```
#|

 Copyright (C) 1994 by Yuan Yu.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Yuan Yu PROVIDES ABSOLUTELY NO WARRANTY.  THE EVENT SCRIPT IS PROVIDED "AS IS"
 WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT
 LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 PARTICULAR PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
 SCRIPT IS WITH YOU.  SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF
 ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

 IN NO EVENT WILL Yuan Yu BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS,
 LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT
 OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS
 OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD
 PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR
 FOR ANY CLAIM BY ANY OTHER PARTY.

|#

;       Proof of the Correctness of an Integer Square Root Program
;
```

EVENT: Start with the library "mc20-2" using the compiled version.

```
#|
This is a revisit to our ISQRT proof.  Dr. Don Good talked with
Dr. Steve Zeigler, vice president for Ada products at Verdix, and
they agreed to let me publicize the machine codes generated by
their present Ada compiler.

The following Ada function ISQRT computes the integer square root of a
given nonnegative integer i.  This is our third proof about ISQRT.

function isqrt (i:integer) return integer is
  j : integer := (i / 2);
 begin
  while ((i / j) < j) loop
    j := (j + (i / j)) / 2;
```

```
 end loop;
 return j;
end isqrt;
```

Here is the MC68020 assembly code of the above ISQRT program. The code is from Dr. Steve Zeigler and generated by their present Ada compiler.

```
  1  function isqrt (i:integer) return integer is
     00000: link.w      a6, #-04
  2     j : integer := (i / 2);
     00004: move.l      d2, d1
     00006: bge.b       06    -> 0e
     00008: addi.l      #01, d1
     0000e: asr.l       #01, d1
  3  begin
  4     while not ((i / j) >= j) loop
     00010: move.l      d2, d0
     00012: divsl.l     d1, d0:d0
     00016: trapv
     00018: cmp.l       d0, d1
     0001a: ble.b       01c    -> 038
  5         j := (j + (i / j)) / 2;
     0001c: add.l       d1, d0
     0001e: trapv
     00020: move.l      d0, d1
     00022: bge.b       06    -> 02a
     00024: addi.l      #01, d1
     0002a: asr.l       #01, d1
  4     while not ((i / j) >= j) loop
     0002c: move.l      d2, d0
     0002e: divsl.l     d1, d0:d0
     00032: trapv
  6  end loop;
     00034: cmp.l       d0, d1
     00036: bgt.b       -01c    -> 01c
  7  return j;
     00038: move.l      d1, d0
     0003a: unlk        a6
     0003c: rts
  8 end isqrt;
```

```
0x4e56  0xfffc  0x2202  0x6c06  0x0681  0x0000  0x0001  0xe281
0x2002  0x4c41  0x0800  0x4e76  0xb280  0x6f1c  0xd081  0x4e76
0x2200  0x6c06  0x0681  0x0000  0x0001  0xe281  0x2002  0x4c41
```

```
0x0800  0x4e76  0xb280  0x6ee4  0x2001  0x4e5e  0x4e75

'(78      86      255     252     34      2       108     6
  6       129     0       0       0       1       226     129
  32      2       76      65      8       0       78      118
  178     128     111     28      208     129     78      118
  34      0       108     6       6       129     0       0
  0       1       226     129     32      2       76      65
  8       0       78      118     178     128     110     228
  32      1       78      94      78      117)
|#

; in the logic, the above program is defined by (isqrt-code).
```

DEFINITION:
ISQRT-CODE
= `'(78 86 255 252 34 2 108 6 6 129 0 0 0 1 226 129 32 2`
`76 65 8 0 78 118 178 128 111 28 208 129 78 118 34 0`
`108 6 6 129 0 0 0 1 226 129 32 2 76 65 8 0 78 118`
`178 128 110 228 32 1 78 94 78 117)`

DEFINITION: $\mathrm{sq}(x) = (x * x)$

```
; isqrt1 is a function in the Logic simulating the loop of the above code.
```

DEFINITION:
$\mathrm{isqrt1}(i, j)$
= **if** $j \simeq 0$ **then** $\mathrm{fix}(i)$
**elseif** $(i \div j) < j$ **then** $\mathrm{isqrt1}(i, (j + (i \div j)) \div 2)$
**else** $\mathrm{fix}(j)$ **endif**

```
; isqrt specifies the semantics of ISQRT in the Logic.  To see why
; the function isqrt computes the square root for any nonnegative
; integer input, please refer to the proof in file isqrt.events.
```

DEFINITION:
$\mathrm{isqrt}(i)$
= **let** *j1* **be** $((i \div 2) + (i \div (i \div 2))) \div 2$
**in**
**if** $i < \mathrm{sq}(i \div 2)$ **then** $\mathrm{isqrt1}(i, j1)$
**else** $i \div 2$ **endif endlet**

```
; the computation time of this program.
```

DEFINITION:

isqrt1-t $(i, j)$
= **if** $j \simeq \mathtt{0}$ **then** $\mathtt{0}$
**elseif** $(i \div j) < j$
**then** splus $(\mathtt{10}$, isqrt1-t $(i, (j + (i \div j)) \div \mathtt{2}))$
**else** $\mathtt{8}$ **endif**

DEFINITION:
isqrt-t $(i)$
= **let** *j1* **be** $((i \div \mathtt{2}) + (i \div (i \div \mathtt{2}))) \div \mathtt{2}$
**in**
**if** $i <$ sq $(i \div \mathtt{2})$ **then** splus $(\mathtt{14}$, isqrt1-t $(i,$ *j1*$))$
**else** $\mathtt{12}$ **endif endlet**

```
; enable a few functions.
```

EVENT: Enable iplus.

EVENT: Enable integerp.

EVENT: Enable iquotient.

EVENT: Enable ilessp.

```
; disable a few functions.
```

EVENT: Disable remainder.

EVENT: Disable quotient.

THEOREM: isqrt-no-overflow
(int-rangep $(\mathtt{2} * j, n) \wedge ((i \div j) < j)$)
$\rightarrow$ int-rangep $(j + (i \div j), n)$

THEOREM: j-nonzerop
$((\mathtt{1} < i) \wedge (\mathtt{0} < j)) \rightarrow (((j + (i \div j)) \div \mathtt{2}) \neq \mathtt{0})$

THEOREM: j-int-rangep
(int-rangep $(\mathtt{2} * j, n) \wedge ((i \div j) < j)$)
$\rightarrow$ int-rangep $(\mathtt{2} * ((j + (i \div j)) \div \mathtt{2}), n)$

```
; an induction hint.
```

DEFINITION:
isqrt-induct $(s,\ i,\ j)$
= **if** $j \simeq 0$ **then t**
**elseif** $(i \div j) < j$
**then** isqrt-induct (stepn $(s,\ 10)$, $i$, $(j + (i \div j)) \div 2$)
**else t endif**

```
; the properties of the initial state.
```

DEFINITION:
isqrt-statep $(s,\ i)$
= ((mc-status $(s)$ = 'running)
∧ evenp (mc-pc $(s)$)
∧ rom-addrp (mc-pc $(s)$, mc-mem $(s)$, 70)
∧ mcode-addrp (mc-pc $(s)$, mc-mem $(s)$, ISQRT-CODE)
∧ ram-addrp (sub (32, 8, read-sp $(s)$), mc-mem $(s)$, 12)
∧ $(i$ = iread-dn (32, 2, $s))$
∧ ilessp (1, $i$))

```
; an intermediate state s0.
```

DEFINITION:
isqrt-s0p $(s,\ i,\ j)$
= ((mc-status $(s)$ = 'running)
∧ evenp (mc-pc $(s)$)
∧ rom-addrp (sub (32, 44, mc-pc $(s)$), mc-mem $(s)$, 70)
∧ mcode-addrp (sub (32, 44, mc-pc $(s)$), mc-mem $(s)$, ISQRT-CODE)
∧ ram-addrp (sub (32, 4, read-an (32, 6, $s)$), mc-mem $(s)$, 12)
∧ $(i$ = iread-dn (32, 2, $s))$
∧ $(j$ = iread-dn (32, 1, $s))$
∧ int-rangep $(2 * j,\ 32)$
∧ ilessp (1, $i$)
∧ ilessp (0, $j$))

THEOREM: initial-j-int-rangep
int-rangep $(i,\ n)$ → int-rangep $(2 * (i \div 2),\ n)$

```
; from the initial state to exit.
```

THEOREM: isqrt-s-exit
(isqrt-statep $(s,\ i) \wedge (i \not< ((i \div 2) * (i \div 2))))$
→ ((mc-status (stepn $(s,\ 12)$) = 'running)
∧ (mc-pc (stepn $(s,\ 12)$) = rts-addr $(s)$)
∧ (iread-dn (32, 0, stepn $(s,\ 12)$) = $(i \div 2)$)
∧ (read-rn (32, 14, mc-rfile (stepn $(s,\ 12)$)))

$=$ read-rn (32, 14, mc-rfile ($s$)))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn ($s$, 12)))
$=$ add (32, read-sp ($s$), 4)))

THEOREM: isqrt-s-exit-rfile
(isqrt-statep ($s$, $i$) $\wedge$ ($i \not< ((i \div 2) * (i \div 2))$) $\wedge$ d2-7a2-5p ($rn$))
$\rightarrow$ (read-rn ($oplen$, $rn$, mc-rfile (stepn ($s$, 12)))
$=$ read-rn ($oplen$, $rn$, mc-rfile ($s$)))

THEOREM: isqrt-s-exit-mem
(isqrt-statep ($s$, $i$)
$\wedge$ ($i \not< ((i \div 2) * (i \div 2))$)
$\wedge$ disjoint ($x$, $k$, sub (32, 8, read-sp ($s$)), 12))
$\rightarrow$ (read-mem ($x$, mc-mem (stepn ($s$, 12)), $k$) = read-mem ($x$, mc-mem ($s$), $k$))

```
; from the initial state to s0.
```

THEOREM: isqrt-s-s0
**let** *j1* **be** $((i \div 2) + (i \div (i \div 2))) \div 2$
**in**
(isqrt-statep ($s$, $i$) $\wedge$ ($i < ((i \div 2) * (i \div 2))$))
$\rightarrow$ (isqrt-s0p (stepn ($s$, 14), $i$, *j1*)
$\wedge$ (linked-rts-addr (stepn ($s$, 14)) = rts-addr ($s$))
$\wedge$ (linked-a6 (stepn ($s$, 14)) = read-an (32, 6, $s$))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn ($s$, 14)))
$=$ sub (32, 4, read-sp ($s$)))) **endlet**

THEOREM: isqrt-s-s0-rfile
(isqrt-statep ($s$, $i$) $\wedge$ ($i < ((i \div 2) * (i \div 2))$) $\wedge$ d2-7a2-5p ($rn$))
$\rightarrow$ (read-rn ($oplen$, $rn$, mc-rfile (stepn ($s$, 14)))
$=$ read-rn ($oplen$, $rn$, mc-rfile ($s$)))

THEOREM: isqrt-s-s0-mem
(isqrt-statep ($s$, $i$)
$\wedge$ ($i < ((i \div 2) * (i \div 2))$)
$\wedge$ disjoint ($x$, $k$, sub (32, 8, read-sp ($s$)), 12))
$\rightarrow$ (read-mem ($x$, mc-mem (stepn ($s$, 14)), $k$) = read-mem ($x$, mc-mem ($s$), $k$))

```
; from s0 to exit (base case), or from s0 to s0 (induction case).
; base case: s0 --> exit.
```

THEOREM: isqrt-s0-exit-base
(isqrt-s0p ($s$, $i$, $j$) $\wedge$ ($(i \div j) \not< j$))
$\rightarrow$ ((mc-status (stepn ($s$, 8)) = 'running)
$\wedge$ (mc-pc (stepn ($s$, 8)) = linked-rts-addr ($s$))

$\wedge$ (iread-dn (32, 0, stepn (*s*, 8)) = *j*)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (*s*, 8))) = linked-a6 (*s*))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (*s*, 8)))
= add (32, read-an (32, 6, *s*), 8)))

THEOREM: isqrt1-s0-exit-rfile-base
(isqrt-s0p (*s*, *i*, *j*) $\wedge$ ((*i* $\div$ *j*) $\not<$ *j*) $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 8)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: isqrt1-s0-exit-mem-base
(isqrt-s0p (*s*, *i*, *j*) $\wedge$ ((*i* $\div$ *j*) $\not<$ *j*))
$\rightarrow$ (read-mem (*x*, mc-mem (stepn (*s*, 8)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; induction case: s0 --> s0.
```

THEOREM: isqrt-s0-s0
(isqrt-s0p (*s*, *i*, *j*) $\wedge$ ((*i* $\div$ *j*) < *j*))
$\rightarrow$ (isqrt-s0p (stepn (*s*, 10), *i*, (*j* + (*i* $\div$ *j*)) $\div$ 2)
$\wedge$ (read-rn (*oplen*, 14, mc-rfile (stepn (*s*, 10)))
= read-rn (*oplen*, 14, mc-rfile (*s*)))
$\wedge$ (linked-a6 (stepn (*s*, 10)) = linked-a6 (*s*))
$\wedge$ (linked-rts-addr (stepn (*s*, 10)) = linked-rts-addr (*s*)))

THEOREM: isqrt-s0-s0-rfile
(isqrt-s0p (*s*, *i*, *j*) $\wedge$ ((*i* $\div$ *j*) < *j*) $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 10)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: isqrt-s0-s0-mem
(isqrt-s0p (*s*, *i*, *j*)
$\wedge$ ((*i* $\div$ *j*) < *j*)
$\wedge$ disjoint (*x*, *k*, sub (32, 8, read-an (32, 6, *s*)), 20))
$\rightarrow$ (read-mem (*x*, mc-mem (stepn (*s*, 10)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

THEOREM: isqrt-s0p-j_nonzerop
isqrt-s0p (*s*, *i*, *j*) $\rightarrow$ ((*j* $\neq$ 0) $\wedge$ (*j* $\in$ **N**))

EVENT: Disable isqrt-statep.

EVENT: Disable isqrt-s0p.

```
; put together:  s0 --> exit.
```

THEOREM: isqrt-s0-exit
**let** *sn* **be** stepn (*s*, isqrt1-t (*i*, *j*))
**in**
isqrt-s0p (*s*, *i*, *j*)
→ ((mc-status (*sn*) = 'running)
∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
∧ (iread-dn (32, 0, *sn*) = isqrt1 (*i*, *j*))
∧ (read-rn (32, 14, mc-rfile (*sn*)) = linked-a6 (*s*))
∧ (read-rn (32, 15, mc-rfile (*sn*))
= add (32, read-an (32, 6, *s*), 8))) **endlet**

THEOREM: isqrt-s0-exit-rfile
(isqrt-s0p (*s*, *i*, *j*) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, isqrt1-t (*i*, *j*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: isqrt-s0-exit-mem
(isqrt-s0p (*s*, *i*, *j*) ∧ disjoint (*x*, *k*, sub (32, 8, read-an (32, 6, *s*)), 20))
→ (read-mem (*x*, mc-mem (stepn (*s*, isqrt1-t (*i*, *j*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*))

EVENT: Disable isqrt-s0p-j_nonzerop.

```
; ISQRT program is correct.
; after an execution of this program,  the machine state satisfies:
;  0. normal exit.
;  1. the pc is returned to the next instruction of the caller.
;  2. the result -- ISQRT(i), is stored in D0.
;  3. a6, used by LINK, is restored to its original content.
;  4. the stack pointer sp(a7) is updated correctly to pop off one frame.
; the registers not touched by this program still have their original values.
; the memory is correctly changed -- the local effect on memory.
```

THEOREM: isqrt-correctness
**let** *sn* **be** stepn (*s*, isqrt-t (*i*))
**in**
isqrt-statep (*s*, *i*)
→ ((mc-status (*sn*) = 'running)
∧ (mc-pc (*sn*) = rts-addr (*s*))
∧ (read-rn (32, 14, mc-rfile (*sn*)) = read-an (32, 6, *s*))
∧ (read-rn (32, 15, mc-rfile (*sn*))
= add (32, read-an (32, 7, *s*), 4))
∧ (d2-7a2-5p (*rn*)
→ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))

$$
\begin{array}{ll}
 & \quad = \quad \text{read-rn}\,(\mathit{oplen},\ \mathit{rn},\ \text{mc-rfile}\,(s)))) \\
\wedge & (\text{disjoint}\,(x,\ k,\ \text{sub}\,(\mathtt{32},\ \mathtt{12},\ \text{read-sp}\,(s)),\ \mathtt{20}) \\
 & \quad \rightarrow \quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(\mathit{sn}),\ k) \\
 & \quad\quad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k))) \\
\wedge & (\text{iread-dn}\,(\mathtt{32},\ \mathtt{0},\ \mathit{sn}) = \text{isqrt}\,(i))) \ \textbf{endlet}
\end{array}
$$

EVENT: Disable isqrt-t.

```
; start to prove isqrt correct.
```

THEOREM: isqrt1-lower-bound
$(j \not\simeq \mathtt{0}) \rightarrow (i \not< \text{sq}\,(\text{isqrt1}\,(i, j)))$

THEOREM: quotient-by-2
$((x \div \mathtt{2}) + (x \div \mathtt{2})) \not< (x - 1)$

THEOREM: main-trick
$\text{sq}\,(1 + ((j + k) \div \mathtt{2})) \not< ((j * k) + j)$

THEOREM: sq-add1-non-zero
$\text{sq}\,(1 + x) \neq \mathtt{0}$

THEOREM: main
$(j \not\simeq \mathtt{0}) \rightarrow (i < \text{sq}\,(1 + ((j + (i \div j)) \div \mathtt{2})))$

THEOREM: isqrt1-upper-bound
$(i < \text{sq}\,(1 + j)) \rightarrow (i < \text{sq}\,(1 + \text{isqrt1}\,(i, j)))$

THEOREM: isqrt->isqrt1
$(\mathtt{1} < i) \rightarrow (i < \text{sq}\,(1 + (i \div \mathtt{2})))$

```
; (isqrt i) is the square root of i:  (isqrt i)^2 <= i < [(isqrt i)+1]^2.
```

THEOREM: isqrt-logic-correctness
$(\mathtt{1} < i) \rightarrow ((i < \text{sq}\,(1 + \text{isqrt}\,(i))) \wedge (i \not< \text{sq}\,(\text{isqrt}\,(i))))$

```
; a simple time analysis.
```

THEOREM: quotient-mono-1
$((y \leq z) \wedge (y \not\simeq \mathtt{0}) \wedge (z \not\simeq \mathtt{0})) \rightarrow ((x \div y) \not< (x \div z))$

THEOREM: mean-lessp-1
$(x \not< y) \rightarrow (x \not< ((x + y) \div \mathtt{2}))$

THEOREM: isqrt1-t-la-0
**let** *j1* **be** $(j + (i \div j)) \div 2$
**in**
$(((i \div j) \leq j) \wedge (1 < i) \wedge (0 < j))$
$\rightarrow \quad ((x - (i \div j)) \not< (x - (i \div j1)))$ **endlet**

THEOREM: mean-difference-1-1
$(i \leq j) \rightarrow ((((j + i) \div 2) - i) = ((j - i) \div 2))$

THEOREM: isqrt1-t-la-1
**let** *j1* **be** $(j + (i \div j)) \div 2$
**in**
$(((i \div j) < j) \wedge (1 < i) \wedge (0 < j))$
$\rightarrow \quad (((j - (i \div j)) \div 2) \not< (j1 - (i \div j1)))$ **endlet**

THEOREM: isqrt1-t-la-2
**let** *j1* **be** $(j + (i \div j)) \div 2$
**in**
$(((i \div j) < j) \wedge (1 < i) \wedge (0 < j))$
$\rightarrow \quad (\log(2, (j - (i \div j)) \div 2) \not< \log(2, j1 - (i \div j1)))$ **endlet**

THEOREM: isqrt-t1-ubound
$((1 < i) \wedge (0 < j))$
$\rightarrow \quad ((18 + (10 * \log(2, j - (i \div j)))) \not< \text{isqrt1-t}(i, j))$

THEOREM: isqrt-t->isqrt1-t
$(1 < i) \rightarrow ((4 + \text{isqrt1-t}(i, i \div 2)) \not< \text{isqrt-t}(i))$

THEOREM: log-mono
$\log(b, y) \not< \log(b, y - x)$

THEOREM: isqrt-t-ubound-la
$(1 < i) \rightarrow ((12 + (10 * \log(2, i))) \not< \text{isqrt-t}(i))$

THEOREM: isqrt-t-ubound
$((i < \exp(2, 31)) \wedge (1 < i)) \rightarrow (\text{isqrt-t}(i) \leq 322)$

# Index