```
#|

 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 STRPBRK 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 strpbrk function in the Berkeley string library.

char *
strpbrk(s1, s2)
register const char *s1, *s2;
{
register const char *scanp;
register int c, sc;

while ((c = *s1++) != 0) {
for (scanp = s2; (sc = *scanp++) != 0;)
```

```
if (sc == c)
return ((char *)(s1 - 1));
}
return (NULL);
}
```

Remark. Should the local variables c and sc have type register char?

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

```
0x2688 <strpbrk>:       linkw fp,#0
0x268c <strpbrk+4>:     moveml d2-d3,sp@-
0x2690 <strpbrk+8>:     moveal fp@(8),a1
0x2694 <strpbrk+12>:    movel fp@(12),d3
0x2698 <strpbrk+16>:    bra 0x26b0 <strpbrk+40>
0x269a <strpbrk+18>:    moveal d3,a0
0x269c <strpbrk+20>:    movel a1,d1
0x269e <strpbrk+22>:    subl #1,d1
0x26a0 <strpbrk+24>:    bra 0x26aa <strpbrk+34>
0x26a2 <strpbrk+26>:    cmpl d0,d2
0x26a4 <strpbrk+28>:    bne 0x26aa <strpbrk+34>
0x26a6 <strpbrk+30>:    movel d1,d0
0x26a8 <strpbrk+32>:    bra 0x26b8 <strpbrk+48>
0x26aa <strpbrk+34>:    moveb a0@+,d0
0x26ac <strpbrk+36>:    extbl d0
0x26ae <strpbrk+38>:    bne 0x26a2 <strpbrk+26>
0x26b0 <strpbrk+40>:    moveb a1@+,d2
0x26b2 <strpbrk+42>:    extbl d2
0x26b4 <strpbrk+44>:    bne 0x269a <strpbrk+18>
0x26b6 <strpbrk+46>:    clrl d0
0x26b8 <strpbrk+48>:    moveml fp@(-8),d2-d3
0x26be <strpbrk+54>:    unlk fp
0x26c0 <strpbrk+56>:    rts
```

The machine code of the above program is:

```
<strpbrk>:     0x4e56  0x0000  0x48e7  0x3000  0x226e  0x0008  0x262e  0x000c
<strpbrk+16>:  0x6016  0x2043  0x2209  0x5381  0x6008  0xb480  0x6604  0x2001
<strpbrk+32>:  0x600e  0x1018  0x49c0  0x66f2  0x1419  0x49c2  0x66e4  0x4280
<strpbrk+48>:  0x4cee  0x000c  0xfff8  0x4e5e  0x4e75

'(78      86      0       0       72      231     48      0
  34      110     0       8       38      46      0       12
```

```
  96      22      32      67      34      9       83      129
  96      8       180     128     102     4       32      1
  96      14      16      24      73      192     102     242
  20      25      73      194     102     228     66      128
  76      238     0       12      255     248     78      94
  78      117)
|#

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

DEFINITION:
STRPBRK-CODE
= `'(78 86 0 0 72 231 48 0 34 110 0 8 38 46 0 12 96 22 32 67 34 9 83 129 96 8 180 128 102 4 32 1 96 14 16 24 73 192 102 242 20 25 73 194 102 228 66 128 76 238 0 12 255 248 78 94 78 117)`

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

DEFINITION:
strpbrk-t0 (*i2*, *n2*, *lst2*, *ch*)
= **if** *i2* < *n2*
  then if get-nth (*i2*, *lst2*) = `0` **then** `3`
    elseif get-nth (*i2*, *lst2*) = *ch* **then** `10`
    else splus (`5`, strpbrk-t0 (1 + *i2*, *n2*, *lst2*, *ch*)) **endif**
  else `0` **endif**

DEFINITION:
strpbrk-t1 (*n2*, *lst2*, *ch*) = splus (`7`, strpbrk-t0 (`0`, *n2*, *lst2*, *ch*))

DEFINITION:
strpbrk-t2 (*i1*, *n1*, *lst1*, *n2*, *lst2*)
= **if** *i1* < *n1*
  then if get-nth (*i1*, *lst1*) = `0` **then** `7`
    elseif strchr1 (`0`, *n2*, *lst2*, get-nth (*i1*, *lst1*))
    then strpbrk-t1 (*n2*, *lst2*, get-nth (*i1*, *lst1*))
    else splus (strpbrk-t1 (*n2*, *lst2*, get-nth (*i1*, *lst1*)),
                strpbrk-t2 (1 + *i1*, *n1*, *lst1*, *n2*, *lst2*)) **endif**
  else `0` **endif**

DEFINITION:
strpbrk-t (*n1*, *lst1*, *n2*, *lst2*) = splus (`5`, strpbrk-t2 (`0`, *n1*, *lst1*, *n2*, *lst2*))

```
; two induction hints.
```

```
DEFINITION:
strpbrk-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 strpbrk-induct0 (stepn (s, 5),
                              add (32, i2*, 1),
                              1 + i2,
                              n2,
                              lst2,
                              ch) endif
   else t endif

DEFINITION:
strpbrk-induct1 (s, i1*, i1, n1, lst1, n2, lst2)
=  if i1 < n1
   then if (get-nth (i1, lst1) = 0)
           ∨  strchr1 (0, n2, lst2, get-nth (i1, lst1)) then t
        else strpbrk-induct1 (stepn (s,
                                     strpbrk-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:
strpbrk-statep (s, str1, n1, lst1, str2, n2, lst2)
=  ((mc-status (s) = 'running)
    ∧  evenp (mc-pc (s))
    ∧  rom-addrp (mc-pc (s), mc-mem (s), 58)
    ∧  mcode-addrp (mc-pc (s), mc-mem (s), STRPBRK-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)
```

$\wedge$ (*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)
$\wedge$ (nat-to-uint (*str1*) $\neq$ 0)
$\wedge$ uint-rangep (nat-to-uint (*str1*) + *n1*, 32))

DEFINITION:
strpbrk-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
$\wedge$ evenp (mc-pc (*s*))
$\wedge$ rom-addrp (sub (32, 40, mc-pc (*s*)), mc-mem (*s*), 58)
$\wedge$ mcode-addrp (sub (32, 40, mc-pc (*s*)), mc-mem (*s*), STRPBRK-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*))
$\wedge$ (*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))

DEFINITION:
strpbrk-s1p (*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, 34, mc-pc (*s*)), mc-mem (*s*), 58)
$\wedge$ mcode-addrp (sub (32, 34, mc-pc (*s*)), mc-mem (*s*), STRPBRK-CODE)
$\wedge$ ram-addrp (sub (32, 8, read-an (32, 6, *s*)), mc-mem (*s*), 24)
$\wedge$ 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-an (32, 6, *s*)), 24, *str1*, *n1*)
∧ disjoint (sub (32, 8, read-an (32, 6, *s*)), 24, *str2*, *n2*)
∧ equal* (read-an (32, 1, *s*), add (32, *str1*, *i1**))
∧ equal* (read-an (32, 0, *s*), add (32, *str2*, *i2**))
∧ equal* (read-dn (32, 1, *s*), sub (32, 1, read-an (32, 1, *s*)))
∧ equal* (read-dn (32, 2, *s*), ext (8, read-dn (8, 2, *s*), 32))
∧ (*str2* = read-dn (32, 3, *s*))
∧ (*ch* = uread-dn (8, 2, *s*))
∧ (slen (*i1*, *n1*, *lst1*) < *n1*)
∧ (slen (0, *n2*, *lst2*) < *n2*)
∧ (slen (*i2*, *n2*, *lst2*) < *n2*)
∧ (*i1** ∈ **N**)
∧ nat-rangep (*i1**, 32)
∧ (*i1* = nat-to-uint (*i1**))
∧ (*i2** ∈ **N**)
∧ nat-rangep (*i2**, 32)
∧ (*i2* = nat-to-uint (*i2**))
∧ (*n1* ∈ **N**)
∧ uint-rangep (*n1*, 32)
∧ (*n2* ∈ **N**)
∧ uint-rangep (*n2*, 32))

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

THEOREM: strpbrk-s-s0
strpbrk-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ strpbrk-s0p (stepn (*s*, 5), 0, 0, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)

THEOREM: strpbrk-s-s0-else
strpbrk-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ ((linked-rts-addr (stepn (*s*, 5)) = rts-addr (*s*))
∧ (linked-a6 (stepn (*s*, 5)) = read-an (32, 6, *s*))
∧ (read-an (32, 6, stepn (*s*, 5)) = sub (32, 4, read-sp (*s*)))
∧ (movem-saved (stepn (*s*, 5), 4, 8, 2)
= readm-rn (32, '(2 3), mc-rfile (*s*))))

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

THEOREM: strpbrk-s-s0-mem

(strpbrk-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ disjoint (x, k, sub (32, 12, read-sp (s)), 24))
→ (read-mem (x, mc-mem (stepn (s, 5)), k) = read-mem (x, mc-mem (s), k))

```
; from s0 to exit.
; base case.  s0 --> sn, when lst1[i1] = 0.
```

THEOREM: strpbrk-s0-sn-base1
(strpbrk-s0p (s, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (*i1*, *lst1*) = 0))
→ ((mc-status (stepn (s, 7)) = 'running)
 ∧ (mc-pc (stepn (s, 7)) = linked-rts-addr (s))
 ∧ (read-dn (32, 0, stepn (s, 7)) = 0)
 ∧ (read-an (32, 6, stepn (s, 7)) = linked-a6 (s))
 ∧ (read-an (32, 7, stepn (s, 7)) = add (32, read-an (32, 6, s), 8))
 ∧ (read-mem (x, mc-mem (stepn (s, 7)), k)
 = read-mem (x, mc-mem (s), k)))

THEOREM: strpbrk-s0-sn-rfile-base1
(strpbrk-s0p (s, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (*i1*, *lst1*) = 0)
 ∧ (*oplen* ≤ 32)
 ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 7)))
 = **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**)

```
; induction case. s0 --> s0, when lst1[i1] =\= 0.
; from s0 to s1: s0 --> s1, when lst1[i1] =\= 0.
```

THEOREM: strpbrk-s0-s1
(strpbrk-s0p (s, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (*i1*, *lst1*) ≠ 0))
→ strpbrk-s1p (stepn (s, 7),
 add (32, *i1**, 1),
 1 + *i1*,
 str1,
 n1,
 lst1,
 0,
 0,
 str2,
 n2,
 lst2,
 get-nth (*i1*, *lst1*))

THEOREM: strpbrk-s0-s1-else
(strpbrk-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (*i1*, *lst1*) ≠ 0))
→ ((read-an (32, 6, stepn (*s*, 7)) = read-an (32, 6, *s*))
 ∧ (linked-a6 (stepn (*s*, 7)) = linked-a6 (*s*))
 ∧ (linked-rts-addr (stepn (*s*, 7)) = linked-rts-addr (*s*))
 ∧ (movem-saved (stepn (*s*, 7), 4, 8, 2) = movem-saved (*s*, 4, 8, 2))
 ∧ (read-mem (*x*, mc-mem (stepn (*s*, 7)), *k*)
 = read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strpbrk-s0-s1-rfile
(strpbrk-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (*i1*, *lst1*) ≠ 0)
 ∧ d4-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 7)))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; from s1 to s0:
; base case. s1 --> s0, when lst2[i2] = 0.
```

THEOREM: strpbrk-s1-s0-base
(strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (get-nth (*i2*, *lst2*) = 0))
→ (strpbrk-s0p (stepn (*s*, 3), *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (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*))
 ∧ (movem-saved (stepn (*s*, 3), 4, 8, 2) = movem-saved (*s*, 4, 8, 2))
 ∧ (read-mem (*x*, mc-mem (stepn (*s*, 3)), *k*)
 = read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strpbrk-s1-s0-rfile-base
(strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (get-nth (*i2*, *lst2*) = 0)
 ∧ d4-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 3)))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; induction case. s1 --> s1, when lst2[i2] =\= 0 and lst2[i2] =\= ch.
```

THEOREM: strpbrk-s1-s1
(strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (get-nth (*i2*, *lst2*) ≠ 0)
 ∧ (get-nth (*i2*, *lst2*) ≠ *ch*))

→ (strpbrk-s1p (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: strpbrk-s1-s1-rfile
(strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
∧ (get-nth (*i2*, *lst2*) ≠ 0)
∧ (get-nth (*i2*, *lst2*) ≠ *ch*)
∧ d4-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 5)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

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

THEOREM: strpbrk-s1p-info
strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
→ ((*i2* < *n2*) = **t**)

THEOREM: strpbrk-s1-s0
let *s0* **be** stepn (*s*, strpbrk-t0 (*i2*, *n2*, *lst2*, *ch*))
in
(strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
∧ (¬ strchr1 (*i2*, *n2*, *lst2*, *ch*)))
→ (strpbrk-s0p (*s0*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ (read-an (32, 6, *s0*) = read-an (32, 6, *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: strpbrk-s1-s0-rfile
(strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (¬ strchr1 (*i2*, *n2*, *lst2*, *ch*))
 ∧ d4-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strpbrk-t0 (*i2*, *n2*, *lst2*, *ch*))))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; from s1 to exit:
; base case. s1 --> sn, when lst2[i2] =\= 0 and lst2[i2] = ch.
```

THEOREM: strpbrk-s1-sn-base
(strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (get-nth (*i2*, *lst2*) ≠ 0)
 ∧ (get-nth (*i2*, *lst2*) = *ch*))
→ ((mc-status (stepn (*s*, 10)) = 'running)
 ∧ (mc-pc (stepn (*s*, 10)) = linked-rts-addr (*s*))
 ∧ (read-dn (32, 0, stepn (*s*, 10)) = add (32, *str1*, sub (32, 1, *i1**)))
 ∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 10))) = linked-a6 (*s*))
 ∧ (read-rn (32, 15, mc-rfile (stepn (*s*, 10)))
 = add (32, read-an (32, 6, *s*), 8))
 ∧ (read-mem (*x*, mc-mem (stepn (*s*, 10)), *k*)
 = read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strpbrk-s1-sn-rfile-base
(strpbrk-s1p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *i2**, *i2*, *str2*, *n2*, *lst2*, *ch*)
 ∧ (get-nth (*i2*, *lst2*) ≠ 0)
 ∧ (get-nth (*i2*, *lst2*) = *ch*)
 ∧ (*oplen* ≤ 32)
 ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 10)))
 = **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**)

```
; put together.  s1 --> sn.
```

THEOREM: strpbrk-s1-sn
let *sn* **be** stepn (*s*, strpbrk-t0 (*i2*, *n2*, *lst2*, *ch*))
in
(strpbrk-s1p (*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*) = add (32, *str1*, sub (32, 1, *i1**)))
 ∧ (read-an (32, 6, *sn*) = linked-a6 (*s*))
 ∧ (read-an (32, 7, *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: strpbrk-s1-sn-rfile
(strpbrk-s1p (*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*, strpbrk-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**)

```
; base case 2. from s0 --> sn.
```

THEOREM: strpbrk-s0-sn-base2
let *ch* **be** get-nth (*i1*, *lst1*),
 sn **be** stepn (*s*, strpbrk-t1 (*n2*, *lst2*, get-nth (*i1*, *lst1*)))
in
(strpbrk-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (*i1*, *lst1*) ≠ 0)
 ∧ strchr1 (0, *n2*, *lst2*, *ch*))
→ ((mc-status (*sn*) = 'running)
 ∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
 ∧ (read-dn (32, 0, *sn*) = add (32, *str1*, *i1**))
 ∧ (read-an (32, 6, *sn*) = linked-a6 (*s*))
 ∧ (read-an (32, 7, *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: strpbrk-s0-sn-rfile-base2
let *ch* **be** get-nth (*i1*, *lst1*)
in
(strpbrk-s0p (*s*, *i1**, *i1*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (*i1*, *lst1*) ≠ 0)
 ∧ strchr1 (0, *n2*, *lst2*, *ch*)
 ∧ (*oplen* ≤ 32)
 ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strpbrk-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.  from s0 --> s0.
```

THEOREM: strpbrk-s0-s0
let *ch* **be** get-nth (*i1*, *lst1*),

```
    s0  be  stepn (s, strpbrk-t1 (n2, lst2, get-nth (i1, lst1)))
in
(strpbrk-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2)
 ∧   (get-nth (i1, lst1) ≠ 0)
 ∧   (¬ strchr1 (0, n2, lst2, ch)))
→   (strpbrk-s0p (s0,
                  add (32, i1*, 1),
                  1 + i1,
                  str1,
                  n1,
                  lst1,
                  str2,
                  n2,
                  lst2)
      ∧   (read-an (32, 6, s0) = read-an (32, 6, 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: strpbrk-s0-s0-rfile

```
let ch  be  get-nth (i1, lst1)
in
(strpbrk-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2)
 ∧   (get-nth (i1, lst1) ≠ 0)
 ∧   (¬ strchr1 (0, n2, lst2, ch))
 ∧   d4-7a2-5p (rn))
→   (read-rn (oplen, rn, mc-rfile (stepn (s, strpbrk-t1 (n2, lst2, ch))))
      =   read-rn (oplen, rn, mc-rfile (s))) endlet
```

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

THEOREM: strpbrk-s0p-info

strpbrk-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2) → ((i1 < n1) = **t**)

THEOREM: strpbrk-s0-sn

```
let sn  be  stepn (s, strpbrk-t2 (i1, n1, lst1, n2, lst2))
in
strpbrk-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2)
→   ((mc-status (sn) = 'running)
      ∧   (mc-pc (sn) = linked-rts-addr (s))
      ∧   (read-dn (32, 0, sn)
            =   if strpbrk (i1, n1, lst1, n2, lst2)
                then add (32,
                          str1,
```

```
                    strpbrk* (i1*, i1, n1, lst1, n2, lst2))
            else 0 endif)
    ∧  (read-an (32, 6, sn) = linked-a6 (s))
    ∧  (read-an (32, 7, 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: strpbrk-s0-sn-rfile
let sn be stepn (s, strpbrk-t2 (i1, n1, lst1, n2, lst2))
in
(strpbrk-s0p (s, i1*, i1, str1, n1, lst1, str2, n2, lst2)
 ∧  d2-7a2-5p (rn)
 ∧  (oplen ≤ 32))
→  (read-rn (oplen, rn, mc-rfile (sn))
    =  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

; the correctness of strpbrk.

THEOREM: strpbrk-correctness
let sn be stepn (s, strpbrk-t (n1, lst1, n2, lst2))
in
strpbrk-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) ∧ (oplen ≤ 32))
        →  (read-rn (oplen, rn, mc-rfile (sn))
            =  read-rn (oplen, rn, mc-rfile (s))))
    ∧  (disjoint (x, k, sub (32, 12, read-sp (s)), 24)
        →  (read-mem (x, mc-mem (sn), k)
            =  read-mem (x, mc-mem (s), k)))
    ∧  (read-dn (32, 0, sn)
        =  if strpbrk (0, n1, lst1, n2, lst2)
           then add (32,
                     str1,
                     strpbrk* (0, 0, n1, lst1, n2, lst2))
           else 0 endif)) endlet

EVENT: Disable strpbrk-t.
```

```
; strpbrk* --> strpbrk.
```

THEOREM: strpbrk*-strpbrk
(strpbrk (*i1*, *n1*, *lst1*, *n2*, *lst2*)
 ∧ (*i1* = nat-to-uint (*i1**))
 ∧ nat-rangep (*i1**, 32)
 ∧ uint-rangep (*n1*, 32))
→ (nat-to-uint (strpbrk* (*i1**, *i1*, *n1*, *lst1*, *n2*, *lst2*))
 = strpbrk (*i1*, *n1*, *lst1*, *n2*, *lst2*))

THEOREM: strpbrk-non-zerop-la
let *sn* **be** stepn (*s*, strpbrk-t (*n1*, *lst1*, *n2*, *lst2*))
in
(strpbrk-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ nat-rangep (*str1*, 32)
 ∧ (nat-to-uint (*str1*) ≠ 0)
 ∧ uint-rangep (nat-to-uint (*str1*) + *n1*, 32)
 ∧ strpbrk (0, *n1*, *lst1*, *n2*, *lst2*))
→ (nat-to-uint (read-dn (32, 0, *sn*)) ≠ 0) **endlet**

THEOREM: strpbrk-non-zerop
let *sn* **be** stepn (*s*, strpbrk-t (*n1*, *lst1*, *n2*, *lst2*))
in
(strpbrk-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ strpbrk (0, *n1*, *lst1*, *n2*, *lst2*))
→ (nat-to-uint (read-dn (32, 0, *sn*)) ≠ 0) **endlet**

EVENT: Disable strpbrk*.

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

Index