```
#|

 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 STRCHR 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 strchr function in the Berkeley string library.

char *
strchr(p, ch)
        register char *p, ch;
{
        for (;; ++p) {
                if (*p == ch)
                        return(p);
                if (!*p)
                        return((char *)NULL);
```

```
        }
        /* NOTREACHED */
}
```

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

```
0x2500 <strchr>:        linkw fp,#0
0x2504 <strchr+4>:      moveal fp@(8),a0
0x2508 <strchr+8>:      moveb fp@(15),d0
0x250c <strchr+12>:     cmpb a0@,d0
0x250e <strchr+14>:     bne 0x2514 <strchr+20>
0x2510 <strchr+16>:     movel a0,d0
0x2512 <strchr+18>:     bra 0x2520 <strchr+32>
0x2514 <strchr+20>:     tstb a0@
0x2516 <strchr+22>:     bne 0x251c <strchr+28>
0x2518 <strchr+24>:     clrl d0
0x251a <strchr+26>:     bra 0x2520 <strchr+32>
0x251c <strchr+28>:     addqw #1,a0
0x251e <strchr+30>:     bra 0x250c <strchr+12>
0x2520 <strchr+32>:     unlk fp
0x2522 <strchr+34>:     rts
```

The machine code of the above program is:

```
<strchr>:      0x4e56  0x0000  0x206e  0x0008  0x102e  0x000f  0xb010  0x6604
<strchr+16>:   0x2008  0x600c  0x4a10  0x6604  0x4280  0x6004  0x5248  0x60ec
<strchr+32>:   0x4e5e  0x4e75

'(78      86      0       0       32      110     0       8
  16      46      0       15      176     16      102     4
  32      8       96      12      74      16      102     4
  66      128     96      4       82      72      96      236
  78      94      78      117)
|#
```

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

DEFINITION:
STRCHR-CODE
= '(78 86 0 0 32 110 0 8 16 46 0 15 176 16 102 4 32 8 96
12 74 16 102 4 66 128 96 4 82 72 96 236 78 94 78 117)

; the computation time of the program.

DEFINITION:
strchr1-t (i, n, lst, ch)
= **if** $i < n$
  then if get-nth (i, lst) = ch **then** 6
    elseif get-nth (i, lst) = 0 **then** 8
    else splus (6, strchr1-t ($1 + i$, n, lst, ch)) **endif**
  else 0 **endif**

DEFINITION:
strchr-t (n, lst, ch) = splus (3, strchr1-t (0, n, lst, ch))

```
; an induction hint.
```

DEFINITION:
strchr-induct (s, i^*, i, n, lst, ch)
= **if** $i < n$
  then if get-nth (i, lst) = ch **then t**
    elseif get-nth (i, lst) = 0 **then** 8
    else strchr-induct (stepn (s, 6),
                add (32, i^*, 1),
                $1 + i$,
                n,
                lst,
                ch) **endif**
  else t endif

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

DEFINITION:
strchr-statep (s, str, n, lst, ch)
= ((mc-status (s) = 'running)
  $\wedge$ evenp (mc-pc (s))
  $\wedge$ rom-addrp (mc-pc (s), mc-mem (s), 38)
  $\wedge$ mcode-addrp (mc-pc (s), mc-mem (s), STRCHR-CODE)
  $\wedge$ ram-addrp (sub (32, 4, read-sp (s)), mc-mem (s), 16)
  $\wedge$ ram-addrp (str, mc-mem (s), n)
  $\wedge$ mem-lst (1, str, mc-mem (s), n, lst)
  $\wedge$ disjoint (sub (32, 4, read-sp (s)), 16, str, n)
  $\wedge$ (str = read-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
  $\wedge$ (ch = uread-mem (add (32, read-sp (s), 11), mc-mem (s), 1))
  $\wedge$ (slen (0, n, lst) $< n$)
  $\wedge$ ($n \in \mathbf{N}$)
  $\wedge$ (nat-to-uint (str) $\neq$ 0)
  $\wedge$ uint-rangep (nat-to-uint (str) + n, 32))

```
; an intermediate state.
```

DEFINITION:
strchr-s0p (s, $i*$, i, str, n, lst, ch)
= ((mc-status (s) = 'running)
∧ evenp (mc-pc (s))
∧ rom-addrp (sub (32, 12, mc-pc (s)), mc-mem (s), 38)
∧ mcode-addrp (sub (32, 12, mc-pc (s)), mc-mem (s), STRCHR-CODE)
∧ ram-addrp (read-an (32, 6, s), mc-mem (s), 16)
∧ ram-addrp (str, mc-mem (s), n)
∧ mem-lst (1, str, mc-mem (s), n, lst)
∧ disjoint (read-an (32, 6, s), 16, str, n)
∧ equal* (read-an (32, 0, s), add (32, str, $i*$))
∧ (ch = nat-to-uint (read-dn (8, 0, s)))
∧ (slen (i, n, lst) < n)
∧ (i < n)
∧ ($i*$ ∈ **N**)
∧ nat-rangep ($i*$, 32)
∧ (i = nat-to-uint ($i*$))
∧ (n ∈ **N**)
∧ uint-rangep (n, 32))

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

THEOREM: strchr-s-s0
strchr-statep (s, str, n, lst, ch) → strchr-s0p (stepn (s, 3), 0, 0, str, n, lst, ch)

THEOREM: strchr-s-s0-else
strchr-statep (s, str, n, lst, ch)
→ ((linked-rts-addr (stepn (s, 3)) = rts-addr (s))
∧ (linked-a6 (stepn (s, 3)) = read-an (32, 6, s))
∧ (read-rn (32, 14, mc-rfile (stepn (s, 3)))
= sub (32, 4, read-sp (s))))

THEOREM: strchr-s-s0-rfile
(strchr-statep (s, str, n, lst, ch) ∧ d2-7a2-5p (rn))
→ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 3)))
= read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strchr-s-s0-mem
(strchr-statep (s, str, n, lst, ch) ∧ disjoint (x, k, sub (32, 4, read-sp (s)), 16))
→ (read-mem (x, mc-mem (stepn (s, 3)), k) = read-mem (x, mc-mem (s), k))

```
; from s0 to exit: s0 --> sn.
; base case 1. s0 --> sn, when lst[i] = ch.
```

THEOREM: strchr-s0-sn-base1
(strchr-s0p (s, $i*$, i, str, n, lst, ch) $\wedge$ (get-nth (i, lst) = ch))
$\rightarrow$ ((mc-status (stepn (s, 6)) = 'running)
$\wedge$ (mc-pc (stepn (s, 6)) = linked-rts-addr (s))
$\wedge$ (read-dn (32, 0, stepn (s, 6)) = add (32, str, $i*$))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 6))) = linked-a6 (s))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 6)))
= add (32, read-an (32, 6, s), 8))
$\wedge$ (read-mem (x, mc-mem (stepn (s, 6)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: strchr-s0-sn-rfile-base1
(strchr-s0p (s, $i*$, i, str, n, lst, ch)
$\wedge$ (get-nth (i, lst) = ch)
$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 6)))
= read-rn ($oplen$, rn, mc-rfile (s)))

```
; base case 2: s0 --> sn, when lst[i] =\= ch and lst[i] = 0.
```

THEOREM: strchr-s0-sn-base2
(strchr-s0p (s, $i*$, i, str, n, lst, ch)
$\wedge$ (get-nth (i, lst) $\neq$ ch)
$\wedge$ (get-nth (i, lst) = 0))
$\rightarrow$ ((mc-status (stepn (s, 8)) = 'running)
$\wedge$ (mc-pc (stepn (s, 8)) = linked-rts-addr (s))
$\wedge$ (read-dn (32, 0, stepn (s, 8)) = 0)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 8))) = linked-a6 (s))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 8)))
= add (32, read-an (32, 6, s), 8))
$\wedge$ (read-mem (x, mc-mem (stepn (s, 8)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: strchr-s0-sn-rfile-base2
(strchr-s0p (s, $i*$, i, str, n, lst, ch)
$\wedge$ (get-nth (i, lst) $\neq$ ch)
$\wedge$ (get-nth (i, lst) = 0)
$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 8)))
= read-rn ($oplen$, rn, mc-rfile (s)))

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

THEOREM: strchr-s0-s0
(strchr-s0p (s, $i*$, i, str, n, lst, ch)

```
 ∧   (get-nth (i, lst) ≠ ch)
 ∧   (get-nth (i, lst) ≠ 0))
 →   (strchr-s0p (stepn (s, 6), add (32, i*, 1), 1 + i, str, n, lst, ch)
      ∧   (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))
      ∧   (read-mem (x, mc-mem (stepn (s, 6)), k)
            =   read-mem (x, mc-mem (s), k)))
```

THEOREM: strchr-s0-s0-rfile

```
(strchr-s0p (s, i*, i, str, n, lst, ch)
 ∧   (get-nth (i, lst) ≠ ch)
 ∧   (get-nth (i, lst) ≠ 0)
 ∧   d2-7a2-5p (rn))
 →   (read-rn (oplen, rn, mc-rfile (stepn (s, 6)))
      =   read-rn (oplen, rn, mc-rfile (s)))
```

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

THEOREM: strchr-s0p-info

```
strchr-s0p (s, i*, i, str, n, lst, ch) → ((i < n) = t)
```

THEOREM: strchr-s0-sn

```
let sn be stepn (s, strchr1-t (i, n, lst, ch))
in
strchr-s0p (s, i*, i, str, n, lst, ch)
 →   ((mc-status (sn) = 'running)
      ∧   (mc-pc (sn) = linked-rts-addr (s))
      ∧   (read-dn (32, 0, sn)
            =   if strchr (i, n, lst, ch)
                then add (32, str, strchr* (i*, i, n, lst, ch))
                else 0 endif)
      ∧   (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: strchr-s0-sn-rfile

```
let sn be stepn (s, strchr1-t (i, n, lst, ch))
in
(strchr-s0p (s, i*, i, str, n, lst, ch) ∧ d2-7a2-5p (rn))
 →   (read-rn (oplen, rn, mc-rfile (sn))
      =   read-rn (oplen, rn, mc-rfile (s))) endlet
```

EVENT: Disable strchr-s0p-info.

```
; the correctness of strchr.
```

THEOREM: strchr-correctness
let *sn* **be** stepn (*s*, strchr-t (*n*, *lst*, *ch*))
in
strchr-statep (*s*, *str*, *n*, *lst*, *ch*)
→ ((mc-status (*sn*) = 'running)
∧ (mc-pc (*sn*) = rts-addr (*s*))
∧ (read-rn (32, 14, mc-rfile (*sn*))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (read-rn (32, 15, mc-rfile (*sn*))
= add (32, read-sp (*s*), 4))
∧ (d2-7a2-5p (*rn*)
→ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))))
∧ (disjoint (*x*, *k*, sub (32, 4, read-sp (*s*)), 16)
→ (read-mem (*x*, mc-mem (*sn*), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))
∧ (read-dn (32, 0, *sn*)
= **if** strchr (0, *n*, *lst*, *ch*)
then add (32, *str*, strchr* (0, 0, *n*, *lst*, *ch*))
else 0 **endif**)) **endlet**

EVENT: Disable strchr-t.

```
; strchr* --> strchr.
```

THEOREM: strchr*-strchr
(strchr (*i*, *n*, *lst*, *ch*)
∧ (*i* = nat-to-uint (*i**))
∧ nat-rangep (*i**, 32)
∧ uint-rangep (*n*, 32))
→ (nat-to-uint (strchr* (*i**, *i*, *n*, *lst*, *ch*)) = strchr (*i*, *n*, *lst*, *ch*))

THEOREM: strchr-non-zerop-la
let *sn* **be** stepn (*s*, strchr-t (*n*, *lst*, *ch*))
in
(strchr-statep (*s*, *str*, *n*, *lst*, *ch*)
∧ nat-rangep (*str*, 32)
∧ (nat-to-uint (*str*) ≠ 0)
∧ uint-rangep (nat-to-uint (*str*) + *n*, 32)
∧ strchr (0, *n*, *lst*, *ch*))
→ (nat-to-uint (read-dn (32, 0, *sn*)) ≠ 0) **endlet**

THEOREM: strchr-non-zerop
let *sn* **be** stepn (s, strchr-t (n, lst, ch))
in
(strchr-statep (s, str, n, lst, ch) $\wedge$ strchr (0, n, lst, ch))
$\rightarrow$ (nat-to-uint (read-dn (32, 0, sn)) $\neq$ 0) **endlet**

EVENT: Disable strchr*.

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

Index

add, 3–7

d2-7a2-5p, 4–7
disjoint, 3, 4, 7

equal*, 4
evenp, 3, 4

get-nth, 3, 5, 6

linked-a6, 4–6
linked-rts-addr, 4–6

mc-mem, 3–7
mc-pc, 3–7
mc-rfile, 4–7
mc-status, 3–7
mcode-addrp, 3, 4
mem-lst, 3, 4

nat-rangep, 4, 7
nat-to-uint, 3, 4, 7, 8

ram-addrp, 3, 4
read-an, 4–6
read-dn, 4–8
read-mem, 3–7
read-rn, 4–7
read-sp, 3, 4, 7
rom-addrp, 3, 4
rts-addr, 4, 7

slen, 3, 4
splus, 3
stepn, 3–8
strchr, 6–8
strchr*, 6, 7
strchr*-strchr, 7
strchr-code, 2–4
strchr-correctness, 7
strchr-induct, 3
strchr-non-zerop, 8
strchr-non-zerop-la, 7
strchr-s-s0, 4
strchr-s-s0-else, 4
strchr-s-s0-mem, 4
strchr-s-s0-rfile, 4
strchr-s0-s0, 5
strchr-s0-s0-rfile, 6
strchr-s0-sn, 6
strchr-s0-sn-base1, 5
strchr-s0-sn-base2, 5
strchr-s0-sn-rfile, 6
strchr-s0-sn-rfile-base1, 5
strchr-s0-sn-rfile-base2, 5
strchr-s0p, 4–6
strchr-s0p-info, 6
strchr-statep, 3, 4, 7, 8
strchr-t, 3, 7, 8
strchr1-t, 3, 6
sub, 3, 4, 7

uint-rangep, 3, 4, 7
uread-mem, 3