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 MEMCHR 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 memchr function in the Berkeley string library.

void *
memchr(s, c, n)
const void *s;
register unsigned char c;
register size_t n;
{
  if (n != 0) {
    register const unsigned char *p = s;
The MC68020 assembly code of the C function memchr on SUN-3 is given as follows. This binary is generated by "gcc -O".

0x27e8 <memchr>:  linkw fp,#0
0x27ec <memchr+4>:  moveb fp@15,d1
0x27f0 <memchr+8>:  moveal fp@8,a0
0x27f4 <memchr+12>:  beq 0x2808 <memchr+32>
0x27f6 <memchr+14>:  movel fp@15,d0
0x27fc <memchr+18>:  cmpb a0+,d1
0x2800 <memchr+20>:  bne 0x2804 <memchr+28>
0x27fe <memchr+22>:  movel a0+,d0
0x2800 <memchr+26>:  subl #1,d0
0x2802 <memchr+28>:  bra 0x280a <memchr+34>
0x2804 <memchr+28>:  subl #1,d0
0x2806 <memchr+28>:  bne 0x27fa <memchr+18>
0x2808 <memchr+32>:  clrl d0
0x280a <memchr+32>:  unlk fp
0x280c <memchr+36>:  rts

The machine code of the above program is:

<memchr>:
0x4e56 0x0000 0x122e 0x000f 0x202e 0x0010 0x6712 0x206e
<memchr+16>:
0x0008 0xb218 0x6606 0x2008 0x5380 0x6006 0x5380 0x66f2
<memchr+32>:
0x4280 0x4e56 0x4e75

'(78 86 0 0 18 46 0 15
32 46 0 16 103 18 32 110
0 8 178 24 102 6 32 8
83 128 96 6 83 128 102 242
66 128 78 94 78 117)

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

DEFINITION:
MEMCHR-CODE
; the computation time of the program.

**Definition:**
\[
\text{memchr-t1}(i, n, lst, ch) = \\
\text{if} \ \text{get-nth}(i, lst) = ch \ \text{then} \ 7 \\
\text{elseif} \ (n - 1) = 0 \ \text{then} \ 7 \\
\text{else} \ \text{splus}(4, \ \text{memchr-t1}(1 + i, n - 1, lst, ch)) \ \text{endif}
\]

**Definition:**
\[
\text{memchr-t}(n, lst, ch) = \\
\text{if} \ n = 0 \ \text{then} \ 7 \\
\text{else} \ \text{splus}(5, \ \text{memchr-t1}(0, n, lst, ch)) \ \text{endif}
\]

; an induction hint.

**Definition:**
\[
\text{memchr-induct}(s, \ i^*, i, n, lst, ch) = \\
\text{if} \ \text{get-nth}(i, lst) = ch \ \text{then} \ t \\
\text{elseif} \ (n - 1) = 0 \ \text{then} \ t \\
\text{else} \ \text{memchr-induct}(\text{stepn}(s, 4), \ \text{add}(32, i^*, 1), 1 + i, n - 1, lst, ch)) \ \text{endif}
\]

; the preconditions of the initial state.

**Definition:**
\[
\text{memchr-statep}(s, \ str, n, lst, ch) = \\
(\text{mc-status}(s) = '\text{running}') \\
\land \ \text{evenp}(\text{mc-pc}(s)) \\
\land \ \text{rom-adr}(\text{mc-pc}(s), \ \text{mc-mem}(s), 38) \\
\land \ \text{mcode-adr}(\text{mc-pc}(s), \ \text{mc-mem}(s), \ \text{MEMCHR-CODE}) \\
\land \ \text{ram-adr}(\text{sub}(32, 4, \ \text{read-sp}(s)), \ \text{mc-mem}(s), 20) \\
\land \ \text{ram-adr}(str, \ \text{mc-mem}(s), n) \\
\land \ \text{mem-lst}(1, str, \ \text{mc-mem}(s), n, lst) \\
\land \ \text{disjoint}(\text{sub}(32, 4, \ \text{read-sp}(s)), 20, str, n) \\
\land \ (str = \ \text{read-mem}(\text{add}(32, \ \text{read-sp}(s), 4), \ \text{mc-mem}(s), 4)) \\
\land \ (ch = \ \text{uread-mem}(\text{add}(32, \ \text{read-sp}(s), 11), \ \text{mc-mem}(s), 1)) \\
\land \ (n = \ \text{uread-mem}(\text{add}(32, \ \text{read-sp}(s), 12), \ \text{mc-mem}(s), 4)) \\
\land \ (\text{nat-to-uint}(str) \neq 0) \\
\land \ \text{uint-range}(\text{nat-to-uint}(str) + n, 32))
\]

; an intermediate state.
**Definition:**

\[
\text{memchr-s0p}(s, i^*, i, str, n, lst, ch, n_\ast) = \begin{cases} 
\text{running} & (\text{evenp}(mc-pc(s)) \\
\text{running} & (\text{rom-addrp}(sub(32, 18, mc-pc(s)), mc-mem(s), 38)) \\
\text{running} & (\text{mcode-addrp}(sub(32, 18, mc-pc(s)), mc-mem(s), \text{MEMCHR-CODE})) \\
\text{running} & (\text{ram-addrp}(\text{read-an}(32, 6, s), mc-mem(s), 20)) \\
\text{running} & (\text{ram-addrp}(str, mc-mem(s), n_\ast)) \\
\text{running} & (\text{mem-lst}(1, str, mc-mem(s), n_\ast, lst)) \\
\text{running} & (\text{disjoint}(\text{read-an}(32, 6, s), 20, str, n_\ast)) \\
\text{running} & (ch = \text{nat-to-uint}(\text{read-dn}(8, 1, s))) \\
\text{running} & (n = \text{nat-to-uint}(\text{read-dn}(32, 0, s))) \\
\text{running} & (i = \text{nat-to-uint}(i^*)) \\
\text{running} & ((i + n) \leq n_\ast) \\
\text{running} & (n \neq 0) \\
\text{running} & (i^* \in \mathbb{N}) \\
\text{running} & (n_\ast \in \mathbb{N}) \\
\text{running} & \text{nat-rangep}(i^*, 32) \\
\text{running} & \text{uint-rangep}(n_\ast, 32)
\end{cases}
\]

; from the initial state \(s\) to exit: \(s \rightarrow s_n\).

**Theorem:** memchr-s-sn

\[
\begin{align*}
\text{(memchr-statep}(s, str, n, lst, ch) \land (n = 0)) \\
\rightarrow & \quad ((\text{mc-status}(\text{stepn}(s, 7)) = \text{running}) \\
& \land (\text{mc-pc}(\text{stepn}(s, 7)) = \text{rts-addr}(s)) \\
& \land (\text{read-dn}(32, 0, \text{stepn}(s, 7)) = 0) \\
& \land (\text{read-rn}(32, 15, \text{mc-rfile}(\text{stepn}(s, 7)))) \\
& \quad = \text{add}(32, \text{read-an}(32, 7, s), 4)) \\
& \land (\text{read-rn}(32, 14, \text{mc-rfile}(\text{stepn}(s, 7)))) = \text{read-an}(32, 6, s)))
\end{align*}
\]

**Theorem:** memchr-s-sn-rfile

\[
\begin{align*}
\text{(memchr-statep}(s, str, n, lst, ch) \land (n = 0) \land d2-7a2-5p(rn)) \\
\rightarrow & \quad (\text{read-rn}(\text{oplen}, rn, \text{mc-rfile}(\text{stepn}(s, 7)))) \\
& \quad = \text{read-rn}(\text{oplen}, rn, \text{mc-rfile}(s)))
\end{align*}
\]

**Theorem:** memchr-s-sn-mem

\[
\begin{align*}
\text{(memchr-statep}(s, str, n, lst, ch) \\
& \land (n = 0) \\
& \land \text{disjoint}(x, k, \text{sub}(32, 4, \text{read-sp}(s)), 20)) \\
\rightarrow & \quad (\text{read-mem}(x, \text{mc-mem}(\text{stepn}(s, 7)), k) = \text{read-mem}(x, \text{mc-mem}(s), k))
\end{align*}
\]

; from the initial state \(s\) to \(s_0\): \(s \rightarrow s_0\).
THEOREM: memchr-s-s0
\[(\text{memchr-statep}(s, \text{str}, \text{n}, \text{lst}, \text{ch}) \land (\text{n} \neq 0))\]
\[\rightarrow \text{memchr-s0p}(\text{stepn}(s, 5), 0, 0, \text{str}, \text{n}, \text{lst}, \text{ch}, \text{n})\]

THEOREM: memchr-s-s0-else
\[(\text{memchr-statep}(s, \text{str}, \text{n}, \text{lst}, \text{ch}) \land (\text{n} \neq 0))\]
\[\rightarrow (\text{linked-rts-addr(\text{stepn}(s, 5)) = rts-addr(s)}\]
\[\land (\text{linked-a6(\text{stepn}(s, 5)) = read-an(32, 6, s)})\]
\[\land (\text{read-rn}(32, 14, \text{mc-rfile(\text{stepn}(s, 5)))) = \text{sub}(32, 4, \text{read-sp}(s))))\]

THEOREM: memchr-s-s0-rfile
\[(\text{memchr-statep}(s, \text{str}, \text{n}, \text{lst}, \text{ch}) \land (\text{n} \neq 0) \land \text{d2-7a2-5p(rn)})\]
\[\rightarrow (\text{read-rn}(\text{oplen}, \text{rn}, \text{mc-rfile(\text{stepn}(s, 5)))) = \text{read-rn}(\text{oplen}, \text{rn}, \text{mc-rfile(s)}))\]

THEOREM: memchr-s-s0-mem
\[(\text{memchr-statep}(s, \text{str}, \text{n}, \text{lst}, \text{ch}) \land (\text{n} \neq 0) \land \text{disjoint}(\text{x, k, sub}(32, 4, \text{read-sp(s)}), 20))\]
\[\rightarrow (\text{read-mem}(\text{x}, \text{mc-mem}(\text{stepn}(s, 5)), \text{k}) = \text{read-mem}(\text{x}, \text{mc-mem}(s), \text{k}))\]

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

THEOREM: memchr-s0-sn-base1
\[(\text{memchr-s0p}(s, i^*, \text{i}, \text{str}, \text{n}, \text{lst}, \text{ch}, \text{n}) \land (\text{get-nth}(\text{i}, \text{lst}) = \text{ch}))\]
\[\rightarrow (\text{mc-status}(\text{stepn}(s, 7)) = \text{'running})\]
\[\land (\text{mc-pc}(\text{stepn}(s, 7)) = \text{linked-rts-addr(s)})\]
\[\land (\text{read-dn}(32, 0, \text{stepn}(s, 7)) = \text{add}(32, \text{str}, i^*))\]
\[\land (\text{read-rn}(32, 14, \text{mc-rfile(\text{stepn}(s, 7)))) = \text{linked-a6(s)})\]
\[\land (\text{read-rn}(32, 15, \text{mc-rfile(\text{stepn}(s, 7)))) = \text{add}(32, \text{read-an}(32, 6, s), 8))\]
\[\land (\text{read-mem}(\text{x}, \text{mc-mem}(\text{stepn}(s, 7)), \text{k}) = \text{read-mem}(\text{x}, \text{mc-mem}(s), \text{k}))))\]

THEOREM: memchr-s0-sn-rfile-base1
\[(\text{memchr-s0p}(s, i^*, i, \text{str}, \text{n}, \text{lst}, \text{ch}, \text{n})\land (\text{get-nth}(\text{i}, \text{lst}) = \text{ch}))\]
\[\land \text{d2-7a2-5p(rn))}\]
\[\rightarrow (\text{read-rn}(\text{oplen}, \text{rn}, \text{mc-rfile(\text{stepn}(s, 7)))) = \text{read-rn}(\text{oplen}, \text{rn}, \text{mc-rfile(s)}))\]

; base case 2: s0 --> sn, when lst[i] =\= ch, n-1 = 0.
Theorem: `memchr-s0-sn-base2`  
(memchr-s0p (s, i*, i, str, n, lst, ch, n)  
∧ (get-nth (i, lst) ≠ ch)  
∧ ((n - 1) = 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-rn (32, 14, mc-rfile (stepn (s, 7)))) = linked-a6 (s))  
∧ (read-rn (32, 15, mc-rfile (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: `memchr-s0-sn-rfile-base2`  
(memchr-s0p (s, i*, i, str, n, lst, ch, n)  
∧ (get-nth (i, lst) ≠ ch)  
∧ ((n - 1) = 0)  
∧ d2-7a2-5p (rn))  
→ (read-rn (oplen, rn, mc-rfile (stepn (s, 7))))  
= read-rn (oplen, rn, mc-rfile (s)))

; induction case: s0 --> s0.

Theorem: `memchr-s0-s0`  
(memchr-s0p (s, i*, i, str, n, lst, ch, n)  
∧ (get-nth (i, lst) ≠ ch)  
∧ ((n - 1) ≠ 0))  
→ (memchr-s0p (stepn (s, 4), add (32, i*, 1), 1 + i, str, n - 1, lst, ch, n)  
∧ (read-rn (32, 14, mc-rfile (stepn (s, 4))))  
= read-rn (32, 14, mc-rfile (s)))  
∧ (linked-a6 (stepn (s, 4)) = linked-a6 (s))  
∧ (linked-rts-addr (stepn (s, 4)) = linked-rts-addr (s))  
∧ (read-mem (x, mc-mem (stepn (s, 4)), k)  
= read-mem (x, mc-mem (s), k)))

Theorem: `memchr-s0-s0-rfile`  
(memchr-s0p (s, i*, i, str, n, lst, ch, n)  
∧ (get-nth (i, lst) ≠ ch)  
∧ ((n - 1) ≠ 0)  
∧ d2-7a2-5p (rn))  
→ (read-rn (oplen, rn, mc-rfile (stepn (s, 4))))  
= read-rn (oplen, rn, mc-rfile (s)))

; put together (s0 --> exit).
Theorem: memchr-s0-sn

let sn be stepn (s, memchr-t1 (i, n, lst, ch))
in
memchr-s0p (s, i*, i, str, n, lst, ch, n_)
→ ((mc-status (sn) = 'running)
∧ (mc-pc (sn) = linked-rts-addr (s))
∧ (read-dn (32, 0, sn)
   = if memchr1 (i, n, lst, ch)
      then add (32, str, memchr* (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: memchr-s0-sn-rfile

(memchr-s0p (s, i*, i, str, n, lst, ch, n_) \& d2-7a2-5p (rn))
→ (read-rn (oplen, rn, mc-rfile (stepn (s, memchr-t1 (i, n, lst, ch))))
   = read-rn (oplen, rn, mc-rfile (s)))

; the correctness of the MEMCHR program.

Theorem: memchr-correctness

let sn be stepn (s, memchr-t (n, lst, ch))
in
memchr-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), 20))
   → (read-mem (x, mc-mem (sn), k)
      = read-mem (x, mc-mem (s), k)))
∧ (read-dn (32, 0, sn)
   = if memchr (n, lst, ch)
      then add (32, str, memchr* (0, 0, n, lst, ch))
      else 0 endif))
endlet

Event: Disable memchr-t.
; memchr* --> memchr.

**Theorem:** memchr*-memchr1
\[
\begin{align*}
&\text{(memchr1} (i, n, lst, ch) \\
&\land (i = \text{nat-to-uint}(i^*)) \\
&\land \text{nat-rangep}(i^*, 32) \\
&\land \text{uint-rangep}(i + n, 32)) \\
&\rightarrow (\text{nat-to-uint}(\text{memchr*}(i^*, i, n, lst, ch)) = \text{memchr1}(i, n, lst, ch))
\end{align*}
\]

**Theorem:** memchr-non-zerop-la
\[
\begin{align*}
\text{let } sn &\text{ be stepn}(s, \text{memchr-t}(n, lst, ch)) \\
\text{in } &\text{(memchr-statep}(s, str, n, lst, ch) \\
&\land (n \in \mathbb{N}) \\
&\land \text{nat-rangep}(str, 32) \\
&\land (\text{nat-to-uint}(str) \neq 0) \\
&\land \text{uint-rangep}(\text{nat-to-uint}(str) + n, 32) \\
&\land \text{memchr}(n, lst, ch)) \\
&\rightarrow (\text{nat-to-uint}(\text{read-dn}(32, 0, sn)) \neq 0) \text{ endlet}
\end{align*}
\]

**Theorem:** memchr-non-zerop
\[
\begin{align*}
\text{let } sn &\text{ be stepn}(s, \text{memchr-t}(n, lst, ch)) \\
\text{in } &\text{(memchr-statep}(s, str, n, lst, ch) \land \text{memchr}(n, lst, ch)) \\
&\rightarrow (\text{nat-to-uint}(\text{read-dn}(32, 0, sn)) \neq 0) \text{ endlet}
\end{align*}
\]

**Event:** Disable memchr*.

; some properties of memchr.
; see file cstring.events.
Index

add, 3–7
d2-7a2-5p, 4–7
disjoint, 3–5, 7
equal*, 4
evenp, 3, 4
get-nth, 3, 5, 6
linked-a6, 5–7
linked-rts-addr, 5–7
mc-mem, 3–7
mc-pc, 3–7
mc-rfile, 4–7
mc-status, 3–7
mcode-addrp, 3, 4
mem-lst, 3, 4
memchr, 7, 8
memchr*, 7, 8
memchr*-memchr1, 8
memchr-code, 2–4
memchr-correctness, 7
memchr-induct, 3
memchr-non-zerop, 8
memchr-non-zerop-la, 8
memchr-s-s0, 5
memchr-s-s0-else, 5
memchr-s-s0-mem, 5
memchr-s-s0-rfile, 5
memchr-s-sn, 4
memchr-s-sn-mem, 4
memchr-s-sn-rfile, 4
memchr-s0-s0, 6
memchr-s0-s0-rfile, 6
memchr-s0-sn, 7
memchr-s0-sn-base1, 5
memchr-s0-sn-base2, 6
memchr-s0-sn-rfile, 7
memchr-s0-sn-rfile-base1, 5
memchr-s0-sn-rfile-base2, 6
memchr-s0p, 4–7
memchr-statep, 3–5, 7, 8
memchr-t, 3, 7, 8
memchr-t1, 3, 7
memchr1, 7, 8
nat-rangep, 4, 8
nat-to-uint, 3, 4, 8
ram-addrp, 3, 4
read-an, 4–7
read-dn, 4–8
read-mem, 3–7
read-rn, 4–7
read-sp, 3–5, 7
rom-addrp, 3, 4
rts-addr, 4, 5, 7
splus, 3
stepn, 3–8
sub, 3–5, 7
uint-rangep, 3, 4, 8
uread-mem, 3