```
#|

 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 STRCSPN 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 strcspn function in the Berkeley string library.

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

/*
```

```
 * Stop as soon as we find any character from s2.  Note that there
 * must be a NUL in s2; it suffices to stop when we find that, too.
 */
for (p = s1;;) {
c = *p++;
spanp = s2;
do {
if ((sc = *spanp++) == c)
return (p - 1 - s1);
} while (sc != 0);
}
/* NOTREACHED */
}
```

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

```
0x2578 <strcspn>:       linkw fp,#0
0x257c <strcspn+4>:     moveml d2-d3,sp@-
0x2580 <strcspn+8>:     movel fp@(8),d0
0x2584 <strcspn+12>:    movel fp@(12),d3
0x2588 <strcspn+16>:    moveal d0,a1
0x258a <strcspn+18>:    movel d0,d2
0x258c <strcspn+20>:    addql #1,d2
0x258e <strcspn+22>:    moveb a1@+,d1
0x2590 <strcspn+24>:    moveal d3,a0
0x2592 <strcspn+26>:    moveb a0@+,d0
0x2594 <strcspn+28>:    cmpb d0,d1
0x2596 <strcspn+30>:    beq 0x259e <strcspn+38>
0x2598 <strcspn+32>:    tstb d0
0x259a <strcspn+34>:    bne 0x2592 <strcspn+26>
0x259c <strcspn+36>:    bra 0x258e <strcspn+22>
0x259e <strcspn+38>:    movel a1,d0
0x25a0 <strcspn+40>:    subl d2,d0
0x25a2 <strcspn+42>:    moveml fp@(-8),d2-d3
0x25a8 <strcspn+48>:    unlk fp
0x25aa <strcspn+50>:    rts
```

The machine code of the above program is:

```
<strcspn>:     0x4e56  0x0000  0x48e7  0x3000  0x202e  0x0008  0x262e  0x000c
<strcspn+16>:  0x2240  0x2400  0x5282  0x1219  0x2043  0x1018  0xb200  0x6706
<strcspn+32>:  0x4a00  0x66f6  0x60f0  0x2009  0x9082  0x4cee  0x000c  0xfff8
<strcspn+48>:  0x4e5e  0x4e75
```

```
'(78      86      0       0       72      231     48      0
  32      46      0       8       38      46      0       12
  34      64      36      0       82      130     18      25
  32      67      16      24      178     0       103     6
  74      0       102     246     96      240     32      9
  144     130     76      238     0       12      255     248
  78      94      78      117)
|#

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

DEFINITION:
STRCSPN-CODE
= `'(78 86 0 0 72 231 48 0 32 46 0 8 38 46 0 12 34 64 36`
`0 82 130 18 25 32 67 16 24 178 0 103 6 74 0 102 246`
`96 240 32 9 144 130 76 238 0 12 255 248 78 94 78 117)`

```
; the computatin time of the program.
```

DEFINITION:
strcspn-t0 (*i2*, *n2*, *lst2*, *ch*)
= **if** *i2* < *n2*
  then if get-nth (*i2*, *lst2*) = *ch* **then** 8
    elseif get-nth (*i2*, *lst2*) = 0 **then** 6
    else splus (5, strcspn-t0 (1 + *i2*, *n2*, *lst2*, *ch*)) **endif**
  else 0 **endif**

DEFINITION:
strcspn-t1 (*n2*, *lst2*, *ch*) = splus (2, strcspn-t0 (0, *n2*, *lst2*, *ch*))

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

DEFINITION:
strcspn-t (*n1*, *lst1*, *n2*, *lst2*) = splus (7, strcspn-t2 (0, *n1*, *lst1*, *n2*, *lst2*))

```
; two induction hints.
```

DEFINITION:

```
strcspn-induct0 (s, i2*, i2, n2, lst2, ch)
=   if i2 < n2
    then if get-nth (i2, lst2) = ch then t
         elseif get-nth (i2, lst2) = 0 then t
         else strcspn-induct0 (stepn (s, 5),
                               add (32, i2*, 1),
                               1 + i2,
                               n2,
                               lst2,
                               ch) endif
    else t endif
```

DEFINITION:

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

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

DEFINITION:

```
strcspn-statep (s, str1, n1, lst1, str2, n2, lst2)
=   ((mc-status (s) = 'running)
    ∧  evenp (mc-pc (s))
    ∧  rom-addrp (mc-pc (s), mc-mem (s), 52)
    ∧  mcode-addrp (mc-pc (s), mc-mem (s), STRCSPN-CODE)
    ∧  ram-addrp (sub (32, 12, read-sp (s)), mc-mem (s), 24)
    ∧  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, 12, read-sp (s)), 24, str1, n1)
    ∧  disjoint (sub (32, 12, read-sp (s)), 24, str2, n2)
    ∧  (str1 = read-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
```

$\wedge$ (*str2* = read-mem (add (32, read-sp (*s*), 8), mc-mem (*s*), 4))
$\wedge$ (slen (0, *n1*, *lst1*) < *n1*)
$\wedge$ (slen (0, *n2*, *lst2*) < *n2*)
$\wedge$ (*n1* $\in$ **N**)
$\wedge$ (*n2* $\in$ **N**)
$\wedge$ uint-rangep (*n1*, 32)
$\wedge$ uint-rangep (*n2*, 32))

```
; an intermediate state s0.
```

DEFINITION:
strcspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
$\wedge$ evenp (mc-pc (*s*))
$\wedge$ rom-addrp (sub (32, 22, mc-pc (*s*)), mc-mem (*s*), 52)
$\wedge$ mcode-addrp (sub (32, 22, mc-pc (*s*)), mc-mem (*s*), STRCSPN-CODE)
$\wedge$ ram-addrp (sub (32, 8, read-an (32, 6, *s*)), mc-mem (*s*), 24)
$\wedge$ 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, 8, read-an (32, 6, *s*)), 24, *str1*, *n1*)
$\wedge$ disjoint (sub (32, 8, read-an (32, 6, *s*)), 24, *str2*, *n2*)
$\wedge$ equal* (read-an (32, 1, *s*), add (32, *str1*, *i1**))
$\wedge$ (*str2* = read-dn (32, 3, *s*))
$\wedge$ (read-dn (32, 2, *s*) = add (32, *str1*, 1))
$\wedge$ (*i1** $\in$ **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* $\in$ **N**)
$\wedge$ (*n2* $\in$ **N**)
$\wedge$ uint-rangep (*n1*, 32)
$\wedge$ uint-rangep (*n2*, 32))

```
; an intermediate state s1.
```

DEFINITION:
strcspn-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, 26, mc-pc (*s*)), mc-mem (*s*), 52)
$\wedge$ mcode-addrp (sub (32, 26, mc-pc (*s*)), mc-mem (*s*), STRCSPN-CODE)
$\wedge$ ram-addrp (sub (32, 8, read-an (32, 6, *s*)), mc-mem (*s*), 24)

$\wedge$ ram-addrp (*str2*, mc-mem (*s*), *n2*)
$\wedge$ mem-lst (1, *str2*, mc-mem (*s*), *n2*, *lst2*)
$\wedge$ disjoint (sub (32, 8, read-an (32, 6, *s*)), 24, *str2*, *n2*)
$\wedge$ equal* (read-an (32, 1, *s*), add (32, *str1*, *i1**))
$\wedge$ equal* (read-an (32, 0, *s*), add (32, *str2*, *i2**))
$\wedge$ (*str2* = read-dn (32, 3, *s*))
$\wedge$ (*ch* = uread-dn (8, 1, *s*))
$\wedge$ (read-dn (32, 2, *s*) = add (32, *str1*, 1))
$\wedge$ (*i1** $\in$ **N**)
$\wedge$ nat-rangep (*i1**, 32)
$\wedge$ (*i1* = nat-to-uint (*i1**))
$\wedge$ (*i2** $\in$ **N**)
$\wedge$ nat-rangep (*i2**, 32)
$\wedge$ (*i2* = nat-to-uint (*i2**))
$\wedge$ (slen (0, *n2*, *lst2*) $<$ *n2*)
$\wedge$ (slen (*i2*, *n2*, *lst2*) $<$ *n2*)
$\wedge$ (*n2* $\in$ **N**)
$\wedge$ uint-rangep (*n2*, 32))

DEFINITION:
strcspn-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
= (strcspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
$\wedge$ ram-addrp (*str1*, mc-mem (*s*), *n1*)
$\wedge$ mem-lst (1, *str1*, mc-mem (*s*), *n1*, *lst1*)
$\wedge$ disjoint (sub (32, 8, read-an (32, 6, *s*)), 24, *str1*, *n1*)
$\wedge$ (slen (*i1*, *n1*, *lst1*) $<$ *n1*)
$\wedge$ (*n1* $\in$ **N**)
$\wedge$ uint-rangep (*n1*, 32))

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

THEOREM: strcspn-s-s0
strcspn-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\rightarrow$ strcspn-s0p (stepn (*s*, 7), 0, 0, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)

THEOREM: strcspn-s-s0-else
strcspn-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\rightarrow$ ((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*)))
$\wedge$ (movem-saved (stepn (*s*, 7), 4, 8, 2)
= readm-rn (32, '(2 3), mc-rfile (*s*))))

THEOREM: strcspn-s-s0-rfile

(strcspn-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$) $\wedge$ d4-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 7)))
$=$ read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strcspn-s-s0-mem
(strcspn-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\wedge$ disjoint (x, k, sub (32, 12, read-sp (s)), 24))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 7)), k) $=$ read-mem (x, mc-mem (s), k))

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

THEOREM: strcspn-s0-s1-1
strcspn-s0p (s, $i1*$, $i1$, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\rightarrow$ strcspn-s1-1p (stepn (s, 2),
add (32, $i1*$, 1),
$1 + i1$,
$str1$,
$n1$,
$lst1$,
0,
0,
$str2$,
$n2$,
$lst2$,
get-nth ($i1$, $lst1$))

THEOREM: strchr-la
((slen ($i2$, $n2$, $lst2$) $< n2$) $\wedge$ ($\neg$ strchr ($i2$, $n2$, $lst2$, ch))) $\rightarrow$ ($ch \neq 0$)

THEOREM: strcspn-s0-s1
(strcspn-s0p (s, $i1*$, $i1$, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\wedge$ ($\neg$ strchr (0, $n2$, $lst2$, get-nth ($i1$, $lst1$))))
$\rightarrow$ strcspn-s1p (stepn (s, 2),
add (32, $i1*$, 1),
$1 + i1$,
$str1$,
$n1$,
$lst1$,
0,
0,
$str2$,
$n2$,
$lst2$,
get-nth ($i1$, $lst1$))

THEOREM: strcspn-s0-s1-else
let *s1* **be** stepn (*s*, 2)
in
strcspn-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*))
∧ (movem-saved (*s1*, 4, 8, 2) = movem-saved (*s*, 4, 8, 2))
∧ (read-mem (*x*, mc-mem (*s1*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

THEOREM: strcspn-s0-s1-rfile
(strcspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) ∧ d4-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 2)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; loop 1.
; base case 1. from s1 to exit: s1 --> sn, when lst1[i] = lst2[j].
```

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

THEOREM: strcspn-s1-sn-rfile-base
(strcspn-s1-1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
∧ (get-nth (*i2*, *lst2*) = *ch*)
∧ (*oplen* ≤ 32)
∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 8)))
= **if** d4-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
else get-vlst (*oplen*, 0, *rn*, '(2 3), movem-saved (*s*, 4, 8, 2)) **endif**)

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

THEOREM: strcspn-s1-s0-base
(strcspn-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
∧ (get-nth (*i2*, *lst2*) ≠ *ch*)

```
 ∧   (get-nth (i2, lst2) = 0))
 →   (strcspn-s0p (stepn (s, 6), i1*, i1, str1, n1, lst1, str2, n2, lst2)
      ∧   (read-rn (32, 14, mc-rfile (stepn (s, 6)))
            =   read-rn (32, 14, mc-rfile (s)))
      ∧   (linked-a6 (stepn (s, 6)) = linked-a6 (s))
      ∧   (linked-rts-addr (stepn (s, 6)) = linked-rts-addr (s))
      ∧   (movem-saved (stepn (s, 6), 4, 8, 2) = movem-saved (s, 4, 8, 2))
      ∧   (read-mem (x, mc-mem (stepn (s, 6)), k)
            =   read-mem (x, mc-mem (s), k)))
```

THEOREM: strcspn-s1-s0-rfile-base

```
 (strcspn-s1p (s, i1*, i1, str1, n1, lst1, i2*, i2, str2, n2, lst2, ch)
  ∧   (get-nth (i2, lst2) ≠ ch)
  ∧   (get-nth (i2, lst2) = 0)
  ∧   d4-7a2-5p (rn))
  →   (read-rn (oplen, rn, mc-rfile (stepn (s, 6)))
       =   read-rn (oplen, rn, mc-rfile (s)))
```

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

THEOREM: strcspn-s1-s1-1

```
 (strcspn-s1-1p (s, i1*, i1, str1, n1, lst1, i2*, i2, str2, n2, lst2, ch)
  ∧   (get-nth (i2, lst2) ≠ ch)
  ∧   (get-nth (i2, lst2) ≠ 0))
  →   (strcspn-s1-1p (stepn (s, 5),
                      i1*,
                      i1,
                      str1,
                      n1,
                      lst1,
                      add (32, i2*, 1),
                      1 + i2,
                      str2,
                      n2,
                      lst2,
                      ch)
       ∧   (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))
       ∧   (movem-saved (stepn (s, 5), 4, 8, 2) = movem-saved (s, 4, 8, 2))
       ∧   (read-mem (x, mc-mem (stepn (s, 5)), k)
             =   read-mem (x, mc-mem (s), k)))
```

THEOREM: strcspn-s1-s1

```
(strcspn-s1p (s, i1*, i1, str1, n1, lst1, i2*, i2, str2, n2, lst2, ch)
 ∧   (get-nth (i2, lst2) ≠ ch)
 ∧   (get-nth (i2, lst2) ≠ 0))
→   strcspn-s1p (stepn (s, 5),
                 i1*,
                 i1,
                 str1,
                 n1,
                 lst1,
                 add (32, i2*, 1),
                 1 + i2,
                 str2,
                 n2,
                 lst2,
                 ch)
```

THEOREM: strcspn-s1-s1-rfile

```
(strcspn-s1-1p (s, i1*, i1, str1, n1, lst1, i2*, i2, str2, n2, lst2, ch)
 ∧   (get-nth (i2, lst2) ≠ ch)
 ∧   (get-nth (i2, lst2) ≠ 0)
 ∧   d4-7a2-5p (rn))
→   (read-rn (oplen, rn, mc-rfile (stepn (s, 5)))
      =   read-rn (oplen, rn, mc-rfile (s)))
```

```
; put together.
; case 1. s1 --> exit, when (strchr i2 n2 lst2 ch).
```

THEOREM: strcspn-s1-1p-info

```
strcspn-s1-1p (s, i1*, i1, str1, n1, lst1, i2*, i2, str2, n2, lst2, ch)
→   ((i2 < n2) = t)
```

THEOREM: strcspn-s1-sn

```
let sn be stepn (s, strcspn-t0 (i2, n2, lst2, ch))
in
(strcspn-s1-1p (s, i1*, i1, str1, n1, lst1, i2*, i2, str2, n2, lst2, ch)
 ∧   strchr (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: strcspn-s1-sn-rfile

(strcspn-s1-1p (s, $i1*$, $i1$, $str1$, $n1$, $lst1$, $i2*$, $i2$, $str2$, $n2$, $lst2$, ch)
 ∧ strchr ($i2$, $n2$, $lst2$, ch)
 ∧ d2-7a2-5p (rn)
 ∧ ($oplen$ ≤ 32))
→ (read-rn ($oplen$, rn, mc-rfile (stepn (s, strcspn-t0 ($i2$, $n2$, $lst2$, ch))))
 = **if** d4-7a2-5p (rn) **then** read-rn ($oplen$, rn, mc-rfile (s))
 else get-vlst ($oplen$, 0, rn, '(2 3), movem-saved (s, 4, 8, 2)) **endif**)

```
; case 2. s1 --> s0, when (not (strchr i2 n2 lst2 ch)).
```

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

THEOREM: strcspn-s1-s0
let $s0$ **be** stepn (s, strcspn-t0 ($i2$, $n2$, $lst2$, ch))
in
(strcspn-s1p (s, $i1*$, $i1$, $str1$, $n1$, $lst1$, $i2*$, $i2$, $str2$, $n2$, $lst2$, ch)
 ∧ (¬ strchr ($i2$, $n2$, $lst2$, ch)))
→ (strcspn-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))
 ∧ (movem-saved ($s0$, 4, 8, 2) = movem-saved (s, 4, 8, 2))
 ∧ (read-mem (x, mc-mem ($s0$), k) = read-mem (x, mc-mem (s), k))) **endlet**

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

THEOREM: strcspn-s1-s0-rfile
(strcspn-s1p (s, $i1*$, $i1$, $str1$, $n1$, $lst1$, $i2*$, $i2$, $str2$, $n2$, $lst2$, ch)
 ∧ (¬ strchr ($i2$, $n2$, $lst2$, ch))
 ∧ d4-7a2-5p (rn))
→ (read-rn ($oplen$, rn, mc-rfile (stepn (s, strcspn-t0 ($i2$, $n2$, $lst2$, ch))))
 = read-rn ($oplen$, rn, mc-rfile (s)))

```
; from s0 --> exit. s0 --> sn.
; base case:  s0 --> sn, when (strchr i2 n2 lst2 ch).
```

THEOREM: strcspn-s0-sn-base
let sn **be** stepn (s, strcspn-t1 ($n2$, $lst2$, get-nth ($i1$, $lst1$)))
in
(strcspn-s0p (s, $i1*$, $i1$, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
 ∧ strchr (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: strcspn-s0-sn-rfile-base

```
let ch be get-nth (i1, lst1)
in
(strcspn-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2)
 ∧   strchr (0, n2, lst2, ch)
 ∧   (oplen ≤ 32)
 ∧   d2-7a2-5p (rn))
→    (read-rn (oplen, rn, mc-rfile (stepn (s, strcspn-t1 (n2, lst2, ch))))
      =   if d4-7a2-5p (rn) then read-rn (oplen, rn, mc-rfile (s))
          else get-vlst (oplen,
                         0,
                         rn,
                         '(2 3),
                         movem-saved (s, 4, 8, 2)) endif) endlet
```

```
; induction case: s0 --> s0, when (not (strchr i2 n2 lst2 ch)).
```

THEOREM: strcspn-s0-s0

```
let s0 be stepn (s, strcspn-t1 (n2, lst2, get-nth (i1, lst1)))
in
(strcspn-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2)
 ∧   (¬ strchr (0, n2, lst2, get-nth (i1, lst1))))
→    (strcspn-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))
    ∧   (movem-saved (s0, 4, 8, 2) = movem-saved (s, 4, 8, 2))
    ∧   (read-mem (x, mc-mem (s0), k) = read-mem (x, mc-mem (s), k))) endlet
```

THEOREM: strcspn-s0-s0-rfile
let *ch* **be** get-nth (*i1*, *lst1*)
in
(strcspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (¬ strchr (0, *n2*, *lst2*, *ch*))
 ∧ d4-7a2-5p (*rn*))
 → (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strcspn-t1 (*n2*, *lst2*, *ch*))))
  = read-rn (*oplen*, *rn*, mc-rfile (*s*))) **endlet**

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

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

THEOREM: strcspn-s0-sn
let *sn* **be** stepn (*s*, strcspn-t2 (*i1*, *n1*, *lst1*, *n2*, *lst2*))
in
strcspn-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*) = strcspn (*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: strcspn-s0-sn-rfile
(strcspn-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ d2-7a2-5p (*rn*)
 ∧ (*oplen* ≤ 32))
 → (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strcspn-t2 (*i1*, *n1*, *lst1*, *n2*, *lst2*))))
  = **if** d4-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
   else get-vlst (*oplen*, 0, *rn*, '(2 3), movem-saved (*s*, 4, 8, 2)) **endif**)

EVENT: Disable strcspn-s0p-info.

```
; now, finally, the correctness of strcspn.
```

THEOREM: strcspn-correctness
let *sn* **be** stepn (*s*, strcspn-t (*n1*, *lst1*, *n2*, *lst2*))
in
strcspn-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 → ((mc-status (*sn*) = 'running)
  ∧ (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, 12, read-sp (s)), 24)
  $\rightarrow$ (read-mem (x, mc-mem (sn), k)
    $=$ read-mem (x, mc-mem (s), k)))
$\wedge$ (uread-dn (32, 0, sn) $=$ strcspn (0, *n1*, *lst1*, *n2*, *lst2*))) **endlet**

EVENT: Disable strcspn-t.

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

Index

add, 4–10, 12–14

d2-7a2-5p, 8, 11–14
d4-7a2-5p, 7–13
disjoint, 4–7, 14

equal*, 5, 6
evenp, 4, 5

get-nth, 3, 4, 7–13
get-vlst, 8, 11–13

head, 12, 13

linked-a6, 6, 8–13
linked-rts-addr, 6, 8–13

mc-mem, 4–14
mc-pc, 4, 5, 8, 10, 12, 13
mc-rfile, 6–14
mc-status, 4, 5, 8, 10, 11, 13
mcode-addrp, 4, 5
mem-lst, 4–6
movem-saved, 6, 8, 9, 11–13

nat-rangep, 5, 6
nat-to-uint, 5, 6, 13

ram-addrp, 4–6
read-an, 5, 6, 8, 10, 12–14
read-dn, 5, 6, 8, 10, 12
read-mem, 4, 5, 7–14
read-rn, 6–14
read-sp, 4–7, 14
readm-rn, 6
rom-addrp, 4, 5
rts-addr, 6, 13

slen, 5–7
splus, 3
stepn, 4, 6–13
strchr, 3, 4, 7, 10–13
strchr-la, 7
strcspn, 13, 14
strcspn-code, 3–5
strcspn-correctness, 13
strcspn-induct0, 4
strcspn-induct1, 4
strcspn-s-s0, 6
strcspn-s-s0-else, 6
strcspn-s-s0-mem, 7
strcspn-s-s0-rfile, 7
strcspn-s0-s0, 12
strcspn-s0-s0-rfile, 13
strcspn-s0-s1, 7
strcspn-s0-s1-1, 7
strcspn-s0-s1-else, 8
strcspn-s0-s1-rfile, 8
strcspn-s0-sn, 13
strcspn-s0-sn-base, 11
strcspn-s0-sn-rfile, 13
strcspn-s0-sn-rfile-base, 12
strcspn-s0p, 5–9, 11–13
strcspn-s0p-info, 13
strcspn-s1-1p, 5–11
strcspn-s1-1p-info, 10
strcspn-s1-s0, 11
strcspn-s1-s0-base, 8
strcspn-s1-s0-rfile, 11
strcspn-s1-s0-rfile-base, 9
strcspn-s1-s1, 10
strcspn-s1-s1-1, 9
strcspn-s1-s1-rfile, 10
strcspn-s1-sn, 10
strcspn-s1-sn-base, 8
strcspn-s1-sn-rfile, 11
strcspn-s1-sn-rfile-base, 8
strcspn-s1p, 6–11
strcspn-s1p-s1-1p, 11
strcspn-statep, 4, 6, 7, 13
strcspn-t, 3, 13
strcspn-t0, 3, 10, 11
strcspn-t1, 3, 4, 11–13

strcspn-t2, 3, 13
sub, 4–8, 10, 14

uint-rangep, 5, 6
uread-dn, 6, 13, 14