```
#|

 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 STRCAT 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 strcat function in the Berkeley string library.

/* concatenate char append[] to the end of s[] */
char *
strcat(s, append)
register char *s, *append;
{
char *save = s;

for (; *s; ++s);
while (*s++ = *append++);
```

```
return(save);
}

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

0x24d8 <strcat>:        linkw fp,#0
0x24dc <strcat+4>:      moveal fp@(8),a0
0x24e0 <strcat+8>:      moveal fp@(12),a1
0x24e4 <strcat+12>:     movel a0,d1
0x24e6 <strcat+14>:     tstb a0@
0x24e8 <strcat+16>:     beq 0x24f0 <strcat+24>
0x24ea <strcat+18>:     addqw #1,a0
0x24ec <strcat+20>:     tstb a0@
0x24ee <strcat+22>:     bne 0x24ea <strcat+18>
0x24f0 <strcat+24>:     moveb a1@+,d0
0x24f2 <strcat+26>:     moveb d0,a0@+
0x24f4 <strcat+28>:     bne 0x24f0 <strcat+24>
0x24f6 <strcat+30>:     movel d1,d0
0x24f8 <strcat+32>:     unlk fp
0x24fa <strcat+34>:     rts

The machine code of the above program is:

<strcat>:      0x4e56  0x0000  0x206e  0x0008  0x226e  0x000c  0x2208  0x4a10
<strcat+16>:   0x6706  0x5248  0x4a10  0x66fa  0x1019  0x10c0  0x66fa  0x2001
<strcat+32>:   0x4e5e  0x4e75

'(78      86      0       0       32      110     0       8
  34      110     0       12      34      8       74      16
  103     6       82      72      74      16      102     250
  16      25      16      192     102     250     32      1
  78      94      78      117)

Bird-eye view of the control flow of the program:


   s -------> s0* -------> s1* --------> exit
    \_____________________/

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

DEFINITION:

STRCAT-CODE
= '(78 86 0 0 32 110 0 8 34 110 0 12 34 8 74 16 103 6 82
72 74 16 102 250 16 25 16 192 102 250 32 1 78 94 78
117)

; the computation time of the program.

DEFINITION:
strcat-t0 (*i*, *n1*, *lst1*)
= **if** *i* < *n1*
then if get-nth (*i*, *lst1*) = 0 **then** 2
else splus (3, strcat-t0 (1 + *i*, *n1*, *lst1*)) **endif**
else 0 **endif**

DEFINITION: strcat-t1 (*n1*, *lst1*) = splus (7, strcat-t0 (1, *n1*, *lst1*))

DEFINITION:
strcat-t2 (*j*, *n2*, *lst2*)
= **if** *j* < *n2*
then if get-nth (*j*, *lst2*) = 0 **then** 6
else splus (3, strcat-t2 (1 + *j*, *n2*, *lst2*)) **endif**
else 0 **endif**

DEFINITION:
strcat-t (*n1*, *lst1*, *n2*, *lst2*)
= **if** get-nth (0, *lst1*) = 0 **then** splus (6, strcat-t2 (0, *n2*, *lst2*))
else splus (strcat-t1 (*n1*, *lst1*), strcat-t2 (0, *n2*, *lst2*)) **endif**

; two induction hints for the two loops in the program.

DEFINITION:
strcat-induct0 (*s*, *i**, *i*, *lst1*, *n1*)
= **if** *i* < *n1*
then if get-nth (*i*, *lst1*) = 0 **then t**
else strcat-induct0 (stepn (*s*, 3),
add (32, *i**, 1),
1 + *i*,
lst1,
n1) **endif**
else t endif

DEFINITION:
strcat-induct1 (*s*, *i**, *i*, *lst1*, *j**, *j*, *n2*, *lst2*)
= **if** *j* < *n2*
then if get-nth (*j*, *lst2*) = 0 **then t**

else strcat-induct1 (stepn (*s*, 3),
add (32, *i**, 1),
1 + *i*,
put-nth (get-nth (*j*, *lst2*), *i*, *lst1*),
add (32, *j**, 1),
1 + *j*,
n2,
lst2) **endif**

else t endif

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

DEFINITION:
strcat-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
∧ evenp (mc-pc (*s*))
∧ rom-addrp (mc-pc (*s*), mc-mem (*s*), 36)
∧ mcode-addrp (mc-pc (*s*), mc-mem (*s*), STRCAT-CODE)
∧ ram-addrp (sub (32, 4, read-sp (*s*)), mc-mem (*s*), 16)
∧ 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-sp (*s*)), 16, *str1*, *n1*)
∧ disjoint (sub (32, 4, read-sp (*s*)), 16, *str2*, *n2*)
∧ disjoint (*str1*, *n1*, *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))
∧ ((slen (0, *n1*, *lst1*) + slen (0, *n2*, *lst2*)) < *n1*)
∧ (slen (0, *n2*, *lst2*) < *n2*)
∧ (*n1* ∈ **N**)
∧ (*n2* ∈ **N**)
∧ uint-rangep (*n1*, 32))

```
; an intermediate state.
```

DEFINITION:
strcat-s0p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
∧ evenp (mc-pc (*s*))
∧ rom-addrp (sub (32, 20, mc-pc (*s*)), mc-mem (*s*), 36)
∧ mcode-addrp (sub (32, 20, mc-pc (*s*)), mc-mem (*s*), STRCAT-CODE)
∧ ram-addrp (read-an (32, 6, *s*), mc-mem (*s*), 16)
∧ 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 (read-an (32, 6, *s*), 16, *str1*, *n1*)
∧ disjoint (read-an (32, 6, *s*), 16, *str2*, *n2*)
∧ disjoint (*str1*, *n1*, *str2*, *n2*)
∧ equal* (read-an (32, 0, *s*), add (32, *str1*, *i**))
∧ (*str1* = read-dn (32, 1, *s*))
∧ (*str2* = read-an (32, 1, *s*))
∧ ((slen (*i*, *n1*, *lst1*) + slen (0, *n2*, *lst2*)) < *n1*)
∧ (slen (0, *n2*, *lst2*) < *n2*)
∧ (*i** ∈ **N**)
∧ nat-rangep (*i**, 32)
∧ (*i* = nat-to-uint (*i**))
∧ (*n1* ∈ **N**)
∧ (*n2* ∈ **N**)
∧ uint-rangep (*n1*, 32))

```
; an intermediate state.
```

DEFINITION:
strcat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n2*, *lst2*, *i*_)
= ((mc-status (*s*) = 'running)
∧ evenp (mc-pc (*s*))
∧ rom-addrp (sub (32, 24, mc-pc (*s*)), mc-mem (*s*), 36)
∧ mcode-addrp (sub (32, 24, mc-pc (*s*)), mc-mem (*s*), STRCAT-CODE)
∧ ram-addrp (read-an (32, 6, *s*), mc-mem (*s*), 16)
∧ 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 (read-an (32, 6, *s*), 16, *str1*, *n1*)
∧ disjoint (read-an (32, 6, *s*), 16, *str2*, *n2*)
∧ disjoint (*str1*, *n1*, *str2*, *n2*)
∧ equal* (read-an (32, 0, *s*), add (32, *str1*, *i**))
∧ equal* (read-an (32, 1, *s*), add (32, *str2*, *j**))
∧ (*str1* = read-dn (32, 1, *s*))
∧ ((*i*_ + slen (*j*, *n2*, *lst2*)) < *n1*)
∧ (slen (*j*, *n2*, *lst2*) < *n2*)
∧ (*i* ≤ (*i*_ + *j*))
∧ (*i*_ ∈ **N**)
∧ (*i** ∈ **N**)
∧ nat-rangep (*i**, 32)
∧ (*i* = nat-to-uint (*i**))
∧ (*j** ∈ **N**)

$\wedge$ nat-rangep (j^*, 32)
$\wedge$ (j = nat-to-uint (j^*))
$\wedge$ ($n1 \in \mathbf{N}$)
$\wedge$ ($n2 \in \mathbf{N}$)
$\wedge$ uint-rangep ($n1$, 32))

```
; from the initial state to s1:  s --> s1, if lst1[0] == 0.
```

THEOREM: strcat-s-s1-1
(strcat-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$) $\wedge$ (get-nth (0, $lst1$) = 0))
$\rightarrow$ (strcat-s1p (stepn (s, 6), 0, 0, $str1$, $n1$, $lst1$, 0, 0, $str2$, $n2$, $lst2$, 0)
$\wedge$ (linked-rts-addr (stepn (s, 6)) = rts-addr (s))
$\wedge$ (linked-a6 (stepn (s, 6)) = read-an (32, 6, s))
$\wedge$ equal* (read-rn (32, 14, mc-rfile (stepn (s, 6))),
sub (32, 4, read-sp (s))))

THEOREM: strcat-s-s1-1-rfile
(strcat-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\wedge$ (get-nth (0, $lst1$) = 0)
$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 6)))
= read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strcat-s-s1-1-mem
(strcat-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\wedge$ (get-nth (0, $lst1$) = 0)
$\wedge$ disjoint (x, k, sub (32, 4, read-sp (s)), 16))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 6)), k) = read-mem (x, mc-mem (s), k))

```
; from the initial state to s0:  s --> s0, if lst1[0] =\= 0.
```

THEOREM: strcat-s-s0
(strcat-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$) $\wedge$ (get-nth (0, $lst1$) $\neq$ 0))
$\rightarrow$ (strcat-s0p (stepn (s, 7), 1, 1, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\wedge$ (linked-rts-addr (stepn (s, 7)) = rts-addr (s))
$\wedge$ (linked-a6 (stepn (s, 7)) = read-an (32, 6, s))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 7)))
= sub (32, 4, read-sp (s))))

THEOREM: strcat-s-s0-rfile
(strcat-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\wedge$ (get-nth (0, $lst1$) $\neq$ 0)
$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 7)))
= read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strcat-s-s0-mem
(strcat-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 $\wedge$ (get-nth (0, *lst1*) $\neq$ 0)
 $\wedge$ disjoint (x, k, sub (32, 4, read-sp (s)), 16))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 7)), k) = read-mem (x, mc-mem (s), k))

```
; from s0 to s1:  s0 --> s1.
; base case: s0 --> s1, when lst1[i] == 0.
```

THEOREM: strcat-s0-s1-base
(strcat-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) $\wedge$ (get-nth (i, *lst1*) = 0))
$\rightarrow$ (strcat-s1p (stepn (s, 2), i*, i, *str1*, *n1*, *lst1*, 0, 0, *str2*, *n2*, *lst2*, i)
 $\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 2)))
 = read-rn (32, 14, mc-rfile (s)))
 $\wedge$ (linked-a6 (stepn (s, 2)) = linked-a6 (s))
 $\wedge$ (linked-rts-addr (stepn (s, 2)) = linked-rts-addr (s))
 $\wedge$ (read-mem (x, mc-mem (stepn (s, 2)), k)
 = read-mem (x, mc-mem (s), k)))

THEOREM: strcat-s0-s1-rfile-base
(strcat-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 $\wedge$ (get-nth (i, *lst1*) = 0)
 $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 2)))
 = read-rn (*oplen*, *rn*, mc-rfile (s)))

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

THEOREM: strcat-s0-s0
(strcat-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) $\wedge$ (get-nth (i, *lst1*) $\neq$ 0))
$\rightarrow$ (strcat-s0p (stepn (s, 3), add (32, i*, 1), 1 + i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 $\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 3)))
 = read-rn (32, 14, mc-rfile (s)))
 $\wedge$ (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: strcat-s0-s0-rfile
(strcat-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 $\wedge$ (get-nth (i, *lst1*) $\neq$ 0)
 $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 3)))
 = read-rn (*oplen*, *rn*, mc-rfile (s)))

```
; put together:  s0 --> s1.
```

THEOREM: strcat-s0p-info
strcat-s0p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) → ((*i* < *n1*) = **t**)

THEOREM: strcat-s0-s1
let *s1* **be** stepn (*s*, strcat-t0 (*i*, *n1*, *lst1*))
in
strcat-s0p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ (strcat-s1p (*s1*,
strlen* (*i**, *i*, *n1*, *lst1*),
strlen (*i*, *n1*, *lst1*),
str1,
n1,
lst1,
0,
0,
str2,
n2,
lst2,
strlen (*i*, *n1*, *lst1*))
∧ (read-rn (32, 14, mc-rfile (*s1*))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-a6 (*s1*) = linked-a6 (*s*))
∧ (linked-rts-addr (*s1*) = linked-rts-addr (*s*))
∧ (read-mem (*x*, mc-mem (*s1*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

EVENT: Disable strcat-s0p-info.

THEOREM: strcat-s0-s1-rfile
(strcat-s0p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strcat-t0 (*i*, *n1*, *lst1*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; from s1 to exit:  s1 --> sn.
; base case:  s1 --> sn, when lst2[j] == 0.
```

THEOREM: strcat-s1-sn-base
(strcat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n2*, *lst2*, *i*_)
∧ (get-nth (*j*, *lst2*) = 0))
→ ((mc-status (stepn (*s*, 6)) = 'running)
∧ (mc-pc (stepn (*s*, 6)) = linked-rts-addr (*s*))
∧ (read-rn (32, 0, mc-rfile (stepn (*s*, 6))) = *str1*)
∧ mem-lst (1, *str1*, mc-mem (stepn (*s*, 6)), *n1*, put-nth (0, *i*, *lst1*))
∧ (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)))

THEOREM: strcat-s1-sn-rfile-base
(strcat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n2*, *lst2*, *i_*)
 ∧ (get-nth (*j*, *lst2*) = 0)
 ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 6)))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: strcat-s1-sn-mem-base
(strcat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n2*, *lst2*, *i_*)
 ∧ (get-nth (*j*, *lst2*) = 0)
 ∧ disjoint (*x*, *k*, *str1*, *n1*))
→ (read-mem (*x*, mc-mem (stepn (*s*, 6)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; induction case: s1 --> s1.
```

THEOREM: strcat-s1-s1
(strcat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n2*, *lst2*, *i_*)
 ∧ (get-nth (*j*, *lst2*) ≠ 0))
→ (strcat-s1p (stepn (*s*, 3),
 add (32, *i**, 1),
 1 + *i*,
 str1,
 n1,
 put-nth (get-nth (*j*, *lst2*), *i*, *lst1*),
 add (32, *j**, 1),
 1 + *j*,
 str2,
 n2,
 lst2,
 *i_*)
 ∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 3)))
 = read-rn (32, 14, mc-rfile (*s*)))
 ∧ (linked-a6 (stepn (*s*, 3)) = linked-a6 (*s*))
 ∧ (linked-rts-addr (stepn (*s*, 3)) = linked-rts-addr (*s*)))

THEOREM: strcat-s1-s1-rfile
(strcat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n2*, *lst2*, *i_*)
 ∧ (get-nth (*j*, *lst2*) ≠ 0)
 ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 3)))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: strcat-s1-s1-mem
(strcat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n2*, *lst2*, *i_*)
 ∧ (get-nth (*j*, *lst2*) ≠ 0)

```
  ∧   disjoint (x, k, str1, n1))
→   (read-mem (x, mc-mem (stepn (s, 3)), k) = read-mem (x, mc-mem (s), k))

; put together. s1 --> sn.

THEOREM: strcat-s1-info
strcat-s1p (s, i*, i, str1, n1, lst1, j*, j, str2, n2, lst2, i_) → ((j < n2) = t)

THEOREM: strcat-s1-sn
let sn be stepn (s, strcat-t2 (j, n2, lst2))
in
strcat-s1p (s, i*, i, str1, n1, lst1, j*, j, str2, n2, lst2, i_)
→   ((mc-status (sn) = 'running)
      ∧   (mc-pc (sn) = linked-rts-addr (s))
      ∧   (read-dn (32, 0, sn) = str1)
      ∧   mem-lst (1,
                   str1,
                   mc-mem (sn),
                   n1,
                   strcpy1 (i, lst1, j, n2, lst2))
      ∧   (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: strcat-s1-sn-rfile
(strcat-s1p (s, i*, i, str1, n1, lst1, j*, j, str2, n2, lst2, i_) ∧ d2-7a2-5p (rn))
→   (read-rn (oplen, rn, mc-rfile (stepn (s, strcat-t2 (j, n2, lst2))))
      =   read-rn (oplen, rn, mc-rfile (s)))

THEOREM: strcat-s1-sn-mem
(strcat-s1p (s, i*, i, str1, n1, lst1, j*, j, str2, n2, lst2, i_)
 ∧   disjoint (x, k, str1, n1))
→   (read-mem (x, mc-mem (stepn (s, strcat-t2 (j, n2, lst2))), k)
      =   read-mem (x, mc-mem (s), k))

; put together: s --> s1, if lst[0] =\= 0.

THEOREM: strcat-s-s1-2
let s1 be stepn (s, strcat-t1 (n1, lst1))
in
(strcat-statep (s, str1, n1, lst1, str2, n2, lst2)
 ∧   (get-nth (0, lst1) ≠ 0))
→   (strcat-s1p (s1,
                 strlen* (1, 1, n1, lst1),
                 strlen (1, n1, lst1),
```

str1,
n1,
lst1,
0,
0,
str2,
n2,
lst2,
strlen (1, *n1*, *lst1*))
∧ (read-rn (32, 14, mc-rfile (*s1*))
= sub (32, 4, read-sp (*s*)))
∧ (linked-a6 (*s1*) = read-an (32, 6, *s*))
∧ (linked-rts-addr (*s1*) = rts-addr (*s*))) **endlet**

THEOREM: strcat-s-s1-2-rfile
(strcat-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ (get-nth (0, *lst1*) ≠ 0)
∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strcat-t1 (*n1*, *lst1*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: strcat-s-s1-2-mem
(strcat-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ (get-nth (0, *lst1*) ≠ 0)
∧ disjoint (*x*, *k*, sub (32, 4, read-sp (*s*)), 16))
→ (read-mem (*x*, mc-mem (stepn (*s*, strcat-t1 (*n1*, *lst1*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*))

```
; the correctness of the strcat program.
```

THEOREM: strcat-correctness
let *sn* **be** stepn (*s*, strcat-t (*n1*, *lst1*, *n2*, *lst2*))
in
strcat-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ ((mc-status (*sn*) = 'running)
∧ (mc-pc (*sn*) = rts-addr (*s*))
∧ (read-an (32, 6, *sn*) = read-an (32, 6, *s*))
∧ (read-an (32, 7, *sn*) = add (32, read-an (32, 7, *s*), 4))
∧ (d2-7a2-5p (*rn*)
→ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))))
∧ ((disjoint (*x*, *k*, *str1*, *n1*)
∧ disjoint (*x*, *k*, sub (32, 4, read-sp (*s*)), 16))
→ (read-mem (*x*, mc-mem (*sn*), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

$\wedge$ (read-dn (32, 0, *sn*) = *str1*)
$\wedge$ mem-lst (1, *str1*, mc-mem (*sn*), *n1*, strcat (*n1*, *lst1*, *n2*, *lst2*))) **endlet**

EVENT: Disable strcat-t.

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

Index