```
#|

 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 STRLEN 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 strlen function in the Berkeley string library.

/* find the length of str[] */
size_t
strlen(str)
const char *str;
{
register const char *s;

for (s = str; *s; ++s);
return(s - str);
```

```
}

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

0x25b0 <strlen>:        linkw fp,#0
0x25b4 <strlen+4>:      movel fp@(8),d0
0x25b8 <strlen+8>:      moveal d0,a0
0x25ba <strlen+10>:     tstb a0@
0x25bc <strlen+12>:     beq 0x25c4 <strlen+20>
0x25be <strlen+14>:     addqw #1,a0
0x25c0 <strlen+16>:     tstb a0@
0x25c2 <strlen+18>:     bne 0x25be <strlen+14>
0x25c4 <strlen+20>:     subl a0,d0
0x25c6 <strlen+22>:     negl d0
0x25c8 <strlen+24>:     unlk fp
0x25ca <strlen+26>:     rts

The machine code of the above program is:

<strlen>:    0x4e56  0x0000  0x202e  0x0008  0x2040  0x4a10  0x6706  0x5248
<strlen+16>: 0x4a10  0x66fa  0x9088  0x4480  0x4e5e  0x4e75

'(78      86      0       0       32      46      0       8
  32      64      74      16      103     6       82      72
  74      16      102     250     144     136     68      128
  78      94      78      117)
|#

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

DEFINITION:
STRLEN-CODE
= '(78 86 0 0 32 46 0 8 32 64 74 16 103 6 82 72 74 16
102 250 144 136 68 128 78 94 78 117)

CONSERVATIVE AXIOM: strlen-load
strlen-loadp (s)
= (evenp (STRLEN-ADDR)
∧ (STRLEN-ADDR ∈ $\mathbf{N}$)
∧ nat-rangep (STRLEN-ADDR, 32)
∧ rom-addrp (STRLEN-ADDR, mc-mem (s), 28)
∧ mcode-addrp (STRLEN-ADDR, mc-mem (s), STRLEN-CODE))

Simultaneously, we introduce the new function symbols *strlen-loadp* and *strlen-*

addr.

THEOREM: stepn-strlen-loadp
strlen-loadp (stepn (*s*, *n*)) = strlen-loadp (*s*)

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

DEFINITION:
strlen1-t (*i*, *n*, *lst*)
= **if** *i* < *n*
  then if get-nth (*i*, *lst*) = NULL **then** 6
    else splus (3, strlen1-t (1 + *i*, *n*, *lst*)) **endif**
  else 0 **endif**

DEFINITION:
strlen-t (*n*, *lst*)
= **if** get-nth (0, *lst*) = 0 **then** 9
  else splus (6, strlen1-t (1, *n*, *lst*)) **endif**

```
; an induction hint.
```

DEFINITION:
strlen-induct (*s*, *i**, *i*, *n*, *lst*)
= **if** *i* < *n*
  then if get-nth (*i*, *lst*) = NULL **then t**
    else strlen-induct (stepn (*s*, 3), add (32, *i**, 1), 1 + *i*, *n*, *lst*) **endif**
  else t endif

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

DEFINITION:
strlen-statep (*s*, *str*, *n*, *lst*)
= ((mc-status (*s*) = 'running)
  ∧ evenp (mc-pc (*s*))
  ∧ rom-addrp (mc-pc (*s*), mc-mem (*s*), 28)
  ∧ mcode-addrp (mc-pc (*s*), mc-mem (*s*), STRLEN-CODE)
  ∧ ram-addrp (sub (32, 4, read-sp (*s*)), mc-mem (*s*), 12)
  ∧ ram-addrp (*str*, mc-mem (*s*), *n*)
  ∧ mem-lst (1, *str*, mc-mem (*s*), *n*, *lst*)
  ∧ disjoint (*str*, *n*, sub (32, 4, read-sp (*s*)), 12)
  ∧ (*str* = read-mem (add (32, read-sp (*s*), 4), mc-mem (*s*), 4))
  ∧ (slen (0, *n*, *lst*) < *n*)
  ∧ (*n* ∈ **N**)
  ∧ uint-rangep (*n*, 32))

```
; an intermediate state.
```

DEFINITION:
strlen-s0p (s, str, $i*$, i, n, lst)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (sub (32, 16, mc-pc (s)), mc-mem (s), 28)
∧ mcode-addrp (sub (32, 16, mc-pc (s)), mc-mem (s), STRLEN-CODE)
∧ ram-addrp (read-an (32, 6, s), mc-mem (s), 12)
∧ ram-addrp (str, mc-mem (s), n)
∧ mem-lst (1, str, mc-mem (s), n, lst)
∧ disjoint (str, n, read-an (32, 6, s), 12)
∧ equal* (read-an (32, 0, s), add (32, str, $i*$))
∧ (str = read-dn (32, 0, s))
∧ (i = nat-to-uint ($i*$))
∧ (slen (i, n, lst) < n)
∧ ($i*$ ∈ **N**)
∧ nat-rangep ($i*$, 32)
∧ (n ∈ **N**)
∧ uint-rangep (n, 32))

```
; from the intial state s to exit: s --> sn.
```

THEOREM: strlen-s-sn
(strlen-statep (s, str, n, lst) ∧ (get-nth (0, lst) = 0))
→ ((mc-status (stepn (s, 9)) = 'running)
∧ (mc-pc (stepn (s, 9)) = rts-addr (s))
∧ (nat-to-uint (read-dn (32, 0, stepn (s, 9))) = 0)
∧ (read-rn (32, 15, mc-rfile (stepn (s, 9)))
= add (32, read-an (32, 7, s), 4))
∧ (read-rn (32, 14, mc-rfile (stepn (s, 9))) = read-an (32, 6, s)))

THEOREM: strlen-s-sn-rfile
(strlen-statep (s, str, n, lst) ∧ (get-nth (0, lst) = 0) ∧ d2-7a2-5p (rn))
→ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 9)))
= read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strlen-s-sn-mem
(strlen-statep (s, str, n, lst)
∧ (get-nth (0, lst) = 0)
∧ disjoint (x, k, sub (32, 4, read-sp (s)), 12))
→ (read-mem (x, mc-mem (stepn (s, 9)), k) = read-mem (x, mc-mem (s), k))

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

THEOREM: strlen-s-s0
(strlen-statep (s, str, n, lst) ∧ (get-nth (0, lst) ≠ 0))
→ strlen-s0p (stepn (s, 6), str, 1, 1, n, lst)

THEOREM: strlen-s-s0-else
(strlen-statep (*s*, *str*, *n*, *lst*) ∧ (get-nth (0, *lst*) ≠ 0))
→ ((linked-rts-addr (stepn (*s*, 6)) = rts-addr (*s*))
∧ (linked-a6 (stepn (*s*, 6)) = read-an (32, 6, *s*))
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 6)))
= sub (32, 4, read-sp (*s*))))

THEOREM: strlen-s-s0-rfile
(strlen-statep (*s*, *str*, *n*, *lst*) ∧ (get-nth (0, *lst*) ≠ 0) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 6)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: strlen-s-s0-mem
(strlen-statep (*s*, *str*, *n*, *lst*)
∧ (get-nth (0, *lst*) ≠ 0)
∧ disjoint (*x*, *k*, sub (32, 4, read-sp (*s*)), 12))
→ (read-mem (*x*, mc-mem (stepn (*s*, 6)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; from s0 to exit (base case), from s0 to s0 (induction case).
; base case: s0 --> exit.
```

THEOREM: strlen-s0-sn-base
(strlen-s0p (*s*, *str*, *i**, *i*, *n*, *lst*) ∧ (get-nth (*i*, *lst*) = 0))
→ ((mc-status (stepn (*s*, 6)) = 'running)
∧ (mc-pc (stepn (*s*, 6)) = linked-rts-addr (*s*))
∧ (nat-to-uint (read-dn (32, 0, stepn (*s*, 6))) = *i*)
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 6))) = linked-a6 (*s*))
∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 6)))
= add (32, read-an (32, 6, *s*), 8))
∧ (read-mem (*x*, mc-mem (stepn (*s*, 6)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strlen-s0-sn-rfile-base
(strlen-s0p (*s*, *str*, *i**, *i*, *n*, *lst*) ∧ d2-7a2-5p (*rn*) ∧ (get-nth (*i*, *lst*) = 0))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 6)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; induction case: s0 --> s0.
```

THEOREM: strlen-s0-s0
(strlen-s0p (*s*, *str*, *i**, *i*, *n*, *lst*) ∧ (get-nth (*i*, *lst*) ≠ 0))
→ (strlen-s0p (stepn (*s*, 3), *str*, add (32, *i**, 1), 1 + *i*, *n*, *lst*)
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 3)))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-a6 (stepn (*s*, 3)) = linked-a6 (*s*))

$\wedge$ (linked-rts-addr (stepn (s, 3)) = linked-rts-addr (s))
$\wedge$ (read-mem (x, mc-mem (stepn (s, 3)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: strlen-s0-s0-rfile
(strlen-s0p (s, str, i^*, i, n, lst) $\wedge$ d2-7a2-5p (rn) $\wedge$ (get-nth (i, lst) $\neq$ 0))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 3)))
= read-rn ($oplen$, rn, mc-rfile (s)))

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

THEOREM: strlen-s0p-info
strlen-s0p (s, str, i^*, i, n, lst) $\rightarrow$ (($i < n$) = **t**)

THEOREM: strlen-s0-sn
let sn **be** stepn (s, strlen1-t (i, n, lst))
in
strlen-s0p (s, str, i^*, i, n, lst)
$\rightarrow$ ((mc-status (sn) = 'running)
$\wedge$ (mc-pc (sn) = linked-rts-addr (s))
$\wedge$ (nat-to-uint (read-dn (32, 0, sn)) = strlen (i, n, lst))
$\wedge$ (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**

EVENT: Disable strlen-s0p-info.

THEOREM: strlen-s0-sn-rfile
(strlen-s0p (s, str, i^*, i, n, lst) $\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, strlen1-t (i, n, lst))))
= read-rn ($oplen$, rn, mc-rfile (s)))

```
; the correctness of the STRLEN program.
```

THEOREM: strlen-statep-info
strlen-statep (s, str, n, lst) $\rightarrow$ ($n \not\simeq$ 0)

THEOREM: strlen-correctness
let sn **be** stepn (s, strlen-t (n, lst))
in
strlen-statep (s, str, n, lst)
$\rightarrow$ ((mc-status (sn) = 'running)
$\wedge$ (mc-pc (sn) = rts-addr (s))
$\wedge$ (read-rn (32, 14, mc-rfile (sn))

$$
\begin{array}{ll}
 & = \quad \text{read-rn}\,(\texttt{32},\ \texttt{14},\ \text{mc-rfile}\,(s))) \\
\wedge & (\text{read-rn}\,(\texttt{32},\ \texttt{15},\ \text{mc-rfile}\,(sn)) \\
 & \quad = \quad \text{add}\,(\texttt{32},\ \text{read-sp}\,(s),\ \texttt{4})) \\
\wedge & (\text{d2-7a2-5p}\,(rn) \\
 & \quad \rightarrow \quad (\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(sn)) \\
 & \qquad = \quad \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))) \\
\wedge & (\text{disjoint}\,(x,\ k,\ \text{sub}\,(\texttt{32},\ \texttt{4},\ \text{read-sp}\,(s)),\ \texttt{12}) \\
 & \quad \rightarrow \quad (\text{read-mem}\,(x,\ \text{mc-mem}\,(sn),\ k) \\
 & \qquad = \quad \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k))) \\
\wedge & (\text{uread-dn}\,(\texttt{32},\ \texttt{0},\ sn) = \text{strlen}\,(\texttt{0},\ n,\ lst))) \ \textbf{endlet}
\end{array}
$$

EVENT: Disable strlen-t.

```
; some properties of strlen.
; see the file cstring.events.
```

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

Index