Verifying Floating-Point Commutativity with GRS

Robin Trüby
University Freiburg
Freiburg, Germany
robin.trueby@web.de

Mathias Fleury
University Freiburg
Freiburg, Germany
fleury@cs.uni-freiburg.de

Armin Biere
University Freiburg
Freiburg, Germany
biere@cs.uni-freiburg.de

I. GRS Description

GRS are a technique to operate on floating operations efficiently following the IEEE 754 norm [1]. It is used inside CPUs for efficiency. It is possible to prove that only 3 additional digits, called guard (G), round (R), and sticky (S). This is sufficient to keep the results correct. This is more efficient for double than converting the numbers to 80-bit floating points and then rounding back. In his Bachelor project, the first author created a webpage [2] in German to illustrate the use of GRS bits.

GRS must be paired with a rounding technique for ties. We use round to even, which is better to keep precision than always rounding down or always rounding up.

The benchmark we are interested correspond to proving that a+b=b+a. We know that such encoding for integer addition is very easy to solve for SAT solvers, while becoming hard for multipliers. To simplify the encoding, we first generated word-level SMT benchmarks. We have submitted some of the (smaller) benchmarks to the SMT Competition directly.

To validate the correctness of our encoding [3], we compared the results of our encoding to the floating point theory of SMT solvers and found no difference.

The idea of addition is that our two numbers a and b are binary encoding from:

$$\begin{aligned} a =& (-1)^{s_a}(1+m_a)2^{n_a} \\ b =& (-1)^{s_b}(1+m_b)2^{n_b} \end{aligned}$$

Then we align the exponents

$$\begin{aligned} a+b = &\big((-1)^{s_a}(1+m_a)2^{n_a-\max(n_a,n_b)}+ \\ = &\quad (-1)^{s_b}(1+m_b)2^{n_b-\max(n_a,n_b)}\big)2^{\max(n_a,n_b)} \end{aligned}$$

Finally, we have to implement the operations. For this, we used a word-level representation in our SMT encoding. Finally, we have handle various special cases, including realigning numbers, changing the exponents, handling denormal numbers, and handling overflows (like NaN or infinity). Finally we can produce the final binary result.

Due to the encoding, we know that all problems are UNSAT. Also we have not found a simple way to generate satisfiable hard benchmarks with GRS.

II. Benchmark Generation

To generate the benchmarks we have written a custom C program (available online [4]) that writes the SMT file. Then we use Bitwuzla [5] to convert the SMT files to DIMACS files without any specific options.

During experimentation, we found out that encoding as (if a then x=s else x=t) makes the problem very easy for SMT solver and we generate benchmarks using x = (if a then s else t).

The naming convention for our benchmarks is grs-<mantissa-size>-<exponent-size>.cnf.

References

[1] IEEE Committee, "IEEE standard for floating-point arithmetic," IEEE Computer Society, New York, NY, USA, Standard IEEE Std 754-2008, Aug. 2008. [Online]. Available: https://web.archive.org/web/20160806053349/http://www.csee.umbc.edu/~tsimo1/CMSC455/IEEE-754-2008.pdf

[2] R. Trüby. (2023) Floating-point arithmetic. Bachelor Project at University Freiburg, in German. [Online]. Available: https://cca.informatik.uni-freiburg.de/teaching/grs-bits/home.html

[3] ——, "Generating word-level floating-point benchmarks," 2023, bachelor Thesis at University Freiburg, Submitted.

[4] ——. (2023) Bachelorarbeit. Accessed April 2023. [Online]. Available: https://github.com/Robin060500/Bachelorarbeit

[5] A. Niemetz and M. Preiner, "Bitwuzla at the SMT-COMP 2020," CoRR, vol. abs/2006.01621, 2020. [Online]. Available: https://arxiv.org/abs/2006.01621