Computing Polynomial Program Invariants

Markus Müller-Olm *
FernUniversität Hagen, LG Praktische Informatik 5
58084 Hagen, Germany
e-mail: mmo@ls5.cs.uni-dortmund.de

Helmut Seidl
TU München, Informatik, I2
85748 München, Germany
e-mail: seidl@informatik.tu-muenchen.de

Keywords: program analysis, polynomial relation, abstract interpretation, computable algebra, program correctness

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., [4, 3]). In this paper we present analyses that check and find valid polynomial relations in programs. A polynomial relation 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 $\mathbf{x}_1, \ldots, \mathbf{x}_k$.[1] Our analyses combine techniques from abstract interpretation and computable algebra and fully interpret assignment statements with *polynomial* expressions on the right hand sides while considering other assignments as non-deterministic. Polynomial disequality guards are also treated precisely[2] while other conditions at branches are ignored. The first analysis automatically checks whether a polynomial relation holds among the program variables whenever control reaches a given target program point. Our second analysis extends this testing procedure to compute precisely the set of all *polynomial relations of an arbitrary given form*, e.g., all polynomial relation of bounded degree that are valid at the target program point among the program variables under the above abstraction.

The following is known as an undecidable problem in non-deterministic flow graphs, if the full standard signature of arithmetic operators (addition, subtraction, multiplication, and division) is available [7, 13]: decide whether a given variable holds a constant value at a given program point in all executions. Clearly, constancy of a variable is a polynomial relation: $\mathbf{x}$ is a constant at program point n if and only if the polynomial relation $\mathbf{x} - c = 0$ is valid at n for some $c \in \mathbb{F}$. Moreover, with polynomials we can write all expressions involving addition, subtraction, and multiplication. Thus, our result allows us to find constants in non-deterministic flow graphs in which just these three operators are used ("*polynomial constants*") and indicates that division is the real cause of undecidability of constant propagation.

The current paper extends and simplifies an earlier conference paper [12] that considered just detection of polynomial *constants*. We improve over this paper in a number of respects:

- A modest generalization is that we show how to check validity of *arbitrary* polynomial relations (Section 3), while in [12] only particular polynomial relations of the form $\mathbf{x} - c = 0$, $c \in \mathbb{F}$, were checked for validity. While checking arbitrary polynomial relations can be done with essentially the same technique this was not made explicit in [12].
- We treat polynomial disequality guards.
- Most importantly, we are now able not just to *check* polynomial relations but to *derive* all valid polynomial relations of some given form (Section 4). Without a systematic way of derivation, we must guess candidate relations by some heuristic or ad-hoc method. In [12], for instance, the constant c for the candidate relation $\mathbf{x} - c = 0$ is determined in an ad-hoc way by executing a single program path.

The main idea of our checking algorithm is to compute a polynomial ideal that represents the weakest precondition

*On leave from Universität Dortmund.

[1] More generally our analyses can handle positive Boolean combinations of polynomial relations.

[2] Again, positive Boolean combinations of such guards can be handled.

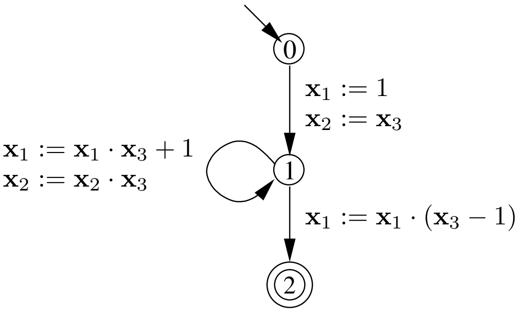

Figure 1. An example program.

for the validity of the given polynomial relation at the given target program point. We rely on results from computable algebra in order to ensure that this computation can be done effectively, most notably, on Hilbert's Basis Theorem and on Buchberger's Algorithm. The polynomial relation in question is valid at the target program point if and only if the computed weakest precondition is valid for all states. The latter can easily be checked.

In the case of derivation, we compute the weakest precondition of a *generic polynomial relation* at the target program point. In this generic relation coefficients are replaced by variables. Again an ideal that represents the weakest-precondition can be computed effectively. We can then characterize the set of values of the new variables for which the weakest precondition is universally valid by means of a linear equation system. The space of solutions of this linear equation system characterizes the coefficients of all polynomial relations (of the given form) which are valid at the target program point.

Looking for valid polynomial relations is a rather general question with many applications. First of all, many classical data flow analysis problems can be seen as problems about polynomial relations. 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 relations found by an automatic analysis are also useful in program verification contexts, as they provide non-trivial valid assertions about the program. In particular, loop invariants can be discovered fully automatically. As polynomial relations express quite complex relationships among variables, the discovered assertions may form the backbone of the program proof and thus significantly simplify the verification task.

As an illustration of the kind of programs and properties our analysis can handle, consider the program in Figure 1. After initializing $\mathbf{x}_1$ with 1 and $\mathbf{x}_2$ with $\mathbf{x}_3$, the program iteratively executes the two assignments $\mathbf{x}_1 := \mathbf{x}_1 \cdot \mathbf{x}_3 + 1; \mathbf{x}_2 := \mathbf{x}_2 \cdot \mathbf{x}_3$ in sequence. It is not difficult to see that after n iterations of the loop, $\mathbf{x}_1$ holds the value $\sum_{i=0}^{n} q^i = \frac{q^{n+1}-1}{q-1}$ (computed by Horner's method) and $\mathbf{x}_2$ holds the value q^{n+1} if q is the (unknown) initial value of $\mathbf{x}_3$. Therefore, after leaving the loop and multiplying $\mathbf{x}_1$ with $\mathbf{x}_3 - 1$ (i.e., the value $q - 1$), we can easily convince ourselves that the equation: $\mathbf{x}_1 - \mathbf{x}_2 + 1 = 0$ holds at program point 2.

2. Polynomial Programs

We model programs by non-deterministic flow graphs as in Figure 1. Let $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ be the set of (global) variables the program operates on. We use $\mathbf{x}$ to denote the vector of variables $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_k)$. We assume that the variables take values in a fixed field $\mathbb{F}$. If $\mathbb{F}$ is finite, we can effectively compute for each program point the possible run-time states by an effective fixpoint computation. From this information we can clearly check validity of polynomial relations and derive valid polynomial relations. Therefore, we assume without loss of generality that $\mathbb{F}$ is infinite. In practice, $\mathbb{F}$ is the field of rational numbers.

A *state* assigning values to the variables is conveniently modeled by a k-dimensional vector $x = (x_1, \ldots, x_k) \in \mathbb{F}^k$; x_i is the value assigned to variable $\mathbf{x}_i$. Note that we distinguish variables and their values by using a different font.

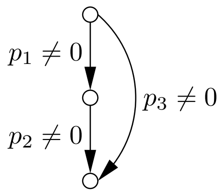

Figure 2. Representation of guard $(p_1 \neq 0 \wedge p_2 \neq 0) \vee p_3 \neq 0$ **by a flow graph.**

We assume that the basic statements in the program are either *polynomial assignments* of the form $\mathbf{x}_j := p$ or *non-deterministic assignments* of the form $\mathbf{x}_j :=?$ where $\mathbf{x}_j \in \mathbf{X}$ and p is a polynomial in $\mathbb{F}[\mathbf{X}]$, the polynomial ring over the field $\mathbb{F}$ with variables from $\mathbf{X}$. Moreover, and in generalization of [12], we allow *polynomial disequality guards* which are of the form $p \neq 0$ where p again is a polynomial. While we allow only single negated polynomial equations as guards, positive Boolean combinations can be treated by coding them as small flow graphs, see Figure 2

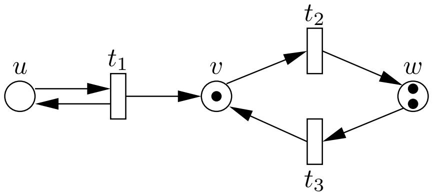

Figure 3. A Petri Net.

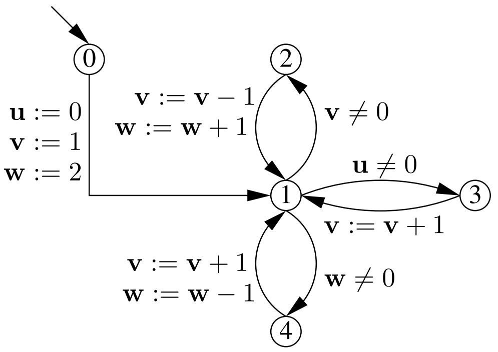

Figure 4. Coding of the Petri Net.

for an illustrative example. Assignments $\mathbf{x}_j := \mathbf{x}_j$ have no effect onto the program state. They are also called skip statements and omitted in pictures. Non-deterministic assignments $\mathbf{x}_j :=?$ represent a safe abstraction of statements in a source program our analysis cannot handle precisely, for example of assignments $\mathbf{x}_j := t$ with non-polynomial expressions t or of read statements $\mathsf{read}(\mathbf{x}_j)$. Similarly, skip statements can be used to abstract guards that are not polynomial disequalities. Let Lab be the set of basic statements and polynomial disequality guards.

A *polynomial program* is given by a *control flow graph* $G = (N, E, A, s)$ that consists of:

- a set N of *program points*;
- a set of edges $E \subseteq N \times N$;
- a mapping $A : E \rightarrow \mathsf{Lab}$ that annotates each edge with a basic statement or polynomial disequality guard;
- a special *entry (or start) point* $\mathsf{s} \in N$.

Note that we cannot allow polynomial equality guards instead of (or even in addition to) disequality guards. As shown in [15] constancy detection is undecidable already for flow graphs with affine assignments and affine equality guards. This clearly implies that polynomial relations cannot be checked or derived completely in programs with polynomial assignments and polynomial equality guards.

Besides being a smooth abstraction of usual programs for which our analyses are complete, polynomial programs are also of interest in their own right. They can, for instance, code Petri nets. As an example consider the Petri net in Fig. 3 with three places u, v, w and three transitions t_1, t_2, t_3. The coding of this net by a polynomial program is shown in Fig. 4. In the coding there is a variable for each place of the Petri net. In the initialization we store the value of the initial marking into these variables (transition from program point 0 to 1). For each transition in the Petri net, there is a loop from program point 1 to itself; the topmost loop in Fig. 4, for instance, corresponds to transition t_2 of the net in Fig. 3. The loop for a transition first checks (by disequality guards) that the transition is enabled and then mirrors the effect of the transition on the marking by appropriate assignments to the variables. It is easy to see that the possible run-time states at program point 1 correspond just to the reachable markings of the Petri net. On the resulting polynomial program we can check and derive polynomial invariants at program point 1 which are valid invariants for all reachable markings of the Petri net. For example $\mathbf{v} + \mathbf{w} = 3$ is valid at program point 1 in Fig. 4. Indeed, as our procedures are complete for polynomial programs, they induce complete procedures for checking or deriving polynomial invariants for the reachable markings of Petri nets. Note that disequality guards are crucial for faithful coding of Petri nets. If, for instance, the guard $\mathbf{u} \neq 0$ is ignored in the program in Fig. 4, the invariant $\mathbf{v} + \mathbf{w} = 3$ is no longer valid at program point 1.

The core part of our algorithm can be understood as a precise abstract interpretation of a constraint system characterizing the program executions that reach a given target program point $t \in N$. We represent program executions or *runs* by finite sequences

$$r \equiv r_1; \ldots ; r_m$$

where each r_i is of the form $p \neq 0$ or $\mathbf{x}_j := p$ where $\mathbf{x}_j \in \mathbf{X}$ and $p \in \mathbb{F}[\mathbf{X}]$. We write Runs for the set of runs. The set of runs *reaching* t *from some program point* $u \in N$ can be characterized as the least solution of a system of subset constraints on run sets. We start by defining the program executions of base edges e in isolation. If e is annotated by a guard, i.e., $A(e) \equiv p \neq 0$, or a polynomial assignment, i.e., $A(e) \equiv \mathbf{x}_j := p$, it gives rise to a single execution: $\mathbf{R}(e) = \{A(e)\}$. The effect of base edges e annotated by a non-deterministic assignment $\mathbf{x}_j :=?$ is captured by all runs that assign some value from $\mathbb{F}$ to $\mathbf{x}_j$:

$$\mathbf{R}(e) = \{\mathbf{x}_j := c \mid c \in \mathbb{F}\}$$

Thus, we capture the effect of non-deterministic assignments by collecting *all constant* assignments. The runs reaching t from program nodes are the smallest solution of the following constraint system $\mathbf{R}$:

$$\begin{array}{ll} [\mathrm{R1}] & \mathbf{R}(t) \supseteq \{\varepsilon\} \\ [\mathrm{R2}] & \mathbf{R}(u) \supseteq f_e(\mathbf{R}(v))\,, \text{ if } e = (u, v) \in E \end{array}$$

where "ε" denotes the empty run, and $f_e(R) = \{r;t \mid r \in \mathbf{R}(e) \wedge t \in R\}$. By [R1], the set of runs reaching the target when starting from the target contains the empty run. By [R2], a run starting from u is obtained by considering an outgoing edge $e = (u, v)$ and concatenating a run corresponding to e with a run starting from v.

So far, we have furnished flow graphs with a symbolic operational semantics only by describing the sets of runs possibly reaching program points. Each of these runs gives rise to a *partial* transformation of the underlying program state $x \in \mathbb{F}^k$; for states outside the domain the run is not executable, because some of the conditions in the run are not satisfied. Every guard $p \neq 0$ induces a partial identity function with domain

$$\mathsf{dom}([\![p \neq 0]\!]) \ = \ \{x \in \mathbb{F}^k \mid p(x) \neq 0\}$$

Polynomial assignments are always executable. Thus, a polynomial assignment $\mathbf{x}_j := p$ gives rise to the transformation with domain $\mathsf{dom}([\![\mathbf{x}_j := p]\!]) = \mathbb{F}^k$ and

$$[\![\mathbf{x}_j := p]\!]\, x = (x_1, \ldots, x_{j-1}, p(x), x_{j+1}, \ldots, x_k)$$

These definitions are inductively extended to runs: $[\![\varepsilon]\!] = \mathsf{Id}$, where Id is the identity function and $[\![r;a]\!] = [\![a]\!] \circ [\![r]\!]$ where '$\circ$" denotes composition of partial functions.

The partial transformation $f = [\![r]\!]$ induced by a run r can always be represented by polynomials $q_0, \ldots, q_k \in \mathbb{F}[\mathbf{X}]$ such that $\mathsf{dom}(f) = \{x \in \mathbb{F}^k \mid q_0(x) \neq 0\}$ and $f(x) = (q_1(x), \ldots, q_k(x))$ for every $x \in \mathsf{dom}(f)$. This is clearly true for the identity transformation induced by the empty path ε (take the polynomials $1, \mathbf{x}_1, \ldots, \mathbf{x}_k$). It is also not hard to see that the transformations induced by polynomial assignments or guards can be represented this way. Moreover, transformations of the given form are closed under composition. To see this, consider a second transformation f' which is given by polynomial $q_0', \ldots, q_k' \in \mathbb{F}[\mathbf{X}]$. Then we have:

$$\begin{array}{ll} & x \in \mathsf{dom}(f' \circ f) \\ \text{iff} & q_0(x) \neq 0 \wedge q_0'(q_1(x), \ldots, q_k(x)) \neq 0 \\ \text{iff} & (q_0 \cdot q_0'[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k])(x) \neq 0 \end{array}$$

such that $f' \circ f$ is given by the polynomials $q_0 \cdot q_0'', q_1'', \ldots, q_k''$ where the q_i'' are obtained by substituting the polynomials q_j for $\mathbf{x}_j$ in q_i', i.e., $q_i'' = q_i'[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k]$.

3. Polynomial Relations and Weakest Preconditions

A *polynomial relation* over a vector space $\mathbb{F}^k$ is an equation $p = 0$ for some $p \in \mathbb{F}[\mathbf{X}]$. Such a relation can be represented as the polynomial p alone. The vector $y \in \mathbb{F}^k$ *satisfies* the polynomial relation p iff $p(y) = 0$.

The polynomial relation (denoted by) p holds after a single run r for those initial states $x \in \mathsf{dom}([\![r]\!])$ that satisfy $p([\![r]\!]x) = 0$. For states $x \notin \mathsf{dom}([\![r]\!])$, p is trivially guaranteed after run r as r is not executable for those states. Thus,

$$\mathbf{x} \notin \mathsf{dom}([\![r]\!]) \ \vee \ p([\![r]\!]\mathbf{x}) = 0$$

represents the *weakest precondition* of the validity of $p = 0$ after run r. Assuming that the transformation induced by the run r is represented by the polynomials $q_0, \ldots, q_k$, we have for each $x \in \mathbb{F}^k$:

$$\begin{array}{ll} & x \notin \mathsf{dom}([\![r]\!]) \ \vee \ p([\![r]\!]x) = 0 \\ \text{iff} & q_0(x) = 0 \ \vee \ p(q_1(x), \ldots, q_k(x)) = 0 \\ \text{iff} & q_0(x) = 0 \ \vee \ p[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k](x) = 0 \\ \text{iff} & (q_0 \cdot p[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k])(x) = 0 \end{array}$$

From this calculation, we deduce that the weakest precondition is again a polynomial relation. Even better: the mapping $[\![r]\!]^\mathsf{T}$ that assigns to each polynomial relation (represented by a single polynomial) its weakest precondition before run r is the *total* function defined by:

$$[\![r]\!]^\mathsf{T} p = q_0 \cdot p[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k] \tag{1}$$

The only polynomial relation which is true for *all program states* is the relation $0 = 0$. Thus, a given polynomial relation p is valid after run r iff $[\![r]\!]^\mathsf{T}\, p = 0$, because the initial state is arbitrary. Moreover, the polynomial relation p is valid at the target node t, iff it is valid after all runs $r \in \mathbf{R}(s)$. Summarizing, we have:

Lemma 1 *The polynomial relation $p_\mathrm{t} \in \mathbb{F}[\mathbf{X}]$ is valid at the target node t iff $[\![r]\!]^\mathsf{T} p_\mathrm{t} = 0$ for all $r \in \mathbf{R}(s)$.*

We conclude that the set $S = \{[\![r]\!]^\mathsf{T} p_\mathrm{t} \mid r \in \mathbf{R}(s)\} \subseteq \mathbb{F}[\mathbf{X}]$ of polynomials gives us a handle to solve the validity problem for the polynomial relation p_t at the target node t: p_t is valid at t iff $S \subseteq \{0\}$. The problem is that we need a representation of this set which is finitary—and find a way to compute it. In this place, we recall that the set $\mathbb{F}[\mathbf{X}]$ of all polynomials forms a *commutative ring*. A nonempty subset I of a commutative ring R satisfying the two conditions:

(i) $a + b \in I$ whenever $a, b \in I$ (closure under sum) and

(ii) $r \cdot a \in I$ whenever $r \in R$ and $a \in I$ (closure under product with arbitrary ring elements)

is called an *ideal*. Ideals (in particular those in polynomial rings) enjoy interesting and useful properties. For a subset $G \subseteq R$, the least ideal containing G is given by

$$\langle G \rangle = \{r_1 g_1 + \ldots + r_n g_n \mid n \geq 0, r_i \in R, g_i \in G\}$$

In this case, G is also called a set of *generators* of $\langle G\rangle$. In particular,

$$\langle G\rangle = \{0\} \quad \text{iff} \quad G \subseteq \{0\}$$

for every $G \subseteq R$. Thus, in our scenario, we can equivalently check $\langle S\rangle = \{0\}$ instead of $S \subseteq \{0\}$. We conclude that we can work with ideals of polynomials instead of sets without losing interesting information. The set $\mathcal{I}_{\mathbf{X}}$ of ideals of $\mathbb{F}[\mathbf{X}]$, ordered by subset inclusion, forms a complete lattice. In particular:

- The least element of $\mathcal{I}_{\mathbf{X}}$ is $\{0\}$.
- The greatest element of $\mathcal{I}_{\mathbf{X}}$ equals $\langle 1\rangle = \mathbb{F}[\mathbf{X}]$.
- The least upper bound $I_1 \sqcup I_2$ of two ideals is defined by $I_1 + I_2 = \langle I_1 \cup I_2\rangle$.

Moreover, we recall Hilbert's famous basis theorem for polynomial ideals over a field:

Theorem 1 (Hilbert, 1888) *Every ideal $I \subseteq \mathbb{F}[\mathbf{X}]$ of a commutative polynomial ring in finitely many variables $\mathbf{X}$ over a field $\mathbb{F}$ is finitely generated, i.e., $I = \langle G\rangle$ for a finite subset $G \subseteq \mathbb{F}[\mathbf{X}]$.* □

This means that each ideal can be effectively represented. For testing validity of a given polynomial relation p_{t} at a given target node t, we are thus left with the task to compute the ideal $\langle\{[\![r]\!]^{\mathsf{T}} p_{\mathrm{t}} \mid r \in \mathbf{R}(s)\}\rangle$ (s the entry point of the program). This ideal can be seen as an abstraction of the set $\mathbf{R}(s)$ of program executions starting in s and reaching t. We are going to compute it by an abstract interpretation of the constraint system for $\mathbf{R}(u)$ from Section 2. The desired abstraction of run sets is described by the mapping $\alpha : 2^{\mathsf{Runs}} \to \mathcal{I}_{\mathbf{X}}$:

$$\alpha(R) = \langle\{[\![r]\!]^{\mathsf{T}} p_{\mathrm{t}} \mid r \in R\}\rangle\,.$$

This definition immediately implies the following identities:

$$\begin{array}{rcl} \alpha(\emptyset) & = & \langle\emptyset\rangle \;=\; \{0\} \\ \alpha(\{r\}) & = & \langle\{[\![r]\!]^{\mathsf{T}} p_{\mathrm{t}}\}\rangle \end{array}$$

for a single run r. For the empty run ε we get:

$$\alpha(\{\varepsilon\}) \quad = \quad \langle\{p_{\mathrm{t}}\}\rangle$$

because $[\![\varepsilon]\!]^{\mathsf{T}} = \mathsf{Id}$.

The mapping α is monotonic (w.r.t. subset ordering on sets of runs and subspaces.) Also, it commutes with arbitrary unions. This is due to the following wellknown lemma:

Lemma 2 *Let $\mathcal{G}$ denote a set of subsets of polynomials. Then*

$$\langle\bigcup\{G \mid G \in \mathcal{G}\}\rangle = \bigsqcup\{\langle G\rangle \mid G \in \mathcal{G}\}$$

□

In order to solve the constraint system for the run sets $\mathbf{R}(u)$ over abstract domain $\mathcal{I}_{\mathbf{X}}$, we need an abstract transformer $f_e^\sharp : \mathcal{I}_{\mathbf{X}} \to \mathcal{I}_{\mathbf{X}}$ corresponding to edges $e = (u, v)$ which exactly abstracts f_e, i.e., the effect of concatenating the fixed run set of the edge e with run sets. We define:

$$f_e^\sharp\, I = \langle\{[\![r]\!]^{\mathsf{T}} p \mid r \in \mathbf{R}(e), p \in I\}\rangle$$

We prove:

Lemma 3 *For every subset G of polynomials,*

$$f_e^\sharp\langle G\rangle = \langle\{[\![r]\!]^{\mathsf{T}} p \mid r \in \mathbf{R}(e), p \in G\}\rangle$$

Proof: Since $G \subseteq \langle G\rangle$, we trivially have the inclusion "$\supseteq$" by monotonicity. For the reverse inclusion, consider a polynomial $p \in f_e^\sharp\langle G\rangle$. Then p can be written as $p = \sum_{i=1}^m q_i \cdot p_i$ for polynomials $p_i = [\![r_i]\!]^{\mathsf{T}} p_i'$ for some $r_i \in \mathbf{R}(e)$ and $p_i' \in \langle G\rangle$, and $q_i \in \mathbb{F}[\mathbf{X}]$. Each p_i' in turn can be written as $p_i' = \sum_{j=1}^{m_i} q_{ij} \cdot g_{ij}$ for $g_{ij} \in G$ and arbitrary polynomials q_{ij}. In particular,

$$\begin{array}{rcl} p_i & = & [\![r_i]\!]^{\mathsf{T}} p_i' \\ & = & [\![r_i]\!]^{\mathsf{T}}(\sum_{j=1}^{m_i} q_{ij} \cdot g_{ij}) \\ & = & \sum_{j=1}^{m_i} q_{ij}' \cdot [\![r_i]\!]^{\mathsf{T}} g_{ij} \end{array}$$

for some polynomials q_{ij}' (unfold the definition of $[\![r_i]\!]^{\mathsf{T}}$ for seeing the last step). Therefore, $p_i \in \langle\{[\![r]\!]^{\mathsf{T}} p \mid r \in \mathbf{R}(e), p \in G\}\rangle$ for all i. But then also $p \in \langle\{[\![r]\!]^{\mathsf{T}} p \mid r \in \mathbf{R}(e), p \in G\}\rangle$ since ideals are closed under sums and products with arbitrary polynomials. □

Using Lemma 3, we calculate:

$$\begin{array}{rcl} f_e^\sharp(\alpha(R)) & = & f_e^\sharp\langle\{[\![r]\!]^{\mathsf{T}} p_{\mathrm{t}} \mid r \in R\}\rangle \\ & = & \langle\{[\![r']\!]^{\mathsf{T}}([\![r]\!]^{\mathsf{T}} p_{\mathrm{t}}) \mid r' \in \mathbf{R}(e), r \in R\}\rangle \\ & = & \langle\{[\![r'; r]\!]^{\mathsf{T}} p_{\mathrm{t}} \mid r' \in \mathbf{R}(e), r \in R\}\rangle \\ & = & \langle\{[\![r]\!]^{\mathsf{T}} p_{\mathrm{t}} \mid r \in f_e(R)\}\rangle \\ & = & \alpha(f_e(R)) \end{array}$$

Therefore, $f_e^\sharp$ is indeed an exact abstraction of f_e. It remains to prove that the application of $f_e^\sharp$ can be effectively computed. This is easy if $A(e)$ is either a guard or a polynomial assignment. Then the set $\mathbf{R}(e)$ consists of a single element, namely, a guard or a polynomial assignment. For any generating system $G \subseteq \mathbb{F}[\mathbf{X}]$, we therefore obtain by Lemma 3,

$$f_e^\sharp\langle G\rangle = \begin{cases} \langle\{p \cdot q \mid q \in G\}\rangle\,, & \text{if } A(e) \equiv p \neq 0 \\ \langle\{q[p/\mathbf{x}_j] \mid q \in G\}\rangle\,, & \text{if } A(e) \equiv \mathbf{x}_j := p \end{cases}$$

In particular, we conclude that for every finite set of generator polynomials G, a finite generating system for the ideal $f_e^\sharp\langle G\rangle$ is effectively computable.

Not quite as obvious is the case where the edge e is labeled with an unknown assignment $\mathbf{x}_j :=?$. Then the run set $\mathbf{R}(e) = \{\mathbf{x}_j := c \mid c \in \mathbb{F}\}$ is infinite. Still, however, the effect of concatenating this run set turns out to be computable. To see this, recall that every polynomial $p \in \mathbb{F}[\mathbf{X}]$ can be uniquely written as a sum

$$p \equiv \sum_{i=0}^{d} p_i \cdot \mathbf{x}_j^i$$

where the $\{\mathbf{x}_j\}$-coefficient polynomials p_i of $\mathbf{x}_j^i$ do not contain occurrences of $\mathbf{x}_j$. Then define $\pi_j : \mathbb{F}[\mathbf{X}] \to 2^{\mathbb{F}[\mathbf{X}]}$ as the mapping which maps p to the set $\{p_0, \ldots, p_d\}$ of its $\{\mathbf{x}_j\}$-coefficient polynomials. We prove:

Lemma 4 *Assume $A(e) \equiv \mathbf{x}_j :=?$. Then for every set G of generator polynomials,*

$$f_e^\sharp\langle G\rangle = \langle\bigcup\{\pi_j(q) \mid q \in G\}\rangle$$

The lemma and its proof are similar to Lemma 8 in [11].

Proof: By definition and Lemma 3, we have:

$$\begin{aligned} f_e^\sharp\langle G\rangle &= \langle[\![\mathbf{x}_j := c]\!]^\top q \mid c \in \mathbb{F}, q \in G\rangle \\ &= \langle q[c/\mathbf{x}_j] \mid c \in \mathbb{F}, q \in G\rangle \end{aligned}$$

Obviously, each polynomial $q[c/\mathbf{x}_j]$ is contained in the ideal generated from the $\{\mathbf{x}_j\}$-coefficient polynomials of q. Therefore, a generator set of the left-hand side $f_e^\sharp\langle G\rangle$ is included in the right-hand side $\langle\bigcup\{\pi_j(q) \mid q \in G\}\rangle$ of the equation, and hence also the generated ideal. This proves the inclusion "$\subseteq$".

For the reverse inclusion, it suffices to prove for an arbitrary polynomial $q \in G$, that the set $\pi_j(q)$ of $\{\mathbf{x}_j\}$-coefficient polynomials of q is contained in $\langle q[c/\mathbf{x}_j] \mid c \in \mathbb{F}\rangle$. Assume that $q = \sum_{i=0}^{d} q_i \cdot \mathbf{x}_j^i$ where the polynomials q_i do not contain occurrences of $\mathbf{x}_j$. Consider the square matrix A defined by:

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^d \\ 1 & x_1 & x_1^2 & \ldots & x_1^d \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & x_d & x_d^2 & \ldots & x_d^d \end{pmatrix}$$

where $x_0, \ldots, x_d \in \mathbb{F}$ are $d+1$ distinct elements. It is not hard to see that

$$A \begin{pmatrix} q_0 \\ \vdots \\ q_d \end{pmatrix} = \begin{pmatrix} q[x_0/\mathbf{x}_j] \\ \vdots \\ q[x_d/\mathbf{x}_j] \end{pmatrix}.$$

The determinant of A is an instance of what is known as Vandermonde's determinant and has the value $\prod_{0 \leq i < m \leq d}(x_m - x_i)$. As the values $x_0, \ldots, x_d$ are distinct, the determinant is different from 0. Therefore, matrix A is invertible and for the inverse matrix, we have

$$\begin{pmatrix} q_0 \\ \vdots \\ q_d \end{pmatrix} = A^{-1} \begin{pmatrix} q[x_0/\mathbf{x}_j] \\ \vdots \\ q[x_d/\mathbf{x}_j] \end{pmatrix}.$$

Thus, the coefficient polynomials of q are even linear combinations of the polynomials $q[x_0/\mathbf{x}_j], \ldots, q[x_d/\mathbf{x}_j]$ which shows that $\pi_j(q) = \{q_0, \ldots, q_d\}$ is contained in the ideal generated by the polynomials $q[c/\mathbf{x}_j], c \in \mathbb{F}$. □

Since for every polynomial p, the set of its $\{\mathbf{x}_j\}$-coefficient polynomials is effectively computable, we conclude that also the generator set $\bigcup\{\pi_j(q) \mid q \in G\}$ of the ideal $f_e^\sharp\langle G\rangle$ is effectively computable—given only that the set G is finite.

For a given target node $t \in N$ and polynomial relation $p_t \in \mathbb{F}[\mathbf{X}]$ let $\mathbf{R}_{p_t}^\sharp$ denote the following abstracted constraint system over the complete lattice $\mathcal{I}_\mathbf{X}$:

$$\begin{array}{lll} [\mathrm{R1}]^\sharp & \mathbf{R}^\sharp(t) \supseteq \langle\{p_t\}\rangle & \\ [\mathrm{R2}]^\sharp & \mathbf{R}^\sharp(u) \supseteq f_e^\sharp(\mathbf{R}^\sharp(v))\,, & \text{if } e = (u, v) \in E \end{array}$$

We find:

Lemma 5 *The constraint system $\mathbf{R}_{p_t}^\sharp$ has a unique least solution $\mathbf{R}^\sharp(u), u \in N$, with the following properties:*

1. *$\mathbf{R}^\sharp(u), u \in N$, is effectively computable.*
2. *$\mathbf{R}^\sharp(u) = \alpha(\mathbf{R}(u))$ for every $u \in N$.*

Proof: Since $\mathcal{I}_\mathbf{X}$ is a complete lattice and all transfer functions $f_e^\sharp$ on right-hand sides of constraints are monotonic, the constraint system $\mathbf{R}_{p_t}^\sharp$ has a unique least solution. Moreover, recall that Hilbert's basis Theorem 1 implies that every ascending sequence of ideals:

$$I_0 \subseteq \ldots \subseteq I_m \subseteq \ldots$$

is ultimately stable, i.e., $I_{m'} = I_m$ for some $m \in \mathbb{N}$ and all $m' \geq m$. We conclude that the least solution can be computed by a finite number of fixpoint iterations. Since each intermediately occurring ideal is finitely generated, each individual fixpoint iteration is computable. By Buchberger's algorithm (cf., e.g.,[2]) it is decidable whether or not a polynomial p is contained in the ideal $\langle G\rangle$ for a finite set of generators G. It follows that it is also decidable whether an ideal I_1 is included in another ideal I_2—given only finite generator sets G_i for the involved ideals I_i [2, Theorem 5.55]. Thus, we can effectively decide when fixpoint iteration for $\mathbf{R}^\sharp_{p_t}$ reaches the least fixpoint. This completes the proof of Assertion 1

For the second assertion, we apply the Transfer Lemma of general fixpoint theory (see, e.g., [1, 5]):

Lemma 6 (Transfer lemma) *Suppose $L, L^\sharp$ are complete lattices, $f : L \to L$ and $g : L^\sharp \to L^\sharp$ are monotonic functions and $\gamma : L \to L^\sharp$ is a completely distributive function. If $\gamma \circ f = g \circ \gamma$ then $\gamma(\mu f) = \mu g$, where μf and μg are the least fixpoints of f and g, respectively, that exist by the Knaster-Tarski fixpoint theorem.*

In our case, L is the $|N|$-fold Cartesian product of 2^{Runs} (one component for each variable $\mathbf{R}(u)$, $u \in N$, in constraint system $\mathbf{R}$) and $L^\sharp$ is the $|N|$-fold Cartesian product of $\mathcal{I}_{\mathbf{X}}$ (one component for each variable $\mathbf{R}^\sharp(u)$, $u \in N$, in constraint system $\mathbf{R}^\sharp_{p_t}$). The mappings f and g are induced from the right side the constraints in $\mathbf{R}$ and $\mathbf{R}^\sharp_{p_t}$ in the standard way, such that their least fixpoints correspond to the least solutions of the constraint systems $\mathbf{R}$ and $\mathbf{R}^\sharp_{p_t}$. The mapping γ maps a vector from L to the vector of the α-images of its components.

By Lemma 2 the abstraction function α is completely distributive which implies that γ is completely distributive as well. By Lemma 3, the transfer functions are exact which together with the fact that α is completely distributive implies $\gamma \circ f = g \circ \gamma$. Hence, the Transfer Lemma ensures that the least solution of the abstracted constraint system $\mathbf{R}^\sharp_{p_t}$ is the abstraction of the least solution of the concrete constraint system $\mathbf{R}$. This is Assertion 2. Thus, the proof is complete. $\square$

We can now put the pieces together and prove the main theorem of this section.

Theorem 2 *There is an effective procedure to decide whether a polynomial relation p_t is valid at a given program point t or not.*

Proof: The polynomial relation p_t is valid at t if and only if $\alpha(\mathbf{R}(\mathsf{s})) = \{0\}$, where s is the entry point of the program. By Lemma 5 we can effectively compute a generating system for $\alpha(\mathbf{R}(\mathsf{s}))$ by computing the least solution of constraint system $\mathbf{R}^\sharp_{p_t}$. As the ideal $\{0\}$ has only the two sets of generators $\emptyset$ and $\{0\}$ it is easy to check, whether the set of generators computed for $\mathbf{R}^\sharp(\mathsf{s})$ generates $\{0\}$ or not. $\square$

Example 1 *Consider the example program from Figure 1. We want to verify that the relation given by the polynomial $p_t \equiv \mathbf{x}_1 - \mathbf{x}_2 + 1$ holds at program point $t = 2$. Starting from the ideal $\langle\{p_t\}\rangle$ for program point 2, we obtain a set of generators for the ideal $\mathbf{R}^\sharp(1)$ of preconditions at program point 1 by first computing:*

$$\begin{aligned} q_1 &= [\![\mathbf{x}_1 := \mathbf{x}_1 \cdot (\mathbf{x}_3 - 1)]\!]^\mathsf{T} p_t \\ &= p_t[\mathbf{x}_1 \cdot (\mathbf{x}_3 - 1)/\mathbf{x}_1] \\ &= \mathbf{x}_1 \cdot \mathbf{x}_3 - \mathbf{x}_1 - \mathbf{x}_2 + 1 \end{aligned}$$

and then iteratively adding to the ideal $\langle q_1\rangle$ further preconditions for the loop until stabilization is reached. We have:

$$\begin{aligned} &[\![\mathbf{x}_1 := \mathbf{x}_1 \cdot \mathbf{x}_3 + 1; \mathbf{x}_2 := \mathbf{x}_2 \cdot \mathbf{x}_3]\!]^\mathsf{T} q_1 \\ &\quad = q_1[\mathbf{x}_1 \cdot \mathbf{x}_3 + 1/\mathbf{x}_1, \mathbf{x}_2 \cdot \mathbf{x}_3/\mathbf{x}_2] \\ &\quad = (\mathbf{x}_1 \cdot \mathbf{x}_3 + 1) \cdot \mathbf{x}_3 - (\mathbf{x}_1 \cdot \mathbf{x}_3 + 1) - \mathbf{x}_2 \cdot \mathbf{x}_3 + 1 \\ &\quad = \mathbf{x}_1 \cdot \mathbf{x}_3^2 - \mathbf{x}_1 \cdot \mathbf{x}_3 - \mathbf{x}_2 \cdot \mathbf{x}_3 + \mathbf{x}_3 \\ &\quad = \mathbf{x}_3 \cdot q_1 \\ &\quad \in \langle q_1\rangle \end{aligned}$$

Thus, the value of the fixpoint for program point 1 is given by $\mathbf{R}^\sharp(1) = \langle\{q_1\}\rangle$. For the entry point 0 of the program we then calculate the set of preconditions for the set $\{q_1\}$:

$$\begin{aligned} [\![\mathbf{x}_1 := 1; \mathbf{x}_2 := \mathbf{x}_3]\!]^\mathsf{T} q_1 &= 1 \cdot \mathbf{x}_3 - 1 - \mathbf{x}_3 + 1 \\ &= 0 \end{aligned}$$

Therefore, $\mathbf{R}^\sharp(0) = \langle 0\rangle = \{0\}$ — implying that the relation $\mathbf{x}_1 - \mathbf{x}_2 + 1 = 0$ indeed holds at program point 2. $\square$

The considerations of this section can easily be extended to checking finite sets of polynomials. A set $G \subseteq \mathbb{F}[\mathbf{X}]$ is valid for a state $y \in \mathbb{F}^k$ iff $p(y) = 0$ for all $p \in G$. Thus, a set represents the conjunction of its members. We can clearly check validity of a given finite set G_t at a given program point t by applying the above procedure for each relation in G_t. We can do better, however, by checking all of them at once. Clearly, we obtain from Lemma 1:

Corollary 1 *The set of polynomial relations $G_t \subseteq \mathbb{F}[\mathbf{X}]$ is valid at the target node t iff $[\![r]\!]^\mathsf{T} p = \{0\}$ for all $r \in \mathbf{R}(s)$, $p \in G_t$.*

Accordingly, we work with the abstraction mapping $\alpha' : 2^{\mathsf{Runs}} \to \mathcal{I}_{\mathbf{X}}$:

$$\alpha'(R) = \langle\{[\![r]\!]^\mathsf{T} p \mid r \in R, p \in G_t\}\rangle\,.$$

This leads to the a slightly modified constraint $[\mathrm{R1}]^\sharp$:

$$[\mathrm{R1}]^\sharp \quad \mathbf{R}^\sharp(t) \supseteq \langle G_t\rangle$$

The rest works as before. We conclude:

Theorem 3 *There is an effective procedure to decide whether a (finite) set of polynomial relations G_t is valid at a given program point t or not.*

Note that we can represent *disjunctions* of polynomial relations by products: $p = 0 \vee p' = 0$ is valid for a state y iff $p \cdot p' = 0$ is valid for y. Thus, by considering sets of polynomials and using products, we can indeed handle arbitrary *positive Boolean combinations* of polynomial relations.

4. Inferring Valid Polynomial Relations

It seems that the algorithm of testing whether a certain given polynomial relation $p_0 = 0$ is valid at some program point contains no clue on how to infer so far unknown valid polynomial relations. This, however, is not quite true. We show in this section how to determine all polynomial relations 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 p is called a D-polynomial if it contains only monomials $b \cdot \mathbf{x}_1^{i_1} \cdot \ldots \cdot \mathbf{x}_k^{i_k}$, $b \in \mathbb{F}$, with $(i_1, \ldots, i_k) \in D$, i.e., if it can be written as

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

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 p where the degree of a monomial $b \cdot \mathbf{x}_1^{i_1} \cdot \ldots \cdot \mathbf{x}_k^{i_k}$, $b \in \mathbb{F}$, 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

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

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

Lemma 7 *Let $p = \sum_{\sigma=(i_k,\ldots,i_k)\in D} a_\sigma \cdot \mathbf{x}_1^{i_1} \cdot \ldots \cdot \mathbf{x}_k^{i_k} \in \mathbb{F}[\mathbf{X}]$ denote a D-polynomial with coefficients $a : \sigma \mapsto a_\sigma$. Then for every run r,*

$$[\![r]\!]^{\mathsf{T}} p = ([\![r]\!]^{\mathsf{T}} p_D)[a/\mathbf{a}]$$

where $[\![r]\!]^{\mathsf{T}}$ on the left-hand side of the equation is computed over $\mathbb{F}[\mathbf{X}]$ whereas on the right-hand side it is computed over $\mathbb{F}[\mathbf{X} \cup \mathbf{A}_D]$.

Proof: By Equation (1), there are polynomials $q_0, \ldots, q_k \in \mathbb{F}[\mathbf{X}]$ such that $[\![r]\!]^{\mathsf{T}} p' = q_0 \cdot p'[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k]$ for every polynomial p'. Therefore,

$$\begin{array}{rcl} [\![r]\!]^{\mathsf{T}} p & = & q_0 \cdot p[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k] \\ & = & q_0 \cdot p_D[a/\mathbf{a}][q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k] \\ & = & (q_0 \cdot p_D[q_1/\mathbf{x}_1, \ldots, q_k/\mathbf{x}_k])[a/\mathbf{a}] \\ & = & ([\![r]\!]^{\mathsf{T}} p_D)[a/\mathbf{a}] \end{array}$$

which proves the asserted equality. □

Lemma 7 tells us that instead of computing the weakest precondition of each D-polynomial separately, we as well may compute the weakest precondition of the single generic D-polynomial p_D once and for all and substitute the concrete coefficients a_σ of the polynomials p into the precondition of p_D later. In particular, we conclude that the following statements are equivalent:

1. p is valid at the target program point t;
2. $[\![r]\!]^{\mathsf{T}} p = 0$ for all $r \in \mathbf{R}(s)$;
3. $([\![r]\!]^{\mathsf{T}} p_D)[a/\mathbf{a}] = 0$ for all $r \in \mathbf{R}(s)$;
4. $q[a/\mathbf{a}] = 0$ for all $q \in \{[\![r]\!]^{\mathsf{T}} p_D \mid r \in \mathbf{R}(s)\}$;
5. $q[a/\mathbf{a}] = 0$ for all $q \in \langle\{[\![r]\!]^{\mathsf{T}} p_D \mid r \in \mathbf{R}(s)\}\rangle$.
6. $q[a/\mathbf{a}] = 0$ for all $q \in G$ in a (finite) generator G of the ideal $\langle\{[\![r]\!]^{\mathsf{T}} p_D \mid r \in \mathbf{R}(s)\}\rangle$.

Now it should be clear how an algorithm may find all polynomial relations $p = 0$ with a D-polynomial p which are valid at program point t: first, we construct the abstract constraint system $\mathbf{R}^\sharp_{p_D}$, now over $\mathcal{I}_{\mathbf{X} \cup \mathbf{A}_D}$, which for each program point u computes (a finite generator set of) the ideal $\mathbf{R}^\sharp(u) = \langle\{[\![r]\!]^{\mathsf{T}} p_D \mid r \in \mathbf{R}(u)\}\rangle$. Then it remains to determine the set of all coefficient maps $a : D \to \mathbb{F}$ such that $q[a/\mathbf{a}] = 0$ for all $q \in \mathbf{R}^\sharp(s)$. Recall that each such polynomial $q[a/\mathbf{a}]$ is a polynomial in $\mathbb{F}[\mathbf{X}]$. Any such polynomial equals 0 iff each coefficient b of each occurring monomial $b \cdot \mathbf{x}_1^{i_1} \ldots \mathbf{x}_k^{i_k}$ equals 0. The polynomial $q \in \mathbb{F}[\mathbf{X} \cup \mathbf{A}_D]$, on the other hand, can uniquely be written as a finite sum

$$q \quad = \sum_{\sigma = (i_1, \ldots, i_k)} q_\sigma \cdot \mathbf{x}_1^{i_1} \cdot \ldots \cdot \mathbf{x}_k^{i_k} \qquad (2)$$

where each $\mathbf{X}$-coefficient q_σ is in $\mathbb{F}[\mathbf{A}_D]$, i.e., may only contain occurrences of variables from $\mathbf{A}_D$. Thus, $q[a/\mathbf{a}] = 0$ iff $q_\sigma[a/\mathbf{a}] = 0$ for all index tuples σ occurring in the sum. Summarizing our considerations so far, we have shown:

Lemma 8 *Let G denote any finite generator set for the ideal $\mathbf{R}^\sharp(s)$. The set of coefficient maps $a : D \to \mathbb{F}$ of the D-polynomials which are valid at program point t equals the set of solutions of the equation system having an equation:*

$$q_\sigma = 0$$

for each $\mathbf{X}$-coefficient q_σ of a polynomial $q \in G$. □

We are not yet done, since in general we are not able to determine the precise set of solutions of an arbitrary polynomial equation system algorithmically. Therefore, we need the following extra observation:

Lemma 9 *Every ideal $\mathbf{R}^\sharp(u)$, $u \in N$, of the least solution of the abstract constraint system $\mathbf{R}^\sharp_{p_D}$ has a finite generator set G consisting of polynomials q whose $\mathbf{X}$-coefficients are of degree at most 1, i.e., are of the form:*

$$\sum_{\sigma \in D} b_\sigma \cdot \mathbf{a}_\sigma$$

for $b_\sigma \in \mathbb{F}$. Moreover, such a generator set can be effectively computed.

Proof: The polynomial p_D has $\mathbf{X}$-coefficients which trivially have degree 1, since these consist of individual variables $\mathbf{a}_\sigma$. Also, applications of the least upper bound operation as well as of the abstract transformers $f_e^\sharp$ when applied to (ideals represented through) finite sets of generators with $\mathbf{X}$-coefficients of degree at most 1 again result in finite sets of generators with this property. Therefore, the assertion of the lemma follows by fixpoint induction. □

Together Lemma 8 and Lemma 9 show that 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. We conclude our second main theorem:

Theorem 4 *The set of all D-polynomials which are valid at some target program point t can be effectively computed.* □

Example 2 *Consider again the example program from Figure 1. We want to determine for program point 2, all valid polynomial relations up to degree 1, i.e., all valid polynomial relations of the form $a_0 + a_1 \cdot \mathbf{x}_1 + a_2 \cdot \mathbf{x}_2 + a_3 \cdot \mathbf{x}_3 = 0$. Let $p_1 \equiv \mathbf{a}_0 + \mathbf{a}_1 \cdot \mathbf{x}_1 + \mathbf{a}_2 \cdot \mathbf{x}_2 + \mathbf{a}_3 \cdot \mathbf{x}_3$ denote the generic D-polynomial for $D = D_1$. Starting from the ideal $\langle \{p_1\} \rangle$ for program point 2, we determine a set of generators for the ideal $\mathbf{R}^\sharp(1)$ of preconditions at program point 1. First, we compute:*

$$\begin{aligned} q_1 &= [\![\mathbf{x}_1 := \mathbf{x}_1 \cdot (\mathbf{x}_3 - 1)]\!]^\top p_1 \\ &= p_1[\mathbf{x}_1 \cdot (\mathbf{x}_3 - 1)/\mathbf{x}_1] \\ &= \mathbf{a}_0 - \mathbf{a}_1 \cdot \mathbf{x}_1 + \mathbf{a}_2 \cdot \mathbf{x}_2 + \mathbf{a}_3 \cdot \mathbf{x}_3 + \mathbf{a}_1 \cdot \mathbf{x}_1\mathbf{x}_3 \end{aligned}$$

Next, we add the preconditions for the body of the loop:

$$\begin{aligned} &[\![\mathbf{x}_1 := \mathbf{x}_1 \cdot \mathbf{x}_3 + 1; \mathbf{x}_2 := \mathbf{x}_2 \cdot \mathbf{x}_3]\!]^\top q_1 \\ &\quad = q_1[\mathbf{x}_1 \cdot \mathbf{x}_3 + 1/\mathbf{x}_1, \mathbf{x}_2 \cdot \mathbf{x}_3/\mathbf{x}_2] \\ &\quad = \mathbf{a}_0 - \mathbf{a}_1 + \mathbf{a}_1 \cdot \mathbf{x}_3 + \mathbf{a}_3 \cdot \mathbf{x}_3 \\ &\qquad -\mathbf{a}_1 \cdot \mathbf{x}_1\mathbf{x}_3 + \mathbf{a}_2 \cdot \mathbf{x}_2\mathbf{x}_3 + \mathbf{a}_1 \cdot \mathbf{x}_1\mathbf{x}_3^2 \\ &\quad = \mathbf{x}_3 \cdot q_1 + \\ &\qquad (\mathbf{a}_0 - \mathbf{a}_1 + (-\mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_3) \cdot \mathbf{x}_3 - \mathbf{a}_3 \cdot \mathbf{x}_3^2) \end{aligned}$$

The polynomial $q_2 \equiv \mathbf{a}_0 - \mathbf{a}_1 + (-\mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}3) \cdot \mathbf{x}_3 - \mathbf{a}_3 \cdot \mathbf{x}_3^2$ is independent of $\mathbf{x}_1$ and $\mathbf{x}_2$. Thus, the ideal $\langle \{q_1, q_2\} \rangle$ remains stable under further iteration and therefore equals $\mathbf{R}^\sharp(1)$. A generator set for $\mathbf{R}^\sharp(0)$ is obtained by computing:

$$\begin{aligned} [\![\mathbf{x}_1 := 1; \mathbf{x}_2 := \mathbf{x}_3]\!]^\top q_1 &= \mathbf{a}_0 - \mathbf{a}_1 + (\mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3) \cdot \mathbf{x}_3 \\ [\![\mathbf{x}_1 := 1; \mathbf{x}_2 := \mathbf{x}_3]\!]^\top q_2 &= q_2 \\ &= \mathbf{a}_0 - \mathbf{a}_1 + \\ &\quad (-\mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_3) \cdot \mathbf{x}_3 - \mathbf{a}_3 \cdot \mathbf{x}_3^2 \end{aligned}$$

The $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$-coefficients of these two polynomials now must equal 0. This gives us the following linear equations:

$$\begin{aligned} \mathbf{a}_0 - \mathbf{a}_1 &= 0 & \qquad \mathbf{a}_0 - \mathbf{a}_1 &= 0 \\ \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3 &= 0 & -\mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_3 &= 0 \\ & & -\mathbf{a}_3 &= 0 \end{aligned}$$

Thus, $\mathbf{a}_3 = 0$, $\mathbf{a}_1 = \mathbf{a}_0$, and $\mathbf{a}_2 = -\mathbf{a}_0$. We conclude that $1 + \mathbf{x}_1 - \mathbf{x}_2 = 0$ is (up to constant multiples) the only polynomial relation of degree at most 1 which is valid at program point 2. □

5. Conclusion

We have presented two analysis algorithms. The first analysis determines for a given program point of a polynomial program with polynomial disequality guards whether a given polynomial relation is valid or not. The second analysis generalizes this algorithm to infer all polynomial relations of an arbitrary given form, e.g., all polynomial relations up to a given degree d.

We do not know any upper complexity bound for our algorithms. The termination proof relies on Hilbert's basis theorem whose 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 the algorithms. A first lower bound for the problems in question is provided in [10] where the problem of must-constant propagation for $\{+, -, *\}$-constants for polynomial constants is proven to be already PSPACE-hard.

Linear algebra techniques have been used in program analysis for a long time. In his seminal paper [8], Karr presents an analysis that determines valid *affine relations* by a forward propagation of affine spaces. His analysis is precise for *affine programs*, i.e., it interprets assignments with affine right-hand sides precisely. In [10] we observe that *checking* a given affine relation for validity at a program point can be performed by a simpler *backward propagating* algorithm. This idea of backward propagation has lead to an interprocedural generalization of Karr's result [11] and also underlies the current paper. In comparison with Karr's result, we have a more general space of properties, namely polynomial relations instead of affine relations. Secondly, our analysis is precise for a larger class of programs, namely polynomial programs instead of affine programs. Finally, we leave the realm of linear algebra and rely on results from computable algebra instead.

We are not aware of much work on using techniques from computable *algebra* in program analysis, like we do here. In the work of Michel le Borgne et. al. (cf., e.g., [9]) and Gunnarsson et. al. (cf., e.g., [6]) polynomials over a finite field are used for representing state spaces in a forward reachability analysis of polynomial dynamical systems or discrete event dynamical systems, respectively. However, they actually work in a finite factorization of a polynomial ring over a finite field and use polynomials for representing state spaces of finite systems and not for treating arithmetic properties. Thus, they use polynomials as a convenient data structure but not to gain new decidability insights. Recently, Sankaranarayanan et. al. [14] proposed a method for generating non-linear loop invariants using techniques from computable algebra. In contrast to our technique their method is approximate: there is no guarantee of completeness for a well-specified class of programs. On the other hand, they provide a non-trivial (but incomplete) treatment of positive polynomial guards. Therefore, the results obtained with these two techniques are incomparable.

It is a challenging open problem whether or not the set of *all* valid polynomial relations can be computed not just the ones of some given form. It is not hard to see that this set is an ideal of $\mathbb{F}[\mathbf{X}]$. Hence, by Hilbert's basis theorem it can be represented by a finite set of generators such that this is a well-posed problem. Another challenge is to treat the inter-procedural case, i.e., to detect or even infer polynomial relations in programs with polynomial assignments and procedures.

References

[1] K. R. Apt and G. D. Plotkin. Countable Nondeterminism and Random Assignment. *Journal of the ACM*, 33(4):724–767, 1986.

[2] T. Becker and V. Weispfennig. *Gröbner Bases. A Computational Approach to commutative Algebra*. Springer Verlag, New York, second edition, 1998.

[3] S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful techniques for the automatic generation of invariants. In *Proceedings of the Eighth International Conference on Computer Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*. Springer, 1996.

[4] N. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87–, 1997.

[5] P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation. *Electronic Notes in Theoretical Computer Science*, 6, 1997. URL: `www.elsevier.nl/locate/entcs/volume6.html`.

[6] J. Gunnarsson, J. Plantin, and R. Germundsson. Verification of a large discrete system using algebraic methods. In *Proc. of the IEE WODES'96*, 1996.

[7] M. S. Hecht. *Flow analysis of computer programs*. Elsevier North-Holland, 1977.

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

[9] H. Marchand and M. le Borgne. The supervisory control problem of discrete event systems using polynomial methods. Technical Report 3790, INRIA Rennes, October 1999.

[10] M. Müller-Olm and O. Rüthing. The Complexity of Constant Propagation. In *10th European Symposium on Programming (ESOP)*, pages 190–205. LNCS 2028, Springer-Verlag, 2001.

[11] M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. Accepted for *31st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL)*, 2004.

[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] 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.

[14] S. Sankaranarayanan, H. Sipma, and Z. Manna. Non-linear loop invariant generation using gröbner bases. In *ACM SIGPLAN Principles of Programming Languages (POPL 2004)*, 2004.

[15] M. M. üller Olm and H. Seidl. A note on karr's algorithm. In *Proceedings of ICALP 2004*, 2004. to appear.