Polynomial Formal Verification of Floating Point Adders

Jan Kleinekathöfer
University of Bremen, Germany
ja_kl@uni-bremen.de

Alireza Mahzoon
University of Bremen, Germany
mahzoon@uni-bremen.de

Rolf Drechsler
University of Bremen/DFKI, Germany
drechsler@uni-bremen.de

***Abstract*—In this paper, we present our verifier that takes advantage of *Binary Decision Diagrams* (BDDs) with case splitting to fully verify a floating point adder. We demonstrate that the traditional symbolic simulation using BDDs has an exponential time complexity and fails for large floating point adders. However, polynomial bounds can be ensured if our case splitting technique is applied in the specific points of the circuit. The efficiency of our verifier is demonstrated by experiments on an extensive set of floating point adders with different exponent and significand sizes.**

***Index Terms*—floating point, verification, symbolic simulation**

I. Introduction

In the last 30 years, several formal methods have been proposed to verify integer arithmetic circuits. Furthermore, polynomial bounds have been proven for some of these techniques [1]–[6]. However, the available approaches totally fail if they are applied to floating point arithmetic: (1) BDD-based verification runs out of memory since the size of graphs grows exponentially with respect to the size of the circuit, (2) *Boolean Satisfiability* (SAT) based verification runs for an indefinite amount of time, and (3) word-level verification techniques based on *Symbolic Computer Algebra* (SCA), e.g., [7] and *Word-Level Decision Diagrams* (WLDDs) , e.g., [8] fail as there is no clear word-level specification. There have been some efforts to extend the existing verification techniques for the floating point arithmetic [9]–[13]. Many of them make use of case splitting to simplify the verification process, but fail to give bounds on the complexity.

In this paper, we propose **a verification method based on symbolic simulation with case splitting to ensure the correctness of big floating point adders**. In symbolic simulation [14], the function represented by the circuit is iteratively built from the inputs towards the outputs using BDDs. The represented function is compared to the expected function by checking the equality of BDDs, which can be done in constant time by BDD packages. This verification technique is successfully used to prove the correctness of integer adders in polynomial time. However, it fails when it comes to the verification of floating point adders since the size of intermediate BDDs grows exponentially with respect to the input sizes. We first find the explosion points in a floating point adder by experimental evaluation, i.e. we detect the points in the circuit where the size of intermediate BBDs grows, dramatically. Subsequently, we come up with a case splitting technique to avoid the explosion. We detect the BDDs in some specific points of the circuit and simplify them by considering several cases. **We keep the size of intermediate BDDs small during the symbolic simulation and make polynomial formal verification possible**. The experimental results confirm that our method verifies floating adders with different sizes, including standard IEEE754 format (i.e. single precision, double precision, and quadruple precision) as well as arbitrary sizes.

This work has been supported by DFG within the project DR 287/36-1.

TABLE I
BDD SIZES OF THE INTERMEDIATE RESULTS FOR DIFFERENT FORMATS

exponent/significand	5/5	5/10	8/15	8/20
exp. comp.	15	15	24	24
exp. diff.	26 (1.7×)	26 (1.7×)	38 (1.6×)	38 (1.6×)
align. shift	115 (4.4×)	153 (5.9×)	593 (15.6×)	701 (18.4×)
sig. add.	367 **(3.2×)**	1677 **(11.0×)**	9729 **(16.4×)**	50471 **(72.0×)**
lead. zero count.	859 (2.3×)	6164 (3.7×)	36399 (3.7×)	211224 (4.2×)
norm. shift	2306 **(2.7×)**	52237 **(8.5×)**	756297 **(20.8×)**	10024731 **(47.5×)**
output	2748 (1.2×)	61823 (1.2×)	841735 (1.1×)	10618756 (1.0×)

II. Challenges

In this section, we investigate the challenges of verifying a floating point adder using BDDs. TABLE I reports the intermediate BDD sizes during the symbolic simulation of the floating point adder. There are two formats with an exponent size of 5 (see the second and third column) and two with an exponent size of 8 (see the fourth and fifth column). For the formats with the exponent size 5(8), the significand has 5(15) or 10(20) bits. Each cell contains two numbers: the first number is the size of the intermediate BDD in the specific point of the circuit and the second number is the growth of the BDD size compared to the last point, i.e. the above cell. It is evident from the table that at two points of the circuit an explosion occurs in the size of intermediate BDDs:

1) The intermediate BDD size grows significantly on the output of the significand addition compared to the previous point, i.e. the output of the alignment shift. This growth is exponential with respect to the significand and exponent sizes, e.g. it reaches a growth of $72.0\times$ for the exponent/significand size of $8/20$. [15] provides a theoretical explanation for this explosion.
2) The explosion happens again on the output of the normalization shift compared to the previous point, i.e. the output of the leading zero counter. Similar to the first blow-up, the growth is exponential with respect to the significand and exponent sizes.

Therefore, the two main points of BDD explosion can be determined as the significand addition and the normalization shift.

III. Case Splitting

We now explain our approach, which takes advantage of symbolic simulation with case splitting to avoid the explosion: first, a set of cases is specified on the intermediate results. This set has to have a polynomial size with respect to the input size to be efficient. Furthermore, every possible value of an intermediate result has to be included in at least one of the cases to keep the verification complete. For every bit of the intermediate result, there is one simplification in the case. The simplification is done by replacing the original BDD with the simplified BDD. The easiest way to simplify an intermediate BDD is to have one case where the BDD is simplified to a terminal one and another case where it is simplified to a terminal zero. Unfortunately, it cannot be done for all bits of an n-bit intermediate result since this would result in 2^n cases. Hence, when creating the set of cases the input space has to be divided smartly.

A. Alignment Shift Case Splitting

To avoid the BDD explosion at the significand addition, a case splitting is introduced. One easy way to define such a case splitting is to split with regard to the shift amount or the exponent difference. While there are exponentially many possible values for the exponent difference with respect to the exponent size, only linear many cases with respect to the significand size are actually needed for the shift. This is due to the fact that it is irrelevant how big the actual shift was if all non-zero values were shifted out. Additionally, by simplifying the exponent comparison result (which multiplies the number of cases by two) the shift direction is determined. The total number of cases is $(n+2) \times 2$ if we assume that the significand size is widened by two to ensure a correct rounding. The maximum BDD size evaluates to $3n-1$ nodes with the simplifications.

B. Leading Zero Case Splitting

To handle the explosion at the normalization shift, the simplification is done on the significand addition result. Creating a case for every possible result of the addition is not possible due to the exponential number of possible results. Instead, one case is created for every possible number of leading zeros. This does not specify a simplification for every bit of the addition result in every case. In total, there are $n+3$ cases from 0 leading zeros to $n+2$ leading zeros, considering two bits extra width for rounding. The n-th case contains $n+1$ simplifications. The biggest BDD with a size of $3n-1$ nodes occurs when 0 leading zeros are present.

C. Subnormal Numbers and Rounding

We also extended our approach for the support of subnormal numbers as well as round-to-nearest rounding. Subnormal numbers are handled by adding a simplification in cases where they can occur. Rounding does not trigger an explosion in BDD size.

IV. Experimental Results

We have implemented the proposed approach in C++. The experiments have been carried out on an AMD Ryzen 7 PRO 5850U with 4.50 GHz boost frequency and 32 GByte of main memory. TABLE II reports the results of verifying floating point adders and consists of three columns: The benchmarks information is given in the first column **Adder Parameters**, consisting of three sub-columns: *Format* shows the name of the used format if possible. The exponent and significand sizes are reported in *Exp.* and *Sig.*, respectively. The second column **Our Method** reports the verification results of our proposed method, including the numbers of output nodes (*Nodes*) and execution times in seconds (*Time*). The results of verification using a classic symbolic simulation without case splitting is given in the third column **Classic Sym. Sim.**. The timeout (T.O.) has been set to two hours for all benchmarks.

TABLE II
Verification results (run-times in seconds)

Adder Parameters			Our Method		Classic Sym. Sim.	
Format	Exp.	Sig.	Nodes	Time	Nodes	Time
	8	20	1,087	1.8	10,618,756	430.4
single prec.	8	24	1,583	2.1	-	T.O.
	10	38	4,146	5.6	-	T.O.
double prec.	11	53	8,190	10.2	-	T.O.
	13	83	20,398	38.3	-	T.O.
quad. prec.	15	113	38,116	82.9	-	T.O.
	18	170	86,808	325.1	-	T.O.

The benchmarks including the three IEEE754 adders can be verified in short times, and the sizes of the output BDDs are much smaller than for the classic symbolic simulation even for smaller formats. For example, a floating point adder with quadruple precision can be verified in less than 2 minutes. Thus, the experimental results confirm the efficiency of our proposed method in verifying large floating point adders. On the other hand, the classic symbolic simulation without case splitting only works for the smallest benchmark, and it runs out of time for the bigger adders.

References

[1] M. Keim, R. Drechsler, B. Becker, M. Martin, and P. Molitor, "Polynomial formal verification of multipliers," *Formal Methods in System Design: An International Journal*, vol. 22, no. 1, pp. 39–58, 2003.

[2] R. Drechsler, "PolyAdd: Polynomial formal verification of adder circuits," in *DDECS*, 2021, pp. 99–104.

[3] A. Mahzoon and R. Drechsler, "Polynomial formal verification of prefix adders," in *ATS*, 2021, pp. 85–90.

[4] ——, "Late breaking results: Polynomial formal verification of fast adders," in *DAC*, 2021, pp. 1376–1377.

[5] R. Drechsler, A. Mahzoon, and M. Goli, "Towards polynomial formal verification of complex arithmetic circuits," in *DDECS*, 2022, pp. 1–6.

[6] R. Drechsler and A. Mahzoon, "Polynomial formal verification: Ensuring correctness under resource constraints."

[7] R. Drechsler and D. Sieling, "Binary decision diagrams in theory and practice," *STTT*, vol. 3, pp. 112–136, 2001.

[8] R. E. Bryant and Y.-A. Chen, "Verification of arithmetic functions with binary moment diagrams," in *DAC*, 1995, pp. 535–541.

[9] R. Kaivola, R. Ghughal, N. Narasimhan, A. Telfer, J. Whittemore, S. Pandav, A. Slobodová, C. Taylor, V. Frolov, E. Reeber, and A. Naik, "Replacing testing with formal verification in Intel® Core™ i7 processor execution engine validation," in *CAV*, 2009, pp. 414–429.

[10] W. Hunt, S. Swords, J. Davis, and A. Slobodova, "Use of formal verification at centaur technology," *Design and Verification of Microprocessor Systems for High-Assurance Applications*, 01 2010.

[11] M. D. Aagaard, R. B. Jones, and C.-J. H. Serger, "Formal verification using parametric representations of boolean constraints," in *DAC*, 1999, p. 402–407.

[12] C. Jacobi, K. Weber, V. Paruthi, and J. Baumgartner, "Automatic formal verification of fused-multiply-add fpus," in *DATE*, 2005, pp. 1298–1303.

[13] Y.-A. Chen and R. E. Bryant, "*PHDD: an efficient graph representation for floating point circuit verification," in *ICCAD*, 1997, pp. 2–7.

[14] R. E. Bryant, "A methodology for hardware verification based on logic simulation," *Journal of the ACM*, vol. 38, no. 2, pp. 299–328, 1991.

[15] J. Kleinekathöfer, A. Mahzoon, and R. Drechsler, "Lower bound proof for the size of bdds representing a shifted addition," 2022, coRR abs/2209.12477.