A Mechanically Checked Proof of IEEE Compliance of the AMD K5 Floating-Point Square Root Microcode

DMR

David M. Russinoff

mailto:russ@onr.com
http://www.onr.com/user/russ/david

IEEE Compliance

Any elementary floating-point operation,

$$+,\ -,\ *,\ /,\ \text{or}\ \sqrt{\ },$$

computed to n bits of precision with respect to a given *mode* of rounding,

> ... shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result ....

Thus, the computed and ideal values are related by

$$val_{comp} = rnd(val_{ideal}, mode, n).$$

K5 Square Root Instruction

- Original algorithm, designed at AMD by Lynch, Schulte, and Ahmed
- Combines table lookup, Newton-Raphson, Tuckerman rounding
- Microcoded algorithm designed to make efficient use of FPU, which supports 64-bit precision and 17-bit exponents:
  - 32-bit multiplier
  - 41-bit registers
  - 82-bit bus
- Supports double-extended inputs (64-bit precision, 15-bit exponents)

ACL2: A Computational Logic for Applicative Common LISP

ACL2 Components:

- A mathematical logic, based on S-expressions, recursive functions, and induction
- A programming language, with data types supporting list processing and rational arithmetic
- A theorem prover (proof-checker), based on arithmetic decision procedures, term rewriting, and induction heuristics

Value to this project:

- Increased confidence in the result
- Executable specification

Disadvantages:

- Mechanical proof-checking is time-consuming
- No support for irrational numbers

Objectives of this Project

- *Precise* mathematical statement of IEEE compliance
  - Floating-point numbers
  - Rounding
- *Accurate* model of K5 square root instruction
  - Represented in a formal language
  - Based on K5 microcode
- *Rigorous* proof of correctness
  - IEEE compliance
  - Conformance to K5 FPU architecture
- *Formal* encoding in ACL2
  - Executable specification
  - Mechanical verification

Floating-Point Representation

Any $x > 0$ admits a unique factorization

$$x = sig(x)2^{expo(x)},$$

where $1 \leq sig(x) < 2$ and $expo(x) \in \mathbf{Z}$.

We shall say that x is *n-exact* if

$$sig(x) \text{ is representable in } n \text{ bits},$$

i.e.,

$$sig(x)2^{n-1} = x2^{n-1-expo(x)} \in \mathbf{Z}.$$

Example: $x = 26 = 11010_2 = 1.101_2 2^4$
is 4-exact, since

$$sig(x)2^3 = 1.101_2 2^3 = 1101_2 = 13 \in \mathbf{Z}.$$

K5 FPU Internal Format

An 82-bit sequence

σ	s_1	$\cdots$	s_{64}	e_1	$\cdots$	e_{17}

represents the rational number

$$x = (-1)^\sigma (s_1.s_2 \ldots s_{64})_2 \cdot 2^{(e_1 \ldots e_{17})_2 - 2^{16} + 1}.$$

It is required that if $x \neq 0$, then $s_1 = 1$
(no denormal representation).

Thus, any $x > 0$ is representable in this format iff

(a) x is 64-exact, and

(b) $-2^{16} + 1 \leq expo(x) \leq 2^{16}$.

Double-Extended Format

An 80-bit sequence

σ	s_1	$\cdots$	s_{64}	e_1	$\cdots$	e_{15}

represents the rational number

$$x = (-1)^\sigma (s_1.s_2 \ldots s_{64})_2 \cdot 2^{(e_1 \ldots e_{15})_2 - 2^{14} + 1}.$$

Here, s_1 may be 1 or 0.
(normal or denormal representation).

If $x > 0$ is representable in this format, then

(a) x is 64-exact, and

(b) $-2^{14} + 1 - 63 \leq expo(x) \leq 2^{14}$.

In particular, x is also representable in the internal format of the K5 FPU.

IEEE Rounding Modes

The IEEE specification supports 3 basic modes:

- $trunc(x,n) = \lfloor 2^{n-1} sig(x) \rfloor 2^{expo(x)-n+1}$
- $away(x,n) = \lceil 2^{n-1} sig(x) \rceil 2^{expo(x)-n+1}$
- If $z = \lfloor 2^{n-1} sig(x) \rfloor$, and $f = 2^{n-1} sig(x) - z$, then

$$near(x,n) = \begin{cases} trunc(x,n) & \text{if } f < 1/2 \\ away(x,n) & \text{if } f > 1/2 \\ trunc(x,n) & \text{if } f = 1/2 \text{ and } z \text{ is even} \\ away(x,n) & \text{if } f = 1/2 \text{ and } z \text{ is odd.} \end{cases}$$

Example: $x = 101.1101_2$, $n = 4$

$$\begin{gathered} sig(x) = 1.011101_2,\ expo(x) = 2 \\ trunc(x,4) = \lfloor 1011.101_2 \rfloor 2^{2-3} = 1011_2 \cdot 2^{-1} = 101.1_2 \\ away(x,4) = \lceil 1011.101_2 \rceil 2^{2-3} = 1100_2 \cdot 2^{-1} = 110.0_2 \\ z = \lfloor 1011.101_2 \rfloor = 1011_2,\ f = .101_2 \\ near(x,4) = away(x,4) = 110.0_2 \end{gathered}$$

Notation: For each mode, $rnd(x, mode, n) = mode(x,n)$.

Extracting Expert Knowledge: The Development of a Mathematical Proposition

Stage 1: Vague Principle

If x is sufficiently exact, then the same result may be computed by either

(a) truncating y and then adding it to x,

or

(b) adding y to x and then truncating the sum.

```
 1xxxxx.xxxxxxxx
   1yyy.yyyyyyyy···
--------------------
1zzzzzz.zzzzzzzz···
```

Development of a Proposition
Stage 2: Precise Statement

Lemma 1 *If* x *is* $[k + expo(x) - expo(y)]$*-exact, then*

$x + trunc(y, k) = trunc(x + y, k + expo(x + y) - expo(y)).$

$$\begin{array}{r} \overbrace{\texttt{1x}\overbrace{\texttt{xxxx.xxxxxxxx}}^{k}}^{k+expo(x)-expo(y)} \\ \texttt{1yyy.yyyyyyyy}\cdots \\ \hline \underbrace{\texttt{1zzzzzz.zzzzzzzz}}_{k+expo(x+y)-expo(y)}\cdots \end{array}$$

Development of a Proposition
Stage 3: Rigorous Proof

Lemma 1 *If x is $[k + expo(x) - expo(y)]$-exact, then*

$x + trunc(y, k) = trunc(x + y, k + expo(x + y) - expo(y))$.

Proof: Let

$$k' = k + expo(x) - expo(y)$$

and

$$k'' = k + expo(x + y) - expo(y).$$

Since x is k'-exact,

$$x2^{k'-1-expo(x)} = x2^{k-1-expo(y)} \in \mathbf{Z}$$

and

$$\begin{aligned}
&x + trunc(y, k) \\
&\quad = x + \lfloor 2^{k-1-expo(y)} y \rfloor 2^{expo(y)+1-k} \\
&\quad = (x2^{k-1-expo(y)} + \lfloor 2^{k-1-expo(y)} y \rfloor) 2^{expo(y)+1-k} \\
&\quad = \lfloor 2^{k-1-expo(y)} (x + y) \rfloor 2^{expo(y)+1-k} \\
&\quad = \lfloor 2^{k''-1-expo(x+y)} (x + y) \rfloor 2^{expo(x+y)+1-k''} \\
&\quad = trunc(x + y, k''). \Box
\end{aligned}$$

Development of a Proposition
Stage 4: Formalization

The following result was proved in three steps, using five previously established lemmas:

```
(defthm plus-trunc
    (implies
      (and (rationalp x)
           (> x 0)
           (rationalp y)
           (> y 0)
           (integerp k)
           (> k 0)
           (exactp x
                   (+ k (- (expo x) (expo y))))
           (> (+ k (- (expo x) (expo y)))
              0))
      (= (+ x (trunc y k))
         (trunc (+ x y)
                (+ k
                   (- (expo (+ x y))
                      (expo y)))))))
```

Exactness: Basic Results

Lemma 2 (Product) *If x is m-exact and y is n-exact, then xy is $(m+n)$-exact.*

Lemma 3 (Square Root) *If x is k-exact (any k) and x^2 is $2n$-exact, then x is n-exact.*

Lemma 4 (Difference) *If x and y are n-exact and*

$$expo(x-y)+k \leq min(expo(x), expo(y)),$$

then $x-y$ is $(n-k)$-exact.

Corollary 5 *If x and y are n-exact and*

$$|x-y| \leq min(|x|, |y|),$$

then $x-y$ is n-exact.

Lemma 6 (Successor) *If x is n-exact, then the smallest n-exact number greater than x is*

$$succ(x,n) = x + 2^{expo(x)+1-n}.$$

Rounding: Basic Results

Lemma 7 (Exactness) *For any mode,*

(a) $rnd(x, mode, n)$ *is* n*-exact*

(b) If x *is* n*-exact, then* $rnd(x, mode, n) = x$

Lemma 8 (Error Bounds)

(a) $x \geq trunc(x, n) > x - 2^{expo(x)-n+1} \geq x(1 - 2^{-n+1})$

(b) $x \leq away(x, n) < x + 2^{expo(x)-n+1} \leq x(1 + 2^{-n+1})$

Lemma 9 (Monotonicity) *If* $x \leq y$*, then* $rnd(x, mode, n) \leq rnd(y, mode, n)$*.*

Lemma 10 (Rounding Boundaries) *Let* $x \leq y$*.*

(a) $trunc(x, n) < trunc(y, n)$
$\Leftrightarrow x < z \leq y$ *for some* n*-exact* z

(b) $away(x, n) < away(y, n)$
$\Leftrightarrow x \leq z < y$ *for some* n*-exact* z

(c) $near(x, n) < near(y, n)$
$\Rightarrow x \leq z \leq y$ *for some* $(n+1)$*-exact* z

Lemma 11 (Decomposition)
If $m \geq n \geq 1$ *and* $mode \neq near$*, then*

$$rnd(rnd(x, mode, m), mode, n) = rnd(x, mode, n).$$

K5 Square Root Algorithm

Input: P, $mode$, $prec$
Output: $rnd(\sqrt{P}, mode, prec)$

(1) If $P = 0$, then return 0; otherwise:

(2) An initial approximation r_0 to $1/\sqrt{P}$ is derived from a table.

(3) Three Newton-Raphson iterations yield r_1, r_2, and r_3.

(4) Two estimates of $\sqrt{P}$: $q_0 = away(r_3 P, 32)$ and $q = trunc(r_3 P, 64)$. If $q_0^2 = p$, then return $rnd(q_0, mode, prec)$; otherwise:

(5) A correction term res is computed as an approximation to $\sqrt{P} - q$. The resulting estimate $q + res$ is accurate to 74 bits.

(6) If $q{+}res$ is not too close to a rounding boundary, return $rnd(q + res, mode, prec)$; the remaining boundary cases are handled separately.

A Simple Programming Language

The algorithm is encoded as a program, $FSQRT$, in a language that includes statements of two types:

(1) Assignment:

$$\langle var \rangle \leftarrow \langle expr \rangle$$

where $\langle expr \rangle$ is one of the following:

- $rnd(x, mode, n)$ or $rnd(x \circ y, mode, n)$
- $sig(x)$
- $lookup(x)$

x and y may be variables or constants; "$\circ$" denotes "$+$", "$-$" , or "$\cdot$" ; the value of $\langle expr \rangle$ must be representable in the K5 FPU internal format.

(2) Conditional:

$$\textbf{if } \langle test \rangle \textbf{ then } \langle prog_1 \rangle \textbf{ else } \langle prog_2 \rangle$$

where $\langle test \rangle$ is a disjunction of equations or in-equalities.

if $P = 0$ then
  $sqrt \leftarrow 0$
else

(1st NR iteration *)*
  $r_0 \leftarrow lookup(P)$
  $P_h \leftarrow trunc(P, 32)$
  $s_0 \leftarrow away(r_0 P_h, 32)$
  $t_0 \leftarrow away(\frac{1}{2} r_0, 32)$
  $u_0 \leftarrow away(s_0 r_0, 32)$
  $v_0 \leftarrow trunc(3 - u_0, 32)$
  $r_1 \leftarrow trunc(v_0 t_0, 32)$

  (2nd NR iteration *)*
  $s_1 \leftarrow away(r_1 P_h, 32)$
  $t_1 \leftarrow away(\frac{1}{2} r_1, 32)$
  $u_1 \leftarrow away(s_1 r_1, 32)$
  $v_1 \leftarrow trunc(3 - u_1, 32)$
  $r_2 \leftarrow trunc(v_1 t_1, 32)$

  (3rd NR iteration *)*
  $s_2 \leftarrow away(r_2 P, 64)$
  $t_2 \leftarrow away(\frac{1}{2} r_2, 64)$
  $u_2 \leftarrow away(s_2 r_2, 64)$
  $v_2 \leftarrow trunc(3 - u_2, 64)$
  $r_3 \leftarrow trunc(v_2 t_2, 64)$

(test for exactness *)*
  $q \leftarrow trunc(r_3 P, 64)$
  $q_0 \leftarrow away(r_3 P, 32)$
  $P_0 \leftarrow trunc(q_0^2, 64)$
  $rem_0 \leftarrow trunc(P - P_0, 64)$
  if $rem_0 = 0$ then
    $sqrt \leftarrow rnd(q_0, mode, prec)$
  else *(* refinement *)*
    $diff \leftarrow near(q - q_0, 64)$
    $q_2 \leftarrow near(2q_0, 64)$
    $prod \leftarrow away(q_2 diff, 64)$
    $sq \leftarrow away(diff^2, 64)$
    $t_3 \leftarrow near(\frac{1}{2} r_3, 64)$
    $rem_1 \leftarrow trunc(rem_0 - prod, 64)$
    $rem_2 \leftarrow trunc(rem_1 - sq, 64)$
    $res \leftarrow odd(rem_2 t_3, 64)$

  (final computation *)*
    $root_0 \leftarrow trunc(q + res, 64)$
    $root_1 \leftarrow near(q + res, 64)$
    $root_2 \leftarrow away(q + res, 64)$
    $used \leftarrow trunc(root_0 - q, 64)$
    $end \leftarrow trunc(res - used, 64)$
    $ulp \leftarrow trunc(root_2 - root_0, 64)$
    $aeb \leftarrow trunc(ulp + end, 9)$
    $bits \leftarrow sig(aeb)$
    if $bits \notin \{2 - 2^{-8}, 1 + 2^{-1} - 2^{-8}\}$ then
      $sqrt \leftarrow rnd(q + res, mode, prec)$
    else if $mode \neq near$ then
      $det \leftarrow away(root_2 root_2, 64)$
      if $P \geq det$
        then $sqrt \leftarrow rnd(root_1 + end, mode, prec)$
        else $sqrt \leftarrow rnd(root_0 + end, mode, prec)$
    else *(* mode = near *)*
      $det \leftarrow trunc(root_1 root_2, 64)$
      if $P \leq det$ then
        $sqrt \leftarrow near(root_0 + end, prec)$
      else if $bits = 2 - 2^{-8}$ then
        $ends \leftarrow trunc(\frac{1}{16} end, 64)$
        $sqrt \leftarrow near(root_2 + ends, prec)$
      else $sqrt \leftarrow near(root_2 - end, prec)$

Correctness of FSQRT

IEEE compliance:

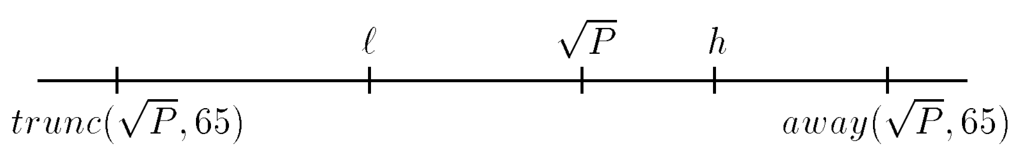
Theorem 1 *Let P be a positive double-extended floating-point number, let mode be an IEEE rounding mode and let $prec \in \mathbf{Z}$, $0 < prec \leq 64$. Then FSQRT executes successfully on $\langle P, mode, prec \rangle$ and*

$$sqrt = rnd(\sqrt{P}, mode, prec).$$

Equivalent rational version, checked with ACL2:

Theorem 2 *Let P be a positive double-extended floating-point number, let mode be an IEEE rounding mode and let $prec \in \mathbf{Z}$, $0 < prec \leq 64$. Let $\ell, h \in \mathbf{Q}$ such that $\ell \geq 0$, $h \geq 0$, and $\ell^2 \leq P \leq h^2$. Then FSQRT executes successfully on $\langle P, mode, prec \rangle$ and*

$$rnd(\ell, mode, prec) \leq sqrt \leq rnd(h, mode, prec).$$

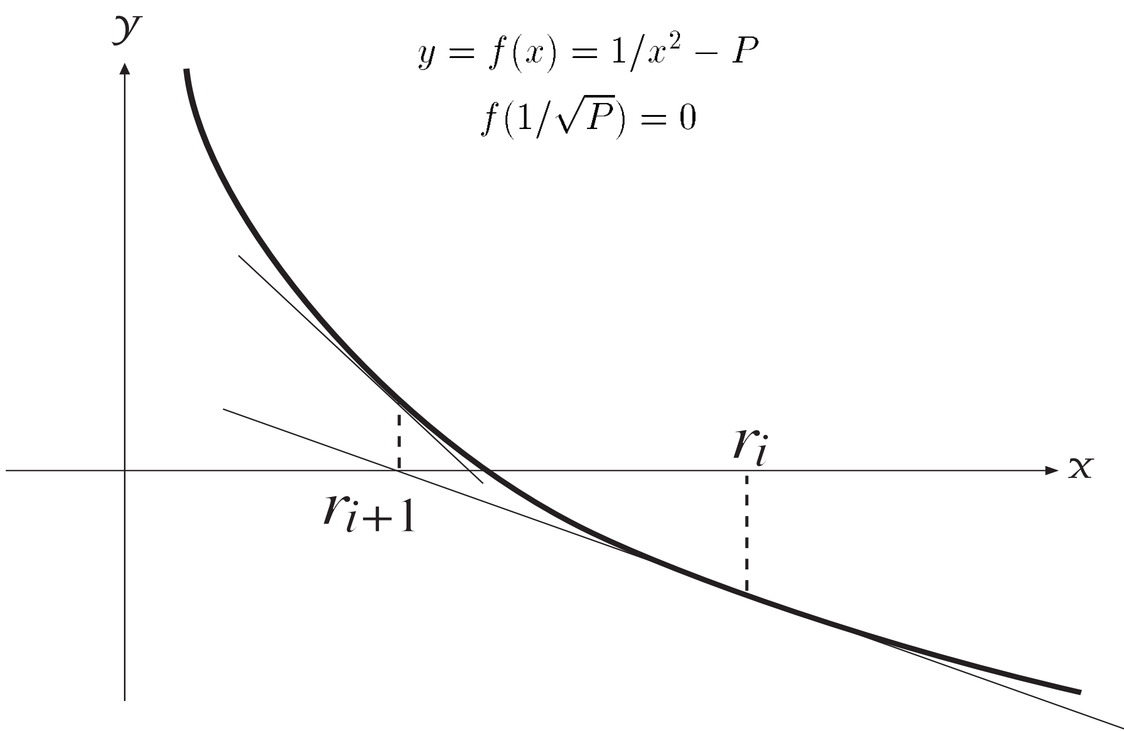


Formalization of Theorem 2

The proof of the following theorem required over 1400 lemmas:

```
(defthm correctness-of-fsqrt
    (implies (and (rationalp p )
                  (>= p 0)
                  (generalized-floating-point-p p 64 15)
                  (ieee-mode-p mode)
                  (integerp prec)
                  (> prec 0)
                  (<= prec 64)
                  (rationalp l)
                  (>= l 0)
                  (<= (* l l) p)
                  (rationalp h)
                  (>= h 0)
                  (>= (* h h) p))
             (and (<= (rnd l mode prec)
                      (compute 'sqrt (fsqrt)
                               (list p mode prec)))
                  (>= (rnd h mode prec)
                      (compute 'sqrt (fsqrt)
                               (list p mode prec)))
                  (all-exponents-bounded-p
                    (fsqrt) (list p mode prec)))))
```

Newton-Raphson Approximation



$$r_{i+1} = r_i - \frac{f(r_i)}{f'(r_i)} = \frac{r_i}{2}(3 - Pr_i^2)$$

$$\begin{aligned} \text{relative error} &= \frac{1/\sqrt{P} - r_i}{1/\sqrt{P}} = 1 - \sqrt{P}r_i \\ &\leq (1 - \sqrt{P}r_i)(1 + \sqrt{P}r_i) = 1 - Pr_i^2 \end{aligned}$$

Lemma 12 *If* $Pr_i^2 \leq 4$, *then* $0 \leq 1 - Pr_{i+1}^2 \leq (1 - Pr_i^2)^2$.

Initial Approximation to $1/\sqrt{P}$

Our initial approximation is given by

$$r_0 = lookup(P) = trunc(\frac{1}{\sqrt{trunc(P,7)}}, 7),$$

which yields

$$1 - Pr_0^2 < 2^{-5}.$$

The value of $lookup(P)$ may be computed efficiently with a table: Let

$$trunc(P,7) = 2^{2h+b}c,$$

and

$$\mathcal{L}(b,c) = trunc(\frac{2}{\sqrt{2^b c}}, 7).$$

Then $\mathcal{L}$ may be implemented as a table of 128 6-bit entries, and

$$lookup(P) = \mathcal{L}(b,c)/2^{h+1}.$$

But how can we define $\mathcal{L}$ in ACL2 and formally estimate $1 - Pr_0^2$?

Formal Definition of Lookup Table

Let $c = 1 + 2^{-6}k$ $(0 \le k < 64)$, let $b \in \{0,1\}$, and for $0 \le j < 127$, let α_j be the j^{th} entry in the sequence of binary integers

$$
\begin{gathered}
1000000, 111111, 111110, 111101, 111100, 111011, 111010, 111001,\\
111000, 110111, 110111, 110110, 110101, 110100, 110011, 110011,\\
110010, 110001, 110001, 110000, 101111, 101111, 101110, 101101,\\
101101, 101100, 101011, 101011, 101010, 101010, 101001, 101001,\\
101000, 100111, 100111, 100110, 100110, 100101, 100101, 100100,\\
100100, 100011, 100011, 100010, 100010, 100010, 100001, 100001,\\
100000, 100000, 011111, 011111, 011111, 011110, 011110, 011101,\\
011101, 011101, 011100, 011100, 011011, 011011, 011011, 011010,\\
011010, 011001, 011001, 011000, 010111, 010111, 010110, 010101,\\
010101, 010100, 010100, 010011, 010011, 010010, 010001, 010001,\\
010000, 010000, 001111, 001111, 001111, 001110, 001110, 001101,\\
001101, 001100, 001100, 001011, 001011, 001011, 001010, 001010,\\
001001, 001001, 001001, 001000, 001000, 001000, 000111, 000111,\\
000111, 000110, 000110, 000101, 000101, 000101, 000101, 000100,\\
000100, 000100, 000011, 000011, 000011, 000010, 000010, 000010,\\
000010, 000001, 000001, 000001, 000001, 000000, 000000, 000000.
\end{gathered}
$$

Then $\mathcal{L}(b, c) = 1 + 2^{-6}\alpha_{k+64b}$.

Formal Derivation of Error Estimate

Lemma 13 *For all b and c,*

$$1 - 2^{-5} < 2^{b-2}c\mathcal{L}(b,c)^2 < 2^{b-2}(c+2^{-6})\mathcal{L}(b,c)^2 < 1 + 2^{-5}.$$

Proof: Direct computation.

Corollary 14 $|1 - P[lookup(P)]^2| < 2^{-5}$.

Proof: Let $trunc(P,7) = 2^{2h+b}c$. Then

$$2^{2h+b}c \leq P < 2^{2h+b}(c + 2^{-6}),$$

$$lookup(P) = \mathcal{L}(b,c)/2^{h+1},$$

and

$$\begin{aligned}
1 - 2^{-5} &< 2^{b-2}c\mathcal{L}(b,c)^2 \\
&= [2^{2h+b}c][\mathcal{L}(b,c)^2/2^{2h+2}] \\
&\leq P[lookup(P)]^2 \\
&< [2^{2h+b}(c+2^{-6})][\mathcal{L}(b,c)^2/2^{2h+2}] \\
&= 2^{b-2}(c+2^{-6})\mathcal{L}(b,c)^2 \\
&< 1 + 2^{-5}. \square
\end{aligned}$$

Newton-Raphson: 3 Iterations

$r_0 \leftarrow lookup(P)$	$(* \approx 1/\sqrt{P}\ *)$
$P_h \leftarrow trunc(P, 32)$	$(* \approx P\ *)$
$s_0 \leftarrow away(r_0 P_h, 32)$	$(* \approx P r_0\ *)$
$t_0 \leftarrow away(\frac{1}{2} r_0, 32)$	$(* = \frac{r_0}{2}\ *)$
$u_0 \leftarrow away(s_0 r_0, 32)$	$(* \approx P r_0^2\ *)$
$v_0 \leftarrow trunc(3 - u_0, 32)$	$(* \approx 3 - P r_0^2\ *)$
$r_1 \leftarrow trunc(v_0 t_0, 32)$	$(* \approx \frac{r_0}{2}(3 - P r_0^2)\ *)$

Lemma 15 $|1 - P r_1^2| \leq 2^{-10}(1 + 2^{-16})$.

$s_1 \leftarrow away(r_1 P_h, 32)$	$(* \approx P r_1\ *)$
$t_1 \leftarrow away(\frac{1}{2} r_1, 32)$	$(* = \frac{r_1}{2}\ *)$
$u_1 \leftarrow away(s_1 r_1, 32)$	$(* \approx P r_1^2\ *)$
$v_1 \leftarrow trunc(3 - u_1, 32)$	$(* \approx 3 - P r_1^2\ *)$
$r_2 \leftarrow trunc(v_1 t_1, 32)$	$(* \approx \frac{r_1}{2}(3 - P r_1^2)\ *)$

Lemma 16 $|1 - P r_2^2| \leq 2^{-20}(1 + 2^{-5})$.

$s_2 \leftarrow away(r_2 P, 64)$	$(* \geq P r_2\ *)$
$t_2 \leftarrow away(\frac{1}{2} r_2, 64)$	$(* = \frac{r_2}{2}\ *)$
$u_2 \leftarrow away(s_2 r_2, 64)$	$(* \geq P r_2^2\ *)$
$v_2 \leftarrow trunc(3 - u_2, 32)$	$(* \leq 3 - P r_2^2\ *)$
$r_3 \leftarrow trunc(v_2 t_2, 32)$	$(* \leq \frac{r_2}{2}(3 - P r_2^2)\ *)$

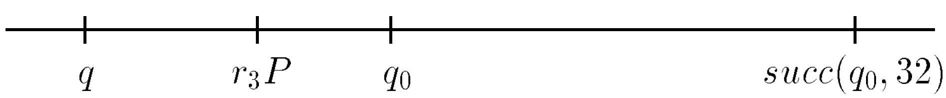
Lemma 17 $0 \leq 1 - P r_3^2 < 2^{-39}$.

Initial Approximation of $\sqrt{P}$

We compute two candidate approximations:

$q \leftarrow trunc(r_3 P, 64)$	(* $\leq \sqrt{P}$ *)
$q_0 \leftarrow away(r_3 P, 32)$	(* $\approx \sqrt{P}$ *)
$P_0 \leftarrow trunc(q_0^2, 64)$	(* $= q_0^2$ *)
$rem_0 \leftarrow trunc(P - P_0, 64)$	(* $= P - q_0^2$ *)
if $rem_0 = 0$	(* $P = q_0^2$ *)
then $sqrt \leftarrow rnd(q_0, mode, prec)$	
else	
$\vdots$	

Lemma 18 *If $P = x^2$ has a floating-point solution, then $P = q_0^2$.*



Henceforth, we assume $rem_0 \neq 0$; we take q as our initial approximation:

Lemma 19 $0 < P - q^2 < 2^{-38} P$.

Refinement

We shall derive a refined approximation $q + res$, based on the estimate

$$\sqrt{P} - q = \frac{P - q^2}{\sqrt{P} + q} > \frac{P - q^2}{2\sqrt{P}} > (P - q^2)\frac{r_3}{2}.$$

$P - q^2$ may be computed using the identity

$$P - q^2 = (P - q_0^2) - 2q_0(q - q_0) - (q - q_0)^2 :$$

$diff \leftarrow near(q - q_0, 64)$	$(* = q - q_0 *)$
$q_2 \leftarrow near(2q_0, 64)$	$(* = 2q_0 *)$
$prod \leftarrow away(q_2 diff, 64)$	$(* = 2q_0(q - q_0) *)$
$sq \leftarrow away(diff^2, 64)$	$(* = (q - q_0)^2 *)$
$t_3 \leftarrow near(\frac{1}{2}r_3, 64)$	$(* = \frac{r_3}{2} *)$
$rem_1 \leftarrow trunc(rem_0 - prod, 64)$	$(* = (P - q_0^2) - 2q_0(q - q_0) *)$
$rem_2 \leftarrow trunc(rem_1 - sq, 64)$	$(* = trunc(P - q^2, 64) *)$
$res \leftarrow trunc(rem_2 t_3, 64)$	$(* \ (q + res)^2 < P *)$

Lemma 20 $(1 - 2^{-74})P < (q + res)^2 < P.$

Final Value

In order to terminate with the assignment

$$sqrt \leftarrow rnd(q + res, mode, prec)$$

we must be able to prove two claims:

(1) $q + res$ is "not too big": if $P \leq h^2$, then

$$rnd(q + res, mode, prec) \leq rnd(h, mode, prec)$$

(2) $q + res$ is "not too small": if $P \geq \ell^2$, then

$$rnd(q + res, mode, prec) \geq rnd(\ell, mode, prec)$$

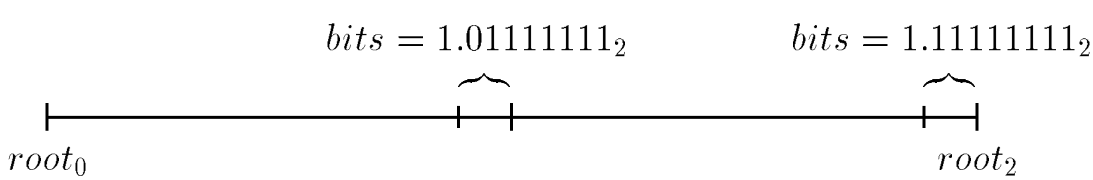
The first is a trivial consequence of Lemma 20, but (2) may fail if $q + res$ is too close to a "rounding boundary".

Test for Boundary Case

$root_0 \leftarrow trunc(q + res, 64)$	(* $\leq q + res$ *)
$root_1 \leftarrow near(q + res, 64)$	(* $root_0$ or $root_2$ *)
$root_2 \leftarrow away(q + res, 64)$	(* $\geq q + res$ *)
$used \leftarrow trunc(root_0 - q, 64)$	(* $root_0 - q$ *)
$end \leftarrow trunc(res - used, 64)$	(* $trunc(q + res - root_0, 64)$ *)
$ulp \leftarrow trunc(root_2 - root_0, 64)$	(* $root_2 - root_0$ *)
$aeb \leftarrow trunc(ulp + end, 9)$	
$bits \leftarrow sig(aeb)$	(* top 8 lost bits of $q + res$*)
if $bits \notin \{(1.01111111_2, 1.11111111_2\}$	(* exclude boundary cases *)
then $sqrt \leftarrow rnd(q + res, mode, prec)$	

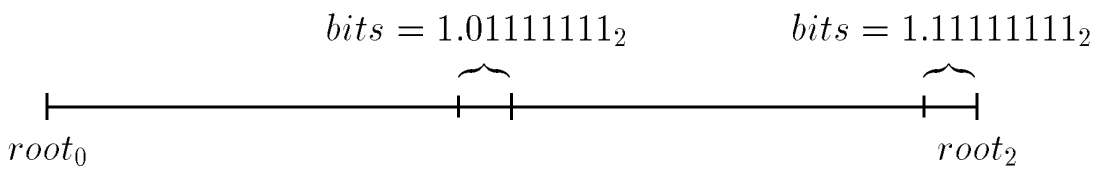
$$
\begin{array}{lllllllllll}
q + res & = (1\,.\,b_1 & b_2 & \ldots b_{62} & b_{63} & b_{64} & b_{65} & \ldots)_2 & \cdot\ 2^e \\
root_0 & = (1\,.\,b_1 & b_2 & \ldots b_{62} & b_{63} & 0 & 0 & \ldots)_2 & \cdot\ 2^e \\
ulp & = (0\,.\,0 & 0 & \ldots 0 & 1 & 0 & 0 & \ldots)_2 & \cdot\ 2^e \\
end & = (0\,.\,0 & 0 & \ldots 0 & 0 & b_{64} & b_{65} & \ldots)_2 & \cdot\ 2^e \\
aeb & = (0\,.\,0 & 0 & \ldots 0 & 1 & b_{64} & b_{65} & \ldots)_2 & \cdot\ 2^e \\
bits & = (1\,.\,b_{64} & b_{65} \ldots b_{71})_2 & & & & & &
\end{array}
$$

Lemma 21 $trunc(q + res, 72) = root_0 + (bits - 1)ulp.$



In all other cases, Inequality (2) holds.

Completing the Proof



The remaining cases are handled by a fairly straight-forward analysis:

else if $mode \neq near$ then
    $det \leftarrow away(root_2 root_2, 64)$
    if $P \geq det$
        then $sqrt \leftarrow rnd(root_1 + end, mode, prec)$
        else $sqrt \leftarrow rnd(root_0 + end, mode, prec)$
else (* $mode = near$ *)
    $det \leftarrow trunc(root_1 root_2, 64)$
    if $P \leq det$ then
        $sqrt \leftarrow near(root_0 + end, prec)$
    else if $bits = 1.11111111_2$ then
        $ends \leftarrow trunc(\frac{1}{16} end, 64)$
        $sqrt \leftarrow near(root_2 + ends, prec)$
    else $sqrt \leftarrow near(root_2 - end, prec)$

A Flaw in the Algorithm?

How do we know that $q + res$ is not 64-exact, i.e., that $root_0 \neq root_2$?

We do know the following:

- q and res are both 64-exact
- $0 < res < 2^{-38} q$
- $q + res \approx \sqrt{P}$ (relative error $< 2^{-74}$)
- $\sqrt{P}$ is not a floating-point number

$$
\begin{aligned}
q &= (1\,.\,x_1 x_2 \;\ldots\ldots\ldots\ldots\; x_{63} 0\,0 \;\ldots\ldots\ldots\ldots)_2 \cdot 2_e \\
res &= (0\,.\,0\,0 \;\ldots\ldots\; 0\,1\, y_1 y_2 \;\ldots\ldots\; y_{63} 0\,0 \;\ldots\ldots)_2 \cdot 2_e \\
\sqrt{P} &= (1\,.\,z_1 z_2 \;\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots)_2 \cdot 2_e
\end{aligned}
$$

But it still seems possible that $q + res$ is 64-exact.

Is there a modification of the algorithm that
(a) admits a proof of correctness,
(b) does not increase the cycle count, and
(c) is easy to implement?

Lynch's Solution: Odd Rounding

We define a new rounding mode for computing res:

$$odd(x, n) = trunc(x, n-1) + 2^{expo(x)+1-n}$$

$$\boxed{res \leftarrow odd(rem_2 t_3, 64) \quad (*\ (q + res)^2 \approx P\ *)}$$

This ensures that $q + res$ is not exact, and still "not too much bigger than $\sqrt{P}$".

Lemma 22 *If $x > 0$, $y > 0$, $k > 1$, and x is n-exact, where $n = k - 1 + expo(x) - expo(y)$, then*

$$x + odd(y, k) = odd(x + y, k + expo(x + y) - expo(y)).$$

Corollary 23 *For some $m \geq 102$,*

$$q + res = odd(q + rem_2 t_3, m).$$

Lemma 24 *$odd(x, n+1)$ is not n-exact.*

Corollary 25 *$q + res$ is not 65-exact.*

"Not Too Much Bigger"

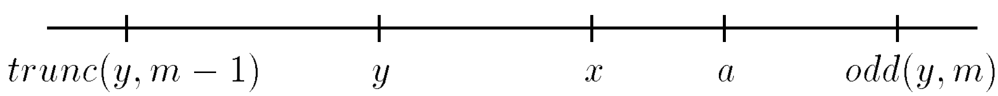
Lemma 26 *If $0 < y < x$, $k + 2 \leq m$, and mode is an IEEE rounding mode, then*

$$rnd(odd(y, m), mode, k) \leq rnd(x, mode, k).$$

Proof: Let a be $(k + 1)$-exact. It suffices to show that a cannot satisfy $x \leq a \leq odd(y, m)$.

If $trunc(y, m - 1) \leq y < x \leq a$, then since both a and $trunc(y, m - 1)$ are $(m - 1)$-exact,

$$\begin{aligned} a &\geq trunc(y, m-1) + 2^{expo(trunc(y,m-1))+1-(m-1)} \\ &> trunc(y, m-1) + 2^{expo(y)+1-m} \\ &= odd(y, m). \square \end{aligned}$$



Corollary 27 *If $h \geq 0$ and $P \leq h^2$, then*

$$rnd(q + res, mode, prec) \leq rnd(h, mode, prec).$$

Proof: $q + res = odd(q + rem_2 t_3, m)$, where

$$m \geq 102 \geq prec + 2. \square$$

Summary

What have we accomplished?

- Rigorous mathematical theory of floating-point rounding
- Formal representation of the K5 square root microcode
- Detection of a possible flaw
- Proof of correctness

What exactly have we proved?

- Theorem 1 (IEEE compliance): Stage 3
- Theorem 2 (rational version): Stage 4

What haven't we proved?

- The definitions are "right"
- The implementation is correct
- ACL2 is sound ("Quis custodiet ipsos custodes?")