Sub-domain Selection Strategies For Floating Point Constraint Systems *

Heytem Zitoun[1], Claude Michel[1], Michel Rueher[1], and Laurent Michel[2]

[1] Université Côte d'Azur, CNRS, I3S, France
firstname.lastname@i3s.unice.fr
[2] University of Connecticut, Storrs, CT 06269-2155
ldm@engr.uconn.edu

Abstract. Program verification is a key issue for critical applications such as aviation, aerospace, or embedded systems. Bounded model checking (BMC) and constraint programming (CBMC, CBPV, ...) approaches are based on counter-examples that violate a property of the program to verify. Searching for such counter-examples can be very long and costly when the programs to check contains floating point computations. This stems from the fact that existing search strategies have been designed for discrete domains and, to a lesser extent, continuous domains. In [12], we have introduced a set of variable choice strategies that take advantages of the specificities of the floats, e.g., domain density, cancellation and absorption phenomena. In this paper we introduce new sub-domain selection strategies targeting domains involved in absorption and using techniques derived from higher order consistencies. Preliminary experiments on a significant set of benchmarks are very promising.

1 Introduction

Programs with floating-point computations control complex and critical systems in numerous domains, including cars and other transportation systems, nuclear energy plants, or medical devices. Floating-point computations are derived from mathematical models on real numbers [8], but computations on floating-point numbers are different from computations on real numbers. For instance, with binary floating-point numbers, some real numbers cannot be represented (e.g., 0.1 does not have an exact representation). Floating point arithmetic operators are neither associative nor distributive, and may be subject to phenomena such as absorption and cancellation. Furthermore, the behavior of programs containing floating-point computations varies with the programming language, the compiler, the operating system, or the hardware architecture.

Figure 1 illustrates how the flow of a very simple program over the floats ($\mathbb{F}$) can differs from the expected flow over the reals ($\mathbb{R}$). When interpreting the program over reals, the instruction `doThenPart` should be executed. However, an absorption on the floats (the value 1 is absorbed by `1e8f`[3]) leads the program through the `else` branch.

* This work was partially supported by ANR COVERIF (ANR-15-CE25-0002).

[3] On simple precision and with rounding set to "to the nearest even".

```
void foo(){
  float a = 1e8f;
  float b = 1.0f;
  float c = -1e8f;
  float r = a + b + c;
  if(r >= 1.0f){
    doThenPart();
  } else {
    doElsePart();
  }
}
```

Fig. 1. Motivation example

In [12], we have introduced a set of *variable selection strategies* based on specific properties of floats like domain density, cancellation and absorption phenomena. The resulting search strategies are much more efficient but do not really scale for harder and more realistic benchmarks. Indeed, like in other applications of constraint techniques, efficient solvers not only requires appropriate variable selection strategies but also need relevant value selection strategies. So, this paper focuses on *value selection strategies* for floating-point constraint solvers dedicated to the search of counter-examples in program verification applications.

Standard value selection strategies over the floats are derived from *sub-domain selection strategies* used over the reals; sub-domains being generated by using various splitting techniques, eg, $x \leq v$ or $x > v$ with $v = \frac{\underline{x}+\overline{x}}{2}$.

In this paper we introduces four new sub-domain selection strategies. The first one, exploits absorption phenomena, the second one embraces ideas derived from strong consistency and the two last ones extend strategies introduced in [12]. We have evaluated these new sub-domain selection strategies on a significant set of benchmarks originate with program verification. We implemented a set of over 300 search strategies that are combinations of variable selection strategies previously introduced, sub-domain selection strategies presented in the following pages and variations of different criteria like filtering. All strategies were implemented in Objective-CP, the optimization tool introduced in [11].

In summary, the contributions are *new sub-domain selection strategies* dedicated to float system.

The rest of this article is organized as follows. Section 2 presents some notations, and definitions necessary for understanding this document. Section 3 provides a brief reminder of the strategies presented in [12]. Section 4 explains the new splitting strategies we propose. Section 5 is devoted to an analysis of the experimental results. Finally, Section 6 discusses the work in progress and the perspectives.

2 Notations and definitions

2.1 Floating point numbers

Floating point numbers approximate real numbers. The IEEE754-2008 standard for floating point numbers [9] sets floating point formats, as well as, some floating point arithmetic properties. The two most common formats defined in the IEEE754 standard are *simple* and *double* floating point number precision which, respectively, use 32 bits and 64 bits. A floating point number is a triple (s, m, e) where $s \in \{0, 1\}$ represents the sign, the mantissa m (also called significant), which is p bits long, and, e the exponent [8]. A *normalized* floating point number is defined by:

$$(-1)^s 1.m \times 2^e$$

To allow gradual underflow, IEEE754 introduces denormalized numbers whose value is given by:

$$(-1)^s 0.m \times 2^0$$

Note that simple precision are represented with 32 bits and a 23 bits mantissa ($p = 23$) while doubles use 64 bits and a 52 bits mantissa ($p = 52$).

2.2 Absorption

Absorption occurs when adding two floating point numbers with different order of magnitude. The result of such an addition is the furthest from zero. For instance, in C, using simple floating point numbers with a rounding mode set "to the nearest even", $10^8 + 1.0$ evaluates to 10^8. Thus, 1.0 is absorbed by 10^8.

2.3 Notations

In the sequel, x, y and z denote variables and $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$, their respective domains. When required, $x_{\mathbb{F}}$, $y_{\mathbb{F}}$ and $z_{\mathbb{F}}$ denote variables over $\mathbb{F}$ and $\mathbf{x}_{\mathbb{F}}$, $\mathbf{y}_{\mathbb{F}}$ and $\mathbf{z}_{\mathbb{F}}$, their respective domains while $x_{\mathbb{R}}$, $y_{\mathbb{R}}$ and $z_{\mathbb{R}}$ denote variables over $\mathbb{R}$ and $\mathbf{x}_{\mathbb{R}}$, $\mathbf{y}_{\mathbb{R}}$ and $\mathbf{z}_{\mathbb{R}}$, their respective domains. Note that $\mathbf{x}_{\mathbb{F}} = [\underline{x}_{\mathbb{F}}, \overline{x}_{\mathbb{F}}] = \{x_{\mathbb{F}} \in \mathbb{F}, \underline{x}_{\mathbb{F}} \leq x_{\mathbb{F}} \leq \overline{x}_{\mathbb{F}}\}$ with $\underline{x}_{\mathbb{F}} \in \mathbb{F}$ and $\overline{x}_{\mathbb{F}} \in \mathbb{F}$. Likewise, $\mathbf{x}_{\mathbb{R}} = [\underline{x}_{\mathbb{R}}, \overline{x}_{\mathbb{R}}] = \{x_{\mathbb{R}} \in \mathbb{R}, \underline{x}_{\mathbb{R}} \leq x_{\mathbb{R}} \leq \overline{x}_{\mathbb{R}}\}$ with $\underline{x}_{\mathbb{R}} \in \mathbb{F}$ and $\overline{x}_{\mathbb{R}} \in \mathbb{F}$. Let $x_{\mathbb{F}} \in \mathbb{F}$, then $x_{\mathbb{F}}^{+}$ is the smallest floating point number strictly superior to $x_{\mathbb{F}}$ and $x_{\mathbb{F}}^{-}$ is the biggest floating point number strictly inferior to $x_{\mathbb{F}}$. In a similar way, $x_{\mathbb{F}}^{+[N]}$ is the N^{th} floating point strictly superior to $x_{\mathbb{F}}$ and $x_{\mathbb{F}}^{-[N]}$ is the N^{th} floating point strictly inferior to $x_{\mathbb{F}}$. In addition, given a constraint c, $vars(c)$ denotes the set of floating point variables appearing in c. Finally, given a set s, $|s|$ denotes the cardinality of s.

3 Search strategies based on floating-point properties

In [12], we have introduced a set of *variable selection strategies* based on specific properties of floats like domain density, cancellation and absorption phenomena. The resulting search strategies are much more efficient but do not really scale for harder and more realistic benchmarks.

Fig. 2. Sub-domains generated by *splitAbs* ($\underline{x} > 0$ and $\underline{y} > 0$)

4 Sub-domain selection strategies

In this section we introduce four new sub-domain selection strategies. The first one takes advantage of absorption, the second is derived from strong consistency filtering techniques and tries to reduce the domain at a limited cost.The two last ones generalize sub-domain selection strategies presented in [26].

4.1 Absorption-based strategy

Let's recall that absorption occurs when adding two floating point numbers with different order of magnitude. The result of such an addition is the number the furthest from zero.

The objective of the *splitAbs* strategy is to concentrate on the most relevant absorption phenomena, in other words, giving priority to the sub-domains of x and y most likely to lead to an absorption.

Before going into the details of this absorption-based sub-domain selection strategy, let us recall the key points of $MaxAbs$, the variable strategy introduced in [12]. $MaxAbs$ is a variable selection strategy that picks the variable "absorbing the most". More precisely, this variable selection strategy needs to branch on two variables involved in absorption. The first variable –represented by `x` in Figure 2– must have the highest absorption rate. After selection of variable `x`, strategy $MaxAbs$ examines addition and subtraction constraints to select a variable `y`, the values of which are most absorbed by the values of by `x`. Coordinated branching on these variables is performed to exploit the latent absorption.

Sub-domain selection strategy : *splitAbs* Once these two variables are selected, the sub-domain selection heuristic will perform at most three splits on each variable. Figure 2 illustrates an instance where two sub-domains are generated (match with the case where domains are positive). It's easy to extend it to the others cases by symmetry. In this Figure, most interesting sub-domains are for $x : [2^{e_{\overline{x}}}, \overline{x}]$ and for $y : y \cap [0, 2^{e_{\overline{x}}-p-1}]$ (where p is the mantissa size). The first represents the sub-domain of x values absorbing y values. The second represents the sub-domain of y totally absorbed by x.

Example 1. Consider the function in Figure 3 and assume that inputs are coming from sensors, and their ranges are $[0.0, 1e{+}04]$ for `x`, and $[-16.0, 4.0]$ for `y`. The `else` branch corresponds to an unstable state of the system. Determining if this state is reachable and from which input values is a legitimate question. This problem is reduced to identifying if z can be equal to x which corresponds to absorption. Figure 4 shows resulting sub-domains. This strategy focuses on $[8.1920009765625000e{+}03, 1.0000000000000000e{+}04]$ for `x` and

```
void foo(float x, float y){
   float z = x + 2 * y;
   if(z != x)
     systemOK();
   else
     systemNOK();
}
```

Fig. 3. A program with absorptions

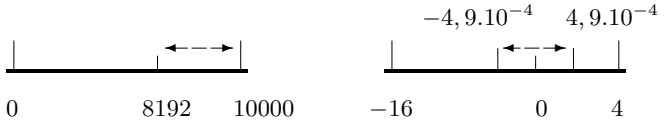

Fig. 4. Resulting sub-domains of unstable example

$[-4.8828122089616954e{-}04, 4.8828122089616954e{-}04]$ for **y** which correspond to domains involved in absorption. No solutions involving a value belonging to another sub-domain exist and the initial filtering has no impact on those domains. The unstable state is reachable with, for instance, values $1e{+}04$ for x and $2.44140625e{-}04$ for y.

This strategy can also be combined with another strategy, which is called whenever x has no values that absorb y.

4.2 Splitting strategy inspired by 3B-consistency : 3BSplit

Our next sub-domain splitting strategy, called `3Bsplit`, is inspired by a higher consistency named 3B-consistency [10]. 3B-Consistency is a relaxation on continuous domains of path consistency, a higher order extension of arc-consistency. Roughly speaking, 3B-Consistency checks whether 2B-consistency (or Hull consistency) can be enforced when the domain of a variable is reduced to the value of one of its bounds in the whole system [3]. 3B-consistency is in practice very effective on problems with multiple occurrence variables. However, insuring such consistency could be costly: the 3B-algorithm might attempt many times to unsuccessfully refute sub-domains near the bounds of the initial domain by means of a 2B-consistency.

The 3Bsplit sub-domain selection also attempts to enforce the consistency at the bounds of a domain at a single variable level. A key observation here is that if a small sub-domain at the bounds of the initial domain, e.g. $sd = [\underline{x}, \underline{x} + \delta]$, is immediately refuted in the next search node, then the resulting domain, $[\underline{x}+\delta, \overline{x}]$, offers a better lower bound than the one of the initial domain. Moreover, there is probably room to still improve this bound if the same step is reiterated using a wider sub-domain like $[\underline{x}, \underline{x} + 2\delta]$. Indeed, such a process is similar to a *shaving* applied to the initial domain but without requiring additional domain state management: in the case of a `3Bsplit`, the capability to return to the initial state is naturally supported by the search.

A sub-domain split might not be refuted immediately in the next search node. It might be refuted either after exploring a deeper search sub-tree or it

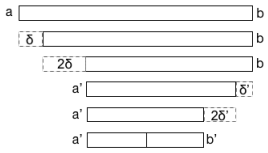

Fig. 5. `Illustration of 3BSplit`

might provide one or more solutions. Both cases underline some difficulties to improve the bound under examination. The next step of the splitting strategy thus switch to the next bound or, if both bounds have been checked, to another search node or strategy. Figure 5 illustrates `3Bsplit` behavior.

To summarize, `3Bsplit` exploits information on the sub-tree to decide whether enforcing the current bound has a chance to be done effortlessly or if it would be wiser to go to the next step. As a result, this strategy dynamically splits the current domain according to the behavior of the search in sub-trees. Note also that choosing an initial small sub-domain at the bounds of the domain is similar to the next strategy (Section 4.3), i.e., `3Bsplit` also provides opportunities to find solutions in the neighbourhood of the current bounds.

4.3 Mixing sub-domain and enumeration

This sub-domain selection strategy extends strategies from [4]. We propose two ways to extend these strategies. The first one `Enum-N`, enumerates `N` values of both bound and one in the middle before considering the rest of the domain. The second one, `Delta-N`, instead of enumerating each `N` values of the bounds, builds the sub-domains implied by `N` floating point numbers. Due to the huge number of evaluated strategies, the value of N is arbitrarily limited to 5 in the experiment part.

Enum-N
This sub-domain selection strategy is a direct generalization of [4]. Generally, the filtering process tightens the bounds until a support is found. This sub-domain selection strategy is optimistic and hope that filtering will lead to find a solution close to the bounds. To achieve this goal, it enumerates few values at the bounds. Figure 6a illustrates this strategy. The domains is split in $2 * N + 3$ sub domains. For instance, with $N = 5$, the 13 following domains will be generated :

- $[min, min]$
- $\cdots$
- $[min^{+[N]}, min^{+[N]}]$
- $[min^{+[N+1]}, mid^{-}]$
- $[mid, mid]$
- $[mid^{+}, max^{-[N+1]}]$
- $[max^{-[N]}, max^{-[N]}]$
- $\cdots$
- $[max, max]$

If the cardinality of the domain is lower than $2 * N + 3$, all values will be enumerated.

Delta-N The first objective of this strategy is to find a solution at the bounds. Enumerating can lead the search to explore a deep sub-tree before finding a

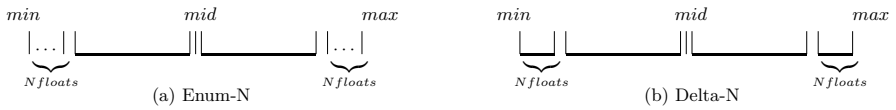

Fig. 6. Illustration of new strategies

solution or performing reduction. Here, considering the sub-domain implied by N floating-point numbers instead of a single value, gives opportunities to the filtering process to operate some pruning through propagation. It also improves the chance to find a solution or to remove all N values without enumerating any of them. This sub-domain selection strategy is very flexible. Indeed, by adapting the value of N, the whole strategy behavior change. For instance, if $N = |\alpha|$, with $\alpha = [\underline{x}, \frac{\underline{x}+\overline{x}}{2}]$, this sub-domain strategy becomes a classic bisection. Finally, a dynamic modification of the value of N during the search can be interesting. Starting the search by a classic bisection ($N = |\alpha|$) and reducing its value might be a good idea. Figure 6b illustrates this strategy. Regardless the value of N and the cardinality of the domain, at most the 5 following sub-domains will be generated :

- $[min, min^{+[N]}]$
- $[min^{+[N+1]}, mid^{-}]$
- $[mid, mid]$
- $[mid^{+}, max^{-[N+1]}]$
- $[max^{-[N]}, max]$

If the cardinality of the domain is lower than $2 * N + 3$, the last two sub-domains will not be generated, and the first two will be balanced with respect to the middle of the domain.

5 Experimental Evaluation

The experiments combine different variable selection strategies with sub-domain selection strategies on a set of 49 benchmarks. They also consider variations on the type of consistency for the strategies, sub-cuts, and the reselection (or not) of the same variable at the next node. Sub-cut corresponds to an alternative sub-domain selection strategy and is relevant for *splitAbs* and $3Bsplit$. For *splitAbs*, sub-cut is called when no absorptions occur. For $3Bsplit$, it is called to refute small sub-domains. In the state of the art, the standard strategy is based on lexicographic variable ordering, and a bisection based on 2B-consistency. These options result in no less than 325 unique strategies evaluated on all 49 benchmarks.

All experiments were performed on a Linux system, with an Intel Xeon processor running at 2.40GHz and with 12GB of memory. All strategies have been implemented into the Objective-CP solver. All the floating point computations are performed in simple precision and with a rounding mode “to the nearest even”.

5.1 Benchmarks

The benchmarks used in these experiments come from test and program verification. SMTLib [1], FPBench [6], and CBMC [2] (but also [5, 4, 7]) are the main

sources. In each case, the goal is to find a counter-example, hence the majority of instances are *satisfiable*. The number of constraints and variables varies from 2 to about 3000. Table summarizes thoses results of all the strategies on the satisfiable instances can be found at `www.i3s.unice.fr/~hzitoun/dp18/benchmark.html`.

5.2 Analysis

In results Tables, the standard strategy (lexicographic order with bisection, 2B filtering at 5% and reselection allowed) is at the 194^{th} position on 325 strategies. So, 193 strategies among those introduced, are clearly more efficient than the standard strategy for solving this kind of problem. The `Virtual Best Strategy` is 120 times faster than the reference strategy. The best strategy (column 1) is 4 times faster than the reference strategy. Using the specificities of floats to guide the search has a clear impact on the resolution time. Among the strategies that are efficient on this set of benchmarks, the variable selection strategies are based on lexicographic order, number of occurrences, density or absorption. While strategies based on width, a conventional variable selection strategy for integer domains, struggle to solve problems as soon as it becomes a bit realistic. Strategies based on cardinality are also in the same cases. The best search based on `MaxCard` and `MaxWidth` are at 160^{th} and 161^{th} positions.

Sub-domain selection strategies introduced in this article are working well. Indeed, the faster strategy exploiting `Delta-N` is in second position, whereas `Enum-N` is at 6^{th} position. The best $3BSplit$ is placed at 11^{th}. *SplitAbs*, for its part, is at 112^{th} position. *SplitAbs* performance are clearly related to the percentage of absorption of the problem. Indeed, as shown in the online tables, the set of benchmarks limited to those with at least 5% of absorption are resolved without timeout. All these strategies perform better than the standard one.

Eight of the 10 best strategies prohibit the repeated selection of a variable at subsequent search nodes. Among the 10 worst, only one of them prohibits reselection. It appears that "reselection" impacts the ability to deliver solutions faster.

6 Conclusion

A previous article proposed a set of variable selection strategies using the specificities of floats to guide the search. These variable selection strategies improve the search of a counter-example outlining a property violation in a program to verify, but aren't sufficient to scale for harder and more realistic benchmarks. Dedicated sub-domain selection strategies for floats are needed. Contributions of this article are a set of sub-domain selection strategies over floats. The first one, exploits absorption phenomena, the second one embraces ideas derived from strong consistency and the two last ones extend strategies introduced in [4]. These strategies are compared on a set of satisfiable benchmarks. Several strategies presented, perform well, and obtain much better results than the standard strategy used to solve this kind of problem.

References

1. Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.
2. Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ansi-c programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176, 2004.
3. Hélène Collavizza, François Delobel, and Michel Rueher. Comparing partial consistencies. *Reliable Computing*, pages 213–228, 1999.
4. Hélène Collavizza, Claude Michel, and Michel Rueher. Searching critical values for floating-point programs. In *Testing Software and Systems - 28th IFIP WG 6.1 International Conference, ICTSS 2016, Graz, Austria, October 17-19, 2016, Proceedings*, pages 209–217, 2016.
5. Hélène Collavizza, Michel Rueher, and Pascal Van Hentenryck. Cpbpv: A constraint-programming framework for bounded program verification. *Constraints*, 15(2):238–264, 2010.
6. Nasrine Damouche, Matthieu Martel, Pavel Panchekha, Chen Qiu, Alexander Sanchez-Stern, and Zachary Tatlock. Toward a standard benchmark format and suite for floating-point analysis. In Sergiy Bogomolov, Matthieu Martel, and Pavithra Prabhakar, editors, *Numerical Software Verification*, pages 63–77, Cham, 2017. Springer International Publishing.
7. Vijay D'Silva, Leopold Haller, Daniel Kroening, and Michael Tautschnig. Numeric bounds analysis with conflict-driven learning. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 48–63, 2012.
8. David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, 1991.
9. IEEE. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard*, 754, 2008.
10. Olivier Lhomme. Consistency techniques for numeric csps. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, pages 232–238, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.
11. Pascal Van Hentenryck and Laurent Michel. The objective-cp optimization system. In *Principles and Practice of Constraint Programming: 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 8–29, Berlin, Heidelberg, 2013.
12. Heytem Zitoun, Claude Michel, Michel Rueher, and Laurent Michel. Search strategies for floating point constraint systems. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, pages 707–722, 2017.