```
#|

 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 "mc20-2" using the compiled version.

```
;          Proof of the Correctness of the MEMCMP Function
#|
This is part of our effort to verify the Berkeley string library.  The
Berkeley string library is widely used as part of the Berkeley Unix OS.

This is the source code of memcmp function in the Berkeley string library.

int
memcmp(s1, s2, n)
const void *s1, *s2;
size_t n;
{
if (n != 0) {
register const unsigned char *p1 = s1, *p2 = s2;

do {
```

```
if (*p1++ != *p2++)
return (*--p1 - *--p2);
} while (--n != 0);
}
return (0);
}
```

The MC68020 assembly code of the C function memcmp on SUN-3 is given as follows. This binary is generated by "gcc -O".

```
0x2810 <memcmp>:        linkw fp,#0
0x2814 <memcmp+4>:      movel d2,sp@-
0x2816 <memcmp+6>:      movel fp@(16),d0
0x281a <memcmp+10>:     beq 0x2842 <memcmp+50>
0x281c <memcmp+12>:     moveal fp@(8),a1
0x2820 <memcmp+16>:     moveal fp@(12),a0
0x2824 <memcmp+20>:     andil #255,d1
0x282a <memcmp+26>:     andil #255,d2
0x2830 <memcmp+32>:     cmpmb a0@+,a1@+
0x2832 <memcmp+34>:     beq 0x283e <memcmp+46>
0x2834 <memcmp+36>:     moveb a1@-,d1
0x2836 <memcmp+38>:     moveb a0@-,d2
0x2838 <memcmp+40>:     movel d1,d0
0x283a <memcmp+42>:     subl d2,d0
0x283c <memcmp+44>:     bra 0x2844 <memcmp+52>
0x283e <memcmp+46>:     subl #1,d0
0x2840 <memcmp+48>:     bne 0x2830 <memcmp+32>
0x2842 <memcmp+50>:     clrl d0
0x2844 <memcmp+52>:     movel fp@(-4),d2
0x2848 <memcmp+56>:     unlk fp
0x284a <memcmp+58>:     rts
```

The machine code of the above program is:

```
<memcmp>:      0x4e56  0x0000  0x2f02  0x202e  0x0010  0x6726  0x226e  0x0008
<memcmp+16>:   0x206e  0x000c  0x0281  0x0000  0x00ff  0x0282  0x0000  0x00ff
<memcmp+32>:   0xb308  0x670a  0x1221  0x1420  0x2001  0x9082  0x6006  0x5380
<memcmp+48>:   0x66ee  0x4280  0x242e  0xfffc  0x4e5e  0x4e75
```

```
'(78      86      0       0       47      2       32      46
  0       16      103     38      34      110     0       8
  32      110     0       12      2       129     0       0
  0       255     2       130     0       0       0       255
  179     8       103     10      18      33      20      32
```

```
  32      1       144     130     96      6       83      128
  102     238     66      128     36      46      255     252
  78      94      78      117)
|#

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

DEFINITION:
MEMCMP-CODE
= `'(78 86 0 0 47 2 32 46 0 16 103 38 34 110 0 8 32 110 0`
`12 2 129 0 0 0 255 2 130 0 0 0 255 179 8 103 10 18`
`33 20 32 32 1 144 130 96 6 83 128 102 238 66 128 36`
`46 255 252 78 94 78 117)`

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

DEFINITION:
memcmp-t1 (i, n, *lst1*, *lst2*)
= **if** get-nth (i, *lst1*) = get-nth (i, *lst2*)
then if ($n - 1$) = `0` **then** `8`
else splus (`4`, memcmp-t1 ($1 + i$, $n - 1$, *lst1*, *lst2*)) **endif**
else `10` **endif**

DEFINITION:
memcmp-t (n, *lst1*, *lst2*)
= **if** n = `0` **then** `8`
else splus (`8`, memcmp-t1 (`0`, n, *lst1*, *lst2*)) **endif**

```
; an induction hint.
```

DEFINITION:
memcmp-induct (s, i^*, i, n, *lst1*, *lst2*)
= **if** get-nth (i, *lst1*) = get-nth (i, *lst2*)
then if ($n - 1$) = `0` **then t**
else memcmp-induct (stepn (s, `4`),
add (`32`, i^*, `1`),
$1 + i$,
$n - 1$,
lst1,
lst2) **endif**
else t endif

```
; the preconditions of the initial state.
```

DEFINITION:

memcmp-statep (s, *str1*, n, *lst1*, *str2*, *lst2*)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (mc-pc (s), mc-mem (s), 60)
∧ mcode-addrp (mc-pc (s), mc-mem (s), MEMCMP-CODE)
∧ ram-addrp (sub (32, 8, read-sp (s)), mc-mem (s), 24)
∧ ram-addrp (*str1*, mc-mem (s), n)
∧ mem-lst (1, *str1*, mc-mem (s), n, *lst1*)
∧ ram-addrp (*str2*, mc-mem (s), n)
∧ mem-lst (1, *str2*, mc-mem (s), n, *lst2*)
∧ disjoint (sub (32, 8, read-sp (s)), 24, *str1*, n)
∧ disjoint (sub (32, 8, read-sp (s)), 24, *str2*, n)
∧ (*str1* = read-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
∧ (*str2* = read-mem (add (32, read-sp (s), 8), mc-mem (s), 4))
∧ (n = uread-mem (add (32, read-sp (s), 12), mc-mem (s), 4)))

```
; an intermediate state.
```

DEFINITION:
memcmp-s0p (s, i^*, i, *str1*, n, *lst1*, *str2*, *lst2*, $n\_$)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (sub (32, 32, mc-pc (s)), mc-mem (s), 60)
∧ mcode-addrp (sub (32, 32, mc-pc (s)), mc-mem (s), MEMCMP-CODE)
∧ ram-addrp (sub (32, 4, read-an (32, 6, s)), mc-mem (s), 24)
∧ ram-addrp (*str1*, mc-mem (s), $n\_$)
∧ mem-lst (1, *str1*, mc-mem (s), $n\_$, *lst1*)
∧ ram-addrp (*str2*, mc-mem (s), $n\_$)
∧ mem-lst (1, *str2*, mc-mem (s), $n\_$, *lst2*)
∧ disjoint (sub (32, 4, read-an (32, 6, s)), 24, *str1*, $n\_$)
∧ disjoint (sub (32, 4, read-an (32, 6, s)), 24, *str2*, $n\_$)
∧ equal* (read-an (32, 1, s), add (32, *str1*, i^*))
∧ equal* (read-an (32, 0, s), add (32, *str2*, i^*))
∧ nat-rangep (read-rn (32, 1, mc-rfile (s)), 8)
∧ nat-rangep (read-rn (32, 2, mc-rfile (s)), 8)
∧ (n = nat-to-uint (read-dn (32, 0, s)))
∧ (i = nat-to-uint (i^*))
∧ ((i + n) ≤ $n\_$)
∧ ($n \neq 0$)
∧ ($i^* \in \mathbf{N}$)
∧ nat-rangep (i^*, 32)
∧ uint-rangep ($n\_$, 32))

```
; from the initial state s to exit: s --> sn, when n = 0.
```

THEOREM: memcmp-s-sn
(memcmp-statep (*s*, *str1*, *n*, *lst1*, *str2*, *lst2*) ∧ (*n* = 0))
→ ((mc-status (stepn (*s*, 8)) = 'running)
∧ (mc-pc (stepn (*s*, 8)) = rts-addr (*s*))
∧ (iread-dn (32, 0, stepn (*s*, 8)) = 0)
∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 8)))
= add (32, read-an (32, 7, *s*), 4))
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 8))) = read-an (32, 6, *s*)))

THEOREM: memcmp-s-sn-rfile
(memcmp-statep (*s*, *str1*, *n*, *lst1*, *str2*, *lst2*)
∧ (*n* = 0)
∧ (*oplen* ≤ 32)
∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 8)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: memcmp-s-sn-mem
(memcmp-statep (*s*, *str1*, *n*, *lst1*, *str2*, *lst2*)
∧ (*n* = 0)
∧ disjoint (*x*, *k*, sub (32, 8, read-sp (*s*)), 24))
→ (read-mem (*x*, mc-mem (stepn (*s*, 8)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; from the initial state s to s0: s --> s0.
```

THEOREM: memcmp-s-s0
(memcmp-statep (*s*, *str1*, *n*, *lst1*, *str2*, *lst2*) ∧ (*n* ≠ 0))
→ memcmp-s0p (stepn (*s*, 8), 0, 0, *str1*, *n*, *lst1*, *str2*, *lst2*, *n*)

THEOREM: memcmp-s-s0-else
(memcmp-statep (*s*, *str1*, *n*, *lst1*, *str2*, *lst2*) ∧ (*n* ≠ 0))
→ ((linked-rts-addr (stepn (*s*, 8)) = rts-addr (*s*))
∧ (linked-a6 (stepn (*s*, 8)) = read-an (32, 6, *s*))
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 8)))
= sub (32, 4, read-sp (*s*)))
∧ (rn-saved (stepn (*s*, 8)) = read-dn (32, 2, *s*)))

THEOREM: memcmp-s-s0-rfile
(memcmp-statep (*s*, *str1*, *n*, *lst1*, *str2*, *lst2*) ∧ (*n* ≠ 0) ∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 8)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: memcmp-s-s0-mem
(memcmp-statep (*s*, *str1*, *n*, *lst1*, *str2*, *lst2*)
∧ (*n* ≠ 0)
∧ disjoint (*x*, *k*, sub (32, 8, read-sp (*s*)), 24))
→ (read-mem (*x*, mc-mem (stepn (*s*, 8)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; from s0 to exit: s0 --> sn.
; base case 1:  s0 --> sn, when lst1[i] =\= lst2[i].
```

THEOREM: memcmp-s0-sn-base1
(memcmp-s0p (*s*, *i**, *i*, *str1*, *n*, *lst1*, *str2*, *lst2*, *n_*)
 ∧ (get-nth (*i*, *lst1*) ≠ get-nth (*i*, *lst2*)))
→ ((mc-status (stepn (*s*, 10)) = 'running)
 ∧ (mc-pc (stepn (*s*, 10)) = linked-rts-addr (*s*))
 ∧ (iread-dn (32, 0, stepn (*s*, 10))
 = idifference (get-nth (*i*, *lst1*), get-nth (*i*, *lst2*)))
 ∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 10))) = linked-a6 (*s*))
 ∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 10)))
 = add (32, read-an (32, 6, *s*), 8))
 ∧ (read-mem (*x*, mc-mem (stepn (*s*, 10)), *k*)
 = read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: memcmp-s0-sn-rfile-base1
(memcmp-s0p (*s*, *i**, *i*, *str1*, *n*, *lst1*, *str2*, *lst2*, *n_*)
 ∧ (get-nth (*i*, *lst1*) ≠ get-nth (*i*, *lst2*))
 ∧ (*oplen* ≤ 32)
 ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 10)))
 = **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
 else head (rn-saved (*s*), *oplen*) **endif**)

```
; base case 2: s0 --> sn, when lst[i] = lst2[i], lst[i] =\= 0, and n-1 = 0.
```

THEOREM: memcmp-s0-sn-base2
(memcmp-s0p (*s*, *i**, *i*, *str1*, *n*, *lst1*, *str2*, *lst2*, *n_*)
 ∧ (get-nth (*i*, *lst1*) = get-nth (*i*, *lst2*))
 ∧ ((*n* − 1) = 0))
→ ((mc-status (stepn (*s*, 8)) = 'running)
 ∧ (mc-pc (stepn (*s*, 8)) = linked-rts-addr (*s*))
 ∧ (iread-dn (32, 0, stepn (*s*, 8)) = 0)
 ∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 8))) = linked-a6 (*s*))
 ∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 8)))
 = add (32, read-an (32, 6, *s*), 8))
 ∧ (read-mem (*x*, mc-mem (stepn (*s*, 8)), *k*)
 = read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: memcmp-s0-sn-rfile-base2
(memcmp-s0p (*s*, *i**, *i*, *str1*, *n*, *lst1*, *str2*, *lst2*, *n_*)
 ∧ (get-nth (*i*, *lst1*) = get-nth (*i*, *lst2*))
 ∧ ((*n* − 1) = 0)
 ∧ (*oplen* ≤ 32)

$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 8)))
= **if** d3-7a2-5p (rn) **then** read-rn ($oplen$, rn, mc-rfile (s))
else head (rn-saved (s), $oplen$) **endif**)

```
; induction case: s0 --> s0, lst[i] = lst2[i], lst[i] =\= 0 and n-1 =\= 0.
```

THEOREM: memcmp-s0-s0
(memcmp-s0p (s, i^*, i, *str1*, n, *lst1*, *str2*, *lst2*, $n_{\_}$)
$\wedge$ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
$\wedge$ (($n - 1$) $\neq$ 0))
$\rightarrow$ (memcmp-s0p (stepn (s, 4),
add (32, i^*, 1),
$1 + i$,
str1,
$n - 1$,
lst1,
str2,
lst2,
$n_{\_}$)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 4)))
= read-rn (32, 14, mc-rfile (s)))
$\wedge$ (linked-a6 (stepn (s, 4)) = linked-a6 (s))
$\wedge$ (linked-rts-addr (stepn (s, 4)) = linked-rts-addr (s))
$\wedge$ (rn-saved (stepn (s, 4)) = rn-saved (s))
$\wedge$ (read-mem (x, mc-mem (stepn (s, 4)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: memcmp-s0-s0-rfile
(memcmp-s0p (s, i^*, i, *str1*, n, *lst1*, *str2*, *lst2*, $n_{\_}$)
$\wedge$ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
$\wedge$ (($n - 1$) $\neq$ 0)
$\wedge$ d3-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 4)))
= read-rn ($oplen$, rn, mc-rfile (s)))

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

THEOREM: memcmp-s0-sn
let sn **be** stepn (s, memcmp-t1 (i, n, *lst1*, *lst2*))
in
memcmp-s0p (s, i^*, i, *str1*, n, *lst1*, *str2*, *lst2*, $n_{\_}$)
$\rightarrow$ ((mc-status (sn) = 'running)
$\wedge$ (mc-pc (sn) = linked-rts-addr (s))
$\wedge$ (iread-dn (32, 0, sn) = memcmp1 (i, n, *lst1*, *lst2*))

$\wedge$ (read-rn (32, 14, mc-rfile (*sn*)) = linked-a6 (*s*))
$\wedge$ (read-rn (32, 15, mc-rfile (*sn*))
= add (32, read-an (32, 6, *s*), 8))
$\wedge$ (read-mem (*x*, mc-mem (*sn*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

THEOREM: memcmp-s0-sn-rfile
(memcmp-s0p (*s*, *i**, *i*, *str1*, *n*, *lst1*, *str2*, *lst2*, *n*_)
$\wedge$ d2-7a2-5p (*rn*)
$\wedge$ (*oplen* $\leq$ 32))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, memcmp-t1 (*i*, *n*, *lst1*, *lst2*))))
= **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
else head (rn-saved (*s*), *oplen*) **endif**)

```
; the correctness of memcmp.
```

THEOREM: memcmp-correctness
let *sn* **be** stepn (*s*, memcmp-t (*n*, *lst1*, *lst2*))
in
memcmp-statep (*s*, *str1*, *n*, *lst1*, *str2*, *lst2*)
$\rightarrow$ ((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*) $\wedge$ (*oplen* $\leq$ 32))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))))
$\wedge$ (disjoint (*x*, *k*, sub (32, 8, read-sp (*s*)), 24)
$\rightarrow$ (read-mem (*x*, mc-mem (*sn*), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))
$\wedge$ (iread-dn (32, 0, *sn*) = memcmp (*n*, *lst1*, *lst2*))) **endlet**

EVENT: Disable memcmp-t.

```
; some properties of memcmp.
; see file cstring.events.
```

Index