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

```assembly
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*,
        j) 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)
∧ mcode-addrp (mc-pc (s), mc-mem (s), STRRCHR-CODE)
∧ ram-addrp (sub (32, 4, read-sp (s)), mc-mem (s), 16)
∧ ram-addrp (str, mc-mem (s), n)
∧ mem-lst (1, str, mc-mem (s), n, lst)
∧ disjoint (sub (32, 4, read-sp (s)), 16, str, n)
∧ (str = read-mem (add (32, read-sp (s), 4), mc-mem (s), 4))
∧ (ch = uread-mem (add (32, read-sp (s), 11), mc-mem (s), 1))
∧ stringp (0, n, lst)
∧ (n ∈ N)
∧ uint-rangep (n, 32)
∧ (nat-to-uint (str) ≠ 0)
∧ uint-rangep (nat-to-uint (str) + n, 32))

; an intermediate state.

**Definition:**

\[
\text{index-j} (\text{str}, j^*, j) = \begin{cases} 
\text{add} (32, \text{str}, j^*) & \text{if } j \\
0 & \text{else}
\end{cases}
\]

**Definition:**

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

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

**Theorem:** strrchr-s-s0

strchr-statep (s, str, n, lst, ch) 
→ strrchr-s0p (stepn (s, 4), 0, 0, str, n, lst, ch, f, f)
Theorem: strrchr-s-s0-else
strrchr-statep \((s, str, n, lst, ch)\)
\[
\rightarrow ((\text{linked-rts-addr (stepn} (s, 4)) = \text{rts-addr} (s))
\wedge (\text{linked-a6 (stepn} (s, 4)) = \text{read-an} (32, 6, s))
\wedge (\text{read-rn} (32, 14, \text{mc-file} \text{ (stepn} (s, 4))))
\]
\[
\quad = \text{sub} (32, 4, \text{read-sp} (s)))
\]

Theorem: strrchr-s-s0-rfile
(strrchr-statep \((s, str, n, lst, ch)\) \wedge d2-7a2-5p (rn))
\[
\rightarrow (\text{read-rn} \text{ (oplen} \text{, \text{rn}, \text{mc-file} \text{ (stepn} (s, 4))))
\]
\[
\quad = \text{read-rn} \text{ (oplen} \text{, \text{rn}, \text{mc-file} (s)))}
\]

Theorem: strrchr-s-s0-mem
(strrchr-statep \((s, str, n, lst, ch)\) \wedge \text{disjoint} \((x, k, \text{sub} (32, 4, \text{read-sp} (s)), 16))
\[
\rightarrow (\text{read-mem} \text{ (x, \text{mc-mem} \text{ (stepn} \text{ (s, 4))}, k) = \text{read-mem} \text{ (x, \text{mc-mem} (s)}, k))
\]

; from s0 to exit: s0 \rightarrow \text{sn}.

; base case 1. s0 \rightarrow \text{sn}, when \text{lst[i]} = \text{ch} and \text{lst[i]} = 0.

Theorem: strrchr-s0-sn-base1
(strrchr-s0p \((s, i^*, i, str, n, lst, ch, j^*, j)\)
\wedge (\text{get-nth} \text{ (i, lst) \neq ch})
\wedge (\text{get-nth} \text{ (i, lst) = 0}))
\[
\rightarrow ((\text{mc-status} \text{ (stepn} (s, 6)) = \text{’running})
\wedge (\text{mc-pc} \text{ (stepn} (s, 6)) = \text{linked-rts-addr} (s))
\wedge (\text{read-dn} (32, 0, \text{stepn} (s, 6)) = \text{index-j} \text{ (str, j^*, j)})
\wedge (\text{read-rn} (32, 14, \text{mc-file} \text{ (stepn} (s, 6)))) = \text{linked-a6} (s))
\wedge (\text{read-rn} (32, 15, \text{mc-file} \text{ (stepn} (s, 6))))
\quad = \text{add} (32, \text{read-an} (32, 6, s), 8))
\wedge (\text{read-mem} \text{ (x, \text{mc-mem} \text{ (stepn} \text{ (s, 6))}, k))
\quad = \text{read-mem} \text{ (x, \text{mc-mem} (s)}, k)))
\]

Theorem: strrchr-s0-sn-rfile-base1
(strrchr-s0p \((s, i^*, i, str, n, lst, ch, j^*, j)\)
\wedge (\text{get-nth} \text{ (i, lst) \neq ch})
\wedge (\text{get-nth} \text{ (i, lst) = 0})
\wedge d2-7a2-5p (rn))
\[
\rightarrow (\text{read-rn} \text{ (oplen} \text{, \text{rn}, \text{mc-file} \text{ (stepn} (s, 6))))
\]
\[
\quad = \text{read-rn} \text{ (oplen} \text{, \text{rn}, \text{mc-file} (s)))}
\]

; base case 2: s0 \rightarrow \text{sn}, when lst[i] =\neq \text{ch} and lst[i] = 0.

Theorem: strrchr-s0-sn-base2
(strrchr-s0p \((s, i^*, i, str, n, lst, ch, j^*, j)\)
\wedge (\text{get-nth} \text{ (i, lst) = ch})
\]
\[
\land (\text{get-nth} (i, \text{lst}) = 0))
\rightarrow ((\text{mc-status} (\text{stepn} (s, 7)) = \text{'running})
\land (\text{mc-pc} (\text{stepn} (s, 7)) = \text{linked-rts-addr} (s))
\land (\text{read-dn} (32, 0, \text{stepn} (s, 7)) = \text{add} (32, \text{str}, i^*))
\land (\text{read-rn} (32, 14, \text{mc-rfile} (\text{stepn} (s, 7))) = \text{linked-a6} (s))
\land (\text{read-rn} (32, 15, \text{mc-rfile} (\text{stepn} (s, 7)))
\quad = \text{add} (32, \text{read-an} (32, 6, s), 8))
\land (\text{read-mem} (x, \text{mc-mem} (\text{stepn} (s, 7)), k)
\quad = \text{read-mem} (x, \text{mc-mem} (s), k)))
\]

**Theorem**: strrchr-s0-sn-rfile-base2

\[
(\text{strrchr-s0p} (s, i^*, i, \text{str}, n, \text{lst}, \text{ch}, j^*, j)
\land (\text{get-nth} (i, \text{lst}) = \text{ch})
\land (\text{get-nth} (i, \text{lst}) = 0)
\land d2-7a2-5p (rn))
\rightarrow (\text{read-rn} (\text{oplen}, \text{rn}, \text{mc-rfile} (\text{stepn} (s, 7)))
\quad = \text{read-rn} (\text{oplen}, \text{rn}, \text{mc-rfile} (s)))
\]

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

**Theorem**: index-j-la

\[
j \rightarrow (\text{index-j} (\text{str}, j^*, j) = \text{add} (32, \text{str}, j^*))
\]

**Theorem**: strrchr-s0-s0-1

\[
(\text{strrchr-s0p} (s, i^*, i, \text{str}, n, \text{lst}, \text{ch}, j^*, j)
\land (\text{get-nth} (i, \text{lst}) = \text{ch})
\land (\text{get-nth} (i, \text{lst}) \neq 0))
\rightarrow (\text{strrchr-s0p} (\text{stepn} (s, 7), \text{add} (32, i^*, 1), 1 + i, \text{str}, n, \text{lst}, \text{ch}, i^*, i)
\land (\text{read-rn} (32, 14, \text{mc-rfile} (\text{stepn} (s, 7)))
\quad = \text{read-rn} (32, 14, \text{mc-rfile} (s)))
\land (\text{linked-a6} (\text{stepn} (s, 7)) = \text{linked-a6} (s))
\land (\text{linked-rts-addr} (\text{stepn} (s, 7)) = \text{linked-rts-addr} (s))
\land (\text{read-mem} (x, \text{mc-mem} (\text{stepn} (s, 7)), k)
\quad = \text{read-mem} (x, \text{mc-mem} (s), k)))
\]

**Theorem**: strrchr-s0-s0-rfile-1

\[
(\text{strrchr-s0p} (s, i^*, i, \text{str}, n, \text{lst}, \text{ch}, j^*, j)
\land (\text{get-nth} (i, \text{lst}) = \text{ch})
\land (\text{get-nth} (i, \text{lst}) \neq 0)
\land d2-7a2-5p (rn))
\rightarrow (\text{read-rn} (\text{oplen}, \text{rn}, \text{mc-rfile} (\text{stepn} (s, 7)))
\quad = \text{read-rn} (\text{oplen}, \text{rn}, \text{mc-rfile} (s)))
\]

; 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)
∧ (get-nth (i, lst) ≠ ch)
∧ (get-nth (i, lst) ≠ 0))
→ (strrchr-s0p (stepn (s, 6), add (32, i*, 1), 1 + i, str, n, lst, ch, j*, j)
∧ (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: strrchr-s0-s0-rfile-2
(strrchr-s0p (s, i*, i, str, n, lst, ch, j*, j)
∧ (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: strrchr-s0p-info
strrchr-s0p (s, i*, i, str, n, lst, ch, j*, j) → (((i < n) = t) ∧ (i ∈ N))

Theorem: strrchr-s0p-la
¬ 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)
→ (((mc-status (sn) = 'running)
∧ (mc-pc (sn) = linked-rts-addr (s))
∧ (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)
∧ (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

Event: Disable strrchr-s0p-info.
**Theorem:** strrchr-s0-sn-rfile

\[(\text{strrchr-s0p}(s, i^*, i, n, lst, ch, j^*, j) \land \text{d2-7a2-5p}(rn)) \rightarrow (\text{read-rn}(\text{oplen}, rn, \text{mc-rfile}(\text{stepn}(s, \text{strrchr-t1}(i, n, lst, ch)))) = \text{read-rn}(\text{oplen}, rn, \text{mc-rfile}(s)))\]

; the correctness of strrchr.

**Theorem:** strrchr-correctness

let sn be \(\text{stepn}(s, \text{strrchr-t}(n, lst, ch))\)

in

\[\text{strrchr-statep}(s, str, n, lst, ch) \rightarrow (\text{mc-status}(sn) = \text{'running}) \land (\text{mc-pc}(sn) = \text{rts-addr}(s)) \land (\text{read-rn}(32, 14, \text{mc-rfile}(sn)) = \text{read-rn}(32, 14, \text{mc-rfile}(s))) \land (\text{read-rn}(32, 15, \text{mc-rfile}(sn)) = \text{add}(32, \text{read-sp}(s), 4)) \land (\text{d2-7a2-5p}(rn) \rightarrow (\text{read-rn}(\text{oplen}, rn, \text{mc-rfile}(sn)) = \text{read-rn}(\text{oplen}, rn, \text{mc-rfile}(s)))) \land (\text{disjoint}(x, k, \text{sub}(32, 4, \text{read-sp}(s)), 16) \rightarrow (\text{read-mem}(x, \text{mc-mem}(sn), k) = \text{read-mem}(x, \text{mc-mem}(s), k))) \land (\text{read-dn}(32, 0, sn) = \text{if strrchr}(0, n, lst, ch, f) \text{ then } \text{add}(32, str, \text{strrchr*}(0, 0, n, lst, ch, f)) \text{ else } 0 \text{ endif})\) endlet

**Event:** Disable strrchr-t.

; strrchr* --> strrchr.

**Theorem:** strrchr*-strrchr

\[(\text{strrchr}(i, n, lst, ch, j) \land (i = \text{nat-to-uint}(i^*)) \land \text{nat-rangep}(i^*, 32) \land \text{uint-rangep}(n, 32)) \rightarrow (\text{nat-to-uint}(\text{strrchr*}(i^*, i, n, lst, ch, j^*))) = \text{if } j = f \text{ then } \text{strrchr}(i, n, lst, ch, f) \text{ else } \text{strrchr}(i, n, lst, ch, \text{nat-to-uint}(j^*)) \text{ endif})\]

**Theorem:** strrchr-non-zerop-la

let sn be \(\text{stepn}(s, \text{strrchr-t}(n, lst, ch))\)

in
(strrchr-statep (s, str, n, lst, ch)
∧ nat-rangep (str, 32)
∧ (nat-to-uint (str) ≠ 0)
∧ uint-rangep (nat-to-uint (str) + n, 32)
∧ strrchr (0, n, lst, ch, f))
→ (nat-to-uint (read-dn (32, 0, sn)) ≠ 0) endlet

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

EVENT: Disable strrchr*.

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

add, 3–8

d2-7a2-5p, 5–8
disjoint, 4, 5, 8
equal*, 4
evenp, 3, 4
get-nth, 3, 5–7
index-j, 4–6
index-j-la, 6
linked-a6, 5–7
linked-rts-addr, 5–7
mc-mem, 3–8
mc-pc, 3–8
mc-rfile, 5–8
mc-status, 3–8
mcode-addrp, 4
mem-lst, 4

nat-rangep, 4, 8, 9
nat-to-uint, 4, 8, 9

ram-addrp, 4
read-an, 4–7
read-dn, 4–9
read-mem, 4–8
read-rn, 5–8
read-sp, 4, 5, 8
rom-addrp, 3, 4
rts-addr, 5, 8

splus, 3
stepn, 3–9
stringp, 4
strchr, 7–9
strchr*, 7, 8
strchr*-strchr, 8
strchr-code, 2, 4

strchr-correctness, 8
strchr-induct, 3
strchr-non-zerop, 9
strchr-non-zerop-la, 8
strchr-s-s0, 4
strchr-s-s0-else, 5
strchr-s-s0-mem, 5
strchr-s-s0-rfile, 5
strchr-s-s0-s0-1, 6
strchr-s-s0-s0-2, 7
strchr-s-s0-s0-rfile-1, 6
strchr-s-s0-s0-rfile-2, 7
strchr-s-s0-sn, 7
strchr-s-s0-sn-base1, 5
strchr-s-s0-sn-base2, 5
strchr-s-s0-sn-rfile, 8
strchr-s-s0-sn-rfile-base1, 5
strchr-s-s0-sn-rfile-base2, 6
strchr-s-s0p, 4–8
strchr-s-s0p-info, 7
strchr-s-s0p-la, 7
strchr-statep, 3–5, 8, 9
strchr-t, 3, 8, 9
strchr-t1, 2, 3, 7, 8
sub, 4, 5, 8

uint-rangep, 4, 8, 9
uread-mem, 4