Square—Root Verification

Jason Hickey
Mark Hayden
May 10, 1994

Outline

- **Restoring algorithm**
- **First non-restoring algorithm**
  - Proof outline
  - Nuprl proof
- **Transformation to second, more efficient nonrestoring algorithm**

Restoring algorithm

```
fun update1 (State{stage, x, y, b}) =
      let val y' = if (y + b) * (y + b) > x then y else (y + b)
      in
            State{stage = stage - 1, x = x, y = y', b = b div 2}
      end

fun sqrt' n debugFn updateFn state =
     let val dFn = debugFn n
     in (dFn o (iterate (updateFn o dFn) n)) state
     end

sqrt' (n - 1) debug update1 (State{      stage = n,
                                         x = radicand,
                                         y = 0,
                                         b = 2 ** (n - 2)})
```

Non—Restoring Algorithm

```
fun update2 (State{x, y, b}) =
    let   val x' = x - y * y
          val y' =   if x' > 0 then
                         y + b
                     else if x' < 0 then
                         y - b
          else
                         y
    in
          State{x = x, y = y', b = b div 2}
    end

sqrt' (n - 1) debug update2 (State{      stage = n,
                                          x = radicand,
                                          y = 2 ** (n - 1),
                                          b = 2 ** (n - 2)})
```

Non—Restoring Trace

n=14	x=-000011111111100001110111000000	y= 000010000000000000000000000000
n=13	x=-000000111111100001110111000000	y= 000000100000000000000000000000
n=12	x=-000000001111100001110111000000	y= 000000001000000000000000000000
n=11	x=-000000000011100001110111000000	y= 000000000010000000000000000000
n=10	x=-000000000000100001110111000000	y= 000000000000100000000000000000
n= 9	x= 000000000000001110001001000000	y= 000000000000001000000000000000
n= 8	x=-000000000000000101110111000000	y= 000000000000000110000000000000
n= 7	x= 000000000000000101001001000000	y= 000000000000000010100000000000
n= 6	x=-000000000000000000000111000000	y= 000000000000000001011000000000
n= 5	x= 000000000000000010100101000000	y= 000000000000000000101010000000
n= 4	x= 000000000000000001010000000000	y= 000000000000000000010101100000
n= 3	x= 000000000000000000100100110000	y= 000000000000000000001010111000
n= 2	x= 000000000000000000001110111100	y= 000000000000000000000101011110
n= 1	x= 000000000000000000000011111111	y= 000000000000000000000101011111

Proof Outline Update

$\texttt{fun}\ Update(x, y, b) =$

$\quad \{b = 2^{i\ \ 1} \wedge (y \quad 2^i)^2 \leq radicand < (y + 2^i)^2 \wedge y \text{ rem } 2^i = 0 \wedge x = radicand\}$

$\quad \texttt{if}\ x > y^2\ \texttt{then}$

$\qquad \{b = 2^{i\ \ 1} \wedge (y \quad 2^i)^2 \leq radicand < (y + 2^i)^2 \wedge y \text{ rem } 2^i = 0 \wedge x = radicand \wedge x > y^2\}$

$\qquad y \leftarrow y + b$

$\qquad \{b = 2^{i\ \ 1} \wedge (y \quad 2^{i\ \ 1})^2 \leq radicand < (y + 2^{i\ \ 1})^2 \wedge y \text{ rem } 2^{i\ \ 1} = 0 \wedge x = radicand\}$

$\quad \texttt{else if}\ x < y^2\ \texttt{then}$

$\qquad \{b = 2^{i\ \ 1} \wedge (y \quad 2^i)^2 \leq radicand < (y + 2^i)^2 \wedge y \text{ rem } 2^i = 0 \wedge x = radicand \wedge x < y^2\}$

$\qquad y \leftarrow y \quad b$

$\qquad \{b = 2^{i\ \ 1} \wedge (y \quad 2^{i\ \ 1})^2 \leq radicand < (y + 2^{i\ \ 1})^2 \wedge y \text{ rem } 2^{i\ \ 1} = 0 \wedge x = radicand\}$

$\quad \texttt{fi}$

$\quad \{b = 2^{i\ \ 1} \wedge (y \quad 2^{i\ \ 1})^2 \leq radicand < (y + 2^{i\ \ 1})^2 \wedge y \text{ rem } 2^{i\ \ 1} = 0 \wedge x = radicand\}$

$\quad b \leftarrow b \quad 2;$

$\quad \{b = 2^{i\ \ 2} \wedge (y \quad 2^{i\ \ 1})^2 \leq radicand < (y + 2^{i\ \ 1})^2 \wedge y \text{ rem } 2^{i\ \ 1} = 0 \wedge x = radicand\}$

Nuprl Update Function

```
* ABS update
Update(state) == let p, y, b = state
                 in
                    let x = p - y * y
                        y = if 0 <z x then y + b else if x <z 0 then y - b else y fi  fi
                    in SqrtState(p, y , b ÷ 2)

* ABS update hyp
UpdateHyp(n, p, state) == let x, y, b = state
                          in
                             let b2 = 2^n
                             in
                               b = 2^(n - 1)
                               ∧ (y - b2) * (y - b2) < p
                               ∧ (y + b2) * (y + b2) > p
                               ∧ y rem b2 = 0
                               ∧ x = p
```

Square Root

```
* THM update thm
∀n:N+
   ∀radicand:Z
    ∀state:SqrtState
     UpdateHyp(n, radicand, state) ⇒ UpdateHyp(n - 1, radicand, Update(state))

* ABS sqrt it
SqrtIterate[n-bit] radicand == Iterate[n] λstate.Update(state) on SqrtState(radicand,
                                                                             2^n,
                                                                             2^(n - 1))
* THM sqrt it thm
∀n:N. ∀radicand:{1..(2^(2 * n)) }. UpdateHyp(0, radicand, SqrtIterate[n - 1-bit] radicand)
```

Second NR Algorithm

```
fun update3 (State{stage, x, y, b}) =
    let   val (x',y')  =    if x>0 then (x - y - b, y + 2 * b)
                            else if x<0 then (x + y - b, y - 2 * b)
                            else (x, y)
    in
    State{       x = x',
                 y = y' div 2,
                 b = b  div 4}
    end

    sqrt' (n - 1) debug update3 (State{       stage = n,
                                              x = radicand - 2 ** (2 * n - 2),
                                              y = 2 ** (2 * n - 2),
                                              b = 2 ** (2 * n - 4)})
```

Third NR Algorithm

```
fun update4 (State{x, y, b}) =
    let val (x',y') =     if x>0 then (x - 2 * y - b, y + b)
                else if x<0 then (x + 2 * y - b, y - b)
                else (x, y)
    in
        let val y'' = if stage = 2 then y' else y' div 2
        in
        State{          x = x',
                        y = y'',
                        b = b  div 4}
        end
    end

sqrt' (n - 1) debug update4 (State{     stage = n,
                                        x = radicand - 2 ** (2 * n - 2),
                                        y = 2 ** (2 * n - 3),
                                        b = 2 ** (2 * n - 4)})
```

Nuprl Algorithm

```
* ABS update 2
Update2(state) == let x,y,b = state
                  in let x',y' = if 0 <z x
                                 then <x - 2 * y - b,y + b>
                                 else if x <z 0
                                      then <(x + 2 * y) - b,y - b>
                                      else <x,y>
                                      fi
                     in <x', y' ÷ 2, b ÷ 4>

* ABS update 2 base
Update2Base(state) == let x,y,b = state
                      in if 0 <z x
                         then y + b
                         else if x <z 0 then y - b else y fi
                         fi
```

Transformation

```
* ABS update 2 trans
Update2Trans == λi,states.let state 1,state 2 = states
                             in let p1,y1,b1 = state 1
                                in let x2,y2,b2 = state 2
                                   in p1 - y1 * y1 = x2 ∧ y1 * b1 = y2 ∧ b1 * b1 = b2 ∧ b1 = 2^i

* THM update 2 base thm
∀s1:Z × Z × Z
   ∀s2:Z × Z × Z. Update2Trans 0 <s1,s2> ⇒ let p1,y1,b1 = Update(s1).in y1 = Update2Base(s2)

* THM update 2 thm
∀i:N+
   ∀state 1:Z × Z × Z
    ∀state 2:Z × Z × Z
     Update2Trans i <state 1,state 2>
     ⇒ Update2Trans (i - 1) <(λs.Update(s)) state 1,(λs.Update2(s)) state 2>

* THM update 2 it thm
∀n:N
   ∀state 1:Z × Z × Z
    ∀state 2:Z × Z × Z
     Update2Trans n <state 1,state 2>
     ⇒ Update2Trans 0 <Iterate[n] λs.Update(s) on state 1,Iterate[n] λs.Update2(s) on state 2>
```