

#|

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.

|#

; Case study: Switch Statement

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

#|

The purpose of this trivial C function here is to study the switch construct in C.

```
int foo(int n)
{
    int i;

    switch(n) {
        case 0: i = 0; break;
        case 1: i = 1; break;
        case 2: i = 4; break;
        case 3: i = 9; break;
        case 4: i = 16; break;
```

```

    default: i = n; break;
};
return i;
}

```

Here is the MC68020 assembly code of the above function. The code is generated by gcc with optimization option.

```

0x23b2 <foo>:      linkw a6,#0
0x23b6 <foo+4>:    movel a6@8,d0
0x23ba <foo+8>:    movel #4,d1
0x23bc <foo+10>:   cmpl d1,d0
0x23be <foo+12>:   bhi 0x23e4 <foo+50>
0x23c0 <foo+14>:   movew 0x23c8[d0.l*2],d1
0x23c4 <foo+18>:   jmp 0x23c8[d1.w]
0x23c8 <foo+22>:   orb #14,a2
0x23cc <foo+26>:   orb #22,a2@
0x23d0 <foo+30>:   orb #-128,a2@+
0x23d4 <foo+34>:   bra 0x23e4 <foo+50>
0x23d6 <foo+36>:   movel #1,d0
0x23d8 <foo+38>:   bra 0x23e4 <foo+50>
0x23da <foo+40>:   movel #4,d0
0x23dc <foo+42>:   bra 0x23e4 <foo+50>
0x23de <foo+44>:   movel #9,d0
0x23e0 <foo+46>:   bra 0x23e4 <foo+50>
0x23e2 <foo+48>:   movel #16,d0
0x23e4 <foo+50>:   unlk a6
0x23e6 <foo+52>:   rts

```

The machine code of the above program is:

```

<foo>:      0x4e56  0x0000  0x202e  0x0008  0x7204  0xb081  0x6224  0x323b
<foo+16>:    0x0a06  0x4efb  0x1002  0x000a  0x000e  0x0012  0x0016  0x001a
<foo+32>:    0x4280  0x600e  0x7001  0x600a  0x7004  0x6006  0x7009  0x6002
<foo+48>:    0x7010  0x4e5e  0x4e75

```

```

' (78      86      0      0      32      46      0      8
   114      4      176     129     98      36      50     59
   10      6      78      251     16      2      0      10
   0       14      0       18      0      22      0      26
   66     128     96      14     112      1      96     10
  112      4      96       6     112      9      96      2
  112     16      78      94      78     117)
|#

```

; in the logic, the above program is specified as (foo-code).

DEFINITION:

FOO-CODE

```
= '(78 86 0 0 32 46 0 8 114 4 176 129 98 36 50 59 10 6
    78 251 16 2 0 10 0 14 0 18 0 22 0 26 66 128 96 14
    112 1 96 10 112 4 96 6 112 9 96 2 112 16 78 94 78
    117)
```

DEFINITION:

foo(*n*)

```
= if between-ileq(0, n, 4) then n * n
   else n endif
```

DEFINITION:

foo-t(*n*)

```
= if (n = 0) ∨ (n = 1) ∨ (n = 2) ∨ (n = 3) then 11
   elseif n = 4 then 10
   else 7 endif
```

DEFINITION:

foo-statep(*s*, *n*)

```
= ((mc-status(s) = 'running)
   ∧ evenp(mc-pc(s))
   ∧ rom-addrp(mc-pc(s), mc-mem(s), 54)
   ∧ mcode-addrp(mc-pc(s), mc-mem(s), FOO-CODE)
   ∧ ram-addrp(sub(32, 4, read-sp(s)), mc-mem(s), 12)
   ∧ disjoint(mc-pc(s), 54, sub(32, 4, read-sp(s)), 12)
   ∧ (n = iread-mem(add(32, read-sp(s), 4), mc-mem(s), 4)))
```

DEFINITION:

foo-snp(*s*, *sn*, *n*, *oplen*, *rn*, *x*, *k*)

```
= ((mc-status(sn) = 'running)
   ∧ (mc-pc(sn) = rts-addr(s))
   ∧ (iread-dn(32, 0, sn) = foo(n))
   ∧ (read-rn(32, 14, mc-rfile(sn))
       = read-rn(32, 14, mc-rfile(s)))
   ∧ (read-rn(32, 15, mc-rfile(sn)) = add(32, read-an(32, 7, s), 4))
   ∧ (read-rn(oplen, rn, mc-rfile(sn))
       = read-rn(oplen, rn, mc-rfile(s)))
   ∧ (read-mem(x, mc-mem(sn), k) = read-mem(x, mc-mem(s), k)))
```

THEOREM: foo-s-sn

(foo-statep(*s*, *n*))

\wedge d2-7a2-5p(rn)
 \wedge disjoint(x, k , sub(32, 4, read-sp(s)), 12))
 \rightarrow foo-snp(s , stepn(s , foo-t(n)), n , $oplen$, rn , x , k)

THEOREM: foo-correctness

let sn **be** stepn(s , foo-t(n))
in
 foo-statep(s , n)
 \rightarrow ((mc-status(sn) = '**running**)
 \wedge (mc-pc(sn) = rts-addr(s))
 \wedge (read-rn(32, 14, mc-rfile(sn))
 $=$ read-rn(32, 14, mc-rfile(s)))
 \wedge (read-rn(32, 15, mc-rfile(sn))
 $=$ add(32, read-an(32, 7, s), 4))
 \wedge (d2-7a2-5p(rn)
 \rightarrow (read-rn($oplen$, rn , mc-rfile(sn))
 $=$ read-rn($oplen$, rn , mc-rfile(s))))
 \wedge (disjoint(x, k , sub(32, 4, read-sp(s)), 12)
 \rightarrow (read-mem(x , mc-mem(sn), k)
 $=$ read-mem(x , mc-mem(s), k)))
 \wedge (iread-dn(32, 0, sn) = foo(n))) **endlet**

Index

add, 3, 4

between-ileq, 3

d2-7a2-5p, 4

disjoint, 3, 4

evenp, 3

foo, 3, 4

foo-code, 3

foo-correctness, 4

foo-s-snp, 3

foo-snp, 3, 4

foo-statep, 3, 4

foo-t, 3, 4

iread-dn, 3, 4

iread-mem, 3

mc-mem, 3, 4

mc-pc, 3, 4

mc-rfile, 3, 4

mc-status, 3, 4

mcode-addrp, 3

ram-addrp, 3

read-an, 3, 4

read-mem, 3, 4

read-rn, 3, 4

read-sp, 3, 4

rom-addrp, 3

rts-addr, 3, 4

stepn, 4

sub, 3, 4