```
#|

 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 STRRCHR 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 strrchr function in the Berkeley string library.

char *
strrchr(p, ch)
register const char *p;
        register char ch;
{
register const char *save;

for (save = NULL;; ++p) {
if (*p == ch)
```

```
save = p;
if (!*p)
return((char *)save);
}
/* NOTREACHED */
}
```

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

```
0x26c8 <strrchr>:       linkw fp,#0
0x26cc <strrchr+4>:     moveal fp@(8),a0
0x26d0 <strrchr+8>:     moveb fp@(15),d1
0x26d4 <strrchr+12>:    clrl d0
0x26d6 <strrchr+14>:    cmpb a0@,d1
0x26d8 <strrchr+16>:    bne 0x26dc <strrchr+20>
0x26da <strrchr+18>:    movel a0,d0
0x26dc <strrchr+20>:    tstb a0@
0x26de <strrchr+22>:    beq 0x26e4 <strrchr+28>
0x26e0 <strrchr+24>:    addqw #1,a0
0x26e2 <strrchr+26>:    bra 0x26d6 <strrchr+14>
0x26e4 <strrchr+28>:    unlk fp
0x26e6 <strrchr+30>:    rts
```

The machine code of the above program is:

```
<strrchr>:     0x4e56  0x0000  0x206e  0x0008  0x122e  0x000f  0x4280  0xb210
<strrchr+16>:  0x6602  0x2008  0x4a10  0x6704  0x5248  0x60f2  0x4e5e  0x4e75

'(78     86     0      0      32     110    0      8
  18     46     0      15     66     128    178    16
  102    2      32     8      74     16     103    4
  82     72     96     242    78     94     78     117)
|#
```

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

DEFINITION:
STRRCHR-CODE
= '(78 86 0 0 32 110 0 8 18 46 0 15 66 128 178 16 102 2
32 8 74 16 103 4 82 72 96 242 78 94 78 117)

; the computation time of the program.

DEFINITION:

```
strrchr-t1 (i, n, lst, ch)
=   if i < n
    then if get-nth (i, lst) = ch
         then if get-nth (i, lst) = 0  then 7
              else splus (7, strrchr-t1 (1 + i, n, lst, ch)) endif
         elseif get-nth (i, lst) = 0  then 6
         else splus (6, strrchr-t1 (1 + i, n, lst, ch)) endif
    else 0 endif
```

DEFINITION:
strrchr-t $(n,\ lst,\ ch)$ = splus (4, strrchr-t1 $(0,\ n,\ lst,\ ch)$)

```
; an induction hint.
```

DEFINITION:

```
strrchr-induct (s, i*, i, n, lst, ch, j*, j)
=   if i < n
    then if get-nth (i, lst) = ch
         then if get-nth (i, lst) = 0  then t
              else strrchr-induct (stepn (s, 7),
                                   add (32, i*, 1),
                                   1 + i,
                                   n,
                                   lst,
                                   ch,
                                   i*,
                                   i) endif
         elseif get-nth (i, lst) = 0  then t
         else strrchr-induct (stepn (s, 6),
                              add (32, i*, 1),
                              1 + i,
                              n,
                              lst,
                              ch,
                              j*,
                              j) endif
    else t endif
```

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

DEFINITION:

```
strrchr-statep (s, str, n, lst, ch)
=   ((mc-status (s) = 'running)
     ∧  evenp (mc-pc (s))
     ∧  rom-addrp (mc-pc (s), mc-mem (s), 32)
```

$\wedge$ mcode-addrp (mc-pc (s), mc-mem (s), STRRCHR-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))
$\wedge$ stringp (0, n, lst)
$\wedge$ ($n \in \mathbf{N}$)
$\wedge$ uint-rangep (n, 32)
$\wedge$ (nat-to-uint (str) $\neq$ 0)
$\wedge$ uint-rangep (nat-to-uint (str) + n, 32))

```
; an intermediate state.
```

DEFINITION:
index-j (str, $j*$, j)
= **if** j **then** add (32, str, $j*$)
else 0 **endif**

DEFINITION:
strrchr-s0p (s, $i*$, i, str, n, lst, ch, $j*$, j)
= ((mc-status (s) = 'running)
$\wedge$ evenp (mc-pc (s))
$\wedge$ rom-addrp (sub (32, 14, mc-pc (s)), mc-mem (s), 32)
$\wedge$ mcode-addrp (sub (32, 14, mc-pc (s)), mc-mem (s), STRRCHR-CODE)
$\wedge$ ram-addrp (read-an (32, 6, 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 (read-an (32, 6, s), 16, str, n)
$\wedge$ equal* (read-an (32, 0, s), add (32, str, $i*$))
$\wedge$ (ch = nat-to-uint (read-dn (8, 1, s)))
$\wedge$ equal* (read-dn (32, 0, s), index-j (str, $j*$, j))
$\wedge$ stringp (i, n, lst)
$\wedge$ ($i < n$)
$\wedge$ ($i* \in \mathbf{N}$)
$\wedge$ nat-rangep ($i*$, 32)
$\wedge$ (i = nat-to-uint ($i*$))
$\wedge$ ($n \in \mathbf{N}$)
$\wedge$ uint-rangep (n, 32))

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

THEOREM: strrchr-s-s0
strrchr-statep (s, str, n, lst, ch)
$\rightarrow$ strrchr-s0p (stepn (s, 4), 0, 0, str, n, lst, ch, **f**, **f**)

THEOREM: strrchr-s-s0-else
strrchr-statep $(s,\ str,\ n,\ lst,\ ch)$
$\rightarrow$ ((linked-rts-addr (stepn $(s,\ 4)$) = rts-addr (s))
  $\wedge$ (linked-a6 (stepn $(s,\ 4)$) = read-an (32, 6, s))
  $\wedge$ (read-rn (32, 14, mc-rfile (stepn $(s,\ 4)$))
    = sub (32, 4, read-sp (s))))

THEOREM: strrchr-s-s0-rfile
(strrchr-statep $(s,\ str,\ n,\ lst,\ ch)$ $\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn $(oplen,\ rn,$ mc-rfile (stepn $(s,\ 4)$))
  = read-rn $(oplen,\ rn,$ mc-rfile (s)))

THEOREM: strrchr-s-s0-mem
(strrchr-statep $(s,\ str,\ n,\ lst,\ ch)$ $\wedge$ disjoint $(x,\ k,$ sub (32, 4, read-sp (s)), 16))
$\rightarrow$ (read-mem $(x,$ mc-mem (stepn $(s,\ 4)$), $k)$ = read-mem $(x,$ mc-mem $(s),\ k)$)

```
; from s0 to exit: s0 --> sn.
; base case 1. s0 --> sn, when lst[i] = ch and lst[i] = 0.
```

THEOREM: strrchr-s0-sn-base1
(strrchr-s0p $(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ j^*,\ j)$
 $\wedge$ (get-nth $(i,\ lst) \neq ch$)
 $\wedge$ (get-nth $(i,\ lst) = 0$))
$\rightarrow$ ((mc-status (stepn $(s,\ 6)$) = 'running)
  $\wedge$ (mc-pc (stepn $(s,\ 6)$) = linked-rts-addr (s))
  $\wedge$ (read-dn (32, 0, stepn $(s,\ 6)$) = index-j $(str,\ j^*,\ j)$)
  $\wedge$ (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: strrchr-s0-sn-rfile-base1
(strrchr-s0p $(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ j^*,\ j)$
 $\wedge$ (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 $(s,\ 6)$))
  = read-rn $(oplen,\ rn,$ mc-rfile (s)))

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

THEOREM: strrchr-s0-sn-base2
(strrchr-s0p $(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ j^*,\ j)$
 $\wedge$ (get-nth $(i,\ lst) = ch$)

$\wedge$ (get-nth $(i,\ lst) = 0$))
$\rightarrow$ ((mc-status (stepn $(s,\ 7)) =$ `'running`)
$\wedge$ (mc-pc (stepn $(s,\ 7))$ = linked-rts-addr (s))
$\wedge$ (read-dn (32, 0, stepn $(s,\ 7))$ = add (32, str, $i*$))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn $(s,\ 7)))$ = linked-a6 (s))
$\wedge$ (read-rn (32, 15, mc-rfile (stepn $(s,\ 7)))$
= add (32, read-an (32, 6, s), 8))
$\wedge$ (read-mem (x, mc-mem (stepn $(s,\ 7)),\ k$)
= read-mem (x, mc-mem $(s),\ k$)))

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

```
; induction case 1: s0 --> s0, when lst[i] = ch and lst[i] =\= 0.
```

THEOREM: index-j-la
$j \rightarrow$ (index-j ($str,\ j*,\ j$) = add (32, $str,\ j*$))

THEOREM: strrchr-s0-s0-1
(strrchr-s0p ($s,\ i*,\ i,\ str,\ n,\ lst,\ ch,\ j*,\ j$)
$\wedge$ (get-nth $(i,\ lst) = ch$)
$\wedge$ (get-nth $(i,\ lst) \neq 0$))
$\rightarrow$ (strrchr-s0p (stepn $(s,\ 7)$, add (32, $i*$, 1), $1 + i,\ str,\ n,\ lst,\ ch,\ i*,\ i$)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn $(s,\ 7)$))
= read-rn (32, 14, mc-rfile (s)))
$\wedge$ (linked-a6 (stepn $(s,\ 7)$) = linked-a6 (s))
$\wedge$ (linked-rts-addr (stepn $(s,\ 7)$) = linked-rts-addr (s))
$\wedge$ (read-mem (x, mc-mem (stepn $(s,\ 7)),\ k$)
= read-mem (x, mc-mem $(s),\ k$)))

THEOREM: strrchr-s0-s0-rfile-1
(strrchr-s0p ($s,\ i*,\ i,\ str,\ n,\ lst,\ ch,\ j*,\ j$)
$\wedge$ (get-nth $(i,\ lst) = ch$)
$\wedge$ (get-nth $(i,\ lst) \neq 0$)
$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen,\ rn$, mc-rfile (stepn $(s,\ 7)$))
= read-rn ($oplen,\ rn$, mc-rfile (s))))

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

THEOREM: strrchr-s0-s0-2
(strrchr-s0p (s, $i*$, i, str, n, lst, ch, $j*$, j)
$\land$ (get-nth (i, lst) $\neq$ ch)
$\land$ (get-nth (i, lst) $\neq$ `0`))
$\rightarrow$ (strrchr-s0p (stepn (s, `6`), add (`32`, $i*$, `1`), $1 + i$, str, n, lst, ch, $j*$, j)
  $\land$ (read-rn (`32`, `14`, mc-rfile (stepn (s, `6`)))
    $=$ read-rn (`32`, `14`, mc-rfile (s)))
  $\land$ (linked-a6 (stepn (s, `6`)) $=$ linked-a6 (s))
  $\land$ (linked-rts-addr (stepn (s, `6`)) $=$ linked-rts-addr (s))
  $\land$ (read-mem (x, mc-mem (stepn (s, `6`)), k)
    $=$ read-mem (x, mc-mem (s), k)))

THEOREM: strrchr-s0-s0-rfile-2
(strrchr-s0p (s, $i*$, i, str, n, lst, ch, $j*$, j)
$\land$ (get-nth (i, lst) $\neq$ ch)
$\land$ (get-nth (i, lst) $\neq$ `0`)
$\land$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, `6`)))
  $=$ read-rn ($oplen$, rn, mc-rfile (s)))

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

THEOREM: strrchr-s0p-info
strrchr-s0p (s, $i*$, i, str, n, lst, ch, $j*$, j) $\rightarrow$ ((($i < n$) $=$ **t**) $\land$ ($i \in \mathbf{N}$))

THEOREM: strrchr-s0p-la
$\neg$ strrchr-s0p (s, $i*$, **f**, str, n, lst, ch, $j*$, j)

THEOREM: strrchr-s0-sn
let sn **be** stepn (s, strrchr-t1 (i, n, lst, ch))
in
strrchr-s0p (s, $i*$, i, str, n, lst, ch, $j*$, j)
$\rightarrow$ ((mc-status (sn) $=$ `'running`)
  $\land$ (mc-pc (sn) $=$ linked-rts-addr (s))
  $\land$ (read-dn (`32`, `0`, sn)
    $=$ **if** strrchr (i, n, lst, ch, j)
      then add (`32`, str, strrchr* ($i*$, i, n, lst, ch, $j*$))
      else `0` **endif**)
  $\land$ (read-rn (`32`, `14`, mc-rfile (sn)) $=$ linked-a6 (s))
  $\land$ (read-rn (`32`, `15`, mc-rfile (sn))
    $=$ add (`32`, read-an (`32`, `6`, s), `8`))
  $\land$ (read-mem (x, mc-mem (sn), k) $=$ read-mem (x, mc-mem (s), k))) **endlet**

EVENT: Disable strrchr-s0p-info.

THEOREM: strrchr-s0-sn-rfile
(strrchr-s0p $(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ j^*,\ j)$ ∧ d2-7a2-5p (rn))
→ (read-rn $(oplen,\ rn,$ mc-rfile (stepn $(s,$ strrchr-t1 $(i,\ n,\ lst,\ ch))))$
  = read-rn $(oplen,\ rn,$ mc-rfile $(s)))$

```
; the correctness of strrchr.
```

THEOREM: strrchr-correctness
let *sn* **be** stepn (s, strrchr-t (n, lst, ch))
in
strrchr-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** strrchr (0, n, lst, ch, **f**)
      then add (32, str, strrchr* (0, 0, n, lst, ch, **f**))
      else 0 **endif**)) **endlet**

EVENT: Disable strrchr-t.

```
; strrchr* --> strrchr.
```

THEOREM: strrchr*-strrchr
(strrchr (i, n, lst, ch, j)
 ∧ (i = nat-to-uint (i^*))
 ∧ nat-rangep (i^*, 32)
 ∧ uint-rangep (n, 32))
→ (nat-to-uint (strrchr* (i^*, i, n, lst, ch, j^*))
  = **if** j = **f** **then** strrchr (i, n, lst, ch, **f**)
    else strrchr (i, n, lst, ch, nat-to-uint (j^*)) **endif**)

THEOREM: strrchr-non-zerop-la
let *sn* **be** stepn (s, strrchr-t (n, lst, ch))
in

(strrchr-statep $(s,\ str,\ n,\ lst,\ ch)$
$\wedge$ nat-rangep $(str,\ 32)$
$\wedge$ (nat-to-uint $(str) \neq 0$)
$\wedge$ uint-rangep (nat-to-uint $(str) + n,\ 32$)
$\wedge$ strrchr $(0,\ n,\ lst,\ ch,\ \mathbf{f})$)
$\rightarrow$ (nat-to-uint (read-dn $(32,\ 0,\ sn)) \neq 0$) **endlet**

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

EVENT: Disable strrchr*.

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

Index