```
#|

 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 STRNCAT 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 strncat function in the Berkeley string library.

char *
strncat(dst, src, n)
char *dst;
const char *src;
register size_t n;
{
if (n != 0) {
register char *d = dst;
register const char *s = src;
```

```
while (*d != 0)
d++;
do {
if ((*d = *s++) == 0)
break;
d++;
} while (--n != 0);
*d = 0;
}
return (dst);
}
```

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

```
0x25d0 <strncat>:       linkw fp,#0
0x25d4 <strncat+4>:     movel d2,sp@-
0x25d6 <strncat+6>:     movel fp@(8),d2
0x25da <strncat+10>:    movel fp@(16),d1
0x25de <strncat+14>:    beq 0x25fe <strncat+46>
0x25e0 <strncat+16>:    moveal d2,a0
0x25e2 <strncat+18>:    moveal fp@(12),a1
0x25e6 <strncat+22>:    tstb a0@
0x25e8 <strncat+24>:    beq 0x25f0 <strncat+32>
0x25ea <strncat+26>:    addqw #1,a0
0x25ec <strncat+28>:    tstb a0@
0x25ee <strncat+30>:    bne 0x25ea <strncat+26>
0x25f0 <strncat+32>:    moveb a1@+,d0
0x25f2 <strncat+34>:    moveb d0,a0@
0x25f4 <strncat+36>:    beq 0x25fc <strncat+44>
0x25f6 <strncat+38>:    addqw #1,a0
0x25f8 <strncat+40>:    subl #1,d1
0x25fa <strncat+42>:    bne 0x25f0 <strncat+32>
0x25fc <strncat+44>:    clrb a0@
0x25fe <strncat+46>:    movel d2,d0
0x2600 <strncat+48>:    movel fp@(-4),d2
0x2604 <strncat+52>:    unlk fp
0x2606 <strncat+54>:    rts
```

The machine code of the above program is:

```
<strncat>:     0x4e56  0x0000  0x2f02  0x242e  0x0008  0x222e  0x0010  0x671e
<strncat+16>:  0x2042  0x226e  0x000c  0x4a10  0x6706  0x5248  0x4a10  0x66fa
```

```
<strncat+32>:  0x1019  0x1080  0x6706  0x5248  0x5381  0x66f4  0x4210  0x2002
<strncat+48>:  0x242e  0xfffc  0x4e5e  0x4e75

'(78      86      0       0       47      2       36      46
  0       8       34      46      0       16      103     30
  32      66      34      110     0       12      74      16
  103     6       82      72      74      16      102     250
  16      25      16      128     103     6       82      72
  83      129     102     244     66      16      32      2
  36      46      255     252     78      94      78      117)
|#

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

DEFINITION:
STRNCAT-CODE
= `'(78 86 0 0 47 2 36 46 0 8 34 46 0 16 103 30 32 66 34 110 0 12 74 16 103 6 82 72 74 16 102 250 16 25 16 128 103 6 82 72 83 129 102 244 66 16 32 2 36 46 255 252 78 94 78 117)`

```
; the computation time of the program.
```

DEFINITION:
strncat-t0 (i, *n1*, *lst1*)
= **if** $i <$ *n1*
  then if get-nth (i, *lst1*) = `0` **then** `2`
    else splus (`3`, strncat-t0 (1 + i, *n1*, *lst1*)) **endif**
  else `0` **endif**

DEFINITION:
strncat-t1 (*n1*, *lst1*) = splus (`10`, strncat-t0 (`1`, *n1*, *lst1*))

DEFINITION:
strncat-t2 (j, n, *lst2*)
= **if** get-nth (j, *lst2*) = `0` **then** `8`
  elseif ($n - 1$) = `0` **then** `11`
  else splus (`6`, strncat-t2 (1 + j, $n - 1$, *lst2*)) **endif**

DEFINITION:
strncat-t (*n1*, *lst1*, n, *lst2*)
= **if** n = `0` **then** `9`
  elseif get-nth (`0`, *lst1*) = `0` **then** splus (`9`, strncat-t2 (`0`, n, *lst2*))
  else splus (strncat-t1 (*n1*, *lst1*), strncat-t2 (`0`, n, *lst2*)) **endif**

```
; two induction hints.
```

DEFINITION:
strncat-induct0 (s, $i*$, i, *n1*, *lst1*)
= **if** $i < n1$
then if get-nth (i, *lst1*) = `0` **then t**
else strncat-induct0 (stepn (s, `3`),
add (`32`, $i*$, `1`),
$1 + i$,
n1,
lst1) **endif**
else t endif

DEFINITION:
strncat-induct1 (s, $i*$, i, *lst1*, $j*$, j, n, *lst2*)
= **if** get-nth (j, *lst2*) = `0` **then t**
elseif $(n - 1) = 0$ **then t**
else strncat-induct1 (stepn (s, `6`),
add (`32`, $i*$, `1`),
$1 + i$,
put-nth (get-nth (j, *lst2*), i, *lst1*),
add (`32`, $j*$, `1`),
$1 + j$,
$n - 1$,
lst2) **endif**

`; the preconditions of the initial state.`

DEFINITION:
strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
= ((mc-status (s) = `'running`)
$\wedge$ evenp (mc-pc (s))
$\wedge$ rom-addrp (mc-pc (s), mc-mem (s), `56`)
$\wedge$ mcode-addrp (mc-pc (s), mc-mem (s), STRNCAT-CODE)
$\wedge$ ram-addrp (sub (`32`, `8`, read-sp (s)), mc-mem (s), `24`)
$\wedge$ ram-addrp (*str1*, mc-mem (s), *n1*)
$\wedge$ mem-lst (`1`, *str1*, mc-mem (s), *n1*, *lst1*)
$\wedge$ ram-addrp (*str2*, mc-mem (s), n)
$\wedge$ mem-lst (`1`, *str2*, mc-mem (s), n, *lst2*)
$\wedge$ disjoint (sub (`32`, `8`, read-sp (s)), `24`, *str1*, *n1*)
$\wedge$ disjoint (sub (`32`, `8`, read-sp (s)), `24`, *str2*, n)
$\wedge$ disjoint (*str1*, *n1*, *str2*, n)
$\wedge$ (*str1* = read-mem (add (`32`, read-sp (s), `4`), mc-mem (s), `4`))
$\wedge$ (*str2* = read-mem (add (`32`, read-sp (s), `8`), mc-mem (s), `4`))
$\wedge$ (n = uread-mem (add (`32`, read-sp (s), `12`), mc-mem (s), `4`))
$\wedge$ ((1 + (slen (`0`, *n1*, *lst1*) + n)) < *n1*)
$\wedge$ (*n1* $\in$ **N**)

∧ uint-rangep (*n1*, 32))

```
; an intermediate state s0.
```

DEFINITION:
strncat-s0p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n*, *lst2*)
= ((mc-status (*s*) = 'running)
∧ evenp (mc-pc (*s*))
∧ rom-addrp (sub (32, 28, mc-pc (*s*)), mc-mem (*s*), 56)
∧ mcode-addrp (sub (32, 28, mc-pc (*s*)), mc-mem (*s*), STRNCAT-CODE)
∧ ram-addrp (sub (32, 4, read-an (32, 6, *s*)), mc-mem (*s*), 24)
∧ ram-addrp (*str1*, mc-mem (*s*), *n1*)
∧ mem-lst (1, *str1*, mc-mem (*s*), *n1*, *lst1*)
∧ ram-addrp (*str2*, mc-mem (*s*), *n*)
∧ mem-lst (1, *str2*, mc-mem (*s*), *n*, *lst2*)
∧ disjoint (sub (32, 4, read-an (32, 6, *s*)), 24, *str1*, *n1*)
∧ disjoint (sub (32, 4, read-an (32, 6, *s*)), 24, *str2*, *n*)
∧ disjoint (*str1*, *n1*, *str2*, *n*)
∧ equal* (read-an (32, 0, *s*), add (32, *str1*, *i**))
∧ (*str1* = read-dn (32, 2, *s*))
∧ (*str2* = read-an (32, 1, *s*))
∧ (*n* = nat-to-uint (read-dn (32, 1, *s*)))
∧ (*i** ∈ **N**)
∧ nat-rangep (*i**, 32)
∧ (*i* = nat-to-uint (*i**))
∧ ((1 + (slen (*i*, *n1*, *lst1*) + *n*)) < *n1*)
∧ (*n1* ∈ **N**)
∧ uint-rangep (*n1*, 32)
∧ (*n* ≠ 0))

```
; an intermediate state s1.
```

DEFINITION:
strncat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n*, *lst2*, *i*_, *n*_)
= ((mc-status (*s*) = 'running)
∧ evenp (mc-pc (*s*))
∧ rom-addrp (sub (32, 32, mc-pc (*s*)), mc-mem (*s*), 56)
∧ mcode-addrp (sub (32, 32, mc-pc (*s*)), mc-mem (*s*), STRNCAT-CODE)
∧ ram-addrp (sub (32, 4, read-an (32, 6, *s*)), mc-mem (*s*), 24)
∧ ram-addrp (*str1*, mc-mem (*s*), *n1*)
∧ mem-lst (1, *str1*, mc-mem (*s*), *n1*, *lst1*)
∧ ram-addrp (*str2*, mc-mem (*s*), *n*_)
∧ mem-lst (1, *str2*, mc-mem (*s*), *n*_, *lst2*)
∧ disjoint (sub (32, 4, read-an (32, 6, *s*)), 24, *str1*, *n1*)
∧ disjoint (sub (32, 4, read-an (32, 6, *s*)), 24, *str2*, *n*_)

$\wedge$ disjoint (*str1*, *n1*, *str2*, $n\_$)
$\wedge$ equal* (read-an (32, 0, s), add (32, *str1*, $i*$))
$\wedge$ equal* (read-an (32, 1, s), add (32, *str2*, $j*$))
$\wedge$ (*str1* = read-dn (32, 2, s))
$\wedge$ (n = nat-to-uint (read-dn (32, 1, s)))
$\wedge$ (($j + n$) $\leq$ $n\_$)
$\wedge$ ((1 + ($i\_ + n\_$)) $<$ *n1*)
$\wedge$ ($i \leq (i\_ + j)$)
$\wedge$ ($n \neq 0$)
$\wedge$ ($i* \in \mathbf{N}$)
$\wedge$ nat-rangep ($i*$, 32)
$\wedge$ (i = nat-to-uint ($i*$))
$\wedge$ ($j* \in \mathbf{N}$)
$\wedge$ nat-rangep ($j*$, 32)
$\wedge$ (j = nat-to-uint ($j*$))
$\wedge$ (*n1* $<$ 4294967296)
$\wedge$ ($n\_ \in \mathbf{N}$)
$\wedge$ ($n\_ <$ 4294967296))

```
; from the initial state s to exit: s --> sn, when n = 0.
```

THEOREM: strncat-s-sn
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*) $\wedge$ ($n = 0$))
$\rightarrow$ ((mc-status (stepn (s, 9)) = 'running)
$\wedge$ (mc-pc (stepn (s, 9)) = rts-addr (s))
$\wedge$ (read-dn (32, 0, stepn (s, 9)) = *str1*)
$\wedge$ mem-lst (1, *str1*, mc-mem (stepn (s, 9)), *n1*, *lst1*)
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 9)))
= add (32, read-an (32, 7, s), 4))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 9))) = read-an (32, 6, s)))

THEOREM: strncat-s-sn-rfile
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
$\wedge$ ($n = 0$)
$\wedge$ (*oplen* $\leq$ 32)
$\wedge$ d2-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 9)))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: strncat-s-sn-mem
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
$\wedge$ ($n = 0$)
$\wedge$ disjoint (x, k, sub (32, 8, read-sp (s)), 24))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 9)), k) = read-mem (x, mc-mem (s), k))

```
; from the initial state s to s0: s --> s0, when n =\= 0, lst1[0] =\= 0.
```

THEOREM: strncat-s-s0
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq 0$)
 ∧ (get-nth (0, *lst1*) ≠ 0))
→ (strncat-s0p (stepn (s, 10), 1, 1, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
  ∧ (linked-rts-addr (stepn (s, 10)) = rts-addr (s))
  ∧ (linked-a6 (stepn (s, 10)) = read-an (32, 6, s))
  ∧ (read-rn (32, 14, mc-rfile (stepn (s, 10)))
    = sub (32, 4, read-sp (s)))
  ∧ (rn-saved (stepn (s, 10)) = read-dn (32, 2, s)))

THEOREM: strncat-s-s0-rfile
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq 0$)
 ∧ (get-nth (0, *lst1*) ≠ 0)
 ∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 10)))
  = read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: strncat-s-s0-mem
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq 0$)
 ∧ (get-nth (0, *lst1*) ≠ 0)
 ∧ disjoint (x, k, sub (32, 8, read-sp (s)), 24))
→ (read-mem (x, mc-mem (stepn (s, 10)), k) = read-mem (x, mc-mem (s), k))

```
; from initial state s to s1, when n =\= 0, lst1[0] = 0.
```

THEOREM: strncat-s-s1
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq 0$)
 ∧ (get-nth (0, *lst1*) = 0))
→ strncat-s1p (stepn (s, 9), 0, 0, *str1*, *n1*, *lst1*, 0, 0, *str2*, n, *lst2*, 0, n)

THEOREM: strncat-s-s1-else
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq 0$)
 ∧ (get-nth (0, *lst1*) = 0))
→ ((linked-rts-addr (stepn (s, 9)) = rts-addr (s))
  ∧ (linked-a6 (stepn (s, 9)) = read-an (32, 6, s))
  ∧ (read-rn (32, 14, mc-rfile (stepn (s, 9)))
    = sub (32, 4, read-sp (s)))
  ∧ (rn-saved (stepn (s, 9)) = read-rn (32, 2, mc-rfile (s))))

THEOREM: strncat-s-s1-rfile
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq$ 0)
 ∧ (get-nth (0, *lst1*) = 0)
 ∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 9)))
 = read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: strncat-s-s1-mem
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq$ 0)
 ∧ (get-nth (0, *lst1*) = 0)
 ∧ disjoint (x, k, sub (32, 8, read-sp (s)), 24))
→ (read-mem (x, mc-mem (stepn (s, 9)), k) = read-mem (x, mc-mem (s), k))

```
; from s0 to s1: s0 --> s1.
; base case:  s0 --> s1, when lst1[i] = 0.
```

THEOREM: strncat-s0-s1-base
(strncat-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, n, *lst2*) ∧ (get-nth (i, *lst1*) = 0))
→ (strncat-s1p (stepn (s, 2), i^*, i, *str1*, *n1*, *lst1*, 0, 0, *str2*, n, *lst2*, i, n)
 ∧ (read-rn (32, 14, mc-rfile (stepn (s, 2)))
 = read-rn (32, 14, mc-rfile (s)))
 ∧ (linked-a6 (stepn (s, 2)) = linked-a6 (s))
 ∧ (linked-rts-addr (stepn (s, 2)) = linked-rts-addr (s))
 ∧ (rn-saved (stepn (s, 2)) = rn-saved (s))
 ∧ (read-mem (x, mc-mem (stepn (s, 2)), k)
 = read-mem (x, mc-mem (s), k)))

THEOREM: strncat-s0-s1-rfile-base
(strncat-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ (get-nth (i, *lst1*) = 0)
 ∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 2)))
 = read-rn (*oplen*, *rn*, mc-rfile (s)))

```
; induction case: s0 --> s0, when lst1[i] =\= 0.
```

THEOREM: strncat-s0-s0
(strncat-s0p (s, i^*, i, *str1*, *n1*, *lst1*, *str2*, n, *lst2*) ∧ (get-nth (i, *lst1*) ≠ 0))
→ (strncat-s0p (stepn (s, 3), add (32, i^*, 1), 1 + i, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ (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))

$\wedge$ (rn-saved (stepn (s, 3)) = rn-saved (s))
$\wedge$ (read-mem (x, mc-mem (stepn (s, 3)), k)
= read-mem (x, mc-mem (s), k)))

THEOREM: strncat-s0-s0-rfile
(strncat-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
$\wedge$ (get-nth (i, *lst1*) $\neq$ 0)
$\wedge$ d3-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 3)))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

```
; put together. s0 --> s1.
```

THEOREM: strncat-s0p-info
strncat-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, n, *lst2*) $\rightarrow$ ((i < *n1*) = t)

THEOREM: strncat-s0-s1
strncat-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
$\rightarrow$ (strncat-s1p (stepn (s, strncat-t0 (i, *n1*, *lst1*)),
strlen* (i*, i, *n1*, *lst1*),
strlen (i, *n1*, *lst1*),
str1,
n1,
lst1,
0,
0,
str2,
n,
lst2,
strlen (i, *n1*, *lst1*),
n)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, strncat-t0 (i, *n1*, *lst1*))))
= read-rn (32, 14, mc-rfile (s)))
$\wedge$ (linked-a6 (stepn (s, strncat-t0 (i, *n1*, *lst1*))) = linked-a6 (s))
$\wedge$ (linked-rts-addr (stepn (s, strncat-t0 (i, *n1*, *lst1*)))
= linked-rts-addr (s))
$\wedge$ (rn-saved (stepn (s, strncat-t0 (i, *n1*, *lst1*))) = rn-saved (s))
$\wedge$ (read-mem (x, mc-mem (stepn (s, strncat-t0 (i, *n1*, *lst1*))), k)
= read-mem (x, mc-mem (s), k)))

EVENT: Disable strncat-s0p-info.

THEOREM: strncat-s0-s1-rfile
(strncat-s0p (s, i*, i, *str1*, *n1*, *lst1*, *str2*, n, *lst2*) $\wedge$ d3-7a2-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, strncat-t0 (i, *n1*, *lst1*)))))
= read-rn (*oplen*, *rn*, mc-rfile (s)))

```
; put together: s --> s1.
```

THEOREM: strncat-s-s1-1
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq$ 0)
 ∧ (get-nth (0, *lst1*) ≠ 0))
→ strncat-s1p (stepn (s, strncat-t1 (*n1*, *lst1*)),
 strlen* (1, 1, *n1*, *lst1*),
 strlen (1, *n1*, *lst1*),
 str1,
 n1,
 lst1,
 0,
 0,
 str2,
 n,
 lst2,
 strlen (1, *n1*, *lst1*),
 n)

THEOREM: strncat-s-s1-else-1
let *s1* **be** stepn (s, strncat-t1 (*n1*, *lst1*))
in
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq$ 0)
 ∧ (get-nth (0, *lst1*) ≠ 0))
→ ((linked-rts-addr (*s1*) = rts-addr (s))
 ∧ (linked-a6 (*s1*) = read-an (32, 6, s))
 ∧ (read-rn (32, 14, mc-rfile (*s1*))
 = sub (32, 4, read-sp (s)))
 ∧ (rn-saved (*s1*) = read-rn (32, 2, mc-rfile (s)))) **endlet**

THEOREM: strncat-s-s1-rfile-1
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq$ 0)
 ∧ (get-nth (0, *lst1*) ≠ 0)
 ∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, strncat-t1 (*n1*, *lst1*))))
 = read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: strncat-s-s1-mem-1
(strncat-statep (s, *str1*, *n1*, *lst1*, *str2*, n, *lst2*)
 ∧ ($n \neq$ 0)
 ∧ (get-nth (0, *lst1*) ≠ 0)
 ∧ disjoint (x, k, sub (32, 8, read-sp (s)), 24))

→ (read-mem (x, mc-mem (stepn (s, strncat-t1 (*n1*, *lst1*))), k)
= read-mem (x, mc-mem (s), k))

```
; from s1 to exit: s1 --> sn.  By induction.
; base case 1: s1 --> sn, when lst2[j] = 0.
```

THEOREM: strncat-s1-sn-base1
(strncat-s1p (s, *i**, i, *str1*, *n1*, *lst1*, *j**, j, *str2*, n, *lst2*, $i\_$, $n\_$)
∧ (get-nth (j, *lst2*) = 0))
→ ((mc-status (stepn (s, 8)) = 'running)
∧ (mc-pc (stepn (s, 8)) = linked-rts-addr (s))
∧ (read-dn (32, 0, stepn (s, 8)) = *str1*)
∧ mem-lst (1, *str1*, mc-mem (stepn (s, 8)), *n1*, put-nth (0, i, *lst1*))
∧ (read-rn (32, 14, mc-rfile (stepn (s, 8))) = linked-a6 (s))
∧ (read-rn (32, 15, mc-rfile (stepn (s, 8)))
= add (32, read-an (32, 6, s), 8)))

THEOREM: strncat-s0-sn-rfile-base1
(strncat-s1p (s, *i**, i, *str1*, *n1*, *lst1*, *j**, j, *str2*, n, *lst2*, $i\_$, $n\_$)
∧ (get-nth (j, *lst2*) = 0)
∧ (*oplen* ≤ 32)
∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 8)))
= **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (s))
else head (rn-saved (s), *oplen*) **endif**)

THEOREM: strncat-s1-sn-mem-base1
(strncat-s1p (s, *i**, i, *str1*, *n1*, *lst1*, *j**, j, *str2*, n, *lst2*, $i\_$, $n\_$)
∧ (get-nth (j, *lst2*) = 0)
∧ disjoint (x, k, *str1*, *n1*))
→ (read-mem (x, mc-mem (stepn (s, 8)), k) = read-mem (x, mc-mem (s), k))

```
; base case 2: s1 --> sn, when lst2[j] =\= 0, n-1 = 0.
```

THEOREM: strncat-s1-sn-base2
(strncat-s1p (s, *i**, i, *str1*, *n1*, *lst1*, *j**, j, *str2*, n, *lst2*, $i\_$, $n\_$)
∧ (get-nth (j, *lst2*) ≠ 0)
∧ ((n − 1) = 0))
→ ((mc-status (stepn (s, 11)) = 'running)
∧ (mc-pc (stepn (s, 11)) = linked-rts-addr (s))
∧ (read-dn (32, 0, stepn (s, 11)) = *str1*)
∧ mem-lst (1,
str1,
mc-mem (stepn (s, 11)),
n1,

put-nth (0, 1 + i, put-nth (get-nth (j, *lst2*), i, *lst1*)))
∧ (read-rn (32, 14, mc-rfile (stepn (s, 11))) = linked-a6 (s))
∧ (read-rn (32, 15, mc-rfile (stepn (s, 11)))
= add (32, read-an (32, 6, s), 8)))

THEOREM: strncat-s1-sn-rfile-base2
(strncat-s1p (s, i*, i, *str1*, *n1*, *lst1*, j*, j, *str2*, n, *lst2*, i_{-}, n_{-})
∧ (get-nth (j, *lst2*) ≠ 0)
∧ ((n − 1) = 0)
∧ (*oplen* ≤ 32)
∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 11)))
= **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (s))
else head (rn-saved (s), *oplen*) **endif**)

THEOREM: strncat-s1-sn-mem-base2
(strncat-s1p (s, i*, i, *str1*, *n1*, *lst1*, j*, j, *str2*, n, *lst2*, i_{-}, n_{-})
∧ (get-nth (j, *lst2*) ≠ 0)
∧ ((n − 1) = 0)
∧ disjoint (x, k, *str1*, *n1*))
→ (read-mem (x, mc-mem (stepn (s, 11)), k) = read-mem (x, mc-mem (s), k))

```
; induction case: s1 --> s1, when lst2[j] =\= 0, n-1 =\= 0.
```

THEOREM: strncat-s1-s1
(strncat-s1p (s, i*, i, *str1*, *n1*, *lst1*, j*, j, *str2*, n, *lst2*, i_{-}, n_{-})
∧ (get-nth (j, *lst2*) ≠ 0)
∧ ((n − 1) ≠ 0))
→ (strncat-s1p (stepn (s, 6),
add (32, i*, 1),
1 + i,
str1,
n1,
put-nth (get-nth (j, *lst2*), i, *lst1*),
add (32, j*, 1),
1 + j,
str2,
n − 1,
lst2,
i_{-},
n_{-})
∧ (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))
∧ (rn-saved (stepn (s, 6)) = rn-saved (s)))

THEOREM: strncat-s1-s1-rfile
(strncat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n*, *lst2*, *i_*, *n_*)
 ∧ (get-nth (*j*, *lst2*) ≠ 0)
 ∧ ((*n* − 1) ≠ 0)
 ∧ d3-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 6)))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: strncat-s1-s1-mem
(strncat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n*, *lst2*, *i_*, *n_*)
 ∧ (get-nth (*j*, *lst2*) ≠ 0)
 ∧ ((*n* − 1) ≠ 0)
 ∧ disjoint (*x*, *k*, *str1*, *n1*))
→ (read-mem (*x*, mc-mem (stepn (*s*, 6)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; put together.  s1 --> sn.
```

THEOREM: strncat-s1-sn
let *sn* **be** stepn (*s*, strncat-t2 (*j*, *n*, *lst2*))
in
strncat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n*, *lst2*, *i_*, *n_*)
→ ((mc-status (*sn*) = 'running)
 ∧ (mc-pc (*sn*) = linked-rts-addr (*s*))
 ∧ (read-dn (32, 0, *sn*) = *str1*)
 ∧ mem-lst (1, *str1*, mc-mem (*sn*), *n1*, strcpy2 (*i*, *lst1*, *j*, *n*, *lst2*))
 ∧ (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))) **endlet**

THEOREM: strncat-s1-sn-rfile
(strncat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n*, *lst2*, *i_*, *n_*)
 ∧ (*oplen* ≤ 32)
 ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, strncat-t2 (*j*, *n*, *lst2*))))
 = **if** d3-7a2-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
 else head (rn-saved (*s*), *oplen*) **endif**)

THEOREM: strncat-s1-sn-mem
(strncat-s1p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *j**, *j*, *str2*, *n*, *lst2*, *i_*, *n_*)
 ∧ disjoint (*x*, *k*, *str1*, *n1*))
→ (read-mem (*x*, mc-mem (stepn (*s*, strncat-t2 (*j*, *n*, *lst2*))), *k*)
 = read-mem (*x*, mc-mem (*s*), *k*))

```
; the correctness of strncat.
```

THEOREM: strncat-correctness
let *sn* **be** stepn (*s*, strncat-t (*n1*, *lst1*, *n*, *lst2*))
in
strncat-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n*, *lst2*)
→ ((mc-status (*sn*) = `'running`)
∧ (mc-pc (*sn*) = rts-addr (*s*))
∧ (read-an (32, 6, *sn*) = read-an (32, 6, *s*))
∧ (read-an (32, 7, *sn*) = add (32, read-an (32, 7, *s*), 4))
∧ ((d2-7a2-5p (*rn*) ∧ (*oplen* ≤ 32))
→ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))))
∧ ((disjoint (*x*, *k*, sub (32, 8, read-sp (*s*)), 24)
∧ disjoint (*x*, *k*, *str1*, *n1*))
→ (read-mem (*x*, mc-mem (*sn*), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))
∧ (read-dn (32, 0, *sn*) = *str1*)
∧ mem-lst (1, *str1*, mc-mem (*sn*), *n1*, strncat (*n1*, *lst1*, *n*, *lst2*))) **endlet**

EVENT: Disable strncat-t.

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

Index