```
#|

 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 STRCMP 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 strcmp function in the Berkeley string library.

/* compare (unsigned) char s1[] to s2[] */
int
strcmp(s1, s2)
register const char *s1, *s2;
{
while (*s1 == *s2++)
if (*s1++ == 0)
return (0);
return (*(unsigned char *)s1 - *(unsigned char *)--s2);
```

```
}

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

0x2528 <strcmp>:        linkw fp,#0
0x252c <strcmp+4>:      movel d2,sp@-
0x252e <strcmp+6>:      moveal fp@(8),a0
0x2532 <strcmp+10>:     moveal fp@(12),a1
0x2536 <strcmp+14>:     bra 0x2540 <strcmp+24>
0x2538 <strcmp+16>:     tstb a0@+
0x253a <strcmp+18>:     bne 0x2540 <strcmp+24>
0x253c <strcmp+20>:     clrl d0
0x253e <strcmp+22>:     bra 0x2550 <strcmp+40>
0x2540 <strcmp+24>:     moveb a0@,d2
0x2542 <strcmp+26>:     cmpb a1@+,d2
0x2544 <strcmp+28>:     beq 0x2538 <strcmp+16>
0x2546 <strcmp+30>:     clrl d0
0x2548 <strcmp+32>:     moveb a0@,d0
0x254a <strcmp+34>:     clrl d1
0x254c <strcmp+36>:     moveb a1@-,d1
0x254e <strcmp+38>:     subl d1,d0
0x2550 <strcmp+40>:     movel fp@(-4),d2
0x2554 <strcmp+44>:     unlk fp
0x2556 <strcmp+46>:     rts

The machine code of the above program is:

<strcmp>:      0x4e56  0x0000  0x2f02  0x206e  0x0008  0x226e  0x000c  0x6008
<strcmp+16>:   0x4a18  0x6604  0x4280  0x6010  0x1410  0xb419  0x67f2  0x4280
<strcmp+32>:   0x1010  0x4281  0x1221  0x9081  0x242e  0xfffc  0x4e5e  0x4e75

'(78      86      0       0       47      2       32      110
  0       8       34      110     0       12      96      8
  74      24      102     4       66      128     96      16
  20      16      180     25      103     242     66      128
  16      16      66      129     18      33      144     129
  36      46      255     252     78      94      78      117)
|#

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

DEFINITION:
STRCMP-CODE

```
=  '(78 86 0 0 47 2 32 110 0 8 34 110 0 12 96 8 74 24 102
     4 66 128 96 16 20 16 180 25 103 242 66 128 16 16 66
     129 18 33 144 129 36 46 255 252 78 94 78 117)
```

CONSERVATIVE AXIOM: strcmp-load
strcmp-loadp (s)
= (evenp (STRCMP-ADDR)
∧ (STRCMP-ADDR ∈ **N**)
∧ nat-rangep (STRCMP-ADDR, 32)
∧ rom-addrp (STRCMP-ADDR, mc-mem (s), 48)
∧ mcode-addrp (STRCMP-ADDR, mc-mem (s), STRCMP-CODE))

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

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

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

DEFINITION:
strcmp1-t (i, *n1*, *lst1*, *lst2*)
= **if** i < *n1*
then if get-nth (i, *lst1*) = get-nth (i, *lst2*)
then if get-nth (i, *lst1*) = NULL **then** 10
else splus (5, strcmp1-t (1 + i, *n1*, *lst1*, *lst2*)) **endif**
else 11 **endif**
else 0 **endif**

DEFINITION:
strcmp-t (*n1*, *lst1*, *lst2*) = splus (5, strcmp1-t (0, *n1*, *lst1*, *lst2*))

```
; an induction hint.
```

DEFINITION:
strcmp-induct (s, i^*, i, *n1*, *lst1*, *lst2*)
= **if** i < *n1*
then if get-nth (i, *lst1*) = get-nth (i, *lst2*)
then if get-nth (i, *lst1*) = NULL **then t**
else strcmp-induct (stepn (s, 5),
add (32, i^*, 1),
1 + i,
n1,
lst1,
lst2) **endif**
else t endif
else t endif

; the preconditions on the initial state.

DEFINITION:
strcmp-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (s) = 'running)
∧ strcmp-loadp (s)
∧ (mc-pc (s) = STRCMP-ADDR)
∧ ram-addrp (sub (32, 8, read-sp (s)), mc-mem (s), 20)
∧ ram-addrp (*str1*, mc-mem (s), *n1*)
∧ mem-lst (1, *str1*, mc-mem (s), *n1*, *lst1*)
∧ ram-addrp (*str2*, mc-mem (s), *n2*)
∧ mem-lst (1, *str2*, mc-mem (s), *n2*, *lst2*)
∧ disjoint (sub (32, 8, read-sp (s)), 20, *str1*, *n1*)
∧ disjoint (sub (32, 8, read-sp (s)), 20, *str2*, *n2*)
∧ (*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))
∧ stringp (0, *n1*, *lst1*)
∧ (*n1* ≤ *n2*)
∧ (*n1* ∈ **N**)
∧ (*n2* ∈ **N**)
∧ uint-rangep (*n2*, 32))

; an intermediate state.

DEFINITION:
strcmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (sub (32, 24, mc-pc (s)), mc-mem (s), 48)
∧ mcode-addrp (sub (32, 24, mc-pc (s)), mc-mem (s), STRCMP-CODE)
∧ ram-addrp (sub (32, 4, read-an (32, 6, s)), mc-mem (s), 20)
∧ ram-addrp (*str1*, mc-mem (s), *n1*)
∧ mem-lst (1, *str1*, mc-mem (s), *n1*, *lst1*)
∧ ram-addrp (*str2*, mc-mem (s), *n2*)
∧ mem-lst (1, *str2*, mc-mem (s), *n2*, *lst2*)
∧ disjoint (sub (32, 4, read-an (32, 6, s)), 20, *str1*, *n1*)
∧ disjoint (sub (32, 4, read-an (32, 6, s)), 20, *str2*, *n2*)
∧ equal* (read-an (32, 0, s), add (32, *str1*, i^*))
∧ equal* (read-an (32, 1, s), add (32, *str2*, i^*))
∧ (i = nat-to-uint (i^*))
∧ stringp (i, *n1*, *lst1*)
∧ (i < *n1*)
∧ (*n1* ≤ *n2*)
∧ (i^* ∈ **N**)
∧ nat-rangep (i^*, 32)

$\wedge$ $(n1 \in \mathbf{N})$
$\wedge$ $(n2 \in \mathbf{N})$
$\wedge$ uint-rangep $(n2,\ 32)$

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

THEOREM: strcmp-s-s0
strcmp-statep $(s,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$
$\rightarrow$ strcmp-s0p (stepn $(s,\ 5),\ 0,\ 0,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$

THEOREM: strcmp-s-s0-else
strcmp-statep $(s,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$
$\rightarrow$ ((linked-rts-addr (stepn $(s,\ 5)$) = rts-addr (s))
$\wedge$ (linked-a6 (stepn $(s,\ 5)$) = read-an $(32,\ 6,\ s)$)
$\wedge$ (read-rn $(32,\ 14,$ mc-rfile (stepn $(s,\ 5)$))
= sub $(32,\ 4,$ read-sp $(s)))$
$\wedge$ (rn-saved (stepn $(s,\ 5)$) = read-dn $(32,\ 2,\ s)))$

THEOREM: strcmp-s-s0-rfile
(strcmp-statep $(s,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$ $\wedge$ d3-7a2-5p (rn))
$\rightarrow$ (read-rn $(oplen,\ rn,$ mc-rfile (stepn $(s,\ 5)$))
= read-rn $(oplen,\ rn,$ mc-rfile $(s)))$

THEOREM: strcmp-s-s0-mem
(strcmp-statep $(s,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$
$\wedge$ disjoint $(x,\ k,$ sub $(32,\ 8,$ read-sp $(s)),\ 20))$
$\rightarrow$ (read-mem $(x,$ mc-mem (stepn $(s,\ 5)),\ k)$ = read-mem $(x,$ mc-mem $(s),\ k))$

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

THEOREM: strcmp-s0-sn-base1
(strcmp-s0p $(s,\ i^*,\ i,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$
$\wedge$ (get-nth $(i,\ lst1)$ $\neq$ get-nth $(i,\ lst2)))$
$\rightarrow$ ((mc-status (stepn $(s,\ 11)$) = `'running`)
$\wedge$ (mc-pc (stepn $(s,\ 11)$) = linked-rts-addr (s))
$\wedge$ (iread-dn $(32,\ 0,$ stepn $(s,\ 11))$
= idifference (get-nth $(i,\ lst1)$, get-nth $(i,\ lst2)))$
$\wedge$ (read-rn $(32,\ 14,$ mc-rfile (stepn $(s,\ 11))$) = linked-a6 $(s))$
$\wedge$ (read-rn $(32,\ 15,$ mc-rfile (stepn $(s,\ 11))$)
= add $(32,$ read-an $(32,\ 6,\ s),\ 8))$
$\wedge$ (read-mem $(x,$ mc-mem (stepn $(s,\ 11)),\ k)$
= read-mem $(x,$ mc-mem $(s),\ k)))$

THEOREM: strcmp-s0-sn-rfile-base1

(strcmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (i, *lst1*) ≠ get-nth (i, *lst2*))
 ∧ ($oplen$ ≤ 32)
 ∧ d2-7a2-5p (rn))
→ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 11)))
 = **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.  lst[i] = lst2[i] and lst[i] = 0.
```

THEOREM: strcmp-s0-sn-base2
(strcmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
 ∧ (get-nth (i, *lst1*) = 0))
→ ((mc-status (stepn (s, 10)) = 'running)
 ∧ (mc-pc (stepn (s, 10)) = linked-rts-addr (s))
 ∧ (iread-dn (32, 0, stepn (s, 10)) = 0)
 ∧ (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: strcmp-s0-sn-rfile-base2
(strcmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
 ∧ (get-nth (i, *lst1*) = 0)
 ∧ ($oplen$ ≤ 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**)

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

THEOREM: strcmp-s0-s0
(strcmp-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
 ∧ (get-nth (i, *lst1*) ≠ 0))
→ (strcmp-s0p (stepn (s, 5), add (32, i^*, 1), 1 + i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (read-rn (32, 14, mc-rfile (stepn (s, 5)))
 = read-rn (32, 14, mc-rfile (s)))
 ∧ (linked-a6 (stepn (s, 5)) = linked-a6 (s))
 ∧ (linked-rts-addr (stepn (s, 5)) = linked-rts-addr (s))
 ∧ (rn-saved (stepn (s, 5)) = rn-saved (s))

$\wedge$ (read-mem (x, mc-mem (stepn (s, 5)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: strcmp-s0-s0-rfile
(strcmp-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ (get-nth (i, *lst1*) = get-nth (i, *lst2*))
$\wedge$ (get-nth (i, *lst1*) $\neq$ 0)
$\wedge$ d3-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 5)))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

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

THEOREM: strcmp-s0p-info
strcmp-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) $\rightarrow$ ((i < *n1*) = **t**)

THEOREM: strcmp-s0-sn
let *sn* **be** stepn (s, strcmp1-t (i, *n1*, *lst1*, *lst2*))
in
strcmp-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\rightarrow$ ((mc-status (*sn*) = 'running)
$\wedge$ (mc-pc (*sn*) = linked-rts-addr (s))
$\wedge$ (iread-dn (32, 0, *sn*) = strcmp (i, *n1*, *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: strcmp-s0-sn-rfile
(strcmp-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ (*oplen* $\leq$ 32)
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, strcmp1-t (i, *n1*, *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 strcmp.
```

THEOREM: strcmp-correctness
let *sn* **be** stepn (s, strcmp-t (*n1*, *lst1*, *lst2*))
in
strcmp-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\rightarrow$ ((mc-status (*sn*) = 'running)
$\wedge$ (mc-pc (*sn*) = rts-addr (s))
$\wedge$ (read-rn (32, 14, mc-rfile (*sn*))

$$
\begin{array}{ll}
 & = \quad \text{read-rn}\,(\texttt{32},\ \texttt{14},\ \text{mc-rfile}\,(s))) \\
\wedge & (\text{read-rn}\,(\texttt{32},\ \texttt{15},\ \text{mc-rfile}\,(sn)) \\
 & = \quad \text{add}\,(\texttt{32},\ \text{read-an}\,(\texttt{32},\ \texttt{7},\ s),\ \texttt{4})) \\
\wedge & (((oplen \leq \texttt{32}) \wedge \text{d2-7a2-5p}\,(rn)) \\
 & \rightarrow \quad (\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(sn)) \\
 & \qquad = \quad \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))) \\
\wedge & (\text{disjoint}\,(x,\ k,\ \text{sub}\,(\texttt{32},\ \texttt{8},\ \text{read-sp}\,(s)),\ \texttt{20}) \\
 & \rightarrow \quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(sn),\ k) \\
 & \qquad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k))) \\
\wedge & (\text{iread-dn}\,(\texttt{32},\ \texttt{0},\ sn) = \text{strcmp}\,(\texttt{0},\ n1,\ lst1,\ lst2))) \ \textbf{endlet}
\end{array}
$$

EVENT: Disable strcmp-t.

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

EVENT: Make the library `"strcmp"` and compile it.

Index