```
#|

 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 STRSPN Function
#|
This is part of our effort to verify the Berkeley C string library.  The
Berkeley string library is widely used as part of the Berkeley Unix OS.

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

size_t
strspn(s1, s2)
const char *s1;
register const char *s2;
{
register const char *p = s1, *spanp;
register char c, sc;

/*
```

```
 * Skip any characters in s2, excluding the terminating \0.
 */
cont:
c = *p++;
for (spanp = s2; (sc = *spanp++) != 0;)
if (sc == c)
goto cont;
return (p - 1 - s1);
}
```

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

```
0x26e8 <strspn>:        linkw fp,#0
0x26ec <strspn+4>:      movel d2,sp@-
0x26ee <strspn+6>:      movel fp@(8),d2
0x26f2 <strspn+10>:     moveal d2,a1
0x26f4 <strspn+12>:     moveb a1@+,d1
0x26f6 <strspn+14>:     moveal fp@(12),a0
0x26fa <strspn+18>:     bra 0x2700 <strspn+24>
0x26fc <strspn+20>:     cmpb d0,d1
0x26fe <strspn+22>:     beq 0x26f4 <strspn+12>
0x2700 <strspn+24>:     moveb a0@+,d0
0x2702 <strspn+26>:     bne 0x26fc <strspn+20>
0x2704 <strspn+28>:     movel d2,d0
0x2706 <strspn+30>:     addql #1,d0
0x2708 <strspn+32>:     subl a1,d0
0x270a <strspn+34>:     negl d0
0x270c <strspn+36>:     movel fp@(-4),d2
0x2710 <strspn+40>:     unlk fp
0x2712 <strspn+42>:     rts
```

The machine code of the above program is:

```
<strspn>:      0x4e56  0x0000  0x2f02  0x242e  0x0008  0x2242  0x1219  0x206e
<strspn+16>:   0x000c  0x6004  0xb200  0x67f4  0x1018  0x66f8  0x2002  0x5280
<strspn+32>:   0x9089  0x4480  0x242e  0xfffc  0x4e5e  0x4e75
```

```
'(78      86      0       0       47      2       36      46
  0       8       34      66      18      25      32      110
  0       12      96      4       178     0       103     244
  16      24      102     248     32      2       82      128
  144     137     68      128     36      46      255     252
  78      94      78      117)
```

```
|#

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

DEFINITION:
STRSPN-CODE
= `'(78 86 0 0 47 2 36 46 0 8 34 66 18 25 32 110 0 12 96 4 178 0 103 244 16 24 102 248 32 2 82 128 144 137 68 128 36 46 255 252 78 94 78 117)`

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

DEFINITION:
strspn-t0 (*i2*, *n2*, *lst2*, *ch*)
= **if** *i2* < *n2*
then if get-nth (*i2*, *lst2*) = `0` **then** `9`
elseif get-nth (*i2*, *lst2*) = *ch* **then** `4`
else splus (`4`, strspn-t0 (1 + *i2*, *n2*, *lst2*, *ch*)) **endif**
else `0` **endif**

DEFINITION:
strspn-t1 (*n2*, *lst2*, *ch*) = splus (`3`, strspn-t0 (`0`, *n2*, *lst2*, *ch*))

DEFINITION:
strspn-t2 (*i1*, *n1*, *lst1*, *n2*, *lst2*)
= **if** *i1* < *n1*
then if strchr1 (`0`, *n2*, *lst2*, get-nth (*i1*, *lst1*))
then splus (strspn-t1 (*n2*, *lst2*, get-nth (*i1*, *lst1*)),
strspn-t2 (1 + *i1*, *n1*, *lst1*, *n2*, *lst2*))
else strspn-t1 (*n2*, *lst2*, get-nth (*i1*, *lst1*)) **endif**
else `0` **endif**

DEFINITION:
strspn-t (*n1*, *lst1*, *n2*, *lst2*) = splus (`4`, strspn-t2 (`0`, *n1*, *lst1*, *n2*, *lst2*))

```
; two induction hints.
```

DEFINITION:
strspn-induct0 (*s*, *i2**, *i2*, *n2*, *lst2*, *ch*)
= **if** *i2* < *n2*
then if get-nth (*i2*, *lst2*) = `0` **then t**
elseif get-nth (*i2*, *lst2*) = *ch* **then t**
else strspn-induct0 (stepn (*s*, `4`),
add (`32`, *i2**, `1`),
1 + *i2*,

n2,
lst2,
ch) **endif**
else t endif

DEFINITION:
strspn-induct1 (*s*, *i1**, *i1*, *n1*, *lst1*, *n2*, *lst2*)
= **if** *i1* < *n1*
then if strchr1 (0, *n2*, *lst2*, get-nth (*i1*, *lst1*))
then strspn-induct1 (stepn (*s*, strspn-t1 (*n2*, *lst2*, get-nth (*i1*, *lst1*))),
add (32, *i1**, 1),
1 + *i1*,
n1,
lst1,
n2,
lst2)
else t endif
else t endif

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

DEFINITION:
strspn-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
∧ evenp (mc-pc (*s*))
∧ rom-addrp (mc-pc (*s*), mc-mem (*s*), 44)
∧ mcode-addrp (mc-pc (*s*), mc-mem (*s*), STRSPN-CODE)
∧ 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))
∧ (slen (0, *n1*, *lst1*) < *n1*)
∧ (slen (0, *n2*, *lst2*) < *n2*)
∧ (*n1* ∈ **N**)
∧ (*n2* ∈ **N**)
∧ uint-rangep (*n1*, 32)
∧ uint-rangep (*n2*, 32))

DEFINITION:
strspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)

$=$ ((mc-status (s) = 'running)
$\wedge$ evenp (mc-pc (s))
$\wedge$ rom-addrp (sub (32, 12, mc-pc (s)), mc-mem (s), 44)
$\wedge$ mcode-addrp (sub (32, 12, mc-pc (s)), mc-mem (s), STRSPN-CODE)
$\wedge$ ram-addrp (sub (32, 4, read-an (32, 6, s)), mc-mem (s), 20)
$\wedge$ ram-addrp ($str1$, mc-mem (s), $n1$)
$\wedge$ mem-lst (1, $str1$, mc-mem (s), $n1$, $lst1$)
$\wedge$ ram-addrp ($str2$, mc-mem (s), $n2$)
$\wedge$ mem-lst (1, $str2$, mc-mem (s), $n2$, $lst2$)
$\wedge$ disjoint (sub (32, 4, read-an (32, 6, s)), 20, $str1$, $n1$)
$\wedge$ disjoint (sub (32, 4, read-an (32, 6, s)), 20, $str2$, $n2$)
$\wedge$ equal* (read-an (32, 1, s), add (32, $str1$, $i1*$))
$\wedge$ ($str2$ = read-mem (add (32, read-an (32, 6, s), 12), mc-mem (s), 4))
$\wedge$ ($str1$ = read-dn (32, 2, s))
$\wedge$ ($i1* \in \mathbf{N}$)
$\wedge$ nat-rangep ($i1*$, 32)
$\wedge$ ($i1$ = nat-to-uint ($i1*$))
$\wedge$ (slen ($i1$, $n1$, $lst1$) $<$ $n1$)
$\wedge$ (slen (0, $n2$, $lst2$) $<$ $n2$)
$\wedge$ ($n1 \in \mathbf{N}$)
$\wedge$ ($n2 \in \mathbf{N}$)
$\wedge$ uint-rangep ($n1$, 32)
$\wedge$ uint-rangep ($n2$, 32))

DEFINITION:
strspn-s1-1p (s, $i1*$, $i1$, $str1$, $n1$, $lst1$, $i2*$, $i2$, $str2$, $n2$, $lst2$, ch)
$=$ ((mc-status (s) = 'running)
$\wedge$ evenp (mc-pc (s))
$\wedge$ rom-addrp (sub (32, 24, mc-pc (s)), mc-mem (s), 44)
$\wedge$ mcode-addrp (sub (32, 24, mc-pc (s)), mc-mem (s), STRSPN-CODE)
$\wedge$ ram-addrp (sub (32, 4, read-an (32, 6, s)), mc-mem (s), 20)
$\wedge$ ram-addrp ($str2$, mc-mem (s), $n2$)
$\wedge$ mem-lst (1, $str2$, mc-mem (s), $n2$, $lst2$)
$\wedge$ disjoint (sub (32, 4, read-an (32, 6, s)), 20, $str2$, $n2$)
$\wedge$ ($str1$ = read-dn (32, 2, s))
$\wedge$ ($str2$ = read-mem (add (32, read-an (32, 6, s), 12), mc-mem (s), 4))
$\wedge$ (ch = uread-dn (8, 1, s))
$\wedge$ equal* (read-an (32, 1, s), add (32, $str1$, $i1*$))
$\wedge$ equal* (read-an (32, 0, s), add (32, $str2$, $i2*$))
$\wedge$ ($i1* \in \mathbf{N}$)
$\wedge$ nat-rangep ($i1*$, 32)
$\wedge$ ($i1$ = nat-to-uint ($i1*$))
$\wedge$ ($i2* \in \mathbf{N}$)
$\wedge$ nat-rangep ($i2*$, 32)

$\wedge$ $(i2 = \text{nat-to-uint}\,(i2^*))$
$\wedge$ $(\text{slen}\,(i2,\ n2,\ lst2) < n2)$
$\wedge$ $(n2 \in \mathbf{N})$
$\wedge$ $\text{uint-rangep}\,(n2,\ 32)$)

DEFINITION:
strspn-s1p $(s,\ i1^*,\ i1,\ str1,\ n1,\ lst1,\ i2^*,\ i2,\ str2,\ n2,\ lst2,\ ch)$
$=$ (strspn-s1-1p $(s,\ i1^*,\ i1,\ str1,\ n1,\ lst1,\ i2^*,\ i2,\ str2,\ n2,\ lst2,\ ch)$
$\wedge$ ram-addrp $(str1,\ \text{mc-mem}\,(s),\ n1)$
$\wedge$ mem-lst $(1,\ str1,\ \text{mc-mem}\,(s),\ n1,\ lst1)$
$\wedge$ disjoint $(\text{sub}\,(32,\ 4,\ \text{read-an}\,(32,\ 6,\ s)),\ 20,\ str1,\ n1)$
$\wedge$ $(\text{slen}\,(i1,\ n1,\ lst1) < n1)$
$\wedge$ $(\text{slen}\,(0,\ n2,\ lst2) < n2)$
$\wedge$ $(n1 \in \mathbf{N})$
$\wedge$ $\text{uint-rangep}\,(n1,\ 32)$)

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

THEOREM: strspn-s-s0
strspn-statep $(s,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$
$\rightarrow$ strspn-s0p $(\text{stepn}\,(s,\ 4),\ 0,\ 0,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$

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

THEOREM: strspn-s-s0-rfile
$(\text{strspn-statep}\,(s,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2) \wedge \text{d3-7a2-5p}\,(rn))$
$\rightarrow$ $(\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 4)))$
$= \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))$

THEOREM: strspn-s-s0-mem
$(\text{strspn-statep}\,(s,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$
$\wedge$ $\text{disjoint}\,(x,\ k,\ \text{sub}\,(32,\ 8,\ \text{read-sp}\,(s)),\ 20))$
$\rightarrow$ $(\text{read-mem}\,(x,\ \text{mc-mem}\,(\text{stepn}\,(s,\ 4)),\ k) = \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k))$

```
; loop 0.
; from s0 to s1: s0 --> s1.
```

THEOREM: strspn-s0-s1-1
strspn-s0p $(s,\ i1^*,\ i1,\ str1,\ n1,\ lst1,\ str2,\ n2,\ lst2)$

```
→   strspn-s1-1p (stepn (s, 3),
                  add (32, i1*, 1),
                  1 + i1,
                  str1,
                  n1,
                  lst1,
                  0,
                  0,
                  str2,
                  n2,
                  lst2,
                  get-nth (i1, lst1))
```

THEOREM: strchr1-la
strchr1 ($i2$, $n2$, $lst2$, ch) → ($ch \neq 0$)

THEOREM: strspn-s0-s1
```
(strspn-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2)
 ∧   strchr1 (0, n2, lst2, get-nth (i1, lst1)))
→   strspn-s1p (stepn (s, 3),
                 add (32, i1*, 1),
                 1 + i1,
                 str1,
                 n1,
                 lst1,
                 0,
                 0,
                 str2,
                 n2,
                 lst2,
                 get-nth (i1, lst1))
```

THEOREM: strspn-s0-s1-else
```
let s1 be stepn (s, 3)
in
strspn-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2)
→   ((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))
     ∧   (rn-saved (s1) = rn-saved (s))
     ∧   (read-mem (x, mc-mem (s1), k) = read-mem (x, mc-mem (s), k))) endlet
```

THEOREM: strspn-s0-s1-rfile
(strspn-s0p (s, $i1*$, $i1$, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$) ∧ d3-7a2-5p (rn))

$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 3)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; loop 1.
; base case 1. from s1 to exit: s1 --> sn, when lst1[i] == 0.
```

THEOREM: strspn-s1-sn-base
(strspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
$\wedge$ (get-nth (*i2*, *lst2*) = 0))
$\rightarrow$ ((mc-status (stepn (*s*, 9)) = 'running)
$\wedge$ (mc-pc (stepn (*s*, 9)) = linked-rts-addr (*s*))
$\wedge$ (read-dn (32, 0, stepn (*s*, 9)) = sub (32, 1, *i1**))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (*s*, 9))) = linked-a6 (*s*))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (*s*, 9)))
= add (32, read-an (32, 6, *s*), 8))
$\wedge$ (read-mem (*x*, mc-mem (stepn (*s*, 9)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strspn-s1-sn-rfile-base
(strspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
$\wedge$ (get-nth (*i2*, *lst2*) = 0)
$\wedge$ (*oplen* $\leq$ 32)
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 9)))
= **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
else head (rn-saved (*s*), *oplen*) **endif**)

```
; base case 2.  s1 --> s0, when lst1[i] =\= 0  and lst2[j] = ch.
```

THEOREM: strspn-s1-s0-base
(strspn-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
$\wedge$ (get-nth (*i2*, *lst2*) $\neq$ 0)
$\wedge$ (get-nth (*i2*, *lst2*) = *ch*))
$\rightarrow$ (strspn-s0p (stepn (*s*, 4), *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ (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*))
$\wedge$ (read-mem (*x*, mc-mem (stepn (*s*, 4)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strspn-s1-s0-rfile-base
(strspn-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
$\wedge$ (get-nth (*i2*, *lst2*) $\neq$ 0)
$\wedge$ (get-nth (*i2*, *lst2*) = *ch*)

$\wedge$ d3-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 4)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; induction case. s1 --> s1, when lst1[i] =\= 0 and lst2[j] =\= ch.
```

THEOREM: strspn-s1-s1-1
(strspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
$\wedge$ (get-nth (*i2*, *lst2*) $\neq$ 0)
$\wedge$ (get-nth (*i2*, *lst2*) $\neq$ *ch*))
$\rightarrow$ (strspn-s1-1p (stepn (*s*, 4),
*i1**,
i1,
str1,
n1,
lst1,
add (32, *i2**, 1),
1 + *i2*,
str2,
n2,
lst2,
ch)
$\wedge$ (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: strspn-s1-s1
(strspn-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
$\wedge$ (get-nth (*i2*, *lst2*) $\neq$ 0)
$\wedge$ (get-nth (*i2*, *lst2*) $\neq$ *ch*))
$\rightarrow$ strspn-s1p (stepn (*s*, 4),
*i1**,
i1,
str1,
n1,
lst1,
add (32, *i2**, 1),
1 + *i2*,
str2,
n2,
lst2,

ch)

THEOREM: strspn-s1-s1-rfile
(strspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (get-nth (*i2*, *lst2*) ≠ 0)
 ∧ (get-nth (*i2*, *lst2*) ≠ *ch*)
 ∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 4)))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; put together.
; case 1. s1 --> exit, when (not (strchr1 i2 n2 lst2 ch)).
```

THEOREM: strspn-s1-1p-info
strspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
→ ((*i2* < *n2*) = **t**)

THEOREM: strspn-s1-sn
let *sn* **be** stepn (*s*, strspn-t0 (*i2*, *n2*, *lst2*, *ch*))
in
(strspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (¬ strchr1 (*i2*, *n2*, *lst2*, *ch*)))
→ ((mc-status (*sn*) = 'running)
 ∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
 ∧ (read-dn (32, 0, *sn*) = sub (32, 1, *i1**))
 ∧ (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))
 ∧ (read-mem (*x*, mc-mem (*sn*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

THEOREM: strspn-s1-sn-rfile
(strspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (¬ strchr1 (*i2*, *n2*, *lst2*, *ch*))
 ∧ d2-7a2-5p (*rn*)
 ∧ (*oplen* ≤ 32))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strspn-t0 (*i2*, *n2*, *lst2*, *ch*))))
 = **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
 else head (rn-saved (*s*), *oplen*) **endif**)

EVENT: Disable strspn-s1-1p-info.

```
; case 2. s1 --> s0, when (strchr1 i2 n2 lst2 ch).
```

THEOREM: strspn-s1p-s1-1p
strspn-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
→ strspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)

THEOREM: strspn-s1-s0
let *s0* **be** stepn (*s*, strspn-t0 (*i2*, *n2*, *lst2*, *ch*))
in
(strspn-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
∧ strchr1 (*i2*, *n2*, *lst2*, *ch*))
→ (strspn-s0p (*s0*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ (read-rn (32, 14, mc-rfile (*s0*))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-a6 (*s0*) = linked-a6 (*s*))
∧ (linked-rts-addr (*s0*) = linked-rts-addr (*s*))
∧ (rn-saved (*s0*) = rn-saved (*s*))
∧ (read-mem (*x*, mc-mem (*s0*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

THEOREM: strspn-s1-s0-rfile
(strspn-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
∧ strchr1 (*i2*, *n2*, *lst2*, *ch*)
∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strspn-t0 (*i2*, *n2*, *lst2*, *ch*))))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; from s0 --> exit. s0 --> sn.
; base case:  s0 --> sn,  when (not (strchr1 i2 n2 lst2 ch)).
```

THEOREM: strspn-s0-sn-base
let *sn* **be** stepn (*s*, strspn-t1 (*n2*, *lst2*, get-nth (*i1*, *lst1*)))
in
(strspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ (¬ strchr1 (0, *n2*, *lst2*, get-nth (*i1*, *lst1*))))
→ ((mc-status (*sn*) = 'running)
∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
∧ (read-dn (32, 0, *sn*) = head (*i1**, 32))
∧ (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))
∧ (read-mem (*x*, mc-mem (*sn*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

THEOREM: strspn-s0-sn-rfile-base
let *ch* **be** get-nth (*i1*, *lst1*)
in
(strspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ (¬ strchr1 (0, *n2*, *lst2*, *ch*))
∧ (*oplen* ≤ 32)
∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strspn-t1 (*n2*, *lst2*, *ch*))))
= **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
else head (rn-saved (*s*), *oplen*) **endif**) **endlet**

```
; induction case: s0 --> s0,  when (strchr1 i2 n2 lst2 ch).
```

THEOREM: strspn-s0-s0
let *s0* **be** stepn (*s*, strspn-t1 (*n2*, *lst2*, get-nth (*i1*, *lst1*)))
in
(strspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ strchr1 (0, *n2*, *lst2*, get-nth (*i1*, *lst1*)))
→ (strspn-s0p (*s0*, add (32, *i1**, 1), 1 + *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (read-rn (32, 14, mc-rfile (*s0*))
 = read-rn (32, 14, mc-rfile (*s*)))
 ∧ (linked-a6 (*s0*) = linked-a6 (*s*))
 ∧ (linked-rts-addr (*s0*) = linked-rts-addr (*s*))
 ∧ (rn-saved (*s0*) = rn-saved (*s*))
 ∧ (read-mem (*x*, mc-mem (*s0*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

THEOREM: strspn-s0-s0-rfile
let *ch* **be** get-nth (*i1*, *lst1*)
in
(strspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ strchr1 (0, *n2*, *lst2*, *ch*)
 ∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strspn-t1 (*n2*, *lst2*, *ch*))))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*))) **endlet**

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

THEOREM: strspn-s0p-info
strspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ (((*i1* < *n1*) = **t**) ∧ (nat-to-uint (head (*i1**, 32)) = fix (*i1*)))

THEOREM: strspn-s0-sn
let *sn* **be** stepn (*s*, strspn-t2 (*i1*, *n1*, *lst1*, *n2*, *lst2*))
in
strspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ ((mc-status (*sn*) = 'running)
 ∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
 ∧ (uread-dn (32, 0, *sn*) = strspn (*i1*, *n1*, *lst1*, *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))
 ∧ (read-mem (*x*, mc-mem (*sn*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

THEOREM: strspn-s0-sn-rfile
(strspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ d2-7a2-5p (*rn*)

$\wedge$ $(oplen \leq 32)$)
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, strspn-t2 ($i1$, $n1$, $lst1$, $n2$, $lst2$))))
= **if** d3-7a2-5p (rn) **then** read-rn ($oplen$, rn, mc-rfile (s))
else head (rn-saved (s), $oplen$) **endif**)

```
; the correctness of strspn.
```

THEOREM: strspn-correctness
let sn **be** stepn (s, strspn-t ($n1$, $lst1$, $n2$, $lst2$))
in
strspn-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-an (32, 6, 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)), 20)
$\rightarrow$ (read-mem (x, mc-mem (sn), k)
= read-mem (x, mc-mem (s), k)))
$\wedge$ (uread-dn (32, 0, sn) = strspn (0, $n1$, $lst1$, $n2$, $lst2$))) **endlet**

EVENT: Disable strspn-s0p-info.

EVENT: Disable strspn-t.

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

Index

add, 3–5, 7–13

d2-7a2-5p, 8, 10–13
d3-7a2-5p, 6–13
disjoint, 4–6, 13

equal*, 5
evenp, 4, 5

get-nth, 3, 4, 7–12

head, 8, 10–13

linked-a6, 6–12
linked-rts-addr, 6–12

mc-mem, 4–13
mc-pc, 4, 5, 8, 10–13
mc-rfile, 6–13
mc-status, 4, 5, 8, 10–13
mcode-addrp, 4, 5
mem-lst, 4–6

nat-rangep, 5
nat-to-uint, 5, 6, 12

ram-addrp, 4–6
read-an, 5, 6, 8, 10–13
read-dn, 5, 8, 10, 11
read-mem, 4–13
read-rn, 6–13
read-sp, 4, 6, 13
rn-saved, 6–13
rom-addrp, 4, 5
rts-addr, 6, 13

slen, 4–6
splus, 3
stepn, 3, 4, 6–13
strchr1, 3, 4, 7, 10–12
strchr1-la, 7
strspn, 12, 13
strspn-code, 3–5
strspn-correctness, 13
strspn-induct0, 3, 4
strspn-induct1, 4
strspn-s-s0, 6
strspn-s-s0-else, 6
strspn-s-s0-mem, 6
strspn-s-s0-rfile, 6
strspn-s0-s0, 12
strspn-s0-s0-rfile, 12
strspn-s0-s1, 7
strspn-s0-s1-1, 6
strspn-s0-s1-else, 7
strspn-s0-s1-rfile, 7
strspn-s0-sn, 12
strspn-s0-sn-base, 11
strspn-s0-sn-rfile, 12
strspn-s0-sn-rfile-base, 11
strspn-s0p, 4, 6–8, 11, 12
strspn-s0p-info, 12
strspn-s1-1p, 5–10
strspn-s1-1p-info, 10
strspn-s1-s0, 11
strspn-s1-s0-base, 8
strspn-s1-s0-rfile, 11
strspn-s1-s0-rfile-base, 8
strspn-s1-s1, 9
strspn-s1-s1-1, 9
strspn-s1-s1-rfile, 10
strspn-s1-sn, 10
strspn-s1-sn-base, 8
strspn-s1-sn-rfile, 10
strspn-s1-sn-rfile-base, 8
strspn-s1p, 6–11
strspn-s1p-s1-1p, 10
strspn-statep, 4, 6, 13
strspn-t, 3, 13
strspn-t0, 3, 10, 11
strspn-t1, 3, 4, 11, 12
strspn-t2, 3, 12, 13
sub, 4–6, 8, 10, 13

uint-rangep, 4–6
uread-dn, 5, 12, 13