

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	2
1.1	Derived parameters for floating point formats	2
1.2	Predicates for the four IEEE formats	3
1.3	Extractors for fields	3
1.4	Partition of numbers into disjoint classes	3
1.5	Special values	4
1.6	Negation operation on floating point values	4
1.7	Real number valuations	4
1.8	Rounding	5
1.9	Definitions of the arithmetic operations	7
1.10	Comparison operations	8
2	Specify float to be double precision and round to even	9
3	Proofs of Properties about Floating Point Arithmetic	11
3.1	Theorems derived from definitions	11
3.2	Properties about ordering and bounding	15
3.3	Algebraic properties about basic arithmetic	25
3.4	Properties about Rounding Errors	28

4	Concrete encodings	33
5	Code Generation Setup for Floats	42
5.1	Conversion from int	44
5.2	Conversion to and from software floats, extracting information	45

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
  by auto

  setup-lifting type-definition-float

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

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

  parse-translation ⟨
    let
      fun float t u = Syntax.const @{type-syntax float} $ t $ u;
      fun len-tr u =
        (case Term-Position.strip-positions u of
          v as Free (x, -) =>
            if Lexicon.is-tid x then
              (Syntax.const @{syntax-const -ofsort} $ v $
                Syntax.const @{class-syntax len})
            else u
          | - => u)
      fun len-float-tr [t, u] =
        float (len-tr t) (len-tr u)
    in
      [(@{syntax-const -float}, K len-float-tr)]
    end
  ⟩

```

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 $emax :: ('e, 'f) \text{ float itself} \Rightarrow \text{nat}$
where $emax\ x = \text{unat } (-1 :: 'e \text{ word})$

abbreviation $fracwidth :: ('e, 'f) \text{ float itself} \Rightarrow \text{nat}$ **where**
 $fracwidth - \equiv \text{LENGTH}('f)$

1.2 Predicates for the four IEEE formats

definition $is_single :: ('e, 'f) \text{ float itself} \Rightarrow \text{bool}$
where $is_single\ x \longleftrightarrow \text{LENGTH}('e) = 8 \wedge \text{wordlength } x = 32$

definition $is_double :: ('e, 'f) \text{ float itself} \Rightarrow \text{bool}$
where $is_double\ x \longleftrightarrow \text{LENGTH}('e) = 11 \wedge \text{wordlength } x = 64$

definition $is_single_extended :: ('e, 'f) \text{ float itself} \Rightarrow \text{bool}$
where $is_single_extended\ x \longleftrightarrow \text{LENGTH}('e) \geq 11 \wedge \text{wordlength } x \geq 43$

definition $is_double_extended :: ('e, 'f) \text{ float itself} \Rightarrow \text{bool}$
where $is_double_extended\ x \longleftrightarrow \text{LENGTH}('e) \geq 15 \wedge \text{wordlength } x \geq 79$

1.3 Extractors for fields

lift-definition $sign :: ('e, 'f) \text{ float} \Rightarrow \text{nat}$ **is**
 $\lambda(s :: 1 \text{ word}, - :: 'e \text{ word}, - :: 'f \text{ word}). \text{unat } s .$

lift-definition $exponent :: ('e, 'f) \text{ float} \Rightarrow \text{nat}$ **is**
 $\lambda(-, e :: 'e \text{ word}, -). \text{unat } e .$

lift-definition $fraction :: ('e, 'f) \text{ float} \Rightarrow \text{nat}$ **is**
 $\lambda(-, -, f :: 'f \text{ word}). \text{unat } f .$

abbreviation $real\text{-of}\text{-word } x \equiv \text{real } (\text{unat } x)$

lift-definition $valof :: ('e, 'f) \text{ float} \Rightarrow \text{real}$
is $\lambda(s, e, f).$
 $\text{let } x = (\text{TYPE} (('e, 'f) \text{ float})) \text{ in}$
 $(\text{if } e = 0$
 $\text{then } (-1 :: \text{real}) \wedge (\text{unat } s) * (2 / (2 \wedge \text{bias } x)) * (\text{real-of-word } f / 2 \wedge (\text{LENGTH}('f)))$
 $\text{else } (-1 :: \text{real}) \wedge (\text{unat } s) * ((2 \wedge (\text{unat } e)) / (2 \wedge \text{bias } x)) * (1 + \text{real-of-word}$
 $f / 2 \wedge \text{LENGTH}('f)))$
 $.$

1.4 Partition of numbers into disjoint classes

definition $is_nan :: ('e, 'f) \text{ float} \Rightarrow \text{bool}$
where $is_nan\ a \longleftrightarrow \text{exponent } a = emax\ \text{TYPE} (('e, 'f) \text{ float}) \wedge \text{fraction } a \neq 0$

definition $is_infinity :: ('e, 'f) \text{ float} \Rightarrow \text{bool}$

where *is-infinity* $a \longleftrightarrow \text{exponent } a = \text{emax } \text{TYPE}((\text{'e}, \text{'f})\text{float}) \wedge \text{fraction } a = 0$

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

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

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

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

1.5 Special values

lift-definition *plus-infinity* $:: (\text{'e}, \text{'f}) \text{float } (\infty) \text{ is } (0, -1, 0)$.

lift-definition *topfloat* $:: (\text{'e}, \text{'f}) \text{float is } (0, -2, 2^{\text{LENGTH}(\text{'f}) - 1})$.

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

lift-definition *zero-float* $:: (\text{'e}, \text{'f}) \text{float is } (0, 0, 0)$.

instance proof qed

end

1.6 Negation operation on floating point values

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

lift-definition *uminus-float* $:: (\text{'e}, \text{'f}) \text{float} \Rightarrow (\text{'e}, \text{'f}) \text{float is } \lambda(s, e, f). (1 - s, e, f)$.

instance proof qed

end

abbreviation (*input*) *minus-zero* $\equiv - (0::(\text{'e}, \text{'f})\text{float})$

abbreviation (*input*) *minus-infinity* $\equiv - \infty$

abbreviation (*input*) *bottomfloat* $\equiv - \text{topfloat}$

1.7 Real number valuations

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

definition *largest* $:: (\text{'e}, \text{'f}) \text{float itself} \Rightarrow \text{real}$

where *largest* $x = (2^{\text{emax } x - 1} / 2^{\text{bias } x}) * (2 - 1/(2^{\text{fracwidth } x}))$

Threshold, used for checking overflow.

definition *threshold* $:: (\text{'e}, \text{'f}) \text{float itself} \Rightarrow \text{real}$

where *threshold* $x = (2^{\wedge}(\text{emax } x - 1) / 2^{\wedge}\text{bias } x) * (2 - 1 / (2^{\wedge}(\text{Suc}(\text{fracwidth } x))))$

Unit of least precision.

lift-definition *one-lp*::('e ,f) float \Rightarrow ('e ,f) float **is** $\lambda(s, e, f). (0, e::'e \text{ word}, 1)$

.

lift-definition *zero-lp*::('e ,f) float \Rightarrow ('e ,f) float **is** $\lambda(s, e, f). (0, e::'e \text{ word}, 0)$

.

definition *ulp* :: ('e, 'f) float \Rightarrow real **where** *ulp* $a = \text{valof } (\text{one-lp } a) - \text{valof } (\text{zero-lp } a)$

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* $v \ p \ s \ x =$

$(\text{SOME } a. \text{is-closest } v \ s \ x \ a \wedge ((\exists b. \text{is-closest } v \ s \ x \ b \wedge p \ b) \longrightarrow p \ a))$

1.8 Rounding

fun *round* :: *roundmode* \Rightarrow real \Rightarrow ('e ,f) float

where

round To-nearest $y =$

$(\text{if } y \leq -\text{threshold } \text{TYPE}((\text{'e ,f}) \text{ float}) \text{ then } \text{minus-infinity}$
 $\text{else if } y \geq \text{threshold } \text{TYPE}((\text{'e ,f}) \text{ float}) \text{ then } \text{plus-infinity}$
 $\text{else } \text{closest } (\text{valof } (\lambda a. \text{even } (\text{fraction } a)) \{a. \text{is-finite } a\} \ y)$

| *round float-To-zero* $y =$

$(\text{if } y < -\text{largest } \text{TYPE}((\text{'e ,f}) \text{ float}) \text{ then } \text{bottomfloat}$
 $\text{else if } y > \text{largest } \text{TYPE}((\text{'e ,f}) \text{ float}) \text{ then } \text{topfloat}$
 $\text{else } \text{closest } (\text{valof } (\lambda a. \text{True}) \{a. \text{is-finite } a \wedge |\text{valof } a| \leq |y|\} \ y)$

| *round To-pinfinity* $y =$

$(\text{if } y < -\text{largest } \text{TYPE}((\text{'e ,f}) \text{ float}) \text{ then } \text{bottomfloat}$
 $\text{else if } y > \text{largest } \text{TYPE}((\text{'e ,f}) \text{ float}) \text{ then } \text{plus-infinity}$
 $\text{else } \text{closest } (\text{valof } (\lambda a. \text{True}) \{a. \text{is-finite } a \wedge \text{valof } a \geq y\} \ y)$

| *round To-ninfinity* $y =$

$(\text{if } y < -\text{largest } \text{TYPE}((\text{'e ,f}) \text{ float}) \text{ then } \text{minus-infinity}$
 $\text{else if } y > \text{largest } \text{TYPE}((\text{'e ,f}) \text{ float}) \text{ then } \text{topfloat}$
 $\text{else } \text{closest } (\text{valof } (\lambda a. \text{True}) \{a. \text{is-finite } a \wedge \text{valof } a \leq y\} \ y)$

Rounding to integer values in floating point format.

definition *is-integral* :: ('e ,f) float \Rightarrow bool

where *is-integral* $a \longleftrightarrow \text{is-finite } a \wedge (\exists n::\text{nat}. |\text{valof } a| = \text{real } n)$

fun *intround* :: *roundmode* \Rightarrow real \Rightarrow ('e ,f) float

where

intround To-nearest $y =$
 (*if* $y \leq -\text{threshold}$ $\text{TYPE}('e, 'f)$ *float*) *then* *minus-infinity*
 else if $y \geq \text{threshold}$ $\text{TYPE}('e, 'f)$ *float*) *then* *plus-infinity*
 else *closest* (*valof*) $(\lambda a. (\exists n::\text{nat. even } n \wedge |\text{valof } a| = \text{real } n)) \{a. \text{is-integral } a\} y)$

intround float-To-zero $y =$
 (*if* $y < -\text{largest}$ $\text{TYPE}('e, 'f)$ *float*) *then* *bottomfloat*
 else if $y > \text{largest}$ $\text{TYPE}('e, 'f)$ *float*) *then* *topfloat*
 else *closest* (*valof*) $(\lambda x. \text{True}) \{a. \text{is-integral } a \wedge |\text{valof } a| \leq |y|\} y)$

intround To-pinfinity $y =$
 (*if* $y < -\text{largest}$ $\text{TYPE}('e, 'f)$ *float*) *then* *bottomfloat*
 else if $y > \text{largest}$ $\text{TYPE}('e, 'f)$ *float*) *then* *plus-infinity*
 else *closest* (*valof*) $(\lambda x. \text{True}) \{a. \text{is-integral } a \wedge \text{valof } a \geq y\} y)$

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

Round, choosing between -0.0 or +0.0

definition *float-round* $:: \text{roundmode} \Rightarrow \text{bool} \Rightarrow \text{real} \Rightarrow ('e, 'f)$ *float*
 where *float-round mode toneg* $r =$
 (*let* $x = \text{round mode } r$ *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* $= (\text{SOME } a. \text{is-nan } a)$

Coercion for signs of zero results.

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

definition *frem* $:: \text{roundmode} \Rightarrow ('e, 'f)$ *float* $\Rightarrow ('e, 'f)$ *float* $\Rightarrow ('e, 'f)$ *float*
 where *frem* $m a b =$
 (*if* *is-nan* $a \vee \text{is-nan } b \vee \text{is-infinity } a \vee \text{is-zero } b$ *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-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* \wedge *is-zero* *b* \wedge *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* \Rightarrow ('e , 'f) float \Rightarrow ('e , 'f) float \Rightarrow ('e , 'f) float

where *fsub* *m a b* =
 (if *is-nan* *a* \vee *is-nan* *b* \vee (*is-infinity* *a* \wedge *is-infinity* *b* \wedge *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* \wedge *is-zero* *b* \wedge *sign* *a* \neq *sign* *b* then *sign* *a*
 else if *m* = *To-ninfinity* then 1 else 0)
 (*round* *m* (*valof* *a* - *valof* *b*)))

definition *fmul* :: *roundmode* \Rightarrow ('e , 'f) float \Rightarrow ('e , 'f) float \Rightarrow ('e , 'f) float

where *fmul* *m a b* =
 (if *is-nan* *a* \vee *is-nan* *b* \vee (*is-zero* *a* \wedge *is-infinity* *b*) \vee (*is-infinity* *a* \wedge *is-zero* *b*)
 then some-nan
 else if *is-infinity* *a* \vee *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* \Rightarrow ('e , 'f) float \Rightarrow ('e , 'f) float \Rightarrow ('e , 'f) float

where *fdiv* *m a b* =
 (if *is-nan* *a* \vee *is-nan* *b* \vee (*is-zero* *a* \wedge *is-zero* *b*) \vee (*is-infinity* *a* \wedge *is-infinity* *b*)
 then some-nan
 else if *is-infinity* *a* \vee *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 $flt\ a\ b \longleftrightarrow fcompare\ a\ b = Lt$

definition $fle :: ('e, 'f)\ float \Rightarrow ('e, 'f)\ float \Rightarrow bool$
where $fle\ a\ b \longleftrightarrow fcompare\ a\ b = Lt \vee fcompare\ a\ b = Eq$

definition $fgt :: ('e, 'f)\ float \Rightarrow ('e, 'f)\ float \Rightarrow bool$
where $fgt\ a\ b \longleftrightarrow fcompare\ a\ b = Gt$

definition $fge :: ('e, 'f)\ float \Rightarrow ('e, 'f)\ float \Rightarrow bool$
where $fge\ a\ b \longleftrightarrow fcompare\ a\ b = Gt \vee fcompare\ a\ b = Eq$

definition $feq :: ('e, 'f)\ float \Rightarrow ('e, 'f)\ float \Rightarrow bool$
where $feq\ a\ b \longleftrightarrow fcompare\ a\ b = Eq$

2 Specify float to be double precision and round to even

instantiation $float :: (len, len)\ plus$
begin

definition $plus\text{-}float :: ('a, 'b)\ float \Rightarrow ('a, 'b)\ float \Rightarrow ('a, 'b)\ float$
where $a + b = fadd\ To\text{-}nearest\ a\ b$

instance ..

end

instantiation $float :: (len, len)\ minus$
begin

definition $minus\text{-}float :: ('a, 'b)\ float \Rightarrow ('a, 'b)\ float \Rightarrow ('a, 'b)\ float$
where $a - b = fsub\ To\text{-}nearest\ a\ b$

instance ..

end

instantiation $float :: (len, len)\ times$
begin

definition $times\text{-}float :: ('a, 'b)\ float \Rightarrow ('a, 'b)\ float \Rightarrow ('a, 'b)\ float$
where $a * b = fmul\ To\text{-}nearest\ a\ b$

instance ..

end

instantiation $float :: (len, len)\ one$

```

begin

lift-definition one-float :: ('a, 'b) float is (0, 2^(LENGTH('a) - 1) - 1, 0) .

instance ..

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 ..

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 ..

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 \Rightarrow ('a, 'b) float
where *abs-float* a = (if sign a = 0 then a else - a)

instance ..
end

The $1 + \varepsilon$ property.

definition *normalizes* :: - itself \Rightarrow real \Rightarrow bool
where *normalizes float-format* x =
 (1 / (2::real) ^ (bias float-format - 1)) \leq |x| \wedge |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
unfolding *Let-def*
by transfer (auto simp: bias-def divide-simps unat-eq-0)

lemma *exponent-le* [*simp*]:

\langle exponent a \leq mask LENGTH('a) \rangle **for** a :: \langle ('a, -) float \rangle
by transfer (auto simp add: of-nat-mask-eq intro: word-unat-less-le split: prod.split)

lemma *exponent-not-less* [*simp*]:

\langle \neg mask LENGTH('a) < IEEE.exponent a \rangle **for** a :: \langle ('a, -) float \rangle
by (simp add: not-less)

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
subgoal by *transfer auto*
subgoal by *transfer auto*
subgoal by *transfer (simp add: emax-def mask-eq-exp-minus-1)*
subgoal by *transfer (simp add: emax-def mask-eq-exp-minus-1)*
subgoal by *transfer auto*
subgoal by *transfer auto*
done

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
subgoal by *transfer auto*
subgoal by *transfer auto*
subgoal by *transfer auto*
subgoal by *transfer auto*
subgoal by *transfer auto*
subgoal by *transfer auto*
done

lemma *emax-eq: emax x = 2 ^ LENGTH('e) - 1*

for *x::('e, 'f)float itself*
by *(simp add: emax-def unsigned-minus-1-eq-mask mask-eq-exp-minus-1)*

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
subgoal by *transfer simp*
subgoal by *transfer (simp add: emax-eq take-bit-minus-small-eq nat-diff-distrib nat-power-eq)*
subgoal by *transfer (simp add: unsigned-minus-1-eq-mask mask-eq-exp-minus-1)*
subgoal by *transfer simp*
subgoal by *transfer (simp add: emax-eq take-bit-minus-small-eq nat-diff-distrib nat-power-eq)*
subgoal by *transfer (simp add: unsigned-minus-1-eq-mask mask-eq-exp-minus-1)*
done

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*: $is\text{-}nan\ a \vee is\text{-}infinity\ a \vee is\text{-}normal\ a \vee is\text{-}denormal\ a \vee is\text{-}zero\ a$
by (*auto simp add: float-defs not-less le-less emax-def unsigned-minus-1-eq-mask*)

lemma *float-cases-finite*: $is\text{-}nan\ a \vee is\text{-}infinity\ a \vee is\text{-}finite\ a$
by (*simp add: float-cases is-finite-def*)

lemma *float-zero1*[*simp*]: $is\text{-}zero\ 0$
unfolding *float-defs*
by *transfer auto*

lemma *float-zero2*[*simp*]: $is\text{-}zero\ (-\ x) \longleftrightarrow is\text{-}zero\ x$
unfolding *float-defs*
by *transfer auto*

lemma *emax-pos*[*simp*]: $0 < emax\ x\ emax\ x \neq 0$
by (*auto simp: emax-def*)

The types of floating-point numbers are mutually distinct.

lemma *float-distinct*:
 $\neg (is\text{-}nan\ a \wedge is\text{-}infinity\ a)$
 $\neg (is\text{-}nan\ a \wedge is\text{-}normal\ a)$
 $\neg (is\text{-}nan\ a \wedge is\text{-}denormal\ a)$
 $\neg (is\text{-}nan\ a \wedge is\text{-}zero\ a)$
 $\neg (is\text{-}infinity\ a \wedge is\text{-}normal\ a)$
 $\neg (is\text{-}infinity\ a \wedge is\text{-}denormal\ a)$
 $\neg (is\text{-}infinity\ a \wedge is\text{-}zero\ a)$
 $\neg (is\text{-}normal\ a \wedge is\text{-}denormal\ a)$
 $\neg (is\text{-}denormal\ a \wedge is\text{-}zero\ a)$
by (*auto simp: float-defs*)

lemma *denormal-imp-not-zero*: $is\text{-}denormal\ f \implies \neg is\text{-}zero\ f$
by (*simp add: is-denormal-def is-zero-def*)

lemma *normal-imp-not-zero*: $is\text{-}normal\ f \implies \neg is\text{-}zero\ f$
by (*simp add: is-normal-def is-zero-def*)

lemma *normal-imp-not-denormal*: $is\text{-}normal\ f \implies \neg is\text{-}denormal\ f$
by (*simp add: is-normal-def is-denormal-def*)

lemma *denormal-zero[simp]*: \neg is-denormal 0 \neg is-denormal minus-zero
using *denormal-imp-not-zero float-zero1 float-zero2* **by** *blast+*

lemma *normal-zero[simp]*: \neg is-normal 0 \neg is-normal minus-zero
using *normal-imp-not-zero float-zero1 float-zero2* **by** *blast+*

lemma *float-distinct-finite*: \neg (is-nan a \wedge is-finite a) \neg (is-infinity a \wedge is-finite a)
by (*auto simp: float-defs*)

lemma *finite-infinity*: is-finite a \implies \neg is-infinity a
by (*auto simp: float-defs*)

lemma *finite-nan*: is-finite a \implies \neg is-nan a
by (*auto simp: float-defs*)

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

lemma *is-closest-exists*:
fixes *v* :: ('e, 'f)float \Rightarrow real
and *s* :: ('e, 'f)float set
assumes *finite*: finite *s*
and *non-empty*: *s* \neq {}
shows $\exists a$. is-closest *v s x a*
using *finite non-empty*
proof (*induct s rule: finite-induct*)
case *empty*
then show ?*case* **by** *simp*
next
case (*insert z s*)
show ?*case*
proof (*cases s = {}*)
case *True*
then have is-closest *v (insert z s) x z*
by (*auto simp: is-closest-def*)
then show ?*thesis* **by** *metis*
next
case *False*
then obtain a where *a*: is-closest *v s x a*
using *insert* **by** *metis*
then show ?*thesis*
proof (*cases |v a - x| |v z - x| rule: le-cases*)
case *le*
then show ?*thesis*
by (*metis insert-iff a is-closest-def*)
next
case *ge*
have $\forall b. b \in s \implies |v a - x| \leq |v b - x|$
by (*metis a is-closest-def*)
then have $\forall b. b \in \text{insert } z \text{ } s \implies |v z - x| \leq |v b - x|$
by (*metis eq-iff ge insert-iff order.trans*)

```

    then show ?thesis using is-closest-def a
      by (metis insertI1)
  qed
qed
qed

lemma closest-is-everything:
  fixes v :: ('e, 'f)float  $\Rightarrow$  real
  and s :: ('e, 'f)float set
  assumes finite: finite s
  and non-empty: s  $\neq$  {}
  shows is-closest v s x (closest v p s x)  $\wedge$ 
    (( $\exists$  b. is-closest v s x b  $\wedge$  p b)  $\longrightarrow$  p (closest v p s x))
  unfolding closest-def
  by (rule someI-ex) (metis assms is-closest-exists [of s v x])

```

```

lemma closest-in-set:
  fixes v :: ('e, 'f)float  $\Rightarrow$  real
  assumes finite s and s  $\neq$  {}
  shows closest v p s x  $\in$  s
  by (metis assms closest-is-everything is-closest-def)

```

```

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)
  by (metis closest-is-everything assms)

```

```

instance float::(len, len) finite proof qed (transfer, simp)

```

```

lemma is-finite-nonempty: {a. is-finite a}  $\neq$  {}
proof -
  have 0  $\in$  {a. is-finite a}
  unfolding float-defs
  by transfer auto
  then show ?thesis by (metis empty-iff)
qed

```

```

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)
  by (rule closest-is-closest-finite) (auto simp: assms)

```

3.2 Properties about ordering and bounding

Lifting of non-exceptional comparisons.

```

lemma float-lt [simp]:
  assumes is-finite a is-finite b

```

shows $a < b \iff \text{valof } a < \text{valof } b$
proof
assume $\text{valof } a < \text{valof } b$
moreover have $\neg \text{is-nan } a$ **and** $\neg \text{is-nan } b$ **and** $\neg \text{is-infinity } a$ **and** $\neg \text{is-infinity } b$
using *assms* **by** (*auto simp: finite-nan finite-infinity*)
ultimately have $fcompare\ a\ b = Lt$
by (*auto simp add: is-infinity-def is-nan-def valof-def fcompare-def*)
then show $a < b$ **by** (*auto simp: float-defs*)
next
assume $a < b$
then have $lt: fcompare\ a\ b = Lt$
by (*simp add: float-defs*)
have $\neg \text{is-nan } a$ **and** $\neg \text{is-nan } b$ **and** $\neg \text{is-infinity } a$ **and** $\neg \text{is-infinity } b$
using *assms* **by** (*auto simp: finite-nan finite-infinity*)
then show $\text{valof } a < \text{valof } b$
using *lt assms*
by (*simp add: fcompare-def is-nan-def is-infinity-def valof-def split: if-split-asm*)
qed

lemma *float-eq* [*simp*]:
assumes *is-finite* a *is-finite* b
shows $a \doteq b \iff \text{valof } a = \text{valof } b$
proof
assume $*$: $\text{valof } a = \text{valof } b$
have $\neg \text{is-nan } a$ **and** $\neg \text{is-nan } b$ **and** $\neg \text{is-infinity } a$ **and** $\neg \text{is-infinity } b$
using *assms float-distinct-finite* **by** *auto*
with $*$ **have** $fcompare\ a\ b = Eq$
by (*auto simp add: is-infinity-def is-nan-def valof-def fcompare-def*)
then show $a \doteq b$ **by** (*auto simp: float-defs*)
next
assume $a \doteq b$
then have $eq: fcompare\ a\ b = Eq$
by (*simp add: float-defs*)
have $\neg \text{is-nan } a$ **and** $\neg \text{is-nan } b$ **and** $\neg \text{is-infinity } a$ **and** $\neg \text{is-infinity } b$
using *assms float-distinct-finite* **by** *auto*
then show $\text{valof } a = \text{valof } b$
using *eq assms*
by (*simp add: fcompare-def is-nan-def is-infinity-def valof-def split: if-split-asm*)
qed

lemma *float-le* [*simp*]:
assumes *is-finite* a *is-finite* b
shows $a \leq b \iff \text{valof } a \leq \text{valof } b$
proof –
have $a \leq b \iff a < b \vee a \doteq b$
by (*auto simp add: float-defs*)
then show *?thesis*
by (*auto simp add: assms*)

qed

Reflexivity of equality for non-NaN's.

lemma *float-eq-refl* [simp]: $a \doteq a \longleftrightarrow \neg \text{is-nan } a$
by (*auto simp: float-defs*)

Properties about Ordering.

lemma *float-lt-trans*: $\text{is-finite } a \implies \text{is-finite } b \implies \text{is-finite } c \implies a < b \implies b < c \implies a < c$
by (*auto simp: le-trans*)

lemma *float-le-less-trans*: $\text{is-finite } a \implies \text{is-finite } b \implies \text{is-finite } c \implies a \leq b \implies b < c \implies a < c$
by (*auto simp: le-trans*)

lemma *float-le-trans*: $\text{is-finite } a \implies \text{is-finite } b \implies \text{is-finite } c \implies a \leq b \implies b \leq c \implies a \leq c$
by (*auto simp: le-trans*)

lemma *float-le-neg*: $\text{is-finite } a \implies \text{is-finite } b \implies \neg a < b \longleftrightarrow b \leq a$
by *auto*

Properties about bounding.

lemma *float-le-infinity* [simp]: $\neg \text{is-nan } a \implies a \leq \text{plus-infinity}$
unfolding *float-defs*
by *auto*

lemma *zero-le-topfloat*[simp]: $0 \leq \text{topfloat} - 0 \leq \text{topfloat}$
by (*auto simp: float-defs field-simps power-gt1-lemma of-nat-diff mask-eq-exp-minus-1*)

lemma *LENGTH-contr*:
 $\text{Suc } 0 < \text{LENGTH}('e) \implies 2 \wedge \text{LENGTH}('e::\text{len}) \leq \text{Suc } (\text{Suc } 0) \implies \text{False}$
by (*metis le-antisym len-gt-0 n-less-equal-power-2 not-less-eq numeral-2-eq-2 one-le-numeral self-le-power*)

lemma *valof-topfloat*: $\text{valof } (\text{topfloat}::('e, 'f)\text{float}) = \text{largest } \text{TYPE}((('e, 'f)\text{float}))$
if $\text{LENGTH}('e) > 1$
using *that LENGTH-contr*
by (*auto simp add: emax-eq largest-def divide-simps float-defs of-nat-diff mask-eq-exp-minus-1*)

lemma *float-frac-le*: $\text{fraction } a \leq 2^{\text{LENGTH}('f)} - 1$
for $a::('e, 'f)\text{float}$
unfolding *float-defs*
using *less-Suc-eq-le*
by *transfer fastforce*

lemma *float-exp-le*: $\text{is-finite } a \implies \text{exponent } a \leq \text{emax } \text{TYPE}((('e, 'f)\text{float})) - 1$
for $a::('e, 'f)\text{float}$
unfolding *float-defs*

by *auto*

lemma *float-sign-le*: $(-1::real) \wedge (\text{sign } a) = 1 \vee (-1::real) \wedge (\text{sign } a) = -1$
by (*metis neg-one-even-power neg-one-odd-power*)

lemma *exp-less*: $a \leq b \implies (2::real) \wedge a \leq 2 \wedge b$ for $a b :: \text{nat}$
by *auto*

lemma *div-less*: $a \leq b \wedge c > 0 \implies a/c \leq b/c$ for $a b c :: 'a::\text{linordered-field}$
by (*metis divide-le-cancel less-asym*)

lemma *finite-topfloat*: *is-finite topfloat*
unfolding *float-defs*
by *auto*

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

lemma *factor-minus*: $x * a - x = x * (a - 1)$
for $x a :: 'a::\text{comm-semiring-1-cancel}$
by (*simp add: algebra-simps*)

lemma *real-le-power-numeral-diff*: $\text{real } a \leq \text{numeral } b \wedge n - 1 \iff a \leq \text{numeral } b \wedge n - 1$
by (*metis (mono-tags, lifting) of-nat-1 of-nat-diff of-nat-le-iff of-nat-numeral one-le-numeral one-le-power semiring-1-class.of-nat-power*)

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

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

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

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

lemma *unat-one-word-le*: $\text{unat } a \leq \text{Suc } 0$ for $a::1 \text{ word}$
using *unat-lt2p*[of a]
by *auto*

lemma *one-word-le*: $a \leq 1$ for $a::1 \text{ word}$
by (*auto simp: word-le-nat-alt unat-one-word-le*)

lemma *sign-cases*[*case-names pos neg*]:
obtains $\text{sign } x = 0 \mid \text{sign } x = 1$
proof *cases*

```

    assume sign x = 0
    then show ?thesis ..
next
    assume sign x ≠ 0
    have sign x ≤ 1
      by transfer (auto simp: unat-one-word-le)
    then have sign x = 1 using ⟨sign x ≠ 0⟩
      by auto
    then show ?thesis ..
qed

lemma is-infinity-cases:
  assumes is-infinity x
  obtains x = plus-infinity | x = minus-infinity
proof (cases rule: sign-cases[of x])
  assume sign x = 0
  then have x = plus-infinity using assms
    apply (unfold float-defs)
    apply transfer
    apply (auto simp add: unat-eq-of-nat emax-def of-nat-mask-eq)
  done
  then show ?thesis ..
next
  assume sign x = 1
  then have x = minus-infinity using assms
    apply (unfold float-defs)
    apply transfer
    apply (auto simp add: unat-eq-of-nat emax-def of-nat-mask-eq)
  done
  then show ?thesis ..
qed

lemma is-zero-cases:
  assumes is-zero x
  obtains x = 0 | x = - 0
proof (cases rule: sign-cases[of x])
  assume sign x = 0
  then have x = 0 using assms
    unfolding float-defs
    by transfer (auto simp: emax-def unat-eq-0)
  then show ?thesis ..
next
  assume sign x = 1
  then have x = minus-zero using assms
    unfolding float-defs
    by transfer (auto simp: emax-def unat-eq-of-nat)
  then show ?thesis ..
qed

```

lemma *minus-minus-float[simp]*: $-(-f) = f$ **for** $f::('e, 'f)\text{float}$
by *transfer auto*

lemma *sign-minus-float*: $\text{sign}(-f) = (1 - \text{sign } f)$ **for** $f::('e, 'f)\text{float}$
by *transfer (auto simp: unat-eq-1 one-word-le unat-sub)*

lemma *exponent-uminus[simp]*: $\text{exponent}(-f) = \text{exponent } f$ **by** *transfer auto*

lemma *fraction-uminus[simp]*: $\text{fraction}(-f) = \text{fraction } f$ **by** *transfer auto*

lemma *is-normal-minus-float[simp]*: $\text{is-normal}(-f) = \text{is-normal } f$ **for** $f::('e, 'f)\text{float}$
by *(auto simp: is-normal-def)*

lemma *is-denormal-minus-float[simp]*: $\text{is-denormal}(-f) = \text{is-denormal } f$ **for** $f::('e, 'f)\text{float}$
by *(auto simp: is-denormal-def)*

lemma *bitlen-normal-mantissa*:

bitlen (abs (normal-mantissa x)) = Suc LENGTH('f) **for** $x::('e, 'f)\text{float}$

proof –

have *fraction $x < 2^{\wedge} \text{LENGTH}('f)$*

using *float-frac-le[of x]*

by *(metis One-nat-def Suc-pred le-imp-less-Suc pos2 zero-less-power)*

moreover have *$-\text{int}(\text{fraction } x) \leq 2^{\wedge} \text{LENGTH}('f)$*

using *negative-zle-0 order-trans zero-le-numeral zero-le-power* **by** *blast*

ultimately show *?thesis*

by *(cases x rule: sign-cases)*

(auto simp: bitlen-le-iff-power bitlen-ge-iff-power nat-add-distrib normal-mantissa-def intro!: antisym)

qed

lemma *less-int-natI*: $x < y$ **if** $0 \leq x$ **and** $\text{nat } x < \text{nat } y$
using *that* **by** *arith*

lemma *normal-exponent-bounds-int*:

$2 - 2^{\wedge}(\text{LENGTH}('e) - 1) - \text{int } \text{LENGTH}('f) \leq \text{normal-exponent } x$

$\text{normal-exponent } x \leq 2^{\wedge}(\text{LENGTH}('e) - 1) - \text{int } \text{LENGTH}('f) - 1$

if *is-normal x*

for $x::('e, 'f)\text{float}$

using *that*

apply *(auto simp add: normal-exponent-def is-normal-def emax-eq bias-def diff-le-eq diff-less-eq*

mask-eq-exp-minus-1 of-nat-diff simp flip: zless-nat-eq-int-zless power-Suc)

apply *(simp flip: power-Suc mask-eq-exp-minus-1 add: nat-mask-eq)*

apply *(simp add: mask-eq-exp-minus-1)*

done

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
subgoal by (rule order-trans[OF - of-int-leI[OF normal-exponent-bounds-int(1)[OF
that]]]) auto
subgoal by (rule order-trans[OF of-int-leI[OF normal-exponent-bounds-int(2)[OF
that]]]) auto
done

```

lemma *float-eqI*:

```

x = y if sign x = sign y fraction x = fraction y exponent x = exponent y
using that by transfer (auto simp add: word-unat-eq-iff)

```

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 -
from float-cases[of a]
consider is-nan a | is-infinity a | is-normal a | is-denormal a | is-zero a by blast
then show ?thesis
proof cases
case 1
then show ?thesis by (rule nan)
next
case 2
then consider a = plus-infinity | a = minus-infinity by (rule is-infinity-cases)
then show ?thesis
by cases (auto intro: infty)
next
case hyps: 3
from hyps have valof a = normal-mantissa a * 2 powr normal-exponent a
by (cases a rule: sign-cases)
(auto simp: valof-eq normal-mantissa-def normal-exponent-def is-normal-def
powr-realpow[symmetric] powr-diff powr-add field-simps)
from hyps this show ?thesis
by (rule normal)

```

```

next
  case hyps: 4
  from hyps have valof a = denormal-mantissa a * 2 powr denormal-exponent
TYPE(('e, 'f)float)
  by (cases a rule: sign-cases)
  (auto simp: valof-eq denormal-mantissa-def denormal-exponent-def is-denormal-def
  powr-realpow[symmetric] powr-diff powr-add field-simps)
  from hyps this show ?thesis
  by (rule denormal)
next
  case 5
  then consider a = 0 | a = minus-zero by (rule is-zero-cases)
  then show ?thesis
  by cases (auto intro: zero)
qed
qed

```

```

lemma infinite-infinity[simp]:  $\neg$  is-finite plus-infinity  $\neg$  is-finite minus-infinity
  by (auto simp: is-finite-def is-normal-def infinity-simps is-denormal-def is-zero-def)

```

```

lemma nan-not-finite[simp]: is-nan x  $\implies$   $\neg$  is-finite x
  using float-distinct-finite(1) by blast

```

```

lemma valof-nonneg:
  valof x  $\geq$  0 if sign x = 0 for x::('e, 'f)float
  by (auto simp: valof-eq that)

```

```

lemma valof-nonpos:
  valof x  $\leq$  0 if sign x = 1 for x::('e, 'f)float
  using that
  by (auto simp: valof-eq is-finite-def)

```

```

lemma real-le-intI: x  $\leq$  y if floor x  $\leq$  floor y x  $\in$   $\mathbb{Z}$  for x y::real
  using that(2,1)
  by (induction rule: Ints-induct) (auto elim!: Ints-induct simp: le-floor-iff)

```

```

lemma real-of-int-le-2-powr-bitlenI:
  real-of-int x  $\leq$  2 powr n - 1 if bitlen (abs x)  $\leq$  m m  $\leq$  n
proof -
  have real-of-int x  $\leq$  abs (real-of-int x) by simp
  also have ...  $<$  2 powr (bitlen (abs x))
  by (rule abs-real-le-2-powr-bitlen)
  finally have real-of-int x  $\leq$  2 powr (bitlen (abs x)) - 1
  by (auto simp: powr-real-of-int bitlen-nonneg intro!: real-le-intI)
  also have ...  $\leq$  2 powr m - 1
  by (simp add: that)
  also have ...  $\leq$  2 powr n - 1
  by (simp add: that)
  finally show ?thesis .

```

qed

lemma *largest-eq*:

largest TYPE('e, 'f)float =
 $(2^{\wedge}(\text{LENGTH}'f) + 1) - 1 * 2 \text{ powr real-of-int } (2^{\wedge}(\text{LENGTH}'e) - 1) -$
 $\text{int LENGTH}'f - 1)$

proof –

have $2^{\wedge} \text{LENGTH}'e - 1 - 1 = (2::\text{nat})^{\wedge} \text{LENGTH}'e - 2$

by *arith*

then have *largest TYPE('e, 'f)float* =

$(2^{\wedge}(\text{LENGTH}'f) + 1) - 1 * 2 \text{ powr } (\text{real } (2^{\wedge} \text{LENGTH}'e) - 2) + 1 - \text{real } (2^{\wedge}(\text{LENGTH}'e) - 1) -$
 $\text{LENGTH}'f)$

by (*auto simp add: largest-def emax-eq bias-def powr-minus field-simps powr-diff*
powr-add of-nat-diff

mask-eq-exp-minus-1 simp flip: powr-realpow)

also

have $2^{\wedge} \text{LENGTH}'e \geq (2::\text{nat})$

by (*simp add: self-le-power*)

then have $(\text{real } (2^{\wedge} \text{LENGTH}'e) - 2) + 1 - \text{real } (2^{\wedge}(\text{LENGTH}'e) - 1) -$
 $\text{LENGTH}'f) =$

$(\text{real } (2^{\wedge} \text{LENGTH}'e) - 2^{\wedge}(\text{LENGTH}'e) - 1) - \text{LENGTH}'f - 1$

by (*auto simp add: of-nat-diff*)

also have $\text{real } (2^{\wedge} \text{LENGTH}'e) = 2^{\wedge} \text{LENGTH}'e$ **by** *auto*

also have $(2^{\wedge} \text{LENGTH}'e) - 2^{\wedge}(\text{LENGTH}'e) - 1 - \text{real LENGTH}'f -$
 $1) =$

$\text{real-of-int } ((2^{\wedge}(\text{LENGTH}'e) - 1) - \text{int } (\text{LENGTH}'f) - 1))$

by (*simp, subst power-Suc[symmetric], simp*)

finally show *?thesis* **by** *simp*

qed

lemma *bitlen-denormal-mantissa*:

bitlen (abs (denormal-mantissa x)) ≤ LENGTH'f **for** $x::('e, 'f)\text{float}$

proof –

have *fraction* $x < 2^{\wedge} \text{LENGTH}'f$

using *float-frac-le*[*of x*]

by (*metis One-nat-def Suc-pred le-imp-less-Suc pos2 zero-less-power*)

moreover have $-\text{int } (\text{fraction } x) \leq 2^{\wedge} \text{LENGTH}'f$

using *negative-zle-0 order-trans zero-le-numeral zero-le-power* **by** *blast*

ultimately show *?thesis*

by (*cases x rule: sign-cases*)

(*auto simp: bitlen-le-iff-power denormal-mantissa-def intro!: antisym*)

qed

lemma *float-le-topfloat*:

fixes $a::('e, 'f)\text{float}$

assumes *is-finite a* $\text{LENGTH}'e > 1$

shows $a \leq \text{topfloat}$

using *assms*(1)

```

proof (induction a rule: float-induct)
  case (normal x)
  note normal(2)
  also have real-of-int (normal-mantissa x) * 2 powr real-of-int (normal-exponent
x) ≤
    (2 powr (LENGTH('f) + 1) - 1) * 2 powr (2 ^ (LENGTH('e) - 1) - int
LENGTH('f) - 1)
    using normal-exponent-bounds-real(2)[OF ‹is-normal x›]
    by (auto intro!: mult-mono real-of-int-le-2-powr-bitlenI
simp: bitlen-normal-mantissa powr-realpow[symmetric] ge-one-powr-ge-zero)
  also have ... = largest TYPE(('e, 'f) IEEE.float)
    unfolding largest-eq
    by (auto simp: powr-realpow powr-add)
  also have ... = valof (topfloat::('e, 'f) float)
    using assms
    by (simp add: valof-topfloat)
  finally show ?case
    by (intro float-leI normal finite-topfloat)
next
  case (denormal x)
  note denormal(2)
  also
  have 3 ≤ 2 powr (1 + real (LENGTH('e) - Suc 0))
  proof -
    have 3 ≤ 2 powr (2::real) by simp
    also have ... ≤ 2 powr (1 + real (LENGTH('e) - Suc 0))
      using assms
      by (subst powr-le-cancel-iff) auto
    finally show ?thesis .
  qed
  then have real-of-int (denormal-mantissa x) * 2 powr real-of-int (denormal-exponent
TYPE(('e, 'f)float)) ≤
    (2 powr (LENGTH('f) + 1) - 1) * 2 powr (2 ^ (LENGTH('e) - 1) - int
LENGTH('f) - 1)
    using bitlen-denormal-mantissa[of x]
    by (auto intro!: mult-mono real-of-int-le-2-powr-bitlenI
simp: bitlen-normal-mantissa powr-realpow[symmetric] ge-one-powr-ge-zero
mask-eq-exp-minus-1
denormal-exponent-def bias-def powr-mult-base of-nat-diff)
  also have ... ≤ largest TYPE(('e, 'f) IEEE.float)
    unfolding largest-eq
    by (rule mult-mono)
    (auto simp: powr-realpow powr-add power-Suc[symmetric] simp del: power-Suc)
  also have ... = valof (topfloat::('e, 'f) float)
    using assms
    by (simp add: valof-topfloat)
  finally show ?case
    by (intro float-leI denormal finite-topfloat)
qed auto

```


lemma *float-val-le-largest*:
valof $a \leq \text{largest } \text{TYPE}('e, 'f)\text{float}$
if *is-finite* $a \text{ LENGTH}('e) > 1$
for $a::('e, 'f)\text{float}$
by (*metis that finite-topfloat float-le float-le-topfloat valof-topfloat*)

lemma *float-val-lt-threshold*:
valof $a < \text{threshold } \text{TYPE}('e, 'f)\text{float}$
if *is-finite* $a \text{ LENGTH}('e) > 1$
for $a::('e, 'f)\text{float}$
proof –
have *valof* $a < \text{largest } \text{TYPE}('e, 'f)\text{float}$
by (*rule float-val-le-largest [OF that]*)
also have $\dots < \text{threshold } \text{TYPE}('e, 'f)\text{float}$
by (*auto simp: largest-def threshold-def divide-simps*)
finally show *?thesis* .
qed

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* $(a + b) \implies (a + b) \doteq (b + a)$
proof –
have $\neg \text{is-nan } a$ **and** $\neg \text{is-nan } b$ **and** $\neg \text{is-infinity } a$ **and** $\neg \text{is-infinity } b$
using *assms* **by** (*auto simp: finite-nan finite-infinity*)
then show $a + b = b + a$
by (*simp add: float-defs fadd-def plus-float-def add commute*)
then show *is-finite* $(a + b) \implies (a + b) \doteq (b + a)$
by (*metis float-eq*)
qed

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

lemma *is-zero-uminus[simp]*: *is-zero* $(- a) \longleftrightarrow \text{is-zero } a$
by (*simp add: is-zero-def*)

lemma *is-infinity-uminus [simp]*: *is-infinity* $(- a) = \text{is-infinity } a$
by (*simp add: is-infinity-def*)

lemma *is-finite-uminus[simp]*: *is-finite* $(- a) \longleftrightarrow \text{is-finite } a$
by (*simp add: is-finite-def*)

lemma *is-nan-uminus[simp]*: *is-nan* $(- a) \longleftrightarrow \text{is-nan } a$
by (*simp add: is-nan-def*)

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

lemma *float-neg-sign*: $(\text{sign } a) \neq (\text{sign } (- a))$
by (*cases a rule: sign-cases*) (*auto simp: sign-minus-float*)

lemma *float-neg-sign1*: $\text{sign } a = \text{sign } (- b) \longleftrightarrow \text{sign } a \neq \text{sign } b$
by (*metis float-neg-sign sign-cases*)

lemma *valof-uminus*:
assumes *is-finite a*
shows $\text{valof } (- a) = - \text{valof } a$ (**is** *?L = ?R*)
by (*cases a rule: sign-cases*) (*auto simp: valof-eq sign-minus-float*)

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

lemma *float-neg-add*:
is-finite a \implies *is-finite b* \implies *is-finite (a - b)* \implies $\text{valof } a + \text{valof } (- b) = \text{valof } a - \text{valof } b$
by (*simp add: valof-uminus*)

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

proof –

have *nab*: $\neg \text{is-nan } a \neg \text{is-nan } (- b) \neg \text{is-infinity } a \neg \text{is-infinity } (- b)$
using *assms* **by** (*auto simp: finite-nan finite-infinity*)

have $a - b = (\text{zerosign } (if \text{is-zero } a \wedge \text{is-zero } b \wedge \text{sign } a \neq \text{sign } b \text{ then } (\text{sign } a) \text{ else } 0) (\text{round To-nearest } (\text{valof } a - \text{valof } b)))$

using *nab*

by (*auto simp: minus-float-def fsub-def*)

also have $\dots =$

$(\text{zerosign } (if \text{is-zero } a \wedge \text{is-zero } (- b) \wedge \text{sign } a = \text{sign } (- b) \text{ then } \text{sign } a \text{ else } 0) (\text{round To-nearest } (\text{valof } a + \text{valof } (- b))))$

using *assms*

by (*simp add: float-neg-sign1 float-neg-add*)

also have $\dots = a + - b$

using *nab* **by** (*auto simp: float-defs fadd-def plus-float-def*)

finally show *?thesis*

using *assms* **by** (*metis float-eq*)

qed

lemma *finite-bottomfloat*: *is-finite bottomfloat*
by (*simp add: finite-topfloat*)

lemma *bottomfloat-eq-m-largest*: $\text{valof } (\text{bottomfloat}::('e, 'f)\text{float}) = - \text{largest } \text{TYPE} (('e, 'f)\text{float})$

if $\text{LENGTH } ('e) > 1$

using *that*

by (*auto simp: valof-topfloat valof-uminus finite-topfloat*)

lemma *float-val-ge-bottomfloat*: $\text{valof } a \geq \text{valof } (\text{bottomfloat}::('e, 'f)\text{float})$
if $\text{LENGTH}('e) > 1$ *is-finite* a
for $a::('e, 'f)\text{float}$
proof –
 have $- a \leq \text{topfloat}$
 using *that*
 by (*auto intro: float-le-topfloat*)
 then show *?thesis*
 using *that*
 by (*auto simp: valof-uminus finite-topfloat*)
qed

lemma *float-ge-bottomfloat*: $\text{is-finite } a \implies a \geq \text{bottomfloat}$
if $\text{LENGTH}('e) > 1$ *is-finite* a
for $a::('e, 'f)\text{float}$
by (*metis finite-bottomfloat float-le float-val-ge-bottomfloat that*)

lemma *float-val-ge-largest*:
fixes $a::('e, 'f)\text{float}$
assumes $\text{LENGTH}('e) > 1$ *is-finite* a
shows $\text{valof } a \geq - \text{largest } \text{TYPE}((('e, 'f)\text{float}))$
proof –
 have $- \text{largest } \text{TYPE}((('e, 'f)\text{float})) = \text{valof } (\text{bottomfloat}::('e, 'f)\text{float})$
 using *assms*
 by (*simp add: bottomfloat-eq-m-largest*)
 also have $\dots \leq \text{valof } a$
 using *assms* **by** (*simp add: float-val-ge-bottomfloat*)
 finally show *?thesis* .
qed

lemma *float-val-gt-threshold*:
fixes $a::('e, 'f)\text{float}$
assumes $\text{LENGTH}('e) > 1$ *is-finite* a
shows $\text{valof } a > - \text{threshold } \text{TYPE}((('e, 'f)\text{float}))$
proof –
 have $\text{largest: } \text{valof } a \geq - \text{largest } \text{TYPE}((('e, 'f)\text{float}))$
 using *assms* **by** (*metis float-val-ge-largest*)
 then have $- \text{largest } \text{TYPE}((('e, 'f)\text{float})) > - \text{threshold } \text{TYPE}((('e, 'f)\text{float}))$
 by (*auto simp: bias-def threshold-def largest-def divide-simps*)
 then show *?thesis*
 by (*metis largest less-le-trans*)
qed

Showing $\text{abs } (- a) = \text{abs } a$.

lemma *float-abs [simp]*: $\neg \text{is-nan } a \implies \text{abs } (- a) = \text{abs } a$
by (*metis IEEE.abs-float-def float-neg-sign1 minus-minus-float zero-simps(1)*)

lemma *neg-zerosign*: $- (\text{zerosign } s a) = \text{zerosign } (1 - s) (- a)$

by (auto simp: zerosign-def)

3.4 Properties about Rounding Errors

definition *error* :: ('e, 'f)float itself \Rightarrow real \Rightarrow 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*: $|x| < \text{threshold } \text{TYPE}((\text{'e}, \text{'f})\text{float})$
assumes *finite*: *is-finite a*
shows $|\text{valof } (\text{round To-nearest } x::(\text{'e}, \text{'f})\text{float}) - x| \leq |\text{valof } a - x|$
proof –
have *: (round To-nearest *x*::('e, 'f)float) =
 closest valof ($\lambda a.$ even (fraction *a*)) {*a. is-finite a*} *x*
using *threshold finite*
by *auto*
have *is-closest* (*valof*) {*a. is-finite a*} *x* (round To-nearest *x*::('e, 'f)float)
using *is-finite-nonempty*
unfolding *
by (*intro closest-is-closest*) *auto*
then show ?*thesis*
using *finite is-closest-def* **by** (*metis mem-Collect-eq*)
qed

lemma *error-at-worst-lemma*:

fixes *a*::('e, 'f)float
assumes *threshold*: $|x| < \text{threshold } \text{TYPE}((\text{'e}, \text{'f})\text{float})$
and *is-finite a*
shows $|\text{error } \text{TYPE}((\text{'e}, \text{'f})\text{float}) x| \leq |\text{valof } a - x|$
unfolding *error-def*
by (*rule bound-at-worst-lemma; fact*)

lemma *error-is-zero* [*simp*]:

fixes *a*::('e, 'f)float
assumes *is-finite a* $1 < \text{LENGTH}(\text{'e})$
shows *error* *TYPE*(('e, 'f)float) (*valof a*) = 0
proof –
have $|\text{valof } a| < \text{threshold } \text{TYPE}((\text{'e}, \text{'f})\text{float})$
by (*metis abs-less-iff minus-less-iff float-val-gt-threshold float-val-lt-threshold*
assms)
then show ?*thesis*
by (*metis abs-le-zero-iff abs-zero diff-self error-at-worst-lemma assms(1)*)
qed

lemma *is-finite-zerosign*[*simp*]: *is-finite* (*zerosign s a*) \longleftrightarrow *is-finite a*

by (*auto simp: zerosign-def is-finite-def*)

lemma *is-finite-closest*: *is-finite* (*closest* (*v*:: \Rightarrow real) *p* {*a. is-finite a*} *x*)

using *closest-is-closest*[*OF is-finite-nonempty, of v x p*]
by (*auto simp: is-closest-def*)

lemma *defloat-float-zerosign-round-finite*:

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

proof –

have (*round To-nearest x::('e, 'f)float*) =
 (*closest valof* ($\lambda a. \text{even } (\text{fraction } a) \{a. \text{is-finite } a\} x$)
using *threshold* **by** (*metis (full-types) abs-less-iff leD le-minus-iff round.simps(1)*)
then have *is-finite* (*round To-nearest x::('e, 'f)float*)
by (*metis is-finite-closest*)
then show *?thesis*
using *is-finite-zerosign* **by** *auto*

qed

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

by (*auto simp add: zerosign-def valof-eq zero-simps*)

lemma *signzero-zero*:

is-zero a \implies valof (zerosign s a) = 0

by (*auto simp add: zerosign-def*)

lemma *val-zero*: *is-zero a \implies valof a = 0*

by (*cases a rule: is-zero-cases*) *auto*

lemma *float-add*:

fixes *a b::('e, 'f)float*

assumes *is-finite a*

and *is-finite b*

and *threshold*: $|valof a + valof b| < \text{threshold } \text{TYPE}((\text{'e}, \text{'f})\text{float})$

shows *finite-float-add*: *is-finite (a + b)*

and *error-float-add*: *valof (a + b) = valof a + valof b + error TYPE((('e, 'f)float) (valof a + valof b))*

proof –

have $\neg \text{is-nan } a$ **and** $\neg \text{is-nan } b$ **and** $\neg \text{is-infinity } a$ **and** $\neg \text{is-infinity } b$

using *assms float-distinct-finite* **by** *auto*

then have *ab*: $(a + b) =$

(*zerosign*

(*if is-zero a \wedge is-zero b \wedge sign a = sign b then (sign a) else 0*)

(*round To-nearest (valof a + valof b)*))

using *assms* **by** (*auto simp add: float-defs fadd-def plus-float-def*)

then show *is-finite* ((*a + b*))

by (*metis threshold defloat-float-zerosign-round-finite*)

have *val-ab*: *valof (a + b) =*

valof (*zerosign*

(*if is-zero a \wedge is-zero b \wedge sign a = sign b then (sign a) else 0*)

(*round To-nearest (valof a + valof b)::('e, 'f)float*))

by (*auto simp: ab is-infinity-def is-nan-def valof-def*)

```

show  $\text{valof } (a + b) = \text{valof } a + \text{valof } b + \text{error } \text{TYPE}((\text{'e}, \text{'f})\text{float}) (\text{valof } a + \text{valof } b)$ 
proof (cases is-zero (round To-nearest (valof a + valof b)::('e, 'f)float))
  case True
  have  $\text{valof } a + \text{valof } b + \text{error } \text{TYPE}((\text{'e}, \text{'f})\text{float}) (\text{valof } a + \text{valof } b) = \text{valof } (\text{round To-nearest } (\text{valof } a + \text{valof } b)::(\text{'e}, \text{'f})\text{float})$ 
  unfolding error-def
  by simp
  then show ?thesis
  by (metis True signzero-zero val-zero val-ab)
next
  case False
  then show ?thesis
  by (metis ab add.commute eq-diff-eq' error-def zerosign-def)
qed
qed

```

lemma float-sub:

```

fixes a b::('e, 'f)float
assumes is-finite a
  and is-finite b
  and threshold:  $|\text{valof } a - \text{valof } b| < \text{threshold } \text{TYPE}((\text{'e}, \text{'f})\text{float})$ 
shows finite-float-sub: is-finite (a - b)
  and error-float-sub:  $\text{valof } (a - b) = \text{valof } a - \text{valof } b + \text{error } \text{TYPE}((\text{'e}, \text{'f})\text{float}) (\text{valof } a - \text{valof } b)$ 
proof -
have  $\neg \text{is-nan } a$  and  $\neg \text{is-nan } b$  and  $\neg \text{is-infinity } a$  and  $\neg \text{is-infinity } b$ 
  using assms by (auto simp: finite-nan finite-infinity)
then have ab:  $a - b =$ 
  (zerosign
    (if is-zero a  $\wedge$  is-zero b  $\wedge$  sign a  $\neq$  sign b then sign a else 0)
    (round To-nearest (valof a - valof b)))
  using assms by (auto simp add: float-defs fsub-def minus-float-def)
then show is-finite (a - b)
by (metis threshold defloat-float-zerosign-round-finite)
have val-ab:  $\text{valof } (a - b) =$ 
   $\text{valof } (\text{zerosign}$ 
    (if is-zero a  $\wedge$  is-zero b  $\wedge$  sign a  $\neq$  sign b then sign a else 0)
    (round To-nearest (valof a - valof b)::('e, 'f)float))
  by (auto simp: ab is-infinity-def is-nan-def valof-def)
show  $\text{valof } (a - b) = \text{valof } a - \text{valof } b + \text{error } \text{TYPE}((\text{'e}, \text{'f})\text{float}) (\text{valof } a - \text{valof } b)$ 
proof (cases is-zero (round To-nearest (valof a - valof b)::('e, 'f)float))
  case True
  have  $\text{valof } a - \text{valof } b + \text{error } \text{TYPE}((\text{'e}, \text{'f})\text{float}) (\text{valof } a - \text{valof } b) = \text{valof } (\text{round To-nearest } (\text{valof } a - \text{valof } b)::(\text{'e}, \text{'f})\text{float})$ 
  unfolding error-def by simp
  then show ?thesis
  by (metis True signzero-zero val-zero val-ab)

```

```

next
  case False
  then show ?thesis
    by (metis ab add.commute eq-diff-eq' error-def zerosign-def)
qed
qed

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 -
  have non:  $\neg is-nan\ a \neg is-nan\ b \neg is-infinity\ a \neg is-infinity\ b$ 
    using assms float-distinct-finite by auto
  then have ab:  $a * b =$ 
    (zerosign (of-bool (sign a  $\neq$  sign b)))
    (round To-nearest (valof a * valof b)::('e, 'f)float)
    using assms by (auto simp: float-defs fmul-def times-float-def)
  then show is-finite (a * b)
    by (metis threshold defloat-float-zerosign-round-finite)
  have val-ab:  $valof\ (a * b) =$ 
     $valof\ (zerosign\ (of-bool\ (sign\ a\ \neq\ sign\ b)))$ 
    (round To-nearest (valof a * valof b)::('e, 'f)float)
    by (auto simp: ab float-defs of-bool-def)
  show  $valof\ (a * b) = valof\ a * valof\ b + error\ TYPE (('e, 'f)float)$  (valof a * valof b)
  proof (cases is-zero (round To-nearest (valof a * valof b)::('e, 'f)float))
  case True
  have  $valof\ a * valof\ b + error\ TYPE (('e, 'f)float)$  (valof a * valof b) =
     $valof\ (round\ To-nearest\ (valof\ a * valof\ b)::('e, 'f)float)$ 
    unfolding error-def
    by simp
  then show ?thesis
    by (metis True signzero-zero val-zero val-ab)
  next
  case False then show ?thesis
    by (metis ab add.commute eq-diff-eq' error-def zerosign-def)
  qed
qed

lemma float-div:
  fixes a b::('e, 'f)float
  assumes is-finite a
    and is-finite b
    and not-zero:  $\neg is-zero\ b$ 

```

```

    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 -
  have ab: a / b =
    (zerosign (of-bool (sign a ≠ sign b))
     (round To-nearest (valof a / valof b)))
    using assms
  by (simp add: divide-float-def fdiv-def finite-infinity finite-nan not-zero float-defs
[symmetric])
  then show is-finite (a / b)
    by (metis threshold defloat-float-zerosign-round-finite)
  have val-ab: valof (a / b) =
    valof (zerosign (of-bool (sign a ≠ sign b))
     (round To-nearest (valof a / valof b))::('e, 'f)float)
    by (auto simp: ab float-defs of-bool-def)
  show valof (a / b) = valof a / valof b + error TYPE(('e, 'f)float) (valof a /
valof b)
proof (cases is-zero (round To-nearest (valof a / valof b))::('e, 'f)float))
  case True
  have valof a / valof b + error TYPE(('e, 'f)float) (valof a / valof b) =
    valof (round To-nearest (valof a / valof b))::('e, 'f)float)
    unfolding error-def
    by simp
  then show ?thesis
    by (metis True signzero-zero val-zero val-ab)
next
  case False then show ?thesis
    by (metis ab add commute eq-diff-eq' error-def zerosign-def)
qed
qed

lemma valof-one[simp]: valof (1 :: ('e, 'f) float) = of-bool (LENGTH('e) > 1)
apply transfer
apply (auto simp add: bias-def unat-mask-eq simp flip: mask-eq-exp-minus-1)
apply (simp add: mask-eq-exp-minus-1)
done

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 $fp64\text{-of}\text{-float} :: float64 \Rightarrow fp64$ **is**
 $\lambda(s::1\ word, e::11\ word, f::52\ word). word\text{-cat}\ s\ (word\text{-cat}\ e\ f::63\ word)$.

lift-definition $float\text{-of}\text{-fp64} :: fp64 \Rightarrow float64$ **is**
 $\lambda x. apsnd\ word\text{-split}\ (word\text{-split}\ x::1\ word * 63\ word)$.

definition $rel\text{-fp64} \equiv (\lambda x\ (y::word64). x = float\text{-of}\text{-fp64}\ y)$

definition $eq\text{-fp64} :: float64 \Rightarrow float64 \Rightarrow bool$ **where** $[simp]: eq\text{-fp64} \equiv (=)$

lemma $float\text{-of}\text{-fp64}\text{-inverse}[simp]: fp64\text{-of}\text{-float}\ (float\text{-of}\text{-fp64}\ a) = a$
by $(auto$
 $simp: fp64\text{-of}\text{-float}\text{-def}\ float\text{-of}\text{-fp64}\text{-def}\ Abs\text{-float}\text{-inverse}\ apsnd\text{-def}\ map\text{-prod}\text{-def}$
 $word\text{-size}$
 $dest!: word\text{-cat}\text{-split}\text{-alt}[rotated]$
 $split: prod.splits)$

lemma $float\text{-of}\text{-fp64}\text{-inj}\text{-iff}[simp]: fp64\text{-of}\text{-float}\ r = fp64\text{-of}\text{-float}\ s \iff r = s$
using $Rep\text{-float}\text{-inject}$
by $(force\ simp: fp64\text{-of}\text{-float}\text{-def}\ word\text{-cat}\text{-inj}\ word\text{-size}\ split: prod.splits)$

lemma $fp64\text{-of}\text{-float}\text{-inverse}[simp]: float\text{-of}\text{-fp64}\ (fp64\text{-of}\text{-float}\ a) = a$
using $float\text{-of}\text{-fp64}\text{-inj}\text{-iff}\ float\text{-of}\text{-fp64}\text{-inverse}$ **by** $blast$

lemma $Quotientfp: Quotient\ eq\text{-fp64}\ fp64\text{-of}\text{-float}\ float\text{-of}\text{-fp64}\ rel\text{-fp64}$
 $— eq\text{-fp64}$ is a workaround to prevent a (failing – TODO: why?) code setup in $setup\text{-lifting}$.
by $(force\ intro!: QuotientI\ simp: rel\text{-fp64}\text{-def})$

setup-lifting $Quotientfp$

lift-definition $fp64\text{-lessThan} :: fp64 \Rightarrow fp64 \Rightarrow bool$ **is**
 $flt :: float64 \Rightarrow float64 \Rightarrow bool$ **by** $simp$

lift-definition $fp64\text{-lessEqual} :: fp64 \Rightarrow fp64 \Rightarrow bool$ **is**
 $fle :: float64 \Rightarrow float64 \Rightarrow bool$ **by** $simp$

lift-definition $fp64\text{-greaterThan} :: fp64 \Rightarrow fp64 \Rightarrow bool$ **is**
 $fgt :: float64 \Rightarrow float64 \Rightarrow bool$ **by** $simp$

lift-definition $fp64\text{-greaterEqual} :: fp64 \Rightarrow fp64 \Rightarrow bool$ **is**
 $fge :: float64 \Rightarrow float64 \Rightarrow bool$ **by** $simp$

lift-definition *fp64-equal*::*fp64* \Rightarrow *fp64* \Rightarrow *bool* **is**
feq::*float64* \Rightarrow *float64* \Rightarrow *bool* **by** *simp*

lift-definition *fp64-abs*::*fp64* \Rightarrow *fp64* **is**
abs::*float64* \Rightarrow *float64* **by** *simp*

lift-definition *fp64-negate*::*fp64* \Rightarrow *fp64* **is**
uminus::*float64* \Rightarrow *float64* **by** *simp*

lift-definition *fp64-sqrt*::*roundmode* \Rightarrow *fp64* \Rightarrow *fp64* **is**
fsqrt::*roundmode* \Rightarrow *float64* \Rightarrow *float64* **by** *simp*

lift-definition *fp64-add*::*roundmode* \Rightarrow *fp64* \Rightarrow *fp64* \Rightarrow *fp64* **is**
fadd::*roundmode* \Rightarrow *float64* \Rightarrow *float64* \Rightarrow *float64* **by** *simp*

lift-definition *fp64-sub*::*roundmode* \Rightarrow *fp64* \Rightarrow *fp64* \Rightarrow *fp64* **is**
fsub::*roundmode* \Rightarrow *float64* \Rightarrow *float64* \Rightarrow *float64* **by** *simp*

lift-definition *fp64-mul*::*roundmode* \Rightarrow *fp64* \Rightarrow *fp64* \Rightarrow *fp64* **is**
fmul::*roundmode* \Rightarrow *float64* \Rightarrow *float64* \Rightarrow *float64* **by** *simp*

lift-definition *fp64-div*::*roundmode* \Rightarrow *fp64* \Rightarrow *fp64* \Rightarrow *fp64* **is**
fdiv::*roundmode* \Rightarrow *float64* \Rightarrow *float64* \Rightarrow *float64* **by** *simp*

lift-definition *fp64-mul-add*::*roundmode* \Rightarrow *fp64* \Rightarrow *fp64* \Rightarrow *fp64* \Rightarrow *fp64* **is**
fmul-add::*roundmode* \Rightarrow *float64* \Rightarrow *float64* \Rightarrow *float64* \Rightarrow *float64* **by** *simp*

end

theory *Conversion-IEEE-Float*
imports
HOL-Library.Float
IEEE-Properties
HOL-Library.Code-Target-Numerical
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* \Longrightarrow *of-finite* *x* = *valof* *x*
by (*induction* *x*) (*auto simp: normal-imp-not-denormal of-finite-def*)

definition *is-normal-Float*::('e, 'f)*float* *itself* \Rightarrow *Float.float* \Rightarrow *bool* **where**
is-normal-Float *x* *f* \longleftrightarrow
mantissa *f* \neq 0 \wedge

$\text{bitlen } |\text{mantissa } f| \leq \text{fracwidth } x + 1 \wedge$
 $-\text{int } (\text{bias } x) - \text{bitlen } |\text{mantissa } f| + 1 < \text{Float.exponent } f \wedge$
 $\text{Float.exponent } f < 2^{\wedge}(\text{LENGTH}('e)) - \text{bitlen } |\text{mantissa } f| - \text{bias } x$

definition *is-denormal-Float*::('e, 'f)float itself \Rightarrow Float.float \Rightarrow bool **where**
is-denormal-Float x f \longleftrightarrow
 $\text{mantissa } f \neq 0 \wedge$
 $\text{bitlen } |\text{mantissa } f| \leq 1 - \text{Float.exponent } f - \text{int } (\text{bias } x) \wedge$
 $1 - 2^{\wedge}(\text{LENGTH}('e) - 1) - \text{int } \text{LENGTH}('f) < \text{Float.exponent } f$

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 \Rightarrow Float.float \Rightarrow bool **where**
is-finite-Float x f \longleftrightarrow *is-normal-Float* x f \vee *is-denormal-Float* x f \vee f = 0

lemma *is-finite-Float-eq*:
is-finite-Float TYPE(('e, 'f)float) f \longleftrightarrow
 $(\text{let } e = \text{Float.exponent } f; \text{bm} = \text{bitlen } (\text{abs } (\text{mantissa } f)))$
 $\text{in } \text{bm} \leq \text{Suc } \text{LENGTH}('f) \wedge$
 $\text{bm} \leq 2^{\wedge}(\text{LENGTH}('e) - 1) - e \wedge$
 $1 - 2^{\wedge}(\text{LENGTH}('e) - 1) - \text{int } \text{LENGTH}('f) < e$

proof –
have *: $(2::\text{int})^{\wedge}(\text{LENGTH}('e) - \text{Suc } 0) - 1 < 2^{\wedge} \text{LENGTH}('e)$
by (*metis Suc-1 diff-le-self lessI linorder-not-less one-less-numeral-iff*
power-strict-increasing-iff zle-diff1-eq)
have **: $1 - 2^{\wedge}(\text{LENGTH}('e) - \text{Suc } 0) < \text{int } \text{LENGTH}('f)$
by (*smt len-gt-0 of-nat-0-less-iff zero-less-power*)
have ***: $2^{\wedge}(\text{LENGTH}('e) - 1) + 1 =$
 $2^{\wedge} \text{LENGTH}('e) - \text{int } (\text{bias } \text{TYPE}(('e, 'f) \text{IEEE.float}))$
by (*simp add: bias-def power-Suc[symmetric] of-nat-diff mask-eq-exp-minus-1*)
have rewr: $x \leq 2^{\wedge} n - e \longleftrightarrow x + e < 2^{\wedge} n + 1$ **for** x::int **and** n e
by *auto*
show ?thesis
unfolding *** rewr
using * **
unfolding *is-finite-Float-def is-normal-Float-def is-denormal-Float-def*
by (*auto simp: Let-def bias-def mantissa-eq-zero-iff of-nat-diff mask-eq-exp-minus-1*
intro: le-less-trans[OF add-right-mono])

qed

lift-definition *normal-of-Float* :: Float.float \Rightarrow ('e, 'f)float
is $\lambda x.$ let m = mantissa x; e = Float.exponent x in
 $(\text{if } m > 0 \text{ then } 0 \text{ else } 1,$
 $\text{word-of-int } (e + \text{int } (\text{bias } \text{TYPE}(('e, 'f)float)) + \text{bitlen } |m| - 1),$
 $\text{word-of-int } (|m| * 2^{\wedge}(\text{Suc } \text{LENGTH}('f) - \text{nat } (\text{bitlen } |m|)) - 2^{\wedge}(\text{LENGTH}('f))))$

lemma *sign-normal-of-Float:sign* (*normal-of-Float* x) = (if $x > 0$ then 0 else 1)
by *transfer* (*auto simp: Let-def mantissa-pos-iff*)

lemma *uint-word-of-int-bitlen-eq*:
uint (*word-of-int* $x::'a::len$ *word*) = x **if** $\text{bitlen } x \leq \text{LENGTH('a)}$ $x \geq 0$
using *that* **by** (*simp add: bitlen-le-iff-power take-bit-int-eq-self unsigned-of-int*)

lemma *fraction-normal-of-Float:fraction* (*normal-of-Float* $x::('e, 'f)\text{float}$) =
 $(\text{nat } |\text{mantissa } x| * 2^{(\text{Suc } \text{LENGTH('f)} - \text{nat } (\text{bitlen } |\text{mantissa } x|))} - 2^{\text{LENGTH('f)})}$
if *is-normal-Float* *TYPE*(('e, 'f)*float*) x
proof –
from *that* **have** *bmp*: $\text{bitlen } |\text{mantissa } x| > 0$
by (*metis abs-of-nonneg bitlen-bounds bitlen-def is-normal-Float-def nat-code(2) of-nat-0-le-iff*
power.simps(1) zabs-less-one-iff zero-less-abs-iff)
have *mless*: $|\text{mantissa } x| < 2^{\text{nat } (\text{bitlen } |\text{mantissa } x|)}$
using *bitlen-bounds* **by** *force*
have *lem*: $2^{\text{nat } (\text{bitlen } |\text{mantissa } x| - 1)} \leq |\text{mantissa } x|$
using *bitlen-bounds is-normal-Float-def that zero-less-abs-iff* **by** *blast*
from *that* **have** *nble*: $\text{nat } (\text{bitlen } |\text{mantissa } x|) \leq \text{Suc } \text{LENGTH('f)}$
using *bitlen-bounds* **by** (*auto simp: is-normal-Float-def*)
have *nn*: $0 \leq |\text{mantissa } x| * 2^{(\text{Suc } \text{LENGTH('f)} - \text{nat } (\text{bitlen } |\text{mantissa } x|))} - 2^{\text{LENGTH('f)}}$
apply (*rule add-le-imp-le-diff*)
apply (*rule order-trans[rotated]*)
apply (*rule mult-right-mono*)
apply (*rule lem, force*)
unfolding *power-add[symmetric]*
using *nble bmp*
by (*auto*)
have $|\text{mantissa } x| * 2^{(\text{Suc } \text{LENGTH('f)} - \text{nat } (\text{bitlen } |\text{mantissa } x|))} < 2 * 2^{\text{LENGTH('f)}}$
apply (*rule less-le-trans*)
apply (*rule mult-strict-right-mono*)
apply (*rule mless*)
apply *force*
unfolding *power-add[symmetric] power-Suc[symmetric]*
apply (*rule power-increasing*)
using *nble*
by *auto*
then **have** $\text{bitlen } (|\text{mantissa } x| * 2^{(\text{Suc } \text{LENGTH('f)} - \text{nat } (\text{bitlen } |\text{mantissa } x|))} - 2^{\text{LENGTH('f)})}$
 $\leq \text{int } \text{LENGTH('f)}$
unfolding *bitlen-le-iff-power*
by *simp*
then *show ?thesis*
apply (*transfer fixing: x*)
unfolding *Let-def split-beta' fst-conv snd-conv uint-nat [symmetric] nat-uint-eq*

[*symmetric*]

using *nn*
apply (*subst uint-word-of-int-bitlen-eq*)
apply (*auto simp: nat-mult-distrib nat-diff-distrib nat-power-eq*)
done
qed

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*
using *that*
apply (*transfer fixing: x*)
apply (*simp flip: uint-nat nat-uint-eq add: Let-def*)
apply (*auto simp: is-normal-Float-def bitlen-le-iff-power uint-word-of-int-bitlen-eq*
Let-def)
apply *transfer*
apply (*simp add: nat-take-bit-eq take-bit-int-eq-self*)
done

lift-definition *denormal-of-Float :: Float.float \Rightarrow ('e, 'f)float*
is λx . *let m = mantissa x; e = Float.exponent x in*
(if m \geq 0 then 0 else 1, 0,
*word-of-int (|m| * 2^{nat (e + bias TYPE(('e, 'f)float)) + fracwidth TYPE(('e,}*
'f)float) - 1)))
.

lemma *sign-denormal-of-Float:sign* (*denormal-of-Float x*) = (*if x \geq 0 then 0 else*
1)
by *transfer (auto simp: Let-def mantissa-nonneg-iff)*

lemma *exponent-denormal-of-Float:exponent* (*denormal-of-Float x::('e, 'f)float*) =
0
by (*transfer fixing: x*) (*auto simp: Let-def*)

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 –

have *mless: |mantissa x| < 2^{nat (bitlen |mantissa x|)}*
using *bitlen-bounds by force*
have $*$: *nat (bitlen |mantissa x|) +*
nat (Float.exponent x + (2^{(LENGTH('e) - Suc 0) + int LENGTH('f)) -}
2)
 \leq *LENGTH('f)*
using *that*
by (*auto simp: is-denormal-Float-def nat-diff-distrib' le-diff-conv mask-eq-exp-minus-1*
bitlen-nonneg nat-le-iff bias-def nat-add-distrib[symmetric] of-nat-diff)
have $|mantissa x| * 2^{nat (Float.exponent x + int (bias TYPE(('e, 'f)float))}$

+
 $LENGTH('f) - 1) < 2 \wedge LENGTH('f)$
apply (rule less-le-trans)
apply (rule mult-strict-right-mono)
apply (rule mless, force)
unfolding power-add[symmetric] power-Suc[symmetric]
apply (rule power-increasing)
apply (auto simp: bias-def)
using that *
by (auto simp: is-denormal-Float-def algebra-simps of-nat-diff mask-eq-exp-minus-1)
then show ?thesis
apply (transfer fixing: x)
apply transfer
apply (simp add: Let-def nat-eq-iff take-bit-eq-mod)
done
qed

definition of-finite-Float :: Float.float \Rightarrow ('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 –

have valof (normal-of-Float x::('e, 'f)float) =
 (- 1) \wedge sign (normal-of-Float x::('e, 'f)float) *
 ((1 + real (nat |mantissa x| * 2 \wedge (Suc LENGTH('f) - nat (bitlen |mantissa
 x|)) - 2 \wedge LENGTH('f)) / 2 \wedge LENGTH('f)) *
 2 powr (bitlen |mantissa x| - 1)) *
 2 powr Float.exponent x
 (is - = ?s * ?m * ?e)
using that
by (auto simp: is-normal-Float-def valof-eq fraction-normal-of-Float
 powr-realpow[symmetric] exponent-normal-of-Float powr-diff powr-add)

also

have |mantissa x| > 0

using that

by (auto simp: is-normal-Float-def)

have bound: 2 \wedge LENGTH('f) \leq nat |mantissa x| * 2 \wedge (Suc LENGTH('f) -
 nat (bitlen |mantissa x|))

proof –

have (2::nat) \wedge LENGTH('f) \leq 2 \wedge nat (bitlen |mantissa x| - 1) * 2 \wedge (Suc
 LENGTH('f) - nat (bitlen |mantissa x|))

by (simp add: power-add[symmetric])

also have ... \leq nat |mantissa x| * 2 \wedge (Suc LENGTH('f) - nat (bitlen
 |mantissa x|))

using bitlen-bounds[of |mantissa x|] that

by (auto simp: is-normal-Float-def)
 finally show ?thesis .
 qed
 have ?m = abs (mantissa x)
 apply (subst of-nat-diff)
 subgoal using bound by auto
 subgoal
 using that
 by (auto simp: powr-realpow[symmetric] powr-add[symmetric]
 is-normal-Float-def bitlen-nonneg of-nat-diff divide-simps)
 done
 finally show ?thesis
 by (auto simp: mantissa-exponent sign-normal-of-Float abs-real-def zero-less-mult-iff)
 qed

lemma valof-denormal-of-Float: valof (denormal-of-Float x::('e, 'f)float) = x
 if is-denormal-Float TYPE(('e, 'f)float) x
proof –
 have less: 0 < Float.exponent x + (int (bias TYPE(('e, 'f) IEEE.float)) + int
 LENGTH('f))
 using that
 by (auto simp: is-denormal-Float-def bias-def of-nat-diff mask-eq-exp-minus-1)
 have valof (denormal-of-Float x::('e, 'f)float) =
 ((- 1) ^ sign (denormal-of-Float x::('e, 'f)float) * |real-of-int (mantissa x)|) *
 (2 powr real (nat (Float.exponent x + int (bias TYPE(('e, 'f) IEEE.float)) +
 int LENGTH('f) - 1)) /
 (2 powr real (bias TYPE(('e, 'f) IEEE.float)) * 2 powr LENGTH('f)) * 2)
 (is - = ?m * ?e)
 by (auto simp: valof-eq exponent-denormal-of-Float fraction-denormal-of-Float
 that
 mantissa-exponent powr-realpow[symmetric])
 also have ?m = mantissa x
 by (auto simp: sign-denormal-of-Float abs-real-def mantissa-neg-iff)
 also have ?e = 2 powr Float.exponent x
 by (auto simp: powr-add[symmetric] divide-simps powr-mult-base less ac-simps)
 finally show ?thesis by (simp add: mantissa-exponent)
 qed

lemma valof-of-finite-Float:
 is-finite-Float (TYPE(('e, 'f) IEEE.float)) x \implies valof (of-finite-Float x::('e,
 'f)float) = x
 by (auto simp: of-finite-Float-def is-finite-Float-def valof-denormal-of-Float valof-normal-of-Float)

lemma is-normal-normal-of-Float:
 is-normal (normal-of-Float x::('e, 'f)float) if is-normal-Float TYPE(('e, 'f)float)
 x
 using that
 by (auto simp: is-normal-def exponent-normal-of-Float that is-normal-Float-def
 emax-eq nat-less-iff of-nat-diff)

lemma *is-denormal-denormal-of-Float*: *is-denormal* (*denormal-of-Float* $x::('e, 'f)\text{float}$)
if *is-denormal-Float* *TYPE*(($'e, 'f$)*float*) x
using *that*
by (*auto simp*: *is-denormal-def* *exponent-denormal-of-Float* *that is-denormal-Float-def*
emax-eq *fraction-denormal-of-Float* *le-nat-iff* *bias-def*)

lemma *is-finite-of-finite-Float*: *is-finite* (*of-finite-Float* x)
by (*auto simp*: *is-finite-def* *of-finite-Float-def* *is-normal-normal-of-Float*
is-denormal-denormal-of-Float)

lemma *Float-eq-zero-iff*: *Float* $m\ e = 0 \longleftrightarrow m = 0$
by (*metis* *Float.compute-is-float-zero* *Float-0-eq-0*)

lemma *bitlen-mantissa-Float*:
shows *bitlen* $|mantissa\ (Float\ m\ e)| = (if\ m = 0\ then\ 0\ else\ bitlen\ |m| + e) -$
Float.exponent (*Float* $m\ e$)
using *bitlen-Float*[*of* $m\ e$] **by** *auto*

lemma *exponent-Float*:
shows *Float.exponent* (*Float* $m\ e$) = (*if* $m = 0$ *then* 0 *else* *bitlen* $|m| + e$) -
bitlen $|mantissa\ (Float\ m\ e)|$
using *bitlen-Float*[*of* $m\ e$] **by** *auto*

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

if *is-normal* x **for** $x::('e, 'f)\text{float}$

proof -

define f **where** $f = Float\ (normal-mantissa\ x)\ (normal-exponent\ x)$
from *that* **have** $f \neq 0$

by (*auto simp*: *f-def* *is-normal-def* *zero-float-def*[*symmetric*]
Float-eq-zero-iff *normal-mantissa-def* *add-nonneg-eq-0-iff*)

from *denormalize-shift*[*OF* f -*def* *this*] **obtain** i **where**

$i: normal-mantissa\ x = mantissa\ f * 2^i$ *normal-exponent* $x = Float.exponent$
 $f - int\ i$

by *auto*

have *mantissa* $f \neq 0$

by (*auto simp*: $\langle f \neq 0 \rangle\ i$ *mantissa-eq-zero-iff* *Float-eq-zero-iff*)

moreover

have *normal-exponent* $x \leq Float.exponent\ f$ **unfolding** i **by** *simp*

then **have** *bitlen* $|mantissa\ f| \leq 1 + int\ LENGTH('f)$

unfolding *bitlen-mantissa-Float* *bitlen-normal-mantissa* f -*def*

by *auto*

moreover

have - *int* (*bias* *TYPE*(($'e, 'f$)*float*)) - *bitlen* $|mantissa\ f| + 1 < Float.exponent$
 f

unfolding *bitlen-mantissa-Float* *bitlen-normal-mantissa* f -*def*

using *that*

by (*auto simp: mantissa-eq-zero-iff abs-mult bias-def normal-mantissa-def normal-exponent-def*
is-normal-def emax-eq less-diff-conv add-nonneg-eq-0-iff)
moreover
have $2^{\wedge}(\text{LENGTH}('e) - \text{Suc } 0) + - (1::\text{int}) * 2^{\wedge}\text{LENGTH}('e) \leq 0$
by *simp*
then have $(2::\text{int})^{\wedge}(\text{LENGTH}('e) - \text{Suc } 0) < 1 + 2^{\wedge}\text{LENGTH}('e)$ **by** *arith*
then have *Float.exponent f <*
 $2^{\wedge}\text{LENGTH}('e) - \text{bitlen } |\text{mantissa } f| - \text{int } (\text{bias } \text{TYPE}(('e, 'f)\text{float}))$
using *normal-exponent-bounds-int[OF that]*
unfolding *bitlen-mantissa-Float bitlen-normal-mantissa f-def*
by (*auto simp: bias-def algebra-simps power-Suc[symmetric] of-nat-diff mask-eq-exp-minus-1*
intro: le-less-trans[OF add-right-mono] normal-exponent-bounds-int[OF that])
ultimately
show *?thesis*
by (*auto simp: is-normal-Float-def f-def*)
qed

lemma *is-denormal-Float-denormal:*

is-denormal-Float TYPE(('e, 'f)\text{float})
(Float (denormal-mantissa x) (denormal-exponent TYPE(('e, 'f)\text{float})))
if *is-denormal x for x::('e, 'f)\text{float}*
proof –
define *f where f = Float (denormal-mantissa x) (denormal-exponent TYPE(('e, 'f)\text{float}))*
from *that have f ≠ 0*
by (*auto simp: f-def is-denormal-def zero-float-def[symmetric]*
Float-eq-zero-iff denormal-mantissa-def add-nonneg-eq-0-iff)
from *denormalize-shift[OF f-def this] obtain i where*
*i: denormal-mantissa x = mantissa f * 2^∧ i denormal-exponent TYPE(('e, 'f)\text{float}) = Float.exponent f - int i*
by *auto*
have *mantissa f ≠ 0*
by (*auto simp: ⟨f ≠ 0⟩ i mantissa-eq-zero-iff Float-eq-zero-iff*)
moreover
have $\text{bitlen } |\text{mantissa } f| \leq 1 - \text{Float.exponent } f - \text{int } (\text{bias } \text{TYPE}(('e, 'f)\text{IEEE.float}))$
using *⟨mantissa f ≠ 0⟩*
unfolding *f-def bitlen-mantissa-Float*
using *bitlen-denormal-mantissa[of x]*
by (*auto simp: denormal-exponent-def*)
moreover
have $2 - 2^{\wedge}(\text{LENGTH}('e) - \text{Suc } 0) - \text{int } \text{LENGTH}('f) \leq \text{Float.exponent } f$
(is ?l ≤ -)
proof –
have $?l \leq \text{denormal-exponent } \text{TYPE}(('e, 'f)\text{float}) + i$
using *that*
by (*auto simp: is-denormal-def bias-def denormal-exponent-def of-nat-diff*)

```

mask-eq-exp-minus-1)
  also have ... = Float.exponent f unfolding i by auto
  finally show ?thesis .
qed
ultimately
show ?thesis
  unfolding is-denormal-Float-def exponent-Float f-def[symmetric]
  by auto
qed

lemma is-finite-Float-of-finite: is-finite-Float TYPE(('e, 'f)float) (of-finite x) for
x::('e, 'f)float
  by (auto simp: is-finite-Float-def of-finite-def is-normal-Float-normal
is-denormal-Float-denormal)

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 ..

setup-lifting type-definition-double

instantiation double :: {uminus, plus, times, minus, zero, one, abs, ord, inverse} begin
lift-definition uminus-double::double  $\Rightarrow$  double is uminus .
lift-definition plus-double::double  $\Rightarrow$  double  $\Rightarrow$  double is plus .
lift-definition times-double::double  $\Rightarrow$  double  $\Rightarrow$  double is times .
lift-definition divide-double::double  $\Rightarrow$  double  $\Rightarrow$  double is divide .
lift-definition inverse-double::double  $\Rightarrow$  double is inverse .
lift-definition minus-double::double  $\Rightarrow$  double  $\Rightarrow$  double is minus .
lift-definition zero-double::double is 0 .
lift-definition one-double::double is 1 .
lift-definition less-eq-double::double  $\Rightarrow$  double  $\Rightarrow$  bool is ( $\leq$ ) .
lift-definition less-double::double  $\Rightarrow$  double  $\Rightarrow$  bool is (<) .
instance proof qed
end

lift-definition eq-double::double $\Rightarrow$ double $\Rightarrow$ bool is float-eq .

lift-definition sqrt-double::double $\Rightarrow$ double is float-sqrt .

```

no-notation *plus-infinity* (∞)

lift-definition *infinity-double::double* (∞) **is** *plus-infinity* .

lift-definition *is-normal::double* \Rightarrow *bool* **is** *IEEE.is-normal* .

lift-definition *is-zero::double* \Rightarrow *bool* **is** *IEEE.is-zero* .

lift-definition *is-finite::double* \Rightarrow *bool* **is** *IEEE.is-finite* .

lift-definition *is-nan::double* \Rightarrow *bool* **is** *IEEE.is-nan* .

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

code-printing constant *0 :: double* \rightarrow
(SML) 0.0 **and** *(OCaml) 0.0*

declare *zero-double.abs-eq*[code del]

code-printing constant *1 :: double* \rightarrow
(SML) 1.0 **and** *(OCaml) 1.0*

declare *one-double.abs-eq*[code del]

code-printing constant *eq-double :: double* \Rightarrow *double* \Rightarrow *bool* \rightarrow
(SML) Real.== (*(-:real)*, *(-)*) **and** *(OCaml) Pervasives.(=)*

declare *eq-double.abs-eq*[code del]

code-printing constant *Orderings.less-eq :: double* \Rightarrow *double* \Rightarrow *bool* \rightarrow
(SML) Real.<= (*(-)*, *(-)*) **and** *(OCaml) Pervasives.(<=)*

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

code-printing constant *Orderings.less :: double* \Rightarrow *double* \Rightarrow *bool* \rightarrow
(SML) Real.< (*(-)*, *(-)*) **and** *(OCaml) Pervasives.<*

declare *less-eq-double-def*[code del]

code-printing constant *(+) :: double* \Rightarrow *double* \Rightarrow *double* \rightarrow
(SML) Real.+ (*(-)*, *(-)*) **and** *(OCaml) Pervasives.(+)*

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

code-printing constant *(*) :: double* \Rightarrow *double* \Rightarrow *double* \rightarrow
*(SML) Real.** (*(-)*, *(-)*) **and** *(OCaml) Pervasives.(*)*

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

code-printing constant *(-) :: double* \Rightarrow *double* \Rightarrow *double* \rightarrow
(SML) Real.- (*(-)*, *(-)*) **and** *(OCaml) Pervasives.(-)*

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

code-printing constant *uminus :: double* \Rightarrow *double* \rightarrow
(SML) Real.~ **and** *(OCaml) Pervasives.(~ -)*

code-printing constant *(/) :: double* \Rightarrow *double* \Rightarrow *double* \rightarrow
(SML) Real./ (*(-)*, *(-)*) **and** *(OCaml) Pervasives.('/)*

```

declare divide-double-def [code del]

code-printing constant sqrt-double :: double  $\Rightarrow$  double  $\rightarrow$ 
  (SML) Math.sqrt and (OCaml) Pervasives.sqrt
declare sqrt-def[code del]

code-printing constant infinity-double :: double  $\rightarrow$ 
  (SML) Real.posInf
declare infinity-double.abs-eq[code del]

code-printing constant is-normal :: double  $\Rightarrow$  bool  $\rightarrow$ 
  (SML) Real.isNormal
declare [[code drop: is-normal]]

code-printing constant is-finite :: double  $\Rightarrow$  bool  $\rightarrow$ 
  (SML) Real.isFinite
declare [[code drop: is-finite]]

code-printing constant is-nan :: double  $\Rightarrow$  bool  $\rightarrow$ 
  (SML) Real.isNaN
declare [[code drop: is-nan]]

```

Mapping natural numbers to doubles.

```

fun float-of :: nat  $\Rightarrow$  double
where
  float-of 0 = 0
| float-of (Suc n) = float-of n + 1

lemma float-of 2 < float-of 3 + float-of 4
  by eval

export-code float-of in SML

```

5.1 Conversion from int

```

lift-definition double-of-int::int  $\Rightarrow$  double is  $\lambda i.$  round To-nearest (real-of-int i) .

```

```

context includes integer.lifting begin

```

```

lift-definition double-of-integer::integer  $\Rightarrow$  double is double-of-int .
end

```

```

lemma float-of-int[code]:
  double-of-int i = double-of-integer (integer-of-int i)
  by (auto simp: double-of-integer-def)

```

```

code-printing

```

```

  constant double-of-integer :: integer  $\Rightarrow$  double  $\rightarrow$  (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 \longleftrightarrow
(let e = Float.exponent f; bm = bitlen (abs (mantissa f))
in (bm \leq 53 \wedge e + bm < 1025 \wedge - 1075 < e))

unfolding *is-finite-Float-eq*

by (*auto simp: Let-def*)

code-printing

code-module *IEEE-Mantissa-Exponent* \rightarrow (*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* \Rightarrow *Float.float* **is** *Conversion-IEEE-Float.of-finite* .

definition *man-exp-of-double::double* \Rightarrow (*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* \Rightarrow *double* **is** *Conversion-IEEE-Float.of-finite-Float*

definition *double-of-man-exp::integer* \Rightarrow *integer* \Rightarrow *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 \Rightarrow (integer * integer) option \rightarrow
(SML) IEEE'-Mantissa'-Exponent.to'-man'-exp'-double |

constant double-of-man-exp :: integer \Rightarrow integer \Rightarrow double option \rightarrow
(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 \Rightarrow Float.float option **is**

λx . if is-finite x then Some (of-finite x) else None .

lift-definition double-of-Float::Float.float \Rightarrow double option **is**

λx . if is-finite-Float TYPE((11, 52)float) x then Some (of-finite-Float x) else None .

lemma compute-Float-of-double[code]:

Float-of-double x =

map-option ($\lambda(m, e)$. Float (int-of-integer m) (int-of-integer e)) (man-exp-of-double x)

apply (rule sym)

by transfer (auto simp: man-exp-of-double-def Let-def mantissa-exponent[symmetric]
Float-mantissa-exponent)

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))

unfolding double-of-man-exp-def Let-def Float-mantissa-exponent int-of-integer-integer-of-int

apply auto

subgoal by transfer auto

subgoal by transfer auto

done

definition check-conversion m e =

(let f = Float (int-of-integer m) (int-of-integer e) in

do {

d \leftarrow 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 \Rightarrow (nat \Rightarrow bool) \Rightarrow bool **where**

check-all 0 P \longleftrightarrow True

| check-all (Suc i) P \longleftrightarrow P i \wedge 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  
  by eval  
  
end
```

References

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