Interprocedurally Analyzing Polynomial Identities

Markus Müller-Olm[1], Michael Petter[2], and Helmut Seidl[2]

[1] Westfälische Wilhelms-Universität Münster, Institut für Informatik
Einsteinstr. 62, 48149 Münster, Germany
mmo@math.uni-muenster.de
[2] TU München, Institut für Informatik, I2
80333 München, Germany
seidl@in.tum.de

Abstract. Since programming languages are Turing complete, it is impossible to decide for all programs whether a given non-trivial semantic property is valid or not. The way-out chosen by abstract interpretation is to provide *approximate* methods which may fail to certify a program property on some programs. Precision of the analysis can be measured by providing classes of programs for which the analysis is complete, i.e., decides the property in question. Here, we consider analyses of polynomial identities between integer variables such as $\mathbf{x}_1 \cdot \mathbf{x}_2 - 2\mathbf{x}_3 = 0$. We describe current approaches and clarify their completeness properties. We also present an extension of our approach based on weakest precondition computations to programs with procedures and equality guards.

1 Introduction

Invariants and intermediate assertions are the key to deductive verification of programs. Correspondingly, techniques for automatically checking and finding invariants and intermediate assertions have been studied (cf., e.g., [3, 2, 22]). In this paper we present analyses that check and find valid polynomial identities in programs. A polynomial identity is a formula $p(\mathbf{x}_1, \ldots, \mathbf{x}_k) = 0$ where $p(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ is a multi-variate polynomial in the program variables $\mathbf{x}_1, \ldots, \mathbf{x}_k$.[3]

Looking for valid polynomial identities is a rather general question with many applications. Many classical data flow analysis problems can be seen as problems about polynomial identities. Some examples are: finding *definite equalities among variables* like $\mathbf{x} = \mathbf{y}$; *constant propagation*, i.e., detecting variables or expressions with a constant value at run-time; *discovery of symbolic constants* like $\mathbf{x} = 5\mathbf{y}+2$ or even $\mathbf{x} = \mathbf{y}\mathbf{z}^2+42$; *detection of complex common sub-expressions* where even expressions are sought which are syntactically different but have the same value at run-time such as $\mathbf{x}\mathbf{y}+42 = \mathbf{y}^2+5$; and *discovery of loop induction variables*.

Polynomial identities found by an automatic analysis are also useful for program verification, as they provide non-trivial valid assertions about the program. In particular, loop invariants can be discovered fully automatically. As polynomial identities express

[3] More generally our analyses can handle positive Boolean combinations of polynomial identities.

quite complex relationships among variables, the discovered assertions may form the backbone of the program proof and thus significantly simplify the verification task.

In the following, we critically review different approaches for determining valid polynomial identities with an emphasis on their precision. In expressions, only addition and multiplication are treated exactly, and, except for guards of the form $p \neq 0$ for polynomials p, conditional choice is generally approximated by non-deterministic choice. These assumptions are crucial for the design of effective exact analyses [12, 13]. Such programs will be called *polynomial* in the sequel.

Much research has been devoted to polynomial programs without procedure calls, i.e., *intraprocedural* analyses. Karr was the first who studied this problem [11]. He considers polynomials of degree at most 1 (*affine* expressions) both in assignments and in assertions and presents an algorithm which, in absence of guards, determines all valid affine identities. This algorithm has been improved by the authors and extended to deal with polynomial identities up to a fixed degree [13]. Gulwani and Necula also re-considered Karr's analysis problem [7] recently. They use randomization in order to improve the complexity of the analysis at the price of a small probability of finding invalid identities.

The first attempt to generalize Karr's method to *polynomial* assignments is [12] where we show that validity of a polynomial identity at a given target program point is decidable for polynomial programs. Later, Rodriguez-Carbonell et al. propose an analysis based on the observation that the set of identities which are valid at a program point can be described by a polynomial *ideal* [20]. Their analysis is based on a constraint system over polynomial ideals whose greatest solution precisely characterizes the set of all valid identities. The problem, however, with this approach is that *descending* chains of polynomial ideals may be infinite implying that no effective algorithm can be derived from this characterization. Therefore, they provide special cases [21] or approximations that allow to infer some valid identities. Opposed to that, our approach is based on effective weakest precondition computations [12, 14]. We consider assertions to be checked for validity and compute for every program point weakest preconditions which also are represented by ideals. In this case, fixpoint iteration results in *ascending* chains of ideals which are guaranteed to terminate by Hilbert's basis theorem. Therefore, our method provides a decision procedure for validity of polynomial identities. By using a *generic* identity with unknowns instead of coefficients, this method also provides an algorithm for *inferring* all valid polynomial identities up to a given degree [14].

An interprocedural generalization of Karr's algorithm is given in [15]. Using techniques from linear algebra, we succeed in inferring all interprocedurally valid affine identities in programs with affine assignments and no guards. The method easily generalizes to inferring also all polynomial identities up to a fixed degree in these programs. A generalization of the intraprocedural randomized algorithm to programs with procedures is possible as well [8]. A first attempt to infer polynomial identities in presence of polynomial assignments and procedure calls is provided by Colon [4]. His approach is based on ideals of polynomial *transition invariants*. We illustrate, though, the pitfalls of this approach and instead show how the idea of precondition computations can be extended to an interprocedural analysis. In a natural way, the latter analysis also extends the interprocedural analysis from [15] where only affine assignments are considered.

The rest of the paper is organized as follows. Section 2 introduces basic notions. Section 3 provides a precise characterization of all valid polynomial identities by means of a constraint system. This characterization is based on forward propagation. Section 4 provides a second characterization based on effective weakest precondition computation. This leads to backwards-propagation algorithms. Both Sections 3 and 4 consider only programs without procedures. Section 5 explains an extension to polynomial programs with procedures based on polynomial transition invariants and indicates its limitations. Section 6 presents a possible extension of the weakest-precondition approach to procedures. Section 7 then indicates how equality guards can be added to the analyses. Finally, Section 8 summarizes and gives further directions of research.

2 The General Set-up

We use similar conventions as in [15, 17, 16] which we recall here in order to be self-contained. Thus, programs are modeled by systems of non-deterministic flow graphs that can recursively call each other as in Fig. 1. Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ be the set of

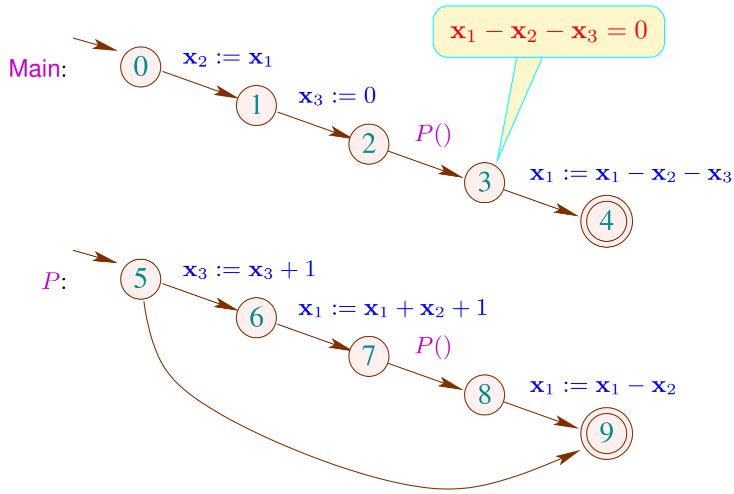

Fig. 1. An interprocedural program.

(global) variables the program operates on. For ease of presentation, we assume that variables take values in the field $\mathbb{Q}$. Similar arguments, though, can also be applied in case values are integers from $\mathbb{Z}$ or even when we consider values from a modular ring $\mathbb{Z}_m$ for some $m = 2^w, w \geq 1$, as used in programming languages such as Java [17, 16].

In the programs we analyze, we assume the assignments to variables to be of the form $\mathbf{x}_j := p$ for some polynomial p from $\mathbb{Q}[\mathbf{X}]$, i.e., the ring of all polynomials with coefficients from $\mathbb{Q}$ and variables from $\mathbf{X}$. Note that this restriction does not come by accident. It is well-known [9, 19] that it is undecidable for non-deterministic flow graphs to determine whether a given variable holds a constant value at a given program point in all executions if the full standard signature of arithmetic operators (addition, subtraction, multiplication, and division) is available. Constancy of a variable is obviously a polynomial identity: $\mathbf{x}$ is a constant at program point n if and only if the polynomial identity $\mathbf{x} - c = 0$ is valid at n for some $c \in \mathbb{Q}$. Clearly, we can write all expressions involving addition, subtraction, and multiplication with polynomials. Thus, if we allow also division, validity of polynomial identities becomes undecidable.

Assignments with non-polynomial expressions or input dependent values are therefore assumed to be abstracted with *non-deterministic assignments*. A non-deterministic assignment of the form $\mathbf{x}_j :=?$ (with $\mathbf{x}_j \in \mathbf{X}$) is meant to represent the non-deterministic choice between all assignments $\mathbf{x}_j := c$, $c \in \mathbb{Q}$. In general, we also assume that conditional branching is abstracted with non-deterministic branching, i.e., either way is possible. Note that in [13] it is pointed out that in presence of equality guards, exact constant propagation again becomes undecidable. The only form of guards at edges which we can handle within our framework precisely are disequality guards of the form $p \neq 0$ for some polynomial p. In order to reduce the number of program points in examples, we sometimes annotate edges with sequences of assignments. Also, we use assignments $\mathbf{x}_j := \mathbf{x}_j$ which have no effect onto the program state as skip-statements and omit these in pictures. For the moment, skip-statements are used to abstract, e.g., equality guards. In Section 7, we will present methods which approximatively deal with equality guards.

A *polynomial program* comprises a finite set Proc of *procedure names* with one distinguished procedure Main. Execution starts with a call to Main. Each procedure $q \in \mathsf{Proc}$ is specified by a distinct finite edge-labeled *control flow graph* with a single start point st_q and a single return point ret_q where each edge is labeled with an assignment, a non-deterministic assignment, a disequality guard or a call to some procedure. For simplicity, we only consider procedures without parameters or return values operating on global variables. The framework, though, can straightforwardly be extended to procedures with local variables, call-by-value parameter passing and return values.

The basic approach of [15, 13, 17] which we take up here is to construct a precise abstract interpretation of a constraint system characterizing the concrete program semantics. For that, we model a *state* attained by program execution when reaching a program point or procedure by a k-dimensional vector $x = [x_1, \ldots, x_k] \in \mathbb{Q}^k$ where x_i is the value assigned to variable $\mathbf{x}_i$. Runs through the program execute sequences of assignments and guards. Each such sequence induces a *partial polynomial transformation* of the program state.

A (total) polynomial transformation τ can be described by a vector of polynomials $\tau = [q_1, \ldots, q_k]$ where τ applied to a vector x equals the vector:

$$\tau(x) = [q_1(x), \ldots, q_k(x)]$$

where we have written $q'(x)$ for the value returned by a polynomial q' for the vector x. A partial polynomial transformation π is a pair $\pi = (q, \tau)$ of a polynomial q and a polynomial transformation τ. If $q(x) \neq 0$ then $\pi(x)$ is defined and returns $\tau(x)$. Otherwise, $\pi(x)$ is undefined. Partial polynomial transformations are closed under composition

[14]. The partial polynomial transformations corresponding to single assignments and disequality guards are given by:

$$\begin{array}{lcl} [\![\mathbf{x}_j := p]\!] & = & (0, [\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}, p, \mathbf{x}_{j+1}, \ldots, \mathbf{x}_k]) \\ [\![q \neq 0]\!] & = & (q, [\mathbf{x}_1, \ldots, \mathbf{x}_k]) \end{array}$$

The definition of a partial polynomial transformation is readily extended to sets of states. Since in general, procedures have multiple runs, we model their semantics by *sets* of partial polynomial transformations.

3 Intraprocedural Analysis: Forward Iteration

Let $\pi = (q, \tau)$ be the partial polynomial transformation induced by some program run. Then, a polynomial identity $p = 0$ is said to be *valid* after this run if, for each initial state $x \in \mathbb{Q}^k$, either $q(x) = 0$ – in this case the run is not executable from x – or $q(x) \neq 0$ and $p(\tau(x)) = 0$ – in this case the run is executable from x and the final state computed by the run is $\tau(x)$. A polynomial identity $p = 0$ is said to be valid at a program point v if it is valid after every run reaching v.

Clearly, if $p = 0$ is valid then also $r \cdot p = 0$ for arbitrary polynomials r. Also, if $p_1 = 0$ and $p_2 = 0$ are valid then also $p_1 + p_2 = 0$ is valid. Thus, the set of polynomials p for which $p = 0$ is valid at v forms a *polynomial ideal*.[4] Recall that, by Hilbert's basis theorem, every polynomial ideal $I \subseteq \mathbb{Q}[\mathbf{X}]$ can be finitely represented by:

$$I = \langle p_1, \ldots, p_m \rangle =_{\mathrm{df}} \{r_1 \cdot p_1 + \ldots + r_m \cdot p_m \mid r_i \in \mathbb{Q}[\mathbf{X}]\}$$

for suitable $p_1, \ldots, p_m \in \mathbb{Q}[\mathbf{X}]$. The set $\{p_1, \ldots, p_m\}$ is also said to *generate* the ideal I. Based on such representations, algorithms have been developed for fundamental operations on ideals [1]. In particular, membership is decidable for ideals as well as containment and equality. Moreover, the set of all ideals $I \subseteq \mathbb{Q}[\mathbf{X}]$ forms a *complete lattice* w.r.t. set inclusion "$\subseteq$" where the least and greatest elements are the zero ideal $\{0\}$ and the complete ring $\mathbb{Q}[\mathbf{X}]$, respectively. The greatest lower bound of a set of ideals is simply given by their intersection while their least upper bound is the ideal *sum*. More precisely, the sum of the ideals I_1 and I_2 is defined by

$$I_1 \oplus I_2 = \{p_1 + p_2 \mid p_1 \in I_1, p_2 \in I_2\}$$

A set of generators for the sum $I_1 \oplus I_2$ is obtained by taking the union of sets of generators for the ideals I_1 and I_2.

For the moment, let us consider *intraprocedural* analysis only, i.e., analysis of programs just consisting of the procedure Main and without procedure calls. Such program consist of a single control-flow graph. As an example, consider the program in Fig. 2.

Given that the set of valid polynomial identities at every program point can be described by polynomial ideals, we can characterize the sets of valid polynomial identities by means of the following constraint system $\mathcal{F}$:

[4] A polynomial ideal I is a set of polynomial which is closed under addition and under multiplication with arbitrary polynomials: $\forall p, q \in I : p + q \in I$ and $\forall p \in I, q \in \mathbb{Q}[\mathbf{X}] : p \cdot q \in I$.

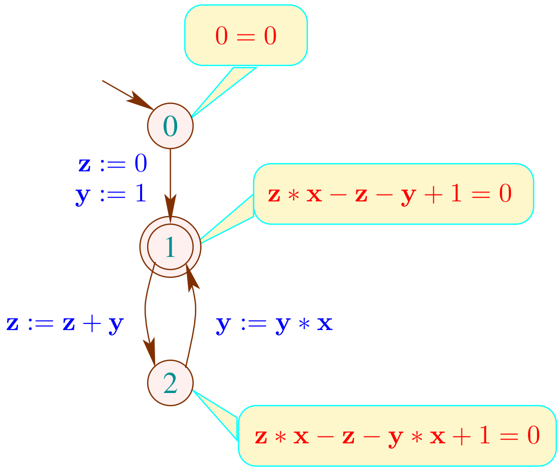

Fig. 2. A program without procedures.

$$
\begin{array}{llll}
\mathcal{F}(\mathsf{start}) & \subseteq & \{0\} & \\
\mathcal{F}(v) & \subseteq & [\![\mathbf{x}_i := p]\!]^\sharp(\mathcal{F}(u)) & (u,v) \text{ an assignment } \mathbf{x}_i := p \\
\mathcal{F}(v) & \subseteq & [\![\mathbf{x}_i := ?]\!]^\sharp(\mathcal{F}(u)) & (u,v) \text{ an assignment } \mathbf{x}_i := ? \\
\mathcal{F}(v) & \subseteq & [\![p \neq 0]\!]^\sharp(\mathcal{F}(u)) & (u,v) \text{ a guard } p \neq 0
\end{array}
$$

where the effects of assignments and disequality guards onto ideals I are given by:

$$
\begin{array}{lll}
[\![\mathbf{x}_i := p]\!]^\sharp(I) & = & \{q \mid q[p/\mathbf{x}_i] \in I\} \\
[\![\mathbf{x}_i := ?]\!]^\sharp(I) & = & \{\sum_{j=0}^m q_j \mathbf{x}_i^j \mid q_j \in I \cap \mathbb{Q}[\mathbf{X} \backslash \{\mathbf{x}_i\}]\} \\
[\![p \neq 0]\!]^\sharp(I) & = & \{q \mid p \cdot q \in I\}
\end{array}
$$

Intuitively, these definitions can be read as follows. A polynomial identity q is valid after an execution step iff its weakest precondition was valid before the step. For an assignment $\mathbf{x}_i := p$, this weakest precondition equals $q[p/\mathbf{x}_i] = 0$. For a non-deterministic assignment $\mathbf{x}_i := ?$, the weakest precondition of a polynomial $q = \sum_{j=0}^m q_j \mathbf{x}_i^j$ with $q_j \in \mathbb{Q}[\mathbf{X} \backslash \{\mathbf{x}_i\}]$ is given by:

$$\forall\, \mathbf{x}_i.\ q = 0 \quad \equiv \quad q_0 = 0 \wedge \ldots \wedge q_m = 0$$

Finally, for a disequality guard $p \neq 0$, the weakest precondition is given by:

$$\neg(p \neq 0) \vee q = 0 \ \equiv\ p = 0 \vee q = 0 \ \equiv\ p \cdot q = 0$$

Obviously, the operations $[\![\mathbf{x}_i := t]\!]^\sharp$, $[\![\mathbf{x}_i := ?]\!]^\sharp$, and $[\![p \neq 0]\!]^\sharp$ are monotonic. Therefore by the fixpoint theorem of Knaster-Tarski, the constraint system $\mathcal{F}$ has a unique greatest solution over the lattice of ideals of $\mathbb{Q}[\mathbf{X}]$. By definition, the operations commute with arbitrary intersections. Therefore, using standard coincidence theorems for completely distributive intraprocedural dataflow frameworks [10], we conclude:

Theorem 1. *Assume p is a program without procedures. The greatest solution of the constraint system $\mathcal{F}$ for p precisely characterizes at every program point v, the set of all valid polynomial identities.* $\square$

The abstract effect of a disequality guard is readily expressed as an ideal quotient for which effective implementations are well-known. The abstract assignment operations, though, which we have used in the constraint system $\mathcal{F}$ are not very explicit. In order to obtain an effective abstract assignment operation, we intuitively proceed as follows. First, we replace the variable $\mathbf{x}_i$ appearing on the left-hand side of the assignment with a new variable $\mathbf{z}$ both in the ideal I and the right-hand side of the assignment. The variable $\mathbf{z}$ thus represents the value of $\mathbf{x}_i$ *before* the assignment. Then we add the new relationship introduced by the assignment (if there is any) and compute the ideal closure to add all implied polynomial relationships between the variables $\mathbf{X}$ and $\mathbf{z}$. Since the old value of the overwritten variable is no longer accessible, we keep from the implied identities only those between the variables from $\mathbf{X}$. Formally, we verify:

Lemma 1. *For every ideal $I = \langle p_1, \ldots, p_n \rangle \subseteq \mathbb{Q}[\mathbf{X}]$ and polynomial $p \in \mathbb{Q}[\mathbf{X}]$,*

1. $\{q \mid q[p/\mathbf{x}_i] \in I\} \;=\; \langle \mathbf{x}_i - s, s_1, \ldots, s_k \rangle \cap \mathbb{Q}[\mathbf{X}]$ *and*
2. $\{\sum_{j=0}^{m} q_j \mathbf{x}_i^j \mid q_j \in I \cap \mathbb{Q}[\mathbf{X} \backslash \{\mathbf{x}_i\}]\} \;=\; \langle s_1, \ldots, s_n \rangle \cap \mathbb{Q}[\mathbf{X}]$,

where $s = p[\mathbf{z}/\mathbf{x}_i]$ and $s_j = p_j[\mathbf{z}/\mathbf{x}_i]$ for $i = 1, \ldots, n$.

Note that the only extra operation on ideals we use here is the restriction of an ideal to polynomials with variables from a subset. This operation is also called *elimination* and standard effective algorithms are known [1].

Proof. Assume that the ideal I is generated from the polynomials $p_1, \ldots, p_n$. We only prove statement (1). Assume $q = q_0(\mathbf{x}_i - s) + \sum_{j=1}^{n} q_j s_j$ does not contain variable $\mathbf{z}$ where $q_0, \ldots, q_n \in \mathbb{Q}[\mathbf{X} \cup \{\mathbf{z}\}]$, $s = p[\mathbf{z}/\mathbf{x}_i]$ and for all j, $s_j = p_j[\mathbf{z}/\mathbf{x}_i]$. Since the s_j do not contain $\mathbf{x}_i$,

$$q[p/\mathbf{x}_i] = q_0'(p - s) + \sum_{j=1}^{n} q_j' s_j$$

for suitable polynomials $q_0', \ldots, q_n'$. Substituting again $\mathbf{x}_i$ for $\mathbf{z}$ in this equation, we therefore obtain:

$$\begin{aligned} q[p/\mathbf{x}_i] &= q_0''(p - p) + \textstyle\sum_{j=1}^{n} q_j'' p_j \\ &= \textstyle\sum_{j=1}^{n} q_j'' p_j \end{aligned}$$

for suitable polynomials $q_0'', \ldots, q_n''$. Therefore, $q[p/\mathbf{x}_i] \in I$.

For the reverse implication assume $q[p/\mathbf{x}_i] \in I$ which means that $q[p/\mathbf{x}_i] = \sum_{j=1}^{n} q_j p_j$ for suitable polynomials q_j. Substituting $\mathbf{z}$ for $\mathbf{x}_i$ in this equation, therefore gives us $q[s/\mathbf{x}_i] = \sum_{j=1}^{n} q_j' s_j$ for suitable polynomials q_j' where $s = p[\mathbf{z}/\mathbf{x}_i]$ and $s_j = p_j[\mathbf{z}/\mathbf{x}_i]$. Now recall the identity (for $k > 0$):

$$\mathbf{x}_i^k - s^k = g_k \cdot (\mathbf{x}_i - s) \quad \text{for} \quad g_k = \sum_{h=0}^{k-1} \mathbf{x}_i^h s^{k-1-h}$$

and assume that $q = \sum_{k=0}^{d} r_k \mathbf{x}_i^k$ for polynomials $r_k \in \mathbb{Q}[\mathbf{X}\backslash\{\mathbf{x}_i\}]$. Then

$$\begin{array}{lcl} q & = & \sum_{k=0}^{d} r_k \cdot (\mathbf{x}_i^k - s^k) + q[s/\mathbf{x}_i] \\ & = & \sum_{k=1}^{d} r_k g_k \cdot (\mathbf{x}_i - s) + \sum_{j=1}^{n} q'_j s_j \\ & = & q'_0 \cdot (\mathbf{x}_i - s) + \sum_{j=1}^{n} q'_j s_j \qquad \text{for the polynomial} \quad q'_0 = \sum_{k=1}^{d} r_k g_k \ . \end{array}$$

Therefore, $q \in \langle \mathbf{x}_i - s, s_1, \ldots, s_n \rangle$. Since $q \in \mathbb{Q}[\mathbf{X}]$, the assertion follows. □

According to Lemma 1, all operations used in the constraint system $\mathcal{F}$ are effective. Nonetheless, this does not in itself provide us with an analysis algorithm. The reason is that the polynomial ring has *infinite decreasing* chains of ideals. And indeed, simple programs can be constructed where fixpoint iteration will not terminate.

Example 1. Consider our simple example from Fig. 2. There, we obtain the ideal for program point 1 as the infinite intersection:

$$\begin{array}{lcl} \mathcal{F}(1) & = & \langle \mathbf{z}, \mathbf{y} - 1 \rangle \ \cap \\ & & \langle \mathbf{z} - 1, \mathbf{y} - \mathbf{x} \rangle \ \cap \\ & & \langle \mathbf{z} - 1 - \mathbf{x}, \mathbf{y} - \mathbf{x}^2 \rangle \ \cap \\ & & \langle \mathbf{z} - 1 - \mathbf{x} - \mathbf{x}^2, \mathbf{y} - \mathbf{x}^3 \rangle \ \cap \\ & & \ldots \end{array}$$

□

Despite infinitely descending chains, the greatest solution of $\mathcal{F}$ has been determined precisely by Rodriguez-Carbonell et al. [21] — but only for a sub-class of programs. Rodriguez-Carbonell et al. consider simple loops whose bodies consist of a finite non-deterministic choice between sequences of assignments satisfying additional restrictive technical assumptions. No complete methods are known for significantly more general classes of programs. Based on constraint system $\mathcal{F}$, we nonetheless obtain an effective analysis which infers *some* valid polynomial identities by applying *widening* for fix-point acceleration [6]. This idea has been proposed, e.g., by Rodriguez-Carbonell and Kapur [20] and Colon [4]. We will not pursue this idea here. Instead, we propose a different approach.

4 Intraprocedural Analysis: Backward Propagation

The key idea of [12, 14] is this: instead of propagating ideals of valid identities in a forward direction, we start with a conjectured identity $q_t = 0$ at some program point v and compute weakest preconditions for this assertion by backwards propagation. The conjecture is proven if and only if the weakest precondition at program entry $\mathsf{start} = \mathsf{st}_{\mathsf{Main}}$ is true. The assertion true, i.e., the empty conjunction is uniquely represented by the ideal $\{0\}$. Note that it is decidable whether or not a polynomial ideal equals $\{0\}$.

Assignments and disequality guards now induce transformations which for every postcondition, return the corresponding weakest precondition:

$$\begin{array}{lcl} [\![\mathbf{x}_i := p]\!]^{\mathsf{T}}\, q & = & \langle q[p/\mathbf{x}_i] \rangle \\ [\![\mathbf{x}_i := ?]\!]^{\mathsf{T}}\, q & = & \langle q_1, \ldots, q_m \rangle \quad \text{where } q = \sum_{j=0}^{m} q_j \mathbf{x}_i^j \text{ with } q_j \in \mathbb{Q}[\mathbf{X}\backslash\{x_i\}] \\ [\![p \neq 0]\!]^{\mathsf{T}}\, q & = & \langle p \cdot q \rangle \end{array}$$

Note that we have represented the disjunction $p = 0 \vee q = 0$ by $p \cdot q = 0$. Also, we have represented conjunctions of equalities by the ideals generated by the respective polynomials. The definitions of our transformers are readily extended to transformers for ideals, i.e., conjunctions of identities. For a given target program point t and conjecture $q_t = 0$, we therefore can construct a constraint system $\mathcal{B}$:

$$\begin{array}{lll} \mathcal{B}(t) \supseteq \langle q_t \rangle & & \\ \mathcal{B}(u) \supseteq [\![\mathbf{x}_i := p]\!]^\top(\mathcal{B}(v)) & & (u,v) \text{ labeled with } \mathbf{x}_i := p \\ \mathcal{B}(u) \supseteq [\![\mathbf{x}_i := ?]\!]^\top(\mathcal{B}(v)) & & (u,v) \text{ labeled with } \mathbf{x}_i := ? \\ \mathcal{B}(u) \supseteq [\![p \neq 0]\!]^\top(\mathcal{B}(v)) & & (u,v) \text{ labeled with } p \neq 0 \end{array}$$

Since the basic operations are monotonic, the constraint system $\mathcal{B}$ has a unique least solution in the lattice of ideals of $\mathbb{Q}[\mathbf{X}]$. Consider a single execution path π whose effect is described by the partial polynomial transformation $(q_0, [q_1, \ldots, q_k])$. Then the corresponding weakest precondition is given by:

$$[\![\pi]\!]^\top\, p \quad = \quad \langle q_0 \cdot p[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k] \rangle$$

The weakest precondition of p w.r.t. a set of execution paths can be described by the ideal generated by the weakest preconditions for every execution path in the set separately. Since the basic operations in the constraint system $\mathcal{B}$ commute with arbitrary least upper bounds, we once more apply standard coincidence theorems to conclude:

Theorem 2. *Assume p is a polynomial program without procedures and t is a program point of p. Assume the least solution of the constraint system $\mathcal{B}$ for a conjecture $q_t = 0$ at t assigns the ideal I to program point* start*. Then, $q_t = 0$ is valid at t iff $I = \{0\}$.* □

Using a representation of ideals through finite sets of generators, the applications of weakest precondition transformers for edges can be effectively computed. A computation of the least solution of the constraint system $\mathcal{B}$ by standard fixpoint iteration leads to ascending chains of ideals. Therefore, in order to obtain an effective algorithm, we only must assure that *ascending* chains of ideals are ultimately stable. Due to Hilbert's basis theorem, this property indeed holds in polynomial rings over fields (as well as over integral domains like $\mathbb{Z}$). Therefore, the fixpoint characterization of Theorem 2 gives us an effective procedure for deciding whether or not a conjectured polynomial identity is valid at some program point of a polynomial program.

Corollary 1. *In a polynomial program without procedures, it can effectively be checked whether or not a polynomial identity is valid at some target point.* □

Example 2. Consider our example program from Fig. 2. If we want to check the conjecture $\mathbf{z} \cdot \mathbf{x} - \mathbf{z} - \mathbf{y} + 1 = 0$ for program point 1, we obtain:

$$\begin{array}{rl} \mathcal{B}(2) \supseteq & \langle (\mathbf{z} \cdot \mathbf{x} - \mathbf{z} - \mathbf{y} + 1)[\mathbf{y} \cdot \mathbf{x}/\mathbf{y}] \rangle \\ = & \langle \mathbf{z} \cdot \mathbf{x} - \mathbf{z} - \mathbf{y} \cdot \mathbf{x} + 1 \rangle \end{array}$$

Since,

$$(\mathbf{z} \cdot \mathbf{x} - \mathbf{z} - \mathbf{y} \cdot \mathbf{x} + 1)[\mathbf{z} + \mathbf{y}/\mathbf{z}] \;=\; \mathbf{z} \cdot \mathbf{x} - \mathbf{z} - \mathbf{y} + 1$$

the fixpoint is already reached for program points 1 and 2. Thus,

$$\begin{array}{rcl}\mathcal{B}(1) & = & \langle \mathbf{z}\cdot\mathbf{x} - \mathbf{z} - \mathbf{y} + 1\rangle \\ \mathcal{B}(2) & = & \langle \mathbf{z}\cdot\mathbf{x} - \mathbf{z} - \mathbf{y}\cdot\mathbf{x} + 1\rangle\end{array}$$

Moreover,

$$\begin{array}{rcl}\mathcal{B}(0) & = & \langle (\mathbf{z}\cdot\mathbf{x} - \mathbf{z} - \mathbf{y} + 1)[0/\mathbf{z}, 1/\mathbf{y}]\rangle \\ & = & \langle 0\rangle \;=\; \{0\}\end{array}$$

Therefore, the conjecture is proved. □

It seems that the algorithm of testing whether a certain given polynomial identity $p_0 = 0$ is valid at some program point contains no clue on how to infer so far unknown valid polynomial identities. This, however, is not quite true. We show now how to determine all polynomial identities of some arbitrary given form that are valid at a given program point of interest. The form of a polynomial is given by a selection of monomials that may occur in the polynomial.

Let $D \subseteq \mathbb{N}_0^k$ be a finite set of exponent tuples for the variables $x_1, \ldots, x_k$. Then a polynomial q is called a D-polynomial if it contains only monomials $b \cdot \mathbf{x}_1^{i_1} \cdot \ldots \cdot \mathbf{x}_k^{i_k}$, $b \in \mathbb{Q}$, with $(i_1, \ldots, i_k) \in D$, i.e., if it can be written as

$$q = \sum_{\sigma=(i_k,\ldots,i_k)\in D} a_\sigma \cdot \mathbf{x}_1^{i_1} \cdot \ldots \cdot \mathbf{x}_k^{i_k}$$

If, for instance, we choose $D = \{(i_1, \ldots, i_k) \mid i_1 + \ldots + i_k \leq d\}$ for a fixed maximal degree $d \in \mathbb{N}$, then the D-polynomials are all the polynomials up to degree d. Here the *degree* of a polynomial is the maximal degree of a monomial occurring in q where the degree of a monomial $b \cdot \mathbf{x}_1^{i_1} \cdot \ldots \cdot \mathbf{x}_k^{i_k}$, $b \in \mathbb{Q}$, equals $i_1 + \ldots + i_k$.

We introduce a new set of variables $\mathbf{A}_D$ given by:

$$\mathbf{A}_D = \{\mathbf{a}_\sigma \mid \sigma \in D\}\,.$$

Then we introduce the *generic* D-polynomial as

$$q_D = \sum_{\sigma=(i_k,\ldots,i_k)\in D} \mathbf{a}_\sigma \cdot \mathbf{x}_1^{i_1} \cdot \ldots \cdot \mathbf{x}_k^{i_k}\,.$$

The polynomial q_D is an element of the polynomial ring $\mathbb{Q}[\mathbf{X} \cup \mathbf{A}_D]$. Note that every concrete D-polynomial $q \in \mathbb{Q}[\mathbf{X}]$ can be obtained from the generic D-polynomial q_D simply by substituting concrete values $a_\sigma \in \mathbb{Q}$, $\sigma \in D$, for the variables $\mathbf{a}_\sigma$. If $a : \sigma \mapsto a_\sigma$ and $\mathbf{a} : \sigma \mapsto \mathbf{a}_\sigma$, we write $q_D[a/\mathbf{a}]$ for this substitution.

Instead of computing the weakest precondition of each D-polynomial q separately, we may compute the weakest precondition of the single generic polynomial q_D once and for all and substitute the concrete coefficients a_σ of the polynomials q into the precondition of q_D later. Indeed, we show in [14]:

Theorem 3. *Assume p is a polynomial program without procedures and let $\mathcal{B}_D(v)$, v program point of p, be the least solution of the constraint system $\mathcal{B}$ for p with conjecture q_D at target t. Then $q = q_D[a/\mathbf{a}]$ is valid at t iff $q'[a/\mathbf{a}] = 0$ for all $q' \in \mathcal{B}_D(\mathsf{start})$.* □

Clearly, it suffices that $q'[a/\mathbf{a}] = 0$ only for a set of generators of $\mathcal{B}_D(\mathsf{start})$. Still, this does not immediately give us an effective method of determining all suitable coefficient vectors, since the precise set of solutions of arbitrary polynomial equation systems are not computable. We observe, however, in [14]:

Lemma 2. *Every ideal $\mathcal{B}_D(u)$, u a program point, of the least solution of the abstract constraint system $\mathcal{B}$ for conjecture q_D at some target node t is generated by a finite set G of polynomials q where each variable $\mathbf{a}_\sigma$ occurs only with degree at most 1. Moreover, such a generator set can be effectively computed.* □

Thus, the set of (coefficient maps) of D-polynomials which are valid at our target program point t can be characterized as the set of solutions of a *linear* equation system. Such equation systems can be algorithmically solved, i.e., finite representations of their sets of solutions can be constructed explicitly, e.g., by Gaussian elimination. We conclude:

Theorem 4. *For a polynomial program p without procedures and a program point t in p, the set of all D-polynomials which are valid at t can be effectively computed.* □

As a side remark, we should mention that instead of working with the larger polynomial ring $\mathbb{Q}[\mathbf{X} \cup \mathbf{A}_D]$, we could work with *modules* over the polynomial ring $\mathbb{Q}[\mathbf{X}]$ consisting of vectors of polynomials whose entries are indexed with $\sigma \in D$. The operations on modules turn out to be practically much faster than corresponding operations on the larger polynomial ring itself, see [18] for a practical implementation and preliminary experimental results.

5 Interprocedural Analysis: Transition Invariants

The main question of precise interprocedural analysis is this: how can the effects of procedure calls be finitely described? An interesting idea (essentially) due to Colon [4] is to represent effects by polynomial *transition invariants*. This means that we introduce a separate copy $\mathbf{X}' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_k\}$ of variables denoting the values of variables before the execution. Then we use polynomials to express possible relationships between pre- and post-states of the execution. Obviously, all such valid relationships again form an ideal, now in the polynomial ring $\mathbb{Q}[\mathbf{X} \cup \mathbf{X}']$.

The transformation ideals for assignments, non-deterministic assignments and disequality guards are readily expressed by:

$$\begin{array}{lcl} [\![\mathbf{x}_i := p]\!]^{\sharp\sharp} & = & \langle \{\mathbf{x}_j - \mathbf{x}'_j \mid j \neq i\} \cup \{\mathbf{x}_i - p[\mathbf{x}'/\mathbf{x}]\} \rangle \\ [\![\mathbf{x}_i := ?]\!]^{\sharp\sharp} & = & \langle \{\mathbf{x}_j - \mathbf{x}'_j \mid j \neq i\} \rangle \\ [\![p \neq 0]\!]^{\sharp\sharp} & = & \langle \{p[\mathbf{x}'/\mathbf{x}] \cdot (\mathbf{x}_j - \mathbf{x}'_j) \mid j = 1, \ldots, k\} \rangle \end{array}$$

In particular, the last definition means that either the guard is wrong before the transition or the states before and after the transition are equal. The basic effects can be composed to obtain the effects of larger program fragments by means of a composition operation "$\circ$". Composition on transition invariants can be defined by:

$$I_1 \circ I_2 \; = \; (I_1[\mathbf{y}/\mathbf{x}'] \oplus I_2[\mathbf{y}/\mathbf{x}]) \cap \mathbb{Q}[\mathbf{X} \cup \mathbf{X}']$$

where a fresh set $\mathbf{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$ is used to store the intermediate values between the two transitions represented by I_1 and I_2 and the postfix operator $[\mathbf{y}/\mathbf{x}]$ denotes renaming of variables in $\mathbf{X}$ with their corresponding copies in $\mathbf{Y}$. Note that "$\circ$" is defined by means of well-known effective ideal operations. Using this operation, we can put up a constraint system $\mathcal{T}$ for ideals of polynomial transition invariants of procedures:

$$\begin{array}{lcll}
\mathcal{T}(v) & \subseteq & \langle \mathbf{x}_i - \mathbf{x}_i' \mid i = 1, \ldots, k\rangle & v \text{ is entry point} \\
\mathcal{T}(v) & \subseteq & [\![\mathbf{x}_i := p]\!]^{\sharp\sharp} \circ \mathcal{T}(u) & (u,v) \text{ is labeled with } \mathbf{x}_i := p \\
\mathcal{T}(v) & \subseteq & [\![\mathbf{x}_i := ?]\!]^{\sharp\sharp} \circ \mathcal{T}(u) & (u,v) \text{ is labeled with } \mathbf{x}_i := ? \\
\mathcal{T}(v) & \subseteq & [\![p \neq 0]\!]^{\sharp\sharp} \circ \mathcal{T}(u) & (u,v) \text{ is labeled with } p \neq 0 \\
\mathcal{T}(v) & \subseteq & \mathcal{T}(f) \circ \mathcal{T}(u) & (u,v) \text{ calls } f \\
\mathcal{T}(f) & \subseteq & \mathcal{T}(v) & v \text{ exit point of } f
\end{array}$$

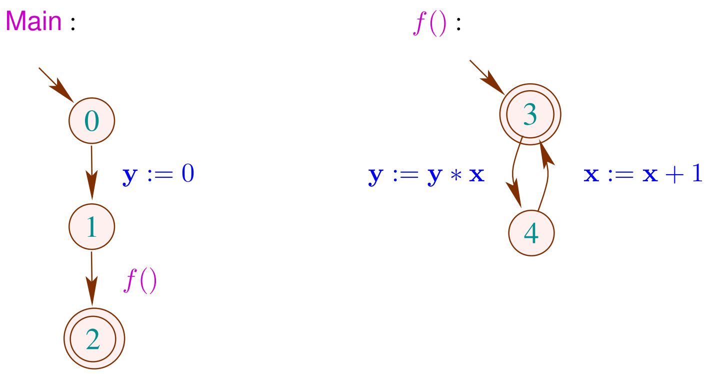

Fig. 3. A simple program with procedures.

Example 3. Consider the program from Fig. 3. We calculate:

$$\begin{array}{rcl}
\mathcal{T}(f) & = & \langle \mathbf{x} - \mathbf{x}', \mathbf{y} - \mathbf{y}' \rangle \ \cap \\
& & \langle \mathbf{x} - \mathbf{x}' - 1, \mathbf{y} - \mathbf{y}' \cdot \mathbf{x}' \rangle \ \cap \\
& & \langle \mathbf{x} - \mathbf{x}' - 2, \mathbf{y} - \mathbf{y}' \cdot \mathbf{x}' \cdot (\mathbf{x}' + 1) \rangle \ \cap \\
& & \ldots \\
& = & \langle 0 \rangle
\end{array}$$

Using this invariant for analyzing the procedure main, we only find the trivial transition invariant 0. On the other hand, we may instead *inline* the procedure f as in Fig. 4. A

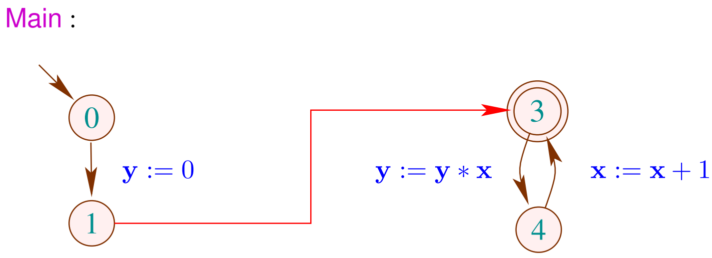

Fig. 4. The inlined version of the example program.

corresponding calculation of the transition invariant of main yields:

$$\begin{array}{rcl} \mathcal{T}(\mathsf{Main}) & = & \langle \mathbf{x} - \mathbf{x}', \mathbf{y} \rangle \ \cap \\ & & \langle \mathbf{x} - \mathbf{x}' - 1, \mathbf{y} \rangle \ \cap \\ & & \langle \mathbf{x} - \mathbf{x}' - 2, \mathbf{y} \rangle \ \cap \\ & & \ldots \\ & = & \langle \mathbf{y} \rangle \end{array}$$

Thus for this analysis, inlining may gain precision. □

Clearly, using transition invariants incurs the same problem as forward propagation for intraprocedural analysis, namely, that fixpoint iteration may result in infinite decreasing chains of ideals. Our minimal example exhibited two more problems, namely that the composition operation is not *continuous*, i.e., does not commute with greatest lower bounds of descending chains in the second argument, and also that a less compositional analysis through inlining may infer more valid transition invariants.

To be fair here, it should be noted that Colon did not propose to use *ideals* for representing transition invariants. Colon instead considered *pseudo*-ideals, i.e., ideals where polynomials are considered only up to a given degree bound. This kind of further abstraction solves the problems of infinite decreasing chains as well as missing continuity — at the expense, though, of further loss in precision. Colon's approach, for example, fails to find a nontrivial invariant in the example program from Fig. 3 for Main.

6 Interprocedural Analysis: Backward Propagation

Due to the apparent weaknesses of the approach through polynomials as transition invariants, we propose to represent effects of procedures by pre-conditions of generic polynomials. Procedure calls are then dealt with through instantiation of generic coefficients. Thus, effects are still described by ideals — over a larger set of variables (or by modules; see the discussion at the end of Section 4). Suppose we have chosen some finite set $D \subseteq \mathbb{N}_0^k$ of exponent tuples and assume that the polynomial $p = p_D[a/\mathbf{a}]$ is the D-polynomial that is obtained from the generic D-polynomial through instantiation of the generic coefficients with a. Assume further that the effect of some procedure call is given by the ideal $I \subseteq \mathbb{Q}[\mathbf{X} \cup \mathbf{A}_D] = \langle q_1, \ldots, q_m \rangle$. Then we determine a precondition of $p = 0$ w.r.t. to the call by:

$$I \ (p) \ = \ \langle q_1[a/\mathbf{a}], \ldots, q_m[a/\mathbf{a}] \rangle$$

This definition is readily extended to ideals I' generated by D-polynomials. There is no guarantee, though, that all ideals that occur at the target program points v of call edges (u, v) will be generated by D-polynomials. In fact, simple examples can be constructed where no uniform set D of exponent tuples can be given. Therefore, we additionally propose to use an abstraction operator $\mathbf{W}$ that splits polynomials appearing as post-condition of procedure calls which are not D-polynomials.

We choose a maximal degree d_j for each program variable x_j and let

$$D = \{(i_1, \ldots, i_k) \mid i_j \leq d_j \text{ for } i = 1, \ldots, k\}$$

The abstraction operator $\mathbf{W}$ takes generators of an ideal I and maps them to generators of a (possibly) larger ideal $\mathbf{W}(I)$ which is generated by D-polynomials. In order to construct such an ideal, we need a heuristics which decomposes an arbitrary polynomial q into a linear combination of D-polynomials $q_1, \ldots, q_m$:

$$q = r_1 q_1 + \ldots + r_m q_m \tag{1}$$

We could, for example, decompose q according to the first variable:

$$q = q_0' + \mathbf{x}_1^{d_1+1} \cdot q_1' + \ldots + \mathbf{x}_1^{s(d_1+1)} \cdot q_s'$$

where each q_i' contains powers of $\mathbf{x}_1$ only up to degree d_1 and repeat this decomposition with the polynomials q_i' for the remaining variables. Given a decomposition (1), we have $q \in \langle q_1, \ldots, q_m \rangle$. Therefore, we can replace every generator of I by D-polynomials in order to obtain an ideal $\mathbf{W}(I)$ with the desired properties.

We use the new application operator as well as the abstraction operator $\mathbf{W}$ to generalize our constraint system $\mathcal{B}$ to a constraint system $\mathcal{E}$ for the effects of procedures:

$$\begin{array}{lcll}
\mathcal{E}(u) & \supseteq & \langle q_D \rangle & u \text{ is exit point} \\
\mathcal{E}(u) & \supseteq & [\![\mathbf{x}_i := p]\!]^\top(\mathcal{E}(v)) & (u, v) \text{ labeled with } \mathbf{x}_i := p \\
\mathcal{E}(u) & \supseteq & [\![\mathbf{x}_i := ?]\!]^\top(\mathcal{E}(v)) & (u, v) \text{ labeled with } \mathbf{x}_i := ? \\
\mathcal{E}(u) & \supseteq & [\![p \neq 0]\!]^\top(\mathcal{E}(v)) & (u, v) \text{ labeled with } p \neq 0 \\
\mathcal{E}(u) & \supseteq & \mathcal{E}(f)(\mathbf{W}(\mathcal{E}(v))) & (u, v) \text{ calls } f \\
\mathcal{E}(f) & \supseteq & \mathcal{E}(u) & u \text{ entry point of } f
\end{array}$$

Example 4. Consider again the example program from Fig. 3. Let us choose $d = 1$ where $p_1 = \mathbf{ay} + \mathbf{bx} + \mathbf{c}$. Then we calculate for f:

$$\begin{array}{rcl}
\mathcal{E}(f) & = & \langle \mathbf{ay} + \mathbf{bx} + \mathbf{c} \rangle \ \oplus \\
& & \langle \mathbf{ayx} + \mathbf{b}(\mathbf{x}+1) + \mathbf{c} \rangle \ \oplus \\
& & \langle \mathbf{ayx}(\mathbf{x}+1) + \mathbf{b}(\mathbf{x}+2) + \mathbf{c} \rangle \ \oplus \\
& & \langle \mathbf{ayx}(\mathbf{x}+1)(\mathbf{x}+2) + \mathbf{b}(\mathbf{x}+3) + \mathbf{c} \rangle \ \oplus \\
& & \ldots \\
& = & \langle \mathbf{ay}, \mathbf{b}, \mathbf{c} \rangle
\end{array}$$

This description tells us that for a linear identity $\mathbf{a}\mathbf{y}+\mathbf{b}\mathbf{x}+\mathbf{c} = 0$ to be valid after a call to f, the coefficients $\mathbf{b}$ and $\mathbf{c}$ necessarily must equal 0. Moreover, either coefficient $\mathbf{a}$ equals 0 (implying that the whole identity is trivial) or $\mathbf{y} = 0$. Indeed, this is the optimal description of the behavior of f with polynomials. $\square$

The effects of procedures as approximated by constraint system $\mathcal{E}$ can be used to check a polynomial conjecture q_t at a given target node t along the lines of constraint system $\mathcal{B}$. We only have to extend it by extra constraints dealing with function calls. Thus, we put up the following constraint system:

$$\begin{array}{rcll}
\mathcal{R}(t) & \supseteq & \langle q_t \rangle & \\
\mathcal{R}(u) & \supseteq & [\![\mathbf{x}_i := p]\!]^{\mathsf{T}}(\mathcal{R}(v)) & (u,v) \text{ labeled with } \mathbf{x}_i := p \\
\mathcal{R}(u) & \supseteq & [\![\mathbf{x}_i := ?]\!]^{\mathsf{T}}(\mathcal{R}(v)) & (u,v) \text{ labeled with } \mathbf{x}_i := ? \\
\mathcal{R}(u) & \supseteq & [\![p \neq 0]\!]^{\mathsf{T}}(\mathcal{R}(v)) & (u,v) \text{ labeled with } p \neq 0 \\
\mathcal{R}(u) & \supseteq & \mathcal{E}(f)(\mathbf{W}(\mathcal{R}(v))) & (u,v) \text{ calls } f \\
\mathcal{R}(f) & \supseteq & \mathcal{R}(u) & u \text{ entry point of } f \\
\mathcal{R}(u) & \supseteq & \mathcal{R}(f) & (u,_) \text{ calls } f
\end{array}$$

This constraint system again has a least solution which can be computed by standard fixpoint iteration. Summarizing, we obtain the following theorem:

Theorem 5. *Assume p is a polynomial program with procedures. Assume further that we assert a conjecture q_t at program point t.*

Safety: *1. For every procedure f, the ideal $\mathcal{E}(f)$ represents a precondition of the identity $p_D = 0$ after the call.*
2. If the ideal $\mathcal{R}(\mathsf{Main})$ equals $\{0\}$, then the conjecture q_t is valid at t.

Completeness: *If during fixpoint computation, all ideals at target program points v of call edges (u, v) are represented by D-polynomials as generators, the conjecture is valid only if the ideal $\mathcal{R}(\mathsf{Main})$ equals $\{0\}$.*

The safety-part of Theorem 5 tells us that our analysis will never assure a wrong conjecture but may fail to certify a conjecture although it is valid. According to the completeness-part, however, the analysis algorithm provides slightly more information: if no approximation steps are necessary at procedure calls, the analysis is precise. For simplicity, we have formulated Theorem 5 in such a way that it only speaks about checking conjectures. In order to infer valid polynomial identities up to a specified degree bound, we again can proceed analogous to the intraprocedural case by considering a generic postcondition in constraint system $\mathcal{R}$.

7 Equality Guards

In this section, we discuss methods for dealing with equality guards $p = 0$. Recall, that in presence of equality guards, the question whether a variable is constantly 0 at a program point or not is undecidable even in absence of procedures and with affine assignments only. Thus, we cannot hope for complete methods here. Still, in practical contexts, equality guards are a major source of information about values of variables.

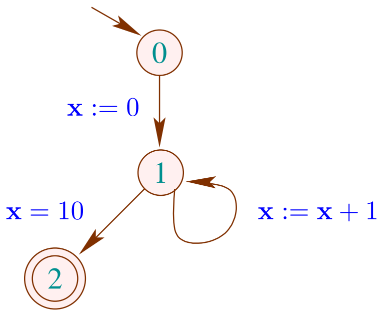

Fig. 5. A simple for-loop.

Consider, e.g., the control flow graph from Fig. 5. Then, according to the equality guard, we definitely know that $\mathbf{x} = 10$ whenever program point 2 is reached. In order to deal with equality guards, we thus extend forward analysis by the constraints:

$$\mathcal{F}(v) \subseteq [\![p = 0]\!]^\sharp(\mathcal{F}(u)) \qquad (u, v) \text{ labeled with } p = 0$$

where the effect of an equality guard is given by:

$$[\![p = 0]\!]^\sharp\ I \;=\; I \oplus \langle p \rangle$$

This formalizes our intuition that after the guard, we additionally know that $p = 0$ holds. Such an approximate treatment of equality guards is common in forward program analysis and already proposed by Karr [11]. A similar extension is also possible for inferring transition invariants. The new effect is monotonic. However, it is, no longer distributive, i.e., it does not commute with intersections. Due to monotonicity, the extended constraint systems $\mathcal{F}$ as well as $\mathcal{T}$ still have greatest solutions which provide safe approximations of the sets of all valid invariants and transition invariants in presence of equality guards, respectively.

Example 5. Consider the program from Fig. 5. For program point 1 we have:

$$\begin{aligned}\mathcal{F}(1) &= \langle \mathbf{x} \rangle \cap \langle \mathbf{x} - 1 \rangle \cap \langle \mathbf{x} - 2 \rangle \cap \ldots \\ &= \{0\}\end{aligned}$$

Accordingly, we find for program point 2,

$$\begin{aligned}\mathcal{F}(2) &= \{0\} \oplus \langle \mathbf{x} - 10 \rangle \\ &= \langle \mathbf{x} - 10 \rangle\end{aligned}$$

Thus, given the lower bound $\{0\}$ for the infinite decreasing chain of program point 1, we arrive at the desired result for program point 2. □

It would be nice if also backward analysis could be extended with some approximate method for equality guards. Our idea for such an extension is based on *Lagrange multipliers*. Recall that the *weakest* precondition for validity of $q = 0$ after a guard $p = 0$ is given by:

$$(p = 0) \Rightarrow (q = 0)$$

which, for every λ, is implied by:

$$q + \lambda \cdot p = 0$$

The value λ is called a *Lagrange*-multiplier and can be arbitrarily chosen. We remark that a related technique has been proposed in [5] for inferring parametric program invariants. Thus, we define:

$$[\![p = 0]\!]^{\mathsf{T}}(q) = \langle q + p \cdot \lambda \rangle \tag{2}$$

where a different formal multiplier λ is chosen for every occurrence of an equality guard. Similar to the treatment of generic postconditions, the parameters λ will occur linearly in a suitably chosen set of generators for the precondition ideal at program start where they can be chosen appropriately.

Example 6. Again consider the program from Fig. 5 and assume that we are interested in identities up to degree 1 at the exit point of the program. Thus we start with the generic polynomial $\mathbf{a}\mathbf{x} + \mathbf{b} = 0$ at node 2. This gives us for program point 1:

$$\begin{aligned} \mathcal{B}_1(1) &= \langle (\mathbf{a} + \lambda) \cdot \mathbf{x} + \mathbf{b} - 10\lambda \rangle \ \oplus \\ &\quad\ \langle (\mathbf{a} + \lambda) \cdot \mathbf{x} + \mathbf{a} + \lambda + \mathbf{b} - 10\lambda \rangle \\ &= \langle \mathbf{a} + \lambda, \mathbf{b} - 10\lambda \rangle \end{aligned}$$

Choosing $\lambda = -\mathbf{a}$, we obtain $\mathbf{b} = -10\mathbf{a}$. Therefore all multiples of the polynomial $\mathbf{x} - 10$ are valid identities for program point 2. $\square$

Instead of using a single variable λ as a Lagrange multipliers we could also use an entire polynomial. This means that we use in (2) a generic polynomial q_D (for some set D of exponent tuples) instead of the variable λ for each equality guard $p = 0$:

$$[\![p = 0]\!]^{\mathsf{T}}(q) = \langle q + p \cdot q_D \lambda \rangle$$

where we use new variables $A_D = \{\mathbf{a}_\sigma \mid \sigma \in D\}$ in q_D for each equality guard. Now, all the variables in A_D can be adjusted in the computed weakest precondition. This may lead to more precise results – at the price of a more expensive analysis.

8 Discussion

We have summarized forward and backward iteration methods for inferring valid polynomial identities. In absence of procedure calls, we arrived at a rather clear picture: we exhibited a finite constraint system which precisely characterizes the set of all valid polynomial identities in a polynomial program. Due to possibly infinite decreasing chains of ideals, it is currently open whether the greatest solution of this constraint system can effectively be computed. On the other hand, backward analysis based on weakest precondition computations relies on increasing chains of ideals — allowing us to decide whether any given conjecture at a program point is valid. Also, this enables us to effectively find all valid polynomial identities up to a given degree.

In presence of procedure calls, the picture is less clear. The natural extension of the intraprocedural forward propagation suggests to use ideals of polynomial transition invariants to describe effects of procedures. The composition operation for such ideals, though, turned out to be non-continuous. Also, our example shows that using polynomial transition invariants may not be precise, i.e., may miss some valid polynomial identities. Therefore, we considered a generalization of backward analysis which describes effects of procedures by means of preconditions of generic polynomials. Here, we obtained a precise finite characterization of identities of some given form only if in the polynomials occurring during the analysis at procedure exits the degrees of the variables are bounded. Note that this approach can be considered as a smooth generalization of the methods in [15] for affine programs where all occurring polynomials are known to have bounded degree.

It still remains open whether precise techniques can be found for lifting the degree bound in the general intraprocedural case. It is also unclear how to deal with recursive programs precisely if the degrees of weakest preconditions grow arbitrarily.

References

1. T. Becker and V. Weispfenning. *Gröbner Bases*. Springer-Verlag, 1993.
2. S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful Techniques for the Automatic Generation of Invariants. In *8th Int. Conf. on Computer Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*. Springer, 1996.
3. N. Bjørner, A. Browne, and Z. Manna. Automatic Generation of Invariants and Intermediate Assertions. *Theoretical Computer Science*, 173(1):49–87–, 1997.
4. M. Colon. Approximating the Algebraic Relational Semantics of Imperative Programs. In *11th Int. Symp. on Static Analysis (SAS)*, pages 296–311. Springer-Verlag, LNCS 3146, 2004.
5. P. Cousot. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming . In *Verification, Model Checking and Abstract Interpretation (VMCAI)*, pages 1–24. Springer-Verlag, LNCS 3385, 2005.
6. P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Recursive Procedures. In E. Neuhold, editor, *IFIP Conf. on Formal Description of Programming Concepts*, pages 237–277. North-Holland, 1977.
7. S. Gulwani and G. Necula. Discovering Affine Equalities Using Random Interpretation. In *30th ACM Symp. on Principles of Programming Languages (POPL)*, pages 74–84, 2003.
8. S. Gulwani and G. Necula. Precise Interprocedural Analysis Using Random Interpretation. In *32th Ann. ACM Symp. on Principles of Programming Languages (POPL)*, pages 324–337, 2005.
9. M. S. Hecht. *Flow Analysis of Computer Programs*. Elsevier North-Holland, 1977.
10. J. Kam and J. Ullman. Global Data Flow Analysis and Iterative Algorithms. *Journal of the ACM (JACM)*, 23(1):158–171, 1976.
11. M. Karr. Affine Relationships Among Variables of a Program. *Acta Informatica*, 6:133–151, 1976.

12. M. Müller-Olm and H. Seidl. Polynomial Constants are Decidable. In *9th Static Analysis Symposium (SAS)*, pages 4–19. LNCS 2477, Springer-Verlag, 2002.
13. M. Müller-Olm and H. Seidl. A Note on Karr's Algorithm. In *31st Int. Coll. on Automata, Languages and Programming (ICALP)*, pages 1016–1028. Springer Verlag, LNCS 3142, 2004.
14. M. Müller-Olm and H. Seidl. Computing Polynomial Program Invariants. *Information Processing Letters (IPL)*, 91(5):233–244, 2004.
15. M. Müller-Olm and H. Seidl. Precise Interprocedural Analysis through Linear Algebra. In *31st ACM Symp. on Principles of Programming Languages (POPL)*, pages 330–341, 2004.
16. M. Müller-Olm and H. Seidl. A Generic Framework for Interprocedural Analysis of Numerical Properties. In *12th Static Analysis Symposium (SAS)*, pages 235–250. LNCS 3672, Springer-Verlag, 2005.
17. M. Müller-Olm and H. Seidl. Analysis of Modular Arithmetic. In *European Symposium on Programming (ESOP)*, pages 46–60. Springer Verlag, LNCS 3444, 2005.
18. M. Petter. Berechnung von polynomiellen Invarianten, 2004. Diploma Thesis.
19. J. R. Reif and H. R. Lewis. Symbolic Evaluation and the Global Value Graph. In *4th ACM Symp. on Principles of Programming Languages POPL'77*, pages 104–118, 1977.
20. E. Rodriguez-Carbonell and D. Kapur. An Abstract Interpretation Approach for Automatic Generation of Polynomial Invariants. In *11th Int. Symp. on Static Analysis (SAS)*, pages 280–295. Springer-Verlag, LNCS 3146, 2004.
21. E. Rodriguez-Carbonell and D. Kapur. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In *Int. ACM Symposium on Symbolic and Algebraic Computation 2004 (ISSAC04)*, pages 266–273, 2004.
22. S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Non-linear Loop Invariant Generation using Gröbner Bases. In *ACM Symp. on Principles of Programming Languages (POPL)*, pages 318–329, 2004.