A Formal Model of IEEE Floating Point Arithmetic

Lei Yu

December 14, 2021

Abstract

This development provides a formal model of IEEE-754 floating-point arithmetic. This formalization, including formal specification of the standard and proofs of important properties of floating-point arithmetic, forms the foundation for verifying programs with floating-point computation. There is also a code generation setup for floats so that we can execute programs using this formalization in functional programming languages. The definitions of the IEEE standard in Isabelle is ported from HOL Light [1].

Contents

1 Specification of the IEEE standard

theory *IEEE*
imports
HOL−Library.Float
Word-Lib.Word-Lemmas
begin

typedef (**overloaded**) (*'e*::*len*, *'f*::*len*) *float* = *UNIV*::(*1 word* × *'e word* × *'f word*) *set*
⟨*proof*⟩

setup-lifting *type-definition-float*

syntax -*float* :: *type* ⇒ *type* ⇒ *type* (*'*(-, -*'*) *float*)

parse (*'a*, *'b*) *float* as (’a::len, ’b::len) float.

⟨*ML*⟩

1.1 Derived parameters for floating point formats

definition *wordlength* :: (*'e*, *'f*) *float itself* ⇒ *nat*
where *wordlength x* = *LENGTH*(*'e*) + *LENGTH*(*'f*) + *1*

definition *bias* :: (*'e*, *'f*) *float itself* ⇒ *nat*
where *bias x* = *2*^(*LENGTH*(*'e*) − *1*) − *1*

definition *emax* :: (*'e*, *'f*) *float itself* ⇒ *nat*
where *emax x* = *unat* (− *1*::*'e word*)

abbreviation *fracwidth*::(*'e*, *'f*) *float itself* ⇒ *nat* **where**
fracwidth - ≡ *LENGTH*(*'f*)

1.2 Predicates for the four IEEE formats

definition *is-single* :: (*'e*, *'f*) *float itself* ⇒ *bool*
where *is-single x* ⟷ *LENGTH*(*'e*) = *8* ∧ *wordlength x* = *32*

definition *is-double* :: (*'e*, *'f*) *float itself* ⇒ *bool*
where *is-double x* ⟷ *LENGTH*(*'e*) = *11* ∧ *wordlength x* = *64*

definition *is-single-extended* :: (*'e*, *'f*) *float itself* ⇒ *bool*

where *is-single-extended x* $\longleftrightarrow$ *LENGTH*('*e*) $\geq$ *11* $\wedge$ *wordlength x* $\geq$ *43*

definition *is-double-extended* :: ('*e*, '*f*) *float itself* $\Rightarrow$ *bool*
where *is-double-extended x* $\longleftrightarrow$ *LENGTH*('*e*) $\geq$ *15* $\wedge$ *wordlength x* $\geq$ *79*

1.3 Extractors for fields

lift-definition *sign*::('*e*, '*f*) *float* $\Rightarrow$ *nat* **is**
λ(*s*::*1 word*, -::'*e word*, -::'*f word*). *unat s* $\langle$*proof*$\rangle$

lift-definition *exponent*::('*e*, '*f*) *float* $\Rightarrow$ *nat* **is**
λ(-, *e*::'*e word*, -). *unat e* $\langle$*proof*$\rangle$

lift-definition *fraction*::('*e*, '*f*) *float* $\Rightarrow$ *nat* **is**
λ(-, -, *f*::'*f word*). *unat f* $\langle$*proof*$\rangle$

abbreviation *real-of-word x* $\equiv$ *real* (*unat x*)

lift-definition *valof* :: ('*e*, '*f*) *float* $\Rightarrow$ *real*
is λ(*s*, *e*, *f*).
let x = (*TYPE*(('*e*, '*f*) *float*)) *in*
(*if e* = *0*
then (*−1*::*real*)^(*unat s*) * (*2* / (*2*^*bias x*)) * (*real-of-word f*/*2*^(*LENGTH*('*f*)))
else (*−1*::*real*)^(*unat s*) * ((*2*^(*unat e*)) / (*2*^*bias x*)) * (*1* + *real-of-word f*/*2*^*LENGTH*('*f*)))
$\langle$*proof*$\rangle$

1.4 Partition of numbers into disjoint classes

definition *is-nan* :: ('*e*, '*f*) *float* $\Rightarrow$ *bool*
where *is-nan a* $\longleftrightarrow$ *exponent a* = *emax TYPE*(('*e*, '*f*)*float*) $\wedge$ *fraction a* $\neq$ *0*

definition *is-infinity* :: ('*e*, '*f*) *float* $\Rightarrow$ *bool*
where *is-infinity a* $\longleftrightarrow$ *exponent a* = *emax TYPE*(('*e*, '*f*)*float*) $\wedge$ *fraction a* = *0*

definition *is-normal* :: ('*e*, '*f*) *float* $\Rightarrow$ *bool*
where *is-normal a* $\longleftrightarrow$ *0* < *exponent a* $\wedge$ *exponent a* < *emax TYPE*(('*e*, '*f*)*float*)

definition *is-denormal* :: ('*e*, '*f*) *float* $\Rightarrow$ *bool*
where *is-denormal a* $\longleftrightarrow$ *exponent a* = *0* $\wedge$ *fraction a* $\neq$ *0*

definition *is-zero* :: ('*e*, '*f*) *float* $\Rightarrow$ *bool*
where *is-zero a* $\longleftrightarrow$ *exponent a* = *0* $\wedge$ *fraction a* = *0*

definition *is-finite* :: ('*e*, '*f*) *float* $\Rightarrow$ *bool*
where *is-finite a* $\longleftrightarrow$ (*is-normal a* $\vee$ *is-denormal a* $\vee$ *is-zero a*)

1.5 Special values

lift-definition *plus-infinity* :: (*'e*, *'f*) *float* (∞) **is** (*0*, − *1*, *0*) ⟨*proof*⟩

lift-definition *topfloat* :: (*'e*, *'f*) *float* **is** (*0*, − *2*, *2^LENGTH('f)* − *1*) ⟨*proof*⟩

instantiation *float*::(*len*, *len*) *zero* **begin**

lift-definition *zero-float* :: (*'e*, *'f*) *float* **is** (*0*, *0*, *0*) ⟨*proof*⟩

instance ⟨*proof*⟩

end

1.6 Negation operation on floating point values

instantiation *float*::(*len*, *len*) *uminus* **begin**
lift-definition *uminus-float* :: (*'e*, *'f*) *float* ⇒ (*'e*, *'f*) *float* **is** λ(*s*, *e*, *f*). (*1* − *s*, *e*, *f*) ⟨*proof*⟩
instance ⟨*proof*⟩
end

abbreviation (*input*) *minus-zero* ≡ − (*0*::(*'e*, *'f*)*float*)
abbreviation (*input*) *minus-infinity* ≡ − ∞
abbreviation (*input*) *bottomfloat* ≡ − *topfloat*

1.7 Real number valuations

The largest value that can be represented in floating point format.

definition *largest* :: (*'e*, *'f*) *float itself* ⇒ *real*
where *largest x* = (*2^(emax x* − *1*) / *2^bias x*) ∗ (*2* − *1*/(*2^fracwidth x*))

Threshold, used for checking overflow.

definition *threshold* :: (*'e*, *'f*) *float itself* ⇒ *real*
where *threshold x* = (*2^(emax x* − *1*) / *2^bias x*) ∗ (*2* − *1*/(*2^(Suc*(*fracwidth x*))))

Unit of least precision.

lift-definition *one-lp*::(*'e* ,*'f*) *float* ⇒ (*'e* ,*'f*) *float* **is** λ(*s*, *e*, *f*). (*0*, *e*::*'e word*, *1*) ⟨*proof*⟩
lift-definition *zero-lp*::(*'e* ,*'f*) *float* ⇒ (*'e* ,*'f*) *float* **is** λ(*s*, *e*, *f*). (*0*, *e*::*'e word*, *0*) ⟨*proof*⟩

definition *ulp* :: (*'e*, *'f*) *float* ⇒ *real* **where** *ulp a* = *valof* (*one-lp a*) − *valof* (*zero-lp a*)

Enumerated type for rounding modes.

datatype *roundmode* = *To-nearest* | *float-To-zero* | *To-pinfinity* | *To-ninfinity*

Characterization of best approximation from a set of abstract values.

definition *is-closest v s x a* $\longleftrightarrow$ $a \in s \wedge (\forall b.\ b \in s \longrightarrow |v\ a - x| \leq |v\ b - x|)$

Best approximation with a deciding preference for multiple possibilities.

definition *closest v p s x* =
(*SOME a. is-closest v s x a* $\wedge$ ((*∃ b. is-closest v s x b* $\wedge$ *p b*) $\longrightarrow$ *p a*))

1.8 Rounding

fun *round* :: *roundmode* $\Rightarrow$ *real* $\Rightarrow$ (*'e* ,*'f*) *float*
where
round To-nearest y =
(*if y* $\leq -$ *threshold TYPE*((*'e* ,*'f*) *float*) *then minus-infinity*
else if y $\geq$ *threshold TYPE*((*'e* ,*'f*) *float*) *then plus-infinity*
else closest (*valof*) (*λa. even* (*fraction a*)) {*a. is-finite a*} *y*)
| *round float-To-zero y* =
(*if y* $<$ $-$ *largest TYPE*((*'e* ,*'f*) *float*) *then bottomfloat*
else if y $>$ *largest TYPE*((*'e* ,*'f*) *float*) *then topfloat*
else closest (*valof*) (*λa. True*) {*a. is-finite a* $\wedge$ |*valof a*| $\leq$ |*y*|} *y*)
| *round To-pinfinity y* =
(*if y* $<$ $-$ *largest TYPE*((*'e* ,*'f*) *float*) *then bottomfloat*
else if y $>$ *largest TYPE*((*'e* ,*'f*) *float*) *then plus-infinity*
else closest (*valof*) (*λa. True*) {*a. is-finite a* $\wedge$ *valof a* $\geq$ *y*} *y*)
| *round To-ninfinity y* =
(*if y* $<$ $-$ *largest TYPE*((*'e* ,*'f*) *float*) *then minus-infinity*
else if y $>$ *largest TYPE*((*'e* ,*'f*) *float*) *then topfloat*
else closest (*valof*) (*λa. True*) {*a. is-finite a* $\wedge$ *valof a* $\leq$ *y*} *y*)

Rounding to integer values in floating point format.

definition *is-integral* :: (*'e* ,*'f*) *float* $\Rightarrow$ *bool*
where *is-integral a* $\longleftrightarrow$ *is-finite a* $\wedge$ (*∃ n*::*nat. |valof a| = real n*)

fun *intround* :: *roundmode* $\Rightarrow$ *real* $\Rightarrow$ (*'e* ,*'f*) *float*
where
intround To-nearest y =
(*if y* $\leq -$ *threshold TYPE*((*'e* ,*'f*) *float*) *then minus-infinity*
else if y $\geq$ *threshold TYPE*((*'e* ,*'f*) *float*) *then plus-infinity*
else closest (*valof*) (*λa.* (*∃ n*::*nat. even n* $\wedge$ |*valof a*| = *real n*)) {*a. is-integral a*} *y*)
|*intround float-To-zero y* =
(*if y* $<$ $-$ *largest TYPE*((*'e* ,*'f*) *float*) *then bottomfloat*
else if y $>$ *largest TYPE*((*'e* ,*'f*) *float*) *then topfloat*
else closest (*valof*) (*λx. True*) {*a. is-integral a* $\wedge$ |*valof a*| $\leq$ |*y*|} *y*)
|*intround To-pinfinity y* =
(*if y* $<$ $-$ *largest TYPE*((*'e* ,*'f*) *float*) *then bottomfloat*
else if y $>$ *largest TYPE*((*'e* ,*'f*) *float*) *then plus-infinity*
else closest (*valof*) (*λx. True*) {*a. is-integral a* $\wedge$ *valof a* $\geq$ *y*} *y*)
|*intround To-ninfinity y* =
(*if y* $<$ $-$ *largest TYPE*((*'e* ,*'f*) *float*) *then minus-infinity*

else if $y >$ *largest* $TYPE(('e, 'f)$ *float*$)$ *then topfloat*
else closest $(valof)$ $(\lambda x.\ True)$ $\{a.\ is\text{-}integral\ a \wedge valof\ a \geq y\}$ $y)$

Round, choosing between -0.0 or +0.0

definition *float-round*::*roundmode* $\Rightarrow$ *bool* $\Rightarrow$ *real* $\Rightarrow$ $('e, 'f)$ *float*
where *float-round mode toneg r* =
$($*let* $x =$ *round mode r in*
if is-zero x
then if toneg
then minus-zero
else 0
else x$)$

Non-standard of NaN.

definition *some-nan* :: $('e, 'f)$ *float*
where *some-nan* $=$ $(SOME\ a.\ is\text{-}nan\ a)$

Coercion for signs of zero results.

definition *zerosign* :: *nat* $\Rightarrow$ $('e, 'f)$ *float* $\Rightarrow$ $('e, 'f)$ *float*
where *zerosign s a* =
$($*if is-zero a then* $($*if* $s = 0$ *then 0 else* $-\ 0)$ *else a*$)$

Remainder operation.

definition *rem* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real*
where *rem x y* =
$($*let* $n =$ *closest id* $(\lambda x.\ \exists\, n{::}nat.\ even\ n \wedge |x| = real\ n)$ $\{x.\ \exists\, n :: nat.\ |x| = real\ n\}$ $(x\ /\ y)$
in $x - n * y$$)$

definition *frem* :: *roundmode* $\Rightarrow$ $('e, 'f)$ *float* $\Rightarrow$ $('e, 'f)$ *float* $\Rightarrow$ $('e, 'f)$ *float*
where *frem m a b* =
$($*if is-nan a* $\vee$ *is-nan b* $\vee$ *is-infinity a* $\vee$ *is-zero b then some-nan*
else zerosign $(sign\ a)$ $(round\ m\ (rem\ (valof\ a)\ (valof\ b))))$

1.9 Definitions of the arithmetic operations

definition *fintrnd* :: *roundmode* $\Rightarrow$ $('e, 'f)$ *float* $\Rightarrow$ $('e, 'f)$ *float*
where *fintrnd m a* =
$($*if is-nan a then* $(some\text{-}nan)$
else if is-infinity a then a
else zerosign $(sign\ a)$ $(intround\ m\ (valof\ a)))$

definition *fadd* :: *roundmode* $\Rightarrow$ $('e, 'f)$ *float* $\Rightarrow$ $('e, 'f)$ *float* $\Rightarrow$ $('e, 'f)$ *float*
where *fadd m a b* =
$($*if is-nan a* $\vee$ *is-nan b* $\vee$ $($*is-infinity a* $\wedge$ *is-infinity b* $\wedge$ *sign a* $\neq$ *sign b*$)$
then some-nan
else if $($*is-infinity a*$)$ *then a*
else if $($*is-infinity b*$)$ *then b*
else

```
    zerosign
      (if is-zero a ∧ is-zero b ∧ sign a = sign b then sign a
       else if m = To-ninfinity then 1 else 0)
      (round m (valof a + valof b)))

definition fsub :: roundmode ⇒ ('e ,'f) float ⇒ ('e ,'f) float ⇒ ('e ,'f) float
  where fsub m a b =
    (if is-nan a ∨ is-nan b ∨ (is-infinity a ∧ is-infinity b ∧ sign a = sign b)
     then some-nan
     else if is-infinity a then a
     else if is-infinity b then − b
     else
      zerosign
        (if is-zero a ∧ is-zero b ∧ sign a ≠ sign b then sign a
         else if m = To-ninfinity then 1 else 0)
        (round m (valof a − valof b)))

definition fmul :: roundmode ⇒ ('e ,'f) float ⇒ ('e ,'f) float ⇒ ('e ,'f) float
  where fmul m a b =
    (if is-nan a ∨ is-nan b ∨ (is-zero a ∧ is-infinity b) ∨ (is-infinity a ∧ is-zero b)
     then some-nan
     else if is-infinity a ∨ is-infinity b
     then (if sign a = sign b then plus-infinity else minus-infinity)
     else zerosign (if sign a = sign b then 0 else 1 ) (round m (valof a ∗ valof b)))

definition fdiv :: roundmode ⇒ ('e ,'f) float ⇒ ('e ,'f) float ⇒ ('e ,'f) float
  where fdiv m a b =
    (if is-nan a ∨ is-nan b ∨ (is-zero a ∧ is-zero b) ∨ (is-infinity a ∧ is-infinity b)
     then some-nan
     else if is-infinity a ∨ is-zero b
     then (if sign a = sign b then plus-infinity else minus-infinity)
     else if is-infinity b
     then (if sign a = sign b then 0 else − 0)
     else zerosign (if sign a = sign b then 0 else 1) (round m (valof a / valof b)))

definition fsqrt :: roundmode ⇒ ('e ,'f) float ⇒ ('e ,'f) float
  where fsqrt m a =
    (if is-nan a then some-nan
     else if is-zero a ∨ is-infinity a ∧ sign a = 0 then a
     else if sign a = 1 then some-nan
     else zerosign (sign a) (round m (sqrt (valof a))))

definition fmul-add :: roundmode ⇒ ('t ,'w) float ⇒ ('t ,'w) float ⇒ ('t ,'w) float
⇒ ('t ,'w) float
  where fmul-add mode x y z =
      (let signP = if sign x = sign y then 0 else 1 in
      let infP = is-infinity x  ∨ is-infinity y
      in
        if is-nan x ∨ is-nan y ∨ is-nan z then some-nan
```

else if is-infinity x ∧ is-zero y ∨
is-zero x ∧ is-infinity y ∨
is-infinity z ∧ infP ∧ signP ≠ sign z then
some-nan
else if is-infinity z ∧ (sign z = 0) ∨ infP ∧ (signP = 0)
then plus-infinity
else if is-infinity z ∧ (sign z = 1) ∨ infP ∧ (signP = 1)
then minus-infinity
else
let r1 = valof x ∗ valof y;
r2 = valof z
in
float-round mode
(if (r1 = 0) ∧ (r2 = 0) ∧ (signP = sign z) then
signP = 1
else mode = To-ninfinity) (r1 + r2))

1.10 Comparison operations

datatype *ccode = Gt | Lt | Eq | Und*

definition *fcompare :: ('e ,'f) float ⇒ ('e ,'f) float ⇒ ccode*
where *fcompare a b =*
(if is-nan a ∨ is-nan b then Und
else if is-infinity a ∧ sign a = 1
then (if is-infinity b ∧ sign b = 1 then Eq else Lt)
else if is-infinity a ∧ sign a = 0
then (if is-infinity b ∧ sign b = 0 then Eq else Gt)
else if is-infinity b ∧ sign b = 1 then Gt
else if is-infinity b ∧ sign b = 0 then Lt
else if valof a < valof b then Lt
else if valof a = valof b then Eq
else Gt)

definition *flt :: ('e ,'f) float ⇒ ('e ,'f) float ⇒ bool*
where *flt a b ⟷ fcompare a b = Lt*

definition *fle :: ('e ,'f) float ⇒ ('e ,'f) float ⇒ bool*
where *fle a b ⟷ fcompare a b = Lt ∨ fcompare a b = Eq*

definition *fgt :: ('e ,'f) float ⇒ ('e ,'f) float ⇒ bool*
where *fgt a b ⟷ fcompare a b = Gt*

definition *fge :: ('e ,'f) float ⇒ ('e ,'f) float ⇒ bool*
where *fge a b ⟷ fcompare a b = Gt ∨ fcompare a b = Eq*

definition *feq :: ('e ,'f) float ⇒ ('e ,'f) float ⇒ bool*
where *feq a b ⟷ fcompare a b = Eq*

2 Specify float to be double precision and round to even

instantiation *float* :: (*len*, *len*) *plus*
begin

definition *plus-float* :: ('*a*, '*b*) *float* ⇒ ('*a*, '*b*) *float* ⇒ ('*a*, '*b*) *float*
where *a* + *b* = *fadd To-nearest a b*

instance ⟨*proof*⟩

end

instantiation *float* :: (*len*, *len*) *minus*
begin

definition *minus-float* :: ('*a*, '*b*) *float* ⇒ ('*a*, '*b*) *float* ⇒ ('*a*, '*b*) *float*
where *a* − *b* = *fsub To-nearest a b*

instance ⟨*proof*⟩

end

instantiation *float* :: (*len*, *len*) *times*
begin

definition *times-float* :: ('*a*, '*b*) *float* ⇒ ('*a*, '*b*) *float* ⇒ ('*a*, '*b*) *float*
where *a* ∗ *b* = *fmul To-nearest a b*

instance ⟨*proof*⟩

end

instantiation *float* :: (*len*, *len*) *one*
begin

lift-definition *one-float* :: ('*a*, '*b*) *float* **is** (*0*, *2*^(*LENGTH*('*a*) − *1*) − *1*, *0*)
⟨*proof*⟩

instance ⟨*proof*⟩

end

instantiation *float* :: (*len*, *len*) *inverse*
begin

definition *divide-float* :: ('*a*, '*b*) *float* ⇒ ('*a*, '*b*) *float* ⇒ ('*a*, '*b*) *float*
where *a div b* = *fdiv To-nearest a b*

definition *inverse-float* :: ('a, 'b) *float* ⇒ ('a, 'b) *float*
where *inverse-float a* = *fdiv To-nearest 1 a*

instance ⟨*proof*⟩

end

definition *float-rem* :: ('a, 'b) *float* ⇒ ('a, 'b) *float* ⇒ ('a, 'b) *float*
where *float-rem a b* = *frem To-nearest a b*

definition *float-sqrt* :: ('a, 'b) *float* ⇒ ('a, 'b) *float*
where *float-sqrt a* = *fsqrt To-nearest a*

definition *ROUNDFLOAT* ::('a, 'b) *float* ⇒ ('a, 'b) *float*
where *ROUNDFLOAT a* = *fintrnd To-nearest a*

instantiation *float* :: (*len*, *len*) *ord*
begin

definition *less-float* :: ('a, 'b) *float* ⇒ ('a, 'b) *float* ⇒ *bool*
where *a* < *b* ⟷ *flt a b*

definition *less-eq-float* :: ('a, 'b) *float* ⇒ ('a, 'b) *float* ⇒ *bool*
where *a* ≤ *b* ⟷ *fle a b*

instance ⟨*proof*⟩

end

definition *float-eq* :: ('a, 'b) *float* ⇒ ('a, 'b) *float* ⇒ *bool* (**infixl** ≐ *70*)
where *float-eq a b* = *feq a b*

instantiation *float* :: (*len*, *len*) *abs*
begin

definition *abs-float* :: ('a, 'b) *float* ⇒ ('a, 'b) *float*
where *abs-float a* = (*if sign a* = *0 then a else* − *a*)

instance ⟨*proof*⟩
end

The *1* + *ε* property.

definition *normalizes* :: - *itself* ⇒ *real* ⇒ *bool*
where *normalizes float-format x* =
(*1*/ (*2*::*real*)^(*bias float-format* − *1*) ≤ |*x*| ∧ |*x*| < *threshold float-format*)

end

3 Proofs of Properties about Floating Point Arithmetic

theory *IEEE-Properties*
imports *IEEE*
begin

3.1 Theorems derived from definitions

lemma *valof-eq*:
valof x =
(*if exponent x* = *0*
then (− *1*) ^ *sign x* ∗ (*2* / *2* ^ *bias TYPE*((*'a*, *'b*) *float*)) ∗
(*real* (*fraction x*) / *2* ^ *LENGTH*(*'b*))
else (− *1*) ^ *sign x* ∗ (*2* ^ *exponent x* / *2* ^ *bias TYPE*((*'a*, *'b*) *float*)) ∗
(*1* + *real* (*fraction x*) / *2* ^ *LENGTH*(*'b*)))
for *x*::(*'a*, *'b*) *float*
⟨*proof*⟩

lemma *exponent-le* [*simp*]:
‹*exponent a* ≤ *mask LENGTH*(*'a*)› **for** *a* :: ‹(*'a*, -) *float*›
⟨*proof*⟩

lemma *exponent-not-less* [*simp*]:
‹¬ *mask LENGTH*(*'a*) < *IEEE.exponent a*› **for** *a* :: ‹(*'a*, -) *float*›
⟨*proof*⟩

lemma *infinity-simps*:
sign (*plus-infinity*::(*'e*, *'f*)*float*) = *0*
sign (*minus-infinity*::(*'e*, *'f*)*float*) = *1*
exponent (*plus-infinity*::(*'e*, *'f*)*float*) = *emax TYPE*((*'e*, *'f*)*float*)
exponent (*minus-infinity*::(*'e*, *'f*)*float*) = *emax TYPE*((*'e*, *'f*)*float*)
fraction (*plus-infinity*::(*'e*, *'f*)*float*) = *0*
fraction (*minus-infinity*::(*'e*, *'f*)*float*) = *0*
⟨*proof*⟩

lemma *zero-simps*:
sign (*0*::(*'e*, *'f*)*float*) = *0*
sign (− *0*::(*'e*, *'f*)*float*) = *1*
exponent (*0*::(*'e*, *'f*)*float*) = *0*
exponent (− *0*::(*'e*, *'f*)*float*) = *0*
fraction (*0*::(*'e*, *'f*)*float*) = *0*
fraction (− *0*::(*'e*, *'f*)*float*) = *0*
⟨*proof*⟩

lemma *emax-eq*: *emax x* = *2* ^ *LENGTH*(*'e*) − *1*
for *x*::(*'e*, *'f*)*float itself*
⟨*proof*⟩

lemma *topfloat-simps*:
sign (*topfloat*::('*e*, '*f*)*float*) = *0*
exponent (*topfloat*::('*e*, '*f*)*float*) = *emax TYPE*(('*e*, '*f*)*float*) − *1*
fraction (*topfloat*::('*e*, '*f*)*float*) = *2* ^ (*fracwidth TYPE*(('*e*, '*f*)*float*)) − *1*
and *bottomfloat-simps*:
sign (*bottomfloat*::('*e*, '*f*)*float*) = *1*
exponent (*bottomfloat*::('*e*, '*f*)*float*) = *emax TYPE*(('*e*, '*f*)*float*) − *1*
fraction (*bottomfloat*::('*e*, '*f*)*float*) = *2* ^ (*fracwidth TYPE*(('*e*, '*f*)*float*)) − *1*
⟨*proof*⟩

lemmas *float-defs* =
is-finite-def is-infinity-def is-zero-def is-nan-def
is-normal-def is-denormal-def valof-eq
less-eq-float-def less-float-def
flt-def fgt-def fle-def fge-def feq-def
fcompare-def
infinity-simps
zero-simps
topfloat-simps
bottomfloat-simps
float-eq-def

lemma *float-cases*: *is-nan a* ∨ *is-infinity a* ∨ *is-normal a* ∨ *is-denormal a* ∨ *is-zero a*
⟨*proof*⟩

lemma *float-cases-finite*: *is-nan a* ∨ *is-infinity a* ∨ *is-finite a*
⟨*proof*⟩

lemma *float-zero1*[*simp*]: *is-zero 0*
⟨*proof*⟩

lemma *float-zero2*[*simp*]: *is-zero* (− *x*) ⟷ *is-zero x*
⟨*proof*⟩

lemma *emax-pos*[*simp*]: *0* < *emax x emax x* ≠ *0*
⟨*proof*⟩

The types of floating-point numbers are mutually distinct.

lemma *float-distinct*:
¬ (*is-nan a* ∧ *is-infinity a*)
¬ (*is-nan a* ∧ *is-normal a*)
¬ (*is-nan a* ∧ *is-denormal a*)
¬ (*is-nan a* ∧ *is-zero a*)
¬ (*is-infinity a* ∧ *is-normal a*)
¬ (*is-infinity a* ∧ *is-denormal a*)
¬ (*is-infinity a* ∧ *is-zero a*)
¬ (*is-normal a* ∧ *is-denormal a*)
¬ (*is-denormal a* ∧ *is-zero a*)

⟨*proof*⟩

lemma *denormal-imp-not-zero*: *is-denormal f* ⟹ ¬*is-zero f*
⟨*proof*⟩

lemma *normal-imp-not-zero*: *is-normal f* ⟹ ¬*is-zero f*
⟨*proof*⟩

lemma *normal-imp-not-denormal*: *is-normal f* ⟹ ¬*is-denormal f*
⟨*proof*⟩

lemma *denormal-zero*[*simp*]: ¬*is-denormal 0* ¬*is-denormal minus-zero*
⟨*proof*⟩

lemma *normal-zero*[*simp*]: ¬*is-normal 0* ¬*is-normal minus-zero*
⟨*proof*⟩

lemma *float-distinct-finite*: ¬ (*is-nan a* ∧ *is-finite a*) ¬(*is-infinity a* ∧ *is-finite a*)
⟨*proof*⟩

lemma *finite-infinity*: *is-finite a* ⟹ ¬ *is-infinity a*
⟨*proof*⟩

lemma *finite-nan*: *is-finite a* ⟹ ¬ *is-nan a*
⟨*proof*⟩

For every real number, the floating-point numbers closest to it always exist.

lemma *is-closest-exists*:
fixes *v* :: (*'e*, *'f*)*float* ⇒ *real*
and *s* :: (*'e*, *'f*)*float set*
assumes *finite*: *finite s*
and *non-empty*: *s* ≠ {}
shows ∃ *a*. *is-closest v s x a*
⟨*proof*⟩

lemma *closest-is-everything*:
fixes *v* :: (*'e*, *'f*)*float* ⇒ *real*
and *s* :: (*'e*, *'f*)*float set*
assumes *finite*: *finite s*
and *non-empty*: *s* ≠ {}
shows *is-closest v s x* (*closest v p s x*) ∧
((∃ *b*. *is-closest v s x b* ∧ *p b*) ⟶ *p* (*closest v p s x*))
⟨*proof*⟩

lemma *closest-in-set*:
fixes *v* :: (*'e*, *'f*)*float* ⇒ *real*
assumes *finite s* **and** *s* ≠ {}
shows *closest v p s x* ∈ *s*
⟨*proof*⟩

lemma *closest-is-closest-finite*:
fixes $v :: ('e, 'f)float \Rightarrow real$
assumes *finite* s **and** $s \neq \{\}$
shows *is-closest* v s x (*closest* v p s x)
⟨*proof*⟩

instance *float*::(*len*, *len*) *finite* ⟨*proof*⟩

lemma *is-finite-nonempty*: $\{a.$ *is-finite* $a\} \neq \{\}$
⟨*proof*⟩

lemma *closest-is-closest*:
fixes $v :: ('e, 'f)float \Rightarrow real$
assumes $s \neq \{\}$
shows *is-closest* v s x (*closest* v p s x)
⟨*proof*⟩

3.2 Properties about ordering and bounding

Lifting of non-exceptional comparisons.

lemma *float-lt* [*simp*]:
assumes *is-finite a is-finite b*
shows $a < b \longleftrightarrow$ *valof a* $<$ *valof b*
⟨*proof*⟩

lemma *float-eq* [*simp*]:
assumes *is-finite a is-finite b*
shows $a \doteq b \longleftrightarrow$ *valof a* $=$ *valof b*
⟨*proof*⟩

lemma *float-le* [*simp*]:
assumes *is-finite a is-finite b*
shows $a \leq b \longleftrightarrow$ *valof a* $\leq$ *valof b*
⟨*proof*⟩

Reflexivity of equality for non-NaNs.

lemma *float-eq-refl* [*simp*]: $a \doteq a \longleftrightarrow \neg$ *is-nan a*
⟨*proof*⟩

Properties about Ordering.

lemma *float-lt-trans*: *is-finite a* $\Longrightarrow$ *is-finite b* $\Longrightarrow$ *is-finite c* $\Longrightarrow a < b \Longrightarrow b < c \Longrightarrow a < c$
⟨*proof*⟩

lemma *float-le-less-trans*: *is-finite a* $\Longrightarrow$ *is-finite b* $\Longrightarrow$ *is-finite c* $\Longrightarrow a \leq b \Longrightarrow b < c \Longrightarrow a < c$
⟨*proof*⟩

lemma *float-le-trans*: *is-finite a* $\Longrightarrow$ *is-finite b* $\Longrightarrow$ *is-finite c* $\Longrightarrow$ $a \leq b \Longrightarrow b \leq c \Longrightarrow a \leq c$
⟨*proof*⟩

lemma *float-le-neg*: *is-finite a* $\Longrightarrow$ *is-finite b* $\Longrightarrow$ $\neg\ a < b \longleftrightarrow b \leq a$
⟨*proof*⟩

Properties about bounding.

lemma *float-le-infinity* [*simp*]: $\neg$ *is-nan a* $\Longrightarrow$ $a \leq$ *plus-infinity*
⟨*proof*⟩

lemma *zero-le-topfloat*[*simp*]: $0 \leq$ *topfloat* $-\ 0 \leq$ *topfloat*
⟨*proof*⟩

lemma *LENGTH-contr*:
$Suc\ 0 < LENGTH('e) \Longrightarrow 2\ \hat{}\ LENGTH('e{::}len) \leq Suc\ (Suc\ 0) \Longrightarrow False$
⟨*proof*⟩

lemma *valof-topfloat*: *valof* $(topfloat{::}('e,\ 'f)float) = largest\ TYPE(('e,\ 'f)float)$
if $LENGTH('e) > 1$
⟨*proof*⟩

lemma *float-frac-le*: *fraction* $a \leq 2\hat{}LENGTH('f) - 1$
for $a{::}('e,\ 'f)float$
⟨*proof*⟩

lemma *float-exp-le*: *is-finite a* $\Longrightarrow$ *exponent* $a \leq emax\ TYPE(('e,\ 'f)float) - 1$
for $a{::}('e,\ 'f)float$
⟨*proof*⟩

lemma *float-sign-le*: $(-1{::}real)\hat{}(sign\ a) = 1 \vee (-1{::}real)\hat{}(sign\ a) = -1$
⟨*proof*⟩

lemma *exp-less*: $a \leq b \Longrightarrow (2{::}real)\hat{}a \leq 2\hat{}b$ **for** $a\ b :: nat$
⟨*proof*⟩

lemma *div-less*: $a \leq b \wedge c > 0 \Longrightarrow a/c \leq b/c$ **for** $a\ b\ c :: 'a{::}linordered\text{-}field$
⟨*proof*⟩

lemma *finite-topfloat*: *is-finite topfloat*
⟨*proof*⟩

lemmas *float-leI* = *float-le*[*THEN iffD2*]

lemma *factor-minus*: $x * a - x = x * (a - 1)$
for $x\ a{::}'a{::}comm\text{-}semiring\text{-}1\text{-}cancel$
⟨*proof*⟩

lemma *real-le-power-numeral-diff*: *real a* $\leq$ *numeral b* $\hat{}$ *n* $-$ *1* $\longleftrightarrow$ *a* $\leq$ *numeral b* $\hat{}$ *n* $-$ *1*
⟨*proof*⟩

definition *denormal-exponent*::(*'e*, *'f*)*float itself* $\Rightarrow$ *int* **where**
denormal-exponent x = *1* $-$ (*int* (*LENGTH*(*'f*)) + *int* (*bias TYPE*((*'e*, *'f*)*float*)))

definition *normal-exponent*::(*'e*, *'f*)*float* $\Rightarrow$ *int* **where**
normal-exponent x = *int* (*exponent x*) $-$ *int* (*bias TYPE*((*'e*, *'f*)*float*)) $-$ *int* (*LENGTH*(*'f*))

definition *denormal-mantissa*::(*'e*, *'f*)*float* $\Rightarrow$ *int* **where**
denormal-mantissa x = (*−1*::*int*)$\hat{}$*sign x* $*$ *int* (*fraction x*)

definition *normal-mantissa*::(*'e*, *'f*)*float* $\Rightarrow$ *int* **where**
normal-mantissa x = (*−1*::*int*)$\hat{}$*sign x* $*$ (*2*$\hat{}$*LENGTH*(*'f*) + *int* (*fraction x*))

lemma *unat-one-word-le*: *unat a* $\leq$ *Suc 0* **for** *a*::*1 word*
⟨*proof*⟩

lemma *one-word-le*: *a* $\leq$ *1* **for** *a*::*1 word*
⟨*proof*⟩

lemma *sign-cases*[*case-names pos neg*]:
obtains *sign x* = *0* | *sign x* = *1*
⟨*proof*⟩

lemma *is-infinity-cases*:
assumes *is-infinity x*
obtains *x* = *plus-infinity* | *x* = *minus-infinity*
⟨*proof*⟩

lemma *is-zero-cases*:
assumes *is-zero x*
obtains *x* = *0* | *x* = $-$ *0*
⟨*proof*⟩

lemma *minus-minus-float*[*simp*]: $-$ ($-$*f*) = *f* **for** *f*::(*'e*, *'f*)*float*
⟨*proof*⟩

lemma *sign-minus-float*: *sign* ($-$*f*) = (*1* $-$ *sign f*) **for** *f*::(*'e*, *'f*)*float*
⟨*proof*⟩

lemma *exponent-uminus*[*simp*]: *exponent* ($-$ *f*) = *exponent f* ⟨*proof*⟩
lemma *fraction-uminus*[*simp*]: *fraction* ($-$ *f*) = *fraction f* ⟨*proof*⟩

lemma *is-normal-minus-float*[*simp*]: *is-normal* ($-$*f*) = *is-normal f* **for** *f*::(*'e*, *'f*)*float*
⟨*proof*⟩

lemma *is-denormal-minus-float*[*simp*]: *is-denormal* $(-f)$ = *is-denormal f* **for** *f*::('*e*, '*f*)*float*
⟨*proof*⟩

lemma *bitlen-normal-mantissa*:
bitlen (*abs* (*normal-mantissa x*)) = *Suc LENGTH*('*f*) **for** *x*::('*e*, '*f*)*float*
⟨*proof*⟩

lemma *less-int-natI*: *x* < *y* **if** *0* ≤ *x nat x* < *nat y*
⟨*proof*⟩

lemma *normal-exponent-bounds-int*:
2 − *2* ^ (*LENGTH*('*e*) − *1*) − *int LENGTH*('*f*) ≤ *normal-exponent x*
normal-exponent x ≤ *2* ^ (*LENGTH*('*e*) − *1*) − *int LENGTH*('*f*) − *1*
if *is-normal x*
for *x*::('*e*, '*f*)*float*
⟨*proof*⟩

lemmas *of-int-leI* = *of-int-le-iff*[*THEN iffD2*]

lemma *normal-exponent-bounds-real*:
2 − *2* ^ (*LENGTH*('*e*) − *1*) − *real LENGTH*('*f*) ≤ *normal-exponent x*
normal-exponent x ≤ *2* ^ (*LENGTH*('*e*) − *1*) − *real LENGTH*('*f*) − *1*
if *is-normal x*
for *x*::('*e*, '*f*)*float*
⟨*proof*⟩

lemma *float-eqI*:
x = *y* **if** *sign x* = *sign y fraction x* = *fraction y exponent x* = *exponent y*
⟨*proof*⟩

lemma *float-induct*[*induct type*:*float*, *case-names normal denormal neg zero infinity nan*]:
fixes *a*::('*e*, '*f*)*float*
assumes *normal*:
⋀*x*. *is-normal x* ⟹ *valof x* = *normal-mantissa x* ∗ *2 powr normal-exponent x* ⟹ *P x*
assumes *denormal*:
⋀*x*. *is-denormal x* ⟹
valof x = *denormal-mantissa x* ∗ *2 powr denormal-exponent TYPE*(('*e*, '*f*)*float*) ⟹
P x
assumes *zero*: *P 0 P minus-zero*
assumes *infty*: *P plus-infinity P minus-infinity*
assumes *nan*: ⋀*x*. *is-nan x* ⟹ *P x*
shows *P a*
⟨*proof*⟩

lemma *infinite-infinity*[*simp*]: ¬ *is-finite plus-infinity* ¬ *is-finite minus-infinity*

⟨*proof*⟩

lemma *nan-not-finite*[*simp*]: *is-nan x* ⟹ ¬ *is-finite x*
⟨*proof*⟩

lemma *valof-nonneg*:
valof x ≥ *0* **if** *sign x* = *0* **for** *x*::(*'e*, *'f*)*float*
⟨*proof*⟩

lemma *valof-nonpos*:
valof x ≤ *0* **if** *sign x* = *1* **for** *x*::(*'e*, *'f*)*float*
⟨*proof*⟩

lemma *real-le-intI*: *x* ≤ *y* **if** *floor x* ≤ *floor y x* ∈ ℤ **for** *x y*::*real*
⟨*proof*⟩

lemma *real-of-int-le-2-powr-bitlenI*:
real-of-int x ≤ *2 powr n* − *1* **if** *bitlen* (*abs x*) ≤ *m m* ≤ *n*
⟨*proof*⟩

lemma *largest-eq*:
largest TYPE((*'e*, *'f*)*float*) =
(*2* ^ (*LENGTH*(*'f*) + *1*) − *1*) ∗ *2 powr real-of-int* (*2* ^ (*LENGTH*(*'e*) − *1*) −
int LENGTH(*'f*) − *1*)
⟨*proof*⟩

lemma *bitlen-denormal-mantissa*:
bitlen (*abs* (*denormal-mantissa x*)) ≤ *LENGTH*(*'f*) **for** *x*::(*'e*, *'f*)*float*
⟨*proof*⟩

lemma *float-le-topfloat*:
fixes *a*::(*'e*, *'f*)*float*
assumes *is-finite a LENGTH*(*'e*) > *1*
shows *a* ≤ *topfloat*
⟨*proof*⟩

lemma *float-val-le-largest*:
valof a ≤ *largest TYPE*((*'e*, *'f*)*float*)
if *is-finite a LENGTH*(*'e*) > *1*
for *a*::(*'e*, *'f*)*float*
⟨*proof*⟩

lemma *float-val-lt-threshold*:
valof a < *threshold TYPE*((*'e*, *'f*)*float*)
if *is-finite a LENGTH*(*'e*) > *1*
for *a*::(*'e*, *'f*)*float*
⟨*proof*⟩

3.3 Algebraic properties about basic arithmetic

Commutativity of addition.

lemma
assumes *is-finite a is-finite b*
shows *float-plus-comm-eq*: $a + b = b + a$
and *float-plus-comm*: *is-finite* $(a + b) \Longrightarrow (a + b) \doteq (b + a)$
⟨*proof*⟩

The floating-point number a falls into the same category as the negation of *a*.

lemma *is-zero-uminus*[*simp*]: *is-zero* $(-a) \longleftrightarrow$ *is-zero a*
⟨*proof*⟩

lemma *is-infinity-uminus* [*simp*]: *is-infinity* $(-a) =$ *is-infinity a*
⟨*proof*⟩

lemma *is-finite-uminus*[*simp*]: *is-finite* $(-a) \longleftrightarrow$ *is-finite a*
⟨*proof*⟩

lemma *is-nan-uminus*[*simp*]: *is-nan* $(-a) \longleftrightarrow$ *is-nan a*
⟨*proof*⟩

The sign of a and the sign of a's negation is different.

lemma *float-neg-sign*: $(sign\ a) \neq (sign\ (-a))$
⟨*proof*⟩

lemma *float-neg-sign1*: $sign\ a = sign\ (-b) \longleftrightarrow sign\ a \neq sign\ b$
⟨*proof*⟩

lemma *valof-uminus*:
assumes *is-finite a*
shows *valof* $(-a) = -$ *valof a* (**is** *?L* = *?R*)
⟨*proof*⟩

Showing $a + (-b) = a - b$.

lemma *float-neg-add*:
is-finite $a \Longrightarrow$ *is-finite* $b \Longrightarrow$ *is-finite* $(a - b) \Longrightarrow$ *valof* $a +$ *valof* $(-b) =$ *valof* $a -$ *valof* b
⟨*proof*⟩

lemma *float-plus-minus*:
assumes *is-finite a is-finite b is-finite* $(a - b)$
shows $(a + -b) \doteq (a - b)$
⟨*proof*⟩

lemma *finite-bottomfloat*: *is-finite bottomfloat*
⟨*proof*⟩

lemma *bottomfloat-eq-m-largest*: *valof* (*bottomfloat*::('*e*, '*f*)*float*) = − *largest TYPE*(('*e*, '*f*)*float*)
if *LENGTH*('*e*) > *1*
⟨*proof*⟩

lemma *float-val-ge-bottomfloat*: *valof a* ≥ *valof* (*bottomfloat*::('*e*, '*f*)*float*)
if *LENGTH*('*e*) > *1 is-finite a*
for *a*::('*e*,'*f*)*float*
⟨*proof*⟩

lemma *float-ge-bottomfloat*: *is-finite a* ⟹ *a* ≥ *bottomfloat*
if *LENGTH*('*e*) > *1 is-finite a*
for *a*::('*e*,'*f*)*float*
⟨*proof*⟩

lemma *float-val-ge-largest*:
fixes *a*::('*e*,'*f*)*float*
assumes *LENGTH*('*e*) > *1 is-finite a*
shows *valof a* ≥ − *largest TYPE*(('*e*,'*f*)*float*)
⟨*proof*⟩

lemma *float-val-gt-threshold*:
fixes *a*::('*e*,'*f*)*float*
assumes *LENGTH*('*e*) > *1 is-finite a*
shows *valof a* > − *threshold TYPE*(('*e*,'*f*)*float*)
⟨*proof*⟩

Showing *abs* (− *a*) = *abs a*.

lemma *float-abs* [*simp*]: ¬ *is-nan a* ⟹ *abs* (− *a*) = *abs a*
⟨*proof*⟩

lemma *neg-zerosign*: − (*zerosign s a*) = *zerosign* (*1* − *s*) (− *a*)
⟨*proof*⟩

3.4 Properties about Rounding Errors

definition *error* :: ('*e*, '*f*)*float itself* ⇒ *real* ⇒ *real*
where *error* - *x* = *valof* (*round To-nearest x*::('*e*, '*f*)*float*) − *x*

lemma *bound-at-worst-lemma*:
fixes *a*::('*e*, '*f*)*float*
assumes *threshold*: |*x*| < *threshold TYPE*(('*e*, '*f*)*float*)
assumes *finite*: *is-finite a*
shows |*valof* (*round To-nearest x*::('*e*, '*f*)*float*) − *x*| ≤ |*valof a* − *x*|
⟨*proof*⟩

lemma *error-at-worst-lemma*:
fixes *a*::('*e*, '*f*)*float*

assumes *threshold*: $|x| < threshold\ TYPE(('e, 'f)float)$
and *is-finite a*
shows $|error\ TYPE(('e, 'f)float)\ x| \leq |valof\ a - x|$
⟨*proof*⟩

lemma *error-is-zero* [*simp*]:
fixes $a::('e, 'f)float$
assumes *is-finite a* $1 < LENGTH('e)$
shows $error\ TYPE(('e, 'f)float)\ (valof\ a) = 0$
⟨*proof*⟩

lemma *is-finite-zerosign*[*simp*]: $is\text{-}finite\ (zerosign\ s\ a) \longleftrightarrow is\text{-}finite\ a$
⟨*proof*⟩

lemma *is-finite-closest*: $is\text{-}finite\ (closest\ (v::-\Rightarrow real)\ p\ \{a.\ is\text{-}finite\ a\}\ x)$
⟨*proof*⟩

lemma *defloat-float-zerosign-round-finite*:
assumes *threshold*: $|x| < threshold\ TYPE(('e, 'f)float)$
shows $is\text{-}finite\ (zerosign\ s\ (round\ To\text{-}nearest\ x::('e, 'f)float))$
⟨*proof*⟩

lemma *valof-zero*[*simp*]: *valof 0 = 0 valof minus-zero = 0*
⟨*proof*⟩

lemma *signzero-zero*:
is-zero a $\Longrightarrow$ $valof\ (zerosign\ s\ a) = 0$
⟨*proof*⟩

lemma *val-zero*: *is-zero a* $\Longrightarrow$ *valof a = 0*
⟨*proof*⟩

lemma *float-add*:
fixes $a\ b::('e, 'f)float$
assumes *is-finite a*
and *is-finite b*
and *threshold*: $|valof\ a + valof\ b| < threshold\ TYPE(('e, 'f)float)$
shows *finite-float-add*: $is\text{-}finite\ (a + b)$
and *error-float-add*: $valof\ (a + b) = valof\ a + valof\ b + error\ TYPE(('e, 'f)float)\ (valof\ a + valof\ b)$
⟨*proof*⟩

lemma *float-sub*:
fixes $a\ b::('e, 'f)float$
assumes *is-finite a*
and *is-finite b*
and *threshold*: $|valof\ a - valof\ b| < threshold\ TYPE(('e, 'f)float)$
shows *finite-float-sub*: $is\text{-}finite\ (a - b)$
and *error-float-sub*: $valof\ (a - b) = valof\ a - valof\ b + error\ TYPE(('e,$

'f)float) (*valof a − valof b*)
⟨*proof*⟩

lemma *float-mul*:
fixes *a b*::(*'e*, *'f*)*float*
assumes *is-finite a*
and *is-finite b*
and *threshold*: |*valof a ∗ valof b*| < *threshold TYPE*((*'e*, *'f*)*float*)
shows *finite-float-mul*: *is-finite* (*a ∗ b*)
and *error-float-mul*: *valof* (*a ∗ b*) = *valof a ∗ valof b* + *error TYPE*((*'e*, *'f*)*float*)
(*valof a ∗ valof b*)
⟨*proof*⟩

lemma *float-div*:
fixes *a b*::(*'e*, *'f*)*float*
assumes *is-finite a*
and *is-finite b*
and *not-zero*: ¬ *is-zero b*
and *threshold*: |*valof a / valof b*| < *threshold TYPE*((*'e*, *'f*)*float*)
shows *finite-float-div*: *is-finite* (*a / b*)
and *error-float-div*: *valof* (*a / b*) = *valof a / valof b* + *error TYPE*((*'e*, *'f*)*float*)
(*valof a / valof b*)
⟨*proof*⟩

lemma *valof-one*[*simp*]: *valof* (*1* :: (*'e*, *'f*) *float*) = *of-bool* (*LENGTH*(*'e*) > *1*)
⟨*proof*⟩

end
theory *FP64*
imports
IEEE
Word-Lib.Word-64
begin

4 Concrete encodings

Floating point operations defined as operations on words. Called "fixed precision" (fp) word in HOL4.

type-synonym *float64* = (*11*,*52*)*float*
type-synonym *fp64* = *64 word*

lift-definition *fp64-of-float* :: *float64* ⇒ *fp64* **is**
λ(*s*::*1 word*, *e*::*11 word*, *f*::*52 word*). *word-cat s* (*word-cat e f*::*63 word*) ⟨*proof*⟩

lift-definition *float-of-fp64* :: *fp64* ⇒ *float64* **is**
λ*x*. *apsnd word-split* (*word-split x*::*1 word ∗ 63 word*) ⟨*proof*⟩

definition *rel-fp64* ≡ (λ*x* (*y*::*word64*). *x* = *float-of-fp64 y*)

definition *eq-fp64*::*float64* ⇒ *float64* ⇒ *bool* **where** [*simp*]: *eq-fp64* ≡ (=)

lemma *float-of-fp64-inverse*[*simp*]: *fp64-of-float* (*float-of-fp64* *a*) = *a*
⟨*proof*⟩

lemma *float-of-fp64-inj-iff*[*simp*]: *fp64-of-float* *r* = *fp64-of-float* *s* ⟷ *r* = *s*
⟨*proof*⟩

lemma *fp64-of-float-inverse*[*simp*]: *float-of-fp64* (*fp64-of-float* *a*) = *a*
⟨*proof*⟩

lemma *Quotientfp*: *Quotient* *eq-fp64* *fp64-of-float* *float-of-fp64* *rel-fp64*
— *eq-fp64* is a workaround to prevent a (failing – TODO: why?) code setup in *setup-lifting*.
⟨*proof*⟩

setup-lifting *Quotientfp*

lift-definition *fp64-lessThan*::*fp64* ⇒ *fp64* ⇒ *bool* **is**
flt::*float64*⇒*float64*⇒*bool* ⟨*proof*⟩

lift-definition *fp64-lessEqual*::*fp64* ⇒ *fp64* ⇒ *bool* **is**
fle::*float64*⇒*float64*⇒*bool* ⟨*proof*⟩

lift-definition *fp64-greaterThan*::*fp64* ⇒ *fp64* ⇒ *bool* **is**
fgt::*float64*⇒*float64*⇒*bool* ⟨*proof*⟩

lift-definition *fp64-greaterEqual*::*fp64* ⇒ *fp64* ⇒ *bool* **is**
fge::*float64*⇒*float64*⇒*bool* ⟨*proof*⟩

lift-definition *fp64-equal*::*fp64* ⇒ *fp64* ⇒ *bool* **is**
feq::*float64*⇒*float64*⇒*bool* ⟨*proof*⟩

lift-definition *fp64-abs*::*fp64* ⇒ *fp64* **is**
abs::*float64*⇒*float64* ⟨*proof*⟩

lift-definition *fp64-negate*::*fp64* ⇒ *fp64* **is**
uminus::*float64*⇒*float64* ⟨*proof*⟩

lift-definition *fp64-sqrt*::*roundmode* ⇒ *fp64* ⇒ *fp64* **is**
fsqrt::*roundmode*⇒*float64*⇒*float64* ⟨*proof*⟩

lift-definition *fp64-add*::*roundmode* ⇒ *fp64* ⇒ *fp64* ⇒ *fp64* **is**
fadd::*roundmode*⇒*float64*⇒*float64*⇒*float64* ⟨*proof*⟩

lift-definition *fp64-sub*::*roundmode* ⇒ *fp64* ⇒ *fp64* ⇒ *fp64* **is**
fsub::*roundmode*⇒*float64*⇒*float64*⇒*float64* ⟨*proof*⟩

lift-definition *fp64-mul*::*roundmode* ⇒ *fp64* ⇒ *fp64* ⇒ *fp64* **is**
fmul::*roundmode*⇒*float64*⇒*float64*⇒*float64* ⟨*proof*⟩

lift-definition *fp64-div*::*roundmode* ⇒ *fp64* ⇒ *fp64* ⇒ *fp64* **is**
fdiv::*roundmode*⇒*float64*⇒*float64*⇒*float64* ⟨*proof*⟩

lift-definition *fp64-mul-add*::*roundmode* ⇒ *fp64* ⇒ *fp64* ⇒ *fp64* ⇒ *fp64* **is**
fmul-add::*roundmode*⇒*float64*⇒*float64*⇒*float64*⇒*float64* ⟨*proof*⟩

end

theory *Conversion-IEEE-Float*
imports
HOL−Library.Float
IEEE-Properties
HOL−Library.Code-Target-Numeral
begin

definition *of-finite* (*x*::('*e*, '*f*)*float*) =
(*if is-normal x then* (*Float* (*normal-mantissa x*) (*normal-exponent x*))
else if is-denormal x then (*Float* (*denormal-mantissa x*) (*denormal-exponent*
TYPE(('*e*, '*f*)*float*)))
else 0)

lemma *float-val-of-finite*: *is-finite x* ⟹ *of-finite x* = *valof x*
⟨*proof*⟩

definition *is-normal-Float*::('*e*, '*f*)*float itself* ⇒ *Float.float* ⇒ *bool* **where**
is-normal-Float x f ⟷
mantissa f ≠ *0* ∧
bitlen |*mantissa f*| ≤ *fracwidth x* + *1* ∧
− *int* (*bias x*) − *bitlen* |*mantissa f*| + *1* < *Float.exponent f* ∧
Float.exponent f < *2*^(*LENGTH*('*e*)) − *bitlen* |*mantissa f*| − *bias x*

definition *is-denormal-Float*::('*e*, '*f*)*float itself* ⇒ *Float.float* ⇒ *bool* **where**
is-denormal-Float x f ⟷
mantissa f ≠ *0* ∧
bitlen |*mantissa f*| ≤ *1* − *Float.exponent f* − *int* (*bias x*) ∧
1 − *2*^(*LENGTH*('*e*) − *1*) − *int LENGTH*('*f*) < *Float.exponent f*

lemmas *is-denormal-FloatD* =
is-denormal-Float-def[*THEN iffD1*, *THEN conjunct1*]
is-denormal-Float-def[*THEN iffD1*, *THEN conjunct2*]

definition *is-finite-Float*::('*e*, '*f*)*float itself* ⇒ *Float.float* ⇒ *bool* **where**
is-finite-Float x f ⟷ *is-normal-Float x f* ∨ *is-denormal-Float x f* ∨ *f* = *0*

lemma *is-finite-Float-eq*:
is-finite-Float TYPE(('*e*, '*f*)*float*) *f* ⟷

(*let e = Float.exponent f; bm = bitlen (abs (mantissa f))*
in bm ≤ Suc LENGTH('f) ∧
bm ≤ 2 ^ (LENGTH('e) − 1) − e ∧
1 − 2 ^ (LENGTH('e) − 1) − int LENGTH('f) < e)
⟨*proof*⟩

lift-definition *normal-of-Float* :: *Float.float* ⇒ ('*e*, '*f*)*float*
is λ*x*. *let m = mantissa x*; *e = Float.exponent x in*
(*if m > 0 then 0 else 1*,
word-of-int (*e + int* (*bias TYPE*(('*e*, '*f*)*float*)) + *bitlen* |*m*| − *1*),
word-of-int (|*m*| ∗ *2 ^* (*Suc LENGTH*('*f*) − *nat* (*bitlen* |*m*|)) − *2 ^* (*LENGTH*('*f*))))
⟨*proof*⟩

lemma *sign-normal-of-Float*:*sign* (*normal-of-Float x*) = (*if x > 0 then 0 else 1*)
⟨*proof*⟩

lemma *uint-word-of-int-bitlen-eq*:
uint (*word-of-int x*::'*a*::*len word*) = *x* **if** *bitlen x* ≤ *LENGTH*('*a*) *x* ≥ *0*
⟨*proof*⟩

lemma *fraction-normal-of-Float*:*fraction* (*normal-of-Float x*::('*e*, '*f*)*float*) =
(*nat* |*mantissa x*| ∗ *2 ^* (*Suc LENGTH*('*f*) − *nat* (*bitlen* |*mantissa x*|)) − *2 ^*
LENGTH('*f*))
if *is-normal-Float TYPE*(('*e*, '*f*)*float*) *x*
⟨*proof*⟩

lemma *exponent-normal-of-Float*:*exponent* (*normal-of-Float x*::('*e*, '*f*)*float*) =
nat (*Float.exponent x* + (*bias TYPE*(('*e*, '*f*)*float*)) + *bitlen* |*mantissa x*| − *1*)
if *is-normal-Float TYPE*(('*e*, '*f*)*float*) *x*
⟨*proof*⟩

lift-definition *denormal-of-Float* :: *Float.float* ⇒ ('*e*, '*f*)*float*
is λ*x*. *let m = mantissa x*; *e = Float.exponent x in*
(*if m ≥ 0 then 0 else 1*, *0*,
word-of-int (|*m*| ∗ *2 ^ nat* (*e + bias TYPE*(('*e*, '*f*)*float*) + *fracwidth TYPE*(('*e*,
'*f*)*float*) − *1*)))
⟨*proof*⟩

lemma *sign-denormal-of-Float*:*sign* (*denormal-of-Float x*) = (*if x ≥ 0 then 0 else*
1)
⟨*proof*⟩

lemma *exponent-denormal-of-Float*:*exponent* (*denormal-of-Float x*::('*e*, '*f*)*float*) =
0
⟨*proof*⟩

lemma *fraction-denormal-of-Float*:*fraction* (*denormal-of-Float x*::('*e*, '*f*)*float*) =
(*nat* |*mantissa x*| ∗ *2 ^ nat* (*Float.exponent x + bias TYPE*(('*e*, '*f*)*float*) +
LENGTH('*f*) − *1*))

if *is-denormal-Float TYPE*(('*e*, '*f*)*float*) *x*
⟨*proof*⟩

definition *of-finite-Float* :: *Float.float* ⇒ ('*e*, '*f*) *float* **where**
of-finite-Float x = (*if is-normal-Float TYPE*(('*e*, '*f*)*float*) *x then normal-of-Float x*
else if is-denormal-Float TYPE(('*e*, '*f*)*float*) *x then denormal-of-Float x*
else 0)

lemma *valof-normal-of-Float*: *valof* (*normal-of-Float x*::('*e*, '*f*)*float*) = *x*
if *is-normal-Float TYPE*(('*e*, '*f*)*float*) *x*
⟨*proof*⟩

lemma *valof-denormal-of-Float*: *valof* (*denormal-of-Float x*::('*e*, '*f*)*float*) = *x*
if *is-denormal-Float TYPE*(('*e*, '*f*)*float*) *x*
⟨*proof*⟩

lemma *valof-of-finite-Float*:
is-finite-Float (*TYPE*(('*e*, '*f*) *IEEE.float*)) *x* ⟹ *valof* (*of-finite-Float x*::('*e*, '*f*)*float*) = *x*
⟨*proof*⟩

lemma *is-normal-normal-of-Float*:
is-normal (*normal-of-Float x*::('*e*, '*f*)*float*) **if** *is-normal-Float TYPE*(('*e*, '*f*)*float*) *x*
⟨*proof*⟩

lemma *is-denormal-denormal-of-Float*: *is-denormal* (*denormal-of-Float x*::('*e*, '*f*)*float*)
if *is-denormal-Float TYPE*(('*e*, '*f*)*float*) *x*
⟨*proof*⟩

lemma *is-finite-of-finite-Float*: *is-finite* (*of-finite-Float x*)
⟨*proof*⟩

lemma *Float-eq-zero-iff*: *Float m e* = *0* ⟷ *m* = *0*
⟨*proof*⟩

lemma *bitlen-mantissa-Float*:
shows *bitlen* |*mantissa* (*Float m e*)| = (*if m* = *0 then 0 else bitlen* |*m*| + *e*) − *Float.exponent* (*Float m e*)
⟨*proof*⟩

lemma *exponent-Float*:
shows *Float.exponent* (*Float m e*) = (*if m* = *0 then 0 else bitlen* |*m*| + *e*) − *bitlen* |*mantissa* (*Float m e*)|
⟨*proof*⟩

lemma *is-normal-Float-normal*:
is-normal-Float TYPE(('*e*, '*f*)*float*) (*Float* (*normal-mantissa x*) (*normal-exponent*

x))
  if *is-normal* x **for** x::$('e, 'f)$*float*
⟨*proof*⟩

lemma *is-denormal-Float-denormal*:
  is-denormal-Float TYPE$(('e, 'f)$*float*$)$
    (*Float* (*denormal-mantissa* x) (*denormal-exponent TYPE*$(('e, 'f)$*float*$)$))
  if *is-denormal* x **for** x::$('e, 'f)$*float*
⟨*proof*⟩

lemma *is-finite-Float-of-finite*: *is-finite-Float TYPE*$(('e, 'f)$*float*$)$ (*of-finite* x) **for** x::$('e, 'f)$*float*
  ⟨*proof*⟩

end

5 Code Generation Setup for Floats

theory *Double*
  imports
    Conversion-IEEE-Float
    HOL−Library.Monad-Syntax
    HOL−Library.Code-Target-Numeral
begin

A type for code generation to SML/OCaml

typedef *double* = *UNIV*::$(11, 52)$ *float set* ⟨*proof*⟩

setup-lifting *type-definition-double*

instantiation *double* :: {*uminus*,*plus*,*times*,*minus*,*zero*,*one*,*abs*,*ord*,*inverse*} **begin**
lift-definition *uminus-double*::*double* ⇒ *double* **is** *uminus* ⟨*proof*⟩
lift-definition *plus-double*::*double* ⇒ *double* ⇒ *double* **is** *plus* ⟨*proof*⟩
lift-definition *times-double*::*double* ⇒ *double* ⇒ *double* **is** *times* ⟨*proof*⟩
lift-definition *divide-double*::*double* ⇒ *double* ⇒ *double* **is** *divide* ⟨*proof*⟩
lift-definition *inverse-double*::*double* ⇒ *double* **is** *inverse* ⟨*proof*⟩
lift-definition *minus-double*::*double* ⇒ *double* ⇒ *double* **is** *minus* ⟨*proof*⟩
lift-definition *zero-double*::*double* **is** *0* ⟨*proof*⟩
lift-definition *one-double*::*double* **is** *1* ⟨*proof*⟩
lift-definition *less-eq-double*::*double* ⇒ *double* ⇒ *bool* **is** (≤) ⟨*proof*⟩
lift-definition *less-double*::*double* ⇒ *double* ⇒ *bool* **is** (<) ⟨*proof*⟩
instance ⟨*proof*⟩
end

lift-definition *eq-double*::*double*⇒*double*⇒*bool* **is** *float-eq* ⟨*proof*⟩

lift-definition *sqrt-double*::*double*⇒*double* **is** *float-sqrt* ⟨*proof*⟩

no-notation *plus-infinity* (∞)

lift-definition *infinity-double*::*double* (∞) **is** *plus-infinity* ⟨*proof*⟩

lift-definition *is-normal*::*double* ⇒ *bool* **is** *IEEE.is-normal* ⟨*proof*⟩
lift-definition *is-zero*::*double* ⇒ *bool* **is** *IEEE.is-zero* ⟨*proof*⟩
lift-definition *is-finite*::*double* ⇒ *bool* **is** *IEEE.is-finite* ⟨*proof*⟩
lift-definition *is-nan*::*double* ⇒ *bool* **is** *IEEE.is-nan* ⟨*proof*⟩

code-printing type-constructor *double* ⇀
(*SML*) *real* **and** (*OCaml*) *float*

code-printing constant *0* :: *double* ⇀
(*SML*) *0.0* **and** (*OCaml*) *0.0*
declare *zero-double.abs-eq*[*code del*]

code-printing constant *1* :: *double* ⇀
(*SML*) *1.0* **and** (*OCaml*) *1.0*
declare *one-double.abs-eq*[*code del*]

code-printing constant *eq-double* :: *double* ⇒ *double* ⇒ *bool* ⇀
(*SML*) *Real.*== ((-:*real*), (-)) **and** (*OCaml*) *Pervasives.*(=)
declare *eq-double.abs-eq*[*code del*]

code-printing constant *Orderings.less-eq* :: *double* ⇒ *double* ⇒ *bool* ⇀
(*SML*) *Real.*<= ((-), (-)) **and** (*OCaml*) *Pervasives.*(<=)
declare *less-double-def* [*code del*]

code-printing constant *Orderings.less* :: *double* ⇒ *double* ⇒ *bool* ⇀
(*SML*) *Real.*< ((-), (-)) **and** (*OCaml*) *Pervasives.*(<)
declare *less-eq-double-def*[*code del*]

code-printing constant (+) :: *double* ⇒ *double* ⇒ *double* ⇀
(*SML*) *Real.*+ ((-), (-)) **and** (*OCaml*) *Pervasives.*(+.)
declare *plus-double-def*[*code del*]

code-printing constant (∗) :: *double* ⇒ *double* ⇒ *double* ⇀
(*SML*) *Real.*∗ ((-), (-)) **and** (*OCaml*) *Pervasives.*(∗.)
declare *times-double-def* [*code del*]

code-printing constant (−) :: *double* ⇒ *double* ⇒ *double* ⇀
(*SML*) *Real.*− ((-), (-)) **and** (*OCaml*) *Pervasives.*(−.)
declare *minus-double-def* [*code del*]

code-printing constant *uminus* :: *double* ⇒ *double* ⇀
(*SML*) *Real.*~ **and** (*OCaml*) *Pervasives.*(~−.)

code-printing constant (/) :: *double* ⇒ *double* ⇒ *double* ⇀
(*SML*) *Real.*′/ ((-), (-)) **and** (*OCaml*) *Pervasives.*(′/.)

declare *divide-double-def* [*code del*]

code-printing constant *sqrt-double* :: *double* ⇒ *double* ⇀
(*SML*) *Math.sqrt* **and** (*OCaml*) *Pervasives.sqrt*
declare *sqrt-def*[*code del*]

code-printing constant *infinity-double* :: *double* ⇀
(*SML*) *Real.posInf*
declare *infinity-double.abs-eq*[*code del*]

code-printing constant *is-normal* :: *double* ⇒ *bool* ⇀
(*SML*) *Real.isNormal*
declare [[*code drop*: *is-normal*]]

code-printing constant *is-finite* :: *double* ⇒ *bool* ⇀
(*SML*) *Real.isFinite*
declare [[*code drop*: *is-finite*]]

code-printing constant *is-nan* :: *double* ⇒ *bool* ⇀
(*SML*) *Real.isNan*
declare [[*code drop*: *is-nan*]]

Mapping natural numbers to doubles.

fun *float-of* :: *nat* ⇒ *double*
where
float-of 0 = *0*
| *float-of* (*Suc n*) = *float-of n* + *1*

lemma *float-of 2* < *float-of 3* + *float-of 4*
⟨*proof*⟩

export-code *float-of* **in** *SML*

5.1 Conversion from int

lift-definition *double-of-int*::*int* ⇒ *double* **is** λ*i*. *round To-nearest* (*real-of-int i*)
⟨*proof*⟩

context includes *integer.lifting* **begin**
lift-definition *double-of-integer*::*integer* ⇒ *double* **is** *double-of-int* ⟨*proof*⟩
end

lemma *float-of-int*[*code*]:
double-of-int i = *double-of-integer* (*integer-of-int i*)
⟨*proof*⟩

code-printing
constant *double-of-integer* :: *integer* ⇒ *double* ⇀ (*SML*) *Real.fromLargeInt*
declare [[*code drop*: *double-of-integer*]]

5.2 Conversion to and from software floats, extracting information

Need to trust a lot of code here...

lemma *is-finite-double-eq*:
is-finite-Float TYPE((*11*, *52*)*float*) *f* ⟷
(*let e* = *Float.exponent f*; *bm* = *bitlen* (*abs* (*mantissa f*))
in (*bm* ≤ *53* ∧ *e* + *bm* < *1025* ∧ − *1075* < *e*))
⟨*proof*⟩

code-printing
code-module *IEEE-Mantissa-Exponent* ⇀ (*SML*)
‹
structure IEEE-Mantissa-Exponent =
struct
fun to-man-exp-double x =
if Real.isFinite x
then case Real.toManExp x of {*man* = *m*, *exp* = *e*} =>
SOME (*Real.floor* (*Real.** (*m*, *Math.pow* (*2.0*, *53.0*))), *IntInf.*− (*e*, *53*))
else NONE
fun normfloat (*m*, *e*) =
(*if m mod 2* = *0 andalso m* <> *0 then normfloat* (*m div 2*, *e* + *1*)
else if m = *0 then* (*0*, *0*) *else* (*m*, *e*))
fun bitlen x = (*if 0* < *x then bitlen* (*x div 2*) + *1 else 0*)
fun is-finite-double-eq m e =
let
val (*m*, *e*) = *normfloat* (*m*, *e*)
val bm = *bitlen* (*abs m*)
in bm <= *53 andalso e* + *bm* < *1025 andalso e* > ~*1075 end*
fun from-man-exp-double m e =
if is-finite-double-eq m e
then SOME (*Real.fromManExp* {*man* = *Real.fromLargeInt m*, *exp* = *e*})
else NONE
end
›

lift-definition *of-finite*::*double* ⇒ *Float.float* **is** *Conversion-IEEE-Float.of-finite*
⟨*proof*⟩

definition *man-exp-of-double*::*double* ⇒ (*integer* ∗ *integer*)*option* **where**
man-exp-of-double d = (*if is-finite d then let f* = *of-finite d in*
Some (*integer-of-int* (*mantissa f*), *integer-of-int* (*Float.exponent f*)) *else None*)

lift-definition *of-finite-Float*::*Float.float* ⇒ *double* **is** *Conversion-IEEE-Float.of-finite-Float*
⟨*proof*⟩

definition *double-of-man-exp*::*integer* ⇒ *integer* ⇒ *double option* **where**
double-of-man-exp m e = (*let f* = *Float* (*int-of-integer m*) (*int-of-integer e*) *in*
if is-finite-Float TYPE((*11*, *52*)*float*) *f*

then Some (of-finite-Float f)
else None)

code-printing
constant *man-exp-of-double* :: *double* ⇒ (*integer* ∗ *integer*) *option* ⇀
(*SML*) *IEEE′-Mantissa′-Exponent.to′-man′-exp′-double* |
constant *double-of-man-exp* :: *integer* ⇒ *integer* ⇒ *double option* ⇀
(*SML*) *IEEE′-Mantissa′-Exponent.from′-man′-exp′-double*
declare [[*code drop*: *man-exp-of-double*]]
declare [[*code drop*: *double-of-man-exp*]]

lift-definition *Float-of-double*::*double* ⇒ *Float.float option* **is**
λx. if is-finite x then Some (of-finite x) else None ⟨*proof*⟩

lift-definition *double-of-Float*::*Float.float* ⇒ *double option* **is**
λx. if is-finite-Float TYPE((*11*, *52*)*float*) *x then Some* (*of-finite-Float x*) *else None* ⟨*proof*⟩

lemma *compute-Float-of-double*[*code*]:
Float-of-double x =
map-option (*λ*(*m*, *e*). *Float* (*int-of-integer m*) (*int-of-integer e*)) (*man-exp-of-double x*)
⟨*proof*⟩

lemma *compute-double-of-Float*[*code*]:
double-of-Float f = *double-of-man-exp* (*integer-of-int* (*mantissa f*)) (*integer-of-int* (*Float.exponent f*))
⟨*proof*⟩

definition *check-conversion m e* =
(*let f* = *Float* (*int-of-integer m*) (*int-of-integer e*) *in*
do {
d ← *double-of-Float f*;
Float-of-double d
} = (*if is-finite-Float TYPE*((*11*, *52*)*float*) *f then Some f else None*))

primrec *check-all*::*nat* ⇒ (*nat* ⇒ *bool*) ⇒ *bool* **where**
check-all 0 P ⟷ *True*
| *check-all* (*Suc i*) *P* ⟷ *P i* ∧ *check-all i P*

definition *check-conversions dm de* =
check-all (*nat* (*2* ∗ *de*)) (*λe. check-all* (*nat* (*2* ∗ *dm*)) (*λm.*
check-conversion (*integer-of-int* (*int m* − *dm*)) (*integer-of-int* (*int e* − *de*))))

lemma *check-conversions 100 100*
⟨*proof*⟩

end

References

[1] J. Harrison. *Floating point verification in HOL light: the exponential function*. Springer, 1997.