Mathematical programming based debugging

Leo Liberti, Stéphane Le Roux [1,2]

LIX, École Polytechnique, 91128 Palaiseau, France

Jeremy Leconte [3]

DI, École Normale Supérieure, 45 Rue d'Ulm, 75230 Paris, France

Fabrizio Marinelli [4]

DIIGA, Università Politecnica delle Marche, Ancona, Italy

Abstract

Verifying that a piece of software has no bugs means proving that it has certain desired properties, such as an array index not taking values outside certain bounds. Abstract interpretation is used in the static analysis of code to establish the inclusion-wise smallest set of values (numerical invariant) that the program variables can attain during program execution. Such sets can be used to detect run-time errors without actually running the program. We present a mathematical program that determines guaranteed smallest interval invariants of computer programs with integer affine arithmetics and compare our results to existing techniques.

Keywords: verification, static analysis, abstract interpretation, reformulation.

[1] Partially supported by grants: System@tic "EDONA"and an Île-de-France postdoctoral fellowship.

[2] E-mail:`{liberti,leroux}@lix.polytechnique.fr`. Corresponding author: `liberti`.

[3] E-mail:`jeremy.leconte1@ens.fr`

[4] E-mail:`marinelli@diiga.univpm.it`

1 Introduction

Static Analysis (SA) by Abstract Interpretation (AI) [4,5] aims to find program invariants as over-approximations (also called *abstract semantics*) of the sets of values (also called *concrete semantics*) that the program variables can take at each control point of the program during the whole execution. We usually restrict abstract semantics to belong to a pre-specified class of sets, e.g. intervals, spheres, polyhedra and so on. Given one such class $\mathcal{L}$ and a lattice $(\mathcal{L}, \subseteq)$, the action of the program can be seen as a function F from $\mathcal{L}$ to itself. Thus a *domain* $X \in \mathcal{L}$ is invariant with respect to F if it does not change when F is applied to it. In other words, it must obey the fixpoint equations:

$$X = F(X), \tag{1}$$

usually called *semantic equations*. In particular, the least fixed point of F in $\mathcal{L}$ is the smallest invariant (for the given domain type) of the computer program encoded by F. Invariants are used to verify given properties of computer programs, such as for example "the variable $\mathtt{x}_i$ never exceeds the bounds $[0, 10]$": if we are able to show that the smallest invariant for $\mathtt{x}_i$ is, say, $[1, 5]$, then we are sure that the property is verified. This should also explain why large invariants are less interesting: the interval $[-\infty, \infty]$ might be an invariant, but it can only prove the trivial property $\mathtt{x}_i \in [-\infty, \infty]$.

Two well-known solution methods for (1) are Kleene's Iteration (KI) [4] and Policy Iteration (PI) [2,6,7]. KI is an iterative, possibly unbounded procedure based on applying F to the largest possible domain in the $\mathcal{L}$ until convergence to a fixed point is attained. PI is a sort of "Newton's method" borrowed from Markov Decision Processes [10,12] and adapted to lattices, which only converges to a guaranteed least fixed point under some additional conditions on F, namely non-expansiveness, playing the same role as convexity in the traditional Newton's method. The alternative approach proposed in this paper (limited for now to interval domains and addressing imperative languages) consists in using Mathematical Programming (MP) for describing the feasible set of (1) and employing a standard Branch-and-Bound (BB) algorithm to solve it exactly. We remark that mathematical programming and numerical optimization techniques were previously employed in software verification [3,13,15] but in different contexts.

The main innovation proposed in this paper is that, at least for computer programs with integer affine arithmetic, our approach provides *both* an optimality guarantee for all such programs *and* a finite, though exponential, bound on the computation time, which is an improvement with respect to

KI or PI (we remark that PI might also be naturally extended to work on expansive programs in exponential time, but such an extension was never described). The second innovation is that we use an essentially static modelling language (namely MP) to describe the dynamic execution of a computer program: modelling occurs recursively on the program operators translated into semantic equations. Furthermore, this work establishes an interesting, precisely defined relation between an imperative language (C) and a declarative one (MP).

2 The computational model

It is well known that flowcharts (i.e., graph representations of computer programs) are Turing-equivalent to a Universal Turing Machine (UTM) [9]. It is not too difficult to show that Turing-equivalence is not lost if we require that no flowchart node has more than two incoming arcs. Given such a flowchart (also called a *program graph*) $G = (V, A)$ representing a computer program, where V is the set of control points of the program and $A = \{a_1, \ldots, a_m\}$ is the set of flow-carrying arcs in the program, we assign a sequence of intervals X_i to each arc a_i, for all $i \leq m$. The interval X_i is an over-approximation of the set of values taken by variable `x` on the arc i over the whole program execution. For the sake of clarity we describe the computational model for computer programs with only one variable; when more then one variable are involved, the mathematical model can be easily updated in order to deal with unreachability issues.

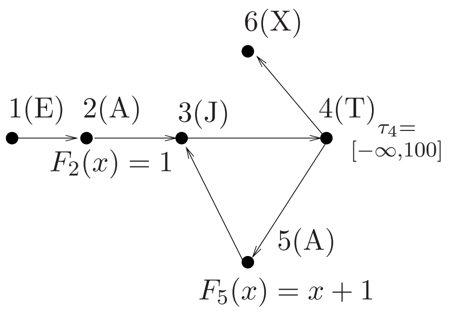
Control points in the program are assigned one of the following labels: *E*ntry, e*X*it, *A*ssignment, *J*oin (i.e. the loop start), *T*est. An operator F_v is assigned to each control point $v \in V$ according to its label. For every flow arc $a_i = (v, u)$, we state the rules that change X_i according to the program as $X_i = F_v(X)$, where $X = (X_1, \ldots, X_m)$. Since each arc has exactly one head vertex, we can index the operators by arc i instead of control point v, so that we obtain the fixpoint equations (1) in the form:

$$\forall i \leq m \quad X_i = F_i(X). \tag{2}$$

Notationwise, we let $F = (F_1 \ldots, F_m)$. An example is given in Fig. 1.

The operator for labels *E*,*X* is the identity Id, the operators for label A are the integer interval arithmetic operators $+, c\times, \uparrow d, \times, 1\div$ (where $+, \times$ are binary operators, $c\times$ is the constant multiplication, $\uparrow d$ is the power to constant, c and $d > 0$ are integer constants) [8], the operator for label J is intervalwise $\cup$ (i.e. the union of two disjoint intervals is the smallest interval

```
//(1)
int x = 1; //(2)
//(3)
while(x <= 100){ //(4)
  x = x + 1; //(5)
}
//(6)
```



$n = 1$; $a_1 = (1,2), a_2 = (2,3), a_3 = (3,4), a_4 = (4,5), a_5 = (4,6), a_6 = (5,3)$

$$\begin{aligned} X_1 &= \mathrm{Id}(\texttt{input}) \\ X_2 &= F_2(X_1) \\ X_3 &= X_2 \cup X_6 \\ X_4 &= X_3 \cap \tau_4 \\ X_5 &= X_3 \cap (X \smallsetminus \tau_4) \\ X_6 &= F_5(X_4). \end{aligned}$$

$$\begin{aligned} X_1 &= [-\infty, \infty] \\ X_2 &= [1, 1] \\ X_3 &= [1, 1] \cup X_6 \\ X_4 &= X_3 \cap [-\infty, 100] \\ X_5 &= X_3 \cap [101, \infty] \\ X_6 &= X_4 + [1, 1]. \end{aligned}$$

Fig. 1. A simple example: program graph, concrete semantic equations, abstract semantic equations.

containing them) and the operator for label T is $\cap$.

3 The mathematical program

We only consider intervals in the inclusion-wise interval lattice $(\mathcal{I}(M), \subseteq)$ of all integer intervals in $[-M, M]$ for some constant $M > 0$. For all $i \leq m$, we represent the interval X_i by a triplet $(x_i^L, x_i^U, \bar{x}_i) \in \mathbb{Z}^2 \times \{0, 1\}$ (subject to $x_i^L \leq x_i^U$) such that $X_i = [x_i^L, x_i^U]$ if and only if $\bar{x}_i = 0$ and $X_i = \emptyset$ otherwise. We also define a width $|X_i| = x_i^U - x_i^L$ if $\bar{x}_i = 0$ and $|X_i| = -1$ otherwise, and extend it to $|X| = \sum_i |X_i|$. This width function is such that the bottom element of any sublattice of $\mathcal{I}(M)$ is minimum in the width function restricted to the sublattice. It is not difficult to establish that all the considered operators are $\subseteq$-monotonic in $\mathcal{I}(M)$. By Tarski's lattice fixpoint theorem [14], the least fixpoint of (2) is

$$\operatorname{argmin}\{|X| : X \supseteq F(X)\}. \tag{3}$$

Eq. (3) can be used to construct a mathematical program as follows. For every operator F_i appearing in the computer program, we define the set

$\{X \mid X_i \supseteq F_i(X)\}$ in terms of inequality constraints $g^i(x^L, x^U, \bar{x}, z) \leq 0$ involving the decision variables $x^L, x^U, \bar{x}$ and possibly some added binary decision variables z for controlling the relative ordering of the intervals and whether an interval bound exceeds $-M, M$. For brevity, we only present here three operators: $+, \cup, \cap$.

3.1 Sum

The semantic of the sum operator $X_i = X_h + X_k$ in the arithmetic of intervals must be extended to the set of closed intervals in $\mathbb{Z} \cup \{\pm\infty\}$. To this aim the following binary variables and constraints are needed:

- $z_+^{Lh} = 1$ if and only if $x_h^L > -\infty$;
- $z_+^{Uh} = 1$ if and only if $x_h^U < +\infty$;
- $z_+^{Lk} = 1$ if and only if $x_k^L > -\infty$;
- $z_+^{Uk} = 1$ if and only if $x_k^U < +\infty$;
- $z_+^L = 1$ if $x_h^L = -\infty$ or $x_k^L = -\infty$;
- $z_+^U = 1$ if $x_h^U = +\infty$ or $x_k^U = +\infty$.

$$
\begin{array}{ll}
(4) & 1 - M(3 - 2z_+^{Lh}) \leq x_h^L \leq M(2z_+^{Lh} - 1) \\
(5) & M(1 - 2z_+^{Uh}) \leq x_h^U \leq M(3 - 2z_+^{Uh}) - 1 \\
(6) & 1 - M(3 - 2z_+^{Lk}) \leq x_k^L \leq M(2z_+^{Lk} - 1) \\
(7) & M(1 - 2z_+^{Uk}) \leq x_k^U \leq M(3 - 2z_+^{Uk}) - 1 \\
(8) & 2z_+^L \geq 2 - z_+^{Lh} - z_+^{Lk} \\
(9) & 2z_+^U \geq 2 - z_+^{Uh} - z_+^{Uk}
\end{array}
$$

$$
\begin{array}{ll}
(10) & x_i^L \leq (x_h^L + x_k^L)(1 - z_+^L) \\
 & \qquad -Mz_+^L + 2M\bar{x}_i \\
(11) & x_i^U \geq (x_h^U + x_k^U)(1 - z_+^U) \\
 & \qquad +Mz_+^U - 2M\bar{x}_i \\
(12) & 2\bar{x}_i \geq \bar{x}_h + \bar{x}_k \\
(13) & \bar{x}_i \leq \bar{x}_h + \bar{x}_k.
\end{array}
$$

Observe that Constraints (10) and (11) are needed to guarantee model feasibility since they correctly allow the operations $x_i^U + M = M$ and $x_i^L - M = -M$. Moreover, it is easy to provide cases having least fixpoints with at least one interval that diverges to infinity.

3.2 Union

The $\cup$ operator $X_i = X_h \cup X_k$ is modelled as follows:

$$
\begin{array}{lll}
\bar{x}_i = \bar{x}_h \bar{x}_k, & x_i^L \leq x_h^L + 2M\bar{x}_i, & x_i^L \leq x_k^L + 2M\bar{x}_i \\
 & x_i^U \geq x_h^U - 2M\bar{x}_i, & x_i^U \geq x_k^U - 2M\bar{x}_i.
\end{array}
$$

3.3 Intersection

The $\cap$ operator $X_i = X_h \cap X_k$ has the following properties:

(i) if $x_h^U < x_k^L$ then $X_i = \emptyset$;
(ii) if $x_h^L > x_k^U$ then $X_i = \emptyset$;
(iii) if $x_h^L \leq x_k^L$ and $x_h^U \leq x_k^U$ and $x_h^U \geq x_k^L$ then $X_i = [x_k^L, x_h^U]$;
(iv) if $x_h^L \leq x_k^L$ and $x_h^U \geq x_k^U$ then $X_i = [x_k^L, x_k^U]$;
(v) if $x_h^L \geq x_k^L$ and $x_h^U \leq x_k^U$ then $X_i = [x_h^L, x_h^U]$;
(vi) if $x_h^L \geq x_k^L$ and $x_h^U \geq x_k^U$ and $x_h^L \leq x_k^U$ then $X_i = [x_h^L, x_k^U]$.

Moreover X_i is empty if at least one between X_h and X_k is empty. The intersection $X_h \cap X_k$ can be modeled by the following binary variables and constraints:

- $z_\cap^{UL} = 1$ if and only if $x_h^U < x_k^L$ (case i.);
- $z_\cap^{LU} = 1$ if and only if $x_h^L > x_k^U$ (case ii.);
- $z_\cap^{Lk} = 1$ if and only if $x_i^L = x_k^L$ (cases iii. or iv.);
- $z_\cap^{Lh} = 1$ if and only if $x_i^L = x_h^L$ (cases v. or vi.);
- $z_\cap^{Uh} = 1$ if and only if $x_i^U = x_h^U$ (cases iii. or v.);
- $z_\cap^{Uk} = 1$ if and only if $x_i^U = x_k^U$ (cases iv. or vi.).

(14) $(1 - z_\cap^{UL})(x_k^L - x_h^U) \leq 0$
(15) $z_\cap^{UL}(x_k^L - x_h^U - 1) \geq 0$
(16) $(1 - z_\cap^{LU})(x_h^L - x_k^U) \leq 0$
(17) $z_\cap^{LU}(x_h^L - x_k^U - 1) \geq 0$
(18) $\bar{x}_h + \bar{x}_k \leq 2\bar{x}_i$
(19) $z_\cap^{UL} + z_\cap^{LU} \leq 2\bar{x}_i$
(20) $z_\cap^{UL} + z_\cap^{LU} + \bar{x}_h + \bar{x}_k \geq \bar{x}_i$
(21) $z_\cap^{Lk} + z_\cap^{Lh} + \bar{x}_i = 1$
(22) $z_\cap^{Uh} + z_\cap^{Uk} + \bar{x}_i = 1$
(23) $z_\cap^{Lk}\left(x_i^L - x_k^L\right) = 0$
(24) $z_\cap^{Lh}\left(x_i^L - x_h^L\right) = 0$
(25) $z_\cap^{Uk}\left(x_i^U - x_k^U\right) = 0$
(26) $z_\cap^{Uh}\left(x_i^U - x_h^U\right) = 0.$

3.4 Mathematical programming classes

If the computer program only uses integer affine arithmetic, the output is a MINLP that can be reformulated exactly to a MILP and solved in practice (provided a correct choice of M) using BB. With integer non-affine arithmetic we obtain a MINLP that can be solved exactly by spatial Branch-and-Bound (sBB) [1]. If floating point arithmetic is used, then we need to introduce a "small" constant $\varepsilon > 0$ that, even if precisely chosen, would yield only over-approximated solutions.

4 Implementation

We implemented a C parser (recognizing a subset of C which is sufficiently rich to be Turing-universal) that outputs the corresponding MP. We compared our results to the PI algorithm on several (small) C programs[5] with integer affine arithmetic, yielding MILPs which we solved using CPLEX 11 [11] on a 2.4GHz Intel Xeon CPU with 8GB RAM. In all of them, we obtained fixed points of width equal or smaller than those obtained by PI, thus validating the approach. CPU time-wise, we are slower than PI by a factor of around 10 (the computational price of the optimality guarantee).

The comparison on the largest instances, qualitywise favourable to MP, is given below. We report the number of lines of code, variables and loops, and the maximum nesting level of loops. For the MP based approach we report the number of CPLEX simplex iterations and BB nodes, the CPU time in seconds and least fixpoint interval width, whereas we report the CPU time and interval width for the PI method (a '-' means PI failed to find a fixpoint; to be fair to PI, we used a very old implementation, the only one which was made available to us for testing).

Instance					CPLEX 11				Policy Iteration	
name	lines	vars	loops	nesting	simplex	nodes	CPU (sec.)	width	CPU (sec.)	width
PI1	13	2	1	1	29	0	0.017	**20398**	0.003	50392
PI2	14	2	2	2	278	46	0.042	**20084**	0.001	60048
PI3	13	2	2	2	83	0	0.026	**21374**	0.004	60582
arrays	22	6	2	1	56	0	0.068	**600139**	-	-
functions	62	11	3	1	509	58	0.144	**444430**	-	-
fun+arrays	53	10	2	2	96	0	0.048	**340105**	-	-

References

[1] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4):597–634, 2009.

[2] A. Costan, S. Gaubert, E. Goubault, M. Martel, and S. Putot. A policy iteration algorithm for computing fixed points in tatic analysis of programs. In K. Etessami and S.K. Rajamani, editors, *Computer Aided Verification*, volume 3576 of *LNCS*, pages 462–475. Springer, 2005.

[5] `http://www.lix.polytechnique.fr/~liberti/verif-instances.zip`

[3] P. Cousot. Proving program invariance and termination by parametric abstraction, lagrangian relaxation and semidefinite programming. In R. Cousot, editor, *Verification, Model Checking and Abstract Interpretation*, volume 3385 of *LNCS*, pages 17–19. Springer, 2005.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction of approximations of fixed points. *Principles of Programming Languages*, 4:238–252, 1977.

[5] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, 1979. ACM Press, New York, NY.

[6] S. Gaubert, E. Goubault, A. Taly, and S. Zennou. Static analysis by policy iteration on relational domains. In R. De Nicola, editor, *European Symposium on Programming (ESOP)*, volume 4421 of *LNCS*, pages 237–252. Springer, 2007.

[7] T. Gawlitza and H. Seidl. Precise fixpoint computation through strategy iteration. In R. De Nicola, editor, *European Symposium on Programming (ESOP)*, volume 4421 of *LNCS*, pages 300–315. Springer, 2007.

[8] E. Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, Inc., New York, 1992.

[9] D. Harel, P. Norvig, J. Rood, and T. To. A universal flowcharter. In *2nd Computers in Aerospace Conference*, volume A79-54378/24-59, pages 218–224, New York, 1979. AAIA.

[10] R. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, 1960.

[11] ILOG. *ILOG CPLEX 11.0 User's Manual.* ILOG S.A., Gentilly, France, 2008.

[12] M. Puterman and S. Brumelle. On the convergence of policy iteration in stationary dynamic programming. *Mathematics of Operations Research*, 4(1):60–69, 1979.

[13] M. Roozbehani, A. Megretski, and E. Feron. Convex optimization proves software correctness. In *Proceedings of the American Control Conference*, 2005.

[14] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[15] Hirotoshi Yasuoka and Tachio Terauchi. Polymorphic fractional capabilities. In Jens Palsberg and Zhendong Su, editors, *SAS*, volume 5673 of *Lecture Notes in Computer Science*, pages 36–51. Springer, 2009.