```
#|

 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 MEMSET 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 memset function in the Berkeley string library.

void *
memset(dst, c, n)
        void *dst;
        register int c;
        register size_t n;
{

        if (n != 0) {
                register char *d = dst;
```

```
                do
                        *d++ = c;
                while (--n != 0);
        }
        return (dst);
}
```

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

```
0x29d0 <memset>:        linkw fp,#0
0x29d4 <memset+4>:      movel d2,sp@-
0x29d6 <memset+6>:      movel fp@(8),d2
0x29da <memset+10>:     movel fp@(16),d0
0x29de <memset+14>:     beq 0x29ec <memset+28>
0x29e0 <memset+16>:     moveal d2,a0
0x29e2 <memset+18>:     moveb fp@(15),d1
0x29e6 <memset+22>:     moveb d1,a0@+
0x29e8 <memset+24>:     subl #1,d0
0x29ea <memset+26>:     bne 0x29e6 <memset+22>
0x29ec <memset+28>:     movel d2,d0
0x29ee <memset+30>:     movel fp@(-4),d2
0x29f2 <memset+34>:     unlk fp
0x29f4 <memset+36>:     rts
```

The machine code of the above program is:

```
<memset>:      0x4e56  0x0000  0x2f02  0x242e  0x0008  0x202e  0x0010  0x670c
<memset+16>:   0x2042  0x122e  0x000f  0x10c1  0x5380  0x66fa  0x2002  0x242e
<memset+32>:   0xfffc  0x4e5e  0x4e75

'(78    86      0       0       47      2       36      46
  0     8       32      46      0       16      103     12
  32    66      18      46      0       15      16      193
  83    128     102     250     32      2       36      46
  255   252     78      94      78      117)
|#
; in the logic, the above program is defined by (memset-code).
```

DEFINITION:
MEMSET-CODE
= '(78 86 0 0 47 2 36 46 0 8 32 46 0 16 103 12 32 66 18
46 0 15 16 193 83 128 102 250 32 2 36 46 255 252 78

```
      94 78 117)

; the computation time of the program.
```

DEFINITION:
memset-t1 (n)
$=$ **if** $(n - 1) = 0$ **then** 7
else splus (3, memset-t1 $(n - 1)$) **endif**

DEFINITION:
memset-t (n)
$=$ **if** $n = 0$ **then** 9
else splus (7, memset-t1 (n)) **endif**

```
; an induction hint.
```

DEFINITION:
memset-induct $(s,\ i^*,\ i,\ n,\ lst,\ ch)$
$=$ **if** $(n - 1) = 0$ **then** **t**
else memset-induct (stepn $(s,\ 3)$,
add (32, i^*, 1),
$1 + i$,
$n - 1$,
put-nth $(ch,\ i,\ lst)$,
ch) **endif**

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

DEFINITION:
memset-statep $(s,\ str,\ n,\ lst,\ ch)$
$=$ ((mc-status (s) = `'running`)
∧ evenp (mc-pc (s))
∧ rom-addrp (mc-pc (s), mc-mem (s), 38)
∧ mcode-addrp (mc-pc (s), mc-mem (s), MEMSET-CODE)
∧ ram-addrp (sub (32, 8, read-sp (s)), mc-mem (s), 24)
∧ ram-addrp (str, mc-mem (s), n)
∧ mem-lst (1, str, mc-mem (s), n, lst)
∧ disjoint (sub (32, 8, read-sp (s)), 24, 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))
∧ (n = uread-mem (add (32, read-sp (s), 12), mc-mem (s), 4)))

```
; an intermediate state.
```

DEFINITION:
memset-s0p $(s,\ i^*,\ i,\ str,\ n,\ lst,\ ch,\ n_{-})$

$$
\begin{aligned}
= \quad & ((\text{mc-status}\,(s) = \texttt{'running}) \\
& \wedge \quad \text{evenp}\,(\text{mc-pc}\,(s)) \\
& \wedge \quad \text{rom-addrp}\,(\text{sub}\,(32,\ 22,\ \text{mc-pc}\,(s)),\ \text{mc-mem}\,(s),\ 38) \\
& \wedge \quad \text{mcode-addrp}\,(\text{sub}\,(32,\ 22,\ \text{mc-pc}\,(s)),\ \text{mc-mem}\,(s),\ \textsc{memset-code}) \\
& \wedge \quad \text{ram-addrp}\,(\text{sub}\,(32,\ 4,\ \text{read-an}\,(32,\ 6,\ s)),\ \text{mc-mem}\,(s),\ 24) \\
& \wedge \quad \text{ram-addrp}\,(\mathit{str},\ \text{mc-mem}\,(s),\ n_{\_}) \\
& \wedge \quad \text{mem-lst}\,(1,\ \mathit{str},\ \text{mc-mem}\,(s),\ n_{\_},\ \mathit{lst}) \\
& \wedge \quad \text{disjoint}\,(\text{sub}\,(32,\ 4,\ \text{read-an}\,(32,\ 6,\ s)),\ 24,\ \mathit{str},\ n_{\_}) \\
& \wedge \quad \text{equal*}\,(\text{read-an}\,(32,\ 0,\ s),\ \text{add}\,(32,\ \mathit{str},\ i^*)) \\
& \wedge \quad (\mathit{str} = \text{read-dn}\,(32,\ 2,\ s)) \\
& \wedge \quad (\mathit{ch} = \text{nat-to-uint}\,(\text{read-dn}\,(8,\ 1,\ s))) \\
& \wedge \quad (i = \text{nat-to-uint}\,(i^*)) \\
& \wedge \quad (n = \text{nat-to-uint}\,(\text{read-dn}\,(32,\ 0,\ s))) \\
& \wedge \quad ((i + n) \le n_{\_}) \\
& \wedge \quad (n \ne 0) \\
& \wedge \quad (i^* \in \mathbf{N}) \\
& \wedge \quad (n_{\_} \in \mathbf{N}) \\
& \wedge \quad \text{nat-rangep}\,(i^*,\ 32) \\
& \wedge \quad \text{uint-rangep}\,(n_{\_},\ 32))
\end{aligned}
$$

```
; from the intial state s to exit: s --> sn.
```

THEOREM: memset-s-sn

$$
\begin{aligned}
& (\text{memset-statep}\,(s,\ \mathit{str},\ n,\ \mathit{lst},\ \mathit{ch}) \wedge (n = 0)) \\
\rightarrow \quad & ((\text{mc-status}\,(\text{stepn}\,(s,\ 9)) = \texttt{'running}) \\
& \wedge \quad (\text{mc-pc}\,(\text{stepn}\,(s,\ 9)) = \text{rts-addr}\,(s)) \\
& \wedge \quad (\text{read-dn}\,(32,\ 0,\ \text{stepn}\,(s,\ 9)) = \mathit{str}) \\
& \wedge \quad \text{mem-lst}\,(1,\ \mathit{str},\ \text{mc-mem}\,(\text{stepn}\,(s,\ 9)),\ n,\ \mathit{lst}) \\
& \wedge \quad (\text{read-rn}\,(32,\ 15,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 9))) \\
& \qquad = \quad \text{add}\,(32,\ \text{read-an}\,(32,\ 7,\ s),\ 4)) \\
& \wedge \quad (\text{read-rn}\,(32,\ 14,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 9))) = \text{read-an}\,(32,\ 6,\ s)))
\end{aligned}
$$

THEOREM: memset-s-sn-rfile

$$
\begin{aligned}
& (\text{memset-statep}\,(s,\ \mathit{str},\ n,\ \mathit{lst},\ \mathit{ch}) \\
& \wedge \quad (n = 0) \\
& \wedge \quad (\mathit{oplen} \le 32) \\
& \wedge \quad \text{d2-7a2-5p}\,(\mathit{rn})) \\
\rightarrow \quad & (\text{read-rn}\,(\mathit{oplen},\ \mathit{rn},\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 9))) \\
& \quad = \quad \text{read-rn}\,(\mathit{oplen},\ \mathit{rn},\ \text{mc-rfile}\,(s)))
\end{aligned}
$$

THEOREM: memset-s-sn-mem

$$
\begin{aligned}
& (\text{memset-statep}\,(s,\ \mathit{str},\ n,\ \mathit{lst},\ \mathit{ch}) \\
& \wedge \quad (n = 0) \\
& \wedge \quad \text{disjoint}\,(x,\ k,\ \text{sub}\,(32,\ 8,\ \text{read-sp}\,(s)),\ 24)) \\
\rightarrow \quad & (\text{read-mem}\,(x,\ \text{mc-mem}\,(\text{stepn}\,(s,\ 9)),\ k) = \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k))
\end{aligned}
$$

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

THEOREM: memset-s-s0

$$(\text{memset-statep}\,(s,\ str,\ n,\ lst,\ ch) \wedge (n \neq 0)) \rightarrow \text{memset-s0p}\,(\text{stepn}\,(s,\ 7),\ 0,\ 0,\ str,\ n,\ lst,\ ch,\ n)$$

THEOREM: memset-s-s0-else

$$\begin{aligned}&(\text{memset-statep}\,(s,\ str,\ n,\ lst,\ ch) \wedge (n \neq 0))\\ \rightarrow\ &((\text{linked-rts-addr}\,(\text{stepn}\,(s,\ 7)) = \text{rts-addr}\,(s))\\ &\wedge\ (\text{linked-a6}\,(\text{stepn}\,(s,\ 7)) = \text{read-an}\,(32,\ 6,\ s))\\ &\wedge\ (\text{read-rn}\,(32,\ 14,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 7)))\\ &\qquad = \text{sub}\,(32,\ 4,\ \text{read-sp}\,(s)))\\ &\wedge\ (\text{rn-saved}\,(\text{stepn}\,(s,\ 7)) = \text{read-rn}\,(32,\ 2,\ \text{mc-rfile}\,(s))))\end{aligned}$$

THEOREM: memset-s-s0-rfile

$$\begin{aligned}&(\text{memset-statep}\,(s,\ str,\ n,\ lst,\ ch) \wedge (n \neq 0) \wedge \text{d3-7a2-5p}\,(rn))\\ \rightarrow\ &(\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 7)))\\ &\ = \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s)))\end{aligned}$$

THEOREM: memset-s-s0-mem

$$\begin{aligned}&(\text{memset-statep}\,(s,\ str,\ n,\ lst,\ ch)\\ &\wedge\ (n \neq 0)\\ &\wedge\ \text{disjoint}\,(x,\ k,\ \text{sub}\,(32,\ 8,\ \text{read-sp}\,(s)),\ 24))\\ \rightarrow\ &(\text{read-mem}\,(x,\ \text{mc-mem}\,(\text{stepn}\,(s,\ 7)),\ k) = \text{read-mem}\,(x,\ \text{mc-mem}\,(s),\ k))\end{aligned}$$

```
; from s0 to exit (base case), from s0 to s0 (induction case).
; base case: s0 --> exit.
```

THEOREM: memset-s0-sn-base

$$\begin{aligned}&(\text{memset-s0p}\,(s,\ i*,\ i,\ str,\ n,\ lst,\ ch,\ n\_) \wedge ((n-1) = 0))\\ \rightarrow\ &((\text{mc-status}\,(\text{stepn}\,(s,\ 7)) = \texttt{'running})\\ &\wedge\ (\text{mc-pc}\,(\text{stepn}\,(s,\ 7)) = \text{linked-rts-addr}\,(s))\\ &\wedge\ (\text{read-dn}\,(32,\ 0,\ \text{stepn}\,(s,\ 7)) = str)\\ &\wedge\ \text{mem-lst}\,(1,\ str,\ \text{mc-mem}\,(\text{stepn}\,(s,\ 7)),\ n\_,\ \text{put-nth}\,(ch,\ i,\ lst))\\ &\wedge\ (\text{read-rn}\,(32,\ 14,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 7))) = \text{linked-a6}\,(s))\\ &\wedge\ (\text{read-rn}\,(32,\ 15,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 7)))\\ &\qquad = \text{add}\,(32,\ \text{read-an}\,(32,\ 6,\ s),\ 8)))\end{aligned}$$

THEOREM: memset-s0-sn-rfile-base

$$\begin{aligned}&(\text{memset-s0p}\,(s,\ i*,\ i,\ str,\ n,\ lst,\ ch,\ n\_)\\ &\wedge\ ((n-1) = 0)\\ &\wedge\ (oplen \leq 32)\\ &\wedge\ \text{d2-7a2-5p}\,(rn))\\ \rightarrow\ &(\text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(\text{stepn}\,(s,\ 7)))\\ &\ = \textbf{if}\ \text{d3-7a2-5p}\,(rn)\ \textbf{then}\ \text{read-rn}\,(oplen,\ rn,\ \text{mc-rfile}\,(s))\\ &\quad\ \textbf{else}\ \text{head}\,(\text{rn-saved}\,(s),\ oplen)\ \textbf{endif})\end{aligned}$$

THEOREM: memset-s0-sn-mem-base
(memset-s0p (s, $i*$, i, str, n, lst, ch, $n\_$)
 ∧ (($n - 1$) = 0)
 ∧ disjoint (x, k, str, $n\_$))
→ (read-mem (x, mc-mem (stepn (s, 7)), k) = read-mem (x, mc-mem (s), k))

```
; induction case: s0 --> s0.
```

THEOREM: memset-s0-s0
(memset-s0p (s, $i*$, i, str, n, lst, ch, $n\_$) ∧ (($n - 1$) ≠ 0))
→ (memset-s0p (stepn (s, 3),
  add (32, $i*$, 1),
  $1 + i$,
  str,
  $n - 1$,
  put-nth (ch, i, lst),
  ch,
  $n\_$)
 ∧ (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))
 ∧ (rn-saved (stepn (s, 3)) = rn-saved (s)))

THEOREM: memset-s0-s0-rfile
(memset-s0p (s, $i*$, i, str, n, lst, ch, $n\_$) ∧ (($n - 1$) ≠ 0) ∧ d3-7a2-5p (rn))
→ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 3)))
 = read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: memset-s0-s0-mem
(memset-s0p (s, $i*$, i, str, n, lst, ch, $n\_$)
 ∧ (($n - 1$) ≠ 0)
 ∧ disjoint (x, k, str, $n\_$))
→ (read-mem (x, mc-mem (stepn (s, 3)), k) = read-mem (x, mc-mem (s), k))

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

THEOREM: memset-s0-sn
let sn **be** stepn (s, memset-t1 (n))
in
memset-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) = str)
 ∧ mem-lst (1, str, mc-mem (sn), $n\_$, memset1 (i, n, lst, ch))

$\wedge$ (read-rn (32, 14, mc-rfile (*sn*)) = linked-a6 (*s*))
$\wedge$ (read-rn (32, 15, mc-rfile (*sn*))
= add (32, read-an (32, 6, *s*), 8))) **endlet**

THEOREM: memset-s0-sn-rfile
(memset-s0p (*s*, *i**, *i*, *str*, *n*, *lst*, *ch*, *n_*) $\wedge$ (*oplen* $\leq$ 32) $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, memset-t1 (*n*))))
= **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
else head (rn-saved (*s*), *oplen*) **endif**)

THEOREM: memset-s0-sn-mem
(memset-s0p (*s*, *i**, *i*, *str*, *n*, *lst*, *ch*, *n_*) $\wedge$ disjoint (*x*, *k*, *str*, *n_*))
$\rightarrow$ (read-mem (*x*, mc-mem (stepn (*s*, memset-t1 (*n*))), *k*)
= read-mem (*x*, mc-mem (*s*), *k*))

```
; the correctness of the MEMSET program.
```

THEOREM: memset-correctness
let *sn* **be** stepn (*s*, memset-t (*n*))
in
memset-statep (*s*, *str*, *n*, *lst*, *ch*)
$\rightarrow$ ((mc-status (*sn*) = 'running)
$\wedge$ (mc-pc (*sn*) = rts-addr (*s*))
$\wedge$ (read-rn (32, 14, mc-rfile (*sn*))
= read-rn (32, 14, mc-rfile (*s*)))
$\wedge$ (read-rn (32, 15, mc-rfile (*sn*))
= add (32, read-sp (*s*), 4))
$\wedge$ (((*oplen* $\leq$ 32) $\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))))
$\wedge$ ((disjoint (*x*, *k*, sub (32, 8, read-sp (*s*)), 24)
$\wedge$ disjoint (*x*, *k*, *str*, *n*))
$\rightarrow$ (read-mem (*x*, mc-mem (*sn*), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))
$\wedge$ (read-dn (32, 0, *sn*) = *str*)
$\wedge$ mem-lst (1, *str*, mc-mem (*sn*), *n*, memset (*n*, *lst*, *ch*))) **endlet**

EVENT: Disable memset-t.

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

Index