Optimal abstraction on real-valued programs

David Monniaux
Laboratoire d'informatique de l'École normale supérieure
45, rue d'Ulm, 75230 Paris cedex 5, France

May 22, 2007

Abstract

In this paper, we show that it is possible to abstract program fragments using real variables using formulas in the theory of real closed fields. This abstraction is compositional and modular. We first propose an exact abstraction for programs without loops. Given an abstract domain (in a wide class including intervals and octagons), we then show how to obtain an optimal abstraction of program fragments with respect to that domain. This abstraction allows computing optimal fixed points inside that abstract domain, without the need for a widening operator.

1 Introduction

In program analysis, it is often necessary to derive relationships between program variables automatically; thus, within the framework of abstract interpretation [5], many *numerical abstract domains* have been developed. In addition to finding relationships between the values of program variables at a certain point, one may also want relationships between the input and the output variables of a program fragment, for instance procedures, so as to obtain a compositional and modular analysis. This paper presents algorithms for obtaining formulas expressing such relationships, at various degrees of precisions, as well as algorithms for extracting numerical results from these formulas. We consider programs operating on real variables — variables whose values lie in the real field ($\mathbb{R}$).

Relational abstract domains — those considering relations between several variables, as opposed to information on each variable separately — are often designed assuming integer ($\mathbb{Z}$), rational ($\mathbb{Q}$) or real ($\mathbb{R}$) variables inside the program to be studied. Why consider programs with real variables,

which are not implementable in practice, while real-life program variables are generally either integer, or floating-point? This is because these relational domains suppose strong algebraic structures (ordered rings and fields) that floating-point numbers do not have (addition is not even associative). Then, real variables can be used as a sound *model* or *abstraction* of floating-point variables, enabling the analysis of programs using floating-point variables.

While it is unsound to assume that floating-point computations behave identically to real number computations, one can model floating-point computations using non-deterministic real number computations: each floating-point computation computes a result close to the ideal result, and the rounding error can be bounded. For instance, assuming IEEE-754 floating-point, the behavior of floating-point addition $x \oplus y$ is over-approximated by the real computation $x \oplus y = x + y + \epsilon$ with $|\epsilon| \leq \epsilon_r|x + y| + \epsilon_a$ chosen non-deterministically, where ϵ_r and ϵ_a are nonnegative coefficients depending on the floating-point type used. [9] Thus, relational abstract domains suitable for programs over real numbers are a worthy tool for the study of programs with floating-point computations.

This paper proposes an interpretation of real-valued programs without loops as formulas in the theory of real closed fields. [3][1, Chapter 2] This interpretation captures perfectly the input-output relationships of such programs. Furthermore, if one is given preconditions of the form $f_j(v_1, \ldots, v_n) \leq c_j$ where v_k are the input values of the variables of the programs, the f_j are polynomials and c_j some coefficient, then one can obtain, automatically and with arbitrary precision, c'_j such that $f_j(v'_1, \ldots, v'_n) \leq c'_j$ where v'_k are the output values of the variables of the program, and these c'_j are optimal bounds. Such pre-conditions and post-conditions encompass a wide variety of abstract domains, including intervals, octagons, octahedra etc. [8, 4], meaning that whole programs without loops can be algorithmically optimally approximated with respect to these abstract domains.

In the case of programs with loops, we cannot hope to have such a result, which would entail solving the halting problem. However, we show how to obtain a formula defining the exact least fixed point of the optimal abstraction of the program semantics. Again, we can then obtain numerical values with arbitrary precision. No "widening operator" is used, and the loss of precision entirely depends on the choice of constraints representable by the abstract domain.

In section 2, we shall recall the definition and classical results of the theory of real closed fields, and then we shall propose algorithms for bounding, with arbitrary precision, real numbers defined by formulas in that theory. In section 3, we shall define the language that we consider and give an "ex-

act" abstract semantics of this language using formulas in the theory of real closed fields. In section 4, we shall see how to get the optimal abstractions we announced.

2 Mathematical preliminaries

Throughout the paper, we shall express relationships between real numbers as formulas within the theory of real closed fields (polynomial equalities and inequalities, logical connectors, quantifiers). This theory is powerful yet decidable. We shall also obtain some real numbers as the unique solution (or model) of a formula in that theory; we shall show that from such a characterization we can compute approximations (lower and upper bounds) with arbitrary precision.

2.1 Real closed fields and algebraic numbers

Let us recall the definition of the syntax and semantics of the theory of real closed fields. In the rest of the paper, by "formula" we shall mean a formula in that theory.

We consider the following formulaic language: atomic formulas are of the form $P(x, y, z, \dots) \bowtie C$ where C is a rational number, P is a polynomial with rational coefficients, and $\bowtie$ is a comparison operator ($<$, $\leq$, $=$, $\neq$, $\geq$, $>$); compound formulas are formed using logical connectors $\neg$, $\wedge$, $\vee$, as well as quantifiers $\exists x$ and $\forall x$ where x is a variable. If a formula contains no quantifier, it is *quantifier-free*. The set of formulas on the set of variables V is noted $\mathcal{F}(V)$, while the set of quantifier-free formulas is noted $\mathcal{F}_{\mathrm{QF}}(V)$.

Notions of free and bound variables are defined as usual. Model candidates m of a formula F are *assignments* for the free variables of F, an assignment being a map from the set $FV(F)$ of free variables of F to the reals. We say that a model candidate m is a *model* of F, and note $m \models F$, with the obvious definition. The set of models of F is noted $\mathcal{M}(F)$. We recall the following result: [1, Th. 2.77]

Theorem 1 (Tarski, 1951). *There exists an algorithm E such that for any formula F in the above language, the algorithm outputs a quantifier-free formula $E(F)$ such that $FV(E(F)) = FV(F)$ and, for any model candidate m, $m \models F$ if and only if $m \models E(F)$.*

In practice, one does not use Tarski's algorithm, which has nonelementary complexity, but one rather uses cylindrical algebraic decomposition, which has "only" 2^{2^n} complexity in the size of the formula. [3][1, Ch. 11]

In many cases, we shall have a set V of "pre" variables and a set V', a disjoint copy of V, of "post" variables; a variable x in V corresponds to a copy x' in V'; we call formulas over such variables *pre-post formulas*. These will encode input-output relationships of program fragments. To an assignment $a \in \mathbb{R}^V$ we associate an assignment $a' \in \mathbb{R}^{V'}$. Given a pre-post formula F and a set of assignments $A \subseteq \mathbb{R}^V$, we define associated "predicate transformers":

$$\overrightarrow{F}(A) = \{b \mid \exists a \in A \ (a, b') \models F\} \tag{1}$$

$$\overleftarrow{F}(A) = \{b \mid \exists a \in A \ (b, a') \models F\} \tag{2}$$

We shall compute using algebraic numbers. An algebraic number r will be specified as a formula $\tilde{r}$ in the theory of real closed fields, with a single free variable x, such that $x \mapsto r$ is the model of $\tilde{r}$. We can restrict the formulas to conjunctions of polynomial equalities and inequalities without loss of generality: if F is a quantifier-free formula with a single assignment $x \mapsto r$ as model, it can algorithmically be put in disjunctive normal form $C_1 \vee \cdots \vee C_n$ where $C_1, \ldots, C_n$ are conjunctions; then inconsistent conjunctions C_i can be removed algorithmically; any of the remaining conjunctions will have a single model $x \mapsto r$ and fits our needs. Algebraic numbers specified in this way can be algorithmically approximated to arbitrary precision, within a framework of *computable reals*, as shown in the following section.

2.2 Computable reals

Our abstract domains will "compute" reals in an indirect way: instead of computing the value of a real number (which is impossible to do exactly in most cases), the abstract domain will define it as the unique solution of a quantifier-free formula with one variable; for instance, $\sqrt{2}$ would be defined as the unique x such that $x^2 = 2 \wedge x > 0$. In this section, we show that given such a characterization, one can algorithmically bound the real number with arbitrary precision; that is, given $\epsilon \in \mathbb{Q}$, $\epsilon > 0$, obtain $m, M \in \mathbb{Q}$ such that $m \leq x \leq M$ and $M - m \leq \epsilon$. More generally, we shall show that for any quantifier-free formula in the theory of real closed fields with one free variable, we can obtain a finite description of its domain of validity, such that all numbers used inside the description are computable with arbitrary precision.[1]

[1] This is a generalization of a result of Turing, that real algebraic numbers are computable [14, §1.vi].

We define computable reals through *approximation functions*: instead of a real r, which cannot be represented directly in a machine, we shall consider a computable function $\tilde{r}$ taking a positive rational number ϵ as a parameter and outputting a couple (m, M) of rational numbers such that $M - m \leq \epsilon$ and $m \leq r \leq M$, called an *ϵ-approximation*.[2]

We shall give our algorithms in a "literate programming" or "proof-carrying code" fashion, mixing each algorithm with a proof of its correctness. For the sake of simplicity, we preferred to give all algorithms "from scratch" instead of relying on advanced techniques.[3]

Let a and b be computable reals given by approximation functions $\tilde{a}$ and $\tilde{b}$. It is straightforward to compare these two numbers, provided that we know that they are different.

Algorithm COMPARE: Compare two computable reals known to be different

If $a \neq b$, then there is an algorithm that decides whether $a < b$ or $a > b$ given approximation functions $\tilde{a}$ and $\tilde{b}$: start with $\epsilon = 1$; compare the intervals $\tilde{a}(\epsilon)$ and $\tilde{b}(\epsilon)$; if they do not overlap, the case is settled, otherwise divide ϵ by 10 and try again. The algorithm will terminate at the latest when $\epsilon < |b - a|/2$.

This algorithm loops forever if $a = b$. Throughout the rest of this section, we shall take precautions so that we never use COMPARE on operands that could be equal.[4] We then define elementary arithmetic operators over approximation functions:

Algorithm PLUS: Add two computable reals

$a + b$ is also a computable real: for $\epsilon > 0$, compute $\epsilon/2$-approximations $[m_a, M_a]$ of a, $[m_b, M_b]$ of b, and output $[m_a+m_b, M_a+M_b]$ as a ϵ-approximation of $a + b$.

A similar algorithm works for $a - b$, defining MINUS.

Algorithm MULT: Multiply two computable reals

---

[2] Turing's original characterization of the class of computable reals [14] [15, Def. 4.1.12] used machines that enumerated the decimals of the number. The class of computable reals defined in this fashion is identical to ours, but there are drawbacks to this representation: it may be necessary in order to compute the n-th digit of a result to go arbitrarily far in the representation of the operands. We thus rather use a representation very close to that of Weihrauch. [15, §1.3.2]

[3] Alternatively, the same result may be reached using published algorithms [1, Alg. 10.4 to 10.17] for isolating roots of polynomials, pairs of polynomials or finding the sign of a polynomial at the roots of another, together with a dichotomy solving method.

[4] This is actually an essential restriction of any representation of computable reals. [15, Th. 4.1.16]

- compute a 1/2-approximation $[m_a, M_a]$ of a; if $0 \in [m_a, M_a]$, then write $a \cdot b = (a+1) \cdot b - b$ and the problem is reduced to the case where a cannot be zero;

- decide whether $a < 0$ or $a > 0$ by testing whether $m_a < 0$: if $a < 0$, write $a \cdot b = -((-a) \cdot b)$ and the problem is reduced to the case where $a > 0$; we also have computed a upper bound $L_a \in \mathbb{Q}$ of a;

- do similarly with b and the problem is reduced to the case where $b > 0$; we also have computed a upper bound $L_b \in \mathbb{Q}$ of b;

- compute a $\epsilon/(2L_b)$-approximation $[m_a, M'_a]$ of a and a $\epsilon/(2L_a)$-approximation $[m_b, M'_b]$ of b; let $M_A = \min(M'_A, L_a)$ and $M_b = \min(M'_b, L_b)$ and output $[m_a m_b, M_a M_b]$; this is a ϵ-approximation of $a.b$ since $M_a M_b - m_a m_b = M_a(M_b - m_b) + m_b(M_a - m_a) \leq \epsilon$.

Now for three algorithms that will be later used as subroutines:

Algorithm DECIDESIGN: Decide the sign of $P(x)$ if $P(x) \neq 0$

It follows that if $P \in \mathbb{Q}[X]$, and r is a real given by $\tilde{r}$ such that $P(r) \neq 0$, then we can decide whether $P(r) < 0$ or $P(r) > 0$ using PLUS and MULT over the polynomial structure, then COMPARE.

Algorithm FINDROOT: Find the unique root of P in an interval $[r_1, r_2]$ of monotonicity

Let $r_1 < r_2$, given by $\tilde{r}_1$ and $\tilde{r}_2$, and P a polynomial such that P is strictly increasing over $[r_1, r_2]$, $P(r_1) < 0$ and $P(r_2) > 0$. Let $\epsilon > 0$. Compute $[m_1, M_1]$ a ϵ-approximation of r_1 and $[m_2, M_2]$ an ϵ-approximation of r_2. If $P(M_1) \geq 0$, then $[m_1, M_1]$ is an ϵ-approximation of r_0. If $P(m_2) \leq 0$, then $[m_2, M_2]$ is a ϵ-approximation of r_0. We thus suppose $P(M_1) < 0$ and $P(m_2) > 0$ and apply a dichotomy algorithm between the two, until we reach the desired precision.

Algorithm FINDROOTINF: Find the unique root of P in an interval $(-\infty, r_2]$ of monotonicity[5]

If we know that P is strictly increasing on $(-\infty, r_2]$, $P(r_2) > 0$, noting r the root of P such that $r < r_2$, then, similarly, let $\epsilon > 0$; compute $[m_2, M_2]$ a ϵ-approximation of r_2; if $P(m_2) \leq 0$ then $[m_2, M_2]$ is a ϵ-approximation of r. If $P(m_2) > 0$ then take $k \in \mathbb{N}$, $-k < m_2$, k increasing until $P(-k) < 0$; then apply the dichotomy algorithm between $-k$ and m_2.

Let us recall a familiar result, which we shall use with $K = \mathbb{Q}$ and $K' = \mathbb{R}$:

[5]We note open intervals (a, b), closed intervals $[a, b]$.

Lemma 2. *Let K be a field and K' an extension of K. If $\xi \in K'$ is a common root of nonzero polynomials P and Q from $K[X]$, then it is a root of their greatest common divisor* $\gcd(P, Q)$ *in $K[X]$. Thus, co-prime polynomials have no common root.*

Proof. $K[X]$ is a principal ring [7, Ch. 4, Th. 1.2], there exist polynomials A and B in $K[X]$ such that $\gcd(P, Q) = A.P + B.Q$. The result follows by applying both members of the equation to ξ. □

Several of our algorithms operate on *sign diagrams*. A sign diagram for a nonzero polynomial $P \in \mathbb{Q}[X]$ is a sequence $-, \tilde{r}_1, +, \tilde{r}_2, -, \tilde{r}_3, +, \ldots, \tilde{r}_n, +$, where the $\tilde{r}_i$ are approximation functions for the roots of P. Such a diagram means that the polynomial function $P(x)$ is negative for large negative x, then passes a root r_1 that can be approximated to arbitrary precision by $\tilde{r}_1$, then becomes positive, etc.

Sign diagrams for polynomials of degrees 0 and 1 are straightforward to compute, as are the first and final signs of the diagram for any polynomial, which are obtained from the parity of the degree of the polynomial and the sign of the leading coefficient. Given the diagram of P and a nonnegative exponent e, it is straightforward to compute the diagram for P^e; and given the diagram for P and a coefficient $a \in \mathbb{Q}$, it is also straightforward to compute the diagram for aP.

Given two polynomials P and Q with no common roots, one obtains the sign diagram for $P.Q$ through a simple sorted list merging procedure using Compare. This algorithm, however, does not apply in case P and Q have common roots. We use the fact (Lem. 2) that the common roots of P and Q are the roots of the greatest common divisor $\gcd(x, y)$ of these polynomials to work around this difficulty. $\gcd(x, y)$ can be computed using Euclid's algorithm.

Algorithm SignDiagram: Compute the sign diagram of a polynomial

We shall now show how to compute the sign diagram of a polynomial P by induction on the degree n of P. We have already noted that it is trivial to compute diagrams for polynomials of degrees 0 and 1. We now shall suppose that we can compute the sign diagrams of polynomials of degree less than n, and show that we can compute the sign diagram of a polynomial of degree n. First, define a subroutine:

Algorithm SignDiagramProduct:

Take as input a list $(P_1, e_1), \ldots, (P_m, e_m)$ of couples each formed of a polynomial of degree less than n and a positive exponent, output the sign diagram of the product $P_1^{e_1} \times \cdots \times P_m^{e_m}$. We proceed by induction on the

sum of the degrees of $P_1, \ldots, P_m$. If this sum is 0 or 1, then the case is trivial.

- Check whether there exist P_i and P_j ($i \neq j$) not co-prime; if so, compute $Q_i = P_i / \gcd(P_i, P_j)$ and $Q_j = P_j / \gcd(P_i, P_j)$, then replace (P_i, e_i) and (P_j, e_j) by (Q_i, e_i), (Q_j, e_j), $(\gcd(P_i, P_j), e_i + e_j)$ in the list. The sum of the degrees has decreased by the degree of $\gcd(P_i, P_j)$, but the product $P_1^{e_1} \times \cdots \times P_m^{e_m}$ has stayed the same, and thus we can solve the problem through a recursive call.
- Otherwise, the P_i are pairwise co-prime. Since they all have degree less than n, we can obtain their sign diagrams. We then apply the exponent algorithm, then the algorithm for the sign diagrams of a product of polynomials with no common roots.

Consider now a polynomial P of degree n.

- If P and its derivative P' are not co-prime, then let $Q = P / \gcd(P, P')$. Q and $\gcd(P, P')$ will have degree at most $n-1$, so we can invoke SignDiagramProduct and obtain the sign diagram of their product P.
- If they are co-prime: P only has single roots. Compute the sign diagram of P', which gives us intervals of monotonicity for P. Then, compute the sign diagram of P as follows:
  - The leftmost sign is deduced from the leading coefficient and parity of the degree of P. Without loss of generality, we shall suppose it is positive.
  - Compute the sign of $P(r_1)$ (using DecideSign) where r_1 is the first root in the sign diagram of P'; this is possible because r_1 is not a root of P. If it is negative, search for a root of P to the left of r_1 using FindRootInf.
  - For each subsequent root r_k of P', compute the sign of $P(r_k)$ (using DecideSign), and if it is different from the sign of $P(r_{k-1})$, search for a root of P in $[r_{k-1}, r_k]$ using FindRoot.

For a system S of polynomial equalities or inequalities over a real variable x, we call *validity diagram* a sequence $b_0, r_1(B_1), b_1, r_2(B_2), \ldots, r_m(B_m)$ where $r_1, \ldots, r_m$ are given by approximation functions $\tilde{r}_1, \ldots, \tilde{r}_m$, and the b_i and B_i are booleans; b_0 says whether S is always or never satisfied over $(-\infty, r_1)$, B_1 whether S is satisfied at r_1, b_1 whether S is always or never satisfied over (r_1, r_2) and so on.

Algorithm DOMAIN: Domain of validity of a quantifier-free formula with one free variable

Consider now a quantifier-free formula F with one free variable, made up of of polynomial equalities and inequalities $P_i \bowtie 0$. Similarly as in SIGNDIAGRAMPRODUCT, take greatest common divisors until obtaining a base B_k of pairwise co-prime polynomials such that for all i, P_i can be written $P_i = B_1^{e_1} \times \cdots \times B_m^{e_m}$. Compute the sign diagrams of all B_k. The validity diagram of F can be computed from the B_k using, as previously, a variant of the merging of sorted lists and the fact the B_k, pairwise, have no common roots.

By preprocessing formulas through quantifier elimination, we can algorithmically approximate to arbitrary precision any (algebraic) real defined by a formula in the theory of real closed fields.

Corollary 3. *If F is a formula of the theory of real closed fields with one free variable, such that F defines a single real, then this real is algebraic and can be algorithmically approximated to arbitrary precision.*

3 Concrete and exact abstract semantics

We consider a simple block-structured programming language without loops, and a concrete semantics as the binary relation between input variables and output variables. This concrete semantics can be exactly represented using formulas in the theory of real closed fields.

3.1 Concrete semantics

We consider the following language L:

- Real expressions are constructed over: real variables (taken in a set V of variable names), arithmetic operators ($+$, $-$, $/$, $\times$), integer constants.
- Boolean expressions are constructed from atomic formulas using $\vee$, $\wedge$, $\neg$
- Atomic formulas are constructed from real expressions and relational operators ($<$, $>$, $=$, $\leq$, $\leq$, $\geq$)
- The only control construct is if-then-else, where the condition is a boolean expression.

- The only instruction is the assignment $x := e$, where x is a real variable and e a real expression.
- There is a nondeterministic choice instruction $x := [m, M]$, choosing x between m and M.

A program in L has a concrete semantics as a binary relation over $\mathbb{R}^V$: $(pre, post) \in [\![P]\!]$ if it is possible to reach the state *post* at the end of the execution of P starting from the state *pre*.

For the sake of simplicity, we shall consider the sub-language L_s where expressions in assignments are to be simple, with only a single operator, and arithmetic expressions in boolean expressions are to be variables; programs in L can be turned into equivalent programs in L_s using additional variables to store intermediate results.

It is immediate that all results in the rest of the paper persist if one replace the theory of real close fields with another arithmetic theory admitting quantifier elimination (Presburger arithmetic, or the theory of rational or real inequalities).

3.2 Quantifier-free closed real field formulas

We compile programs without loops, compositionally, into quantifier-free formulas from the theory of real closed fields such that for a program P, the formula $[\![P]\!]_F$ models the input-output relationship $[\![P]\!]$ exactly.

We consider disjoint copies V' and $V_\exists$ of the set V: V will be used for free formula variables denoting the input values of program variables, V' for free variables denoting the output values of program variables, and $V_\exists$ for variables bound by existential quantifiers, which are to be removed from the formulas by quantifier elimination. $[\![]\!]_F$ is defined as follows:

Arithmetics Addition $[\![a := b + c]\!]_F \triangleq a' = b + c$

Subtraction $[\![a := b - c]\!]_F \triangleq a' = b - c$

Multiplication $[\![a := b * c]\!]_F \triangleq a' = b \times c$

Division $[\![a := b/c]\!]_F \triangleq b = a' \times c$

Tests $[\![\texttt{if } c \texttt{ then } p_1 \texttt{ else } p_2]\!] \triangleq (c \wedge [\![p_1]\!]_F) \vee (\neg c \wedge [\![p_2]\!]_F)$

Composition $[\![P_1; P_2]\!]_F \triangleq E(\exists v_1 \ldots \exists v_n\ f_1 \wedge f_2)$ where f_1 is $[\![P_1]\!]_F$ where all variables in V' have been replaced by their copy in $V_\exists$, f_2 is $[\![P_2]\!]_F$ where all variables in V have been replaced by their copy in $V_\exists$, and

$v_1, \ldots, v_n$ are the free variables of f_1 and f_2 that are in $V_\exists$. This is the only place where we need quantifier elimination.

Nondeterministic choice $[\![x := [m, M]]\!]_F \triangleq m \leq x' \leq M$

Proposition 4. *Let P be a program in L_s. Let $pre, post \in \mathbb{R}^V$. Then $(pre, post) \in [\![P]\!]$ if and only if $(pre, post') \models [\![P]\!]_F$.*

Alternatively, we could allow formulas including quantifiers and defer quantifier elimination until it is actually needed.

What happens with programs with loops? On programs operating over integers, one can obtain logical relations linking inputs and outputs: sequences of values of program variables across iterations can be encoded into couples of integers using Gödel's β function [16, Chapter 7], and one can thus construct a finite formula defining the strongest loop invariant. [6] Then, of course, there is no way to decide the formulas obtained.

In the case of programs with reals, the situation is different. There exist some programs such that there is no formula in the theory of real closed fields that precisely links the inputs and the outputs. Consider, for instance:

```
s=0; x=1; k=1;
while (k < n) { x=x/k; s=s+x; k=k+1; }
```

As $n \to \infty$, the output s of this program increases and tends to $\exp(1)$.

Let us suppose that there exists a relationship $P(n, s')$ in the theory of real closed fields between the initial value of n and the final value of s. The formula

$$(\forall n \exists s'\ P(n, s') \Rightarrow s' \leq l) \wedge (\forall b\ (\forall n \exists s'\ P(n, s') \Rightarrow s' \leq b) \implies l \leq b). \quad (3)$$

defines a single real, the least upper bound of the possible outputs, which is $\exp(1)$. By Cor. 3, this real is algebraic; but it is well-known that $\exp(1)$ is transcendental, a contradiction. Thus, in general, strongest loop invariants cannot be expressed within the theory of real closed fields. We shall thus aim at expressing some kind of loop invariant, not necessarily the strongest.

4 Optimal abstraction over polynomial constraint domains

We now consider the abstraction of program states (in $\mathbb{R}^V$) using domains defined by polynomial constraints. This family of domains includes many

[6] A similar construction proves Cook's theorem: Floyd-Hoare axiomatic semantics on programs using integers is complete with respect to Peano's arithmetic [16, Th. 7.5].

"classical" numerical abstract domains. We shall show that, with respect to such domains, optimal abstractions are computable (in the sense of: one can compute approximations to arbitrary precision) for programs without loops. Furthermore, we shall show that least fixed points in such domains are also computable; we thus obtain a semantics for loops, optimal in a certain sense. Finally, we shall see how to define a general computable abstract semantics for programs with loops.

Throughout this section, we shall be concerned with the forward propagation problem: given an abstract precondition $s^\sharp$ and a program fragment P, characterize an abstract postcondition $e^\sharp$ such that $\overrightarrow{[\![P]\!]} \circ \gamma_f(s^\sharp) \subseteq \gamma_f(e^\sharp)$ ("if this precondition holds, then this postcondition must also hold"), and in particular an optimal $e^\sharp$ in a certain sense. However, all results work if one seeks to compute preconditions, using $\overleftarrow{[\![P]\!]}$ ("if this postcondition holds, then the input of the program must have fit this precondition").

4.1 Polynomial constraint domains: definition

A polynomial constraint domain $D_f^\sharp$ is defined by a family $(f_\lambda)_{\lambda\in\Lambda}$ of polynomials,[7] with variables in V. An element of $D_f^\sharp$ is either $\bot$, either a vector in $(-\infty;+\infty]^\Lambda$ of *parameters*. (We assume $\Lambda \neq \emptyset$.)[8]

The $\gamma_f : D_f^\sharp \to \mathcal{P}\left(\mathbb{R}^V\right)$ maps each element of $D_f^\sharp$ to the set of program states that it represents, that is, its *concretization*: $\gamma_f(\bot) = \emptyset$, and $\gamma_f((x_\lambda)_{\lambda\in\Lambda})$ is the set of variable assignments $a \in \mathbb{R}^V$ such that for all λ, $f_\lambda(a) \leq x_\lambda$. We exclude from $D_f^\sharp$ vectors x such that $\gamma(x)$ would be empty, that is, vectors specifying inconsistent constraints; we do so in order to have $\bot$ as the sole representation of the empty set.

Several "classical" numerical domains can be interpreted as polynomial constraint domains:

Intervals Polynomials are v and $-v$, for all $v \in V$.

Difference matrices Polynomials are $v_1 - v_2$, for all $v_1, v_2 \in V$.

[7] Polynomials are used to define constraints of the form $f_\lambda(v_1, \ldots, v_n) \leq d_\lambda$. More generally, our framework applies to any predicate $P_\lambda(v_1, \ldots, v_n; d_\lambda)$ of the theory of real closed fields, such that the set of models $\mathcal{M}(d_\lambda)$ for the $v_1, \ldots, v_n$ is left-continuous with respect to the parameter d_λ: $\mathcal{M}(\inf D_\lambda) = \bigcap_{d_\lambda \in D_\lambda} \mathcal{M}(d_\lambda)$. All the results given in the following sections also apply to that extended framework.

[8] We can also consider an additional family of predicates without parameters, abstracted by their truth value.

Octagons Polynomials are $\pm v$, for all $v \in V$, and $\pm v_1 \pm v_2$, for all $v_1, v_2 \in V$. [8]

Octahedra Polynomials are $\pm v_1 \pm v_2 \pm \cdots \pm v_n$, for all $v_1, v_2, \ldots, v_n \in V$. [4]

Template linear constraints Restriction to linear polynomials. [12]

Such a domain can be fitted with a straightforward complete lattice structure $(\sqsubseteq, \sqcup, \sqcap)$, making γ_f increasing with respect to $\sqsubseteq$ and $\subseteq$:

- $\bot$ is the unique least element;
- $x \sqsubseteq y$, $x, y \in (-\infty, +\infty]^\Lambda$, if for all $\lambda \in \Lambda$, $x_\lambda \leq y_\lambda$;
- the least upper bound and greatest lower bounds of a family of vectors are defined coordinate-wise.

We can also provide an optimal abstraction function[9] α_f such that (α_f, γ_f) form a Galois connection [5, §4.2.2]: $\alpha_f(\emptyset) = \bot$; $\alpha_f(S)$ (where $S \neq \emptyset$) is the vector $(x_\lambda)_{\lambda \in \Lambda}$ where $x_\lambda = \sup_{s \in S} f_\lambda(s)$. From its definition, it is obvious that α_f preserves least upper bounds. [5, Prop. 6]

α_f distinguishes among several possible abstractions of some set X the abstraction $\alpha_f(X)$ such that $\alpha_f(X)$ is minimal. Not only does this avoid taking a non-optimal abstraction — if we chose $y^\sharp$ when there exists $x^\sharp$ such that $X \subseteq \gamma_f(x^\sharp) \subsetneq \gamma_f(y^\sharp)$, then $y^\sharp$ is a non-optimal choice as an abstraction — but it also provides for a "canonical" representation. Indeed, the concretization function γ_f is injective in the case of the intervals, but needs not be so in general. In the case of the difference matrices and the octagons, there may be an infinity of abstract elements with the same concretization: if we have the constraints $v_1 - v_2 \leq C_{1,2}$, $v_2 - v_3 \leq C_{2,3}$, and $v_1 - v_3 \leq C_{1,3}$, then we get the same concretization as long as $C_{1,3} \leq C_{1,2} + C_{2,3}$. These domains, however, are fitted with a *reduction* or *closure* operation[10] such that they provide results that are minimal with respect to $\sqsubseteq$. In this example, the closure operation realizes that the constraint $v_1 - v_3 \leq C_{1,2} + C_{2,3}$ can be derived from the first two and can be used to refine the third one

[9] Neither α_f nor γ_f are computable functions: they operate on unbounded sets of rational or real numbers. The purpose of this paper is to show how to compute certain quantities defined mathematically using α_f or γ_f.

[10] The lower closure operation $\alpha_f \circ \gamma_f$ is defined for every Galois connection [5, §4.2.2], but it needs not be computable in general. In the case of difference matrices and octagons with rational coefficients, it is effectively computable by a shortest path algorithm, and certain operations require their operands to be closed. [8, §V.B]

if $C_{1,2} + C_{2,3} < C_{1,3}$. The closure operation can also detect that some constraints are inconsistent and the result should be $\bot$.

Let $\psi : \mathcal{P}\left(\mathbb{R}^V\right) \to \mathcal{P}\left(\mathbb{R}^V\right)$. A function $\psi^\sharp : D_f^\sharp \to D_f^\sharp$ is said to be an *abstraction* of ψ if $\forall d^\sharp \in D_f^\sharp,\ \psi \circ \gamma_f(d^\sharp) \subseteq \gamma_f \circ \psi^\sharp(d^\sharp)$. The *optimal abstraction* of ψ is $\alpha_f \circ \psi \circ \gamma_f$. In program analysis in general, it is possible that this optimal abstraction is not computable. However, in the next sub-section, we shall show that this optimal abstraction is computable on programs without loops with the kind of domains that we consider here.

4.2 Optimal abstraction without fixed points

In section 3.2, we have shown that the input-output relationship of concrete program variables can be represented as a formula in the theory of real closed fields. Such a formula links the output value of the program variables to their input values. Here, we shall see that the optimal abstraction of a program fragment with respect to a polynomial constraint abstract domain can also be represented as a formula in that theory; that is, we shall give a formula linking the input and output parameters for these constraints.

Consider now a set of program states abstracted by $d^\sharp \in D_f^\sharp$, and a pre-post formula ϕ over $\mathbb{R}^V \times \mathbb{R}^{V'}$. We are interested in finding the optimal abstraction of $\overrightarrow{\phi}(d^\sharp)$. We will show that, in the case where $d^\sharp \neq \bot$, the coefficients of the vector defining this optimal abstraction $d^{\sharp\prime}$ are related to the coefficients of $d^\sharp$ through a formula of the theory of real closed fields, and that the cases where $\bot$ appear are settled by deciding a formula of that theory.

An element of $d^\sharp$ is either $\bot$, or a vector, indexed by $\lambda \in \Lambda$, of $d_\lambda \in \mathbb{R} \cup \{+\infty\}$. We use a representation using only real variables: $(d_b, (d_\lambda)_{\lambda\in\Lambda}, (\bar{d_\lambda})_{\lambda\in\Lambda})$, where:

- $d^\sharp = \bot$ is encoded by $d_b = 1$ and any other value elsewhere;
- $d^\sharp = (d_\lambda^\sharp)_{\lambda\in\Lambda}$ is encoded as follows: $d_b = 0$ and for all $\lambda \in \Lambda$, either $d_\lambda^\sharp < \infty$ and $d_\lambda = d_\lambda^\sharp$, $\bar{d}_\lambda = 0$, or $d_\lambda^\sharp = \infty$ and $\bar{d}_\lambda = 1$.

If $d^\sharp = \bot$, then this optimal abstraction is obviously $\bot$. For the sake of ease of notation, let $\Lambda = \{1, \ldots, m\}$ and $V = \{v_1, \ldots, v_n\}$. Let $\mathit{abstracts}(d, x)$ be the formula $\bar{d} = 1 \vee (\bar{d} = 0 \wedge x \leq d)$. Let $\mathit{isNonEmpty}(d^\sharp)$ be the formula $\exists v_1 \ldots \exists v_n\ \mathit{abstracts}(d_1, f_1(v_1, \ldots, v_n)) \wedge \cdots \wedge \mathit{abstracts}(d_m, f_m(v_1, \ldots, v_n))$; if $\mathit{isNonEmpty}(d^\sharp)$ is false then $\overrightarrow{\phi}(d^\sharp) = \emptyset$ and, again, $\bot$ is an optimal abstraction. $\mathit{abstracts}(d, x)$ may be decided algorithmically by quantifier elimination

if the $d_1, \ldots, d_m$ are rational numbers or algebraic numbers specified as in Sec. 2.2.

Let

$$step(\phi, d^\sharp) \triangleq d_b = 0 \wedge \exists v_1 \ldots \exists v_n \; abstracts(d_1, f_1(v_1, \ldots, v_n)) \wedge \\ \cdots \wedge abstracts(d_m, f_m(v_1, \ldots, v_n)) \wedge \phi. \quad (4)$$

The models of $step(\phi, d^\sharp)$ are exactly the assignments for $v'_1, \ldots, v'_n$ such that $(v'_1, \ldots, v'_n) \in \overrightarrow{\phi} \circ \gamma_f(d^\sharp)$. Now, we need to define the image of that set by α_f using formulas.

The projections of those assignments over the f_λ constraints are defined by:

$$step_\lambda(\phi, d^\sharp, x) \triangleq \exists v'_1 \ldots \exists v'_n \; step(\phi, d^\sharp) \wedge x = f_\lambda(v'_1, \ldots, v'_n) \quad (5)$$

The following formula has models $(d, \bar{d}) \models isSup(d, x, P)$ such that $\bar{d} = 1$ if $\{x \mid P(x)\}$ has no upper bound, and otherwise $\bar{d} = 0$ and $d = \sup\{x \mid P(x)\}$:

$$isSup(d, x, P) \triangleq (\bar{d} = 1 \wedge \forall y \exists x \; y \leq x \wedge P(x)) \vee \\ (\bar{d} = 0 \wedge (\forall x \; P(x) \implies x \leq d) \wedge (\forall y (\forall x \; P(x) \implies x \leq y) \implies d \leq y) \quad (6)$$

Thus, the formula for defining the optimal parameter for the constraint indexed by λ is: $supStep_\lambda(\phi, d^\sharp, d') \triangleq isSup(d', x, step_\lambda(\phi, d^\sharp, x))$.

Finally, we define:

$$abstrStep(\phi, d^\sharp, d'^\sharp) \triangleq (d'_b = 1 \wedge \neg \exists v'_1 \ldots \exists v'_n \; step(\phi, d^\sharp)) \vee \\ (d'_b = 0 \wedge supStep_1(\phi, d^\sharp, d'_1) \wedge \cdots \wedge supStep_m(\phi, d^\sharp, d'_m)) \quad (7)$$

We thus have lifted a formula ϕ between concrete states to an optimal formula $abstrStep(\phi, d^\sharp, d'^\sharp)$ between abstract states, and the following holds:

Theorem 5. *Let ϕ be an input-output formula over variables V. Then, the constructed formula $abstrStep(\phi, d^\sharp, d'^\sharp)$ has models $(d^\sharp, d'^\sharp) \models abstrStep(\phi, d^\sharp, d'^\sharp)$, where $d^\sharp = (d_b, (d_\lambda)_{\lambda \in \Lambda}, (\bar{d}_\lambda)_{\lambda \in \Lambda})$ and $d'^\sharp = (d'_b, (d'_\lambda)_{\lambda \in \Lambda}, (\bar{d}'_\lambda)_{\lambda \in \Lambda})$, exactly such that $d'^\sharp = \alpha_f \circ \overrightarrow{\phi} \circ \gamma_f(d^\sharp)$.*

We can in particular take ϕ to be the formula defining the input-output relationships of a program in L (that is, with real variables without loops), following the constructs in §3.2. By using 2.2 we can state:

Corollary 6. *Let $(D_f^\sharp, \alpha_f, \gamma_f)$ be the polynomial constraint domain defined by a finite family of polynomials $(f_\lambda)_{\lambda\in\Lambda}$. There is an algorithm that computes, given P a program in L, and an element $d^\sharp \in D_f^\sharp$ with rational coefficients, rational bounds on the coefficients of the optimal approximation $\alpha_f \circ \overrightarrow{[\![P]\!]} \circ \gamma_f(d^\sharp)$, with arbitrary precision. The same holds with $\overleftarrow{[\![P]\!]}$ or if the coefficients of $d^\sharp$ are algebraic numbers defined by real closed field formulas.*

While we can obtain rational *bounds*, it is possible that the optimal coefficients are irrational: the optimal output interval of `if(x <0 || x*x >= 2) { x=0; }` is $[0, \sqrt{2}]$.

4.3 Optimal abstract fixpoints

We shall now show that we can also derive a relationship, expressed as a formula in the theory of real closed fields, between the parameters of an abstract state $s^\sharp$ and the least fixed point of the abstract semantics of a program fragment greater than $s^\sharp$. This gives a formula linking the parameters of the abstraction of the precondition of a loop to the parameters of an output abstract postcondition.

Let ϕ be a pre-post formula (over $V \cup V'$) and let $s^\sharp \in D_f^\sharp$. We are interested in the least fixed point (in $D_f^\sharp$) of $\alpha_f \circ \overrightarrow{\phi} \circ \gamma_f$. We shall show that it is possible to characterize such a fixed point using a formula in the theory of real closed fields. This means that for any polynomial constraint abstract domain, and any formula (for instance, a formula expressing the semantics of a program), the least fixed point in the abstract domain can be effectively computed.

In order to analyze program loops and similar constructs, we are interested in the strongest invariant containing some set z_0; an invariant of a monotonic function f is a post-fixed point, that is, z such that $f(z) \sqsubseteq z$. We recall the following result, similar to Tarski's fixed point theorem:

Lemma 7. *Let $(Z, \sqsubseteq, \sqcup, \sqcap)$ be a complete lattice and $z_0 \in Z$. Let $\psi : Z \to Z$ be an order-preserving operator. Then ψ has a least post-fixed point above z_0, noted $lpfp_{z_0}\,\psi$; and this least post fixed point is $\inf\{z \in Z \mid z_0 \sqsubseteq z \wedge f(z) \sqsubseteq z\}$. $lpfp_\bot\, f$ is the least fixed point of f.*

It is possible to define $\sqsubseteq$ using a formula such that $d^\sharp \sqsubseteq d'^\sharp \iff (d^\sharp, d'^\sharp) \models \mathit{incl}(d^\sharp, d'^\sharp)$ where:

$$\mathit{incl}(d^\sharp, d'^\sharp) \triangleq \quad d_b = 1 \vee (d'_b = 0 \wedge \mathit{lessEq}(d_1, d'_1) \wedge \cdots \wedge \mathit{lessEq}(d_m, d'_m)) \quad (8)$$

$$\mathit{less}(d, d') \triangleq \quad \bar{d}' = 1 \vee (\bar{d}' = 0 \wedge \bar{d} = 0 \wedge d \leq d') \quad (9)$$

Now define:

$$\begin{aligned} \mathit{isFix}(d^\sharp, \psi^\sharp, d_0^\sharp) &\triangleq & \mathit{incl}(d_0^\sharp, d^\sharp) \wedge \psi^\sharp(d^\sharp, d^\sharp) \\ \mathit{isLfp}(d^\sharp, \psi^\sharp, d_0^\sharp) &\triangleq & \mathit{isFix}(d^\sharp, \psi^\sharp, d_0^\sharp) \wedge \forall d''^\sharp \ \mathit{isFix}(d''^\sharp, \psi^\sharp, d_0^\sharp) \Rightarrow \mathit{incl}(d^\sharp, d''^\sharp) \end{aligned}$$

From these definitions, the following holds:

Theorem 8. *Let $\psi^\sharp$ be a formula over the variables $d^\sharp = (d_b, (d_\lambda)_{\lambda\in\Lambda}, (\bar{d_\lambda})_{\lambda\in\Lambda})$ and $d'^\sharp = (d'_b, (d'_\lambda)_{\lambda\in\Lambda}, (\bar{d'_\lambda})_{\lambda\in\Lambda})$, such that $(d^\sharp, d'^\sharp) \models \psi^\sharp$ defines an order-preserving function $\overrightarrow{\psi^\sharp} : d^\sharp \mapsto d'^\sharp$. Then, $\mathit{isLfp}(d^\sharp, \psi^\sharp, d_0^\sharp)$ has models $(d^\sharp, d_0^\sharp) \models \mathit{isLfp}(d^\sharp, \psi^\sharp, d_0^\sharp)$ such that $d^\sharp$ is the least fixed point of $\overrightarrow{\psi^\sharp}$ over $d_0^\sharp$.*

By taking $\psi^\sharp = \mathit{abstrStep}(\phi, d^\sharp, d'^\sharp)$ and applying theorem 5 :

Corollary 9. *Let ϕ be an input-output formula over variables V. Then, the constructed formula $\mathit{isLfp}(d^\sharp, \mathit{abstrStep}(d^\sharp, \psi, d'^\sharp), o^\sharp)$ has models $(d^\sharp, o^\sharp)$, where $d^\sharp = (d_b, (d_\lambda)_{\lambda\in\Lambda}, (\bar{d_\lambda})_{\lambda\in\Lambda})$ and $o^\sharp = (o_b, (o_\lambda)_{\lambda\in\Lambda}, (\bar{o}_\lambda)_{\lambda\in\Lambda})$, exactly such that $d^\sharp = \mathit{lpfp}_{o^\sharp}(\alpha_f \circ \overrightarrow{\phi} \circ \gamma_f)$.*

The application to program analysis is that *fixpoints in the abstract domain may be approximated optimally, without the use of any widening operator*:

Corollary 10. *Let $(D_f^\sharp, \alpha_f, \gamma_f)$ be the polynomial constraint domain defined by a finite family of polynomials $(f_\lambda)_{\lambda\in\Lambda}$. There is an algorithm that computes, given P a program in L, and an element $s^\sharp \in D_f^\sharp$ with rational coefficients, bounds on the coefficients of the optimal approximation $\mathit{lpfp}_{s^\sharp}(\alpha_f \circ \overrightarrow{[\![P]\!]} \circ \gamma_f)$, with arbitrary precision. The same holds with $\overleftarrow{[\![P]\!]}$ and/or if the coefficients of $s^\sharp$ are algebraic numbers defined by real closed field formulas.*

In order to compute an abstraction of the postcondition of a program `while` (*condition*) `{` *block* `}` given an abstraction $s^\sharp$ of the precondition, one can compute an abstraction of the reachable states at the head of the loop, and filter by $\neg$*condition*. The set of reachable states at the head of the loop is the least fixpoint of $X \mapsto [\![\mathit{block}]\!](X \cap [\![\mathit{condition}]\!])$ greater than $\gamma(s^\sharp)$, and an abstraction of this set is sought as a post-fixpoint of $[\![\mathit{condition}; \mathit{block}]\!]^\sharp$ greater than $s^\sharp$. Generally, this post-fixpoint is obtained using a *widening operator* [5, §4.3]: a sequence of candidates $s_1^\sharp, s_2^\sharp, \ldots$ is tried for being post-fixpoints, with a guarantee of termination; however, there is no guarantee of optimality. The above corollary gives an optimal characterization of the

least fixed point $d^\sharp$ of $[\![\mathit{condition}; \mathit{block}]\!]^\sharp$ above $s^\sharp$, through formulas linking the coefficients of $d^\sharp$ to those of $s^\sharp$. Using the algorithms in §2.2, we can obtain bounds on the coefficients of this least fixed point, with arbitrary precision.

Note, however, that the optimal result that we obtain is the least fixed point *within the abstract domain*. In general, it is not the most precise abstraction of the concrete least fixed point. Computing the most precise abstraction of least fixed point would entail being able to solve the halting problem, and since the programs over the reals include the programs over the integers, this is impossible.

The difference between the two is as follows: instead of computing the abstraction $\alpha_f(\mathrm{lpfp}\,\psi)$ of the least fixed point of an operator, we compute $\mathrm{lpfp}\,\psi^\sharp$ the least fixed point of the optimal abstraction of that operator, $\psi^\sharp = \alpha_f \circ \psi \circ \gamma_f$. If ψ is the semantics of a program fragment without loops, then ψ is additive (the image of a union of sets is the union of the images of these sets); so is α_f (the image of a union of sets is the least upper bound of the images of these sets). Then, $\mathrm{lpfp}\,\psi = \bigcup_n \psi^n(\emptyset)$ [5, §4.1] and $\alpha_f(\mathrm{lpfp}\,\psi) = \alpha_f\left(\bigsqcup_n \psi^n(\emptyset)\right) = \bigsqcup_n \alpha_f \circ \psi^n \circ \gamma_f(\bot)$. In comparison, $\mathrm{lpfp}\,\psi^\sharp$ is $\bigsqcup_n (\alpha_f \circ \psi \circ \gamma_f)^n(\bot)$. Note that the two are identical if $\gamma_f \circ \alpha_f \circ \psi \circ \gamma_f = \psi \circ \gamma_f$. $\gamma_f \circ \alpha_f$ is a *upper closure operator* that maps each set $W \subseteq \mathbb{R}^V$ to the least superset representable in the abstract domain. This means that the whole loss of precision (difference between $\alpha_f(\mathrm{lpfp}\,\psi)$ and $\mathrm{lpfp}\,\psi^\sharp$) is caused by the loss of precision introduced by this closure.

4.4 Abstracting programs with loops

In section 4.2, we have shown how to effectively compute a family of optimal relations, which we shall note $[\![P]\!]_f^{\rightarrow\sharp}$, between the coefficients of an abstract value $d^\sharp$ in a polynomial constraint domain and those of the optimal abstraction of the postcondition, $\alpha_f \circ \overrightarrow{[\![P]\!]} \circ \gamma_f$.

In section 4.3, we have shown how to effectively compute a family of optimal relations between the coefficients of an abstract value $d^\sharp$ in a polynomial constraint domain and those of the least fixpoint of $\alpha_f \circ \overrightarrow{[\![\mathit{condition}; \mathit{block}]\!]} \circ \gamma_f$ greater than $d^\sharp$.

We can thus define a modular, compositional, forward abstract semantics $[\![P]\!]_f^{\rightarrow\sharp}$ by induction on the structure of the program. Given P, this semantics yields a family of formulas in the theory of real closed fields, linking the parameters of an abstract precondition $d^\sharp$ in $D_f^\sharp$ and the parameters of an abstract postcondition $d^{\sharp\prime}$ in $D_f^\sharp$ such that each of these parameters is uniquely

defined. By using the algorithms given in section 2.2, these relationships can be used to compute the parameters in $d^{\sharp'}$ to arbitrary precision.

5 Related works and conclusion

We have defined an optimally precise abstract domain for programs without loops, based on formulas within the theory of real closed fields. This abstract domain can be used to derive optimal abstract transformers for a wide class of other domains, including familiar ones such as the interval and octagons domains. In addition, it can be also be used to provide optimal fixed points within those domains. The symbolic results that are computed can be algorithmically bounded with arbitrary precision, after applying quantifier elimination.

We have therefore demonstrated that, for a class of domains, widening operators are not needed in order to compute invariants. Moreover, our method, contrary to widenings, produces invariants that are optimal in a certain sense (the least invariant verifiable by the abstract transfer function).

There have been several published approaches to finding nonlinear relationships between program variables. One approach obtains polynomial equalities through computations on ideals using Gröbner bases [11]. This work only deals with equalities (not inequalities), uses a classical approach of computing output constraints from a set of input constraints (instead of finding relationships between the two sets of constraints), and deals with loops using a widening operator. In comparison, our approach abstracts whole program fragments, and is modular — it is possible to "plug" the result of the analysis of a procedure at the location of a procedure call.

Kapur, in some other work [6], proposes to use quantifier elimination to obtain invariants: he considers program invariants with parameters, and derives constraints over those parameters from the program.[11] Our work improves on his by noting that least invariants of the chosen shape can be obtained, not just any invariant; that the abstraction can be done modularly and compositionally (a program fragment can be analyzed, and the result of its analysis can be plugged into the analysis of a larger program), or combined into a "conventional" abstract interpretation framework (by using invariants of a shape compatible with that framework), and that the resulting invariants can be "projected" to obtain numerical quantities.

There have been other methods proposed for generating invariants from fixed parametric "shapes", using constraints over the parameters. Some ap-

[11] We thank Enea Zaffanella for pointing out this work to us.

proaches apply numerical algorithms, such as linear programming [12] or other forms of constraint solving. One difference with our work is that such methods will solve the invariant problem for *one* set of numerical values for input constraints, while we provide formulas that are valid for all sets of inputs (and then, that can be instanciated as many times as necessary to obtain numerical invariants). Some of the proposed techniques consider non-linear invariants; for instance, some [13, 10] find coefficients for algebraic *equalities* ($P(v_1, \ldots, v_n) = 0$), using techniques of Gröbner bases. Our technique finds optimal algebraic *inequalities*, and, furthermore, obtains constraints linking their parameters to constraints on program inputs.

Quantifier elimination in the theory of real closed fields is a very costly operation; thus, our algorithms, taken "as is", are likely not to be tractable beyond simple cases. However, from a theoretical point of view, it is interesting to note that widening operators are not needed in order to guarantee the computability of least fixed points in e.g. the real interval domain. We also hope that "approximate" quantifier elimination techniques (providing Q such that $\exists x\ P \implies Q$, instead of Q such that $\exists x\ P \iff Q$) may make some of our algorithms more tractable. Some experiments suggest that the algorithms can be made more efficient in the linear case, using geometric techniques, and we hope to provide more results in that respect.

With respect to applications, we envision the automatic synthesis of transfer functions for "conventional" abstract interpreters. One limitation of systems such as Astrée [2] is that, for each program construct, and each abstract domain, an abstract transfer function must be programmed by hand. If one feels like a whole block of instructions should be analyzed as a whole, in order to get more precision, then one has to derive the necessary transfer function and implement it, with risks of introducing bugs. We think that techniques such as the one in this paper, or improvements thereof, could be used to provide generic transfer functions, possibly through dynamic code generation.

References

[1] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry.* Algorithms and computation in mathematics. Springer, 2003.

[2] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded

software. In *The Essence of Computation: Complexity, Analysis, Transformation*, number 2566, pages 85–108. 2002.

[3] Bob F. Caviness and Jeremy R Johnson, editors. *Quantifier elimination and cylindrical algebraic decomposition*. Springer, 1998.

[4] Robert Clarisó and Jordi Cortadella. The octahedron abstract domain. In *SAS*, number 3148 in LNCS. Springer, 2004.

[5] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992. See `http://www.di.ens.fr/~cousot` a correctly typeset version.

[6] Deepak Kapur. Automatically generating loop invariants using quantifier elimination. In *ACA (Applications of Computer Algebra)*, 2004.

[7] Serge Lang. *Algebra*. Addison-Wesley, 3rd edition, 1993.

[8] Antoine Miné. The octagon abstract domain. In A. Simon, A. King, and J. Howe, editors, *WCRE,Analysis, Slicing, and Transformation*, pages 310–319. IEEE, 2001.

[9] Antoine Miné. Relational abstract domains for the detection of floating-point runtime errors. In *ESOP*, volume 2986 of *LNCS*. Springer, 2004.

[10] E. Rodríguez-Carbonell and D. Kapur. Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations. In *International Symposium on Symbolic and Algebraic Computation 2004 (ISSAC04)*, pages 266–273. ACM Press, 2004.

[11] Enric Rodríguez-Carbonell and Deepak Kapur. An abstract interpretation approach for automatic generation of polynomial invariants. In *SAS*, number 3148 in LNCS. Springer, 2004.

[12] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, volume 3385, pages 21–47. Springer Verlag, 2005.

[13] Henny B. Sipma Sriram Sankaranarayanan and Zohar Manna. Non-linear loop invariant generation using gröbner bases. In *POPL*. ACM, 2004.

[14] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, series 2*, 42:230–265, 1936.

[15] Klaus Weihrauch. *Computable analysis: an introduction*. Texts in Theoretical Computer Science. Springer, 2000.

[16] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. Foundations of Computing. MIT Press, 1993.