Polynomial Constants are Decidable [⋆]

Markus Müller-Olm[1] and Helmut Seidl[2]

[1] University of Dortmund, FB 4, LS5, 44221 Dortmund, Germany
mmo@ls5.cs.uni-dortmund.de
[2] Trier University, FB 4-Informatik, 54286 Trier, Germany
seidl@uni-trier.de

Abstract. Constant propagation aims at identifying expressions that always yield a unique constant value at run-time. It is well-known that constant propagation is undecidable for programs working on integers even if guards are ignored as in non-deterministic flow graphs. We show that *polynomial constants* are decidable in non-deterministic flow graphs. In polynomial constant propagation, assignment statements that use the operators $+, -, *$ are interpreted exactly but all assignments that use other operators are conservatively interpreted as non-deterministic assignments.
We present a generic algorithm for constant propagation via a symbolic weakest precondition computation and show how this generic algorithm can be instantiated for polynomial constant propagation by exploiting techniques from computable ring theory.

1 Introduction

Constant propagation is one of the most widely used optimizations in practical optimizing compilers (cf. [1, 9, 15]). Its goal is to replace expressions that always yield a unique constant value at run-time by this value. This both speeds up execution and reduces code size. Even more importantly, it can enable powerful further transformations like elimination of dynamically unreachable branches.

In order to come to grips with fundamental computability problems one often abstracts guarded branching to non-deterministic branching in program analysis. But even this abstraction leaves constant propagation undecidable for programs working on integer variables. This has already been observed in the seventies independently by Hecht [9] and by Reif and Lewis [17]. We briefly recall the construction of Reif and Lewis. It is based on a reduction of Hilbert's tenth problem, whether a (multivariate) polynomial has a zero in the natural numbers, a very famous undecidable problem [12].

Assume given a (non-zero) polynomial $p(x_1, \ldots, x_n)$ in n variables $x_1, \ldots, x_n$ with integer coefficients and consider the (non-deterministic) program in Fig. 1. The initializations and the loops choose arbitrary natural values for the variables x_i. If the chosen values constitute a zero of $p(x_1, \ldots, x_n)$, then $p(x_1, \ldots, x_n)^2 +$

[⋆] The work was supported by the RTD project IST-1999-20527 "DAEDALUS" of the European FP5 programme.

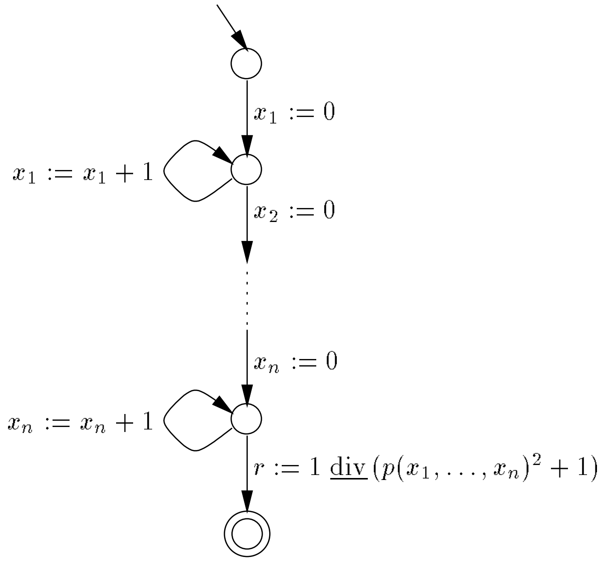

Fig. 1. Undecidability of constant detection; the reduction of Reif and Lewis [17].

$1 = 1$ and r is set to 1. Otherwise, $p(x_1, \ldots, x_n)^2 + 1 \geq 2$ such that r is set to 0. Therefore, r is a constant (of value 0) at the end of the program if and only if $p(x_1, \ldots, x_n)$ does not have a natural zero.

On the other hand there are well-known and well-defined classes of constants that can be detected, even efficiently. In *copy constant detection* [7] only assignments of the form $x := c$, where c is either an (integer) constant or a program variable are interpreted; assignments with composite expressions on the right hand side are conservatively assumed to make a variable non-constant. In *linear constants* [18] also assignments of the form $x := a \cdot y + b$, where a and b are integer constants and y is a program variable, are interpreted. Another decidable class of constants are *finite constants* [19]. This motivated Müller-Olm and Rüthing [16] to study the complexity of constant propagation for classes that derive from restricting the signature of interpreted integer operators.

An interesting question they left open concerns the class of constants obtained by interpreting just $+, -, *$, i.e., all standard integer operators except of the division operators. While they called the corresponding class of constants $+, -, *$-constants, we prefer the term *polynomial constants* here, as these signature allows just to write (multivariate) polynomials. The detection problem for polynomial constants is PSPACE-hard [16] but no upper bound is known. In the current paper we show that polynomial constants are decidable by applying results from computable ideal theory. This decidability result suggests that the division operator is the real cause for undecidability of general constant detection.

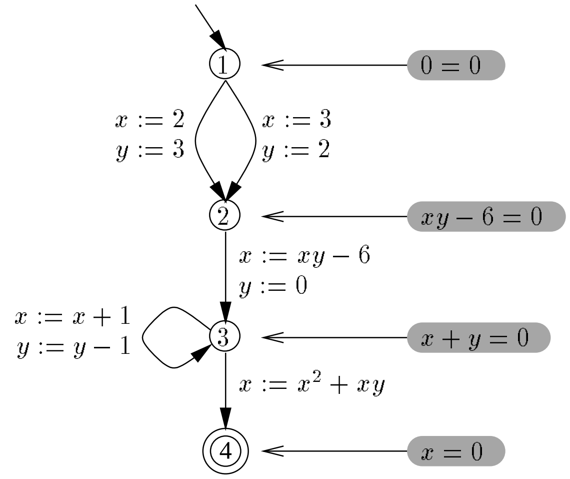

Fig. 2. An example flow graph.

The paper is organized as follows. In the next section we illustrate our algorithm at an example that cannot be handled by other constant propagation algorithms. Afterwards we define flow graphs, the constant propagation problem, and weakened versions thereof. This enables us to define polynomial constants formally. In Section 4 we present a generic algorithmic idea for constant propagation via a symbolic weakest precondition computation and discuss the requirements for making it effective. In Section 5 we recall results from computable ring theory that are needed in our algorithm and proceed with further observations on $\mathbb{Z}[x_1, \ldots, x_n]$, the set of multivariate polynomials with integer coefficients. These results are exploited in Section 7, where we show how the generic algorithm from Section 4 can be instantiated for polynomial constant propagation. We finish the paper with concluding remarks and directions for future research.

2 An Example

Let us illustrate the power of our algorithm. In the flow graph in Fig. 2, x is a constant (of value 0) at node 4, but no known constant propagation algorithms can handle this example. Standard *simple constant propagation* [1] propagates variable assignments through the program, in which each variable is assigned either a definite value or a special value unknown. Simple constant propagation computes the value unknown at program point 2 for both x and y and cannot recover from this loss of precision. More powerful constant propagation algorithms like the algorithm for *linear constants* [18] or Presburger constants [16] cannot handle the expressions $xy - 6$ and $x^2 + xy$. Even the EXPTIME algorithm of Knoop and Steffen for finite constants [19] cannot handle this example because no finite unfolding of the loop suffices to prove constancy of $x^2 + xy$ after the loop.

Before we turn to the technical development, we discuss informally how our algorithm detects constancy of x at node 4. In a first phase some path from node 1 to 4 is executed, e.g., $\langle 1, 2, 3, 4\rangle$, and the value of x, viz. 0, after execution of this path is computed. This implies that x can only be a constant of value 0 at program point 4 – if it is a constant at all. In order to check this, our algorithm propagates the assertion $\mathcal{A}_0 : x = 0$ backwards from node 4 towards the start node which amounts to a symbolic computation of the weakest precondition of $\mathcal{A}_0$ at node 4. Propagation over statement $x := x^2 + xy$ results in the assertion $\mathcal{A}_1 : x^2 + xy = 0$. Assertion $\mathcal{A}_1$ is then propagated through the loop. This results in the assertion $\mathcal{A}_2 : (x + 1)^2 + (x + 1)(y - 1) = 0$ that can be simplified to $\mathcal{A}_3 : x^2 + xy + x + y = 0$. Both $\mathcal{A}_1$ and $\mathcal{A}_3$ must be valid at program point 3 in order to guarantee validity of $\mathcal{A}_0$ at program point 4. We can simplify the assertion $\mathcal{A}_1 \wedge \mathcal{A}_3$: because $\mathcal{A}_1$ guarantees that $x^2 + xy$ equals 0, we can simplifiy $\mathcal{A}_3$ to $\mathcal{A}_4 : x + y = 0$; now, as $\mathcal{A}_1$ can be written in the form $x(x + y) = 0$, we see that $\mathcal{A}_1$ is indeed implied by $\mathcal{A}_4$. Thus, validity of $\mathcal{A}_4$ suffices to guarantee both $\mathcal{A}_1$ and $\mathcal{A}_3$. $\mathcal{A}_4$ is again propagated through the loop; this results in $\mathcal{A}_4$ again; hence no further propagation through the loop is necessary. In this way propagation goes on and results in the assertions shown in Fig. 2. The assertion computed for the start node, $0 = 0$ is universally valid; this proves that x is indeed a constant of value 0 at node 4.

In the algorithm developed in the remainder of this paper, assertions are represented by Gröbner bases of ideals in the polynomial ring $\mathbb{Z}[x_1, \ldots, x_n]$. As Gröbner bases are a *canonic* representation this also takes care of simplifications.

3 The Framework

Flow Graphs. Suppose given a finite set of variables $X = \{x_1, \ldots, x_n\}$. Let Expr be a set of expressions over X; the precise nature of expressions is immaterial at the moment. A (deterministic) assignment is a pair consisting of a variable and an expression written as $x := t$; the set of assignment statements is denoted by Asg. A non-deterministic assignment statement consists of a variable and is written $x :=?$; the set of nondeterministic assignment statements is denoted by NAsg.

A (non-deterministic) flow graph is a structure $G = (N, E, A, \mathbf{s}, \mathbf{e})$ with finite node set N, edge set $E \subseteq N \times N$, a unique start node $\mathbf{s} \in N$, and a unique end node $\mathbf{e} \in N$. We assume that each program point $u \in N$ lies on a path from $\mathbf{s}$ to $\mathbf{e}$. The mapping $A : E \to \mathsf{Asg} \cup \mathsf{NAsg} \cup \{\mathsf{skip}\}$ associates each edge with a deterministic or non-deterministic assignment statement or the statement skip. Edges represent the branching structure and the statements of a program, while nodes represent program points. The set of successors of program point $u \in N$ is $Succ[u] = \{v \mid (u, v) \in E\}$.

A *path* reaching a given program point $u \in N$ is a sequence of edges $p = \langle e_1, \ldots, e_k\rangle$ with $e_i = (u_i, v_i) \in E$ such that $u_1 = \mathbf{s}$, $v_k = u$, and $v_i = u_{i+1}$ for

$1 \leq i < k$. In addition $p = \varepsilon$, the empty sequence, is a path reaching the start node $\mathbf{s}$. We write $\mathsf{R}[u]$ for the set of paths reaching u.

Let Val be a set of values. A mapping $\sigma : X \to \mathsf{Val}$ that assigns a value to each variable is called a *state*; we write $\Sigma = \{\sigma \mid \sigma : X \to \mathsf{Val}\}$ for the set of states. For $x \in X$, $d \in \mathsf{Val}$, and $\sigma \in \Sigma$ we write $\sigma[x \mapsto d]$ for the state that maps x to d and coincides for the other variables with σ. We assume a fixed interpretation for the operators used in terms and we assume that the value of term t in state σ, which we denote by t^σ, is defined in the standard way.

In order to accommodate non-deterministic assignments we interpret statements by relations on Σ rather than functions. The relation associated with assignment statement $x := t$ is $[\![x := t]\!] \overset{\text{def}}{=} \{(\sigma, \sigma') \mid \sigma' = \sigma[x \mapsto t^\sigma]\}$; the relation associated with non-deterministic assignment $x :=?$ is $[\![x :=?]\!] \overset{\text{def}}{=} \{(\sigma, \sigma') \mid \exists d \in \mathsf{Val} : \sigma' = \sigma[x \mapsto d]\}$; and the relation associated with skip is the identity: $[\![\mathsf{skip}]\!] \overset{\text{def}}{=} \{(\sigma, \sigma') \mid \sigma = \sigma'\}$. This local interpretation of statements is straightforwardly extended to paths $p = \langle e_1, \ldots, e_k \rangle \in E^*$ as follows: $[\![p]\!] = [\![A(e_1)]\!] ; \ldots ; [\![A(e_k)]\!]$, where ; denotes relational composition.

Constant Propagation. A variable $x \in X$ is a *constant* at program point $u \in N$ if there is $d \in \mathsf{Val}$ such that $\sigma(x) = d$ for all $p \in \mathsf{R}[u]$, $(\sigma_0, \sigma) \in [\![p]\!]$. Arbitrary choice of initial state σ_0 reflects that we do not know the state in which the program is started.

We can weaken the demands for a constant detection algorithm by selecting a certain subset of expressions $S \subseteq \mathsf{Expr}$ that are interpreted precisely and assuming conservatively that assignments whose right hand does not belong to S assign an arbitrary value to their respective target variable. This idea can be made formal as follows.

For a given flow graph $G = (N, E, A, \mathbf{s}, \mathbf{e})$ and subset of expressions $S \subseteq \mathsf{Expr}$, let $G_S = (N, E, A_S, \mathbf{s}, \mathbf{e})$ be the flow graph with the same underlying graph but with the following weakened edge annotation:

$$A_S(u, v) = \begin{cases} x :=?\,, & \text{if } A(u, v) = (x := t) \text{ and } t \notin S \\ A(u, v)\,, & \text{otherwise} \end{cases}$$

A variable $x \in X$ is then called an *S-constant* at program point $u \in N$ in flow graph G if it is a constant at u in the weakened flow graph G_S. Clearly, if x is an S-constant at u it is also a constant at u but not vice versa. The *detection problem for S-constants* is the problem of deciding for a given set of variables X, flow graph G, variable x, and program point u whether x is an S-constant at u or not.

To study weakened versions of constant detection problems is particularly interesting for programs computing on the integers, i.e., if Expr is the set of integer expressions formed from integer constants and variables with the standard operators $+, -, *, \underline{\text{div}}, \underline{\text{mod}}$: we have seen in the introduction that the general constant detection problem is undecidable in this case.

Let us discuss some examples for illustration. S-constants with respect to the set $S = X \cup \mathbb{Z}$, i.e., the set of non-composite expressions, are known as *copy* *constants* [7]. S-constants with respect to the set $S = \{a*x+b \mid a,b \in \mathbb{Z}, x \in X\}$ are known as *linear constants* [18]. In this paper we tackle constants with respect to the set $S = \mathbb{Z}[x_1, \ldots, x_n]$, the set of multivariate polynomials in the variables $x_1, \ldots, x_n$ with coefficients in $\mathbb{Z}$, which we call *polynomial constants.*

We should emphasize two points about the above framework that make the construction of S-constant detection algorithms more challenging. Firstly, in contrast to the setup in [16], we allow assignment statements, whose right hand side does not belong to S. They are interpreted as non-deterministic assignments. Forbidding them is adequate for studying lower complexity bounds for analysis questions, which is the main concern of [16]. It is less adequate when we are concerned with detection algorithms because in practice we want to detect S-constants in the context of other code.

Secondly, a variable can be an S-constant although its value statically depends on an expression that is not in S. As a simple example consider the flow graph in Fig. 3 and assume that the expressions 0 and $y-y$ belong to S but e does not. Because $y-y$ is 0 for any value $y \in \mathbb{Z}$, an S-constant detection algorithm must identify x as a constant (of value 0), although its value statically depends on the uninterpreted expression e. Hence, S-constant detection must handle arithmetic properties of the expressions in S.

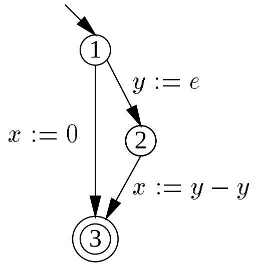

Fig. 3: An S-constant.

4 A Generic Algorithm

Suppose given a variable $x \in X$ and a program point $w \in N$. In this chapter we describe a generic algorithm for deciding whether x is an S-constant at w or not. While standard constant propagation works by forward propagation of variable assignments, we use a three phase algorithm that employs a backwards propagation of assertions. For the moment we can think of assertions as predicates on states as in program verification.

Phase 1: In the first phase we follow an arbitrary cycle-free path from s to w, for instance using depth-first search, and compute the value c, referred to as the *candidate value*, that x holds after this path is executed. This implies that, if x is a constant at w, it must be a constant of value c.

Phase 2: In the second phase we compute the weakest precondition for the assertion $x = c$ at program point w in G_S by means of a backwards data flow analysis.

Phase 3: Finally, we check whether the computed weakest precondition for $x = c$ at w is true, i.e., is valid for all states.

It is obvious that this algorithm is correct. The problem is that Phase 2 and 3 are in general not effective. However, as only assignments of a restricted form appear in G_S, the algorithm becomes effective for certain sets S, if assertions are represented appropriately. In the remainder of this section we analyze the requirements for adequate representations. For this purpose, we first characterize weakest preconditions in flow graphs.

Semantically, an *assertion* is a subset of states $\phi \subseteq \Sigma$. Given an assertion ϕ and a statement s, the *weakest precondition* of s for ϕ, $\mathsf{wp}(s)(\phi)$, is the largest assertion ϕ' such that execution of s from all states in ϕ' is guaranteed to terminate only in states in ϕ.[1] The following identities for the weakest precondition of assignment and skip statements are well-known:

$$\begin{aligned}
\mathsf{wp}(x := e)(\phi) &\overset{\text{def}}{=} \phi[e/x] &&\overset{\text{def}}{=} \{\sigma \mid \sigma[x \mapsto e^\sigma] \in \phi\}\\
\mathsf{wp}(x := ?)(\phi) &\overset{\text{def}}{=} \forall x(\phi) &&\overset{\text{def}}{=} \{\sigma \mid \forall d \in \mathbb{Z} : \sigma[x \mapsto d] \in \phi\}\\
\mathsf{wp}(\mathsf{skip})(\phi) &\overset{\text{def}}{=} \phi
\end{aligned}$$

These identities characterize weakest preconditions of basic statements. Let us now consider the following more general situation in a given flow graph $G = (N, E, A, \mathbf{s}, \mathbf{e})$: we are given an assertion $\phi \subseteq \Sigma$ as well as a program point $w \in N$ and we are interested in the weakest precondition that guarantees validity of ϕ whenever execution reaches w. The latter can be characterized as follows.

Let $W_0[w] = \phi$ and $W_0[u] = \Sigma$ and consider the following equation system consisting of one equation for each program point $u \in N$:

$$\mathbf{W}[u] \;=\; W_0[u] \cap \bigcap_{v \in Succ[u]} \mathsf{wp}(A(u,v))(\mathbf{W}[v])\,. \tag{1}$$

By the Knaster-Tarski fixpoint theorem, this equation system has a largest solution (w.r.t. subset inclusion) because $\mathsf{wp}(s)$ is well-known to be monotonic. By abuse of notation, we denote the weakest solution by the same letter $\mathbf{W}[u]$. For each program point $u \in N$, $\mathbf{W}[u]$ is the weakest assertion such that execution starting from u with any state in $\mathbf{W}[u]$ guarantees that ϕ holds whenever execution reaches w. In particular, $\mathbf{W}[\mathrm{s}]$ is the weakest precondition for validity of ϕ at w. The intuition underlying equation (1) is the following: firstly, $W_0[u]$ must be implied by $\mathbf{W}[u]$ and, secondly, for all successors v, we must guarantee that their associated condition $\mathbf{W}[u]$ is valid after execution of the statement $A(u,v)$ associated with the edge (u,v); hence $\mathsf{wp}(A(u,v))(\mathbf{W}[v])$ must be valid at u too.

For two reasons, the above equation system cannot be solved directly in general: firstly, because assertions may be infinite sets of states they cannot be represented explicitly; secondly, there are infinitely long descending chains of assertions such that standard fixpoint iteration does not terminate in general.

In order to construct an algorithm that detects S-constants we represent assertions by the members of a lattice $(\mathbb{D}, \sqsubseteq)$. Let us assume that $\gamma : \mathbb{D} \to 2^\Sigma$ captures how the lattice element represent assertions. First of all, we require

[1] In the sense of Dijkstra [6] this is the weakest *liberal* precondition as it does not guarantee termination. For simplicity we omit the qualifying prefix "liberal" in this paper.

$$\begin{tikzcd} L \arrow[loop left, "f"] \arrow[r, "\gamma"] & L' \arrow[loop right, "g"] \end{tikzcd}$$

Fig. 4. Situation in the transfer lemma.

(a) $\mathbb{D}$ has no infinite decreasing chains, i.e., there is no infinite chain $d_1 \sqsupseteq d_2 \sqsupseteq d_3 \sqsupseteq \ldots$.

This guarantees that maximal fixpoints of monotonic functions can effectively be computed by standard fixpoint iteration. Secondly, we suppose

(b) γ is universally conjunctive, i.e., $\gamma(\sqcap X) = \bigcap\{\gamma(d) \mid d \in X\}$ for all $X \subseteq \mathbb{D}$.

The most important reason for making this assumption is that it ensures that we can validly compute on representations without loosing precision: if we precisely mirror the equations characterizing weakest preconditions on representations, the largest solution of the resulting equation system on representations characterizes the representation of the weakest precondition by the following well-known lemma. It appears in the literature (for the dual situation of least fixpoints) under the name *Transfer Lemma* [2] or *μ-Fusion Rule* [11].

Lemma 1. *Suppose L, L' are complete lattices, $f : L \to L$ and $g : L' \to L'$ are monotonic functions and $\gamma : L \to L'$ (cf. Fig. 4).*

If γ is universally conjunctive and $\gamma \circ f = g \circ \gamma$ then $\gamma(\nu f) = \nu g$, where νf and νg are the largest fixpoints of f and g, respectively.

We must mirror the elements comprising the equation system characterizing weakest preconditions on representations precisely. Firstly, we must represent the start value, W_0. Universal conjunctivity of γ implies that $\gamma(\top) = \Sigma$, i.e., the top value of $\mathbb{D}$ is a precise representation of Σ. In addition, we require:

(c) Assertion $x = c$ can be represented precisely: for each $x \in X$, $c \in \mathsf{Val}$ we can effectively determine $d_{x=c} \in \mathbb{D}$ with $\gamma(d_{x=c}) = \{\sigma \in \Sigma \mid \sigma(x) = c\}$.

Secondly, we need effective representations for the operators appearing in equations. Requirement (b) implies that the meet operation of $\mathbb{D}$ precisely abstracts intersection of assertions. In order to enable effective computation of intersections, we require in addition:

(d) for given $d, d' \in \mathbb{D}$, we can effectively compute $d \sqcap d'$.

By induction this implies that we can compute finite meets $d_1 \sqcap \ldots \sqcap d_k$ effectively.

The only remaining operations on assertions are the weakest precondition transformers of basic statements. We must represent $\mathsf{wp}(x := t)$ for expressions $t \in S$, which is the substitution operator $(\cdot)[t/x]$ on assertions. As the S-constant detection algorithm computes the weakest precondition in weakened flow graph G_S, assignments $x := t$ with $t \notin S$ do not occur.

(e) There is a computable substitution operation $(\cdot)[t/x] : \mathbb{D} \to \mathbb{D}$ for each $x \in X$, $t \in S$, which satisfies $\gamma(d[t/x]) = \gamma(d)[t/x]$ for all $d \in \mathbb{D}$.

Obviously, $\mathsf{wp}(\mathsf{skip})$, the identity, is precisely represented by the identity on R. Thus, it remains to represent $\mathsf{wp}(x := ?)$:

(f) There is a computable projection operation $\mathit{proj}_i : \mathbb{D} \to \mathbb{D}$ for each variable $x_i \in X$ such that $\gamma(\mathit{proj}_i(d)) = \forall x_i(\gamma(d))$ for all $d \in \mathbb{D}$.

Finally, we need the following in order to make Phase 3 of the algorithm effective.

(g) Assertion true is decidable, i.e., there is a decision procedure that decides for a given $d \in \mathbb{D}$, whether $\gamma(d) = \Sigma$ or not.

If, for a given set $S \subseteq \mathsf{Expr}$, we can find a lattice satisfying requirements (a)–(g), we can effectively execute the three phase algorithm from the beginning of this section by representing assertions by elements from this lattice. This results in a detection algorithm for S-constants.

In this paper we are interested in detection of polynomial constants. Thus, from now on, let $\mathsf{Val} = \mathbb{Z}$ and $S = \mathbb{Z}[x_1, \ldots, x_n]$. The key idea for the detection of polynomial constants is to represent assertions by the zeros of ideals in the polynomial ring $\mathbb{Z}[x_1, \ldots, x_n]$ and to apply techniques from computable ring theory. A full introduction to this area is beyond the scope of this paper but we recall the facts needed in the next section and make some additional observations in Section 6. Accessible introductions can be found in standard textbooks on computer algebra. The case of polynomial rings over fields is covered, e.g., by [5, 8, 20], while [14] treats the more general case of polynomial rings over rings, that is of relevance here, as $\mathbb{Z}$ is an integral domain but not a field.

5 A Primer on Computable Ideal Theory

Recall that $\mathbb{Z}$ together with addition and multiplication forms a commutative ring, i.e., a structure $(R, +, \cdot)$ with a non-empty set R and two inner operations $+$ and $\cdot$ such that $(R, +)$ is an Abelian group, $\cdot$ is associative and commutative, and the distributive law $a \cdot (b + c) = a \cdot b + a \cdot c$ is valid for all $a, b, c \in R$. On the set of polynomials, $\mathbb{Z}[x_1, \ldots, x_n]$, we can define addition and multiplication operations in the standard way; this makes $\mathbb{Z}[x_1, \ldots, x_n]$ a commutative ring as well.

A non-empty subset $I \subseteq R$ of a ring R is called an *ideal* if $a + b \in I$ and $r \cdot a \in I$ for all $a, b \in I$, $r \in R$. The ideal *generated* by a subset $B \subseteq R$ is

$$(B) = \{r_1 \cdot b_1 + \ldots + r_k \cdot b_k \mid r_1, \ldots, r_k \in R, b_1, \ldots, b_k \in B\},$$

and B is called a *basis* or *generating system* of I if $I = (B)$. An ideal is called *finitely generated* if it has a finite basis $B = \{b_1, \ldots, b_m\}$. Hilbert's famous basis theorem tells us that $\mathbb{Z}[x_1, \ldots, x_n]$ is *Noetherian*, since $\mathbb{Z}$ is Noetherian, i.e., that there are no infinitely long strictly increasing chains $I_1 \subset I_2 \subset I_3 \subset \ldots$ of ideals in $\mathbb{Z}[x_1, \ldots, x_n]$. This implies that every ideal of $\mathbb{Z}[x_1, \ldots, x_n]$ is finitely generated.

It is important for our algorithm that we can compute effectively with ideals. While Hilbert's basis theorem ensures that we can represent every ideal of $\mathbb{Z}[x_1, \ldots, x_n]$ by a finite basis, in itself it does not give effective procedures for basic questions like membership tests or equality tests of ideals represented in this way. Indeed, Hilbert's proof of the basis theorem was famous (and controversial) at its time for its non-constructive nature.

Fortunately, the theory of Gröbner bases and the Buchberger algorithm provides a solution for some of these problems. While a complete presentation of this theory is way beyond the scope of this paper – the interested reader is pointed to the books mentioned above – a few sentences are in order here. A Gröbner basis is a basis for an ideal that has particularly nice properties. It can effectively be computed from any given finite basis by the Buchberger algorithm. There is a natural notion of reduction of a polynomial with respect to a set of polynomials such that reduction of a polynomial p with respect to a Gröbner basis always terminates and yields a unique result. This result is the zero polynomial if and only if p belongs to the ideal represented by the Gröbner basis. Hence reduction with respect to a Gröbner basis yields an effective membership test, that in turn can be used to check equality and inclusion of ideals.

In the terminology of [14], $\mathbb{Z}[x_1, \ldots, x_n]$ is a *strongly computable ring.* This implies that the following operations are computable for polynomials $p \in \mathbb{Z}[x_1, \ldots, x_n]$ and ideals $I, I' \subseteq \mathbb{Z}[x_1, \ldots, x_n]$ given by finite bases B, B', cf. [14]:

Ideal membership: Given an ideal I and a polynomial p. Is $p \in I$?
Ideal inclusion: Given two ideals I, I'. Is $I \subseteq I'$?
Ideal equality: Given two ideals I, I'. Is $I = I'$?
Sum of ideals: Given two ideals I, I'. Compute a basis for $I + I' \overset{\text{def}}{=} \{p + p' \mid p \in I, p' \in I'\}$. As a matter of fact, $I + I' = (B \cup B')$.
Intersection of ideals: Given two ideals I, I'. Compute a basis for $I \cap I'$.

It is well-known that $I + I'$ and $I \cap I'$ are again ideals if I and I' are. We can use the above operations as basic operations in our algorithms.

6 More About $\mathbb{Z}[x_1, \ldots, x_n]$

$\mathbb{Z}[x_1, \ldots, x_n]$ as a Complete Lattice. Interestingly, the ideals in $\mathbb{Z}[x_1, \ldots, x_n]$ form also a complete lattice under subset inclusion $\subseteq$. Suppose given a set $\mathcal{I}$ of ideals in $\mathbb{Z}[x_1, \ldots, x_n]$. Then the largest ideal contained in all ideals in $\mathcal{I}$ obviously is $\bigcap \mathcal{I}$, and the smallest ideal that contains all ideals in $\mathcal{I}$ is $\sum \mathcal{I} := \{r_1 \cdot a_1 + \ldots + r_k \cdot a_k \mid r_1, \ldots, r_k \in \mathbb{Z}[x_1, \ldots, x_n], a_1, \ldots, a_k \in \bigcup \mathcal{I}\}$. The least element of the lattice is the zero ideal $\{0\}$ that consists only of the zero polynomial and the largest element is $\mathbb{Z}[x_1, \ldots, x_n]$. While this lattice does not have finite height it is Noetherian by Hilbert's basis theorem such that we can effectively compute least fixpoints of monotonic functions on ideals of $\mathbb{Z}[x_1, \ldots, x_n]$ by standard fixpoint iteration.

Zeros. We represent assertions by the zeros of ideals in our algorithm. A state σ is called a *zero* of polynomial p if $p^\sigma = 0$; we denote the set of zeros of polynomial p by $\mathcal{Z}(p)$. More generally, for a subset $B \subseteq \mathbb{Z}[x_1, \dots, x_n]$, $\mathcal{Z}(B) = \{\sigma \mid \forall p \in B : p^\sigma = 0\}$. For later use, let us state some facts concerning zeros, in particular of the relationship of operations on ideals with operations on their zeros.

Lemma 2. *Suppose B, B' are sets of polynomials, q is a polynomial, I, I' are ideals, and $\mathcal{I}$ is a set of ideals in $\mathbb{Z}[x_1, \dots, x_n]$.*

1. *If $B \subseteq B'$ then $\mathcal{Z}(B) \supseteq \mathcal{Z}(B')$.*
2. *$\mathcal{Z}(B) = \mathcal{Z}((B)) = \bigcap_{p \in B} \mathcal{Z}(p)$. In particular, $\mathcal{Z}(q) = \mathcal{Z}((q))$.*
3. *$\mathcal{Z}(\sum \mathcal{I}) = \bigcap\{\mathcal{Z}(I) \mid I \in \mathcal{I}\}$. In particular, $\mathcal{Z}(I + I') = \mathcal{Z}(I) \cap \mathcal{Z}(I')$.*
4. *$\mathcal{Z}(\bigcap \mathcal{I}) = \bigcup\{\mathcal{Z}(I) \mid I \in \mathcal{I}\}$, if $\mathcal{I}$ is finite. In particular, $\mathcal{Z}(I \cap I') = \mathcal{Z}(I) \cup \mathcal{Z}(I')$.*
5. *$\mathcal{Z}(\{0\}) = \Sigma$ and $\mathcal{Z}(\mathbb{Z}[x_1, \dots, x_n]) = \emptyset$.*
6. *$\mathcal{Z}(I) = \Sigma$ if and only if $I = \{0\} = (0)$.*

Substitution. Suppose given a polynomial $p \in \mathbb{Z}[x_1, \dots, x_n]$ and a variable $x \in X$. We can define a substitution operation on ideals I as follows: $I[p/x] = (\{q[p/x] \mid q \in I\})$, where the substitution of polynomial p for x in q, $q[p/x]$, is defined as usual. By definition, $I[p/x]$ is the smallest ideal that contains all polynomials $q[p/x]$ with $q \in I$. From a basis for I, a basis for $I[p/x]$ is obtained in the expected way: if $I = (B)$, then $I[p/x] = (\{b[p/x] \mid b \in B\})$. Thus, we can easily obtain a finite basis for $I[p/x]$ from a finite basis for I: if $I = (b_1, \dots, b_k)$ then $I[p/x] = (b_1[p/x], \dots, b_k[p/x])$. Hence we can add substitution to our list of computable operations.

The substitution operation on ideals defined in the previous paragraph mirrors precisely semantic substitution in assertions which has been defined in connection with $\mathsf{wp}(x := e)$.

Lemma 3. *$\mathcal{Z}(I)[p/x] = \mathcal{Z}(I[p/x])$.*

We leave the proof of this equation that involves the substitution lemma known from logic to the reader.

Projection. In this section we define projection operators $proj_i$, $i = 1, \dots, n$, such that for each ideal I, $\mathcal{Z}(proj_i(I)) = \forall x_i(\mathcal{Z}(I))$. Semantic universal quantification over assertions has been defined in connection with $\mathsf{wp}(x :=?)$.

A polynomial $p \in \mathbb{Z}[x_1, \dots, x_n]$ can uniquely be written as a polynomial in x_i with coefficients in $\mathbb{Z}[x_1, \dots, x_{i-1}, x_{i+1}, x_n]$, i.e., in the form $p = c_k x_i^k + \ldots + c_0 x_i^0$, where $c_0, \dots, c_k \in \mathbb{Z}[x_1, \dots, x_{i-1}, x_{i+1}, x_n]$, and $c_k \neq 0$ if $k > 0$. We call $c_0, \dots, c_k$ the coefficients of p with respect to x_i and let $\mathcal{C}_i(p) = \{c_0, \dots, c_k\}$.

Lemma 4. *$\forall x_i(\mathcal{Z}(p)) = \mathcal{Z}(\mathcal{C}_i(p))$.*

Proof. Let $p = c_k x_i^k + \ldots + c_0 x_i^0$ with $\mathcal{C}_i(p) = \{c_0, \dots, c_k\}$.

'$\supseteq$': Let $\sigma \in \mathcal{Z}(\mathcal{C}_i(p))$. We have $c_k^{\sigma[x_i \mapsto d]} = c_k^\sigma = 0$ for all $d \in \mathbb{Z}$ because c_k is independent of x_i. Hence, $p^{\sigma[x_i \mapsto d]} = c_k^{\sigma[x_i \mapsto d]} d^k + \ldots + c_0^{\sigma[x_i \mapsto d]} d^0 = 0d^k + \ldots + 0d^0 = 0$ for all $d \in \mathbb{Z}$, i.e. $\sigma \in \forall x_i(\mathcal{Z}(p))$.
'$\subseteq$': Let $\sigma \in \forall x_i(\mathcal{Z}(p))$. We have $c_k^{\sigma[x_i \mapsto d]} = c_k^\sigma$ for all $d \in \mathbb{Z}$ because c_k is independent of x_i. Therefore, $c_k^\sigma d^k + \ldots + c_0^\sigma d^0 = c_k^{\sigma[x_i \mapsto d]} d^k + \ldots + c_0^{\sigma[x_i \mapsto d]} d^0 = p^{\sigma[x \mapsto d]} = 0$ for all $d \in \mathbb{Z}$ because of $\sigma \in \forall x_i(\mathcal{Z}(p))$. This means that the polynomial $c_k^\sigma x_i^k + \ldots + c_0^\sigma x_i^0$ vanishes for all values of x_i. Hence, it has more than k zeros which implies that it is the zero polynomial. Consequently, $c_j^\sigma = 0$ for all $j = 0, \ldots, k$, i.e., $\sigma \in \mathcal{Z}(\mathcal{C}_i(p))$. □

Suppose $I \subseteq \mathbb{Z}[x_1, \ldots, x_n]$ is an ideal with basis B.

Lemma 5. $\forall x_i(\mathcal{Z}(I)) = \mathcal{Z}(\bigcup_{p \in B} \mathcal{C}_i(p))$.

Proof. $\forall x_i(\mathcal{Z}(I)) = \forall x_i(\mathcal{Z}(B)) = \forall x_i(\bigcap_{p \in B} \mathcal{Z}(p)) = \bigcap_{p \in B} \forall x_i(\mathcal{Z}(p)) = \bigcap_{p \in B} \mathcal{Z}(\mathcal{C}_i(p)) = \mathcal{Z}(\bigcup_{p \in B} \mathcal{C}_i(p))$. □

In view of this formula, it is natural to define $\mathit{proj}_i(I) = (\bigcup_{p \in B} \mathcal{C}_i(p))$ where B is a basis of I. It is not hard to show that this definition is independent of the basis; we leave this proof to the reader. Obviously, proj_i is effective: if I is given by a finite basis $\{b_1, \ldots, b_k\}$ then $\mathit{proj}_i(I)$ is given by the finite basis $\bigcup_{j=1}^k \mathcal{C}_i(b_j)$.

Corollary 6. $\forall x_i(\mathcal{Z}(I)) = \mathcal{Z}(\mathit{proj}_i(I))$.

Proof. $\forall x_i(\mathcal{Z}(I)) = \mathcal{Z}(\bigcup_{p \in B} \mathcal{C}_i(p)) = \mathcal{Z}((\bigcup_{p \in B} \mathcal{C}_i(p))) = \mathcal{Z}(\mathit{proj}_i(I))$. □

7 Detection of Polynomial Constants

We represent assertions by ideals of the polynomial ring $\mathbb{Z}[x_1, \ldots, x_n]$ in the detection algorithm for polynomial constants. Thus, let $\mathbb{D}$ be the set of ideals of $\mathbb{Z}[x_1, \ldots, x_n]$) and $\sqsubseteq$ be $\supseteq$. The representation mapping is $\gamma(I) = \mathcal{Z}(I)$. Note that the order is *reverse* inclusion of ideals. This is because larger ideals have smaller sets of zeros. Thus, the *meet* operation is the *sum* operation of ideals and the top element is the ideal $\{0\} = (0)$.

In a practical algorithm, ideals are represented by finite bases. For transparency, we suppress this further representation step but ensure that only operations that can effectively be computed on bases are used.

The lattice $(\mathbb{D}, \supseteq)$ satisfies requirements (a)–(g) of Section 4:

(a) $\mathbb{Z}[x_1, \ldots, x_n]$ is Noetherian.
(b) By the identity $\mathcal{Z}(\sum \mathcal{I}) = \bigcap\{\mathcal{Z}(I) \mid I \in \mathcal{I}\}$, $\mathcal{Z}$ is universally conjunctive.
(c) Suppose $x \in X$ and $c \in \mathbb{Z}$. Certainly, a state is a zero of the ideal generated by the polynomial $x - c$ if and only if it maps x to c. Hence, we choose $d_{x=c}$ as the ideal $(x - c)$ generated by $x - c$.
(d) In Section 5 we have seen that the sum of two ideals can effectively be computed on bases.
(e) By Section 6, $(\cdot)[p/x]$ is an adequate, computable substitution operation.

(f) Again by Section 6, $proj_i$ is an adequate, computable projection operation.

(g) We know that $\mathcal{Z}(I) = \Sigma$ if and only if $I = \{0\}$. Moreover, the only basis of the ideal $\{0\}$ is $\{0\}$ itself. Hence, in order to decide whether an ideal I given by a basis B represents Σ, we only need to check whether $B = \{0\}$.

We can thus apply the generic algorithm from Section 4 for the detection of polynomial constants. In order to make this more specific, we put the pieces together, and describe the resulting algorithm in more detail.

Suppose given a variable $x \in X$ and a program point $w \in N$ in a flow graph $G = (N, E, A, \mathbf{s}, \mathbf{e})$. Then the following algorithm decides whether x is a polynomial constant at w or not:

Phase 1: Determine a candidate value $c \in \mathbb{Z}$ for x at w by executing an arbitrary (cycle-free) path from $\mathbf{s}$ to w.

Phase 2: Associate with each edge $(u, v) \in E$ a transfer function $f_{(u,v)} : \mathbb{D} \to \mathbb{D}$ that represents $\mathsf{wp}(A_S(u, v))$:

$$f_{(u,v)}(I) = \begin{cases} I & \text{if } A(u,v) = \mathsf{skip} \\ I[p/x] & \text{if } A(u,v) = (x := p) \text{ with } p \in \mathbb{Z}[x_1, \dots, x_n] \\ proj_x(I) & \text{if } A(u,v) = (x := t) \text{ with } t \notin \mathbb{Z}[x_1, \dots, x_n] \\ proj_x(I) & \text{if } A(u,v) = (x :=?) \end{cases}$$

Set $A_0[w] = (x - c)$ and $A_0[u] = (0)$ for all $u \in N \backslash \{w\}$ and compute the largest solution (w.r.t. $\sqsubseteq = \supseteq$) of the equation system

$$\mathbf{A}[u] = A_0[u] + \sum_{v \in Succ[u]} f_{(u,v)}(\mathbf{A}[v]) \quad \text{for each } u \in N\,.$$

We can do this as follows. Starting from $A_0[u]$ we iteratively compute, simultaneously for all program points $u \in N$, the following sequences of ideals

$$A_{i+1}[u] = A_i[u] + \sum_{v \in Succ[u]} f_{(u,v)}(\mathbf{A}_i[v])\,.$$

We stop upon stabilization, i.e., when we encounter an index i_s with $A_{i_s+1}[u] = A_{i_s}[u]$ for all $u \in N$. Obviously, $A_0[u] \subseteq A_1[u] \subseteq A_2[u] \subseteq \dots$, such that computation must terminate eventually because $\mathbb{Z}[x_1, \dots, x_n]$ is Noetherian. In this computation we represent ideals by finite bases and perform Gröbner basis computations in order to check whether $A_{i+1}[u] = A_i[u]$.[2]

Phase 3: Check if the ideal computed for the start node, $A_{i_s}[\mathbf{s}]$, is (0). If so, x is a polynomial constant of value v at w; otherwise, x is not a polynomial constant at w.

Phase 2 can be seen as a backwards data flow analysis in a framework in which ideals of $\mathbb{Z}[x_1, \dots, x_n]$ constitute data flow facts, the transfer functions

[2] As $A_{i+1}[u] \supseteq A_i[u]$ by construction, it suffices to check $A_{i+1}[u] \subseteq A_i[u]$.

are the functions $f_{(u,v)}$ specified above, and the start value is A_0. Of course, we can use any evaluation strategy instead of naive iteration.

We do not know any complexity bound for our algorithm. Our termination proof relies on Hilbert's basis theorem and its standard proof is non-constructive and does not provide an upper bound for the maximal length of strictly increasing chains of ideals. Therefore, we cannot bound the number of iterations performed by our algorithm.

8 Conclusion

In this paper we have shown that polynomial constants are decidable. Our algorithm can easily be extended to handle conditions of the form $p \neq 0$ with $p \in \mathbb{Z}[x_1, \ldots, x_n]$. The weakest precondition is $\mathsf{wp}(p \neq 0)(\phi) = (p \neq 0 \Rightarrow \phi) = (p = 0 \vee \phi)$ and if ϕ is represented by an ideal I, the assertion $p = 0 \vee \phi$ is represented by the ideal $I \cap (p)$ according to Lemma 2. This observation can be used to handle such conditions in our algorithm. We can extend this easily to an arbitrary mixture of disjunctions and conjunctions of conditions of the form $p \neq 0$. Of course, we cannot handle the dual form of conditions, $p = 0$: with both types of conditions we can obviously simulate two-counter machines.

The idea to detect constants with a symbolic weakest precondition computation has previously been used in a polynomial-time algorithm for detection of *Presburger constants* [16]. In Presburger constant detection only the integer operators $+$ and $-$ are interpreted and assertions are represented by affine vector spaces over $\mathbb{Q}$. In contrast to our algorithm, the Presburger constant detection algorithm cannot easily be extended to conditions as affine spaces are not closed under union.

Standard constant propagation relies on forward propagation while we use backwards propagation of assertions. Interestingly, Presburger constants can also be detected by forward propagation of affine spaces. Karr [10] describes such an algorithm but does not address completeness issues. In forward propagation of assertions we effectively compute strongest postconditions rather than weakest precondition and this computation involves union of assertions rather than intersection. Because affine spaces are not closed under union, Karr defines a (complicated) union operator of affine spaces that over-approximates their actual union by an affine space. One is tempted to consider forward propagation of ideals of $\mathbb{Z}[x_1, \ldots, x_n]$. At first glance, this idea looks promising, because ideals are closed under intersection and intersection of ideals corresponds to union of their sets of zeros, such that we can even precisely represent the union of assertions. There is, however, another problem: $\mathbb{Z}[x_1, \ldots, x_n]$ is not 'co-Noetherian', i.e., there are infinitely long strictly *decreasing* chains of ideals, e.g., $(x) \supset (x^2) \supset (x^3) \supset \ldots$. Therefore, strongest postcondition computations with ideals cannot be guaranteed to terminate in general.

Our approach to compute weakest preconditions symbolically with effective representations is closely related to abstract interpretation [3, 4]. Requirement (b) of the generic algorithm, universal conjunctivity of the representation mapping $\gamma : \mathbb{D} \to 2^{\Sigma}$, implies that γ has a lower adjoint, i.e., that there is a monotonic mapping $\alpha : 2^{\Sigma} \to \mathbb{D}$ such that (α, γ) is a Galois connection [13]. In the standard abstract interpretation framework, we are interested in computation of least fixpoints and the lower adjoint, α, is the abstraction mapping. Here, we are in the dual situation: we are interested in computation of greatest fixpoints. Thus, the role of the abstraction is played by the upper adjoint, $\gamma : \mathbb{D} \to 2^{\Sigma}$. Funnily, this means that in a technical sense the members of $\mathbb{D}$ provide more concrete information than the members of 2^{Σ} and that we compute on the concrete side of the abstract interpretation. Thus, we interpret the lattice $\mathbb{D}$ as an *exact partial representation* rather than an abstract interpretation. The representation via $\mathbb{D}$ is *partial* because it does not represent all assertions exactly; this is indispensable due to countability reasons because we cannot represent all assertions effectively. It is an *exact representation* because it allows us to infer the weakest preconditions arising in the S-constant algorithms precisely, which is achieved by ensuring that the initial value of the fixpoint computation is represented exactly and that the occurring operations on representations mirror the corresponding operations on assertions precisely.

By the very nature of Galois connections, the representation mapping γ and its lower adjoint α satisfy the two inequalities $\alpha \circ \gamma \sqsubseteq \mathsf{Id}_{\mathbb{D}}$ and $\mathsf{Id}_{2^{\Sigma}} \subseteq \gamma \circ \alpha$, where $\mathsf{Id}_{\mathbb{D}}$ and $\mathsf{Id}_{2^{\Sigma}}$ are the identities on $\mathbb{D}$ and 2^{Σ}, respectively. Interestingly, none of these inequalities degenerates to an equality when we represent assertions by ideals of $\mathbb{Z}[x_1, \dots, x_n]$ as in our algorithm for detection of polynomial constants. On the one hand, $\gamma \circ \alpha \neq \mathsf{Id}_{2^{\Sigma}}$ because the representation is necessarily partial. On the other hand, $\alpha \circ \gamma \neq \mathsf{Id}_{\mathbb{D}}$ because the representation of assertions is not unique. For example, if $p \in \mathbb{Z}[x_1, \dots, x_n]$ does not have a zero in the integers, we have $\mathcal{Z}((p)) = \emptyset$ such that $\mathcal{Z}((p)) = \mathcal{Z}((1)) = \mathcal{Z}(\mathbb{Z}[x_1, \dots, x_n])$. But by undecidability of Hilbert's tenth problem, we cannot decide whether we are faced with such a polynomial p and thus cannot effectively identify (p) and (1). This forces us to work with a non-unique representation. While we cannot decide whether the set of zeros of an ideal I given by a basis B is empty, we can decide whether it equals Σ because this only holds for $I = (0)$. Fortunately, this is the only question that needs to be answered for the weakest precondition.

As a consequence of non-uniqueness, the weakest precondition computation on ideals does not necessarily stop once it has found a collection of ideals that represents the largest fixpoint on assertions but may proceed to larger ideals that represent the same assertions. Fortunately, we can still prove termination by arguing on ideals directly.

Let us discuss some possible directions for future work. Firstly, it is interesting to implement the detection algorithm for polynomial constants and evaluate how it performs in practice. Secondly, we can look for other applications of the generic algorithm. Of course, we can tackle, e.g, polynomial constants over $\mathbb{Q}$ rather than $\mathbb{Z}$, where we can use essentially the same algorithm because $\mathbb{Q}[x_1, \dots, x_n]$ is also a strongly computable ring. But we may also identify other classes where assertions can be represented symbolically. On the theoretical side, there is the challenge to diminish the gap between the upper and lower complexity bound for the detection problem of polynomial constants. Currently, we have decidability as an upper bound, as witnessed by the algorithm in this paper, and PSPACE-hardness as a lower bound [16].

Acknowledgment. We thank the anonymous referees for their comments that helped to improve the submitted version.

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.
2. K. R. Apt and G. D. Plotkin. Countable nondeterminism and random assignment. *Journal of the ACM*, 33(4):724–767, 1986.
3. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings 4th POPL*, Los Angeles, California, 1977.
4. P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Logic Computat.*, 4(2):511–547, 1992.
5. J. H. Davenport, Y. Siret, and E. Tournier. *Computer Algebra: Systems and Algorithms for Algebraic Computation.* Academic Press, 1988.
6. E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.
7. C. Fischer and R. LeBlanc. *Crafting a Compiler.* Benjamin/Cummings Publishing Co., Inc., Menlo Park, CA, 1988.
8. K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for Computer Algebra.* Kluwer, 1992.
9. M. S. Hecht. *Flow analysis of computer programs.* Elsevier North-Holland, 1977.
10. M. Karr. Affine relationships among variables of a program. *Acta Informatica*, 6:133–151, 1976.
11. Mathematics of Program Construction Group. Fixed-point calculus. *Information Processing Letters*, 53(3):131–136, 1995.
12. Y. V. Matiyasevich. *Hilbert's Tenth Problem.* The MIT Press, 1993.
13. A. Melton, D. A. Schmidt, and G. E. Strecker. Galois connections and computer science applications. In D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Category Theory and Computer Programming*, LNCS 240, pages 299–312. Springer-Verlag, 1985.
14. B. Mishra. *Algorithmic Algebra.* Springer-Verlag, 1993.
15. S. S. Muchnick. *Advanced compiler design and implementation.* Morgan Kaufmann Publishers, San Francisco, California, 1997.
16. M. Müller-Olm and O. Rüthing. The complexity of constant propagation. In D. Sands, editor, *ESOP 2001*, LNCS 2028, pages 190–205. Springer, 2001.
17. J. R. Reif and H. R. Lewis. Symbolic evaluation and the global value graph. In *Conf. Rec. 4th ACM Symposium on Principles of Programming Languages POPL'77*, pages 104–118, Los Angeles, CA, January 1977.
18. M. Sagiv, T. Reps, and S. Horwitz. Precise interprocedural dataflow analysis with applications to constant propagation. *Theoretical Computer Science*, 167(1–2):131–170, 1996.
19. B. Steffen and J. Knoop. Finite constants: Characterizations of a new decidable set of constants. *Theoretical Computer Science*, 80(2):303–318, 1991.
20. F. Winkler. *Polynomial Algorithms.* Springer-Verlag, 1996.