```
#|

 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 "strcmp" using the compiled version.

```
;          Proof of the Correctness of the STRCOLL 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 strcoll function in the Berkeley string library.

/*
 * Compare strings according to LC_COLLATE category of current locale.
 */
strcoll(s1, s2)
        const char *s1, *s2;
{
        /* LC_COLLATE is unimplemented, hence always "C" */
        return (strcmp(s1, s2));
}
```

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

0x2388 <strcoll>:       linkw fp,#0
0x238c <strcoll+4>:     movel fp@(12),sp@-
0x2390 <strcoll+8>:     movel fp@(8),sp@-
0x2394 <strcoll+12>:    jsr @#0x2358 <strcmp>
0x239a <strcoll+18>:    unlk fp
0x239c <strcoll+20>:    rts

The machine code of the above program is:

<strcoll>:     0x4e56  0x0000  0x2f2e  0x000c  0x2f2e  0x0008  0x4eb9  0x0000
<strcoll+16>: 0x2358  0x4e5e  0x4e75

'(78     86      0       0       47      46      0       12
  47     46      0       8       78      185     0       0
  35     88      78      94      78      117)
|#

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

DEFINITION:
STRCOLL-CODE
= `'(78 86 0 0 47 46 0 12 47 46 0 8 78 185 -1 -1 -1 -1 78 94 78 117)`

CONSERVATIVE AXIOM: strcoll-load
strcoll-loadp (s)
$=$ (evenp (STRCOLL-ADDR)
$\land$ (STRCOLL-ADDR $\in \mathbf{N}$)
$\land$ nat-rangep (STRCOLL-ADDR, 32)
$\land$ rom-addrp (STRCOLL-ADDR, mc-mem (s), 22)
$\land$ mcode-addrp (STRCOLL-ADDR, mc-mem (s), STRCOLL-CODE)
$\land$ strcmp-loadp (s)
$\land$ (pc-read-mem (add (32, STRCOLL-ADDR, 14), mc-mem (s), 4)
$=$ STRCMP-ADDR))

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

THEOREM: stepn-strcoll-loadp
strcoll-loadp (stepn $(s,\ n)$) $=$ strcoll-loadp (s)

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

DEFINITION:
strcoll-t (*n1*, *lst1*, *lst2*) = splus (4, splus (strcmp-t (*n1*, *lst1*, *lst2*), 2))

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

DEFINITION:
strcoll-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
 ∧ strcoll-loadp (*s*)
 ∧ (mc-pc (*s*) = STRCOLL-ADDR)
 ∧ ram-addrp (sub (32, 24, read-sp (*s*)), mc-mem (*s*), 36)
 ∧ 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, 24, read-sp (*s*)), 36, *str1*, *n1*)
 ∧ disjoint (sub (32, 24, read-sp (*s*)), 36, *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*)
 ∧ ($n1 \leq n2$)
 ∧ ($n1 \in \mathbf{N}$)
 ∧ ($n2 \in \mathbf{N}$)
 ∧ uint-rangep (*n2*, 32))

```
; the intermediate state right before the execution of the subroutine strcmp.
```

DEFINITION:
strcoll-s0p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
 ∧ strcoll-loadp (*s*)
 ∧ (mc-pc (*s*) = STRCMP-ADDR)
 ∧ (rts-addr (*s*) = add (32, STRCOLL-ADDR, 18))
 ∧ ram-addrp (sub (32, 20, read-an (32, 6, *s*)), mc-mem (*s*), 36)
 ∧ 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, 20, read-an (32, 6, *s*)), 36, *str1*, *n1*)
 ∧ disjoint (sub (32, 20, read-an (32, 6, *s*)), 36, *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))
 ∧ equal* (read-sp (*s*), sub (32, 12, read-an (32, 6, *s*))))

$\wedge$ stringp (0, *n1*, *lst1*)
$\wedge$ (*n1* $\leq$ *n2*)
$\wedge$ (*n1* $\in \mathbf{N}$)
$\wedge$ (*n2* $\in \mathbf{N}$)
$\wedge$ uint-rangep (*n2*, 32))

```
; the intermediate state right after the execution of the subroutine strcmp.
```

DEFINITION:
strcoll-s1p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = `'running`)
$\wedge$ strcoll-loadp (*s*)
$\wedge$ (mc-pc (*s*) = add (32, STRCOLL-ADDR, 18))
$\wedge$ ram-addrp (sub (32, 20, read-an (32, 6, *s*)), mc-mem (*s*), 36)
$\wedge$ (iread-dn (32, 0, *s*) = strcmp (0, *n1*, *lst1*, *lst2*)))

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

THEOREM: strcoll-s-s0
strcoll-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\rightarrow$ strcoll-s0p (stepn (*s*, 4), *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)

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

THEOREM: strcoll-s-s0-rfile
(strcoll-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 4)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: strcoll-s-s0-mem
(strcoll-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ disjoint (*x*, *k*, sub (32, 24, read-sp (*s*)), 36))
$\rightarrow$ (read-mem (*x*, mc-mem (stepn (*s*, 4)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; from s0 to s1: s0 --> s1. by strcmp.
```

THEOREM: strcoll-s0p-strcmp-statep
strcoll-s0p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\rightarrow$ strcmp-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)

THEOREM: strcoll-s0-s1
let *s1* **be** stepn (*s*, strcmp-t (*n1*, *lst1*, *lst2*))
in
strcoll-s0p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ strcoll-s1p (*s1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) **endlet**

THEOREM: strcoll-s0-s1-else
let *s1* **be** stepn (*s*, strcmp-t (*n1*, *lst1*, *lst2*))
in
strcoll-s0p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ ((read-rn (32, 14, mc-rfile (*s1*))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-rts-addr (*s1*) = linked-rts-addr (*s*))
∧ (linked-a6 (*s1*) = linked-a6 (*s*))) **endlet**

THEOREM: strcoll-s0-s1-rfile
let *s1* **be** stepn (*s*, strcmp-t (*n1*, *lst1*, *lst2*))
in
(strcoll-s0p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ d2-7a2-5p (*rn*)
∧ (*oplen* ≤ 32))
→ (read-rn (*oplen*, *rn*, mc-rfile (*s1*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))) **endlet**

THEOREM: strcoll-s0-s1-mem
let *s1* **be** stepn (*s*, strcmp-t (*n1*, *lst1*, *lst2*))
in
(strcoll-s0p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ disjoint (*x*, *k*, sub (32, 20, read-an (32, 6, *s*)), 36))
→ (read-mem (*x*, mc-mem (*s1*), *k*) = read-mem (*x*, mc-mem (*s*), *k*)) **endlet**

```
; from s1 to exit: s1 --> sn.
```

THEOREM: strcoll-s1-sn
strcoll-s1p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ ((mc-status (stepn (*s*, 2)) = 'running)
∧ (mc-pc (stepn (*s*, 2)) = linked-rts-addr (*s*))
∧ (iread-dn (32, 0, stepn (*s*, 2)) = strcmp (0, *n1*, *lst1*, *lst2*))
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 2))) = linked-a6 (*s*))
∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 2)))
= add (32, read-an (32, 6, *s*), 8))
∧ (read-mem (*x*, mc-mem (stepn (*s*, 2)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strcoll-s1-sn-rfile

(strcoll-s1p (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 2)))
  = read-rn (*oplen*, *rn*, mc-rfile (s)))

```
; the correctness of strcoll.
```

THEOREM: strcoll-correctness
let *sn* **be** stepn (s, strcoll-t (*n1*, *lst1*, *lst2*))
in
strcoll-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*))
    = 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, 24, read-sp (s)), 36)
    $\rightarrow$ (read-mem (x, mc-mem (*sn*), k)
      = read-mem (x, mc-mem (s), k)))
  $\wedge$ (iread-dn (32, 0, *sn*) = strcoll (*n1*, *lst1*, *lst2*))) **endlet**

EVENT: Disable strcoll-t.

```
; some properties of strcoll.
; the same as strcmp.
```

Index