bar{\alpha}^*) =$$

$$= \sum_{j=1}^{m} q_j(\bar{\alpha}, \bar{\alpha}^*, \bar{\gamma})(-\bar{\alpha}_j + \bar{\alpha}_j) = 0$$

since $\bar{\alpha} = f_i^{s_0}(\bar{\gamma})$, $(\bar{\gamma}, \bar{\alpha}^*) \in \mathbb{V}(\phi(R))$ and $P_l \in \mathbb{IV}(\phi(R))$ for $1 \leq l \leq k$.

Now we have to check that $\forall T \in \mathcal{P}$ such that

$$\bigvee_{s=1}^{\infty} \exists \bar{y}(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*)) \Rightarrow T(\bar{x}, \bar{x}^*)$$

then $\varphi_i(R)(\bar{x}, \bar{x}^*) \Rightarrow T(\bar{x}, \bar{x}^*)$. First, let us show that $\forall p \in \mathbb{Q}[\bar{x}, \bar{x}^*]$ such that

$$\bigvee_{s=1}^{\infty} \exists \bar{y}(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*)) \Rightarrow p(\bar{x}, \bar{x}^*) = 0$$

we have $p \in \langle B \rangle$. Let us assume that $p \notin \langle B \rangle$ and we will get a contradiction. If this is the case then $\exists s_0 \in \mathbb{N}, s_0 \geq 1$ such that

$$p \notin \left\langle (-\bar{x} + f_i^{s_0}(\bar{y})) \bigcup \Big(\bigcup_{p \in \mathbb{IV}(\phi(R))} p(\bar{y}, \bar{x}^*) \Big) \right\rangle$$

Let us consider any monomial ordering $\succ$ such that $x_j \succ x_j^*, y_j$ for $1 \leq j \leq m$, for example $\text{lex}(x_1 > ... > x_m > x_1^* > ... > x_m^* > y_1 > ... > y_m)$. Let us consider the set $H = \{-x_j + \pi_j(f_i^{s_0}(\bar{y})) \mid 1 \leq j \leq m\}$, where again $\pi_j : \mathbb{Q}^m \longrightarrow \mathbb{Q}$ is the projection over the j-th coordinate. Applying the division algorithm for polynomials in multiple variables to p and H, we get certain polynomials r, q_j $(1 \leq j \leq m)$ such that

$$p(\bar{x}, \bar{x}^*) = r(\bar{x}^*, \bar{y}) + \sum_{j=1}^{m} q_j(\bar{x}, \bar{x}^*, \bar{y})(-x_j + \pi_j(f_i^{s_0}(\bar{y})))$$

Since the leading monomial of $-x_j + \pi_j(f_i^{s_0}(\bar{y}))$ with respect to $\succ$ is x_j, we can guarantee that r does not depend on $\bar{x}$.

Now let us take an arbitrary $(\bar{\gamma}, \bar{\alpha}^*) \in \mathbb{V}(\phi(R))$. Then $R(\bar{\gamma}, \bar{\alpha}^*)$ holds, and taking $\bar{y} = \bar{\gamma}$ the formula

$$\exists \bar{y}(f_i^{s_0}(\bar{\gamma}) = f_i^{s_0}(\bar{y}) \wedge R(\bar{y}, \bar{\alpha}^*))$$

also holds. By definition of p, taking $\bar{y} = \bar{\gamma}$ again

$$0 = p(f_i^{s_0}(\bar{\gamma}), \bar{\alpha}^*) = r(\bar{\alpha}^*, \bar{\gamma}) + \sum_{j=1}^{m} q_j(f_i^{s_0}(\bar{\gamma}), \bar{\alpha}^*, \bar{\gamma})(-\pi_j(f_i^{s_0}(\bar{\gamma})) + \pi_j(f_i^{s_0}(\bar{\gamma}))) =$$

$$= r(\bar{\alpha}^*, \bar{\gamma})$$

Then we have that $r \in \mathbb{IV}(\phi(R))$, since $(\bar{\gamma}, \bar{\alpha}^*) \in \mathbb{V}(\phi(R))$ was arbitrary. So

$$p \in \left\langle (-\bar{x} + f_i^{s_0}(\bar{y})) \bigcup \Big(\bigcup_{p \in \mathbb{IV}(\phi(R))} p(\bar{y}, \bar{x}^*) \Big) \right\rangle$$

which is a contradiction.

Now if T is such that

$$\bigvee_{s=1}^{\infty} \exists \bar{y}(\bar{x} = f_i^s(\bar{y}) \wedge R(\bar{y}, \bar{x}^*)) \Rightarrow T(\bar{x}, \bar{x}^*)$$

then $\forall p \in \phi(T)\ p \in \langle B \rangle$. And this clearly implies that $\varphi_i(R)(\bar{x}, \bar{x}^*) \Rightarrow T(\bar{x}, \bar{x}^*)$. □

C Powers of solvable mappings

First of all we need the following lemma, which describes the form of the solutions of the recurrences that arise when computing powers of solvable mappings:

Lemma 8. *Consider a recurrence*

$$\begin{pmatrix} x_1^{(s+1)} \\ \vdots \\ x_r^{(s+1)} \end{pmatrix} = M \begin{pmatrix} x_1^{(s)} \\ \vdots \\ x_r^{(s)} \end{pmatrix} + Q(s, \bar{y})$$

where $M \in \mathbb{Q}^{r \times r}$ is a matrix with rational eigenvalues and Q is a vector of r functions of the form $\sum_{l=1}^{k} Q_l(s, \bar{y})\mu_l^s$, where for $1 \leq l \leq k$, $Q_l \in \mathbb{Q}[s, \bar{y}]$ and $\mu_l \in \mathbb{Q}$ (the $\bar{y}$ variables represent parameters). Then the solutions of the recurrence have the form:

$$x_j^{(s)} = \sum_{l=1}^{k_j} P_{jl}(s, \bar{y}, \bar{x}^{(0)})\gamma_{jl}^s$$

where for $1 \leq j \leq r$, $1 \leq l \leq k_j$, $P_{jl} \in \mathbb{Q}[s, \bar{y}, \bar{x}^{(0)}]$ and the $\gamma_{jl} \in \mathbb{Q}$ are either the eigenvalues of M or belong to the set of bases of exponentials in $Q(s, \bar{y})$.

Proof. If $M \in \mathbb{Q}^{r \times r}$ is a matrix with rational eigenvalues, then $\exists S, J \in \mathbb{Q}^{r \times r}$ such that $\det(S) \neq 0$ and $J = S^{-1}MS$ is the Jordan normal form of M. By making a change of variables and splitting the variables in independent sets we can assume without loss of generality that M has the structure of a Jordan block, so that for a certain $\lambda \in \mathbb{Q}$ eigenvalue of M

$$M = \begin{pmatrix} \lambda & & & \\ 1 & \lambda & & \\ & & \ddots & \\ & & 1 & \lambda \end{pmatrix}$$

Now we use the theory of generating functions, linear recurrences with constant coefficients and rational functions (see [Sta97]). We denote by $F_j(z)$ the generating function of the sequence $(x_j^{(s)})_{s \in \mathbb{N}}$. Since the components of $Q(s, \bar{y})$

are linear combinations of exponentials with polynomial coefficients, the corresponding generating functions are rational functions with rational coefficients $G_j(z,\bar{y})/H_j(z)$ such that the roots of the H_j are in the set of bases of exponentials in $Q(s,\bar{y})$.

From the recurrence we get the following system of equations for the F_j's:

$$\begin{pmatrix} \frac{F_1(z)-x_1^{(0)}}{z} \\ \vdots \\ \frac{F_r(z)-x_r^{(0)}}{z} \end{pmatrix} = M \begin{pmatrix} F_1(z) \\ \vdots \\ F_r(z) \end{pmatrix} + \begin{pmatrix} \frac{G_1(z,\bar{y})}{H_1(z)} \\ \vdots \\ \frac{G_r(z,\bar{y})}{H_r(z)} \end{pmatrix}$$

The solution to this system is

$$\begin{pmatrix} F_1(z) \\ \vdots \\ F_r(z) \end{pmatrix} = (I - zM)^{-1} \left(\begin{pmatrix} x_1^{(0)} \\ \vdots \\ x_r^{(0)} \end{pmatrix} + z \begin{pmatrix} \frac{G_1(z,\bar{y})}{H_1(z)} \\ \vdots \\ \frac{G_r(z,\bar{y})}{H_r(z)} \end{pmatrix} \right)$$

where

$$(I - zM)^{-1} = \begin{pmatrix} \frac{1}{1-\lambda z} & & & \\ \frac{z}{(1-\lambda z)^2} & \frac{1}{1-\lambda z} & & \\ \vdots & \ddots & \ddots & \\ \frac{z^{r-1}}{(1-\lambda z)^r} & \cdots & \frac{z}{(1-\lambda z)^2} & \frac{1}{1-\lambda z} \end{pmatrix}$$

Therefore the generating functions $F_j(z)$ are also rational functions with poles either in the eigenvalues of M or in the set of bases of exponentials in $Q(s,\bar{y})$. From the theory of rational generating functions we get that the solutions to the recurrence have the form as in the statement of the lemma.
□

Now we are able to describe the form of the general powers of solvable mappings, using the equivalence of computing powers of mappings and solving recurrences:

Theorem 3. *Let $g \in \mathbb{Q}[\bar{x}]^m$ be a solvable mapping with rational eigenvalues. Then for $1 \leq j \leq m$ $g_j^s(\bar{x})$, the j-th component of $g^s(\bar{x})$, can be expressed as*

$$g_j^s(\bar{x}) = \sum_{l=1}^{k_j} P_{jl}(s,\bar{x})(\gamma_{jl})^s,\ 1 \leq j \leq m,\ s \geq 0$$

where for $1 \leq j \leq m,\ 1 \leq l \leq k_j$, $P_{jl} \in \mathbb{Q}[s,\bar{x}]$ and $\gamma_{jl} \in \mathbb{Q}$. Moreover, the γ_{jl} are products of the eigenvalues of g.

Proof. It is clear that the statement is equivalent to showing the following. Given a solvable mapping with rational eigenvalues $g \in \mathbb{Q}[\bar{x}]^m$, we have to prove that the general solution of the recurrence $\bar{x}^{(s+1)} = g(\bar{x}^{(s)})$ has the form for $1 \leq j \leq m$:

$$x_j^{(s)} = \sum_{l=1}^{k_j} P_{jl}(s,\bar{x}^{(0)})(\gamma_{jl})^s,\ 1 \leq j \leq m,\ s \geq 0$$

where for $1 \leq j \leq m,\ 1 \leq l \leq k_j$, $P_{jl} \in \mathbb{Q}[s, \bar{x}^{(0)}]$ and $\gamma_{jl} \in \mathbb{Q}$; we also have to show that the γ_{jl} are products of the eigenvalues of g.

Since g is solvable there exists a partition of the set of variables $\bar{x}$, $\bar{x} = \bigcup_{i=1}^{k} U_i$ with $U_i \cap U_j = \emptyset$ if $i \neq j$, such that $\forall i : 1 \leq i \leq k$ we have

$$g_{U_i}(\bar{x}) = M_i U_i^T + P_i(U_1, ..., U_{i-1})$$

where $M_i \in \mathbb{Q}^{|U_i| \times |U_i|}$ is a matrix with rational eigenvalues and P_i is a vector of $|U_i|$ polynomials with coefficients in $\mathbb{Q}$ and depending on the variables in $U_1, ..., U_{i-1}$.

Let us prove the proposition by induction on i, the counter of the sets in the partition. By renaming the variables we can assume without loss of generality that there exist $0 = r_0 \leq r_1 \leq r_2 \leq \cdots \leq r_k = m$ such that $\forall i : 1 \leq i \leq k$ $U_i = \{x_{r_{i-1}+1}, x_{r_{i-1}+2}, ..., x_{r_i}\}$.

For $i = 0$ we want to prove that $\forall x_j$ with $1 \leq j \leq r_1$, $x_j^{(s)}$ has the form like in the statement. For the first r_1 variables we have the recurrence:

$$\begin{pmatrix} x_1^{(s+1)} \\ \vdots \\ x_{r_1}^{(s+1)} \end{pmatrix} = M_1 \begin{pmatrix} x_1^{(s)} \\ \vdots \\ x_{r_1}^{(s)} \end{pmatrix} + P_1$$

where M_1 is a matrix with rational eigenvalues and P_1 is a constant vector. By Lemma 8, the $x_j^{(s)}$ have the desired form for $1 \leq j \leq r_1$. Moreover, since P_1 is constant, for $1 \leq j \leq r_1$ the bases of exponentials in $x_j^{(s)}$ are eigenvalues of M_1, and therefore eigenvalues of g.

Now for $i > 0$ we have the recurrence:

$$\begin{pmatrix} x_{r_{i-1}+1}^{(s+1)} \\ \vdots \\ x_{r_i}^{(s+1)} \end{pmatrix} = M_i \begin{pmatrix} x_{r_{i-1}+1}^{(s)} \\ \vdots \\ x_{r_i}^{(s)} \end{pmatrix} + P_i(x_1^{(s)}, ..., x_{r_{i-1}}^{(s)})$$

By induction hypothesis $\forall j : 1 \leq j \leq r_{i-1}$, $x_j^{(s)}$ has the form like in the statement. Therefore, if $\forall j : 1 \leq j \leq r_{i-1}$ we plug the solution $x_j^{(s)}$ in $P_i(x_1^{(s)}, ..., x_{r_{i-1}}^{(s)})$ we get that $P_i(x_1^{(s)}, ..., x_{r_{i-1}}^{(s)})$ is a vector of functions of the form

$$\sum_{l=1}^{k} Q_l(s, x_1^{(0)}, ..., x_{r_{i-1}}^{(0)}) \mu_l^s$$

where for $1 \leq l \leq k$, $Q_l \in \mathbb{Q}[s, x_1^{(0)}, ..., x_{r_{i-1}}^{(0)}]$ and $\mu_l \in \mathbb{Q}$. Notice that the bases of exponentials in $P_i(x_1^{(s)}, ..., x_{r_{i-1}}^{(s)})$ have to be products of eigenvalues of g, since we know that the bases of exponentials in the solutions $x_j^{(s)}$ for $j : 1 \leq j \leq r_{i-1}$ are products of eigenvalues of g. By Lemma 8 again, for $r_{i-1} < j \leq r_i$ the $x_j^{(s)}$ have the required form, and the bases of exponentials appearing in them

are either eigenvalues of M_i or bases of exponentials in $P_i(x_1^{(s)}, ..., x_{r_{i-1}}^{(s)})$; in any case they are products of eigenvalues of g, which is what we wanted to see.
□

D Proof of $\langle L \rangle = \mathcal{L}$

Proposition 2. *Let $\Lambda = \{\lambda_1, ..., \lambda_k\} \subset \mathbb{N}$ be a finite set of prime numbers, $\mathcal{L} = \{l \in \mathbb{Q}[s, \bar{u}, \bar{v}] \mid l(t, \bar{\lambda}^t, \bar{\lambda}^{-t}) = 0 \ \forall t \in \mathbb{N}\}$ the set of polynomial relations between the powers of these numbers, and $L = \{u_1 v_1 - 1, ..., u_k v_k - 1\}$. Then $\langle L \rangle_{\mathbb{Q}[s,\bar{u},\bar{v}]} = \mathcal{L}$*

Proof. Let $l \in \mathcal{L}$. Let us take any monomial ordering $>$ and let us divide l into L. Then we get polynomials $r, p_1, ..., p_k \in \mathbb{Q}[s, \bar{u}, \bar{v}]$ such that

$$l(s, \bar{u}, \bar{v}) = r(s, \bar{u}, \bar{v}) + \sum_{i=1}^{k} p_i(s, \bar{u}, \bar{v}) \cdot (u_i v_i - 1)$$

We want to show that $r = 0$. Let us assume that $r \neq 0$ and we will get a contradiction. We can write

$$r(s, \bar{u}, \bar{v}) = \sum_{\alpha,\beta \in \mathbb{N}^k} P_{\alpha,\beta}(s) \bar{u}^{\alpha} \bar{v}^{\beta}$$

for certain polynomials $P_{\alpha,\beta} \in \mathbb{Q}[s]$ such that at least one of them is not null, and only finitely many of them are not null.

Then $\forall t \in \mathbb{N}$ we have that

$$0 = l(t, \bar{\lambda}^t, \bar{\lambda}^{-t}) = r(t, \bar{\lambda}^t, \bar{\lambda}^{-t}) = \sum_{\alpha,\beta \in \mathbb{N}^k} P_{\alpha,\beta}(t) \prod_{i=1}^{k} \lambda_i^{(\alpha_i - \beta_i)t}$$

Given $\alpha, \beta \in \mathbb{N}^k$, let us define $\lambda_{\alpha,\beta} = \prod_{i=1}^{k} \lambda_i^{\alpha_i - \beta_i}$. Then the above equation can be expressed as $\forall t \in \mathbb{N}$

$$0 = \sum_{\alpha,\beta \in \mathbb{N}^k} P_{\alpha,\beta}(t) \lambda_{\alpha,\beta}^t$$

Now let us see that if $(\alpha, \beta) \neq (\gamma, \delta)$, then $\lambda_{\alpha,\beta} \neq \lambda_{\gamma,\delta}$. Let us assume the contrary. Then we have that $\prod_{i=1}^{k} \lambda_i^{\alpha_i - \beta_i - \gamma_i + \delta_i} = 1$. As the λ_i are different prime numbers, we necessarily have that $\alpha_i - \beta_i - \gamma_i + \delta_i = 0$ for $1 \leq i \leq k$. Moreover, since no monomial of r can be divided by the $u_i v_i$ by the properties of the division algorithm, either $\alpha_i = 0$ or $\beta_i = 0$, and either $\gamma_i = 0$ or $\delta_i = 0$. If $\alpha_i = 0$, then $\beta_i = \delta_i - \gamma_i$, and as $\beta_i \geq 0$ and either $\gamma_i = 0$ or $\delta_i = 0$, we get that $0 = \gamma_i = \alpha_i$ and $\beta_i = \delta_i$. The case $\beta_i = 0$ is symmetric. So $\lambda_{\alpha,\beta} = \lambda_{\gamma,\delta}$ implies $(\alpha, \beta) = (\gamma, \delta)$.

Let $\alpha^*, \beta^* \in \mathbb{N}^k$ be such that

$$\lambda_{\alpha^*,\beta^*} = \max\{\lambda_{\alpha,\beta} | P_{\alpha,\beta} \neq 0\}$$

Notice that α^*, β^* are well defined, since $r \neq 0$ by hypothesis. By definition, and as $(\alpha, \beta) \neq (\gamma, \delta)$ implies $\lambda_{\alpha,\beta} \neq \lambda_{\gamma,\delta}$, we have that $(\alpha, \beta) \neq (\alpha^*, \beta^*)$ implies $\lambda_{\alpha,\beta} < \lambda_{\alpha^*,\beta^*}$.

Now we divide into $(\lambda_{\alpha^*,\beta^*})^t$ and get that $\forall t \in \mathbb{N}$

$$0 = \sum_{\alpha,\beta \in \mathbb{N}^l} P_{\alpha,\beta}(t) \Big(\frac{\lambda_{\alpha,\beta}}{\lambda_{\alpha^*,\beta^*}} \Big)^t$$

Taking limits,

$$0 = \lim_{t \to \infty} P_{\alpha^*,\beta^*}(t)$$

which is in contradiction with the fact that $P_{\alpha^*,\beta^*} \neq 0$.

□

E Correctness and Completeness of the Implementation

First of all we need the following technical lemma. It intuitively means that $F_i(-s, \bar{v}, \bar{u}, \cdot)$ and $F_i(s, \bar{u}, \bar{v}, \cdot)$ are "inverses modulo $\langle L \rangle$":

Lemma 9. *For $1 \leq i \leq n$ and $\forall q \in \mathbb{Q}[\bar{x}, \bar{x}^*]$ we have that*

$$q(\bar{x}, \bar{x}^*) - q(F_i(s, \bar{u}, \bar{v}, F_i(-s, \bar{v}, \bar{u}, \bar{x})), \bar{x}^*) \in \langle L \rangle_{\mathbb{Q}[s,\bar{u},\bar{v},\bar{x},\bar{x}^*]}$$

Proof. We can write

$$q(\bar{x}, \bar{x}^*) - q(F_i(s, \bar{u}, \bar{v}, F_i(-s, \bar{v}, \bar{u}, \bar{x})), \bar{x}^*) =$$

$$= \sum_{\alpha,\beta \in \mathbb{N}^m} R_{\alpha,\beta}(s, \bar{u}, \bar{v})(\bar{x})^\alpha (\bar{x}^*)^\beta$$

with only a finite number of the $R_{\alpha,\beta}$ different from 0. Then $\forall t \in \mathbb{N}$

$$q(\bar{x}, \bar{x}^*) - q(F_i(t, \bar{\lambda}^t, \bar{\lambda}^{-t}, F_i(-t, \bar{\lambda}^{-t}, \lambda^t, \bar{x})), \bar{x}^*) =$$
$$= q(\bar{x}, \bar{x}^*) - q(f_i^t(f_i^{-t}(\bar{x})), \bar{x}^*) = q(\bar{x}, \bar{x}^*) - q(\bar{x}, \bar{x}^*) = 0$$

Therefore $\forall t \in \mathbb{N}$ we have

$$\sum_{\alpha,\beta \in \mathbb{N}^m} R_{\alpha,\beta}(t, \bar{\lambda}^t, \bar{\lambda}^{-t})(\bar{x})^\alpha (\bar{x}^*)^\beta = 0$$

which implies that $\forall \alpha, \beta \in \mathbb{N}^m$ $R_{\alpha,\beta} \in \mathcal{L} = \langle L \rangle_{\mathbb{Q}[s,\bar{u},\bar{v}]}$. Thus

$$q(\bar{x}, \bar{x}^*) - q(F_i(s, \bar{u}, \bar{v}, F_i(-s, \bar{v}, \bar{u}, \bar{x})), \bar{x}^*) \in \langle L \rangle_{\mathbb{Q}[s,\bar{u},\bar{v},\bar{x},\bar{x}^*]}$$

□

Let $P_\infty = \phi(R_\infty)$. The following result intuitively means that we do not lose invariant polynomials in our approximation and so maintain completeness:

Proposition 3. *$P_\infty \subseteq \langle S\rangle_{\mathbb{Q}[\bar{x},\bar{x}^*]}$ is invariant in the implementation.*

Proof. Since by construction $\langle S_0\rangle_{\mathbb{Q}[\bar{x}^*]} = \mathbb{IV}(\langle S_0\rangle_{\mathbb{Q}[\bar{x}^*]})$, it can be proved that then $P_\infty \subseteq \langle\{x_1^* - x_1, ..., x_m^* - x_m\} \cup S_0\rangle_{\mathbb{Q}[\bar{x},\bar{x}^*]}$, and therefore the inclusion holds when entering the loop.

Now it remains to be seen that it is preserved at each iteration. From now on the $\langle\cdot\rangle$ means $\langle\cdot\rangle_{\mathbb{Q}[s,\bar{u},\bar{v},\bar{x},\bar{x}]}$. It suffices to see that

$$P_\infty \subseteq \left\langle L \cup \bigcap_{i=1}^{n} subs(F_i(-s,\bar{v},\bar{u},\cdot), P_\infty)\right\rangle$$

since by definition $P_\infty \subseteq \mathbb{Q}[\bar{x},\bar{x}^*]$. But

$$\left\langle L \cup \bigcap_{i=1}^{n} subs(F_i(-s,\bar{v},\bar{u},\cdot), P_\infty)\right\rangle$$

$$= \bigcap_{i=1}^{n} \langle L \cup subs(F_i(-s,\bar{v},\bar{u},\cdot), P_\infty)\rangle$$

So for $1 \leq i \leq n$ we have to see that

$$P_\infty \subseteq \langle L \cup subs(F_i(-s,\bar{v},\bar{u},\cdot), P_\infty)\rangle$$

Let $P = \{p_1, ..., p_k\} \in P_\infty$ be a Gröbner basis for P_∞. Let $q \in P_\infty$. We want to show that

$$q \in \langle L \cup subs(F_i(-s,\bar{v},\bar{u},\cdot), P_\infty)\rangle$$

First, let us see $q(F_i(s,\bar{u},\bar{v},\bar{x}),\bar{x}^*) \in \langle L \cup P_\infty\rangle$. If we divide $q(F_i(s,\bar{u},\bar{v},\bar{x}),\bar{x}^*)$ into $p_1, ..., p_k$ with respect to $\succ$, we get $R, Q_1, ..., Q_k \in \mathbb{Q}[s,\bar{u},\bar{v},\bar{x},\bar{x}^*]$ such that

$$q(F_i(s,\bar{u},\bar{v},\bar{x}),\bar{x}^*) = R + \sum_{j=1}^{k} Q_j p_j$$

We want to show that $R \in \langle L\rangle$.

Since $q \in P_\infty$, it is easy to see that $\forall t \in \mathbb{N}$

$$q(F_i(t,\bar{\lambda}^t,\bar{\lambda}^{-t},\bar{x}),\bar{x}^*) = q(f_i^t(\bar{x}),\bar{x}^*) \in P_\infty$$

But the remainder obtained when dividing $q(f_i^t(\bar{x}),\bar{x}^*) \in P_\infty$ into $p_1, ..., p_k$ is 0. As $p_1, ..., p_k$ is a Gröbner base, it can be proved that $\forall t \in \mathbb{N}$

$$R(t,\bar{\lambda}^t,\bar{\lambda}^{-t},\bar{x},\bar{x}^*) = 0$$

Now if we write $R(s,\bar{u},\bar{v},\bar{x},\bar{x}^*) = \sum_{\lambda,\mu\in\mathbb{N}^m} R_{\lambda,\mu}(s,\bar{u},\bar{v})\bar{x}^\lambda(\bar{x}^*)^\mu$ (only a finite number of the $R_{\lambda,\mu}$ are different from 0), we have that $\forall t \in \mathbb{N}$

$$0 = R(t,\bar{\lambda}^t,\bar{\lambda}^{-t},\bar{x},\bar{x}^*) = \sum_{\lambda,\mu\in\mathbb{N}^m} R_{\lambda,\mu}(t,\bar{\lambda}^t,\bar{\lambda}^{-t})\bar{x}^\lambda(\bar{x}^*)^\mu$$

So $\forall \lambda, \mu \in \mathbb{N}^m$ and $\forall t \in \mathbb{N}$ we get that $R_{\lambda,\mu}(t, \bar{\lambda}^t, \bar{\lambda}^{-t}) = 0$, i.e $R_{\lambda,\mu} \in \mathcal{L} = \langle L \rangle_{\mathbb{Q}[s,\bar{u},\bar{v}]}$. Thus $R \in \langle L \rangle$ and $q(F_i(s, \bar{u}, \bar{v}, \bar{x}), \bar{x}^*) \in \langle L \cup P_\infty \rangle$.

Since $q(F_i(s, \bar{u}, \bar{v}, \bar{x}), \bar{x}^*) \in \langle L \cup P_\infty \rangle$ and $L \subset \mathbb{Q}[s, \bar{u}, \bar{v}]$, substituting $\bar{x}$ by $F_i(-s, \bar{v}, \bar{u}, \bar{x})$ we have that

$$q(F_i(s, \bar{u}, \bar{v}, F_i(-s, \bar{v}, \bar{u}, \bar{x})), \bar{x}) \in$$

$$\in \langle L \cup subs(F_i(-s, \bar{v}, \bar{u}, \cdot), P_\infty) \rangle$$

From Lemma 9 we know that

$$q(\bar{x}, \bar{x}^*) - q(F_i(s, \bar{u}, \bar{v}, F_i(-s, \bar{v}, \bar{u}, \bar{x})), \bar{x}^*) \in \langle L \rangle$$

So $q(\bar{x}, \bar{x}^*) \in \langle L \cup subs(F_i(-s, \bar{v}, \bar{u}, \cdot), P_\infty) \rangle$, which is what we wanted to see. □

Finally, the last theorem implies trivially that the implementation is correct and complete:

Theorem 4. *If the Polynomial Procedure terminates with output I^*, the implementation terminates in at most the same number of iterations and with output S^* such that $\langle S^* \rangle_{\mathbb{Q}[\bar{x},\bar{x}^*]} = I^*$.*

Proof. Let us denote by $\Im_N$ the ideal stored in the variable I at the header of the loop at the end of the N-th iteration in the Polynomial Procedure; and analogously let $\mathfrak{S}_N$ be the ideal generated by the set of polynomials stored in the variable S at the end of the N-th iteration in the implementation.

First of all we will prove that $\forall N \in \mathbb{N}$ we have $\mathfrak{S}_N \subseteq \Im_N$. Then the termination of the Polynomial Procedure will imply a chain of equalities that will yield the proposition.

So let us prove that $\forall N \in \mathbb{N}$, $\mathfrak{S}_N \subseteq \Im_N$ by induction on N. If $N = 0$ there is nothing to prove since by definition $\langle S_0 \rangle_{\mathbb{Q}[\bar{x}^*]} = I_0$ and so $\mathfrak{S}_0 = \Im_0$.

If $N > 0$, we have that

$$\Im_N = \bigcap_{t=0}^{\infty} \bigcap_{i=1}^{n} subs(f_i^{-t}, \Im_{N-1})$$

$$\mathfrak{S}_N = \mathbb{Q}[\bar{x}, \bar{x}^*] \cap \left\langle L \cup \left(\bigcap_{i=1}^{n} subs(F_i(-s, \bar{v}, \bar{u}, \cdot), \mathcal{I}_{N-1}) \right) \right\rangle_{\mathbb{Q}[s,\bar{u},\bar{v},\bar{x},\bar{x}^*]} =$$

$$= \mathbb{Q}[\bar{x}, \bar{x}^*] \cap \left(\bigcap_{i=1}^{n} \langle L \cup subs(F_i(-s, \bar{v}, \bar{u}, \cdot), \mathfrak{S}_{N-1}) \rangle \right)$$

Let $q \in \mathfrak{S}_N$. For $1 \leq i \leq n$ and $t \in \mathbb{N}$ we have to show that $q \in subs(f_i^{-t}, \Im_{N-1})$. By induction hypothesis, it is enough to see that $q \in subs(f_i^{-t}, \mathfrak{S}_{N-1})$.

Now, if $p_1, ..., p_l \in \mathbb{Q}[\bar{x}, \bar{x}^*]$ is a base for $\mathfrak{S}_{N-1}$ and $L = \{u_1 v_1 - 1, ..., u_k v_k - 1\}$ as defined in Section 4.1, there exist polynomials $P_r, L_j \in \mathbb{Q}[s, \bar{u}, \bar{v}, \bar{x}, \bar{x}^*]$ for $1 \leq r \leq l$ and $1 \leq j \leq k$ such that

$$q(\bar{x}, \bar{x}^*) = \sum_{r=1}^{l} P_r(s, \bar{u}, \bar{v}, \bar{x}, \bar{x}^*)\ p_r(F_i(-s, \bar{v}, \bar{u}, \bar{x}), \bar{x}^*) +$$

$$+\sum_{j=1}^{k} L_j(s,\bar{u},\bar{v},\bar{x},\bar{x}^*)\ (u_j v_j - 1)$$

By taking an arbitrary $t \in \mathbb{N}$ and evaluating conveniently the auxiliar variables we have that

$$q(\bar{x},\bar{x}^*) = \sum_{r=1}^{l} P_r(t,\bar{\lambda}^t,\bar{\lambda}^{-t},\bar{x},\bar{x}^*)\ p_r(F_i(-t,\bar{\lambda}^{-t},\bar{\lambda}^t,\bar{x}),\bar{x}^*) +$$

$$+\sum_{j=1}^{k} L_j(t,\bar{\lambda}^t,\bar{\lambda}^{-t},\bar{x},\bar{x}^*)\ (\lambda_j^t \cdot \lambda_j^{-t} - 1) =$$

$$= \sum_{r=1}^{l} P_r(t,\bar{\lambda}^t,\bar{\lambda}^{-t},\bar{x},\bar{x}^*)\ p_r(f_i^{-t}(\bar{x}),\bar{x}^*)$$

So $q(\bar{x},\bar{x}^*) \in subs(f_i^{-t},\mathfrak{S}_{N-1})$ indeed. Therefore $\forall N \in \mathbb{N}\ \mathfrak{S}_N \subseteq \Im_N$.

Now if the Polynomial Procedure terminates in N iterations, then $I^* = \Im_N = \Im_{N-1}$ for a certain $N \geq 1$, and we have that $\Im_{N-1} = P_\infty$. Then by Proposition 3 $\mathfrak{S}_{N-1} \subseteq \Im_{N-1} = P_\infty \subseteq \mathfrak{S}_N$. But it is clear that $\mathfrak{S}_N \subseteq \mathfrak{S}_{N-1}$. So $\mathfrak{S}_{N-1} = \mathfrak{S}_N = P_\infty = S^*$, and the implementation terminates in at most the same number of iterations as the Polynomial Procedure.

□