A Mixed Real and Floating-Point Solver

Rocco Salvia[1], Laura Titolo[2], Marco A. Feliú[2], Mariano M. Moscato[2], César A. Muñoz[3], and Zvonimir Rakamarić[1]

[1] University of Utah,
{rocco,zvonimir}@cs.utah.edu
[2] National Institute of Aerospace,
{laura.titolo,marco.feliu,mariano.moscato}@nianet.org
[3] NASA Langley Research Center,
cesar.a.munoz@nasa.gov

Abstract. Reasoning about mixed real and floating-point constraints is essential for developing accurate analysis tools for floating-point programs. This paper presents FPRoCK, a prototype tool for solving mixed real and floating-point formulas. FPRoCK transforms a mixed formula into an equi-satisfiable one over the reals. This transformed formula is then input to different off-the-shelf SMT solvers for resolution. The proposed tool is integrated inside the static analyzer PRECiSA, which computes a sound estimation of the round-off error of a floating-point program. FPRoCK is used to detect unfeasible computational paths, improving the accuracy of PRECiSA.

1 Introduction

Floating-point numbers are frequently used as an approximation of real numbers in computer programs. A round-off error originates with the difference between a real number and its floating-point representation, and accumulates throughout a computation. The resulting error may affect both the computed value of arithmetic expressions, as well as the control flow of the program.

To reason about floating-point computations with possibly diverging control flows, it is essential to solve mixed real and floating-point arithmetic constraints. This is known to be a difficult problem. In fact, constraints that do not have a solution over the reals may hold over the floats and vice-versa. In addition, combining the theories is not trivial since floating-point arithmetic and real arithmetic do not enjoy the same properties.

Modern SMT solvers such as Mathsat [3] and Z3 [10] encode floating-point numbers with bit-vectors. This technique is usually inefficient due to the size of the binary representation of floating-point numbers. For this reason, several abstraction techniques have been proposed to approximate floating-point formulas and to solve them in the theory of real numbers. Approaches based on the Counter Example Guided Abstraction Refinement (CEGAR) framework [2,13,17] simplify a floating-point formula and solve it in a proxy theory that is more efficient than the original one. If a solution is found for the simplified

formula, a check on whether it is also a solution for the original formula is performed. If the answer is yes, the solution is returned, otherwise, the proxy theory is refined. Realizer [8] is a framework built on the top of Z3 to solve floating-point formulas by translating them into equivalent ones in real arithmetic. Molly [13] implements a CEGAR loop where floating-point constraints are lifted in the proxy theory of mixed real and floating-point arithmetics. To this aim, it uses an extension of Realizer that supports mixed real and floating-point arithmetics. However, this extension is embedded in Molly and cannot be used as a standalone tool. The constraint solver Colibri [9] handles the combination of real and floating-point arithmetics by using disjoint floating-point intervals and difference constraints. Unfortunately, the publicly available version of Colibri does not provide support for all the rounding modalities and for the negation of Boolean formulas.

This paper presents FPRoCk, a prototype solver of mixed real and floating-point constraints. The prototype tool is in the process of being released under NASA's Open Source Agreement. FPRoCk extends the transformation defined in [8] for Realizer to mixed real/floating-point constraints. Given a mixed real-float formula, FPRoCk computes an equi-satisfiable real arithmetic formula that can be solved by an external SMT solver. In constrast to Realizer, FPRoCk supports mixed precision floating-point expressions and different ranges for the input variables. This paper illustrates how FPRoCk is employed to improve the accuracy of the static analyzer PRECiSA [15]. In particular, the proposed tool identifies spurious execution traces whose path conditions are unsatisfiable, allowing PRECiSA to discard them.

2 Solving mixed Real/Floating-Point Formulas

A *floating-point number* [7], or simply a *float*, can be represented by a tuple (s, m, exp) where s is a sign bit, m is an integer called the *significand* (or *mantissa*), and exp is an integer *exponent*. A float (s, m, exp) encodes the real number $(-1)^s \cdot m \cdot 2^{exp}$. Henceforth, $\mathbb{F}$ represents the set of floating-point numbers. Let $\tilde{v}$ be a floating-point number that represents a real number r. The difference $|\tilde{v} - r|$ is called the *round-off error* (or *rounding error*) of $\tilde{v}$ with respect to r. Each floating-point number has a format f that specify its dimensions and precision, i.e., single or double. The expression $F_f(r)$ denotes the floating-point number in format f *closest* to r assuming a given rounding mode.

Let $\mathbb{V}$ and $\widetilde{\mathbb{V}}$ be two disjoint sets of variables representing real and floating-point values respectively. The sets $\mathbb{A}$ of mixed arithmetic expressions is defined by the following grammar.

$$A ::= d \mid x \mid \tilde{d} \mid \tilde{x} \mid A \odot A \mid A \widetilde{\odot} A \mid F_f(A),$$

where $d \in \mathbb{R}$, $x \in \mathbb{V}$, $\odot \in \{+, -, *, /, |\cdot|\}$ (the set of basic real number arithmetic operators), $\tilde{d} \in \mathbb{F}$, $\tilde{x} \in \widetilde{\mathbb{V}}$, $\widetilde{\odot} \in \{\tilde{+}_f, \tilde{-}_f, \tilde{*}_f, \tilde{/}_f\}$ (the set of basic floating-point arithmetic operators) and $f \in \{single, double\}$ denotes the desired precision for the result. The rounding operator F_f is naturally extended to arithmetic expressions.

According to the IEEE-754 standard [7], each floating-point operation shall be computed in exact real arithmetic and then rounded to the nearest float, i.e., $A \widetilde{\odot}_f A = F_f(A \odot A)$. Since floating-point numbers can be exactly represented as real numbers, an explicit transformation from floats to real numbers is not necessary. The set of mixed real-float Boolean expressions $\mathbb{B}$ is defined by the following grammar.

$$B ::= \mathit{true} \mid \mathit{false} \mid B \wedge B \mid B \vee B \mid \neg B \mid A < A \mid A = A,$$

where $A \in \mathbb{A}$.

The input to FPRoCk is a formula $\tilde{\phi} \in \mathbb{B}$ that may contain both real and floating-point variables and arithmetic operators. Each variable is associated to a type (real, single precision, or double precision floating-point) and to an initial range that can be bounded, e.g. $[1, 10]$, or unbounded, e.g., $[-\infty, +\infty]$. The precision of a mixed precision floating arithmetic operation is automatically detected and set to the maximum precision of its arguments. Given a mixed formula $\tilde{\phi}$, FPRoCk computes a formula ϕ over the reals such that $\tilde{\phi}$ and ϕ are equi-satisfiable. Floating-point expressions are transformed in equivalent real-valued expressions using the approach in [8] while the real variables and operators are left unchanged. It is possible to define $x \widetilde{\odot} y$ as follows [8]:

$$x \widetilde{\odot} y = \left(\frac{\rho\left(\frac{x \odot y}{2^{exp}} \cdot 2^p\right)}{2^p} \right) \cdot 2^{exp}, \tag{2.1}$$

where p is the precision of the format, $exp = max\{i \in \mathbb{Z} \mid 2^i \leq |x \odot y|\}$, and $\rho : \mathbb{R} \to \mathit{Int}$ is a function implementing the rounding modality. Therefore, given a floating-point formula $\tilde{\phi}$, an equi-satisfiable formula without floating-point operators is obtained by replacing every occurrence of $x \widetilde{\odot} y$ using Formula (2.1). This is equivalent to replacing the occurrences of $x \widetilde{\odot} y$ with a new fresh real-valued variable v and imposing $v = x \widetilde{\odot} y$. By Equation (2.1), it follows that $v \cdot 2^{p-exp} = F((x \odot y) \cdot 2^{p-exp})$. Thus, the final formula ϕ is obtained as follows:

$$\phi := \tilde{\phi}[v / x \widetilde{\odot} y] \wedge v \cdot 2^{p-exp} = \rho((x \odot y) \cdot 2^{p-exp}). \tag{2.2}$$

where $\tilde{\phi}[v / x \widetilde{\odot} y]$ denotes the Boolean formula $\tilde{\phi}$ where all the occurrences of $x \widetilde{\odot} y$ are replaced by v. The precision p is a constant that depends on the chosen floating-point format, while exp is an integer representing the exponent of the binary representation of $x \widetilde{\odot} y$. In order to search an assignment for the exponent exp, FPRoCk performs in parallel a sequential and a binary search on the dimension of $x \widetilde{\odot} y$, as opposed to the simple sequential search implemented in Realizer. The implementation of the function ρ depends on the selected rounding-mode and can be defined by means of floor and ceiling operators (see [8] for details). Therefore, the transformed formula ϕ is free of any floating-point operator and can be solved by any SMT solver that supports the fragment of real/integer arithmetics handling floors and ceilings. FPRoCk uses three off-the-shelf SMT solvers as back-end procedures to solve the transformed formula: Mathsat [3], Z3 [10], and CVC4 [1]. Optionally, the constraint solver Colibri [9] is also available for use within FPRoCk.

FPRoCk provides the option to relax the restriction on the minimum exponent to handle subnormal floats. This solution is sound in the sense that it preserves the unsatisfiability of the original formula. However, if this option is used, it is possible that FPRoCk finds an assignment to a float that is not representable in the chosen precision, therefore this assignment is not a solution for the original formula.

3 Integrating FPRoCk in PRECiSA

PRECiSA[4] (Program Round-off Error Certifier via Static Analysis) [15] is a static analyzer based on abstract interpretation [4] that estimates round-off errors of floating-point programs. PRECiSA accepts as input a floating-point program and automatically generates a sound over-approximation of the floating-point round-off error and a proof certificate in the Prototype Verification System (PVS) [12] ensuring its correctness.

For every possible combination of real and floating-point execution paths, PRECiSA computes a *conditional error bound* of the form $\langle \eta, \widetilde{\eta} \rangle \twoheadrightarrow (r, e)$, where η is a symbolic path condition over the reals; $\widetilde{\eta}$ is a symbolic path condition over the floats; r and e are symbolic arithmetic expressions over the reals. Intuitively, $\langle \eta, \widetilde{\eta} \rangle \twoheadrightarrow (r, e)$ indicates that if the conditions η and $\widetilde{\eta}$ are satisfied, the output of the program using exact real number arithmetic is r and the round-off error of the floating-point implementation is bounded by e.

PRECiSA initially computes round-off error estimations in symbolic form so that the analysis is modular. Given the initial ranges for the input variables, PRECiSA uses the Kodiak global optimizer [11] to maximize the symbolic error expression e. Since the analysis collects information about real and floating-point execution paths, it is possible to consider the error of taking the incorrect branch compared to the ideal execution using real arithmetic. This happens when the guard of a conditional statement contains a floating-point expression whose round-off error makes the actual Boolean value of the guard differ from the value that would be obtained assuming real arithmetic. When the floating-point computation diverges from the real one, it is said to be *unstable*.

Example 1 (Sign). Consider the function $sign(\tilde{x}) = $*if* $\tilde{x} \geq 0$ *then* 1 *else* -1. PRECiSA computes a set of four different conditional error bounds: $\{\langle \chi_r(\tilde{x}) \geq 0, \tilde{x} \geq 0 \rangle \twoheadrightarrow (r = 1, e = 0), \langle \chi_r(\tilde{x}) < 0, \tilde{x} < 0 \rangle \twoheadrightarrow (r = -1, e = 0), \langle \chi_r(\tilde{x}) \geq 0, \tilde{x} < 0 \rangle \twoheadrightarrow (r = -1, e = 2), \langle \chi_r(\tilde{x}) < 0, \tilde{x} \geq 0 \rangle \twoheadrightarrow (r = 1, e = 2)\}$. The function $\chi_r : \widetilde{\mathbb{V}} \to \mathbb{V}$ associates to the floating-point variable $\tilde{x}$ a variable $x \in \mathbb{V}$ representing the real value of $\tilde{x}$. The first two elements correspond to the cases where real and floating-point computational flows coincide. In these cases, the error is 0 since the output is an integer number with no rounding error. The other two elements model the unstable paths. In these cases, the error is 2, which corresponds to the difference between the output of the two branches.

[4] The PRECiSA distribution is available at `https://github.com/nasa/PRECiSA`.

PRECiSA may produce conditional error bounds with unsatisfiable symbolic conditions (usually unstable), which correspond to execution paths that cannot take place. The presence of these spurious elements affects the accuracy of the computed error bound. For instance, in Example 1, if $|\chi_r(\tilde{x}) - \tilde{x}| \leq 0$, both unstable cases can be removed, and the overall error would be 0 instead of 2.

Real and floating-point conditions can be checked separately by using SMT solvers that support real and/or floating-point arithmetic. However, the inconsistency often follows from the combination of the real and floating-point conditions. In fact, the floating-point expressions occurring in the conditions are implicitly related to their real arithmetic counterparts by the rounding-error. Therefore, besides checking the two conditions separately, it is necessary to check them in conjunction with a set of constraints relating each arithmetic expression $\widetilde{expr}$ occurring in the conditions with its real number counterpart $R_{\mathbb{A}}(\widetilde{expr})$. $R_{\mathbb{A}}(\widetilde{expr})$ is defined by simply replacing in $\widetilde{expr}$ each floating-point operation with the corresponding one on real numbers and by applying χ_r to floating-point variables.

FPRoCk is suitable for solving these constraints thanks to its ability to reason about mixed real and floating-point formulas. For each conditional error bound $c = \langle \eta, \widetilde{\eta} \rangle_t \twoheadrightarrow (r, e)$ computed by PRECiSA, a formula ψ modeling the information contained in the path conditions is passed to FPRoCk. Given a set ι of ranges for the input variables, for each conditional error bound $c = \langle \eta, \widetilde{\eta} \rangle_t \twoheadrightarrow (r, e)$ computed by PRECiSA, a formula in the following form is generated and checked by FPRoCk.

$$\psi := \eta \wedge \widetilde{\eta} \wedge \bigwedge \{ |\widetilde{expr} - R_{\mathbb{A}}(\widetilde{expr})| \leq \epsilon \mid \widetilde{expr} \text{ occurs in } \widetilde{\eta}, \quad \widetilde{expr} \notin \widetilde{\mathbb{V}}, \widetilde{expr} \notin \mathbb{F}, \epsilon = max(e)|_\iota \} \tag{3.1}$$

The value $max(e)|_\iota$ is the round-off error of $\widetilde{expr}$ assuming the input ranges in ι and it is obtained by maximizing the symbolic error expression e with the Kodiak global optimizer [11]. If ψ is unsatisfiable, then c is dropped from the solutions computed by PRECiSA. Otherwise, a counterexample is generated that may help discovering cases for which the computation is diverging or unsound.

Since FPRoCk is limited to the basic arithmetic operators whereas PRECiSA provides support for a broader variety of operators including transcendental functions, a sound approximation is needed for converting PRECiSA conditions in a valid input for FPRoCk. The proposed approach replaces in ψ each floating-point (respectively real) arithmetic expression with a fresh floating-point (respectively real) variable. This approach is sound but not complete, therefore it preserves just the unsatisfiability of the original formula. In other words, if $\psi[v_i/\widetilde{expr}_i]_{i=1}^n$ is unsatisfiable it follows that ψ is unsatisfiable, but if a solution is found for $\psi[v_i/\widetilde{expr}_i]_{i=1}^n$ there is no guarantee that an assignment satisfying ψ exists. This is enough for the purpose of eliminating spurious conditional bounds since it is assured that no genuine condition is eliminated. In practice, it is accurate enough to detect spurious unstable paths.

When a path condition is deemed unsatisfiable by FPRoCk, PRECiSA states such unsatisfiability in the PVS formal certificate. For simple path conditions,

	PRECiSA		PRECiSA+FPRoCk		Rosa	
	error	time	error	time	error	time
cubicSpline	2.70E+01	**0.074**	2.70E+01	97.76	**2.50E-01**	24.101
eps_line	2.00E+00	**0.02**	**1.00E+00**	48.797	2.00E+00	15.468
jetApprox	1.51E+01	**12.768**	8.11E+00	263.286	**4.97E+00**	924.835
linearFit	1.08E+00	**0.056**	5.42E-01	259.702	**3.19E-01**	12.384
los	2.00E+00	**0.02**	**1.00E+00**	46.173	not supported	n/a
quadraticFit	3.68E+00	**0.898**	3.68E+00	259.818	**1.27E-01**	82.405
sign	2.00E+00	**0.018**	**1.00E+00**	32.12	2.00E+00	4.741
simpleInterpolator	2.25E+02	**0.026**	1.16E+02	93.813	**3.33E+01**	6.289
smartRoot	**1.75E+00**	**0.32**	**1.75E+00**	0.624	not supported	n/a
styblinski	9.35E+01	**1.062**	6.66E+01	260.134	**6.55E+00**	77.031
tau	8.40E+06	**0.03**	**8.00E+06**	101.754	8.40E+06	20.672

Table 1. Experimental results for absolute round-off error bounds and execution time in seconds (best results in bold).

this property can be automatically checked by PVS. Unfortunately, there are cases where human intervention is required to verify this part of the certificates.

Table 1, compares the original version of PRECiSA with the enhanced version that uses FPRoCk to detect the unsatisfiable conditions, along with the tool Rosa [6] which also handles unstable conditionals. All the benchmarks contain conditionals and are obtained by applying the transformation defined in [16] to fragments of code from avionics software and from the FPBench library [5]. The transformed program is guaranteed to return either the result of the original floating-point program, when it can be assured that both its real and its floating-point flows agree, or a warning when these flows may diverge.

4 Conclusion

This paper presents FPRoCk, a prototype tool for solving mixed real and floating-point formulas. FPRoCk extends the technique used in Realizer by adding support for these mixed formulas. The paper illustrates the integration of FPRoCk into PRECiSA, a static analyzer of floating-point programs, to improve its precision. Similarly, it can be integrated into other static analyzers, such as FPTaylor [14]. The current version of FPRoCk has some limitations in terms of expressivity and efficiency. It handles the basic arithmetic operators. Support for other operators, including transcendental functions, is contingent on the expressive power of the underlying SMT solvers and will be explored in the future. FPRoCk has been tested on simple expressions with few variables. Finding an assignment for the exponent of each floating-point variable is still the major bottleneck of the analysis. This negatively impacts the overall performance of the prototype especially when the initial ranges for the variables are large. The use of a branch-and-bound search to divide the state-space may help to mitigate this problem. In addition, computing the ranges of the intermediate results of the floating-point operations should also improve the scalability of the proposed approach.

References

1. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011). Springer-Verlag (2011)
2. Brillout, A., Kroening, D., Wahl, T.: Mixed abstractions for floating-point arithmetic. In: 9th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2009). pp. 69–76 (2009)
3. Cimatti, A., Griggio, A., Schaafsma, B.J., R., S.: The MathSAT5 SMT Solver. In: 19th International Conference onTools and Algorithms for the Construction and Analysis of Systems (TACAS 2013). Lecture Notes in Computer Science, vol. 7795, pp. 93–107. Springer (2013)
4. Cousot, P., Cousot, R.: Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In: Conference Record of tha 4th ACM Symposium on Principles of Programming Languages, POPL 1977. pp. 238–252. ACM (1977)
5. Damouche, N., Martel, M., Panchekha, P., Qiu, C., Sanchez-Stern, A., Tatlock, Z.: Toward a standard benchmark format and suite for floating-point analysis. In: 9th International Workshop on Numerical Software Verification (NSV 2016). Lecture Notes in Computer Science, vol. 10152, pp. 63–77 (2016)
6. Darulova, E., Kuncak, V.: Sound compilation of reals. In: Proceedings of the 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2014. pp. 235–248. ACM (2014)
7. IEEE: IEEE standard for binary floating-point arithmetic. Tech. rep., Institute of Electrical and Electronics Engineers (2008)
8. Leeser, M., Mukherjee, S., Ramachandran, J., Wahl, T.: Make it real: Effective floating-point reasoning via exact arithmetic. In: Design, Automation & Test in Europe Conference & Exhibition, (DATE 2014). pp. 1–4 (2014)
9. Marre, B., Bobot, F., Chihani, Z.: Real Behavior of Floating Point Numbers. 15th International Workshop on Satisfiability Modulo Theories (SMT 2017) (2017)
10. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008)
11. Narkawicz, A., Muñoz, C.: A formally verified generic branching algorithm for global optimization. In: Revised Selected Papers of VSTTE 2013. Lecture Notes in Computer Science, vol. 8164, pp. 326–343. Springer (2013)
12. Owre, S., Rushby, J., Shankar, N.: PVS: A prototype verification system. In: Proceedings of CADE 1992. vol. 607, pp. 748–752. Springer (1992)
13. Ramachandran, J., Wahl, T.: Integrating proxy theories and numeric model lifting for floating-point arithmetic. In: Formal Methods in Computer-Aided Design, (FMCAD 2016). pp. 153–160 (2016)
14. Solovyev, A., Jacobsen, C., Rakamaric, Z., Gopalakrishnan, G.: Rigorous Estimation of Floating-Point Round-off Errors with Symbolic Taylor Expansions. In: Proceedings of FM 2015. Lecture Notes in Computer Science, vol. 9109, pp. 532–550. Springer (2015)
15. Titolo, L., Feliú, M., Moscato, M., Muñoz, C.: An Abstract Interpretation Framework for the Round-Off Error Analysis of Floating-Point Programs. In: Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2018. vol. 10747, pp. 516–537. Springer (2018)

16. Titolo, L., Muñoz, C., Feliú, M., Moscato, M.: Eliminating Unstable Tests in Floating-Point Programs. In: Proceedings of the 28th International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR 2018. vol. 10747. Springer (2018)
17. Zeljic, A., Backeman, P., Wintersteiger, C.M., Rümmer, P.: Exploring Approximations for Floating-Point Arithmetic Using UppSAT. In: 9th International Joint Conference on Automated Reasoning (IJCAR 2018). pp. 246–262 (2018)