NASA/CR-2006-214298
NIA Report No. 2006-01

A High-Level Formalization of Floating-Point Numbers in PVS

Sylvie Boldo
Ecole Normale Superieure de Lyon, France

Cesar Munoz
National Institute of Aerospace, Hampton, Virginia

October 2006

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results ... even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at (301) 621-0134

- Phone the NASA STI Help Desk at (301) 621-0390

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/CR-2006-214298
NIA Report No. 2006-01

A High-Level Formalization of Floating-Point Numbers in PVS

Sylvie Boldo
Ecole Normale Superieure de Lyon, France

Cesar Munoz
National Institute of Aerospace, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NCC1-02043

October 2006

Available from:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

A HIGH-LEVEL FORMALIZATION OF FLOATING-POINT NUMBERS IN PVS*

Sylvie Boldo† and César Muñoz ‡

ABSTRACT

We develop a formalization of floating-point numbers in PVS based on a well-known formalization in Coq. We first describe the definitions of all the needed notions, e.g., floating-point number, format, rounding modes, etc. Then, we present an application to polynomial evaluation for elementary function evaluation. The application already existed in Coq, but our formalization shows a clear improvement in the quality of the result due to the automation provided by PVS. Finally, we integrate our formalization into a PVS hardware-level formalization of the IEEE-854 standard previously developed at NASA.

1 INTRODUCTION

Floating-point numbers are the internal representation of real numbers used by most general-purpose processors. Floating-point arithmetic is described by the IEEE-754 [22, 23] and IEEE-854 standards [8]. These standards define the format, rounding modes, and operations that can be performed on floating-point numbers. For more information on floating-point numbers and numerical computation, see [13, 15, 18, 24].

The correctness of floating-point computations is critical to engineering applications (see, for example, the Pentium Bug [9]). For that reason, floating-point arithmetic is an active subject of research in the formal methods community. Formal techniques have been successfully applied, both for hardware-level verification (AMD, Intel) and high-level algorithms (evaluation of the exponential) in a variety of proof assistants and model-checkers [1, 6, 7, 14, 17, 20].

The work presented in this report is based on the formalization of floating-point numbers in Coq by Daumas, Rideau, and Théry described in [11]. That formalization has been thoroughly used and it forms the kernel of the Coq's library on floating-point arithmetic (`http://lipforge.ens-lyon.fr/projects/pff`). It is especially useful when dealing with high-level algorithms [3] because it does not consider the machine-level array of bits, but only a representation of floating-point numbers by integer numbers that are more easily handled by a person or a proof assistant.

In this report, we describe the port of the floating-point arithmetic formalization from Coq [2, 10] to PVS [19]. The rest of this paper is organized as follows. Section 2 defines the basic concepts. The rounding modes are presented in Section 3. Section 4 states the

*This work was done while the first author was visiting the National Institute of Aerospace, supported by the National Aeronautics and Space Administration, Langley Research Center under the Research Cooperative Agreement No. NCC-1-02043, and by the French National Center for Scientific Research, CNRS PICS No. 2533.

†École Normale Supérieure de Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France. Email: `sylvie.boldo@ens-lyon.fr`, Web: `http://perso.ens-lyon.fr/sylvie.boldo/`.

‡National Institute of Aerospace (NIA), 100 Exploration Way, Hampton VA, 23666. Email: `munoz@nianet.org`, Web: `http://research.nianet.org/~munoz`.

fundamental properties of floating-point numbers. Section 5 illustrates the application of the formalization to polynomial evaluation. Section 6 shows the integration of the high-level formalization to a hardware-level specification of the IEEE-854 standard developed at NASA [17]. We give conclusions and perspectives in Section 7.

2 FLOATING-POINT NUMBERS

Following the definition in [11], a floating-point number is represented by a pair of integers, e.g., the radix-2 floating-point number 1.001E1 is represented as $(9, -2)$, i.e., $1.001\text{E}1_2 = 9 \times 2^{-2}$. Henceforth, we take the names used in the current revision of the IEEE-754 standard[1]. The left part of a float is called the *significand* and the right part is the *exponent*. Note that the exponent is shifted compared to the exponent of the IEEE machine number. In PVS, we use a record with two fields, `Fnum` and `Fexp`, that correspond to the significand and the exponent, respectively.

```
float: TYPE = [# Fnum:int, Fexp:int #]
```

The radix is defined as 2 in the IEEE-754 standard and can be either 2 or 10 in the IEEE-854 standard. In this formalization, the radix β (`radix`, in PVS) is a parameter of the specification and it is declared as an integer greater than 1. Therefore, a float can be interpreted as a real value as follows:

$$(n, e) \in \mathbb{Z}^2 \quad \hookrightarrow \quad n \times \beta^e \in \mathbb{R}$$

```
FtoR(f):real = Fnum(f)*radix^(Fexp(f))
CONVERSION FtoR
```

Note that we declare FtoR as a *conversion*. This way, elements of the type `float` are automatically converted into real numbers when needed. We also define some basic operations on floats, e.g., `Fabs(f)` is a float such that its real value is the absolute value of the real value of `f` and `Fopp(f)` is a float having the negative value of `f`.

```
Fabs(f):float = (# Fnum:=abs(Fnum(f)), Fexp:=Fexp(f) #)
Fopp(f):float = (# Fnum:=-(Fnum(f)), Fexp:=Fexp(f) #)

FoppCorrect : lemma Fopp(f)=-f
FabsCorrect : lemma Fabs(f)=abs(f)
```

2.1 Bounded Floats

The type `float` represents an infinite number of numbers and only a finite number of these can be represented as machine floating-point numbers. We have to restrict this type to the numbers that fit in a given floating-point format. A floating-point format (typically IEEE single or double precision) is a pair of integers (p, E). The integer p is called the *precision* of the floating-point format and E is the *minimal exponent*. For example, the IEEE double

[1] See `http://754r.ucbtest.org/` for drafts and minutes.

precision is specified by the pair $(53, 1074)$ and the single precision is specified by the pair $(24, 149)$. For a given format (p, E), we say that a float (n, e) is *bounded* if and only if

$$|n| < \beta^p, \quad \text{and} \tag{1}$$
$$-E \leq e. \tag{2}$$

In PVS, a format is a record with fields `Prec` and `dExp` that correspond to p and E, respectively.

```
Format: TYPE = [# Prec:above(1), dExp:nat #]

vNum(b:Format):posnat = radix^Prec(b)

Fbounded?(b)(f):bool = abs(Fnum(f)) < vNum(b) AND -dExp(b) <= Fexp(f)
```

The lower bound on the exponent is needed as it creates subnormal numbers, whose behavior is often unexpected. In this formalization, we do not consider overflows and we argue that they can be handled at a higher specification level. Overflows create infinities and values that do not represent a real number (NaN), but they are usually propagated until the end of the computation. Therefore, overflows are more easily detected than underflows as subnormal numbers are silent even when the loss of accuracy is huge.

2.2 Canonical Floats

The chosen representation of floats in this formalization is redundant, i.e., several floats may have the same real value. This is true even if the floats are bounded. For example, using radix 2 and 4 bits of precision, the floats $(8, 0)$, $(4, 1)$, $(2, 2)$ and $(1, 3)$ are all bounded and have the real value 8. The sets of floats that share the same real value are called a *cohort*.

In order to represent IEEE machine floating-point numbers, which by definition are unique, we have to define a canonical set of floats. A *canonical float* is a float that is either normal or subnormal. A *normal float* is a float such that its significand cannot be multiplied by the radix and still fit in the format. This means that the first digit of the significand, represented in base β, is nonzero. A *subnormal float* is a float having the minimal exponent such that its significand could be multiplied by the radix and still fit in the format.

```
Fnormal?(b)(f):bool = Fbounded?(b)(f) AND vNum(b)<=abs(radix*Fnum(f))

Fsubnormal?(b)(f):bool = Fbounded?(b)(f) AND Fexp(f)=-dExp(b)AND
                         abs(radix*Fnum(f)) < vNum(b)

Fcanonic?(b)(f):bool = Fnormal?(b)(f) OR Fsubnormal?(b)(f)
```

By definition, normal and subnormal floats are disjoint. Subnormal floats are the smallest representable floats (in absolute value) and their characteristics are very different from the normal floats. They may produce surprising numerical results due to their uncommon characteristics.

We prove that canonical floats are unique: if two floats are canonical and have the same real value, then they are identical. Then, we prove that any bounded float has a canonical representation obtained by applying the function `Fnormalize`. We take advantage of PVS sub-typing to guarantee that for all bounded float `f` and format `b`, `Fnormalize(b)(f)` is a canonical float and equal to `f` (in real value).

```
FcanonicUnique: lemma
    Fcanonic?(b)(p) AND Fcanonic?(b)(q) AND FtoR(p)=FtoR(q)
    => p=q

Fnormalize(b)(f:(Fbounded?(b))): recursive
{x : (Fcanonic?(b)) | FtoR(x)=FtoR(f) AND Fexp(x) <=  Fexp(f)} =
   if Fnum(f) = 0 then
     (# Fnum:=0, Fexp:= -dExp(b)#)
   elsif Fexp(f) = -dExp(b) or
                    abs(radix*Fnum(f)) >= vNum(b) then f
   else Fnormalize(b)((# Fnum:=radix*Fnum(f), Fexp:=Fexp(f)-1 #))
   endif
   measure vNum(b) - abs(Fnum(f))
```

2.3 Ulp

The *unit in the last place (ulp)* is the value of the least significand digit of the representation of the float. It is also the increment to add to a positive float to get the successor of the float. Here, as we handle significands, this value is the radix to the power of the exponent, if the float is canonical.

```
Fulp(b)(f:(Fbounded?(b))):real = radix^(Fexp(Fnormalize(b)(f)))
```

The ulp is generally used as a measure of the error made during a computation. Note that there is a major difference between normal and subnormal floats when considering the ulp of a float: for normal floats, we have $\text{ulp}(f) \le \beta^{1-p}|f|$. In this case, $\text{ulp}(f) \ll |f|$. For subnormal floats, the ulp is always β^{-E}. In particular, $\text{ulp}(\beta^{-E}) = \beta^{-E}$, i.e., the ulp can be as big as the real value of the float. We prove that

$$\text{ulp}(f) \le \max\left(\beta^{1-p}|f|, \beta^{-E}\right).$$

```
FulpLe : lemma
  Fbounded?(b)(p)
  => Fulp(b)(p) <= max(abs(p) * radix/vNum(b), radix^(-dExp(b)))
```

2.4 Predecessor and Successor

The predecessor of a given float f is the greatest float strictly less than f. The successor of a given float f is the smallest float strictly greater than f.

```
Fsucc(b)(f):float = IF Fnum(f)=vNum(b)-1
    THEN (# Fnum:=vNum(b)/radix, Fexp:=Fexp(f)+1 #)
    ELSIF  Fnum(f)=-vNum(b)/radix AND Fexp(f)>-dExp(b)
         THEN  (# Fnum:=-(vNum(b)-1), Fexp:=Fexp(f)-1 #)
         ELSE  (# Fnum:=Fnum(f)+1, Fexp:=Fexp(f) #)
    ENDIF

Fpred(b)(f):float = IF Fnum(f)=-(vNum(b)-1)
    THEN (# Fnum:=-vNum(b)/radix, Fexp:=Fexp(f)+1 #)
    ELSIF  Fnum(f)=vNum(b)/radix AND Fexp(f)>-dExp(b)
         THEN  (# Fnum:=vNum(b)-1, Fexp:=Fexp(f)-1 #)
         ELSE  (# Fnum:=Fnum(f)-1, Fexp:=Fexp(f) #)
    ENDIF
    ENDIF
```

We prove several useful properties of these functions. For example, the opposite of the successor is the predecessor of the opposite (`FpredFoppFsucc`).

```
FpredFoppFsucc: lemma Fpred(b)(Fopp(f))=Fopp(Fsucc(b)(f))
FsuccFoppFpred: lemma Fsucc(b)(Fopp(f))=Fopp(Fpred(b)(f))
FpredLt       : lemma Fpred(b)(f) < f

FsuccFpred  : lemma Fcanonic?(b)(f) => Fsucc(b)(Fpred(b)(f))=f
FpredCanonic: lemma Fcanonic?(b)(f) => Fcanonic?(b)(Fpred(b)(f))
FpredPos    : lemma Fcanonic?(b)(p) AND 0 < p => 0 <= Fpred(b)(p)

FpredDiff: lemma
  Fcanonic?(b)(f) AND 0 < f
    => f-Fpred(b)(f)=Fulp(b)(Fpred(b)(f))

FpredProp: lemma
  Fcanonic?(b)(p) AND Fcanonic?(b)(q) AND p < q
  => p <= Fpred(b)(q)
```

3 ROUNDING MODES

Floating-point operations in the IEEE standards are defined such that the result is the same as if the operation is computed with infinite precision and then rounded to the destination format. Hence, instead of a direct definition of *floating-point addition* $\oplus$, we define a *rounding mode* $\circ$ over real expressions, and from there, $f \oplus g$ can be defined based on $\circ(f+g)$. In practice, there are several possible definitions of the rounding operation $\circ$. For instance, the *rounding toward* $-\infty$ (`isMin?`, in PVS) is the biggest floating-point number whose value is smaller than the real number. Similarly, the *rounding toward* $+\infty$ (`isMax?`, in PVS) is the smallest bounded floating-point number whose value is bigger than the real number.

As several floats may represent the same floating-point number, we define the rounding operation as a relation between real numbers and bounded floats, rather than a function from real numbers to bounded floats.

```
RND : TYPE = [b:Format -> [[real,(Fbounded?(b))]->bool]]

isMin?(b)(r:real,min:(Fbounded?(b))):bool =
  min <= r AND
  forall (f:(Fbounded?(b))): f <= r => f <= min

isMax?(b)(r:real,max:(Fbounded?(b))):bool =
  r <= max AND
  forall (f:(Fbounded?(b))): r <= f => max <= f
```

Note that we do not explain (yet) how to compute these rounding modes, we just state the properties that they satisfy. Furthermore, we say that a rounding mode is *well-defined* if it is

- *total*, i.e., all reals can be rounded,
- *compatible*, i.e., if two floats have the same real value and one is a rounding of a real value, then the other one is too,
- *minormax*, i.e., each rounding is either the rounding toward $+\infty$ or $-\infty$ of the real number, and
- *monotone*, i.e., non-decreasing.

Some rounding modes are moreover *unique*, i.e., each real number has only one rounding, which may have a cohort of representations.

```
P : VAR RND
Total?(b)(P):bool = forall (r:real):
   exists (f:(Fbounded?(b))): P(b)(r,f)

Compatible?(b)(P):bool = forall (r1,r2:real, f1,f2:(Fbounded?(b))):
   P(b)(r1,f1) AND r1=r2 AND FtoR(f1)=FtoR(f2)
   => P(b)(r2,f2)

MinOrMax?(b)(P):bool = forall (r:real,f:(Fbounded?(b))):
   P(b)(r,f) => isMin?(b)(r,f) OR isMax?(b)(r,f)

Monotone?(b)(P):bool = forall (r1,r2:real, f1,f2:(Fbounded?(b))):
   r1 < r2 AND P(b)(r1,f1) AND P(b)(r2,f2)
   => f1 <= f2

RoundedMode?(b)(P):bool =
   Total?(b)(P) AND Compatible?(b)(P) AND
   MinOrMax?(b)(P) AND Monotone?(b)(P)

Unique?(b)(P):bool = forall (r:real,f1,f2:(Fbounded?(b))):
   P(b)(r,f1) AND P(b)(r,f2)
   => FtoR(f1)=FtoR(f2)
```

3.1 IEEE Rounding Modes

Originally, the IEEE standards defined 4 rounding modes, but a fifth one has been added in the revision of the IEEE-754 standard. We have already defined the rounding toward $\pm\infty$, we now add all the other rounding modes defined by the IEEE-754 standard and its revision. The *nearest* rounding mode yields the float that is nearer to the real value. Note that this rounding is not unique when the real number is exactly in the middle of two floats. The revision of the IEEE-754 standard defines two unique rounding modes to the nearest: an *even* rounding mode, which when in the middle, chooses the float having an even significand, and an *away from zero* rounding mode, which when in the middle, chooses the one with the greater absolute value.

```
ToZero?(b)(r:real,c:(Fbounded?(b))):bool =
    if 0 <= r then isMin?(b)(r,c)
              else isMax?(b)(r,c) endif

Nearest?(b)(r:real,c:(Fbounded?(b))):bool =
  (forall (f:(Fbounded?(b))): abs(c-r) <= abs(f-r))

EvenNearest?(b)(r:real,c:(Fbounded?(b))):bool =  Nearest?(b)(r,c) AND
  (even?(Fnum(Fnormalize(b)(c))) OR
      (forall (f:(Fbounded?(b))): Nearest?(b)(r,f) => FtoR(f)=FtoR(c)))

AFZNearest?(b)(r:real,c:(Fbounded?(b))):bool =  Nearest?(b)(r,c) AND
  (abs(r) <= abs(c) OR
      (forall (f:(Fbounded?(b))): Nearest?(b)(r,f) => FtoR(f)=FtoR(c)))
```

We prove that these rounding modes are well-defined and, except for the nearest rounding mode, that they are unique.

```
isMin_RoundedMode      : lemma RoundedMode?(b)(isMin?)
isMax_RoundedMode      : lemma RoundedMode?(b)(isMax?)
ToZero_RoundedMode     : lemma RoundedMode?(b)(ToZero?)
Nearest_RoundedMode    : lemma RoundedMode?(b)(Nearest?)
EvenNearest_RoundedMode: lemma RoundedMode?(b)(EvenNearest?)
AFZNearest_RoundedMode : lemma RoundedMode?(b)(AFZNearest?)

isMin_Unique      : lemma Unique?(b)(isMin?)
isMax_Unique      : lemma Unique?(b)(isMax?)
ToZero_Unique     : lemma Unique?(b)(ToZero?)
EvenNearest_Unique: lemma Unique?(b)(EvenNearest?)
AFZNearest_Unique : lemma Unique?(b)(AFZNearest?)
```

Finally, we provide functional specifications of the toward $\pm\infty$ and even nearest rounding modes, and prove that they are correct.

```
RND_aux(b)(x:nonneg_real): (Fcanonic?(b)) =
  if (x < radix^(-dExp(b)-1)*vNum(b))
     then (# Fnum:=floor(x*radix^(dExp(b))), Fexp:=-dExp(b) #)
     else let e=floor(ln(x*radix/vNum(b))/ln(radix)) in
          (# Fnum:=floor(x*radix^(-e)), Fexp:=e #)
  endif

RND_Min(b)(x:real): (Fcanonic?(b)) =
  if (0 <= x)
    then RND_aux(b)(x)
    elsif Fopp(RND_aux(b)(-x))=x then Fopp(RND_aux(b)(-x))
    else  Fpred(b)(Fopp(RND_aux(b)(-x)))
  endif

RND_Max(b)(x:real): (Fcanonic?(b)) = Fopp(RND_Min(b)(-x))

RND_ENearest(b)(x:real): (Fcanonic?(b)) =
 if     abs(RND_Min(b)(x)-x) < abs(RND_Max(b)(x)-x) then RND_Min(b)(x)
  elsif abs(RND_Max(b)(x)-x) < abs(RND_Min(b)(x)-x) then RND_Max(b)(x)
  elsif RND_Min(b)(x)=RND_Max(b)(x)::real then RND_Min(b)(x)
  elsif even?(Fnum(RND_Min(b)(x))) then RND_Min(b)(x)
  else  RND_Max(b)(x)
 endif

RND_Min_isMin          : lemma isMin?(b)(r,RND_Min(b)(r))
RND_Max_isMax          : lemma isMax?(b)(r,RND_Max(b)(r))
RND_ENearest_isEnearest: lemma EvenNearest?(b)(r,RND_ENearest(b)(r))
```

Note that the IEEE standards only require correct rounding for $+$, $-$, $\times$, $/$, $\sqrt{}$, and for the fused multiply-and-add (FMA): $a \times b + c$ with only one rounding in the revision of the IEEE-754 standard. Therefore, although these rounding modes can be used to round any real number, e.g., exp(2), there is no guarantee that the result is the same as the floating-point computation of exp(2) on a particular processor.

3.2 Properties of Rounding Modes

Here are some basic and well-known properties about rounding modes. Even if our definition of rounding modes is uncommon, we can easily prove these properties.

A useful property of the rounding modes concerns the rounding of opposite numbers: the rounding down of r is the opposite of the rounding up of $-r$.

```
MinOppMax  : lemma
  Fbounded?(b)(p) AND isMin?(b)(r,p)
  => isMax?(b)(-r,Fopp(p))

MaxOppMin  : lemma
  Fbounded?(b)(p) AND isMax?(b)(r,p)
  => isMin?(b)(-r,Fopp(p))

NearestFopp: lemma
  Fbounded?(b)(p) AND Nearest?(b)(r,p)
  => Nearest?(b)(-r,Fopp(p))

NearestFabs: lemma
  Fbounded?(b)(p) AND Nearest?(b)(r,p)
  => Nearest?(b)(abs(r),Fabs(p))
```

Another useful property is the fact that the sign of a real number is preserved by any rounding mode: a non-negative real is always rounded into a non-negative float.

```
RleRoundedR0 : lemma
  Fbounded?(b)(f) AND RoundedMode?(b)(P) AND P(b)(r,f) AND 0 <= r
  => 0 <= f

RleRoundedLessR0 : lemma
  Fbounded?(b)(f) AND RoundedMode?(b)(P) AND P(b)(r,f) AND r <= 0
  => f <= 0
```

Moreover, a bounded float is always rounded to itself.

```
RoundedProjectorEq : lemma
  Fbounded?(b)(f) AND Fbounded?(b)(p) AND RoundedMode?(b)(P) AND P(b)(f,p)
  => FtoR(p)=FtoR(f)

RoundedProjector : lemma
  Fbounded?(b)(f) AND RoundedMode?(b)(P)
  => P(b)(f,f)
```

4 FUNDAMENTAL PROPERTIES

4.1 Round-off Errors

The round-off error is the difference between the real value and its rounding. It is usually described in terms of the ulp (Section 2.3). We prove that for any rounding mode, this difference is strictly less than one ulp. Furthermore, for any rounding mode to the nearest, this difference is less than or equal to half the ulp:

```
RoundedModeUlp : lemma
  Fbounded?(b)(p) AND RoundedMode?(b)(P) AND P(b)(r,p)
  => abs(p-r) < Fulp(b)(p)

NearestUlp : lemma
  Fbounded?(b)(p) AND Nearest?(b)(r,p)
  => abs(p-r) <= Fulp(b)(p)/2
```

4.2 Canonical Floats

The canonical representation is the one having the smallest exponent of the cohort. This is a new and unexpected property as the notion of cohort was defined by the commission revising the IEEE-754 standard.

```
CanonicLeastExp: lemma
   Fcanonic?(b)(p) AND Fbounded?(b)(q) AND FtoR(p)=FtoR(q)
   => Fexp(p) <= Fexp(q)
```

4.3 Lexicographical Order

This property states that given two nonnegative IEEE floating-point numbers f and g, f is smaller than g if the string of bits representing f is less, in lexicographical order, than the string of bits representing g. In our formalization, we express that property as the fact that the real value and the exponent of two positive floats are in the same order relation.

```
Lexico: lemma
   Fcanonic?(b)(p) AND Fcanonic?(b)(q) AND 0 <= p AND p <= q
   => Fexp(p) <= Fexp(q)
```

4.4 Exact Subtraction

This property has been known for decades: it can be found in [21] but its paternity may be due to W. Kahan. This theorem gives sufficient conditions for a subtraction to be exact. The theorem states that if p and q are bounded floats such that $\frac{p}{2} \leq q \leq 2\,p$, then the float `Fminus(q,p)`, which has the value $q-p$, is bounded.

```
Sterbenz : theorem
 Fbounded?(b)(p) AND Fbounded?(b)(q) AND p/2 <= q AND q <= 2*p
 => Fbounded?(b)(Fminus(q,p))
```

By Lemma `RoundedProjector` (Section 3.2), a bounded float is exactly rounded. Therefore, the computation $\circ(q-p)$ is correct for any rounding mode $\circ$. Note that we have here exhibited a bounded float equal to $q-p$, which is not necessarily canonical (we can always normalize it afterward if needed). The way this lemma is stated makes it easy to use it: instead of "there exists a bounded float such that ...", it gives a particular float that is bounded.

4.5 Representable Errors

It has been known since the 1970's that the error of a floating-point addition (when rounding to the nearest) or of a floating-point multiplication fits in a floating-point number of the same format [12,16]. We give necessary and sufficient conditions for this error to be representable, even when underflow occurs [4]. Furthermore, we compute the exponent of the exhibited bounded float that represents the error term.

```
errorBoundedPlus : lemma
   Fbounded?(b)(p) AND Fbounded?(b)(q) AND Fbounded?(b)(f) AND
   Nearest?(b)(p+q,f)
   => (exists (e:(Fbounded?(b))): e=p+q-f AND
       Fexp(e)=min(Fexp(p),Fexp(q)))

errorBoundedMult : lemma
   Fbounded?(b)(p) AND Fbounded?(b)(q) AND Fbounded?(b)(f) AND
   RoundedMode?(b)(P) AND P(b)(p*q,f) AND -dExp(b) <= Fexp(p)+Fexp(q)
   => (exists (e:(Fbounded?(b))): e=p*q-f AND Fexp(e)=Fexp(p)+Fexp(q))
```

5 POLYNOMIAL EVALUATION

In this section, we present an application of our formalization to polynomial evaluation. This application was originally developed in Coq [5]. Due to the powerful automation features provided by PVS, the results presented here are significantly better than the original ones.

When computing a polynomial evaluation using Horner's rule after an argument reduction, the last step usually creates the biggest error in the final result. For example, for the evaluation of the exponential, we compute $1 + x + \frac{x^2}{2} + \ldots$ with $|x| \leq \frac{\ln(2)}{2} \ll 1$. The errors in computing $\frac{x^2}{2} + \ldots$ are negligible compared to the final result whose value is about 1.

Therefore, we need to accurately compute expressions of the form $a \times x + y$, where a, x and y represent approximations of the ideal real values a', x' and y'. An exact rounding is impossible to guarantee. However, we will describe and prove that a faithful rounding can still be obtained.

5.1 Faithful Computations

A *faithful computation* is a relation between a real number and the float numbers that are either the rounding up or the rounding down of the real value.

```
MinOrMax?(r:real,f:(Fbounded?(b))):bool=
   isMin?(b)(r,f) OR isMax?(b)(r,f)
```

We can prove the following sufficient conditions for a given computation to be faithful (see [5] for more details).

```
MinOrMax1 : lemma
  Fcanonic?(b)(f) AND 0 < f AND abs(f-z) < Fulp(b)(Fpred(b)(f))
  => MinOrMax?(z,f)

MinOrMax2 : lemma
  Fcanonic?(b)(f) AND 0 < f AND abs(f-z) < Fulp(b)(f) AND f <= z
  => MinOrMax?(z,f)
```

5.2 Round-off Error

The following theorems allow us to bound a float with the real values it rounds. These bounds are better than the ones presented in [5].

If $f = \circ(r)$ is canonical and non-zero, then

$$\frac{|r|}{1 + \frac{1}{2 \times |n_f|}} \leq |f| \leq \frac{|r|}{1 - \frac{1}{2 \times |n_f|}}.$$

Note that the bounds above do not require f to be normal. If f is known to be normal, we can deduce that

$$\frac{|r|}{1 + \frac{\beta^{p-1}}{2}} \leq |f| \leq \frac{|r|}{1 - \frac{\beta^{p-1}}{2}}.$$

```
RoundLe : lemma
  Fcanonic?(b)(f) AND f /= 0 AND Nearest?(b)(z,f)
  => abs(f) <= abs(z)/(1-1/(2*abs(Fnum(f))))

RoundGe : lemma
  Fcanonic?(b)(f) AND f /= 0 AND Nearest?(b)(z,f)
  => abs(z)/(1+1/(2*abs(Fnum(f)))) <= abs(f)
```

The next theorem allows us to handle the case where p is near a power of the radix. In this case, the ulp of its predecessor is twice smaller and the preceding theorems are not good enough. This theorem states that even in this case, the rounding to the nearest is closer to the real value than its predecessor and this distance can be expressed with the ulp of the predecessor.

```
NearestUlp2 : lemma
    Fcanonic?(b)(p) AND Nearest?(b)(r,p) AND
    abs(r) <= abs(p) + Fulp(b)(Fpred(b)(Fabs(p)))/2
    => abs(p-r) <= Fulp(b)(Fpred(b)(Fabs(p)))/2
```

5.3 Sufficient Conditions

In order to compute $a \times x + y$, we first compute $t = \circ(a \times x)$ and, then, $u = \circ(t + y)$, where $\circ$ is the rounding to the nearest. In this section, we provide sufficient conditions on a, x, y, a', x', y' for these computations to be faithful.

The lemmas in the previous sections allow us to prove the following result.

```
AxpyPos : lemma
  Nearest?(b)(a*x,t) AND Nearest?(b)(t+y,u) AND 0 < u AND
 (Fnormal?(b)(t) => radix*abs(t) <= Fpred(b)(u)) AND
 abs(y1-y+a1*x1-a*x) < Fulp(b)(Fpred(b)(u))/4
 => MinOrMax?(y1+a1*x1,u)
```

Unfortunately, we do not have a priori the outputs u and t to check for the sufficient conditions on the lemma AxpyPos. The following lemma provides conditions that can be checked a priori and without knowing the argument.

```
Axpy_opt : lemma
  Nearest?(b)(a*x,t) AND Nearest?(b)(t+y,u) AND Prec(b) >= 6 AND
  (radix+1+radix^(4-Prec(b)))*abs(a*x) <= abs(y) AND
  abs(y1-y+a1*x1-a*x) < abs(y)*radix^(1-Prec(b))/(6*radix)
  => MinOrMax?(y1+a1*x1,u)
```

The last theorem corresponds to the more usual case: if the radix is 2 and the precision greater or equal to 24 (single precision) then the conditions

$$3.000001\ |a \times x| \leq |y| \quad \text{and} \quad |y' - y + a' \times x' - a \times x| < \frac{|y| \times 2^{1-p}}{12}$$

are sufficient to guarantee that $u = \circ(y + \circ(a \times x))$ is a faithful computation of the exact real value $y' + a' \times x'$.

```
Axpy_simpl : lemma
  Nearest?(b)(a*x,t) AND Nearest?(b)(t+y,u) AND
  Prec(b) >= 24 AND radix = 2 AND (3+1/100000)*abs(a*x) <= abs(y) AND
  abs(y1-y+a1*x1-a*x) < abs(y)*2^(1-Prec(b))/12
  => MinOrMax?(y1+a1*x1,u)
```

6 INTEGRATION INTO IEEE-854 SPECIFICATION

In 1995, Paul Miner and Víctor Carreño formalized the IEEE-854 standard in PVS and in HOL [6,7,17]. This is a complete hardware-level specification of the IEEE-854 standard that represents significands as vectors of bits (or digits). In this sense, that formalization is more detailed than ours. For example, it specifies *overflow* and assumes that the radix is either 2 or 10. However, it does not provide any of the high-level properties of our formalization.

After updating the IEEE-854 formalization to PVS3.2, we integrate our development into it to combine the strength of both works. First, we define a PVS theory with the same parameters and the same hypotheses as the ones in the IEEE-854 formalization (see [17] for more details).

```
IEEE_link[radix, p: above(1), alpha, E_max, E_min: integer]: THEORY
 ASSUMING
   Base_values: ASSUMPTION radix = 2 OR radix = 10
   Exponent_range: ASSUMPTION (E_max - E_min) / p > 5
     ...
 ENDASSUMING
```

Then, we define a `Format` to be used in our formalization.

```
b: Format = (# Prec := p, dExp := -E_min + p - 1 #)
```

Finally, we define the functions that map floating-point numbers from one representation to the other.

```
ieee : VAR (finite?)
f    : VAR float

IEEE_to_float(ieee): {x: (Fbounded?(b)) |
  value(ieee) = x:: real AND abs(x) < radix^(E_max+1)} =
    (# Fnum := (-1) ^ sign(ieee) * radix ^ (p - 1) *
                 Sum(p, value_digit(d(ieee))),
       Fexp := Exp(ieee) + 1 - p #)

float_to_IEEE(f: (Fcanonic?(b)) | abs(f) < radix^(E_max+1)):
      {x: (finite?) | f = value(x)} =
    finite(sign_of(Fnum(f)), Fexp(f) + p - 1,
           (LAMBDA (i: below(p)):
              mod(floor(radix ^ (i+1-p) * abs(Fnum(f))), radix)))
```

Note the type constraints on the inputs and outputs of these functions. For example, to be transformed into an IEEE float, our floats must not overflow, i.e., their value must be less that $\beta^{E_{max}+1}$.

6.1 Properties

We now can use those functions to prove properties on one formalization using the results of the other one. For example, we prove that a bounded, non-zero, non-overflowing float has a value between `min_pos`$= \beta^{E_{min}+1-p}$ and `max_pos`$= \beta^{E_{max}+1} - \beta^{E_{max}+1-p}$.

```
value_nonzero_bis: lemma
  Fbounded?(b)(f) AND abs(f) < radix^(E_max+1) AND f /= 0
  => min_pos <= abs(f) AND abs(f) <= max_pos
```

More interesting is the exact subtraction theorem (Section 4.4) using IEEE-854 numbers.

```
ieee,ieee2: VAR (finite?)
Sterbenz_bis : lemma
   value(ieee)/2 <= value(ieee2) AND  value(ieee2) <= 2*value(ieee)
   =>  (exists (s:(finite?)): value(s)=value(ieee2)-value(ieee))
```

In this case, there is no need for additional hypotheses: we guarantee that overflow cannot occur if the inputs are regular IEEE floating-point numbers (`finite?`, meaning neither a `NaN`, nor an infinity).

6.2 Rounding Modes

The rounding modes of the IEEE-854 formalization are the rounding up, down, toward zero, and to the nearest with the even tie breaking rule. We prove that, for any real number `r`, the result of rounding `r` using one of those rounding modes is the same as rounding `r` using the corresponding rounding mode in our formalization (Section 4). In other words, we prove that our rounding modes are coherent with the IEEE-854 rounding modes. Of course, we have to add the hypothesis that there is no overflow.

```
Roundings_eq_1: lemma
  NOT trap_enabled?(underflow(FALSE)) AND
  max_neg <= r AND r < radix ^ (E_max + 1)
  => fp_round(r, to_neg) = FtoR[radix](RND_Min(b)(r))

Roundings_eq_2: lemma
  NOT trap_enabled?(underflow(FALSE)) AND
  - radix ^ (E_max + 1) < r AND r <= max_pos
  => fp_round(r, to_pos) = FtoR[radix](RND_Max(b)(r))

Roundings_eq_3: lemma
  NOT trap_enabled?(underflow(FALSE)) AND
  abs(r) < radix ^ (E_max + 1) - (1 / 2) * radix ^ (E_max + 1 - p)
  => fp_round(r, to_nearest) = FtoR[radix](RND_ENearest(b)(r))
```

7 CONCLUSION

We have presented a formalization in PVS of the floating-point arithmetic based on an existing formalization in Coq. The formalization contains a total of 280 lemmas, including 109 TCCs, in three theories. Using PVS 3.2 on a 2.60GHz processor, it takes more than 20 minutes to check all the proofs. The complete hierarchy of the PVS theories described here is illustrated in Figure 1. Each node corresponds to a theory and the arrows show the dependencies between theories (for example, the theory `axpy` described in Section 5 depends on the theory `float` described in Sections 2, 3, and 4). The gray nodes correspond to theories of the IEEE-854 specification that were updated to PVS 3.2.

The combination of a well-known formalization and a mechanical theorem prover with powerful automation capabilities will enhance the future verification of numerical applications that rely on floating-point computations.

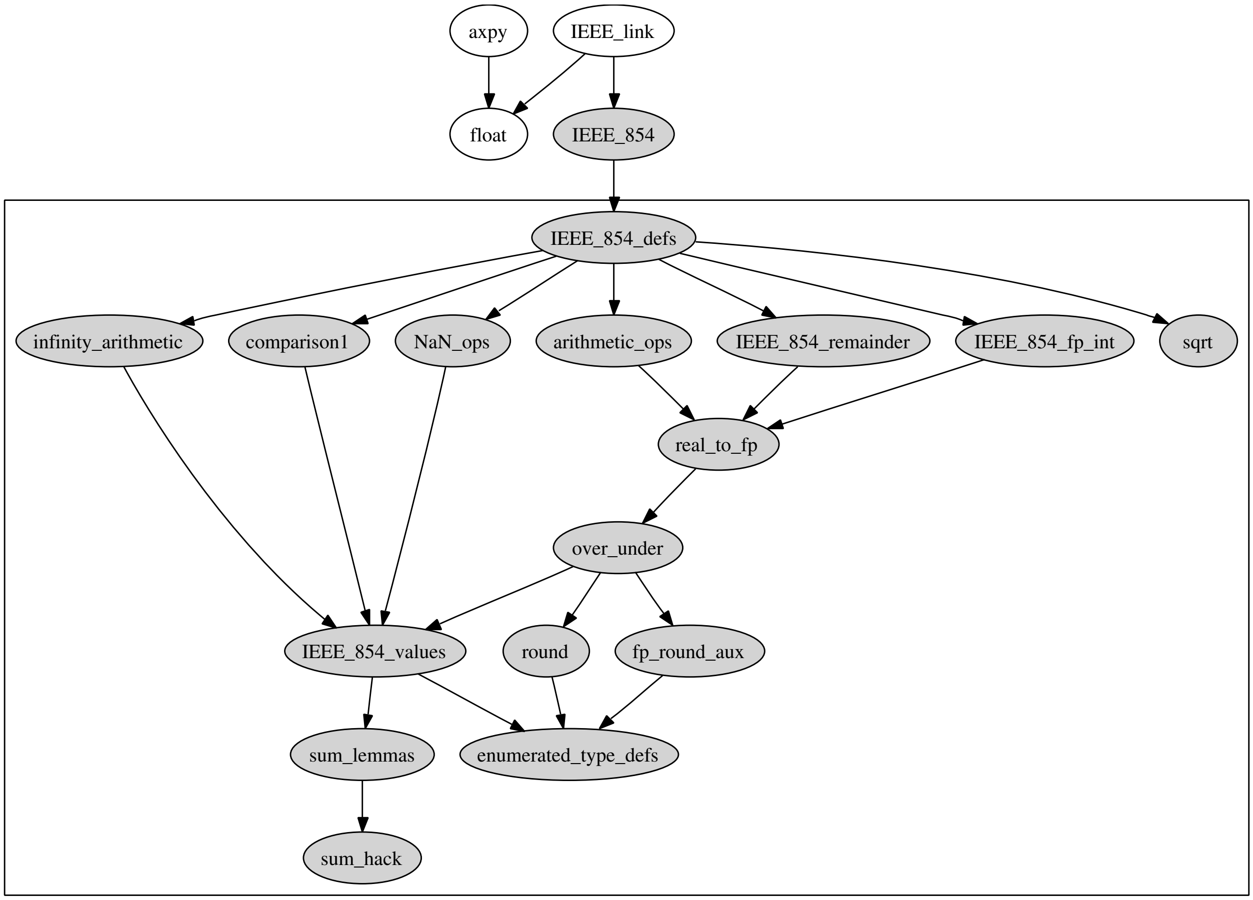

Figure 1: *Hierarchy of the PVS theories*

REFERENCES

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

[2] Yves Bertot and Pierre Casteran. *Interactive Theorem Proving and Program Development. Coq'Art : the Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[3] Sylvie Boldo. *Preuves formelles en arithmétiques à virgule flottante*. PhD thesis, École Normale Supérieure de Lyon, November 2004.

[4] 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.

[5] Sylvie Boldo and Marc Daumas. A simple test qualifying the accuracy of horner's rule for polynomials. *Numerical Algorithms*, 37(1-4):45–60, 2004.

[6] Victor A. Carreño. Interpretation of IEEE-854 floating-point standard and definition in the HOL system. Technical Report Technical Memorandum 110189, NASA Langley Research Center, 1995.

[7] 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.

[8] 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.

[9] Tim Coe. Inside the Pentium FDIV bug. *Dr. Dobb's Journal*, 20(4):129–135, 148, 1995.

[10] The Coq Development Team, LogiCal Project, INRIA. *The Coq Proof Assistant: Reference Manual, Version 8.0*, 2004.

[11] 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.

[12] Theodorus J. Dekker. A floating point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971.

[13] David Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1):5–47, 1991.

[14] John Harrison. Floating point verification in HOL light: the exponential function. Technical Report 428, University of Cambridge Computer Laboratory, 1997.

[15] Nicholas J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002. Seconde dition.

[16] Donald E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley, 1997. Troisime dition.

[17] Paul S. Miner. Defining the IEEE-854 floating-point standard in PVS. Technical Report 110167, NASA Langley Research Center, 1995.

[18] Michael Overton. Notes on numerical computing. manuscript, University of New York, 1995.

[19] Sam Owre, John. M. Rushby, and Natarajan Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[20] 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.

[21] Pat H. Sterbenz. *Floating point computation*. Prentice Hall, 1974.

[22] David Stevenson et al. A proposed standard for binary floating point arithmetic. *IEEE Computer*, 14(3):51–62, 1981.

[23] David Stevenson et al. An American national standard: IEEE standard for binary floating point arithmetic. *ACM SIGPLAN Notices*, 22(2):9–25, 1987.

[24] Sun Microsystems. *Numerical Computation Guide*, 1996.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01/10/2006	Contractor Report	

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
A High-Level Formalization of Floating-Point Numbers in PVS	
	5b. GRANT NUMBER NCC1-02043
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Boldo, Sylvie Munoz, Cesar	5d. PROJECT NUMBER 2204
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
NASA Langley Research Center Hampton, VA 23681-2199 National Institute of Aerospace (NIA) 100 Exploration Way Hampton, VA 23666	NIA Report No. 2006-01

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITOR'S ACRONYM(S)
National Aeronautics and Space Administration Washington, DC 20546-0001	NASA
	11. SPONSORING/MONITORING REPORT NUMBER NASA/CR-2006-214298

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 61
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES

Langley Technical Monitor: Ricky Butler

14. ABSTRACT

We develop a formalization of floating-point numbers in PVS based on a well-known formalization in Coq. We first describe the definitions of all the needed notions, e.g., floating-point number, format, rounding modes, etc.; then, we present an application to polynomial evaluation for elementary function evaluation. The application already existed in Coq, but our formalizatioin shows a clear improvement in the quality of the result due to the automation provided by PVS. We finally integrate our formalization into a PVS hardware-level formalization of the IEEE-854 standard previously developed at NASA.

15. SUBJECT TERMS

PVS; Floating Point Numbers

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON / 19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	23	STI Help Desk (email: help@sti.nasa.gov) (301) 621-0390

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18