A Parameterized Floating-Point Formalizaton in HOL Light

Charles Jacobsen[a,1,2] Alexey Solovyev[a,3,5]
Ganesh Gopalakrishnan[a,4,5]

[a] *School of Computing*
University of Utah
Salt Lake City, USA

Abstract

We present a new, open-source formalization of fixed and floating-point numbers for arbitrary radix and precision that is now part of the HOL Light distribution [10]. We prove correctness and error bounds for the four different rounding modes, and formalize a subset of the IEEE 754 [1] standard by gluing together a set of fixed-point and floating-point numbers to represent the subnormals and normals. In our floating-point proofs, we treat phases of floating-point numbers as copies of fixed-point numbers of varying precision so that we can reuse fixed-point rounding theorems.

Keywords: IEEE-754-2008, floating point, fixed point, formalization

1 Introduction

Programmers need tools that calculate error bounds and automate the tedious and error-prone reasoning involved in proofs of correctness for their floating-point algorithms. They follow the IEEE 754 standard because most hardware and software libraries support it and provide fast implementations with multiple levels of precision. As examples of adoption of this standard, four levels of precision are available on Intel CPUs [13] and three on Nvidia GPUs [2], and decimal floating-point numbers are in a package developed by Cornea, et al. [7].

But it is hard to reason about floating-point algorithms because floating-point numbers and operations are an approximate model of the real numbers and do not share desirable properties like associativity that their real-valued counterparts have. Programmers may miss edge cases that produce large errors, miss ways to improve

[1] Supported in part by NSF Research Experience for Undergraduates (REU), Award CCF 1430497.
[2] Email: charlesj@cs.utah.edu
[3] Email: monad@cs.utah.edu
[4] Email: ganesh@cs.utah.edu
[5] Support in part by NSF Awards CCF 1421726 and 7298529.

accuracy by re-ordering operations, or use a precision that is too conservative. For example, Goualard [9] shows that Intlab V5.5 [14] reports μ as the midpoint of the interval $[-\mu, \mu]$ instead of 0, where μ is the smallest subnormal number. While the designers of Intlab V5.5 may have been aware of this edge case, it shows how subtle the errors in floating-point algorithms can be.

We present a new formalization of fixed- and floating-point numbers that can be used to reason about programs that use IEEE floating-point numbers. [6] Our work is motivated by John Harrison's formalization in HOL Light for binary floating-point [11]. To our knowledge, this formalization is not publicly available and doesn't use a fixed- point theory as a basis for floating-point theory, as ours does. Daumas, Rideau, and Théry [8] and Boldo and Melquiond [4] have developed advanced formalizations of fixed and floating point for arbitrary radix and precision in Coq. We chose not to use the Coq formalizations nor translate them to HOL Light since these formalizations are large and complicated and our immediate need was to formalize only a subset of fixed- and floating-point theory. HOL Light also provides the non-constructive Hilbert choice operator, which makes the formalization easier to write since we aren't trying to compute floating-point numbers in our formalization. Other formalizations for binary floating point have been done in Z, PVS, HOL4, Isabelle/HOL, and ACL2 [3,6,5,12,16,15]. To our knowledge, the Z formalization is not publicly available. Second, the theorems in the Isabelle/HOL formalization are for single precision only, and would not generalize easily. Finally, our formalization does not rely on higher level operations and corresponding theories, like real-valued `floor` as in ACL2.

2 Formalization Overview

We model IEEE floating-point numbers as a subset of $\mathbb{R}$. Subnormal numbers and zero are represented using a set of real numbers that are a fixed distance apart (fixed-point numbers). Normal numbers are represented using a subset of an infinite set of real numbers that are varying distances apart (floating-point numbers). Finally, we take any real number whose magnitude is above a carefully chosen threshold to be floating-point infinity. We do not model signed zero or NaNs. Each set is explained in more detail in the following sections.

We have defined rounding for fixed- and floating-point numbers in the four rounding modes (round to nearest with ties to even, round to zero, round to positive infinity, and round to negative infinity), and proved fundamental properties of rounding, like the $(1 + \delta)$ error rule.

2.1 Subnormal Numbers and Zero

In the IEEE standard, the subnormal numbers and zero are separated by a fixed distance of r^{e-p+1} where r is the radix, e is the minimum exponent, and p is the precision, determined by the format. For example, for binary single precision, r is 2, e is -126, and p is 24, and the fixed distance is 2^{-149}. We can represent

[6] Our formalization is available at https://code.google.com/p/hol-light/source/browse/#svn/trunk/IEEE

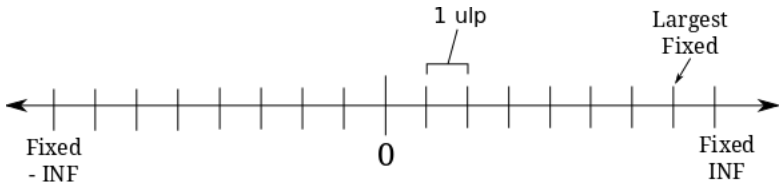

Fig. 1. Model of the subnormal numbers and zero as a subset of $\mathbb{R}$ of fixed-point numbers.

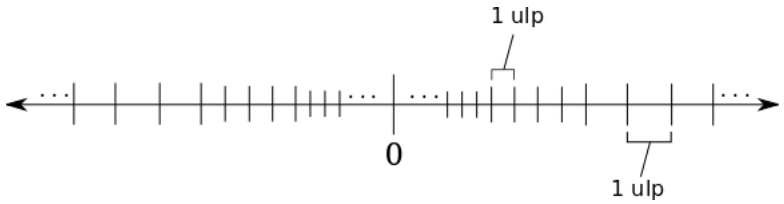

Fig. 2. An infinite set of floating-point numbers in $\mathbb{R}$. There is no upper or (tight) lower bound on their magnitude. Zero is not included.

them using a set of real numbers whose magnitude is of the form $f \cdot r^{e-p+1}$ where $f \in \mathbb{N}, 0 \leq f < r^{p-1}$.

For our definition of rounding to work smoothly when we combine the fixed- and floating-point numbers together, we let those real numbers with $f = r^{p-1}$ be a special 'fixed-point infinity' that are 1 ulp away from the largest subnormal numbers. Note that its magnitude is $r^{p-1} \cdot r^{e-p+1} = r^e$, the magnitude of the smallest IEEE normal number for the format.

Definition 2.1 $Fixed(r, p, e) = \{\, x \in \mathbb{R} \,:\, |x| = f \cdot r^{e-p+1}, f \in \mathbb{N}, 0 \leq f \leq r^{p-1} \,\}$.

The unit in the last place (*ulp*) is then taken to be the distance between the fixed-point numbers. See Figure 1.

Definition 2.2 $ulp(r, p, e) = r^{e-p+1}$.

Our formalization of rounding for fixed point is as follows: To round a real number x to $x^* \in Fixed(r, p, e)$, we calculate the closest $x^-, x^+ \in Fixed(r, p, e)$ such that $x^- \leq x \leq x^+$ and choose either of x^- or x^+ depending on the rounding mode. Note that if $|x|$ is bigger than the largest fixed-point number, it may be rounded to fixed-point infinity, depending on the rounding mode; and this is correct since fixed-point infinity can be identified with the smallest IEEE normal number. We have proved that our formalization of rounding is correct for the four rounding modes (e.g., that rounding to nearest with ties to even does in fact round to even), and that rounding to nearest for fixed point satisfies a key property:

Theorem 2.3 $|x| \leq Fixed\ INF \Rightarrow roundfixed(x, near) = x(1 + \delta)$,
where $0 \leq |\delta| \leq 1$.

Note that this theorem and all others for fixed point are qualified with the condition that $|x|$ is no bigger than fixed-point infinity (otherwise, x^- and x^+ may not be defined).

2.2 Normal Numbers

Normal numbers in the IEEE standard can be modeled as floating-point numbers of the form $f \cdot r^{e-p+1}$ where r is the radix and p is the precision, and e and f are integers such that $emin \leq e \leq emax$ and $r^{p-1} \leq f < r^p$. The parameters r, p, $emin$, and $emax$ depend on the format. For example, for binary single precision, $emin$ is -126 and $emax$ is 127.

Following Harrison [11] and Boldo and Melquiond [4], to make floating-point rounding easier to formalize, we first construct an infinite set of floating point numbers where the integer exponent e is unbounded (above *and* below) and the significand f is less restricted with $0 < f < r^p$. Note that $f > 0$, so zero is not included in this set of floating-point numbers.

Definition 2.4 $Float(r,p) = \{\ x \in \mathbb{R}\ :\ |x| = f \cdot r^{e-p+1}, f \in \mathbb{N}, e \in \mathbb{Z}, 0 < f < r^p\ \}$.

For any real number x, let $x^-, x^+ \in Float(r,p)$ be the closest floating-point numbers with $x^- \leq x < x^+$. Because the distance between consecutive floating-point numbers varies, we define the unit in the last place as $x^+ - x^-$, a function of x. See Figure 2. Because $1 = r^{p-1} \cdot r^{0-p+1}$, 1 is a floating-point number for any format; so we can also define the *machine epsilon* as half the distance between 1 and the next largest floating-point number, or $r^{-(p-1)}/2$.

Definition 2.5 $float\_eps(r,p) = r^{-(p-1)}/2$.

We call groups of floating-point numbers with the same sign and same exponent e a floating-point 'phase' (other literature call these binades for binary floating point). Notice that floating-point numbers in a phase are the same distance apart, and that each of them can be expressed as $r^e + y$ where $y \in Fixed(r,p,e)$. We can also determine which phase a real number x is in by finding the largest e such that $r^e \leq |x|$.

Our formalization of rounding for floating-point numbers is designed to re-use our formalization and proofs for fixed-point numbers. For any non-zero real number x, we compute the largest e such that $r^e \leq |x|$. We then compute $y \in \mathbb{R}$ such that $y = x - r^e$ if x is positive and $y = x + r^e$ if x is negative. Finally, we compute the rounding result as $r^e + roundfixed(y, mode)$, where $roundfixed(y, mode)$ is fixed-point rounding of y.

We have proved that our formalization of floating-point rounding satisfies key properties, and that it satisfies the usual model for rounding:

Theorem 2.6 $x \neq 0 \Rightarrow roundfloat(x, near) = x(1+\delta)$, *where* $0 \leq |\delta| \leq float\_eps$.

We now model the normal numbers in IEEE by placing a bound $emin \leq e \leq emax$ on the exponent in $Float(r,p)$.

Definition 2.7 $Normal(r,p,emin,emax) = \{\ x \in \mathbb{R}\ :\ |x| = f \cdot r^{e-p+1}, f \in \mathbb{N}, e \in \mathbb{Z}, 0 < f < r^p, emin \leq e \leq emax\ \}$.

Note that the largest magnitude of a normal number is $(r^p - 1) \cdot r^{emax-p+1}$. For binary single precision, this is $(2^{24} - 1) \cdot 2^{127-24+1}$ or $2^{127}(2 - 2^{1-24})$.

Definition 2.8 $maxnormal(r,p,emin,emax) = (r^p - 1) \cdot r^{emax-p+1}$.

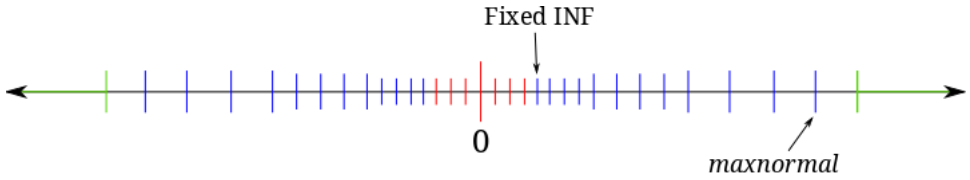

Fig. 3. The set of IEEE numbers for $r = 2$, $p = 3$, $emin = -1$, and $emax = 1$. The red ticks are fixed-point numbers, blue ticks are normal numbers, and the green regions are infinite IEEE numbers. fixed-point infinity should be colored both red and blue.

2.3 Floating-Point Infinities

Our goal is for floating-point rounding to work smoothly when we piece everything together in the larger IEEE formalization. We also want to use HOL Light's theories for real arithmetic. We accomplish this as follows: We define a predicate $is\_inf(x)$ on $\mathbb{R}$ where $is\_inf(x)$ is true when x's magnitude is at least $maxnormal + u$, where $maxnormal \in Normal(r, p, emin, emax)$ is the largest normal number and $u = r^{emax-p+1}$ is the unit in the last place for $maxnormal$. Intuitively, $is\_inf(x)$ is true when x is outside what is representable as a finite IEEE number *and* cannot be rounded to one either.

Definition 2.9
$is\_inf(r, p, emin, emax, x) = |x| \geq maxnormal(r, p, emin, emax) + r^{emax-p+1}$.

Just as fixed-point rounding worked correctly when values were rounded to a fixed-point infinity, floating-point rounding should work correctly for real values that round toward infinity (the current formalization does not contain this proof, however).

2.4 IEEE: Putting It All Together

Given r, p, $emin$, and $emax$ in an IEEE format, we say $x \in \mathbb{R}$ is an IEEE number if at least one of the following is true:

- $x \in Fixed(r, p, emin)$
- $x \in Normal(r, p, emin, emax)$
- $is\_inf(r, p, emin, emax, x)$

See Figure 3. Our formalization restricts $r > 1$ and even, $p > 1$, and $emax \geq emin$, but doesn't have some of the other IEEE restrictions (e.g., that $emin = -emax+1$).

We formalize rounding for IEEE numbers as follows: For any real number x, if $is\_inf(x)$ is true, the rounding result is just x. Otherwise, we use *roundfixed* or *roundfloat*, depending on the range that x's magnitude falls into. We have not proved correctness for IEEE rounding, but we have 'lifted' the key error properties of rounding from the fixed and floating point counterparts.

3 Concluding Remarks

In this work, we presented a new formalization of fixed and floating point that is part of the current HOL Light distribution. We believe that this public release of a HOL Light theory for floating point will enable the HOL Light community to

work in their familiar framework, without having to resort to cross theorem-prover analysis which is currently not very well supported or developed. We also plan to investigate round-off errors in floating-point computations to obtain tight upper bounds on round-off error in a separate project. We plan to employ our work in that project for checking proof certificates of the reported error.

References

[1] IEEE standard for floating point arithmetic. *IEEE Std. 754-2008*, 2008.

[2] CUDA C Programming guide. *NVIDIA Corporation, July*, 2012.

[3] Geoff Barrett. Formal methods applied to a floating-point number system. *Software Engineering, IEEE Transactions on*, 15(5):611–621, 1989.

[4] Sylvie Boldo and Guillaume Melquiond. Flocq: A unified library for proving floating-point algorithms in Coq. In *Computer Arithmetic (ARITH), 2011 20th IEEE Symposium on*, pages 243–252. IEEE, 2011.

[5] Sylvie Boldo and César Muñoz. A high-level formalization of floating-point numbers in PVS. Contractor Report NASA/CR-2006-214298, NASA, Langley Research Center, Hampton VA 23681-2199, USA, October 2006.

[6] Victor A Carreño and Paul S Miner. Specification of the IEEE-854 floating-point standard in HOL and PVS. *High Order Logic Theorem Proving and Its Applications*, 1995.

[7] Marius Cornea, John Harrison, Cristina Anderson, Peter Tang, Eric Schneider, and Evgeny Gvozdev. A software implementation of the IEEE 754R decimal floating-point arithmetic using the binary encoding format. *Computers, IEEE Transactions on*, 58(2):148–162, 2009.

[8] Marc Daumas, Laurence Rideau, and Laurent Théry. A generic library for floating-point numbers and its application to exact computing. In *Theorem Proving in Higher Order Logics*, pages 169–184. Springer, 2001.

[9] Frédéric Goualard. How do you compute the midpoint of an interval? *ACM Transactions on Mathematical Software (TOMS)*, 40(2):11, 2014.

[10] John Harrison. HOL Light: A tutorial introduction. In *Formal Methods in Computer-Aided Design*, pages 265–269. Springer, 1996.

[11] John Harrison. A machine-checked theory of floating point arithmetic. In *Theorem Proving in Higher Order Logics*, pages 113–130. Springer, 1999.

[12] Floating-point formalization in HOL4. https://github.com/HOL-Theorem-Prover/HOL/tree/master/src/floating-point.

[13] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual*. November 2014.

[14] Siegfried M Rump. *INTLAB—INTerval LABoratory*. Springer, 1999.

[15] David M Russinoff. A case study in formal verification of register-transfer logic with ACL2: The floating point adder of the AMD Athlon™ Processor. In *Formal Methods in Computer-Aided Design*, pages 22–55. Springer, 2000.

[16] Lei Yu. A formal model of IEEE floating point arithmetic. *Archive of Formal Proofs*, July 2013. http://afp.sf.net/entries/IEEE_Floating_Point.shtml, Formal proof development.