```
#|

 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 "strncmp" using the compiled version.

```
;          Proof of the Correctness of the STRSTR 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 strstr function in the Berkeley string library.

/* find pointer to first occurrence of find[] in s[] */
char *
strstr(s, find)
register const char *s, *find;
{
register char c, sc;
register size_t len;

if ((c = *find++) != 0) {
```

```
len = strlen(find);
do {
do {
if ((sc = *s++) == 0)
return (NULL);
} while (sc != c);
} while (strncmp(s, find, len) != 0);
s--;
}
return ((char *)s);
}
```

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

```
0x2718 <strstr>:        linkw fp,#0
0x271c <strstr+4>:      moveml d2-d3/a2-a3,sp@-
0x2720 <strstr+8>:      moveal fp@(8),a2
0x2724 <strstr+12>:     moveal fp@(12),a3
0x2728 <strstr+16>:     moveb a3@+,d2
0x272a <strstr+18>:     beq 0x275a <strstr+66>
0x272c <strstr+20>:     movel a3,sp@-
0x272e <strstr+22>:     jsr @#0x25b0 <strlen>
0x2734 <strstr+28>:     movel d0,d3
0x2736 <strstr+30>:     addqw #4,sp
0x2738 <strstr+32>:     moveb a2@+,d0
0x273a <strstr+34>:     bne 0x2740 <strstr+40>
0x273c <strstr+36>:     clrl d0
0x273e <strstr+38>:     bra 0x275c <strstr+68>
0x2740 <strstr+40>:     cmpb d0,d2
0x2742 <strstr+42>:     bne 0x2738 <strstr+32>
0x2744 <strstr+44>:     movel d3,sp@-
0x2746 <strstr+46>:     movel a3,sp@-
0x2748 <strstr+48>:     movel a2,sp@-
0x274a <strstr+50>:     jsr @#0x2608 <strncmp>
0x2750 <strstr+56>:     addaw #12,sp
0x2754 <strstr+60>:     tstl d0
0x2756 <strstr+62>:     bne 0x2738 <strstr+32>
0x2758 <strstr+64>:     subqw #1,a2
0x275a <strstr+66>:     movel a2,d0
0x275c <strstr+68>:     moveml fp@(-16),d2-d3/a2-a3
0x2762 <strstr+74>:     unlk fp
0x2764 <strstr+76>:     rts
```

```
The machine code of the above program is:

<strstr>:      0x4e56  0x0000  0x48e7  0x3030  0x246e  0x0008  0x266e  0x000c
<strstr+16>:   0x141b  0x672e  0x2f0b  0x4eb9  0x0000  0x25b0  0x2600  0x584f
<strstr+32>:   0x101a  0x6604  0x4280  0x601c  0xb400  0x66f4  0x2f03  0x2f0b
<strstr+48>:   0x2f0a  0x4eb9  0x0000  0x2608  0xdefc  0x000c  0x4a80  0x66e0
<strstr+64>:   0x534a  0x200a  0x4cee  0x0c0c  0xfff0  0x4e5e  0x4e75

'(78      86      0       0       72      231     48      48
  36      110     0       8       38      110     0       12
  20      27      103     46      47      11      78      185
  0       0       37      176     38      0       88      79
  16      26      102     4       66      128     96      28
  180     0       102     244     47      3       47      11
  47      10      78      185     0       0       38      8
  222     252     0       12      74      128     102     224
  83      74      32      10      76      238     12      12
  255     240     78      94      78      117)
|#

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

DEFINITION:
STRSTR-CODE
= '(78 86 0 0 72 231 48 48 36 110 0 8 38 110 0 12 20 27
103 46 47 11 78 185 -1 -1 -1 -1 38 0 88 79 16 26 102
4 66 128 96 28 180 0 102 244 47 3 47 11 47 10 78 185
-1 -1 -1 -1 222 252 0 12 74 128 102 224 83 74 32 10
76 238 12 12 255 240 78 94 78 117)

CONSERVATIVE AXIOM: strstr-load
strstr-loadp (s)
= (evenp (STRSTR-ADDR)
∧ (STRSTR-ADDR ∈ $\mathbf{N}$)
∧ nat-rangep (STRSTR-ADDR, 32)
∧ rom-addrp (STRSTR-ADDR, mc-mem (s), 78)
∧ mcode-addrp (STRSTR-ADDR, mc-mem (s), STRSTR-CODE)
∧ strlen-loadp (s)
∧ strncmp-loadp (s)
∧ (pc-read-mem (add (32, STRSTR-ADDR, 24), mc-mem (s), 4)
= STRLEN-ADDR)
∧ (pc-read-mem (add (32, STRSTR-ADDR, 52), mc-mem (s), 4)
= STRNCMP-ADDR))

Simultaneously, we introduce the new function symbols *strstr-loadp* and *strstr-*

addr.

THEOREM: stepn-strstr-loadp
strstr-loadp (stepn (*s*, *n*)) = strstr-loadp (*s*)

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

DEFINITION:
strstr-t0 (*n2*, *lst2*) = splus (8, splus (strlen-t (*n2* − 1, cdr (*lst2*)), 2))

DEFINITION:
strstr-t1 (*i*, *n1*, *lst1*, *lst2*)
= **if** *i* < *n1*
then if get-nth (*i*, *lst1*) = 0 **then** 7
elseif get-nth (*i*, *lst1*) = get-nth (0, *lst2*) **then** 8
else splus (4, strstr-t1 (1 + *i*, *n1*, *lst1*, *lst2*)) **endif**
else 0 **endif**

DEFINITION:
strstr-t2 (*i*, *n1*, *lst1*, *lst2*, *len*)
= **let** *j* **be** strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*))
in
if *j* ∈ **N**
then if strncmp (*len*, mcdr (1 + *j*, *lst1*), mcdr (1, *lst2*)) = 0
then splus (strstr-t1 (*i*, *n1*, *lst1*, *lst2*),
splus (strncmp-t (*len*,
mcdr (1 + *j*, *lst1*),
mcdr (1, *lst2*)),
8))
else splus (strstr-t1 (*i*, *n1*, *lst1*, *lst2*),
splus (strncmp-t (*len*,
mcdr (1 + *j*, *lst1*),
mcdr (1, *lst2*)),
3)) **endif**
else strstr-t1 (*i*, *n1*, *lst1*, *lst2*) **endif endlet**

DEFINITION:
strstr-t3 (*i*, *n1*, *lst1*, *lst2*, *len*)
= **if** *i* < *n1*
then let *j* **be** strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*))
in
if *j* ∈ **N**
then if strncmp (*len*, mcdr (1 + *j*, *lst1*), mcdr (1, *lst2*))
= 0 **then** strstr-t2 (*i*, *n1*, *lst1*, *lst2*, *len*)
else splus (strstr-t2 (*i*, *n1*, *lst1*, *lst2*, *len*),

```
                                strstr-t3 (1 + j, n1, lst1, lst2, len)) endif
          else strstr-t1 (i, n1, lst1, lst2) endif endlet
     else 0 endif

DEFINITION:
strstr-t (n1, lst1, n2, lst2)
=    if get-nth (0, lst2) = 0 then 10
     else splus (strstr-t0 (n2, lst2),
                 strstr-t3 (0,
                            n1,
                            lst1,
                            lst2,
                            strlen (0, n2 - 1, mcdr (1, lst2)))) endif

; two induction hints.

DEFINITION:
strstr-induct1 (s, i*, i, n1, lst1, lst2)
=    if i < n1
     then if get-nth (i, lst1) = 0 then t
          elseif get-nth (i, lst1) = get-nth (0, lst2) then t
          else strstr-induct1 (stepn (s, 4),
                               add (32, i*, 1),
                               1 + i,
                               n1,
                               lst1,
                               lst2) endif
     else t endif

DEFINITION:
strstr-induct2 (s, i*, i, n1, lst1, lst2, len)
=    if i < n1
     then let j* be strchr1* (i*, i, n1, lst1, get-nth (0, lst2)),
              j  be strchr1 (i, n1, lst1, get-nth (0, lst2))
          in
          if j ∈ N
          then if strncmp (len, mcdr (1 + j, lst1), mcdr (1, lst2))
                    = 0 then t
               else strstr-induct2 (stepn (s,
                                           strstr-t2 (i,
                                                      n1,
                                                      lst1,
                                                      lst2,
                                                      len)),
                                    add (32, j*, 1),
```

$1 + j$,
n1,
lst1,
lst2,
len) **endif**
else t endif endlet
else t endif

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

DEFINITION:
strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
∧ strstr-loadp (*s*)
∧ (mc-pc (*s*) = STRSTR-ADDR)
∧ ram-addrp (sub (32, 48, read-sp (*s*)), mc-mem (*s*), 60)
∧ ram-addrp (*str1*, mc-mem (*s*), *n1*)
∧ mem-lst (1, *str1*, mc-mem (*s*), *n1*, *lst1*)
∧ ram-addrp (*str2*, mc-mem (*s*), *n2*)
∧ mem-lst (1, *str2*, mc-mem (*s*), *n2*, *lst2*)
∧ disjoint (*str1*, *n1*, sub (32, 48, read-sp (*s*)), 60)
∧ disjoint (*str2*, *n2*, sub (32, 48, read-sp (*s*)), 60)
∧ (*str1* = read-mem (add (32, read-sp (*s*), 4), mc-mem (*s*), 4))
∧ (*str2* = read-mem (add (32, read-sp (*s*), 8), mc-mem (*s*), 4))
∧ (slen (0, *n1*, *lst1*) < *n1*)
∧ (slen (0, *n2*, *lst2*) < *n2*)
∧ (nat-to-uint (*str1*) ≠ 0)
∧ (*n1* ≯ 0)
∧ uint-rangep (nat-to-uint (*str1*) + *n1*, 32)
∧ (*n2* ≯ 0)
∧ uint-rangep (*n2*, 32))

```
; the intermediate state right before the call to strlen.
```

DEFINITION:
strstr-s0p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
= ((mc-status (*s*) = 'running)
∧ strstr-loadp (*s*)
∧ (mc-pc (*s*) = STRLEN-ADDR)
∧ (rts-addr (*s*) = add (32, STRSTR-ADDR, 28))
∧ ram-addrp (sub (32, 44, read-an (32, 6, *s*)), mc-mem (*s*), 60)
∧ ram-addrp (*str1*, mc-mem (*s*), *n1*)
∧ mem-lst (1, *str1*, mc-mem (*s*), *n1*, *lst1*)
∧ ram-addrp (*str2*, mc-mem (*s*), *n2*)
∧ mem-lst (1, *str2*, mc-mem (*s*), *n2*, *lst2*)

$\wedge$ disjoint (*str1*, *n1*, sub (32, 44, read-an (32, 6, *s*)), 60)
$\wedge$ disjoint (*str2*, *n2*, sub (32, 44, read-an (32, 6, *s*)), 60)
$\wedge$ (*str1* = read-an (32, 2, *s*))
$\wedge$ (nat-to-uint (read-dn (8, 2, *s*)) = get-nth (0, *lst2*))
$\wedge$ equal* (read-an (32, 3, *s*), add (32, *str2*, 1))
$\wedge$ equal* (read-sp (*s*), sub (32, 24, read-an (32, 6, *s*)))
$\wedge$ (read-an (32, 3, *s*)
= read-mem (add (32, read-sp (*s*), 4), mc-mem (*s*), 4))
$\wedge$ (slen (0, *n1*, *lst1*) < *n1*)
$\wedge$ (slen (1, *n2*, *lst2*) < *n2*)
$\wedge$ (*n1* $\not\simeq$ 0)
$\wedge$ uint-rangep (*n1*, 32)
$\wedge$ (*n2* $\not\simeq$ 0)
$\wedge$ uint-rangep (*n2*, 32))

```
; the intermediate state returned from the call to strlen.
```

DEFINITION:
strstr-s1p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
= ((mc-status (*s*) = 'running)
$\wedge$ strstr-loadp (*s*)
$\wedge$ (mc-pc (*s*) = add (32, STRSTR-ADDR, 28))
$\wedge$ ram-addrp (sub (32, 44, read-an (32, 6, *s*)), mc-mem (*s*), 60)
$\wedge$ ram-addrp (*str1*, mc-mem (*s*), *n1*)
$\wedge$ mem-lst (1, *str1*, mc-mem (*s*), *n1*, *lst1*)
$\wedge$ ram-addrp (*str2*, mc-mem (*s*), *n2*)
$\wedge$ mem-lst (1, *str2*, mc-mem (*s*), *n2*, *lst2*)
$\wedge$ disjoint (*str1*, *n1*, sub (32, 44, read-an (32, 6, *s*)), 60)
$\wedge$ disjoint (*str2*, *n2*, sub (32, 44, read-an (32, 6, *s*)), 60)
$\wedge$ (*str1* = read-an (32, 2, *s*))
$\wedge$ (nat-to-uint (read-dn (8, 2, *s*)) = get-nth (0, *lst2*))
$\wedge$ equal* (read-an (32, 3, *s*), add (32, *str2*, 1))
$\wedge$ equal* (read-sp (*s*), sub (32, 20, read-an (32, 6, *s*)))
$\wedge$ (*len* = uread-dn (32, 0, *s*))
$\wedge$ (slen (0, *n1*, *lst1*) < *n1*)
$\wedge$ (slen (1, *n2*, *lst2*) < *n2*)
$\wedge$ (*n1* $\in$ **N**)
$\wedge$ uint-rangep (*n1*, 32)
$\wedge$ (*n2* $\in$ **N**)
$\wedge$ uint-rangep (*n2*, 32))

```
; the intermediate state right before the outer loop.
```

DEFINITION:
strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)

$=$ ((mc-status (s) = `'running`)
$\wedge$ strstr-loadp (s)
$\wedge$ (mc-pc (s) = add (32, STRSTR-ADDR, 32))
$\wedge$ ram-addrp (sub (32, 44, read-an (32, 6, s)), mc-mem (s), 60)
$\wedge$ ram-addrp ($str1$, mc-mem (s), $n1$)
$\wedge$ mem-lst (1, $str1$, mc-mem (s), $n1$, $lst1$)
$\wedge$ ram-addrp ($str2$, mc-mem (s), $n2$)
$\wedge$ mem-lst (1, $str2$, mc-mem (s), $n2$, $lst2$)
$\wedge$ disjoint ($str1$, $n1$, sub (32, 44, read-an (32, 6, s)), 60)
$\wedge$ disjoint ($str2$, $n2$, sub (32, 44, read-an (32, 6, s)), 60)
$\wedge$ (uread-dn (8, 2, s) = get-nth (0, $lst2$))
$\wedge$ equal* (read-an (32, 2, s), add (32, $str1$, $i*$))
$\wedge$ equal* (read-an (32, 3, s), add (32, $str2$, 1))
$\wedge$ equal* (read-sp (s), sub (32, 16, read-an (32, 6, s)))
$\wedge$ (len = uread-dn (32, 3, s))
$\wedge$ (slen (i, $n1$, $lst1$) $<$ $n1$)
$\wedge$ (slen (1, $n2$, $lst2$) $<$ $n2$)
$\wedge$ ($i* \in \mathbf{N}$)
$\wedge$ nat-rangep ($i*$, 32)
$\wedge$ (i = nat-to-uint ($i*$))
$\wedge$ ($n1 \not\simeq 0$)
$\wedge$ uint-rangep ($n1$, 32)
$\wedge$ ($n2 \not\simeq 0$)
$\wedge$ uint-rangep ($n2$, 32))

`; the intermediate state right before the call to strncmp.`

DEFINITION:
strstr-s3p (s, $i*$, i, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, len)
$=$ ((mc-status (s) = `'running`)
$\wedge$ strstr-loadp (s)
$\wedge$ (mc-pc (s) = STRNCMP-ADDR)
$\wedge$ (rts-addr (s) = add (32, STRSTR-ADDR, 56))
$\wedge$ ram-addrp (sub (32, 44, read-an (32, 6, s)), mc-mem (s), 60)
$\wedge$ ram-addrp ($str1$, mc-mem (s), $n1$)
$\wedge$ mem-lst (1, $str1$, mc-mem (s), $n1$, $lst1$)
$\wedge$ ram-addrp ($str2$, mc-mem (s), $n2$)
$\wedge$ mem-lst (1, $str2$, mc-mem (s), $n2$, $lst2$)
$\wedge$ disjoint (sub (32, 44, read-an (32, 6, s)), 60, $str1$, $n1$)
$\wedge$ disjoint (sub (32, 44, read-an (32, 6, s)), 60, $str2$, $n2$)
$\wedge$ (uread-dn (8, 2, s) = get-nth (0, $lst2$))
$\wedge$ equal* (read-an (32, 2, s), add (32, $str1$, $i*$))
$\wedge$ equal* (read-an (32, 3, s), add (32, $str2$, 1))
$\wedge$ equal* (read-sp (s), sub (32, 32, read-an (32, 6, s)))

$\wedge$ (*len* = uread-dn (32, 3, *s*))
$\wedge$ (read-an (32, 2, *s*)
= read-mem (add (32, read-sp (*s*), 4), mc-mem (*s*), 4))
$\wedge$ (read-an (32, 3, *s*)
= read-mem (add (32, read-sp (*s*), 8), mc-mem (*s*), 4))
$\wedge$ (read-dn (32, 3, *s*)
= read-mem (add (32, read-sp (*s*), 12), mc-mem (*s*), 4))
$\wedge$ (slen (*i*, *n1*, *lst1*) < *n1*)
$\wedge$ (slen (1, *n2*, *lst2*) < *n2*)
$\wedge$ (*i** $\in$ **N**)
$\wedge$ nat-rangep (*i**, 32)
$\wedge$ (*i* = nat-to-uint (*i**))
$\wedge$ (*n1* $\not\simeq$ 0)
$\wedge$ uint-rangep (*n1*, 32)
$\wedge$ (*n2* $\not\simeq$ 0)
$\wedge$ uint-rangep (*n2*, 32))

```
; the intermediate state right after the call to strncmp.
```

DEFINITION:
strstr-s4p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
= ((mc-status (*s*) = 'running)
$\wedge$ strstr-loadp (*s*)
$\wedge$ (mc-pc (*s*) = add (32, STRSTR-ADDR, 56))
$\wedge$ ram-addrp (sub (32, 44, read-an (32, 6, *s*)), mc-mem (*s*), 60)
$\wedge$ ram-addrp (*str1*, mc-mem (*s*), *n1*)
$\wedge$ mem-lst (1, *str1*, mc-mem (*s*), *n1*, *lst1*)
$\wedge$ ram-addrp (*str2*, mc-mem (*s*), *n2*)
$\wedge$ mem-lst (1, *str2*, mc-mem (*s*), *n2*, *lst2*)
$\wedge$ disjoint (*str1*, *n1*, sub (32, 44, read-an (32, 6, *s*)), 60)
$\wedge$ disjoint (*str2*, *n2*, sub (32, 44, read-an (32, 6, *s*)), 60)
$\wedge$ (iread-dn (32, 0, *s*)
= strncmp (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)))
$\wedge$ (uread-dn (8, 2, *s*) = get-nth (0, *lst2*))
$\wedge$ equal* (read-an (32, 2, *s*), add (32, *str1*, *i**))
$\wedge$ equal* (read-an (32, 3, *s*), add (32, *str2*, 1))
$\wedge$ equal* (read-sp (*s*), sub (32, 28, read-an (32, 6, *s*)))
$\wedge$ (*len* = uread-dn (32, 3, *s*))
$\wedge$ (slen (*i*, *n1*, *lst1*) < *n1*)
$\wedge$ (slen (1, *n2*, *lst2*) < *n2*)
$\wedge$ (*i** $\in$ **N**)
$\wedge$ nat-rangep (*i**, 32)
$\wedge$ (*i* = nat-to-uint (*i**))
$\wedge$ (*n1* $\not\simeq$ 0)

$\wedge$ uint-rangep ($n1$, 32)
$\wedge$ ($n2 \not\simeq$ 0)
$\wedge$ uint-rangep ($n2$, 32))

```
; from the initial state to exit:  s --> sn, when lst2[0] == 0.
```

THEOREM: strstr-s-sn
(strstr-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$) $\wedge$ (get-nth (0, $lst2$) = 0))
$\rightarrow$ ((mc-status (stepn (s, 10)) = 'running)
$\wedge$ (mc-pc (stepn (s, 10)) = rts-addr (s))
$\wedge$ (read-dn (32, 0, stepn (s, 10)) = $str1$)
$\wedge$ (read-rn (32, 15, mc-rfile (stepn (s, 10)))
= add (32, read-an (32, 7, s), 4))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 10))) = read-an (32, 6, s)))

THEOREM: strstr-s-sn-rfile
(strstr-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\wedge$ (get-nth (0, $lst2$) = 0)
$\wedge$ ($oplen \leq$ 32)
$\wedge$ d2-7a2-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile (stepn (s, 10)))
= read-rn ($oplen$, rn, mc-rfile (s)))

THEOREM: strstr-s-sn-mem
(strstr-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)
$\wedge$ (get-nth (0, $lst2$) = 0)
$\wedge$ disjoint (x, k, sub (32, 48, read-sp (s)), 60))
$\rightarrow$ (read-mem (x, mc-mem (stepn (s, 10)), k) = read-mem (x, mc-mem (s), k))

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

THEOREM: strstr-s-s0
(strstr-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$) $\wedge$ (get-nth (0, $lst2$) $\neq$ 0))
$\rightarrow$ strstr-s0p (stepn (s, 8), $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$)

THEOREM: strstr-s-s0-else
(strstr-statep (s, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$) $\wedge$ (get-nth (0, $lst2$) $\neq$ 0))
$\rightarrow$ ((linked-rts-addr (stepn (s, 8)) = rts-addr (s))
$\wedge$ (linked-a6 (stepn (s, 8)) = read-an (32, 6, s))
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (s, 8)))
= sub (32, 4, read-sp (s)))
$\wedge$ (movem-saved (stepn (s, 8), 4, 16, 4)
= readm-rn (32, '(2 3 10 11), mc-rfile (s))))

THEOREM: strstr-s-s0-rfile

(strstr-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (0, *lst2*) ≠ 0)
 ∧ d4-7a4-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (s, 8)))
 = read-rn (*oplen*, *rn*, mc-rfile (s)))

THEOREM: strstr-s-s0-mem
(strstr-statep (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
 ∧ (get-nth (0, *lst2*) ≠ 0)
 ∧ disjoint (x, k, sub (32, 48, read-sp (s)), 60))
→ (read-mem (x, mc-mem (stepn (s, 8)), k) = read-mem (x, mc-mem (s), k))

```
; from s0 to s1:  s0 --> s1.  by strlen.
```

THEOREM: strstr-s0p-strlen-statep
strstr-s0p (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ strlen-statep (s, add (32, *str2*, 1), *n2* − 1, mcdr (1, *lst2*))

THEOREM: strstr-s0-s1
let *s1* **be** stepn (s, strlen-t (*n2* − 1, mcdr (1, *lst2*)))
in
strstr-s0p (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ strstr-s1p (*s1*,
 str1,
 n1,
 lst1,
 str2,
 n2,
 lst2,
 strlen (0, *n2* − 1, mcdr (1, *lst2*))) **endlet**

THEOREM: strstr-s0-s1-else
let *s1* **be** stepn (s, strlen-t (*n2* − 1, cdr (*lst2*)))
in
strstr-s0p (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ ((read-rn (32, 14, mc-rfile (*s1*))
 = read-rn (32, 14, mc-rfile (s)))
 ∧ (linked-rts-addr (*s1*) = linked-rts-addr (s))
 ∧ (linked-a6 (*s1*) = linked-a6 (s))
 ∧ (movem-saved (*s1*, 4, 16, 4) = movem-saved (s, 4, 16, 4))) **endlet**

THEOREM: strstr-s0-s1-rfile
let *s1* **be** stepn (s, strlen-t (*n2* − 1, cdr (*lst2*)))
in
(strstr-s0p (s, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*) ∧ d2-7a2-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (*s1*))
 = read-rn (*oplen*, *rn*, mc-rfile (s))) **endlet**

THEOREM: strstr-s0-s1-mem
let *s1* **be** stepn (*s*, strlen-t (*n2* − 1, cdr (*lst2*)))
in
(strstr-s0p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ disjoint (*x*, *k*, sub (32, 44, read-an (32, 6, *s*)), 60))
→ (read-mem (*x*, mc-mem (*s1*), *k*) = read-mem (*x*, mc-mem (*s*), *k*)) **endlet**

```
; from s1 to s2:  s1 --> s2.
```

THEOREM: strstr-s1-s2
strstr-s1p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
→ strstr-s2p (stepn (*s*, 2), 0, 0, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)

THEOREM: strstr-s1-s2-else
strstr-s1p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
→ ((linked-rts-addr (stepn (*s*, 2)) = linked-rts-addr (*s*))
∧ (linked-a6 (stepn (*s*, 2)) = linked-a6 (*s*))
∧ (read-rn (*oplen*, 14, mc-rfile (stepn (*s*, 2)))
= read-rn (*oplen*, 14, mc-rfile (*s*)))
∧ (movem-saved (stepn (*s*, 2), 4, 16, 4) = movem-saved (*s*, 4, 16, 4))
∧ (read-mem (*x*, mc-mem (stepn (*s*, 2)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strstr-s1-s2-rfile
(strstr-s1p (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*) ∧ d4-7a4-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 2)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; from s2 to exit:  s2 --> sn, when lst1[i] == 0.
```

THEOREM: strstr-s2-sn-base
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ (get-nth (*i*, *lst1*) = 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: strstr-s2-sn-base-rfile
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ (get-nth (*i*, *lst1*) = 0)

$\wedge$ d2-7a2-5p (*rn*)
$\wedge$ (*oplen* $\leq$ 32))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 7)))
= **if** d4-7a4-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
else get-vlst (*oplen*,
0,
rn,
'(2 3 10 11),
movem-saved (*s*, 4, 16, 4)) **endif**)

```
; from s2 to s3:  s2 --> s3, when lst1[i] =\= 0 and lst1[i] == lst2[0].
```

THEOREM: strstr-s2-s3-base
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
$\wedge$ (get-nth (*i*, *lst1*) $\neq$ 0)
$\wedge$ (get-nth (*i*, *lst1*) = get-nth (0, *lst2*)))
$\rightarrow$ (strstr-s3p (stepn (*s*, 8),
add (32, *i**, 1),
1 + *i*,
str1,
n1,
lst1,
str2,
n2,
lst2,
len)
$\wedge$ (read-rn (32, 14, mc-rfile (stepn (*s*, 8)))
= read-rn (32, 14, mc-rfile (*s*)))
$\wedge$ (linked-a6 (stepn (*s*, 8)) = linked-a6 (*s*))
$\wedge$ (linked-rts-addr (stepn (*s*, 8)) = linked-rts-addr (*s*))
$\wedge$ (movem-saved (stepn (*s*, 8), 4, 16, 4) = movem-saved (*s*, 4, 16, 4)))

THEOREM: strstr-s2-s3-base-rfile
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
$\wedge$ (get-nth (*i*, *lst1*) $\neq$ 0)
$\wedge$ (get-nth (*i*, *lst1*) = get-nth (0, *lst2*))
$\wedge$ d4-7a4-5p (*rn*))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 8)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

THEOREM: strstr-s2-s3-base-mem
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
$\wedge$ (get-nth (*i*, *lst1*) $\neq$ 0)
$\wedge$ (get-nth (*i*, *lst1*) = get-nth (0, *lst2*))
$\wedge$ disjoint (*x*, *k*, sub (32, 44, read-an (32, 6, *s*)), 60))
$\rightarrow$ (read-mem (*x*, mc-mem (stepn (*s*, 8)), *k*) = read-mem (*x*, mc-mem (*s*), *k*))

```
; from s2 to s2:  s2 --> s2.
```

THEOREM: strstr-s2-s2
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
 ∧ (get-nth (*i*, *lst1*) ≠ 0)
 ∧ (get-nth (*i*, *lst1*) ≠ get-nth (0, *lst2*)))
→ (strstr-s2p (stepn (*s*, 4),
add (32, *i**, 1),
1 + *i*,
str1,
n1,
lst1,
str2,
n2,
lst2,
len)
∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 4)))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-rts-addr (stepn (*s*, 4)) = linked-rts-addr (*s*))
∧ (rts-addr (stepn (*s*, 4)) = rts-addr (*s*))
∧ (movem-saved (stepn (*s*, 4), 4, 16, 4) = movem-saved (*s*, 4, 16, 4))
∧ (read-mem (*x*, mc-mem (stepn (*s*, 4)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strstr-s2-s2-rfile
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
 ∧ (get-nth (*i*, *lst1*) ≠ 0)
 ∧ (get-nth (*i*, *lst1*) ≠ get-nth (0, *lst2*))
 ∧ d4-7a4-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 4)))
= read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; from s3 to s4:  s3 --> s4. The call to strncmp.
```

THEOREM: strstr-s3p-strncmp-statep
strstr-s3p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
→ strncmp-statep (*s*,
add (32, *str1*, *i**),
n1 − *i*,
mcdr (*i*, *lst1*),
add (32, *str2*, 1),
n2 − 1,
mcdr (1, *lst2*),
len)

THEOREM: strstr-s3-s4
let *s4* **be** stepn (*s*, strncmp-t (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)))
in
strstr-s3p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
→ strstr-s4p (*s4*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*) **endlet**

THEOREM: strstr-s3-s4-else
let *s4* **be** stepn (*s*, strncmp-t (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)))
in
strstr-s3p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
→ ((read-rn (32, 14, mc-rfile (*s4*))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-rts-addr (*s4*) = linked-rts-addr (*s*))
∧ (linked-a6 (*s4*) = linked-a6 (*s*))
∧ (movem-saved (*s4*, 4, 16, 4) = movem-saved (*s*, 4, 16, 4))) **endlet**

THEOREM: strstr-s3-s4-rfile
let *s4* **be** stepn (*s*, strncmp-t (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)))
in
(strstr-s3p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ d2-7a2-5p (*rn*)
∧ (*oplen* ≤ 32))
→ (read-rn (*oplen*, *rn*, mc-rfile (*s4*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))) **endlet**

THEOREM: strstr-s3-s4-mem
let *s4* **be** stepn (*s*, strncmp-t (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)))
in
(strstr-s3p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ disjoint (*x*, *k*, sub (32, 44, read-an (32, 6, *s*)), 60))
→ (read-mem (*x*, mc-mem (*s4*), *k*) = read-mem (*x*, mc-mem (*s*), *k*)) **endlet**

```
; from s4 to exit:  s4 --> sn, when strncmp == 0.
```

THEOREM: strstr-s4-sn
(strstr-s4p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ (strncmp (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)) = 0))
→ ((mc-status (stepn (*s*, 8)) = 'running)
∧ (mc-pc (stepn (*s*, 8)) = linked-rts-addr (*s*))
∧ (read-dn (32, 0, stepn (*s*, 8)) = add (32, *str1*, sub (32, 1, *i**)))
∧ (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))
∧ (read-mem (*x*, mc-mem (stepn (*s*, 8)), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strstr-s4-sn-rfile
(strstr-s4p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
 ∧ (strncmp (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)) = 0)
 ∧ d2-7a2-5p (*rn*)
 ∧ (*oplen* ≤ 32))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 8)))
 = **if** d4-7a4-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
 else get-vlst (*oplen*,
 0,
 rn,
 '(2 3 10 11),
 movem-saved (*s*, 4, 16, 4)) **endif**)

```
; from s4 to s2: s4 --> s2, when strncmp =\= 0.
```

THEOREM: strstr-s4-s2
(strstr-s4p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
 ∧ (strncmp (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)) ≠ 0))
→ (strstr-s2p (stepn (*s*, 3), *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
 ∧ (read-rn (32, 14, mc-rfile (stepn (*s*, 3)))
 = read-rn (32, 14, mc-rfile (*s*)))
 ∧ (linked-rts-addr (stepn (*s*, 3)) = linked-rts-addr (*s*))
 ∧ (linked-a6 (stepn (*s*, 3)) = linked-a6 (*s*))
 ∧ (movem-saved (stepn (*s*, 3), 4, 16, 4) = movem-saved (*s*, 4, 16, 4))
 ∧ (read-mem (*x*, mc-mem (stepn (*s*, 3)), *k*)
 = read-mem (*x*, mc-mem (*s*), *k*)))

THEOREM: strstr-s4-s2-rfile
(strstr-s4p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
 ∧ (strncmp (*len*, mcdr (*i*, *lst1*), mcdr (1, *lst2*)) ≠ 0)
 ∧ d2-7a4-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (stepn (*s*, 3)))
 = read-rn (*oplen*, *rn*, mc-rfile (*s*)))

```
; put together: s2 --> s3.
```

THEOREM: strstr-s2p-info
strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
→ ((*i* ∈ **N**) ∧ ((*i* < *n1*) = **t**))

THEOREM: strstr-s2-s3
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
 ∧ (strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*)) ∈ **N**))
→ strstr-s3p (stepn (*s*, strstr-t1 (*i*, *n1*, *lst1*, *lst2*)),
 add (32, strchr1* (*i**, *i*, *n1*, *lst1*, get-nth (0, *lst2*)), 1),

1 + strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*)),
str1,
n1,
lst1,
str2,
n2,
lst2,
len)

THEOREM: strstr-s2-s3-else
let *s3* **be** stepn (*s*, strstr-t1 (*i*, *n1*, *lst1*, *lst2*))
in
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ (strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*)) ∈ **N**))
→ ((read-rn (32, 14, mc-rfile (*s3*))
= read-rn (32, 14, mc-rfile (*s*)))
∧ (linked-rts-addr (*s3*) = linked-rts-addr (*s*))
∧ (linked-a6 (*s3*) = linked-a6 (*s*))
∧ (movem-saved (*s3*, 4, 16, 4) = movem-saved (*s*, 4, 16, 4))) **endlet**

THEOREM: strstr-s2-s3-rfile
let *s3* **be** stepn (*s*, strstr-t1 (*i*, *n1*, *lst1*, *lst2*))
in
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ (strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*)) ∈ **N**)
∧ d4-7a4-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (*s3*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))) **endlet**

THEOREM: strstr-s2-s3-mem
let *s3* **be** stepn (*s*, strstr-t1 (*i*, *n1*, *lst1*, *lst2*))
in
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ (strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*)) ∈ **N**)
∧ disjoint (*x*, *k*, sub (32, 44, read-an (32, 6, *s*)), 60))
→ (read-mem (*x*, mc-mem (*s3*), *k*) = read-mem (*x*, mc-mem (*s*), *k*)) **endlet**

```
; put together: s2 --> exit.
```

THEOREM: strstr-s2-sn
let *sn* **be** stepn (*s*, strstr-t1 (*i*, *n1*, *lst1*, *lst2*))
in
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ (strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*)) ∉ **N**))
→ ((mc-status (*sn*) = 'running)

$\wedge$ (mc-pc (*sn*) = linked-rts-addr (*s*))
$\wedge$ (read-dn (32, 0, *sn*) = 0)
$\wedge$ (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))
$\wedge$ (read-mem (*x*, mc-mem (*sn*), *k*) = read-mem (*x*, mc-mem (*s*), *k*))) **endlet**

THEOREM: strstr-s2-sn-rfile
let *sn* **be** stepn (*s*, strstr-t1 (*i*, *n1*, *lst1*, *lst2*))
in
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
$\wedge$ (strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*)) $\notin$ **N**)
$\wedge$ d2-7a2-5p (*rn*)
$\wedge$ (*oplen* $\leq$ 32))
$\rightarrow$ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= **if** d4-7a4-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
else get-vlst (*oplen*,
0,
rn,
'(2 3 10 11),
movem-saved (*s*, 4, 16, 4)) **endif**) **endlet**

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

THEOREM: strstr-s-s2
let *s2* **be** stepn (*s*, strstr-t0 (*n2*, *lst2*))
in
(strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ (get-nth (0, *lst2*) $\neq$ 0))
$\rightarrow$ strstr-s2p (*s2*,
0,
0,
str1,
n1,
lst1,
str2,
n2,
lst2,
strlen (0, *n2* $-$ 1, mcdr (1, *lst2*))) **endlet**

THEOREM: strstr-s-s2-else
let *s2* **be** stepn (*s*, strstr-t0 (*n2*, *lst2*))
in
(strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ (get-nth (0, *lst2*) $\neq$ 0))

→ ((linked-rts-addr (*s2*) = rts-addr (*s*))
∧ (linked-a6 (*s2*) = read-an (32, 6, *s*))
∧ (read-rn (32, 14, mc-rfile (*s2*))
= sub (32, 4, read-sp (*s*)))
∧ (movem-saved (*s2*, 4, 16, 4)
= readm-rn (32, '(2 3 10 11), mc-rfile (*s*)))) **endlet**

THEOREM: strstr-s-s2-rfile
let *s2* **be** stepn (*s*, strstr-t0 (*n2*, *lst2*))
in
(strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ (get-nth (0, *lst2*) ≠ 0)
∧ d4-7a4-5p (*rn*))
→ (read-rn (*oplen*, *rn*, mc-rfile (*s2*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))) **endlet**

THEOREM: strstr-s-s2-mem
let *s2* **be** stepn (*s*, strstr-t0 (*n2*, *lst2*))
in
(strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
∧ (get-nth (0, *lst2*) ≠ 0)
∧ disjoint (*x*, *k*, sub (32, 48, read-sp (*s*)), 60))
→ (read-mem (*x*, mc-mem (*s2*), *k*) = read-mem (*x*, mc-mem (*s*), *k*)) **endlet**

```
; put together: s2 --> s2.
```

THEOREM: strstr-s2-s2-1
let *s2* **be** stepn (*s*, strstr-t2 (*i*, *n1*, *lst1*, *lst2*, *len*)),
*j** **be** strchr1* (*i**, *i*, *n1*, *lst1*, get-nth (0, *lst2*)),
j **be** strchr1 (*i*, *n1*, *lst1*, get-nth (0, *lst2*))
in
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ (*j* ∈ **N**)
∧ (strncmp (*len*, mcdr (1 + *j*, *lst1*), mcdr (1, *lst2*)) ≠ 0))
→ (strstr-s2p (*s2*,
add (32, *j**, 1),
1 + *j*,
str1,
n1,
lst1,
str2,
n2,
lst2,
len)
∧ (read-rn (32, 14, mc-rfile (*s2*))

$=$ read-rn (32, 14, mc-rfile (s)))
$\wedge$ (linked-rts-addr ($s2$) = linked-rts-addr (s))
$\wedge$ (linked-a6 ($s2$) = linked-a6 (s))
$\wedge$ (movem-saved ($s2$, 4, 16, 4) = movem-saved (s, 4, 16, 4))) **endlet**

THEOREM: strstr-s2-s2-rfile-1
let $s2$ **be** stepn (s, strstr-t2 (i, $n1$, $lst1$, $lst2$, len)),
j **be** strchr1 (i, $n1$, $lst1$, get-nth (0, $lst2$))
in
(strstr-s2p (s, $i*$, i, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, len)
$\wedge$ ($j \in \mathbf{N}$)
$\wedge$ (strncmp (len, mcdr (1 + j, $lst1$), mcdr (1, $lst2$)) $\neq$ 0)
$\wedge$ ($oplen \leq 32$)
$\wedge$ d4-7a4-5p (rn))
$\rightarrow$ (read-rn ($oplen$, rn, mc-rfile ($s2$))
$=$ read-rn ($oplen$, rn, mc-rfile (s))) **endlet**

THEOREM: strstr-s2-s2-mem-1
let $s2$ **be** stepn (s, strstr-t2 (i, $n1$, $lst1$, $lst2$, len)),
j **be** strchr1 (i, $n1$, $lst1$, get-nth (0, $lst2$))
in
(strstr-s2p (s, $i*$, i, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, len)
$\wedge$ ($j \in \mathbf{N}$)
$\wedge$ (strncmp (len, mcdr (1 + j, $lst1$), mcdr (1, $lst2$)) $\neq$ 0)
$\wedge$ disjoint (x, k, sub (32, 44, read-an (32, 6, s)), 60))
$\rightarrow$ (read-mem (x, mc-mem ($s2$), k) = read-mem (x, mc-mem (s), k)) **endlet**

```
; put together: s2 --> sn.
```

THEOREM: strstr-s2-sn-base-1
let sn **be** stepn (s, strstr-t2 (i, $n1$, $lst1$, $lst2$, len)),
$j*$ **be** strchr1* ($i*$, i, $n1$, $lst1$, get-nth (0, $lst2$)),
j **be** strchr1 (i, $n1$, $lst1$, get-nth (0, $lst2$))
in
(strstr-s2p (s, $i*$, i, $str1$, $n1$, $lst1$, $str2$, $n2$, $lst2$, len)
$\wedge$ ($j \in \mathbf{N}$)
$\wedge$ (strncmp (len, mcdr (1 + j, $lst1$), mcdr (1, $lst2$)) = 0))
$\rightarrow$ ((mc-status (sn) = 'running)
$\wedge$ (mc-pc (sn) = linked-rts-addr (s))
$\wedge$ (read-dn (32, 0, sn) = add (32, $str1$, $j*$))
$\wedge$ (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: strstr-s2-sn-base-rfile-1

```
let sn be stepn(s, strstr-t2(i, n1, lst1, lst2, len)),
    j be strchr1(i, n1, lst1, get-nth(0, lst2))
in
(strstr-s2p(s, i*, i, str1, n1, lst1, str2, n2, lst2, len)
 ∧  (j ∈ N)
 ∧  (strncmp(len, mcdr(1 + j, lst1), mcdr(1, lst2)) = 0)
 ∧  (oplen ≤ 32)
 ∧  d2-7a2-5p(rn))
→  (read-rn(oplen, rn, mc-rfile(sn))
      =  if d4-7a4-5p(rn) then read-rn(oplen, rn, mc-rfile(s))
         else get-vlst(oplen,
                       0,
                       rn,
                       '(2 3 10 11),
                       movem-saved(s, 4, 16, 4)) endif) endlet
```

THEOREM: strstr-s2-sn-base-mem-1

```
let sn be stepn(s, strstr-t2(i, n1, lst1, lst2, len)),
    j be strchr1(i, n1, lst1, get-nth(0, lst2))
in
(strstr-s2p(s, i*, i, str1, n1, lst1, str2, n2, lst2, len)
 ∧  (j ∈ N)
 ∧  (strncmp(len, mcdr(1 + j, lst1), mcdr(1, lst2)) = 0)
 ∧  disjoint(x, k, sub(32, 44, read-an(32, 6, s)), 60))
→  (read-mem(x, mc-mem(sn), k) = read-mem(x, mc-mem(s), k)) endlet

; put together: s2 --> sn.
```

THEOREM: strstr-s2-sn-2

```
let sn be stepn(s, strstr-t3(i, n1, lst1, lst2, len))
in
strstr-s2p(s, i*, i, str1, n1, lst1, str2, n2, lst2, len)
→  ((mc-status(sn) = 'running)
    ∧  (mc-pc(sn) = linked-rts-addr(s))
    ∧  (read-dn(32, 0, sn)
          =  if strstr1(i, n1, lst1, n2, lst2, len)
             then add(32,
                      str1,
                      strstr1*(i*, i, n1, lst1, n2, lst2, len))
             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))) endlet
```

THEOREM: strstr-s2-sn-rfile-2

let *sn* **be** stepn (*s*, strstr-t3 (*i*, *n1*, *lst1*, *lst2*, *len*))
in
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ d2-7a2-5p (*rn*)
∧ (*oplen* ≤ 32))
→ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= **if** d4-7a4-5p (*rn*) **then** read-rn (*oplen*, *rn*, mc-rfile (*s*))
else get-vlst (*oplen*,
0,
rn,
'(2 3 10 11),
movem-saved (*s*, 4, 16, 4)) **endif**) **endlet**

THEOREM: strstr-s2-sn-mem-2
let *sn* **be** stepn (*s*, strstr-t3 (*i*, *n1*, *lst1*, *lst2*, *len*))
in
(strstr-s2p (*s*, *i**, *i*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*, *len*)
∧ disjoint (*x*, *k*, sub (32, 44, read-an (32, 6, *s*)), 60))
→ (read-mem (*x*, mc-mem (*sn*), *k*) = read-mem (*x*, mc-mem (*s*), *k*)) **endlet**

EVENT: Disable strstr-s2p-info.

```
; the correctness of strstr.
```

THEOREM: strstr-statep-info
strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ ((*str1* ∈ **N**) ∧ nat-rangep (*str1*, 32))

THEOREM: strstr-correctness
let *sn* **be** stepn (*s*, strstr-t (*n1*, *lst1*, *n2*, *lst2*))
in
strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
→ ((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*) ∧ (*oplen* ≤ 32))
→ (read-rn (*oplen*, *rn*, mc-rfile (*sn*))
= read-rn (*oplen*, *rn*, mc-rfile (*s*))))
∧ (disjoint (*x*, *k*, sub (32, 48, read-sp (*s*)), 60)
→ (read-mem (*x*, mc-mem (*sn*), *k*)
= read-mem (*x*, mc-mem (*s*), *k*)))

$\wedge$ (read-dn (32, 0, *sn*)
= **if** strstr (*n1*, *lst1*, *n2*, *lst2*)
then add (32, *str1*, strstr* (*n1*, *lst1*, *n2*, *lst2*))
else 0 **endif**)) **endlet**

EVENT: Disable strstr-statep-info.

EVENT: Disable strstr-t.

```
; strstr* --> strstr.
```

THEOREM: strchr1*-strchr1
(strchr1 (*i*, *n*, *lst*, *ch*)
$\wedge$ (*i* = nat-to-uint (*i**))
$\wedge$ nat-rangep (*i**, 32)
$\wedge$ uint-rangep (*n*, 32))
$\rightarrow$ (nat-to-uint (strchr1* (*i**, *i*, *n*, *lst*, *ch*)) = strchr1 (*i*, *n*, *lst*, *ch*))

THEOREM: strstr*-strstr
(strstr1 (*i*, *n1*, *lst1*, *n2*, *lst2*, *len*)
$\wedge$ (*i* = nat-to-uint (*i**))
$\wedge$ nat-rangep (*i**, 32)
$\wedge$ uint-rangep (*n1*, 32))
$\rightarrow$ (nat-to-uint (strstr1* (*i**, *i*, *n1*, *lst1*, *n2*, *lst2*, *len*))
= strstr1 (*i*, *n1*, *lst1*, *n2*, *lst2*, *len*))

THEOREM: strstr-non-zerop-la
let *sn* **be** stepn (*s*, strstr-t (*n1*, *lst1*, *n2*, *lst2*))
in
(strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ nat-rangep (*str1*, 32)
$\wedge$ (nat-to-uint (*str1*) $\neq$ 0)
$\wedge$ uint-rangep (nat-to-uint (*str1*) + *n1*, 32)
$\wedge$ (*str1* $\in$ **N**)
$\wedge$ strstr (*n1*, *lst1*, *n2*, *lst2*))
$\rightarrow$ (nat-to-uint (read-dn (32, 0, *sn*)) $\neq$ 0) **endlet**

THEOREM: strstr-non-zerop
let *sn* **be** stepn (*s*, strstr-t (*n1*, *lst1*, *n2*, *lst2*))
in
(strstr-statep (*s*, *str1*, *n1*, *lst1*, *str2*, *n2*, *lst2*)
$\wedge$ strstr (*n1*, *lst1*, *n2*, *lst2*))
$\rightarrow$ (nat-to-uint (read-dn (32, 0, *sn*)) $\neq$ 0) **endlet**

EVENT: Disable strstr*.

```
; some properties of strstr.
; see the file cstring.events.
```

Index