Formal Verification of a Theory of IEEE Rounding

Christian Jacobi

Saarland University, Computer Science Department
66123 Saarbrücken, Germany
cj@cs.uni-sb.de
Tel +49-681-302-4129, Fax -4290

August 8, 2001

Abstract. We report on the formal verification of a theory of IEEE rounding in the theorem prover PVS. The theory consists of a formalization of the IEEE standard, and notations and theorems facilitating the verification of floating point hardware. In particular, the concepts of α-equivalence and round decomposition are formalized, allowing for a subdivision of floating point units into smaller building blocks, which then can be verified separately. The theory has been successfully applied to the verification of a fully IEEE compliant floating point unit.

1 Introduction

In [6, 15], a theory of IEEE rounding was presented. This theory is used in [15] to prove the correctness of a floating point unit (FPU). This paper describes the formal verification of this theory of rounding in the theorem prover PVS [17]. The verified theory has been successfully used to formally verify the complete FPU [3].

The theory consists of a formalization of the IEEE standard 754 [11], and notations and theorems facilitating the verification of floating point hardware. Since the design of floating point units is an error prone process and the correctness of the floating point hardware strongly depends on the correctness of the rounding theory, we have formally verified this rounding theory.

The major concepts of the theory are *factorings*, *α-equivalence*, and *round decomposition*. *Factorings* are an abstraction of IEEE floating point numbers, which allows talking about numbers instead of their bitvector encodings. This enables concise statements without having to deal with the actual IEEE formats and special cases (e.g., $\pm\infty$ and NaN).

The concept of *α-equivalence* partitions the real numbers into equivalence classes such that equivalent numbers are rounded to the same floating point number. This enables the decomposition of the FPU into computational units (e.g., adder and multiplier), and a rounding unit. The computational unit delivers a result to the rounder that needs not to be the exact result of the operation but an equivalent result. The rounder therefrom computes the rounded floating point number and the exceptions. The exception computation is a central part of the rounding unit.

The process of rounding a real number to the appropriate representable floating point number is split into three simple steps by means of *round decomposition*. This enables a similar decomposition of the rounding hardware into three smaller modules.

Furthermore, round decomposition is a useful tool for proving properties of the rounding function.

The verification of complex hardware systems such as FPUs depends on a subdivision of the system into smaller parts, which then are verified separately. In this sense, α-equivalence and round decomposition ease the problem of floating point hardware verification. This has been successfully applied in a project at Saarland University, where we formally verified a fully IEEE compliant floating point unit [3].

Project Status. The verification of the theory of rounding is part of a larger project aiming to formally verify the VAMP microprocessor. The VAMP microprocessor is a variant of the DLX [9, 15], a RISC processor based on the MIPS instruction set. The VAMP features an out-of-order Tomasulo scheduler, delayed branch, precise and nested interrupts, cache memory, and a fully IEEE compliant FPU.

We have verified a library of basic circuits, upon which more complex circuits are built [4]. The verification of the Tomasulo CPU core is finished [12]. The verification of the cache has just begun. The verification of the floating point theory and hardware is complete. The floating point unit supports both single and double precision. Denormal numbers are handled in hardware. Exceptions are computed as mandated by the standard. The supported operations are addition, subtraction, multiplication, division (by Newton-Raphson iteration), comparisons, and conversion between the floating point formats and between floating point numbers and integers [3].

The complete hardware of the VAMP processor is described on the gate level in PVS. We have developed a translation tool to automatically convert the PVS hardware description to Verilog HDL. The verified VAMP processor will be implemented on a Xilinx FPGA. We have already converted the complete multiplication/division-FPU to Verilog and have implemented it on the FPGA.

Related Work. As mentioned above, the central concepts in this paper are taken from [6, 15]. The paper-and-pencil proofs in [6, 15] served as guidelines in our formal verification. The significance of the proofs in this paper is that they are formally verified; they are excerpts of the actual PVS proofs.[1] Some proofs in [6, 15] had bugs, and most proofs had gaps which had to be filled for the formal verification.

Miner has previously formalized the IEEE standard in PVS [13]. Our definition of the rounding function is based on this work, since the definition in [6, 15] is informal. Miner's formalization does not comprise theorems related to our α-equivalence and round decomposition theorems.

Another formalization of the IEEE standard was given by Harrison [8] in the theorem prover HOL Light. Harrison does not discuss exponent wrapping, which introduces some ambiguities in the definition of the inexact exception (cf. Sect. 4). Harrison's formalization has no counterpart to round decomposition. He has theorems related to the computation of exceptions of α-equivalent numbers [8, Sect. 5.3], but does not relate them to sticky-bit computations. However, this is essential to subdivide the FPU into computational units and a rounder unit in our verification project.

[1] The PVS files are available at http://www-wjp.cs.uni-sb.de/projects/verification/

Barrett has specified parts of the IEEE standard in Z [2]. In [14], Moore et al. verify the AMD K5 floating point division algorithm. They have a definition of sticky bit computations, which is similar to our α-equivalence. They do not describe exceptions and round decomposition. In [18–20], Russinoff proves the correctness of the AMD K5 square root, and the AMD Athlon square root, division, and addition algorithms. His formalization of the rounding function and sticky bit computations is similar to [14]. Russinoff does not cover denormals, exceptions, and round decomposition; however, he states that he handles denormals in unpublished work (private communication).

There are other verification projects for floating point hardware, e.g., [1, 5, 16]. All these projects use less intuitive formalizations of IEEE rounding. They do not cover denormals and exceptions.

2 Factorings

2.1 Basic Definitions

We abstract IEEE numbers as defined in the standard to *factorings*. A factoring is a triple (s, e, f) with sign bit $s \in \{0, 1\}$, exponent $e \in \mathbb{Z}$, and significand $f \in \mathbb{R}_{\geq 0}$. Note that exponent range and significand precision are unbounded. The value of a factoring is

$$[\![s, e, f]\!] := (-1)^s \cdot 2^e \cdot f.$$

The standard introduces an exponent width N, from which constants $e_{min} := -2^{N-1} + 2$ and $e_{max} := 2^{N-1} - 1$ are derived. These constants are used to bound the exponent range.

We call a factoring (s, e, f) *normal* if $e \geq e_{min}$ and $1 \leq f < 2$. A factoring is called *denormal* if $e = e_{min}$ and $0 \leq f < 1$. We call a factoring an *IEEE factoring* if it is either normal or denormal.

Lemma 1. *A factoring (s,e,f) has zero value, iff* $f = 0$.[2]

The next lemma states that nonzero IEEE factorings are unique:

Lemma 2. *Let* (s, e, f) *and* (s', e', f') *be IEEE-factorings with nonzero value. It holds*

$$[\![s, e, f]\!] = [\![s', e', f']\!] \Longleftrightarrow (s, e, f) = (s', e', f').$$

Zero has two IEEE factorings $(0, e_{min}, 0)$ *and* $(1, e_{min}, 0)$*, called* $+0$ *and* -0*, respectively.*

2.2 IEEE Factorings

Next, we define the normalization algorithm. We start by defining a function $\widehat{norm}$, which maps nonzero factorings to factorings with significand between 1 and 2:

$$\widehat{norm}(s, e, f) := (s, e + \lfloor \log_2 f \rfloor, f \cdot 2^{-\lfloor \log_2 f \rfloor}).$$

[2] We omit the proofs of trivial lemmas.

We proceed with the definition of the function $norm$, which maps any (possibly zero) factoring to an IEEE-factoring. Let $(\hat{s}, \hat{e}, \hat{f}) := \widehat{norm}(s, e, f)$:

$$norm(s,e,f) := \begin{cases} (\hat{s}, \hat{e}, \hat{f}) & \text{if } f \neq 0 \text{ and } \hat{e} \geq e_{min}, \\ (\hat{s}, e_{min}, \hat{f} \cdot 2^{\hat{e}-e_{min}}) & \text{if } f \neq 0 \text{ and } \hat{e} < e_{min}, \\ (s, e_{min}, 0) & \text{if } f = 0. \end{cases}$$

The following lemma summarizes the most important properties of the normalization functions:

Lemma 3. *Let (s, e, f) be an arbitrary factoring. It holds:*[3]

1. $[\![\widehat{norm}(s,e,f)]\!] = [\![s,e,f]\!]$, *if* $f \neq 0$,
2. $1 \leq \widehat{norm}_f(s,e,f) < 2$, *if* $f \neq 0$,
3. $[\![norm(s,e,f)]\!] = [\![s,e,f]\!]$,
4. $norm(s,e,f)$ *is an IEEE-factoring.*

Having defined the normalization algorithm, we define conversion functions η and $\hat{\eta}$, which assign factorings to reals x:

$$\begin{aligned} \hat{\eta}(x) &:= \widehat{norm}(sign(x), 0, |x|) && \text{for } x \neq 0, \\ \eta(x) &:= norm(sign(x), 0, |x|) && \text{for arbitrary } x, \end{aligned}$$

where $sign(x) = 0$ if $x \geq 0$, and $sign(x) = 1$ otherwise.[4]

Lemma 4. *Let $x \in \mathbb{R}$. It holds:*

1. $x = [\![\hat{\eta}(x)]\!]$ *if* $x \neq 0$,
2. $x = [\![\eta(x)]\!]$

Lemma 5. *Let $x \in \mathbb{R}$ with $x \neq 0$ in the context of $\hat{\eta}$. It holds:*

$$\hat{\eta}_e(x) = \lfloor \log_2 |x| \rfloor \qquad \eta_e(x) = \begin{cases} \lfloor \log_2 |x| \rfloor & \text{if } x \neq 0 \text{ and } \lfloor \log_2 |x| \rfloor \geq e_{min}, \\ e_{min} & \text{otherwise.} \end{cases}$$

$$\hat{\eta}_f(x) = |x| \cdot 2^{-\hat{\eta}_e(x)} \qquad \eta_f(x) = |x| \cdot 2^{-\eta_e(x)}$$

Lemmas 3 to 5 are proved by definition unfolding.

Lemma 6. *Let (s, e, f) be an arbitrary factoring with nonzero value $x := [\![s,e,f]\!]$. It holds*

1. $|x| \geq 2^{e_{min}} \Longrightarrow \eta(x) = \hat{\eta}(x)$, *i.e., η and $\hat{\eta}$ coincide for normal numbers.*
2. *If* $1 \leq f < 2$, *it holds* $(s,e,f) = \hat{\eta}([\![s,e,f]\!])$.
3. *If (s,e,f) is an IEEE-factoring, it holds* $(s,e,f) = \eta([\![s,e,f]\!])$.

Statements 1 and 2 are simple consequences of lemma 5. Statement 3 is proved by using lemma 2 with $(s', e', f') = \eta([\![s,e,f]\!])$.

[3] $\widehat{norm}_f(s,e,f)$ denotes the f-component of $\widehat{norm}(s,e,f)$; analogous for other functions and components.

[4] We distinguish $+0$ and -0 in our theory of factorings, but for the conversion from reals to factorings we convert $0 \in \mathbb{R}$ to $+0$.

2.3 Representable Factorings

Let P be the significand precision as defined in the standard. A significand f is called *representable*, if f has at most $P-1$ digits behind the binary point, i.e., if $2^{P-1} \cdot f \in \mathbb{N}_0$. We call an IEEE-factoring (s, e, f) *semi-representable*, if f is representable. We call an IEEE-factoring *representable*, if it is semi-representable, and furthermore $e \leq e_{max}$ holds. We call a real x (semi-)representable, if $\eta(x)$ is (semi-) representable.

Representable numbers exactly correspond to the representable numbers as defined in the standard. Common values for (N, P) are $(8, 24)$ and $(11, 53)$, called single and double precision, respectively. However, the theory described here is not limited to these values of N and P. We only assume $N > 2$ and $P > 1$. The standard defines an encoding of single and double precision IEEE factorings into bit strings of length 32 and 64, respectively. The idea behind factorings is to leave the bitvector level and argue about the more abstract factorings. This speeds up the verification of hardware.

The following lemma bounds (semi-)representable numbers.

Lemma 7. *Let (s, e, f) be a semi-representable factoring, and $i > e$ be an integer. It holds*

1. $f \leq 2 - 2^{1-P}$,
2. $|[\![s, e, f]\!]| \leq 2^i - 2^{i-P}$,
3. $X_{max} := 2^{e_{max}} \cdot (2 - 2^{1-P})$ *is the largest representable number.*

The following lemma characterizes the distance between distinct semi-representable factorings:

Lemma 8. *Let (s, e, f) and (s', e', f') be semi-representable factorings with values $x := [\![s, e, f]\!]$ and $x' := [\![s', e', f']\!]$, let $x \neq x'$, and i be an integer. It holds*

$$e \geq i \text{ and } e' \geq i \Longrightarrow |x - x'| \geq 2^{i-(P-1)}.$$

3 Rounding

Since (semi-)representable numbers are not closed under arithmetic operations (e.g., addition, division), the IEEE-standards defines four rounding modes: round to nearest, round up, round down, and round to zero. In this section, we define the rounding function, which maps arbitrary reals to semi-representable factorings according to the standard. The definition is similar to Miner's definition [13]; it only differs in cases of overflow and underflow (Sect. 4).

3.1 Definition

We start with the definition of a function $r_{int}(\,\cdot\,, \mathcal{M})$ for each rounding mode $\mathcal{M} \in \{near, up, down, zero\}$, which rounds reals x to integers:

$$r_{int}(x, up) := \lceil x \rceil \qquad r_{int}(x, near) := \begin{cases} \lfloor x \rfloor & \text{if } x - \lfloor x \rfloor < \lceil x \rceil - x, \\ \lceil x \rceil & \text{if } x - \lfloor x \rfloor > \lceil x \rceil - x, \\ x & \text{if } \lfloor x \rfloor = \lceil x \rceil, \\ 2 \lfloor \lceil x \rceil / 2 \rfloor & \text{otherwise.} \end{cases}$$

$$r_{int}(x, down) := \lfloor x \rfloor \qquad r_{int}(x, zero) := (-1)^{sign(x)} \cdot \lfloor |x| \rfloor$$

By scaling by 2^{P-1}, reals are rounded to rationals with $P-1$ fractional bits:

$$r_{rat}(x, \mathcal{M}) := 2^{-(P-1)} \cdot r_{int}(x \cdot 2^{P-1}, \mathcal{M}).$$

Further scaling with 2^e, $e := \eta_e(x)$, yields the IEEE rounding function:

$$rd(x, \mathcal{M}) := 2^e \cdot r_{rat}(x \cdot 2^{-e}, \mathcal{M}).$$

It is not obvious that this definition conforms with the IEEE standard. In section 3.3 we prove a theorem to convince the reader of the conformance.

3.2 Decomposition Theorem

The decomposition theorem we prove in this section decomposes the computation of the rounding function into three steps: η-computation (sometimes called pre-normalization in the literature), significand rounding, and a post-normalization. The benefit of having the decomposition theorem is that it simplifies the design and verification of rounder implementations. Furthermore, it is a powerful tool in other proofs, e.g., in theorem 7.

The η-computation step computes the IEEE factoring $\eta(x)$, where x is the number to be rounded. The significand round step then rounds the significand computed in the η-computation to $P-1$ digits behind the binary point. This is formalized in the function $sigrd$:

$$sigrd(X, \mathcal{M}) := \left| r_{rat}\left((-1)^s \cdot f, \mathcal{M}\right) \right|,$$

where $X = (s, e, f)$ is an IEEE factoring, and $\mathcal{M}$ is a rounding mode. The following lemma states some properties of the $sigrd$ function:

Lemma 9.

1. $sigrd(X, \mathcal{M}) = |rd([\![X]\!], \mathcal{M})| \cdot 2^{-e}$,
2. $0 \leq sigrd(X, \mathcal{M}) \leq 2$,
3. $1 \leq f \Longrightarrow 1 \leq sigrd(X, \mathcal{M})$,
4. $sigrd(X, \mathcal{M}) \cdot 2^{P-1}$ *is an integer.*

Part 1 follows by expanding the definitions of $sigrd$ and rd. For parts 2 and 3 one expands the definition down to r_{int} and applies basic properties of the floor and ceiling functions. Part 4 is a direct consequence of the definition of r_{rat}.

In the case that the significand round returns 0 or 2, the factoring has to be post-normalized; if the significand round returns 0, the sign bit is forced to 0 in order to yield $\eta(0)$. In case the significand round returns 2, the exponent is incremented, and the significand is forced to 1:

$$postnorm(X, \mathcal{M}) = \begin{cases} (s, e, sigrd(X, \mathcal{M})) & \text{if } 0 < sigrd(X, \mathcal{M}) < 2, \\ (s, e+1, 1) & \text{if } sigrd(X, \mathcal{M}) = 2, \\ (0, e_{min}, 0) & \text{if } sigrd(X, \mathcal{M}) = 0. \end{cases}$$

Theorem 1. *The result $postnorm(X, \mathcal{M})$ of the post-normalization is a semi-representable IEEE-factoring.*

Proof. The case $sigrd(X, \mathcal{M}) \in \{0, 2\}$ is trivial. Assume $0 < sigrd(X, \mathcal{M}) < 1$. By lemma 9.3 we know $f < 1$, and hence $e = e_{min}$ since X is an IEEE-factoring. Therefore $postnorm(X, \mathcal{M})$ is an IEEE-factoring, and with lemma 9.4 it is a semi-representable factoring.

Now assume $1 \leq sigrd(X, \mathcal{M}) < 2$. Since the input X is an IEEE-factoring, we know $e \geq e_{min}$, and hence $(s, e, sigrd(X, \mathcal{M})) = postnorm(X, \mathcal{M})$ is an IEEE-factoring; semi-representability now follows from lemma 9.4. □

Lemma 10. $[\![postnorm(X, \mathcal{M})]\!] = rd([\![X]\!], \mathcal{M})$.

Proof. Apply lemma 9.1 and expand definitions. □

Theorem 2 (Decomposition Theorem). *For any real x, and rounding mode $\mathcal{M} \in \{near, up, down, zero\}$, it holds*

$$postnorm\big(\eta(x), \mathcal{M}\big) = \eta\big(rd(x, \mathcal{M})\big).$$

Proof. For nonzero rounding results, the claim follows from lemmas 6.3 and 10. Otherwise, the claim follows from the definitions of $norm, \eta$, and $postnorm$. □

The IEEE factoring of the rounding result can therefore be computed by first computing the IEEE factoring of x, then rounding the significand, and finally post-normalizing the result. This decomposition of the rounding function is well known, but has been proved explicitly for the first time in [15]. The IEEE formalizations by Miner [13] and Harrison [8] do not feature a counterpart of round decomposition.

The decomposition theorem has proved to be of great value in the verification of the VAMP rounding hardware [3]. We have verified the individual hardware components for η-computation, significand rounding, and post-normalization, and have concluded the correctness of the rounding hardware by means of the decomposition theorem. We believe that the verification of the rounding hardware without the concept of decomposition would have been far more complicated.

3.3 Correctness of the Rounding Function

We now demonstrate that the definition of the IEEE rounding function rd conforms with the IEEE standard. The specification of the round to nearest mode in the IEEE standard is as follows:

> *... In this mode the representable value nearest to the infinitely precise result* [*of any floating point operation*] *shall be delivered; if the two nearest representable values are equally near, the one with its least significant bit zero shall be delivered. ...*

Since our formal definition of the function rd does not obviously coincide with this informal definition, the following theorem is proved. This theorem hopefully convinces the reader of the conformance of our rounding definition.

Theorem 3. *Let $x, x' \in \mathbb{R}$ and x' be a semi-representable number.*

1. *For any rounding mode $\mathcal{M}$, $rd(x, \mathcal{M})$ is semi-representable.*
2. *$rd(x, near)$ is a* nearest *semi-representable number: $|x - x'| \geq |x - rd(x, near)|$.*
3. *If there are* two *nearest numbers, then the one with least significant bit zero is chosen: $x' \neq rd(x, near)$ and $|x - x'| = |x - rd(x, near)|$ implies $\left(\eta_f(rd(x, near)) \cdot 2^{P-1}\right)$ is even.*

Similar theorems exist for the three remaining rounding modes.

Proof. Part 1 is a trivial consequence of theorems 1 and 2. Part 2 and 3 rely on the following fact proved by Miner in PVS [13]:

$$|x - r_{int}(x, near)| \leq \tfrac{1}{2} \text{ and}$$
$$|x - r_{int}(x, near)| = \tfrac{1}{2} \Longrightarrow r_{int}(x, near) \text{ is even.}$$

Let $(s, e, f) := \eta(x)$ and $(s', e', f') = \eta(x')$. It is easy to adopt the above fact to the rd-function:

$$|x - rd(x, near)| \leq 2^{e-P} \text{ and} \tag{1}$$
$$|x - rd(x, near)| = 2^{e-P} \Longrightarrow \left(rd(x, near) \cdot 2^{-(1+e-P)}\right) \text{ is even.}$$

We now prove part 2. We may assume that $x' \neq rd(x, near)$, since otherwise the claim is trivial. From the decomposition theorem and the definition of the post-normalization we know that $\eta_e(rd(x, near)) \geq e$. Now assume $e' \geq e$. Using lemma 8 (where we set $(s', e', f') = \eta(rd(x, near))$ and $i = e$) results in

$$|rd(x, near) - x'| \geq 2^{e-(P-1)} = 2 \cdot 2^{e-P}. \tag{2}$$

Using the triangle inequality, (1), and (2) together yield

$$|x - x'| \geq 2^{e-P}. \tag{3}$$

Equations (1) and (3) yield part 2. Assume otherwise that $e' < e$. Since $e_{min} \leq e'$ we have $e_{min} < e$, and therefore $f \geq 1$, since (s, e, f) is an IEEE factoring. Hence $|x| \geq 2^e$. Lemma 7.2 with $i = e$ gives $|x'| \leq 2^e - 2^{e-P}$. Together this implies

$$|x' - x| \geq 2^{e-P}. \tag{4}$$

Again, (1) and (4) yield part 2. The proof of part 3 is similar. □

Similar informal specifications exist in the standard for the three remaining rounding modes, and conformance theorems for these have been proved in PVS.

Theorem 4. *For any real x and rounding mode $\mathcal{M}$, x is semi-representable, if and only if $rd(x, \mathcal{M}) = x$.*

Proof. If $rd(x, \mathcal{M}) = x$, x is semi-representable by theorem 3.1. Conversely, if x is semi-representable and $\mathcal{M} = near$, then the round result must equal x by theorem 3.2. The claim for the remaining rounding modes follows analogously from their respective conformance theorems. □

4 Exceptions and Wrapped Exponents

The IEEE standard defines five exceptions: invalid operation (*INV*), division by zero (*DIVZ*), overflow (*OVF*), underflow (*UNF*), and inexact result (*INX*). The INV and DIVZ exceptions are trivial, and therefore not of interest in this paper.

The standard requires that each occurrence of an exception shall set a status flag and call a trap handler. The trap handler can be disabled on the user's request. We do not describe the actual handling of the status flags and the trap handling, since this is part of the CPU, not part of the FPU. However, since the detection of exceptions, as well as the final result of floating point operations depend on whether the trap handlers are enabled or disabled, we need the enable flags for the overflow and underflow exceptions *OVFen* and *UNFen*, respectively. They are provided by the CPU.

Overflow. The standard defines the overflow exception as follows:

> *The overflow exception shall be signaled whenever the destination format's largest finite number is exceeded in magnitude by what would have been the rounded floating-point result were the exponent range unbounded. . . .*

In lemma 7 we stated that $X_{max} = 2^{e_{max}} \cdot (2 - 2^{1-P})$ is the format's largest representable value. Since our rounding function by definition rounds as if the exponent range was unbounded above, we can define the *OVF* exception as follows:

$$\mathit{OVF}(x, \mathcal{M}) := (rd(x, \mathcal{M}) > X_{max})\,.$$

Here, x is the exact result of a floating point operation. The *OVF* exception depends on the rounding mode, since different rounding modes round large numbers differently to either X_{max}, or to the next value outside the format's range.

Underflow. The standard defines the underflow exception as follows:

> *Two correlated events contribute to underflow. One is the creation of a tiny nonzero result between* $\pm 2^{e_{min}}$ *(. . .). The other is extraordinary loss of accuracy . . .*
> *When an underflow trap (. . .) is not enabled (. . .), underflow shall be signaled when both tininess and loss of accuracy have been detected. When an underflow trap (. . .) is enabled, underflow shall be signaled when tininess is detected regardless of loss of accuracy. . . .*

For each of the contributing events, the standard leaves the choice between two different implementations. We use *tininess before rounding* (instead of after rounding) and *inexact result* as loss of accuracy (instead if denormalization loss). Tininess before rounding occurs

> *. . . when a nonzero result computed as though both exponent range and the precision were unbounded would lie strictly between* $\pm 2^{e_{min}}$.

This is formalized as

$$\mathit{TINY}(x) := (x \neq 0 \wedge |x| < 2^{e_{min}})\,.$$

Here again, x is the exact result of a floating point operation, and therefore is "computed as though both exponent range and the precision were unbounded." An inexact result occurs

> *... when the delivered result differs from what would have been computed were both exponent range and precision unbounded.*

We formalize this as

$$\mathit{LOSS}(x, \mathcal{M}) := (rd(x, \mathcal{M}) \neq x)\,.$$

Loss of accuracy only syntactically depends on the rounding mode, since this is a required parameter to the rd-function. From theorem 4 it follows $\mathit{LOSS}(x, \mathcal{M}_1) = \mathit{LOSS}(x, \mathcal{M}_2)$ for distinct rounding modes $\mathcal{M}_i$.

Having defined tininess and loss of accuracy, we can define the underflow exception:

$$\mathit{UNF}(x, \mathcal{M}) := \mathit{TINY}(x) \wedge (\mathit{LOSS}(x, \mathcal{M}) \vee \mathit{UNFen})\,.$$

As mentioned above, the standard leaves other choices for the definition of *TINY* and *LOSS*. We refer the reader to [8, 15] for lemmas about the relations between the different definitions.

Wrapped Exponent. In case of an overflow or underflow with corresponding traps enabled, the standard requests to deliver a biased result to the trap handler:

> *Trapped overflows (...) shall deliver to the trap handler the result obtained by dividing the infinitely precise result by 2^A and then rounding. The bias adjust A is 192 in the single, 1536 in the double format. ...*

Note that $A = 3 \cdot 2^{N-2}$ with exponent width $N = 8$ and $N = 11$, respectively. Analogously to overflows, trapped underflows shall deliver the result obtained by multiplying the exact result with 2^A and then rounding. This is captured in the following definition. Again, x is the exact result of a floating point operation:

$$wrapped(x, \mathcal{M}) := \begin{cases} x \cdot 2^{-A} & \text{if } \mathit{OVF}(x, \mathcal{M}) \text{ and } \mathit{OVFen}, \\ x \cdot 2^{A} & \text{if } \mathit{UNF}(x, \mathcal{M}) \text{ and } \mathit{UNFen}, \\ x & \text{otherwise.} \end{cases}$$

Now we are ready to define the final floating point result of operations yielding the exact result x:

$$result(x, \mathcal{M}) := rd(wrapped(x, \mathcal{M}), \mathcal{M})$$

Multipliying the result with $2^{\pm A}$ before rounding scales the result into the representable range.

Lemma 11. *Let x be the exact result of an operation, and $\mathcal{M}$ be a rounding mode. Assume $|x| > 2^{e_{min}-A}$ (this is fulfilled for $+, -, \times, \div$ operations). Let $(\hat{s}, \hat{e}, \hat{f}) := \hat{\eta}(x)$.*

1. $\mathrm{OVF}(x, \mathcal{M}) \implies \eta(x \cdot 2^{-A}) = (\hat{s}, \hat{e} - A, \hat{f})$
2. $\mathrm{TINY}(x) \implies \eta(x \cdot 2^{A}) = (\hat{s}, \hat{e} + A, \hat{f})$

The IEEE factoring of the wrapped result can therefore be computed by adding/subtracting A from the exponent of the exact $\hat{\eta}$-factoring. The above lemma is used in conjunction with the decomposition theorem to round the wrapped result.

If an overflow is detected with disabled trap, the $result$ definition above returns a result exceeding X_{max}. The standard however requests a final result of either $\pm X_{max}$ or $\pm\infty$, depending on the sign and the rounding mode. This is formalized as a case-split. We do not cover the details here, since this is a transliteration of Sect. 7.3. of the standard.

Inexact. The standard defines the inexact exception as follows:

> *If the rounded result of an operation is not exact or if it overflows without an overflow trap, then the inexact exception shall be signaled. . . .*

It is not clear if the rounded result is meant to be $rd(x, \mathcal{M})$ without being wrapped or $result(x, \mathcal{M})$, which potentially has been wrapped. Harrison [8] defines the inexact exception as

$$\mathit{INX}(x, \mathcal{M}) := \mathit{LOSS}(x, \mathcal{M}) \vee (\mathit{OVF}(x, \mathcal{M}) \wedge \overline{\mathit{OVFen}}).$$

In contrast, a test on Intel's Pentium II with the operation $x := x_{min}/2$ with enabled underflow trap and $\mathcal{M} = up$ did not yield an *INX* signal (where x_{min} is the smallest representable value). If the *INX* signal was computed as $rd(x, \mathcal{M}) \neq x$, the rounded result $rd(x, up) = x_{min}$ would differ from x and so an *INX* signal should be set.[5]

We define the inexact exception as

$$\mathit{INX}(x, \mathcal{M}) := \mathit{LOSS}(wrapped(x, \mathcal{M}), \mathcal{M}) \vee (\mathit{OVF}(x, \mathcal{M}) \wedge \overline{\mathit{OVFen}}).$$

This is the definition also used in IBM's S/390 [10, Pg. 19-22] and in [15], e.g. It has the advantage that programs can distinguish exact (except for exponent wrapping) from inexact computations in case of trapped overflows and underflows. For example, the above computation $x := x_{min}/2$ can be represented exactly after having been multiplied with 2^A.

Harrison and we think that the standard is ambiguous in this point (personal communication).

5 α-Equivalence

We now define the concept of α-equivalence and α-representatives [15]. This concept is a very concise way to speak about sticky-bit computations.

Let α be an integer. Two reals x and y are said to be α-*equivalent* ($x \equiv_\alpha y$), if $x = y$ or if there exists some $q \in \mathbb{Z}$ with $q \cdot 2^\alpha < x, y < (q+1) \cdot 2^\alpha$, i.e., if both x and y lie in the same open interval between two consecutive integral multiples of 2^α. Clearly, if such an q exists, it must be $q_\alpha(x) := \lfloor x \cdot 2^{-\alpha} \rfloor$. The α-representative of x is defined as

$$[x]_\alpha := \begin{cases} x & \text{if } x = q_\alpha(x) \cdot 2^\alpha, \\ (q_\alpha(x) + \frac{1}{2}) \cdot 2^\alpha & \text{otherwise.} \end{cases}$$

[5] The test-program is available at out website.

If x is an integral multiple of 2^α, the representative of x is x itself, and the midpoint of the interval between the surrounding multiples of 2^α otherwise. The following lemma summarizes some important facts:

Lemma 12. *Let x, y be reals, and α, k be integers.*

1. $\equiv_\alpha$ *is an equivalence relation,*
2. $x \equiv_\alpha [x]_\alpha$,
3. $x \equiv_\alpha y \iff [x]_\alpha = [y]_\alpha$, *(representative equivalence)*
4. $x \equiv_\alpha y \iff -x \equiv_\alpha -y$, *and* $[-x]_\alpha = -[x]_\alpha$, *(negative value)*
5. $x \equiv_\alpha y \iff 2^k \cdot x \equiv_{\alpha+k} 2^k \cdot y$, *and* $[2^k \cdot x]_{\alpha+k} = 2^k \cdot [x]_\alpha$, *(scaling)*
6. $x \equiv_\alpha y \iff x + k \cdot 2^\alpha \equiv_\alpha y + k \cdot 2^\alpha$, *(translation)*
7. $x \equiv_\alpha y \implies x \equiv_{\alpha+k} y$ *if* $k \geq 0$, *(coarsening)*
8. $x = 0 \iff x \equiv_\alpha 0 \iff [x]_\alpha = 0$, *(zero value)*

Parts 1-5 are simple consequences of the definition, parts 6-8 are proved by induction on k.

The following theorem describes equivalence on factorings:

Lemma 13. *Let x, x' be nonzero reals, $e := \eta_e(x), e' := \eta_e(x'), \hat{e} := \hat{\eta}_e(x), \hat{e}' := \hat{\eta}_e(x')$, and α be an integer. It holds*

1. $x \equiv_\alpha y \implies sign(x) = sign(y)$,
2. $\alpha \leq \hat{e}$ *and* $x \equiv_\alpha x' \implies \hat{e} = \hat{e}'$,
3. $\alpha \leq e$ *and* $x \equiv_\alpha x' \implies e = e'$,
4. $|x| \geq 2^{e_{min}}$ *and* $\alpha \leq e \implies \hat{e} = \hat{\eta}_e([x]_\alpha)$,
5. $|x| < 2^{e_{min}}$ *and* $\alpha \leq e \implies \hat{\eta}_e([x]_\alpha) < e_{min}$.

Proof. Due to lack of space, we only prove part 2. With lemma 12.7 it suffices to proof the claim for $\alpha = \hat{e}$. By part 1 and lemma 12.4 we may assume $x, x' \geq 0$.

Since the claim is trivial for $x = x'$, we further assume that $q_{\hat{e}}(x) \cdot 2^{\hat{e}} < x, x' < (q_{\hat{e}}(x)+1) \cdot 2^{\hat{e}}$ by definition of α-equivalence. From lemma 3.2, we know $1 \leq x \cdot 2^{-\hat{e}} < 2$, and therefore $q_{\hat{e}}(x) = \lfloor x \cdot 2^{-\hat{e}} \rfloor = 1$. We then have $2^{\hat{e}} < x, x' < 2^{\hat{e}+1}$, and therefore $\hat{e} = \lfloor \log x \rfloor = \lfloor \log x' \rfloor$. Lemma 5 proves the claim. □

Now we are ready to prove an important theorem, which allows the easy computation of IEEE-factorings corresponding to representatives:

Theorem 5. *Let $x \in \mathbb{R}$, let $(s, e, f) := \eta(x)$ be the corresponding IEEE factoring, and let $p \geq 0$ be an integer. The IEEE factoring of $[x]_{e-p}$ can be computed by computing the representative $[f]_{-p}$ of f:*

$$\eta([x]_{e-p}) = (s, e, [f]_{-p}).$$

Proof. From lemma 13.1 and 13.3 we know that $\eta_s([x]_{e-p}) = s$ and $\eta_e([x]_{e-p}) = e$. From lemma 5 we know $\eta_f([x]_{e-p}) = |[x]_{e-p}| \cdot 2^{-e}$. With lemma 12.4 and 12.5, we have $|[x]_{e-p}| \cdot 2^{-e} = [|x| \cdot 2^{-e}]_{-p}$. Lemma 5 gives $|x| \cdot 2^{-e} = f$, and hence $\eta_f([x]_{e-p}) = [f]_{-p}$. □

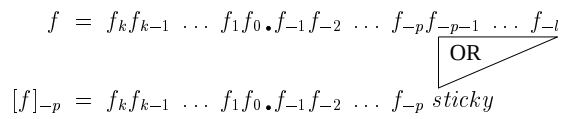

Fig. 1. Computing representatives by sticky-computation

Next, we show that the representative of f can be computed by a *sticky bit computation*. Let $f \geq 0$ be a real in binary format $f_k, \ldots, f_0, f_{-1} \ldots, f_{-l} \in \{0,1\}^{k+l+1}$ such that $f = \sum_{i=-l}^{k} f_i \cdot 2^i$. Let p be an integer, $k \geq -p > -l$. The $(-p)$-sticky-bit of f is the logical OR of all bits $f_{-p-1}, \ldots, f_{-l}$ (cf. Fig. 1):

$$sticky_{-p}(f) := f_{-p-1} \vee \ldots \vee f_{-l}.$$

Theorem 6. *With the above definitions, the representative $[f]_{-p}$ of f can be computed by replacing the less significant bits by the sticky bit:*

$$[f]_{-p} = \sum_{i=-p}^{k} f_i \cdot 2^i + 2^{-p-1} \cdot sticky_{-p}(f)$$

Proof. By definition, $q_{-p}(f) = \lfloor f \cdot 2^p \rfloor$, and therefore $q_{-p}(f) = \sum_{i=-p}^{k} f_i \cdot 2^{i+p}$. Furthermore, $f = q_{-p}(f) \cdot 2^{-p}$, iff $sticky_{-p}(f) = 0$. Applying this in the definition of $[\,\cdot\,]_{-p}$ proves the claim. □

Theorems 5 and 6 together allow a very efficient computation of representatives (respectively their IEEE-factorings) by or-ing the less significant bits in an OR tree, and replacing them by the sticky bit. This technique is well known [7], but introducing the formalism with α-representatives allows for a very concise argumentation about these sticky computations. The verification of the adder circuitry in [3], e.g., relies heavily on the concept of α-equivalence.

6 Rounding Representatives

The valuable property of α-representatives is that rounding x and its representative $[x]_{e-P}$ yields the same result:

Theorem 7. *Let $x \in \mathbb{R}$, $(s,e,f) := \eta(x)$, and $\mathcal{M}$ be a rounding mode. It holds*

$$rd(x, \mathcal{M}) = rd([x]_{e-P}, \mathcal{M}).$$

Proof. In [15], this theorem is proved by geometrical arguments. It is technically very tedious to prove this theorem in the PVS theorem prover. We only give a sketch of the PVS proof.

By theorems 2 and 5 it suffices to show

$$sigrd\big((s,e,f), \mathcal{M}\big) = sigrd\big((s,e,[f]_{-P}), \mathcal{M}\big).$$

By unfolding the definitions of $sigrd$ and r_{rat}, this is equivalent to

$$r_{int}\big((-1)^s \cdot f \cdot 2^{P-1}, \mathcal{M}\big) = r_{int}\big((-1)^s \cdot [f]_{-P} \cdot 2^{P-1}, \mathcal{M}\big). \tag{5}$$

Since the claim is trivial if $[f]_{-P} = f$, we can assume by the definition of α-equivalence that $f \cdot 2^P \notin \mathbb{Z}$, and $[f]_{-P} = (q+0.5)\cdot 2^{-P}$ with $q := q_{-P}(f) = \lfloor f \cdot 2^P \rfloor$. Substituting $[f]_{-P}$ in (5) yields

$$r_{int}((-1)^s \cdot f \cdot 2^{P-1}, \mathcal{M}) = r_{int}((-1)^s \cdot \left(\tfrac{1}{4} + \tfrac{1}{2}\lfloor f \cdot 2^P \rfloor\right), \mathcal{M}). \tag{6}$$

The theorem now follows from the next two lemmas (this "now follows" is the technically complicated part). Lemma 14 proves that the claim is correct if $\mathcal{M} \neq near$. Lemma 15 proves that the same cases apply in the definition of $rd_{int}(\,\cdot\,, near)$ on both sides of equation (6). Then the claim again follows by lemma 14. □

Lemma 14. *For all $z \in (\mathbb{R} \setminus \mathbb{Z})$ and $s \in \{0,1\}$, it holds*

$$\lfloor (-1)^s \cdot z \rfloor = \left\lfloor (-1)^s \cdot \left(\tfrac{1}{4} + \tfrac{1}{2}\lfloor 2z \rfloor\right)\right\rfloor,$$

$$\lceil (-1)^s \cdot z \rceil = \left\lceil (-1)^s \cdot \left(\tfrac{1}{4} + \tfrac{1}{2}\lfloor 2z \rfloor\right)\right\rceil.$$

Lemma 15. *For all $z \in \mathbb{R}$, $2z \notin \mathbb{Z}$ and $s \in \{0,1\}$, it holds*

$$\lceil z \rceil - z > z - \lfloor z \rfloor \iff \lceil z \rceil - \left(\tfrac{1}{4} + \tfrac{1}{2}\lfloor 2z \rfloor\right) > \left(\tfrac{1}{4} + \tfrac{1}{2}\lfloor 2z \rfloor\right) - \lfloor z \rfloor,$$

$$\lceil z \rceil - z < z - \lfloor z \rfloor \iff \lceil z \rceil - \left(\tfrac{1}{4} + \tfrac{1}{2}\lfloor 2z \rfloor\right) < \left(\tfrac{1}{4} + \tfrac{1}{2}\lfloor 2z \rfloor\right) - \lfloor z \rfloor.$$

Lemma 14 can be proved by induction on $\lfloor z \rfloor$, and some basic properties of the floor and ceiling-functions from the PVS library. The proof, however, is technical and tedious. The first part of lemma 15 is proved automatically by the PVS command `grind`, the second part needs little manual assistance.

From the above theorem, one can conclude that the wrapped and rounded result $result(x, \mathcal{M})$ can be computed using equivalence, too. However, in case of trapped underflow one needs more precision, namely $(\hat{e} - P)$-equivalence instead of $(e - P)$-equivalence:

Theorem 8. *Let $x \in \mathbb{R}$, $\hat{e} := \hat{\eta}_e(x)$, and $\mathcal{M}$ be a rounding mode. It holds*

$$result(x, \mathcal{M}) = result([x]_{\hat{e}-P}, \mathcal{M}).$$

Proof. This follows from lemma 11 and theorem 7. □

Not only the rounding can be accomplished by using the representative, but also the detection of exceptions.

Theorem 9. *Let $x \in \mathbb{R}$, $(s, e, f) := \eta(x)$, and $\mathcal{M}$ be a rounding mode. It holds*

1. $\mathrm{OVF}(x, \mathcal{M}) \iff \mathrm{OVF}([x]_{e-P}, \mathcal{M})$,
2. $\mathrm{TINY}(x) \iff \mathrm{TINY}([x]_{e-P})$,
3. $\mathrm{LOSS}(x, \mathcal{M}) \iff \mathrm{LOSS}([x]_{e-P}, \mathcal{M})$,
4. $\mathrm{UNF}(x, \mathcal{M}) \iff \mathrm{UNF}([x]_{e-P}, \mathcal{M})$,
5. $\mathrm{INX}(x, \mathcal{M}) \iff \mathrm{INX}([x]_{\hat{e}-P}, \mathcal{M})$, *where $\hat{e} := \hat{\eta}_e(x)$. Here one needs more precision analogously to theorem 8.*

Proof. Part 1 is an immediate consequence of theorem 7. Part 2 follows from lemmas 13.4 and 13.5. Part 3 is slightly more complicated. We have to prove

$$rd(x, \mathcal{M}) \neq x \iff rd([x]_{e-P}, \mathcal{M}) \neq [x]_{e-P}$$

By theorem 4, this is equivalent to

$$\eta(x) \text{ is semi-representable} \iff \eta([x]_{e-P}) \text{ is semi-representable.}$$

By theorem 5 and by definition of representability, this is equivalent to

$$f \cdot 2^{P-1} \in \mathbb{Z} \iff [f]_{-P} \cdot 2^{P-1} \in \mathbb{Z}.$$

We may assume $f \neq [f]_{-P}$, since otherwise the claim is trivial. Now assume $f \cdot 2^{P-1} \in \mathbb{Z}$. Then $q_{-P}(f) = \lfloor f \cdot 2^P \rfloor = f \cdot 2^P$ and hence $[f]_{-P} = f$. In the other case $f \cdot 2^{P-1} \notin \mathbb{Z}$ we have $[f]_{-P} = \left(q_{-P}(f) + \frac{1}{2}\right) \cdot 2^{-P}$. Hence, $[f]_{-P} \cdot 2^{P-1} = \frac{1}{2} \lfloor f \cdot 2^P \rfloor + \frac{1}{4} \notin \mathbb{Z}$.

Part 4 is a trivial consequence of the former parts. Part 5 follows analogously to part 3 together with lemma 11. □

Theorems 8 and 9 enable a subdivision of a complete FPU into computation units (e.g., adder, multiplier) and a rounder. The computation units compute a result which need not be exact but $(\hat{e} - P)$-equivalent to the exact result. The rounder therefrom rounds to the correct floating point number, and computes the exceptions. The passing of $(\hat{e} - P)$-equivalent approximates of the exact results saves very large intermediate results, e.g., during addition of the format's smallest and largest representable numbers.

7 Special Operands

The standard defines special operands such as $\pm\infty$ and Not-a-Number (NaN). The results of operations on these special operands are explicitly defined in the standard. The definitions in PVS are a transliteration of the respective sections in the standard. We omit the details.

8 Summary

We have described a formally verified theory of rounding. The verification was performed using the theorem prover PVS. The central concepts of the theory are *factorings*, *round decomposition*, and α*-equivalence*. These concepts are taken from [6, 15]. The theorems and proofs presented in this paper are largely based on the paper-and-pencil proofs in [15].

Since the definition of the rounding function is informal in [6,15], we had to replace it with a formal one. Our definition is based on the definition of rounding in Miner's IEEE formalization [13]. The change of the definition of the rounding function had major impact on some proofs from [15], in particular in theorems 7 and 9. We had to fill the informal arguments on rounding with formal ones. Furthermore, we had to fill the usual gaps and bugs in informal proofs. The proof effort for this theory was about nine months, including the time to get familiar with PVS. Most of the proofs are work intensive and use only little automation.

The theory described in this paper has been successfully applied in the verification of a fully IEEE compliant floating point unit [3], which is developed in the textbook

[15]. The concepts of α-equivalence and round decomposition greatly simplified the verification of the hardware, since we could break up the hardware into smaller building blocks, which then were verified separately.

Acknowledgments The author would like to thank Christoph Berg, Daniel Kröning, Silvia Müller, Wolfgang Paul, and Jochen Preiß for valuable discussions.

References

1. M. D. Aagaard and C.-J. H. Seger. The formal verification of a pipelined double-precision IEEE floating-point multiplier. In *ICCAD*, pages 7–10. IEEE, Nov. 1995.
2. G. Barrett. Formal methods applied to a floating-point number system. *IEEE Transactions on Software Engineering*, 15(5):611–621, May 1989.
3. C. Berg and C. Jacobi. Formal verification of the VAMP floating point unit. In *CHARME 2001*, volume 2144 of *LNCS*, 2001.
4. C. Berg, C. Jacobi, and D. Kroening. Formal verification of a basic circuits library. In *Proc. of IASTED Int. Conf. on Applied Informatics, Innsbruck (AI 2001)*. ACTA Press, 2001.
5. Y.-A. Chen and R. E. Bryant. Verification of floating point adders. In *CAV'98*, volume 1427 of *LNCS*, 1998.
6. G. Even and W. Paul. On the design of IEEE compliant floating point units. In *Proceedings of the 13th Symposium on Computer Arithmetic*. IEEE Computer Society Press, 1997.
7. D. Goldberg. Computer arithmetic. *In [9]*, 1996.
8. J. Harrison. A machine checked theory of floating point arithmetic. In *TPHOL '99*, volume 1690 of *LNCS*. Springer, 1999.
9. J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Mateo, CA, second edition, 1996.
10. IBM. *z/Architecture Principles of Operation*. Poughkeepsie, NY, Dec. 2000.
11. Institute of Electrical and Electronics Engineers. *ANSI/IEEE standard 754–1985, IEEE Standard for Binary Floating-Point Arithmetic*, 1985.
12. D. Kroening. *Formal Verification of Pipelined Microprocessors*. PhD thesis, Saarland University, Computer Science Department, 2001.
13. P. S. Miner. Defining the IEEE-854 floating-point standard in PVS. Technical Report TM-110167, NASA Langley Research Center, 1995.
14. J. S. Moore, T. Lynch, and M. Kaufmann. A mechanically checked proof of the AMD5K86 floating point division program. *IEEE Transactions on Computers*, 47(9):913–926, 1998.
15. S. M. Mueller and W. J. Paul. *Computer Architecture. Complexity and Correctness*. Springer, 2000.
16. J. O'Leary, X. Zhao, R. Gerth, and C.-J. H. Seger. IA-64 floating point operations and the IEEE standard for binary floating-point arithmetic. *Intel Technology Journal*, Q4, 1999.
17. S. Owre, N. Shankar, and J. M. Rushby. PVS: A prototype verification system. In *CADE 11*, volume 607 of *LNAI*, pages 748–752. Springer, 1992.
18. D. M. Russinoff. A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor. *LMS Journal of Computation and Mathematics*, 1:148–200, 1998.
19. D. M. Russinoff. A mechanically checked proof of correctness of the AMD K5 floating point square root microcode. *Formal Methods in System Design*, 14(1):75–125, Jan. 1999.
20. D. M. Russinoff. A case study in formal verification of register-transfer logic with ACL2: The floating point adder of the AMD Athlon processor. In *Proceeding of FMCAD-00*, volume 1954 of *LNCS*. Springer, 2000.