```
#|

 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.

|#
```

EVENT: Start with the library "gcd" using the compiled version.

```
#|

The following C program computes the greatest common divisor of
three nonnegative integers a, b and c. We investigate the machine
code of this program generated by a widely used C compiler gcc,
and verify the correctness of the code.  The aim here is to see
how to handle subroutine calls.

gcd3(a, b, c)
long int a, b, c;
{
  gcd(gcd(a, b), c);
}

Here is the MC68020 assembly code of the above GCD program.  The code is
generated by gcc.
```

```
0x2324 <gcd3>:          linkw a6,#0
0x2328 <gcd3+4>:        movel a2,sp@-
0x232a <gcd3+6>:        movel a6@(16),sp@-
0x232e <gcd3+10>:       movel a6@(12),sp@-
0x2332 <gcd3+14>:       movel a6@(8),sp@-
0x2336 <gcd3+18>:       lea @#0x2350 <gcd>,a2
0x233c <gcd3+24>:       jsr a2@
0x233e <gcd3+26>:       addqw #8,sp
0x2340 <gcd3+28>:       movel d0,sp@-
0x2342 <gcd3+30>:       jsr a2@
0x2344 <gcd3+32>:       moveal a6@(-4),a2
0x2348 <gcd3+36>:       unlk a6
0x234a <gcd3+38>:       rts

The machine code of the above program is:

<gcd3>:       0x4e56  0x0000  0x2f0a  0x2f2e  0x0010  0x2f2e  0x000c  0x2f2e
<gcd3+16>:    0x0008  0x45f9  0x0000  0x2350  0x4e92  0x504f  0x2f00  0x4e92
<gcd3+32>:    0x246e  0xfffc  0x4e5e  0x4e75

'(78      86      0       0       47      10      47      46
  0       16      47      46      0       12      47      46
  0       8       69      249     0       0       35      80
  78      146     80      79      47      0       78      146
  36      110     255     252     78      94      78      117)
|#

; now we start to verify this GCD3 program,  defined by (gcd3-code).
```

DEFINITION:
GCD3-CODE
= '(78 86 0 0 47 10 47 46 0 16 47 46 0 12 47 46 0 8 69
  249 -1 -1 -1 -1 78 146 80 79 47 0 78 146 36 110 255
  252 78 94 78 117)

CONSERVATIVE AXIOM: gcd3-load
gcd3-loadp (s)
= (evenp (GCD3-ADDR)
  ∧ (GCD3-ADDR $\in \mathbf{N}$)
  ∧ nat-rangep (GCD3-ADDR, 32)
  ∧ rom-addrp (GCD3-ADDR, mc-mem (s), 40)
  ∧ mcode-addrp (GCD3-ADDR, mc-mem (s), GCD3-CODE)
  ∧ gcd-loadp (s)
  ∧ (pc-read-mem (add (32, GCD3-ADDR, 20), mc-mem (s), 4) = GCD-ADDR))

Simultaneously, we introduce the new function symbols *gcd3-loadp* and *gcd3-addr*.

THEOREM: stepn-gcd3-loadp
gcd3-loadp (stepn (s, n)) = gcd3-loadp (s)

DEFINITION: gcd3 (a, b, c) = gcd (gcd (a, b), c)

DEFINITION: gcd3-t0 (a, b, c) = 7

DEFINITION: gcd3-t1 (a, b, c) = gcd-t (a, b)

DEFINITION: gcd3-t2 (a, b, c) = 3

DEFINITION: gcd3-t3 (a, b, c) = gcd-t (gcd (a, b), c)

DEFINITION: gcd3-t4 (a, b, c) = 3

DEFINITION:
gcd3-t (a, b, c)
= splus (gcd3-t0 (a, b, c),
splus (gcd3-t1 (a, b, c),
splus (gcd3-t2 (a, b, c), splus (gcd3-t3 (a, b, c), gcd3-t4 (a, b, c)))))

```
; the initial state.
```

DEFINITION:
gcd3-statep (s, a, b, c)
= ((mc-status (s) = 'running)
∧ gcd3-loadp (s)
∧ (mc-pc (s) = GCD3-ADDR)
∧ ram-addrp (sub (32, 36, read-sp (s)), mc-mem (s), 52)
∧ (a = iread-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
∧ (b = iread-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
∧ (c = iread-mem (add (32, read-sp (s), 12), mc-mem (s), 4))
∧ ($0 < a$)
∧ ($0 < b$)
∧ ($0 < c$))

```
; the state after the execution of the first JSR instruction, but before
; the execution of the subroutine GCD.
```

DEFINITION:
gcd3-s0p (s, a, b, c)
= ((mc-status (s) = 'running)
∧ gcd3-loadp (s)

$\wedge$ (mc-pc $(s) =$ GCD-ADDR)
$\wedge$ (read-an (32, 2, s) = GCD-ADDR)
$\wedge$ (rts-addr $(s) =$ add (32, GCD3-ADDR, 26))
$\wedge$ ram-addrp (sub (32, 12, read-sp (s)), mc-mem (s), 52)
$\wedge$ equal* (read-an (32, 6, s), add (32, read-sp (s), 20))
$\wedge$ (a = iread-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
$\wedge$ (b = iread-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
$\wedge$ (c = iread-mem (add (32, read-sp (s), 12), mc-mem (s), 4))
$\wedge$ $(0 < a)$
$\wedge$ $(0 < b)$
$\wedge$ $(0 < c)$)

```
; the state right after return from the first call to subroutine GCD.
```

DEFINITION:
gcd3-s1p $(s,\ a,\ b,\ c)$
= ((mc-status (s) = 'running)
$\wedge$ gcd3-loadp (s)
$\wedge$ (read-an (32, 2, s) = GCD-ADDR)
$\wedge$ (mc-pc $(s) =$ add (32, GCD3-ADDR, 26))
$\wedge$ ram-addrp (sub (32, 16, read-sp (s)), mc-mem (s), 52)
$\wedge$ equal* (read-an (32, 6, s), add (32, read-sp (s), 16))
$\wedge$ (iread-dn (32, 0, s) = gcd $(a,\ b)$)
$\wedge$ (c = iread-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
$\wedge$ $(0 < a)$
$\wedge$ $(0 < b)$
$\wedge$ $(0 < c)$)

```
; the state after the execution of the second JSR, but before the
; execution of the subroutine GCD.
```

DEFINITION:
gcd3-s2p $(s,\ a,\ b,\ c)$
= ((mc-status (s) = 'running)
$\wedge$ gcd3-loadp (s)
$\wedge$ (mc-pc $(s) =$ GCD-ADDR)
$\wedge$ (rts-addr $(s) =$ add (32, GCD3-ADDR, 32))
$\wedge$ ram-addrp (sub (32, 16, read-sp (s)), mc-mem (s), 52)
$\wedge$ equal* (read-an (32, 6, s), add (32, read-sp (s), 16))
$\wedge$ (gcd $(a,\ b)$ = iread-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
$\wedge$ (c = iread-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
$\wedge$ $(0 < a)$
$\wedge$ $(0 < b)$
$\wedge$ $(0 < c)$)

```
; the state returned from the second call to the subroutine GCD.
```

DEFINITION:
gcd3-s3p (s, a, b, c)
$=$ ((mc-status (s) = 'running)
∧ gcd3-loadp (s)
∧ (mc-pc (s) = add (32, GCD3-ADDR, 32))
∧ ram-addrp (sub (32, 12, read-sp (s)), mc-mem (s), 44)
∧ equal* (read-an (32, 6, s), add (32, read-sp (s), 12))
∧ (gcd (gcd (a, b), c) = iread-dn (32, 0, s))
∧ $(0 < a)$
∧ $(0 < b)$
∧ $(0 < c)$)

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

THEOREM: gcd3-s-s0
let *s0* **be** stepn (s, gcd3-t0 (a, b, c))
in
gcd3-statep (s, a, b, c)
$\rightarrow$ (gcd3-s0p (*s0*, a, b, c)
∧ (linked-rts-addr (*s0*) = rts-addr (s))
∧ (linked-a6 (*s0*) = read-an (32, 6, s))
∧ (read-rn (32, 14, mc-rfile (*s0*))
= sub (32, 4, read-sp (s)))
∧ (rn-saved (*s0*) = read-an (32, 2, s))) **endlet**

THEOREM: gcd3-s-s0-rfile
(gcd3-statep (s, a, b, c) ∧ d2-7a3-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, gcd3-t0 (a, b, c))))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: gcd3-s-s0-mem
(gcd3-statep (s, a, b, c) ∧ disjoint (x, k, sub (32, 36, read-sp (s)), 52))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, gcd3-t0 (a, b, c))), k)
= read-mem (x, mc-mem (s), k))

```
; from s0 to s1.
```

THEOREM: gcd3-s0-s1
let *s1* **be** stepn (s, gcd3-t1 (a, b, c))
in
gcd3-s0p (s, a, b, c)
$\rightarrow$ (gcd3-s1p (*s1*, a, b, c)
∧ (linked-rts-addr (*s1*) = linked-rts-addr (s))

∧ (read-rn (32, 14, mc-rfile (*s1*))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-a6 (*s1*) = linked-a6 (*s*))
∧ (rn-saved (*s1*) = rn-saved (*s*))) **endlet**

THEOREM: gcd3-s0-s1-rfile
(gcd3-s0p (*s*, *a*, *b*, *c*) ∧ d2-7a2-5p (*rn*) ∧ (*oplen* ≤ 32))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, gcd3-t1 (*a*, *b*, *c*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: gcd3-s0-s1-mem
(gcd3-s0p (*s*, *a*, *b*, *c*) ∧ disjoint (*x*, *k*, sub (32, 32, read-an (32, 6, *s*)), 52))
→ (read-mem (*x*, mc-mem (stepn (*s*, gcd3-t1 (*a*, *b*, *c*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*))

```
; from s1 to s2.
```

THEOREM: gcd3-s1-s2
let *s2* **be** stepn (*s*, gcd3-t2 (*a*, *b*, *c*))
in
gcd3-s1p (*s*, *a*, *b*, *c*)
→ (gcd3-s2p (*s2*, *a*, *b*, *c*)
∧ (linked-rts-addr (*s2*) = linked-rts-addr (*s*))
∧ (read-rn (32, 14, mc-rfile (*s2*))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-a6 (*s2*) = linked-a6 (*s*))
∧ (rn-saved (*s2*) = rn-saved (*s*))) **endlet**

THEOREM: gcd3-s1-s2-rfile
(gcd3-s1p (*s*, *a*, *b*, *c*) ∧ d2-7a3-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, gcd3-t2 (*a*, *b*, *c*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: gcd3-s1-s2-mem
(gcd3-s1p (*s*, *a*, *b*, *c*) ∧ disjoint (*x*, *k*, sub (32, 32, read-an (32, 6, *s*)), 52))
→ (read-mem (*x*, mc-mem (stepn (*s*, gcd3-t2 (*a*, *b*, *c*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*))

```
; from s2 to s3.
```

THEOREM: gcd-nonzero
$((a \not\simeq 0) \land (b \not\simeq 0)) \rightarrow (\mathrm{gcd}(a, b) \neq 0)$

THEOREM: gcd3-s2-s3
let *s3* **be** stepn (*s*, gcd3-t3 (*a*, *b*, *c*))
in

gcd3-s2p (*s*, *a*, *b*, *c*)
→ (gcd3-s3p (*s3*, *a*, *b*, *c*)
∧ (linked-rts-addr (*s3*) = linked-rts-addr (*s*))
∧ (read-rn (`32`, `14`, mc-rfile (*s3*))
= read-rn (`32`, `14`, mc-rfile (*s*)))
∧ (linked-a6 (*s3*) = linked-a6 (*s*))
∧ (rn-saved (*s3*) = rn-saved (*s*))) **endlet**

THEOREM: gcd3-s2-s3-rfile
(gcd3-s2p (*s*, *a*, *b*, *c*) ∧ d2-7a2-5p (*rn*) ∧ (*oplen* ≤ `32`))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, gcd3-t3 (*a*, *b*, *c*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: gcd3-s2-s3-mem
(gcd3-s2p (*s*, *a*, *b*, *c*) ∧ disjoint (*x*, *k*, sub (`32`, `32`, read-an (`32`, `6`, *s*)), `52`))
→ (read-mem (*x*, mc-mem (stepn (*s*, gcd3-t3 (*a*, *b*, *c*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*))

```
; from s3 to exit.
```

THEOREM: gcd3-s3-sn
let *sn* **be** stepn (*s*, gcd3-t4 (*a*, *b*, *c*))
in
gcd3-s3p (*s*, *a*, *b*, *c*)
→ ((mc-status (*sn*) = `'running`)
∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
∧ (iread-dn (`32`, `0`, *sn*) = gcd (gcd (*a*, *b*), *c*))
∧ (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: gcd3-s3-sn-rfile
(gcd3-s3p (*s*, *a*, *b*, *c*) ∧ (*oplen* ≤ `32`) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, gcd3-t4 (*a*, *b*, *c*))))
= **if** *rn* = `10` **then** head (rn-saved (*s*), *oplen*)
else read-rn (*oplen*, *rn*, mc-rfile (*s*)) **endif**)

THEOREM: gcd3-s3-sn-mem
(gcd3-s3p (*s*, *a*, *b*, *c*) ∧ disjoint (*x*, *k*, sub (`32`, `32`, read-an (`32`, `6`, *s*)), `52`))
→ (read-mem (*x*, mc-mem (stepn (*s*, gcd3-t4 (*a*, *b*, *c*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*))

EVENT: Disable gcd3-t0.

EVENT: Disable gcd3-t1.

EVENT: Disable gcd3-t2.

EVENT: Disable gcd3-t3.

EVENT: Disable gcd3-t4.

```
; the correctness of the program GCD3.
```

THEOREM: gcd3-correctness

$$
\begin{array}{l}
\textbf{let } sn \textbf{ be } \text{stepn}\,(s,\ \text{gcd3-t}\,(a,\ b,\ c)) \\
\textbf{in} \\
\text{gcd3-statep}\,(s,\ a,\ b,\ c) \\
\rightarrow \quad ((\text{mc-status}\,(sn) = \texttt{'running}) \\
\qquad \land \quad (\text{mc-pc}\,(sn) = \text{rts-addr}\,(s)) \\
\qquad \land \quad (\text{read-rn}\,(\mathsf{32},\ \mathsf{14},\ \text{mc-rfile}\,(sn)) \\
\qquad\qquad = \quad \text{read-rn}\,(\mathsf{32},\ \mathsf{14},\ \text{mc-rfile}\,(s))) \\
\qquad \land \quad (\text{read-rn}\,(\mathsf{32},\ \mathsf{15},\ \text{mc-rfile}\,(sn)) \\
\qquad\qquad = \quad \text{add}\,(\mathsf{32},\ \text{read-rn}\,(\mathsf{32},\ \mathsf{15},\ \text{mc-rfile}\,(s)),\ \mathsf{4})) \\
\qquad \land \quad (((oplen \le \mathsf{32}) \land \text{d2-7a2-5p}\,(rn)) \\
\qquad\qquad \rightarrow \quad (\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(sn)) \\
\qquad\qquad\qquad = \quad \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))) \\
\qquad \land \quad (\text{disjoint}\,(x,\ k,\ \text{sub}\,(\mathsf{32},\ \mathsf{36},\ \text{read-sp}\,(s)),\ \mathsf{52}) \\
\qquad\qquad \rightarrow \quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(sn),\ k) \\
\qquad\qquad\qquad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k))) \\
\qquad \land \quad (\text{iread-dn}\,(\mathsf{32},\ \mathsf{0},\ sn) = \text{gcd}\,(\text{gcd}\,(a,\ b),\ c))) \ \textbf{endlet}
\end{array}
$$

EVENT: Disable gcd3-t.

```
; in the logic, the function gcd3 does computes the greatest common divisor
; of its three arguments.
```

THEOREM: remainder-trans

$$(((a \textbf{ mod } b) = \mathsf{0}) \land ((b \textbf{ mod } c) = \mathsf{0})) \rightarrow ((a \textbf{ mod } c) = \mathsf{0})$$

THEOREM: gcd3-is-cd

$$
\begin{array}{l}
((a \textbf{ mod } \text{gcd3}\,(a,\ b,\ c)) = \mathsf{0}) \\
\land \quad ((b \textbf{ mod } \text{gcd3}\,(a,\ b,\ c)) = \mathsf{0}) \\
\land \quad ((c \textbf{ mod } \text{gcd3}\,(a,\ b,\ c)) = \mathsf{0})
\end{array}
$$

THEOREM: cd-divides-gcd

$$(((a \textbf{ mod } x) = \mathsf{0}) \land ((b \textbf{ mod } x) = \mathsf{0})) \rightarrow ((\text{gcd}\,(a,\ b) \textbf{ mod } x) = \mathsf{0})$$

THEOREM: gcd3-the-greatest

$$
\begin{array}{ll}
((a \not\simeq \mathsf{0}) & \\
\;\wedge & (b \not\simeq \mathsf{0}) \\
\;\wedge & (c \not\simeq \mathsf{0}) \\
\;\wedge & ((a \textbf{ mod } x) = \mathsf{0}) \\
\;\wedge & ((b \textbf{ mod } x) = \mathsf{0}) \\
\;\wedge & ((c \textbf{ mod } x) = \mathsf{0})) \\
\rightarrow & (\text{gcd3}\,(a,\ b,\ c) \not< x)
\end{array}
$$

EVENT: Disable remainder-trans.

Index

add, 2–5, 7, 8

cd-divides-gcd, 8

d2-7a2-5p, 6–8
d2-7a3-5p, 5, 6
disjoint, 5–8

equal*, 4, 5
evenp, 2

gcd, 3–8
gcd-addr, 2, 4
gcd-loadp, 2
gcd-nonzero, 6
gcd-t, 3
gcd3, 3, 8, 9
gcd3-addr, 2–5
gcd3-code, 2
gcd3-correctness, 8
gcd3-is-cd, 8
gcd3-load, 2
gcd3-loadp, 2–5
gcd3-s-s0, 5
gcd3-s-s0-mem, 5
gcd3-s-s0-rfile, 5
gcd3-s0-s1, 5
gcd3-s0-s1-mem, 6
gcd3-s0-s1-rfile, 6
gcd3-s0p, 3, 5, 6
gcd3-s1-s2, 6
gcd3-s1-s2-mem, 6
gcd3-s1-s2-rfile, 6
gcd3-s1p, 4–6
gcd3-s2-s3, 6
gcd3-s2-s3-mem, 7
gcd3-s2-s3-rfile, 7
gcd3-s2p, 4, 6, 7
gcd3-s3-sn, 7
gcd3-s3-sn-mem, 7
gcd3-s3-sn-rfile, 7
gcd3-s3p, 5, 7
gcd3-statep, 3, 5, 8
gcd3-t, 3, 8
gcd3-t0, 3, 5
gcd3-t1, 3, 5, 6
gcd3-t2, 3, 6
gcd3-t3, 3, 6, 7
gcd3-t4, 3, 7
gcd3-the-greatest, 9

head, 7

iread-dn, 4, 5, 7, 8
iread-mem, 3, 4

linked-a6, 5–7
linked-rts-addr, 5–7

mc-mem, 2–8
mc-pc, 3–5, 7, 8
mc-rfile, 5–8
mc-status, 3–5, 7, 8
mcode-addrp, 2

nat-rangep, 2

pc-read-mem, 2

ram-addrp, 3–5
read-an, 4–7
read-mem, 5–8
read-rn, 5–8
read-sp, 3–5, 8
remainder-trans, 8
rn-saved, 5–7
rom-addrp, 2
rts-addr, 4, 5, 8

splus, 3
stepn, 3, 5–8
stepn-gcd3-loadp, 3
sub, 3–8