Combining Constraint Programming and Abstract Interpretation for Value Analysis of Floating-point Programs

Olivier Ponsini, Claude Michel and Michel Rueher
University of Nice–Sophia Antipolis, I3S/CNRS
BP 121, 06903 Sophia Antipolis Cedex, France
Email: firstname.lastname@unice.fr

***Abstract*—Abstract interpretation-based value analysis is a classical approach for verifying programs with floating-point computations. However, state-of-the-art tools compute an over-approximation of the variable values that can be very coarse. Constraint solvers have recently been used to significantly refine the approximations computed by such tools. In this paper, we introduce a hybrid approach that combines abstract interpretation and constraint programming techniques in a single static and automatic analysis. First experiments showed that this approach can successfully analyze programs that could not be handled by abstract interpretation or constraint programming tools alone.**

***Keywords*-program verification; abstract interpretation; constraint solving; floating-point computation**

I. Introduction

Value analysis consists in computing the set of all possible values for the variables of a program. In program verification, this analysis is often used to check the absence of run-time error, such as invalid integer or floating-point operations, as well as simple user assertions [1]. Here, we focus on programs with floating-point computations. Such programs control complex and critical physical systems in various domains as transportation, nuclear energy, or medicine. Floating-point computations are error-prone because of the many pitfalls of floating-point arithmetic [2]. Floating-point computations are usually derived from mathematical models on real numbers. As real and floating-point computation models are different, value analysis can also help in estimating the precision of floating-point computations with respect to the same sequence of operations in an idealized semantics of real numbers.

FLUCTUAT [3] is a static analyzer of C programs based on abstract interpretation that deals with floating-point and real numbers. However, it may roughly over-approximate the possible variable values for some programming constructs and expressions. We showed in [4] that constraint programming solvers can improve these results. However, the proposed approach requires an exhaustive analysis of all execution paths in a program.

Here, we propose a hybrid approach for the value analysis of floating-point programs. More precisely, we propose to combine abstract interpretation and constraint programming techniques in a single static and automatic analysis to avoid the combinatorial explosion of the number of paths to explore. Before going into the details, let us recall the main advantages of abstract interpretation and constraint programming.

II. Abstract Interpretation

Abstract interpretation captures a superset of all possible executions of a program. This allows to compute safe over-approximations of variable values. Systems like FLUCTUAT use the weakly relational abstract domain of zonotopes [5]. Zonotopes are sets of affine forms that improve over interval arithmetic: linear correlations between variables are preserved. They offer a good trade-off between performance and precision for floating-point and real number computations. Indeed the analysis:

- is fast and scales well;
- processes accurately linear expressions;
- computes accurate approximations of some unbounded loops;
- keeps track of the statements involved in the loss of accuracy of floating-point computations.

However, over-approximations computed by FLUCTUAT may be very large because the abstract domains used do not handle well conditional statements and non-linear expressions.

III. Constraint Programming

Over real numbers, constraint solvers compute safe approximations of continuous solution sets using correctly rounded interval methods and filtering algorithms. We developed FPCS [6], a constraint solver correct over floating-point numbers, i.e. it does not lose any solution. FPCS is based on $2B$-consistency [7] along with projection functions adapted to floating-point arithmetic.

In a previous work [4], we proposed to build a constraint system for each execution path in a program. This was done on-the-fly while parsing the program and non-executable paths were discarded as soon as the inconsistency of the associated constraint system was detected. So, we could take advantage of the refutation capabilities of filtering algorithms to refine the approximations of variable values computed by abstract interpretation.

This approach could reduce drastically the over-approximations computed by FLUCTUAT on programs with conditional statements or non-linear expressions. However, it did not scale well because of the combinatorial explosion of the number of paths to explore: all executable paths had to be explored and loops needed to be bounded and unfolded.

IV. Hybrid approach

Here, we propose to take advantage of the strengths of both techniques in a hybrid approach. The idea is to avoid the path combinatorial explosion of the constraint programming approach while still improving the accuracy of the abstract interpretation-based analysis. To this end, the variable values in each program conditional branch are merged as soon as the branches join, as in a standard abstract interpretation-based analysis. Let n be the number of conditional statements of a program, we explore at most $2n$ branches instead of the 2^n paths of our previous work.

For each piece of program comprised between two merge points, we call FLUCTUAT to compute a first approximation of the variable domains. Then, the piece of program is converted into constraints with the approximation from abstract interpretation as initial domains for the variables. We apply *shaving* techniques [7], [8] to reduce the domains. Over real numbers, we use the combination of hull and box consistencies implemented in REAL PAVER [9]. Over floating-point numbers, we use $2B$-consistency as implemented in FPCS.

The cooperation between abstract interpretation and constraint solvers allows to handle loops in two ways. On the one hand, FLUCTUAT unfolds a fixed number of times each loop. This may help to compute a better approximation for the whole loop. Constraint solving can then improve the approximation of abstract interpretation for the unfolded part of the loop, and therefore for the whole loop. On the other hand, the approximation computed by abstract interpretation for a loop can be substituted to the loop in the constraint programming approach when the loop cannot be completely unfolded.

V. Discussion

The proposed hybrid approach relies on an analysis that explores each conditional branch in a program instead of each execution path as in our previous work [4]. As expected, this approach is faster than our previous one. However, it may be less accurate: merging variable values when conditional branches join may yield an over-approximation for some execution paths. The converse is observed with respect to abstract interpretation alone: the hybrid approach is slower but more accurate. More interesting, the hybrid approach may compute good approximations of variable values of programs for which neither the abstract interpretation nor the constraint programming approach alone can.

As a perspective, the collaboration between abstract interpretation and constraint programming could be tighter. In [10], the authors devised an extension to the abstract domain of zonotopes that integrates constraints. The collaboration could then take place at the level of the abstract domain instead of the variable values.

References

[1] P. Cousot, R. Cousot, J. Feret, A. Miné, L. Mauborgne, D. Monniaux, and X. Rival, “Varieties of static analyzers: A comparison with ASTRÉE,” in *First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE’07)*. IEEE Computer Society, 2007, pp. 3–20.

[2] D. Goldberg, “What every computer scientist should know about floating point arithmetic,” *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.

[3] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine, “Towards an industrial use of fluctuat on safety-critical avionics software,” in *14th International Workshop on Formal Methods for Industrial Critical Systems (FMICS’09)*, ser. Lecture Notes in Computer Science, vol. 5825. Springer, 2009, pp. 53–69.

[4] O. Ponsini, C. Michel, and M. Rueher, “Refining abstract interpretation-based approximations with a floating-point constraint solver,” in *Fourth International Workshop on Numerical Software Verification*, 2011, http://users.polytech.unice.fr/˜rueher/Publis/nsv11.pdf.

[5] E. Goubault and S. Putot, “Static analysis of numerical algorithms,” in *13th International Symposium on Static Analysis (SAS’06)*, ser. Lecture Notes in Computer Science, vol. 4134. Springer, 2006, pp. 18–34.

[6] C. Michel, “Exact projection functions for floating-point number constraints,” in *7th International Symposium on Artificial Intelligence and Mathematics (AIMA’02)*, 2002, http://rutcor.rutgers.edu/ amai/aimath02/PAPERS/21.ps.

[7] O. Lhomme, “Consistency techniques for numeric CSPs,” in *13th International Joint Conference on Artificial Intelligence (IJCAI’93)*, 1993, pp. 232–238.

[8] P. Martin and D. B. Shmoys, “A new approach to computing optimal schedules for the job-shop scheduling problem,” in *5th International IPCO Conference*. Springer, 1996, pp. 389–403.

[9] L. Granvilliers and F. Benhamou, “Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques,” *ACM Transactions on Mathematical Software*, vol. 32, no. 1, pp. 138–156, 2006.

[10] K. Ghorbal, E. Goubault, and S. Putot, “A logical product approach to zonotope intersection,” in *22nd International Conference on Computer Aided Verification (CAV’10)*, ser. Lecture Notes in Computer Science, vol. 6174. Springer, 2010, pp. 212–226.