Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations [*] [†]

Enric Rodríguez-Carbonell
LSI Department
Technical University of Catalonia
Barcelona, Spain
erodri@lsi.upc.es

Deepak Kapur
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico, USA
kapur@cs.unm.edu

ABSTRACT

This paper presents the algebraic foundation for an approach for generating polynomial loop invariants in imperative programs. It is first shown that the set of polynomials serving as loop invariants has the algebraic structure of an ideal. Using this connection, a procedure for finding loop invariants is given in terms of operations on ideals, for which Gröbner basis constructions can be employed. Most importantly, it is proved that if the assignment statements in a loop are solvable (in particular, affine) mappings with positive eigenvalues, then the procedure terminates in at most $2m+1$ iterations, where m is the number of variables in the loop. The proof is done by showing that the irreducible subvarieties of the variety associated with the polynomial ideal approximating the invariant polynomial ideal of the loop either stay the same or increase their dimension in every iteration. This yields a *correct* and *complete* algorithm for inferring conjunctions of polynomial equations as invariants. The method has been implemented in Maple using the `Groebner` package. The implementation has been used to automatically discover nontrivial invariants for several examples to illustrate the power of the techniques.

Categories and Subject Descriptors: F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs – Invariants; D.2.4 [Software Engineering]: Software / Program Verification – Formal methods; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages – Program analysis.

General Terms: Algorithms, Verification.

Keywords: Gröbner basis, ideal, loop invariant.

1. INTRODUCTION

Program verification based on Floyd-Hoare-Dijkstra's inductive assertion method, using pre/postconditions and loop invariants, was considered a major research problem in the seventies, leading to the development of many program verification systems. However, limited progress was made in achieving the goal of mechanical verification of properties of programs because: (i) theorem provers, needed to establish the validity of verification conditions, were not powerful enough; and (ii), the user had to manually annotate programs with loop invariants, because the few existing tools [10] for this purpose at that moment were not sufficiently effective.

Nonetheless, for life-critical applications it is still imperative to verify properties of programs [11]. With substantial progress in automated reasoning, several techniques for verification have emerged in the form of static analysis of programs (type checking, type inference, extended static checking, etc.), model checking as well as verifying properties of software and hardware using theorem proving techniques. However, the annotation burden remains. Our work attempts to deal with the problem of automatically generating loop invariants, which is still unsolved.

In [17], an abstract framework for finding invariants of simple loops with assignments and conditional statements was presented. Properties of the language used for expressing invariants were identified so that a generic correct and complete procedure for computing loop invariants could be formulated. This framework was then instantiated for the language of conjunctions of polynomial equations.

In this companion paper, we provide the algebraic foundation for the approach when invariants are expressed as conjunction of polynomial equations. It is shown that for a given loop, the set of polynomials p such that $p=0$ is invariant, i.e. p evaluates to 0 at the header whenever the loop is executed, is a polynomial ideal; this ideal is henceforth called the *invariant polynomial ideal* of the loop (this is very satisfying as it establishes a connection between the concept of invariant as used in programming languages and the concept of invariant as used in algebraic geometry). Any conjunction of polynomial equations such that the polynomials are a basis of this ideal is shown to be *inductive*, i.e. it holds when entering the loop and is preserved by every iteration of the loop. Moreover, such formula is the strongest among all the inductive invariants of the loop when invariants are conjunctions of polynomial equations. Using Hilbert's basis theorem, we also establish the existence of such an inductive invariant for a given loop.

One of the main results in this paper is a proof of termination of a procedure for discovering such invariants in the case of loops with solvable assignments (in particular, affine) that have positive eigenvalues. It is shown that the invariant ideal is computed in at

[*] 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, an FPU grant from the Spanish Secretaría de Estado de Educación y Universidades, ref. AP2002-3693, and the Spanish project MCYT TIC2001-2476-C03-01.

[†] An extended version of this paper is available at the website `www.lsi.upc.es/~erodri`.

ISSAC'04, July 4–7, 2004, Santander, Spain.

most $2m+1$ iterations, where m is the number of variables in the loop. The termination proof uses techniques from algebraic geometry to analyze the variety associated with the ideal approximating, at every iteration of this procedure, the invariant polynomial ideal of the loop. It is shown that either the invariant polynomial ideal is already computed, or the minimum dimension of the noninvariant irreducible components of the variety increases. The procedure is shown to be correct and complete. If a loop does not have any polynomial invariant, the procedure will generate the polynomial 0 (which is equivalent to *true*) as the invariant.

If assignment mappings commute, it can also be shown that the procedure for discovering invariants terminates in at most $n+1$ iterations, where n is the number of branches in the body of the loop. In particular, if there are no conditionals in the body of the loop, the procedure terminates in at most 2 iterations.

The procedure for discovering invariants has been implemented in Maple using the `Groebner` package for manipulating ideals (see [6] as an introduction to Gröbner basis algorithm, commutative algebra and algebraic geometry). The procedure has been successfully applied on several nontrivial imperative loop programs to automatically generate conjunctions of polynomial equations as loop invariants. Some of these examples are used in this paper to illustrate key concepts of the proposed approach. In Section 6, a table is given providing information on other examples successfully attempted using the procedure.

The rest of the paper is organized as follows. After reviewing related work in the next subsection, in Section 2 we introduce the notation and describe the kind of loops and the domain of variables we are going to consider. From Section 3 onwards, we focus on conjunctions of polynomial equations as loop invariants. It is shown that the set of invariant polynomials of a loop has the algebraic structure of an ideal, which immediately suggests that polynomial ideal theory and algebraic geometry can give insight into the problem of finding loop invariants. Section 4 presents the procedure for finding polynomial invariants, expressed in terms of ideals. The key ideas used in its proof of termination are discussed. In Section 5 we show how to implement this procedure using Gröbner bases, and in Section 6 we demonstrate the power of the method with a table of examples. Finally, Section 7 concludes with a summary of the contributions of the paper and an overview of future research.

1.1 Related Work

Automatic discovery of invariants was an active research topic in the early 70's, but interest in it dwindled in the 80's. Now, it seems to have caught on again.

Our techniques build upon the *difference equations method* [8, 14], which proceeds in two steps: i) 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; and ii), the variable s is eliminated to obtain invariant relations. The main obstacles in mechanizing this approach were: i) finding a generic expression for the values of variables in loops with conditionals or nested loops was not normally possible; and ii), eliminating loop counters had to be done by hand. For polynomial invariants, we show that quantifier elimination and elimination theory can be quite helpful.

Karr [13] gave an algorithm for finding linear equalities as loop invariants. This work was extended by Cousot and Halbwachs [5], who applied the model of abstract interpretation [4] for finding invariant linear inequalities using widening operators. Like our techniques, both of these papers were based on forward propagation and fixed point computation (see [21]). Recently, Colón et al [3] have applied nonlinear constraint solving for discovering linear inequalities without having to use widening operators and/or fixed point computation.

Karr's approach has been recently extended by Müller-Olm and Seidl [16] for finding polynomial equalities of bounded degree as invariants of programs with affine assignments. They used backward propagation and weakest preconditions, instead of forward propagation and strongest postconditions.

During the course of this research, we learned in November 2003 about [18], in which the authors proposed a method for generating nonlinear polynomial loop invariants. The method starts with a template polynomial with undetermined coefficients and attempts to find values for the coefficients using the Gröbner basis algorithm so that the instantiated template is invariant. A similar approach was presented by Kapur in a colloquium at the Department of Computer Science, University of New Mexico, in November 2003 (see `www.cs.unm.edu/colloquia/index.html`).

Our method, in contrast to [16, 18], does not have to assume a priori any structure (such as bounded degree) on the polynomials serving as invariants. All the examples shown in [18] can be handled using our method, which is complete for the class of loops considered in this paper, whereas [18] have to employ lots of heuristics to solve the examples in their paper. Moreover, we are able to handle solvable mappings as assignments, which generalize affine mappings. Finally, our method is already implemented.

2. BACKGROUND: PROGRAMMING MODEL AND DEFINITIONS

For clarity, a simple programming language consisting of multiple assignment statements, nondeterministic conditional statements and loop constructs is assumed. Using the proposed approach, it is possible to handle loop and conditional tests expressed as conjunction of polynomial equations; however, we assume them to be *true* to keep the presentation clean (in practice, this loss of information does not affect the results in the examples we deal with).

Let $\bar{x} = (x_1, x_2, ..., x_m)$ be the tuple of program variables of the loop. Loops are assumed to have the following structure, with a nondeterministic conditional statement with n branches:

```
while true do
    if true → $\bar{x} := f_1(\bar{x})$;
    ...
    [] true → $\bar{x} := f_i(\bar{x})$;
    ...
    [] true → $\bar{x} := f_n(\bar{x})$;
    end if
end while
```

where $f_i : \mathbb{Q}^m \to \mathbb{Q}^m$ for $1 \le i \le n$ to reflect that an assignment statement is, in general, a multiple assignment possibly changing all the variables. More precisely, the allowed assignment mappings f_i's are *solvable* mappings as defined below, for which the values of the x_i's at any iteration can be expressed as polynomials in the initial values of the variables when the loop is entered, using additional variables to represent exponentials (e.g. 2^n to deal with assignments such as $x := 2x$); affine mappings, that is to say, polynomial mappings of degree 1, are a particular case of solvable mappings.

To analyze execution paths of a given loop with n branches, consider the set of all finite strings over the alphabet $[n] = \{1, ..., n\}$. Let (f) stand for the tuple $f_1, ..., f_n$. Corresponding to every string σ, we inductively define $(f)^\sigma$ as

$$(f)^\lambda(\bar{x}) = \bar{x}, \quad (f)^{\sigma.k}(\bar{x}) = f_k((f)^\sigma(\bar{x})), 1 \le k \le n,$$

where λ denotes the empty string. Each σ represents an execution path, and $(f)^{\sigma}$ maps initial values of the variables to the values after executing the path σ.

2.1 Definitions

Given a field $\mathbb{K}$, we denote by $\mathbb{K}[\bar{z}] = \mathbb{K}[z_1,...,z_l]$ the ring of polynomials in the variables $z_1,...,z_l$ with coefficients from $\mathbb{K}$. An ideal is a set $I \subseteq \mathbb{K}[\bar{z}]$ that contains 0, is closed under addition and such that, if $p \in \mathbb{K}[\bar{z}]$ and $q \in I$, then $pq \in I$. Given a set of polynomials $S \subseteq \mathbb{K}[\bar{z}]$, the ideal spanned by S is

$$\{p \in \mathbb{K}[\bar{z}] \mid \exists k \geq 1 \;\; p = \sum_{j=1}^{k} p_j q_j \text{ with } p_j \in \mathbb{K}[\bar{z}], q_j \in S\}.$$

We denote it by $\langle S \rangle$. The *variety* of S over $\mathbb{K}^l$ is defined as its set of zeroes, $\mathbf{V}(S) = \{\bar{\alpha} \in \mathbb{K}^l \mid p(\bar{\alpha}) = 0 \; \forall p \in S\}$. For an ideal $I \subseteq \mathbb{K}[\bar{z}]$, a set $S \subseteq \mathbb{K}[\bar{z}]$ such that $I = \langle S \rangle$ is called a *basis* of I. Finally, if $A \subseteq \mathbb{K}^l$ the ideal $\mathbf{I}(A) = \{p \in \mathbb{K}[\bar{z}] \mid p(\bar{\alpha}) = 0 \; \forall \bar{\alpha} \in A\}$ is the *annihilator* of A. If $S \subseteq \mathbb{K}[\bar{z}]$, we will write $\mathbf{IV}(S)$ instead of $\mathbf{I}(\mathbf{V}(S))$.

A mapping $g : \mathbb{K}^m \to \mathbb{K}^m$ is called *affine* if it is of the form $g(\bar{x}) = A\bar{x} + b$, where A is an $m \times m$ matrix with coefficients in $\mathbb{K}$, and $b \in \mathbb{K}^m$. Thus, each variable is replaced in an affine assignment by a linear polynomial.

A polynomial mapping $g = (g_1,...,g_m) \in \mathbb{K}[\bar{z},\bar{x}]^m$, where $\bar{z}$ are auxiliary variables, generalizes an affine mapping. For given g and $S \subseteq \mathbb{K}[\bar{x},\bar{x}^*]$, $subs(g,S)$ is defined as the ideal

$$\langle \{p(g(\bar{z},\bar{x}),\bar{x}^*) \in \mathbb{K}[\bar{z},\bar{x},\bar{x}^*] \mid p(\bar{x},\bar{x}^*) \in S\} \rangle$$

obtained by replacing each variable x_i in p by the polynomial g_i in g. In particular, if B is a basis of an ideal I, then $subs(g,B) = subs(g,I)$.

A polynomial mapping $g \in \mathbb{K}[\bar{x}]^m$ is *polynomially invertible* if there exists $g' \in \mathbb{K}[\bar{x}]^m$ polynomial mapping such that $g'(g(\bar{x})) = \bar{x}$; we will denote it by g^{-1}.

As discussed later, the termination of the procedure proposed in Section 4 for generating polynomial invariants can be established only for those invertible polynomial mappings that are *solvable* and such that the eigenvalues of their transformation matrices (which is A in the case of an affine mapping $A\bar{x} + b$) are positive reals. Intuitively, a solvable mapping g is a polynomial mapping such that the recurrence $\bar{x}_{s+1} = g(\bar{x}_s)$ can be *solved* effectively and its solution (which is given by the general power g^s) can be expressed as a polynomial in $\bar{x}$, s and some additional variables needed to deal with exponentials. We formally define solvable mappings in Section 4.1 before discussing the proof of termination of the proposed procedure.

3. INVARIANT POLYNOMIALS FORM AN IDEAL

An invariant is a formula that is true at the beginning of the loop whenever it is executed. In order to deal with variables that are initialized to unknown parameters rather than to values in $\mathbb{Q}$ or even uninitialized, an invariant may depend not only on variables $\bar{x}$ representing the values of the variables at any iteration at the header of the loop, but also on variables $\bar{x}^*$ standing for the initial values before entering the loop. However, if any of the variables is initialized to some specific value or a relation among initial values is given, this information can also be used for discovering invariants. In the examples discussed in the paper, unknowns $\bar{x}^*$ are assumed as the initial values of $\bar{x}$; we also use rational initial values for the variables (or any relation on initial values) whenever it is possible. After general expressions for the invariants are discovered, any other known initial values for the variables can be substituted in the invariants.

We only consider polynomial equations (strictly speaking, conjunctions of polynomial equations) as invariants. We will often abuse the notation by writing a polynomial equation $p = 0$ as the polynomial p. Any relation among the initial values of variables is assumed to be expressed as conjunction of polynomial equations as well.

DEFINITION 1. *Given* $I_0 \subseteq \mathbb{Q}[\bar{x}^*]$, $p \in \mathbb{Q}[\bar{x},\bar{x}^*]$ *is* invariant *with respect to* I_0 *if* $\forall\sigma \; \forall\bar{\alpha}^* \in \mathbf{V}(I_0)$, $p((f)^{\sigma}(\bar{\alpha}^*),\bar{\alpha}^*) = 0$.

If a polynomial p is invariant with respect to I_0, then it is invariant with respect to $\langle I_0 \rangle$, as $\mathbf{V}(I_0) = \mathbf{V}(\langle I_0 \rangle)$. So I_0 can be assumed to be an ideal.

The following result states that the set of all invariant polynomials, which we denote by P_∞, is an ideal:

THEOREM 1. *Given an ideal* $I_0 \subseteq \mathbb{Q}[\bar{x}^*]$,

$$P_\infty = \bigcap_{\sigma} \{p \in \mathbb{Q}[\bar{x},\bar{x}^*] \mid \forall\bar{\alpha}^* \in \mathbf{V}(I_0), p((f)^{\sigma}(\bar{\alpha}^*),\bar{\alpha}^*) = 0\}$$

is an ideal.

We will refer to this ideal as the *invariant polynomial ideal* of the loop. By Hilbert's basis theorem, the ideal P_∞ above has a finite basis. Any finite basis of P_∞, say B, can be used to construct a formula $\bigwedge_{p \in B} p = 0$. This formula is proved to be the strongest invariant expressible as a conjunction of polynomial equations for the above loop.

4. INVARIANT GENERATION PROCEDURE

We discuss below a procedure for computing the invariant polynomial ideal P_∞ for a given loop. Since the procedure is based on the forward semantics of the assignment statement, it is necessary to require each assignment mapping f_i to be not only a polynomial mapping but also polynomially invertible. Later, for showing the termination of the procedure, it is further required that f_i be *solvable* as defined below, with its associated transformation matrices having positive real eigenvalues. The ideal I_0 relating initial values of variables is assumed to have the property that $I_0 = \mathbf{IV}(I_0)$. Recall that in the procedure below, f_i is a m-tuple of polynomials corresponding to m simultaneous assignments.

Invariant Generation Procedure

```
Input: Invertible mappings $f_1,..., f_n$ of assignments
       An ideal $I_0$ of polynomials satisfied by
       the initial values such that $I_0 = \mathbf{IV}(I_0)$
Output: Invariant ideal $P_\infty$

var $I', I$ : ideals in $\mathbb{Q}[\bar{x},\bar{x}^*]$
$I' := \mathbb{Q}[\bar{x},\bar{x}^*]$
$I := \langle \{x_1 - x_1^*, ..., x_m - x_m^*\} \cup I_0 \rangle$
while $I' \neq I$ do
    $I' := I$
    $I := \cap_{s=0}^{\infty} \cap_{i=1}^{n} subs(f_i^{-s}, I)$
end while
return $I$
```

In [17], an abstract procedure for generating loop invariants is given in terms of formulas in first-order predicate calculus. After instantiation for the case of conjunctions of polynomial equations, this abstract procedure yields the above procedure. The following theorem is a direct consequence of the results in that paper and ensures that on termination, the output, i.e. the ideal stored in the variable I, is **correct**, in the sense that all polynomials contained in it are invariant for the loop, and **complete**, in the sense that it contains the whole invariant polynomial ideal.

THEOREM 2. *If the Invariant Generation Procedure terminates,* $I = P_\infty$.

4.1 Termination of the Procedure

The main result of the paper is that the above procedure terminates if assignment mappings are *solvable* and the eigenvalues of the associated transformation matrices are positive real numbers. These conditions are met for most of the practical programs we have dealt with so far.

THEOREM 3. *If each assignment mapping f_i in a loop is solvable and the associated transformation matrices have positive real eigenvalues, then the Invariant Generation Procedure terminates in at most $2m+1$ iterations, where m is the number of variables in the loop.*

Due to lack of space, the proof of the theorem is not included below; we informally review the main idea and state the key steps. A complete proof is given in the long version of the paper. First we define solvable mappings, which generalize affine mappings.

4.1.1 Solvable Mappings

A solvable mapping is defined recursively by partitioning the variables $\bar{x}$ into a sequence of subsets $\bar{w}_1, \ldots, \bar{w}_k$ with the mapping $g_{\bar{w}_1} = (g_i)_{x_i \in \bar{w}_1}$ for the bottom subset defined to be affine and serving as the basis. The mapping $g_{\bar{w}_j} = (g_i)_{x_i \in \bar{w}_j}$ for a subset of variables higher in the sequence is defined only in terms of lower subsets.

DEFINITION 2. *Given a polynomial mapping $g \in \mathbb{Q}[\bar{x}]^m$, g is* solvable *if there exists a partition of $\bar{x}$, $\bar{x} = \bar{w}_1 \cup \cdots \cup \bar{w}_k$, $\bar{w}_i \cap \bar{w}_j = \emptyset$ if $i \neq j$, such that $\forall j : 1 \leq j \leq k$ we have*

$$g_{\bar{w}_j}(\bar{x}) = M_j \bar{w}_j^T + P_j(\bar{w}_1, \ldots, \bar{w}_{j-1})$$

where $M_j \in \mathbb{Q}^{|\bar{w}_j| \times |\bar{w}_j|}$ is a matrix and P_j is a vector of $|\bar{w}_j|$ polynomials in $\mathbb{Q}[\bar{w}_1, \ldots, \bar{w}_{j-1}]$. For $j = 1$, P_1 must be a constant vector, implying that $g_{\bar{w}_1}$ is an affine mapping.

The eigenvalues *of g are defined as the union of the eigenvalues of the matrices M_j, $1 \leq j \leq k$.*

Note that any affine mapping $g(\bar{x}) = A\bar{x} + b$ is solvable, since $\bar{w}_1 = \bar{x}$, $M_1 = A$, $P_1 = b$; the eigenvalues of g are the eigenvalues of A. For example, the assignments in the following loop, which is an abstraction of a program for computing the product of two natural numbers:

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

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

The nonlinear mapping $g(a,b,p,q) = (a-1, b, p, q+bp)$ (which appears in the second example in Section 5.1) is solvable, even though it is not affine because of the 4th component. Take $\bar{w}_1 = (a,b,p)$, $M_1 = diagonal(1,1,1)$, $P_1 = (-1,0,0)$. Then, $\bar{w}_2 = q$, with $M_2 = (1)$ and $P_2 = bp$. The eigenvalues of g are just $\{1\}$.

It can be shown that $g^s(a,b,p,q) = (a-s, b, p, q+bps)$. The reader should note that g^s can indeed be expressed as polynomials in a, b, p, q and in the auxiliary variable s.

For $g(x,y,z) = (2x, y/2, z)$, $g^s(x,y,z) = (2^s x, 1/2^s y, z)$. This can be expressed as a polynomial mapping using additional variables u, v standing for $2^s, 1/2^s$, respectively, giving $g^s(x,y,z) = (ux, vy, z)$ with $uv = 1$. This is discussed in detail in Section 5.

4.1.2 Outline of the Termination Proof

Let J_N be the ideal computed at the end of the N-th iteration of the Invariant Generation Procedure. The variety $\mathbf{V}(J_N)$ can be decomposed as the union of irreducible subvarieties.[1] The variety $\mathbf{V}(J_{N+1})$ associated with J_{N+1} at the end of the $(N+1)$-th iteration is related to $\mathbf{V}(J_N)$ by separately analyzing the effect of each assignment mapping f_i on each irreducible subvariety of $\mathbf{V}(J_N)$. To illustrate the key ideas below, consider the following loop:

```
$(x,y)$:=$(0,0)$;
while $true$ do
    if $true \rightarrow (x,y)$ :=$(x+1,y)$;
    [] $true \rightarrow (x,y)$ :=$(x,y+1)$;
    end if
end while
```

This toy program begins with the point $(0,0)$ and then repeteadly chooses nondeterministically to move either horizontally or vertically, covering all the pairs of natural numbers $\mathbb{N} \times \mathbb{N}$.

Applying the above procedure,

$$f_1(x,y) = (x+1, y) \quad \text{and} \quad f_2(x,y) = (x, y+1),$$

$$f_1^s(x,y) = (x+s, y) \quad \text{and} \quad f_2^s(x,y) = (x, y+s).$$

As both x and y are initialized to 0 before entering the loop, $I_0 = \langle x^*, y^* \rangle$. So $J_0 = \langle x^*, y^*, x - x^*, y - y^* \rangle = \langle x^*, y^*, x, y \rangle$ after simplifying the basis. Then

$$subs(f_1^{-s}, J_0) = \langle x^*, y^*, x-s, y \rangle$$

and

$$\cap_{s=0}^{\infty}\, subs(f_1^{-s}, J_0) = \langle x^*, y^*, y \rangle.$$

x can have any value after the first assignment has been executed arbitrarily many times. Similarly,

$$subs(f_2^{-s}, J_0) = \langle x^*, y^*, x, y-s \rangle,$$

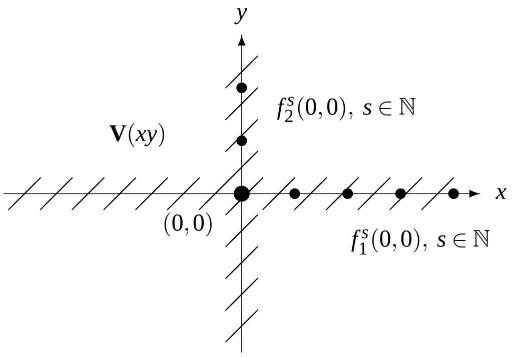
$$\cap_{s=0}^{\infty}\, subs(f_2^{-s}, J_0) = \langle x^*, y^*, x \rangle,$$

$$J_1 = (\cap_{s=0}^{\infty}\, subs(f_1^{-s}, J_0)) \cap (\cap_{s=0}^{\infty}\, subs(f_2^{-s}, J_0)) =$$

$$= \langle x^*, y^*, xy \rangle.$$

The dimension of the associated variety has gone up by 1. The projection of the variety on x and y, with the initial point and its successive images by f_1 and f_2, is given below.

[1] It is a well-known result in algebraic geometry that any variety V can be expressed in a unique way as a finite union of irreducible varieties V_i such that $V_i \not\subset V_j$ for $i \neq j$ (i.e. they are irredundant) [6]. The varieties V_i's appearing in this unique decomposition are called the irreducible components of V.


y
x
$\mathbf{V}(xy)$
$f_2^s(0,0),\ s \in \mathbb{N}$
$(0,0)$
$f_1^s(0,0),\ s \in \mathbb{N}$

At the next step,

$$\cap_{s=0}^{\infty} subs(f_1^{-s}, J_1) = \cap_{s=0}^{\infty} subs(f_2^{-s}, J_1) = \langle x^*, y^* \rangle.$$

Thus

$$J_2 = (\cap_{s=0}^{\infty} subs(f_1^{-s}, J_1)) \cap (\cap_{s=0}^{\infty} subs(f_2^{-s}, J_1)) =$$

$$= \langle x^*, y^* \rangle$$

Again the dimension of the associated variety is gone up by 1. Since there are no more polynomials in the x, y variables anymore, the procedure terminates in the next iteration (in this case with the trivial invariant $0 = 0$). The projection of $\mathbf{V}(J_2)$ on x, y is the whole plane $\mathbb{R}^2$.

In this example, the dimension of the variety of the computed ideal increases at each step until getting the invariant polynomial ideal, where the variety and its dimension do not change. In general, at each step either the invariant ideal has been computed, or the minimum dimension of the noninvariant irreducible components of the variety increases. We now give the key arguments in the proof of termination.

1. It is shown that given a prime ideal J, its image by *subs* under an invertible polynomial mapping g is also a prime ideal.

   Using this proposition, it is proved that given an irreducible variety V (and its associated prime ideal J), the Zariski closure of the union of its iterated images, i.e. $\cup_{s \in \mathbb{N}} g^s(V)$ under a solvable mapping g is either V itself or another irreducible variety V' such that $\dim(V') > \dim(V)$, provided the transformation matrices associated with g have positive real eigenvalues. In the example, we have that the Zarisky closure of $\cup_{s \in \mathbb{N}} f_1^s(0,0)$ corresponds to the x-axis, which is irreducible and has one dimension more than $(0,0)$, the origin. Analogously, the Zarisky closure of $\cup_{s \in \mathbb{N}} f_2^s(0,0)$ is the y-axis, which again has dimension 1.

2. Each irreducible subvariety of $\mathbf{V}(J_N)$ thus transforms under such a solvable assignment mapping to the same irreducible subvariety in an irreducible decomposition of $\mathbf{V}(J_{N+1})$ or to an irreducible subvariety of a higher dimension. But we may not get an irredundant union of irreducible subvarieties of $\mathbf{V}(J_{N+1})$, as two distinct irreducible subvarieties of $\mathbf{V}(J_N)$ may transform to the same irreducible subvariety under g. In the second iteration of the example, it can be seen that f_1 transforms the y-axis into $\mathbb{R}^2$ and leaves the x-axis the same; and that f_2 transforms the x-axis into $\mathbb{R}^2$ as well, and leaves the y-axis as it was.

   Furthermore, since an iteration of the procedure captures the effect of all assignment mappings f_i's in the case the loop has many branches, an irreducible component of $\mathbf{V}(J_N)$ can map to a reducible subvariety in $\mathbf{V}(J_{N+1})$. But different irreducible subvarieties of $\mathbf{V}(J_N)$ map under distinct solvable mappings f_i's to irreducible subvarieties of $\mathbf{V}(J_{N+1})$. In the example we have that the origin $(0,0)$, which is irreducible, yields $\mathbf{V}(xy)$, which is the union of two straight lines and so is reducible; still, each f_i has given a different irreducible component of $\mathbf{V}(xy)$, namely each of the two coordinate axes.

3. If $\mathbf{V}(J_{N+1})$ is different from $\mathbf{V}(J_N)$, there is at least one irreducible subvariety of $\mathbf{V}(J_N)$ that is transformed to an irreducible subvariety of higher dimension in $\mathbf{V}(J_{N+1})$. Furthermore, irreducible subvarieties that become invariant under assignment mappings in any given iteration of the procedure will continue to remain invariant in subsequent iterations as well. Consequently, the dimension of all the noninvariant irreducible subvarieties keeps increasing. Therefore the minimum dimension of the irreducible subvarieties that do not remain invariant under assignment mappings keeps increasing. In our example, we would get the trace of dimensions 0-1-2.

4. Since there are $2m$ variables, the dimension can only increase from 0 to $2m$, thus implying that, after at most $2m+1$ iterations, J_N becomes invariant.

4.1.3 Commuting Solvable Mappings

If the assignment mappings f_i's are solvable and *commute*, i.e. $f_i \circ f_j = f_j \circ f_i$ for $1 \le i, j \le n$, it can be shown that the Invariant Generation Procedure terminates in at most $n+1$ iterations, where n is the number of branches in the body of the loop. In particular, if $n = 1$, or equivalently there are no conditional statements, the Invariant Generation Procedure takes at most 2 iterations to terminate.

The number of branches in a nontrivial loop is typically much less than the number of variables (including the case when there is only one branch in a loop, as in Example 3 in Section 5.1). Then, if assignment mappings commute, the termination is achieved in fewer iterations. The requirement on solvable mappings that their transformation matrices have positive eigenvalues can also be relaxed.

THEOREM 4. *If all pairs of invertible assignment mappings f_i and f_j commute, i.e. $f_i \circ f_j = f_j \circ f_i$, then the Invariant Generation Procedure terminates in at most $n+1$ iterations, where n is the number of branches in the loop.*

The proof of termination, which is totally orthogonal to the previous one, provides additional insight into J_N, the ideal computed after N iterations of the Invariant Generation Procedure. The proof is based on the following general observation: at the N-th iteration of the Invariant Generation Procedure, the ideal J_N generated is the intersection of the invariant polynomial ideals for all execution paths of the loop with branches alternated with each other at most $N-1$ times. Informally speaking, this amounts to capturing the effect of all possible compositions of assignment mappings with at most $N-1$ alternations.

Given that assignment mappings commute, all execution paths of a loop can be rearranged to have the same effect as that of all execution paths with at most $n-1$ alternations. Therefore after the $(n+1)$-th iteration the Invariant Generation Procedure will reach its fixed point, as $J_{n+1} = J_n$.

5. APPROXIMATING WITH GRÖBNER BASES

The Invariant Generation Procedure cannot be directly implemented because of

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

since infinite intersection of ideals cannot be effectively computed.

By treating s as a new variable, if a general expression for each f_i^{-s} for $1 \leq i \leq n$ can be computed as a polynomial mapping in $\bar{x}, s$, then s can be eliminated. Assuming that $f_i^{-s}(\bar{x}) \in \mathbb{Q}[s, \bar{x}]$ and given a basis for $I \subseteq \mathbb{Q}[\bar{x}, \bar{x}^*]$, we get a basis for the ideal $subs(f_i^{-s}, I) \subseteq \mathbb{Q}[s, \bar{x}, \bar{x}^*]$ by substituting the $\bar{x}$ variables by the expression $f_i^{-s}(\bar{x})$ in the polynomials in the basis of I. Then Gröbner bases methods can be used to compute the finite intersection $\cap_{i=1}^{n} subs(f_i^{-s}, I)$. From the ideal thus obtained we can compute an elimination ideal in $\bar{x}$ eliminating s, again using a Gröbner basis algorithm with a term ordering in which s is bigger than all other variables in $\bar{x}$. In other words, we eliminate s from $\cap_{i=1}^{n} subs(f_i^{-s}, I)$.

It is shown below that if f_i is a solvable mapping with rational eigenvalues, then f_i^{-s} can be expressed as a polynomial mapping in $\bar{x}, s$ and possibly some additional variables to substitute for exponentials of rationals. Consider the assignment mappings discussed in Section 4.1.1:

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

It is easy to see that

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

which is not a polynomial mapping in terms of x, y, z, s. However, if new auxiliary variables are introduced to replace $2^s, (1/2)^s$, say u, v, respectively, then

$$F_2(u,v,x,y,z) = f_2^s(x,y,z) = (ux, vy, z),$$

subject to the polynomial relation $uv = 1$. In a similar way, for $f_1(x,y,z) = (2x, y/2 - 1/2, x + z)$,

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

which can also be expressed as:

$$F_1(u,v,x,y,z) = f_1^s(x,y,z) = (ux, vy + v - 1, z + (u-1)x).$$

For both f_1 and f_2 the eigenvalues are $\{2, 1/2, 1\}$, which yield the exponential terms 2^s and $(1/2)^s$ in f_1^s, f_2^s; the variables u and v are introduced to substitute these exponentials so that f_1^s, f_2^s can be represented as polynomial mappings. In general, the following result enables to express powers of solvable mappings with rational eigenvalues as polynomial mappings using auxiliary variables similarly:

THEOREM 5. *Let $g \in \mathbb{Q}[\bar{x}]^m$ be a solvable mapping with rational eigenvalues. Then $g_j^s(\bar{x})$, the j-th component of $g^s(\bar{x})$ for $1 \leq j \leq m$, is*

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

where for $1 \leq j \leq m$, $1 \leq l \leq r_j$, $P_{jl} \in \mathbb{Q}[s, \bar{x}]$ and each $\gamma_{jl} \in \mathbb{Q}$ is a product of a subset of the eigenvalues of g.

The proof of the above theorem is based on two facts: first of all, a matrix $M \in \mathbb{Q}^{h \times h}$ with rational eigenvalues can be decomposed as $M = S^{-1}TS$, where $S, T \in \mathbb{Q}^{h \times h}$, $\det(S) \neq 0$ and T is the Jordan normal form of M; and secondly, a sequence $(\varphi_s)_{s \in \mathbb{N}}$ is of the form

$$\varphi_s = \sum_{l=1}^{r} P_l(s)(\gamma_l)^s, \qquad s \geq 0$$

with the P_l's polynomials for $1 \leq l \leq r$ if and only if its generating function $\Phi(z) = \sum_{s=0}^{\infty} \varphi_s z^s$ is a rational function (see [19] for an introduction to generating functions).

To represent g^s as a polynomial mapping, auxiliary variables are introduced to substitute for exponentials of eigenvalues (e.g. u, v for $2^s, (1/2)^s$, respectively, in the above example). These auxiliary variables are related to each other using relations (e.g. $uv = 1$). It can be shown that operations on ideals over the quotient structure of polynomial rings as induced by the relations on auxiliary variables have equivalent behaviour. For the above example, operations on ideals are done in the quotient structure induced by the ideal $\langle uv - 1 \rangle$ on $\mathbb{Q}[u, v, x, y, z]$ using a term ordering in which u, v are bigger than x, y, z.

5.1 Implementation

In the algorithm below, ideals are represented by their respective finite bases (often their Gröbner bases using some term ordering). Checking that the assignment mappings f_i's are solvable with positive rational eigenvalues can be done easily using linear algebra. Then the powers f_i^s's are computed and expressed as polynomial mappings denoted by F_i's, possibly employing the parameter s and additional auxiliary variables $\bar{u}, \bar{v}$ introduced to replace exponentials of eigenvalues and their inverses; relations among auxiliary variables are specified using a set of polynomials L, for instance $\{uv - 1\}$ in our running example, which can be computed automatically.

```
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 $\langle S_0 \rangle = \mathbf{IV}(\langle S_0 \rangle)$.
Output: A finite basis for the invariant ideal $P_\infty$

var $S', S$ : sets of polynomials in $\mathbb{Q}[\bar{x}, \bar{x}^*]$
    $S_{aux}$ : set of polynomials in $\mathbb{Q}[s, \bar{u}, \bar{v}, \bar{x}, \bar{x}^*]$
compute $f_1^s, ..., f_n^s, F_1, ..., F_n, L$
$S' := \{1\}$
$S :=$gbasis$(\{x_1 - x_1^*, ..., x_m - x_m^*\} \cup S_0, \succ)$
while $S' \neq S$ do
    $S' := S$
    $S_{aux} :=$gbasis$(\cap_{i=1}^{n} \langle$subs$(F_i(-s, \bar{v}, \bar{u}, \cdot), S)\rangle, >)$
    $S_{aux} :=$gbasis$(S_{aux} \cup L, \succ)$
    $S := \{$ polynomials in $S_{aux}$ without $s, \bar{u}, \bar{v}\}$
end while
return $S$
```

The function gbasis computes the reduced Gröbner basis of the input ideal, specified as a finite set of polynomials, with respect to a term ordering. The intersection of ideals is performed by using Gröbner bases methods. For eliminating the variables $s, \bar{u}, \bar{v}$, either block term ordering or lexicographic term ordering in which $s, \bar{u}, \bar{v}$ are the highest can be used. The equality test on ideals is implemented by comparing reduced Gröbner bases with respect to the same ordering, as every ideal has a unique reduced Gröbner basis once the ordering is fixed.

A variation of the above algorithm employing additional heuristics to speed up the computation has been implemented in Maple. The implementation has been successfully used to automatically discover invariants of many nontrivial programs. Some of these are discussed below as well as in the next section. As the reader will notice, the invariants of the examples below are not easy to deduce even by hand.

Example 1. Consider the loop introduced in Section 4.1.1. A Gröbner basis of the computed ideal as well as the dimension of the corresponding variety after every iteration are given below.

`iteration 0` $\longrightarrow \quad \{z^*, x-x^*, y-y^*, z-z^*\}$, dimension 2

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 $z^* = 0$.

`iteration 1` $\longrightarrow \quad \{z^*, xz-z-zx^*, -x^*y^*+z+xy, yz+zyx^*-zx^*y^*+z\}$, dimension 3

`iterations 2,3` $\longrightarrow \quad \{z^*, -x^*y^*+z+xy\}$, dimension 4

In 3 iterations, the algorithm terminates. The polynomial equation $z^* = 0$ is the equation satisfied by the initial values. Replacing x^* and y^* in $-x^*y^*+z+xy$ by their initial values X and Y, the invariant $-XY+z+xy = 0$, i.e. $z+xy = XY$ is obtained.

Example 2. Let us take the following loop, which is a simplification of a program for computing the product of two natural numbers (notice that the first assignment mapping was introduced in Section 4.1.1):

```
$(a,\ b,\ p,\ q)$:=$(A,\ B,\ 1,\ 0)$;
while $true$ do
    if $true \rightarrow (a,b,p,q) := (a-1,b,p,q+bp)$;
    [] $true \rightarrow (a,b,p,q) := (a/2,b/2,4p,q)$;
    end if
end while
```

The trace of the algorithm is as follows:

`iteration 0` $\longrightarrow \{q^*, p^*-1, a-a^*, b-b^*, p-p^*, q-q^*\}$, dimension 2

`iteration 1` $\longrightarrow \{q^*, p^*-1, pq-q, qb-qb^*, ba^*-b^*a-q, b^*aq-qa^*b^*+q^2, pb^2-b^{*2}, bpa-a^*b^*+q, a^2p^2-a^2p-pa^{*2}+a^{*2}, a^2pb^*-a^{*2}b^*+qa+qa^*\}$, dimension 3

`iterations 2,3` $\longrightarrow \{q^*, p^*-1, pb^2-b^{*2}, bpa-a^*b^*+q, ba^*b^*-b^{*2}a-qb, a^2pb^{*2}-a^{*2}b^{*2}+2qa^*b^*-q^2\}$, dimension 4

After substituting the initial values, the invariant obtained is:

$$pb^2 = B^2 \ \wedge \ bpa+q = AB \ \wedge$$

$$\wedge \ bAB = B^2a+qb \ \wedge \ a^2pB^2+2qAB = A^2B^2+q^2$$

Example 3. The following example is a program for computing the floor of the square root of a natural number:

```
$(a,\ s,\ t)$:=$(0,\ 1,\ 1)$;
while $s \leq N$ do
    $(a,\ s,\ t)$ :=$(a+1,\ s+t+2,\ t+2)$;
end while
```

In this case the trace is:

`iteration 0` $\longrightarrow \{t^*-1, s^*-1, a^*, a-a^*, s-s^*, t-t^*\}$, dimension 0

`iterations 1,2` $\longrightarrow \{t^*-1, s^*-1, a^*, 2a-t+1, a^2-s+2a+1\}$, dimension 1

The conjunction $t = 2a+1 \wedge s = a^2+2a+1$ is automatically generated as an invariant in 2 iterations.

6. EXAMPLES

The implementation has been used to find invariants for many programs. The table below gives a representative list of the examples attempted so far. There is a row for each program; the columns provide the following information:

- (GENERAL INFORMATION) 1st column is the name of the program; 2nd column states what the program does (the "toy" entry stands for a toy example); 3rd column gives the citation from where the program was picked (the entry $(*)$ is for the example developed up by the authors).
- (CHARACTERISTICS) 4th column is the number of variables in the loop; 5th column gives the number of branches in the body of the loop.
- (RESULTS) 6th column gives the number of polynomials in the invariant; 7th column gives the maximum degree of the invariant polynomials; 8th column gives the number of times the main loop of the Invariant Generation Procedure is executed; 9th column gives the time taken by the implementation running on a Pentium 4 2.5 GHz. processor with 512 Mb of memory. Except for the last program, the implementation took less than 6 seconds for all other examples in the table, indicating that it works quite fast.

1	2	3	4	5	6	7	8	9
freire1	$\sqrt[2]{\ }$	[9]	2	1	1	2	2	< 3 s.
freire2	$\sqrt[3]{\ }$	[9]	3	1	6	4	2	< 5 s.
cohencu	cube	[2]	4	1	4	2	2	< 5 s.
cousot	toy	[5]	2	2	0	-	4	< 4 s.
divbin	division	[12]	3	2	1	2	4	< 5 s.
dijkstra	$\sqrt[2]{\ }$	[7]	3	2	1	2	4	< 6 s.
fermat2	factor	[1]	3	2	1	2	4	< 4 s.
wensley2	division	[20]	4	2	3	2	4	< 5 s.
euclidex2	gcd	$(*)$	6	2	5	2	5	< 6 s.
lcm2	lcm	[7]	4	2	1	2	5	< 5 s.
factor	factor	[15]	4	4	1	3	7	< 20 s.

7. CONCLUSION

The main contributions of this paper are:

1. We prove that the set of invariant polynomials of a loop has the algebraic structure of an ideal. Moreover, for any finite basis of this ideal, the corresponding conjunction of polynomial equations is the strongest possible inductive invariant for the loop expressible as a conjunction of polynomial equations.
2. For solvable assignment mappings with positive real eigenvalues, we prove that the invariant polynomial ideal is computed in at most $2m+1$ steps, where m is the number of variables in the loop.
3. For solvable assignment mappings that commute, that is to say $f_i \circ f_j = f_j \circ f_i$ for $1 \leq i, j \leq n$, we show that the invariant polynomial ideal is computed in at most $n+1$ steps, where n is the number of branches in the loop.
4. We show how the procedure for finding the invariant polynomial ideal can be approximated using Gröbner bases. Moreover, for solvable mappings with rational positive eigenvalues, this approximation is exact, i.e. the algorithm computes the invariant ideal.
5. The algorithm has been implemented in Maple and successfully used to compute invariant conjunctions of polynomial equations for many nontrivial examples.

For future work, we are interested in exploring the proposed research along several directions: *i*) enrich the programming model to consider nested loops and procedure calls, as well as using guards in conditional statements and loops to discover invariants; *ii*) identify other languages that can specify properties of data structures such as arrays, records, pointers, etc. and to which the above techniques extend; *iii*) integrate this and other methods for mechanically inferring loop invariants, together with theorem proving components, into a powerful tool for verifying properties of programs.

Acknowledgements. We thank A. Chtcherba, G. Godoy, R. Nieuwenhuis and A. Oliveras for their help, advice and comments.

8. REFERENCES

[1] D. M. Bressoud. *Factorization and Primality Testing*. Springer-Verlag, 1989.

[2] E. Cohen. *Programming in the 1990s*. Springer-Verlag, 1990.

[3] M. A. Colón, S. Sankaranarayanan, and H. 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.

[4] 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.

[5] 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.

[6] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer-Verlag, 1998.

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

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

[9] P. Freire. `www.pedrofreire.com/crea2_en.htm?`

[10] S. German and B. Wegbreit. A Synthesizer of Inductive Assertions. *IEEE Transactions on Software Engineering*, 1(1):68–75, 1975.

[11] T. Hoare. The Verifying Compiler: A Grand Challenge for Computing Research. *Journal of the ACM*, 50(1):63–69, 2003.

[12] A. Kaldewaij. *Programming. The Derivation of Algorithms*. Prentice-Hall, 1990.

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

[14] S. Katz and Z. Manna. Logical Analysis of Programs. *Communications of the ACM*, 19(4):188–206, April 1976.

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

[16] M. Müller-Olm and H. Seidl. Computing Interprocedurally Valid Relations in Affine Programs. In *ACM SIGPLAN Principles of Programming Languages (POPL 2004)*, pages 330–341, 2004.

[17] E. Rodríguez-Carbonell and D. Kapur. Program Verification Using Automatic Generation of Polynomial Invariants. `www.lsi.upc.es/~erodri`.

[18] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Non-linear Loop Invariant Generation Using Gröbner Bases. In *ACM SIGPLAN Principles of Programming Languages (POPL 2004)*, pages 318–329, 2004.

[19] R. Stanley. *Enumerative Combinatorics*, volume 1. Cambridge University Press, 1997.

[20] B. Wegbreit. The Synthesis of Loop Predicates. *Communications of the ACM*, 17(2):102–112, February 1974.

[21] B. Wegbreit. Property Extraction in Well-founded Property Sets. *IEEE Transactions on Software Engineering*, 1(3):270–285, September 1975.