

Bridging the gap between formal specification and bit-level floating-point arithmetic

Sylvie Boldo^{a,*},

^a *Laboratoire de l'Informatique du Parallélisme
UMR 5668 CNRS - ENS Lyon - INRIA - UCB Lyon
46 allée d'Italie, 69364 Lyon Cedex 07, France*

Abstract

Floating-point arithmetic is defined by the IEEE-754 standard and has often been formalized. We propose a new Coq formalization based on the bit-level representation of the standard and we prove strong links between this new formalization and a previous high-level one. In this process, we have defined functions for any rounding mode described by the standard. Our library can now be applied to certify both software and hardware. Developing results in those two dramatically different directions, like no other formal development so far, guarantees that nothing was forgotten or poorly specified in our formalization. It also lets us compare our work with the existing bit-level formalizations developed with other proof assistants.

Key words: floating-point, Coq, IEEE-754, formal proof checking

1 Introduction

Formalizing floating-point arithmetic is not easy: the floating-point data type is complex, may be partially unspecified and may contain errors or strange behaviors. Moreover, it involves strong considerations on cost and efficiency and many features are the result of a long history of increasingly better prototypes. The floating-point arithmetic, defined by the IEEE-754 standard [24,25], is both widely used and little known [13]. Most programs, including numeric simulations, weather forecast and aeronautics use lots of computations which are mainly floating-point computations.

*

Email address: Sylvie.Boldo@ens-lyon.fr (Sylvie Boldo).

Moreover, floating-point arithmetic is a sensible topic. As the Pentium Bug [8], bugs can be very expensive and people have long tried to get rid of them using formal proofs [22,15]. Many formalizations were developed [6,19,23,14,17], sometimes made by processor constructors. They either consider the signification of the float or the vector of bits involved depending on the wanted type of proofs. Our formalization [10] considers the signification, is in Coq [16] and has already led to the proofs of both old and new results [11,3,18,4].

We could have based our formalization of IEEE-754 standard on the one done in Coq by Loiseleur [19] but we do not. We want to be as generic as possible to cover from very small to very wide formats, and there are only 5 formats defined by Loiseleur with given number of bits for the fraction and the exponent. We also want to use the Coq type for real numbers \mathbb{R} and its automations whereas the floats of [19] are valued in a strange dyadic type not very usable. Lastly, the vector of bits used by Loiseleur are “registers” which are more difficult to use and have less proved properties than the recent `Bvector` library of J. Duprat.

All the proofs of this paper are available at this address:

<http://lipforge.ens-lyon.fr/www/pff/>,

the corresponding files are `IEEE.v` and `RND.v`.

Our high-level formalization is easy to use but does not consider the hardware implementation required by the IEEE-754 standard. We want to formally define floats also as vectors of bits in the Coq system and prove the equivalence between the two visions. In the following, we show that one of our float can be transformed into an IEEE float and that this transformation is correct and bijective. The aim is not to create just another formalization, but to guarantee that our floats correspond to the reality.

We first describe the floats defined by the IEEE-754 standard and then our initial formalization in Section 3. We describe new results about rounding modes in Section 4. We then explain our new definitions and types in Section 5 and prove the equivalences in Section 6 before the conclusion.

2 Machine floats

2.1 Definitions

The machine floats are strictly defined by the IEEE-754 standard: a floating-point number is composed of three fields, the sign s , the unbiased exponent e and the fraction f as shown by Figure 1.

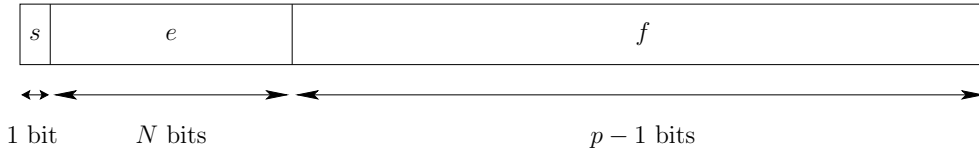


Fig. 1. A machine float

The sizes of the fraction and the unbiased exponent are defined in a format, that is the number of bits allowed for the fraction and the unbiased exponent. The standard defines the following formats:

- the single precision format (32 bits): 8 bits for the unbiased exponent and 23 bits for the fraction,
- the double precision format (64 bits): 11 bits for the unbiased exponent and 52 bits for the fraction,
- the double-extended precision format (at least 80 bits): at least 15 bits for the unbiased exponent and at least 63 bits for the fraction.

We denote p and call “precision” the number of bits allowed for the fraction plus 1 ($p = 24$ in single precision and $p = 53$ in double) as in Figure 1. There is one bit less in the fraction as the implicit one (see next subsection) is added when the float is really computed. We denote N the number of bits allowed for the unbiased exponent ($N = 8$ in single precision and $N = 11$ in double).

The IEEE-754 standard defines three types of special values:

- $\pm\infty$: they are the results of an operation such that the correct result is too big (in magnitude) to be represented.
- ± 0 : both have the value 0. The sign information indicates which infinity is the answer of the operation $1/0$.
- NaN , or Not-A-Number: it is the result of an absurd or impossible operation such as $\sqrt{-1}$ or $0/0$ or $+\infty - \infty$. A NaN means that no value (or any value) is acceptable as result for this operation.

The IEEE-754 moreover governs the way all these special values behaves when computed on. The laws are quite reasonable: if one operand is a NaN , the result is a NaN , infinities behave as mathematical infinities. . .

2.2 Interpretation

These vectors of bits must be understood as either real values or special values. We take a machine float f composed of its sign s , its unbiased exponent e and its fraction f . It belongs to a format as defined above (N bits for the unbiased exponent and $p - 1$ for the fraction). As the values of the vectors of bits are positive and as we want to get positive or negative exponent, the vector of

bits must be biased and the bias is $B = 2^{N-1} - 1$.

We denote by $1 \bullet f$ (resp. $0 \bullet f$) the value of the fixed-point number $1.b_1b_2 \dots b_{p-1}$ (resp. $0.b_1b_2 \dots b_{p-1}$) that is to say $1 + \sum_{i=1}^{p-1} b_i 2^{-i}$ (resp. $\sum_{i=1}^{p-1} b_i 2^{-i}$).

Depending on the values of e and f , the following table describes the interpretation of machine floats. It adds the way such numbers are called.

Unbiased exponent	Fraction	Value	Type
$e = 2^N - 1$	$f \neq 0$	NaN	Not-a-Number
$e = 2^N - 1$	$f = 0$	$(-1)^s \infty$	Infinity
$0 < e < 2^N - 1$		$(-1)^s \times 2^{e-B} \times (1 \bullet f)$	Normal
$e = 0$	$f \neq 0$	$(-1)^s \times 2^{1-B} \times (0 \bullet f)$	Subnormal
$e = 0$	$f = 0$	$(-1)^s 0$	Zero

For a normal float of unbiased exponent e , the biased exponent is $e - B$, so that the biased exponent range is nearly symmetrical: between $1 - B = -2^{N-1}$ and $B = 2^{N-1} - 1$. The biased exponent of subnormal floats is $1 - B$ in order to compensate the $0 \bullet f$ instead of $1 \bullet f$: there is therefore equal distance between floats both between subnormal and between the smallest positive normal float and its successor.

The IEEE-754 standard also sets many other things, including rounding modes (see Section 4), conversions, exceptions and flags.

3 Our high-level formalization of floats: pairs

This formalization is presented in [10] for Coq developments. It takes advantage of the easiness to manipulate integers with Coq, unlike rationals. Floating-point numbers are here pairs of signed integers (n, e) and are of type `float` in Coq. As we use both a signed mantissa and a signed exponent, we do not have the complexity of either the sign bit or the bias.

We then define a coercion that transforms a pair into the corresponding real value:

$$(n, e) \quad \hookrightarrow \quad n \times \beta^e \quad \in \mathbb{R}$$

where β is the radix of the floating-point system. We note $=_{\mathbb{R}}$ the equality of the real values, that is to say after coercion.

The minimal exponent is denoted by $-e_{min}$ and we do not consider an upper bound on the exponent. Unsigned mantissas are supposed to fit in p bits. A

pair (n, e) is therefore called bounded (**Fbounded** property) if $|n| < \beta^p$ and $e \geq -e_{min}$. A bound \mathcal{B} is defined as both p and e_{min} .

This definition allows possibly many pairs to share the same real value. For example using radix 2, the floats $(8, 0)$, $(4, 1)$ and $(1, 3)$ all represent the real value 8. We call normal a bounded pair such that $\beta \cdot |n| \geq \beta^p$ and subnormal a bounded pair such that $\beta \cdot |n| < \beta^p$ and $e = -e_{min}$. We then easily prove that any bounded pair is equivalent (has the same real value) to a unique canonical pair, either normal or subnormal.

These many representations are sometimes a pain: we lose for example the intuitive uniqueness of the pair corresponding to a value. Anyway, we could work with canonical pairs only and this allows the usual behavior of machine floats. But these multiple representation permit new theorems: L. Théry proved the correctness of an algorithm using two pairs f and g under the assumption that $e_f \leq e_g$. It means that the algorithm is correct if the canonical representations of f and g have this property. It also means that the algorithm is correct if there exist two pairs representing f and g having this property. This is a much stronger result: $f = (5, 3)$ and $g = (1, 3)$ fulfill $e_f \leq e_g$ but their canonical representations do not.

4 Rounding

As many floats may represent the same value, rounding is not defined as a function, but as a relation. A rounding is a relation between a real and a float (given a radix and a format), this rounding property tells that the float is a possible rounding of the real. This can be misleading as we are used to state “let u be the rounding to the nearest of $x + y$ ”. We cannot make people state “let u be one of the possible roundings to the nearest of $x + y$ ”. We therefore decide to define rounding functions. They are also useful as we are able to name the result of an operation: for a given property P , we can express something like $P(\nabla(r))$ instead of $\forall u : \text{float}, \nabla(r, u) \rightarrow P(u)$.

We base these definitions on the fact that \mathbb{R} is an Archimedean field, meaning that there exists a function called up from the reals \mathbb{R} to the signed integers \mathbb{Z} such that: $\forall r \in \mathbb{R}, \text{up}(r) > r$ and $\text{up}(r) - r \leq 1$. We then define $\lfloor \cdot \rfloor$ as $\lfloor r \rfloor = \text{up}(r) - 1$. We then have that $\lfloor r \rfloor \leq r < \lfloor r \rfloor + 1$, we then get some kind of integer rounding down.

Let F be the first normal positive float. More precisely, F only depends on the format: $F = (\beta^{p-1}, -e_{min}) =_{\mathbb{R}} \beta^{p-1-e_{min}}$. We then define the rounding down

of a positive real as

$$\nabla_p(r) = \begin{cases} (\lfloor r \times \beta^{-e} \rfloor, e) & \text{if } r \geq F, \text{ with } e = \lfloor \frac{\ln(r)}{\ln(\beta)} + 1 - p \rfloor \\ (\lfloor r \times \beta^{e_{min}} \rfloor, -e_{min}) & \text{if } r < F. \end{cases}$$

We then prove that, for all $r \geq 0$, $\nabla_p(r)$ is a canonic float (normal in the first case and subnormal in the second case), and is a rounding down of r .

Given these facts, the roundings up and down on any float were easy to define and prove. First, the rounding up of a positive real is:

$$\Delta_p(r) = \begin{cases} \nabla_p(r) & \text{if } r =_{\mathbb{R}} \nabla_p(r) \\ \nabla_p(r)^+ & \text{if not.} \end{cases}$$

where, for any bounded float f , the float f^+ is the successor of f , *i.e.* the smallest bounded float strictly greater than f . There is no definition problem as we have no overflow: all floats have a successor and all reals have a bounded rounding.

$$\nabla(r) = \begin{cases} \nabla_p(r) & \text{if } r \geq 0 \\ -\Delta_p(-r) & \text{if not.} \end{cases} \quad \Delta(r) = \begin{cases} \Delta_p(r) & \text{if } r \geq 0 \\ -\nabla_p(-r) & \text{if not.} \end{cases}$$

For any real r , the float $\Delta(r)$ (resp. $\nabla(r)$) is canonical and a rounding up (resp. down) of r .

For the rounding to the nearest, there are many available choices: when the two floats bracketing the real are at equal distance, both are correct roundings to the nearest. But the IEEE-754 standard defines an even rounding to the nearest, meaning that when this case occurs, the float with the even mantissa must be returned. We define two roundings to the nearest: the first one is the rounding to the nearest up (when the case occurs, the biggest float is returned) and the second is the even rounding. The rounding to the nearest up is defined in the 754R revision of the IEEE-754 standard.

$$\circ_{up}(r) = \begin{cases} \Delta(r) & \text{if } |\Delta(r) - r| \leq |\nabla(r) - r| \\ \nabla(r) & \text{if not.} \end{cases}$$

$$\circ_{even}(r) = \begin{cases} \nabla(r) & \text{if } |\nabla(r) - r| < |\Delta(r) - r| \\ \nabla(r) & \text{if } |\nabla(r) - r| = |\Delta(r) - r| \\ & \text{and the mantissa of } \nabla(r) \text{ is even} \\ \Delta(r) & \text{in the other cases.} \end{cases}$$

This last function is correct because, if $\nabla(r)$ and $\Delta(r)$ are different, then $\Delta(r) = \nabla(r)^+$ and if one has an even mantissa, the other has an odd mantissa. From this fact and the preceding ones, we prove that the results of the last two functions are canonical floats, and that, for all real values r , $\circ_{up}(r)$ is a rounding to the nearest of r and $\circ_{even}(r)$ is the rounding to the nearest even of r .

One may wonder how constructive are these rounding functions. The answer is that they are as constructive as the real numbers are. As we use the standard real numbers (\mathbb{R}) of Coq which are axiomatized [20], we cannot really touch or use these real numbers and their values are not computable. The constructiveness of our functions is directly linked to the constructiveness of the set used for \mathbb{R} . In all our other developments, we did not use any special properties of \mathbb{R} except the fact that it is an ordered field. We here need the Archimedean property, which is an axiom of the standard real numbers of Coq. We could have used a constructive set for \mathbb{R} as in [12] but we prefer to use the standard library to make our development compatible.

5 Definitions: IEEE floats and associated functions

5.1 Basic type

Unlike the preceding sections, we set the radix to 2 from now on. To define the IEEE floats in Coq, we rely on the `Bvector` definitions of the Coq standard library done by J. Duprat. We denote by \bar{v} a vector of bits and $val(\bar{v})$ the integer value corresponding to the vector of bits. This value is obtained using the Coq function `binary_value`.

A format is defined as two integers, the p and the N of the Section 2, that is to say the number of bits allowed for the fraction plus 1 and the number of bits allowed for the unbiased exponent. The IEEE floats are then composed of a sign, a fraction of length $p - 1$ and a unbiased exponent of length N . We then describe the various functions on this type. Some of them are described in Figure 2. We assume from now on that $p \geq 2$ and $N \geq 1$.

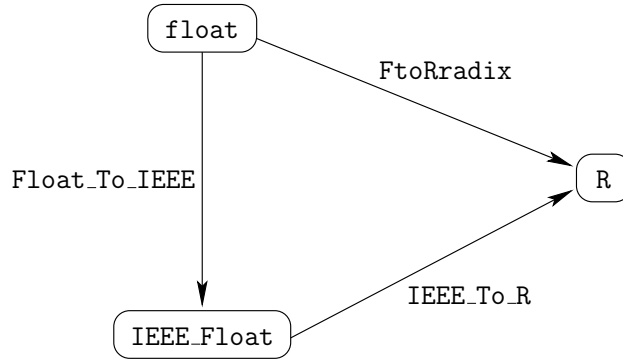


Fig. 2. Pairs/IEEE floats/Real numbers Diagram

5.2 From IEEE Floats to real numbers

We recall that the bias defined by the IEEE-754 standard is $B = 2^{N-1} - 1$. We also use $p - 1$ as the bias corresponding to the fact that the mantissas of our pairs are integers and not fixed-point numbers as in the standard. This transformation was done by multiplying the IEEE mantissa $b \bullet f = b.b_1b_2 \dots b_{p-1}$ by 2^{p-1} . Let us consider an IEEE float x denoted by (s, \bar{m}, \bar{e}) where s is the sign, \bar{m} the fraction of size $p - 1$ and \bar{e} the unbiased exponent of size N .

An IEEE float is normal if his unbiased exponent is neither zero, nor the maximum exponent $2^N - 1$. If x is normal, then its real value is

$$\left(2^{p-1} + \text{val}(\bar{m})\right) 2^{\text{val}(\bar{e}) - B - p + 1}$$

if s is *false* and the opposite of this value if s is *true*.

An IEEE float is subnormal if his unbiased exponent is zero. If x is subnormal, then its real value is

$$\left(\text{val}(\bar{m})\right) 2^{2 - B - p}$$

if s is *false* and the opposite of this value if s is *true*.

We can then define `IEEE_To_R` as a function from `IEEE_Float` to `R`. If the exponent is zero, we compute the value in the subnormal case. If not, we compute in the normal case. But if the exponent is the maximal value, the theoretic value of the float is either *NaN* or $\pm\infty$ and no value of \mathbb{R} can represent it. We decide to construct a function that works only on a part (normal and subnormal) of the possible inputs, but that returns a real. Note that $+0$ and -0 are associated to the same real value 0 .

5.3 From pairs to IEEE floats

The definitions of Section 5.1 exactly correspond to IEEE-754 standard. We then have to write a function `Float_To_IEEE` transforming our pairs into IEEE floats. We use the function `Z_to_binary` that transforms an integer into a binary vector of given size representing it. Of course, this function gives a correct answer only if the integer is positive and not too big. We note \bar{z}^ℓ the result of `Z_to_binary` on the integer z using a vector of size ℓ .

Let f be a pair (n_f, e_f) . Let $\tilde{f} = (n_{\tilde{f}}, e_{\tilde{f}})$ be the canonical pair associated with f . The values of the parts of the IEEE float corresponding to f are given in the following table.

\tilde{f} is	we have	sign	unbiased exponent	fraction
normal	$n_f > 0$	<i>false</i>	$\overline{e_{\tilde{f}} + B + p - 1}^N$	$\overline{n_{\tilde{f}} - 2^{p-1}}^{p-1}$
normal	$n_f < 0$	<i>true</i>	$\overline{e_{\tilde{f}} + B + p - 1}^N$	$\overline{2^{p-1} - n_{\tilde{f}}}^{p-1}$
subnormal	$n_f \geq 0$	<i>false</i>	$\bar{0}^N$	$\overline{n_{\tilde{f}}}^{p-1}$
subnormal	$n_f < 0$	<i>true</i>	$\bar{0}^N$	$\overline{-n_{\tilde{f}}}^{p-1}$

This function always returns $+0$ when the pair has the value zero. As there is only one null pair, there is no way to get a -0 .

5.4 From IEEE floats to pairs

In a symmetrical way, we define a function that transforms a IEEE float x into a pair. Let $x = (s, \bar{m}, \bar{e})$. The following table describe the different cases of `IEEE_To_Float`:

s	$val(\bar{e})$	mantissa	exponent
<i>false</i>	0	$val(\bar{m})$	$2 - B - p$
<i>true</i>	0	$-val(\bar{m})$	$2 - B - p$
<i>false</i>	$\neq 0$	$2^{p-1} + val(\bar{m})$	$val(\bar{e}) - B - p + 1$
<i>true</i>	$\neq 0$	$-2^{p-1} - val(\bar{m})$	$val(\bar{e}) - B - p + 1$

6 Results

6.1 Allowed pairs

We cannot allow any pair to be handled. To be transformable into IEEE floats, the pairs must correspond to machine floats that fit into the floating-point format. It means that the mantissa and the exponent must be in given integer intervals. For an easier use and to increase compatibility with our library, we define this using the very used property `Fbounded` on pairs described in Section 3. We then define the bound \mathcal{B} as (p, e_{min}) where p is the precision of the machine floating-point format and

$$e_{min} = B + p - 2.$$

The smallest positive float is indeed $2^{-e_{min}}$ and is also the machine float with an unbiased exponent worth 0 and a mantissa with all zeros except the last bit which is worth 1. This subnormal float is worth $2^{1-B}0 \bullet 0 \dots 1 = 2^{1-B}2^{1-p}$.

This is not enough as the exponent must have an upper bound. We therefore define the property `InDomain` on a pair f as:

$$f \text{ is bounded by } \mathcal{B} \quad \wedge \quad e_{\tilde{f}} \leq B - p + 1 = 2^{N-1} - p$$

where \tilde{f} is the unique canonical pair associated with f . In fact, this property on the pair f does not bound the exponent of f , but of \tilde{f} . This is the the loosest requirement as we have already proved that the canonical pair has the lowest exponent possible, as far as equivalent bounded pairs are concerned.

This statement is equivalent to $|f| \leq \text{Maxfloat}$ with Maxfloat being the biggest machine floating fitting in the format: $\text{Maxfloat} = (2^p - 1, B - p + 1)$. But the chosen statement is more easy to use afterwards.

6.2 Function

To prove that `Float_To_IEEE` is a function, considering the equivalence property $=_{\mathbb{R}}$, we prove that, if f and g are two pairs sharing the same real value, then they share the same IEEE representation. We prove that in the theorem `Float_To_IEEE_fn`, assuming that f and g are in the domain.

6.3 Same real value

We here want to prove that the diagram of Figure 2 commutes. It means that, given a pair, its real value is the same as if it was first transformed into an IEEE float and then valued.

The main theorem (`float_equals_IEEE`) states that, if a pair f has the above `InDomain` property, then the diagram commutes, i.e.

$$\text{FtoRradix}(f) = \text{IEEE_To_R}(\text{Float_To_IEEE}(f)).$$

6.4 Bijection

To prove that the function `Float_To_IEEE` is a bijection, we define its inverse `IEEE_To_Float` and prove that the diagram of Figure 3 commutes.

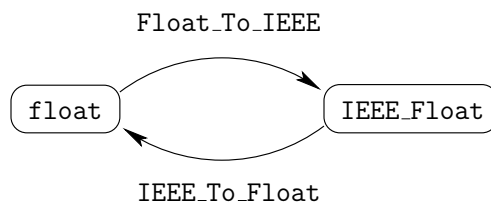


Fig. 3. Pairs/IEEE floats Diagram

We first prove that the result of `IEEE_To_Float` is always a bounded float. We then want to prove that, for any IEEE float, if we apply successively `IEEE_To_Float` and `Float_To_IEEE` to it, we come back to the initial IEEE float. Unfortunately, this is not true for one float: -0 (the IEEE float where the sign is worth 1, the fraction and the unbiased exponent are worth 0). This float is indeed transformed into a pair worth zero, which is transformed into the $+0$ IEEE Float. Except this one, all the other IEEE floats are left untouched, as stated in the `IEEE_To_Float_To_IEEE` theorem:

$$x \neq -0 \quad \Rightarrow \quad \text{Float_To_IEEE}(\text{IEEE_To_Float}(x)) = x.$$

On the other hand, given a pair f that is in the domain (has the `InDomain` above property), if we apply `Float_To_IEEE` and `IEEE_To_Float` to it, we get nearly f : we get the canonical representation associated to f (note that this is Leibniz's equality, not an equality over \mathbb{R}):

$$\text{IEEE_To_Float}(\text{Float_To_IEEE}(f)) = \tilde{f}.$$

7 Conclusion

Getting those two formalizations in one system is interesting as we can mix them: a natural extension of this bit-level representation is the way the processors round their results: this is a 1978 result by Coonen [9]. For the usual operations $(+, -, *, /, \sqrt{})$, 3 bits are enough to guarantee the correct rounding: those bits are called GRS (guard-round-sticky). The certification of this process can only be done using a bit-level representation as those bits do not correspond to an increase in the precision for the sticky bit, but to something more complicated. The high-level certification is more useful to guarantee that a real value is rounded to a given float as the definitions are easy to read. A bit-level definition of roundings is very error-prone compared to a high-level one. Mixing the two representations allows us to express high-level properties of bit-level algorithms.

The obtained results show the equivalence between the floats as defined by the IEEE-754 standard and a subset of the floats used in our formalization. But this equivalence requires that the floats are bounded by a precise bound \mathcal{B} , but also that they have an upper bound on the exponent. Our developments do not consider this case, because it leads to Overflow and exceptional behaviors (creation of *NaN*, traps...). We do not handle those behaviors yet and they can easily be detected *a posteriori*: this is usually not an expected behavior and we will probably have to check that these exceptions do not occur and hardly that the result is a *NaN* or an infinite. This also means increasing the number of uninteresting hypotheses of any lemma.

Even so, proofs are rather long and tedious as each case (normal/subnormal, positive,negative) is to be studied to get inequalities needed to prove that the values put in the vectors actually fit in it and are the good ones: this work is about 2400 lines of Coq long of which about 200 lines are for the formalizations. Moreover, proofs of computations on real values, coercions and casts between Peano integers (\mathbb{N} , `nat`), signed integers (\mathbb{Z} , `Z`) and real numbers (\mathbb{R} , `R`) lack automation [5]. Difficulties were softened by the use of graphical interfaces, namely Pcoq [1] and ProofGeneral [2].

We are getting closer to formalize the IEEE-754 standard in its whole. We lack the special values: *NaN* and $\pm\infty$, but also -0 that we will probably encode as a special value. We are working on a nice and complete way to handle exceptions and traps as required by the IEEE-754 standard. There are in the standard five exceptions defined: Invalid, Division by zero, Overflow, Underflow and Inexact. Any computation that signals an exception can call a trap: the program that has signaled the exception is stopped and another function (possibly user-defined) is executed before the initial program could be restarted. This mechanism requires that any program using floating-point

computations may be stopped any time and that the execution flow may be redirected any time. This means that a computation is not only a function taking the operation, the operands and the rounding mode. It also takes and returns the exceptions signaled and may execute a given or user-defined function depending on the exception(s) signaled.

A natural extension is the formalization of the IEEE-854 standard [7], as this standard defines the floating-point arithmetic with any radix (in fact, only radix-2 and radix-10). It may seem strange, as our initial formalization is radix-independent, to define IEEE floats only with radix 2. This choice is based firstly on the fact that we want to come as near as possible to the IEEE-754 standard and the real hardware floats, which means the radix is 2. More, we want to use only types defined by the Coq standard library and only vectors of bits are available, namely the `Bvector` library of J. Duprat. There is no general vector of bits type with digits between 0 and $\beta - 1$ (or even more general). This is probably due to the fact that general-radix applications are very few (except when the radix is 2^n [21] that is equivalent to groups of digits) and not hardware-implemented.

References

- [1] Ahmed Amerkad, Yves Bertot, Laurence Rideau, and Loïc Pottier. Mathematics and proof presentation in Pcoq. In *Proceedings of Proof Transformation and Presentation and Proof Complexities*, Siena, Italy, 2001.
- [2] David Aspinall. Software and formal methods tools - Proof General: A generic tool for proof development. *Lecture Notes in Computer Science*, 0(1785):38–42, 2000.
- [3] Sylvie Boldo and Marc Daumas. Representable correcting terms for possibly underflowing floating point operations. In Jean-Claude Bajard and Michael Schulte, editors, *Proceedings of the 16th Symposium on Computer Arithmetic*, pages 79–86, Santiago de Compostela, Spain, 2003.
- [4] Sylvie Boldo and Marc Daumas. A simple test qualifying the accuracy of Horner’s rule for polynomials. *Numerical Algorithms*, 2004.
- [5] Sylvie Boldo, Marc Daumas, and Laurent Théry. Formal proofs and computations in finite precision arithmetic. In *Proceedings of the 11th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, Roma, Italy, 2003.
- [6] Victor A. Carreño and Paul S. Miner. Specification of the IEEE-854 floating-point standard in HOL and PVS. In *1995 International Workshop on Higher Order Logic Theorem Proving and its Applications*, Aspen Grove, Utah, 1995. supplemental proceedings.

- [7] William J. Cody, Richard Karpinski, et al. A proposed radix and word-length independent standard for floating point arithmetic. *IEEE Micro*, 4(4):86–100, 1984.
- [8] Robert R. Collins. Inside the Pentium II math bug. *Dr. Dobb's Journal*, 22(8):52, 55–57, 1997.
- [9] Jerome T. Coonen. Specification for a proposed standard for floating point arithmetic. Memorandum ERL M78/72, University of California, Berkeley, 1978.
- [10] Marc Daumas, Laurence Rideau, and Laurent Théry. A generic library of floating-point numbers and its application to exact computing. In *14th International Conference on Theorem Proving in Higher Order Logics*, pages 169–184, Edinburgh, Scotland, 2001.
- [11] Laurent Fousse and Paul Zimmermann. Accurate summation: towards a simpler and formal proof. In *Real Numbers and Computers*, Lyon, France, September 2003.
- [12] Herman Geuvers, Randy Pollack, Freek Wiedijk, and Jan Zwanenburg. A constructive algebraic hierarchy in Coq. *Journal of Symbolic Computation*, 34(4):271–286, 2002.
- [13] David Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1):5–47, 1991.
- [14] John Harrison. A machine-checked theory of floating point arithmetic. In Yves Bertot, Gilles Dowek, André Hirschowitz, Christine Paulin, and Laurent Théry, editors, *12th International Conference on Theorem Proving in Higher Order Logics*, pages 113–130, Nice, France, 1999.
- [15] John Harrison. Formal verification of floating point trigonometric functions. In Warren A. Hunt and Steven D. Johnson, editors, *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pages 217–233, Austin, Texas, 2000.
- [16] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The Coq proof assistant: a tutorial: version 7.2. Technical Report 256, Institut National de Recherche en Informatique et en Automatique, Le Chesnay, France, 2002.
- [17] Christian Jacobi. Formal verification of a theory of IEEE rounding. In *14th International Conference on Theorem Proving in Higher Order Logics*, pages 239–254, Edinburgh, Scotland, 2001. supplemental proceedings.
- [18] Ren-Cang Li, Sylvie Boldo, and Marc Daumas. Theorems on efficient argument reductions. In Jean-Claude Bajard and Michael Schulte, editors, *Proceedings of the 16th Symposium on Computer Arithmetic*, pages 129–136, Santiago de Compostela, Spain, 2003.
- [19] Patrick Loiseleur. Formalisation en Coq de la norme IEEE-754 sur l'arithmétique à virgule flottante. Master's thesis, École Normale Supérieure, Paris, France, 1997.

- [20] Micaela Mayero. *Formalisation et automatisisation de preuves en analyses réelle et numérique*. PhD thesis, Université Paris VI, décembre 2001.
- [21] Jean-Michel Muller. *Elementary functions, algorithms and implementation*. Birkhauser, 1997.
- [22] John Rushby and Friedrich von Henke. Formal verification of algorithms for critical systems. In *Proceedings of the Conference on Software for Critical Systems*, pages 1–15, New Orleans, Louisiana, 1991.
- [23] David 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.
- [24] David Stevenson et al. A proposed standard for binary floating point arithmetic. *IEEE Computer*, 14(3):51–62, 1981.
- [25] David Stevenson et al. An American national standard: IEEE standard for binary floating point arithmetic. *ACM SIGPLAN Notices*, 22(2):9–25, 1987.