```
#|

 Copyright (C) 1994 by Computational Logic, Inc.  All Rights Reserved.

 This script is hereby placed in the public domain, and therefore unlimited
 editing and redistribution is permitted.

 NO WARRANTY

 Computational Logic, Inc. 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 Computational Logic, Inc. 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.
|#

;; Matt Kaufmann

;; See CLI Internal Note 216 for explanation.  I thank Stan Letovsky
;; for suggesting this example and for taking the lead in formalizing
;; the problem for the Boyer-Moore theorem prover.
```

EVENT: Start with the initial **nqthm** theory.

THEOREM: append-associativity
append (append (x, y), z) = append (x, append (y, z))

DEFINITION:
length (*lst*)
= **if** listp (*lst*) **then** 1 + length (cdr (*lst*))
else 0 **endif**

EVENT: Add the shell *make-op*, with recognizer function symbol *op?* and 2 accessors: *op-type*, with type restriction (none-of) and default value zero; *tid*, with type restriction (one-of numberp) and default value zero.

EVENT: Add the shell *make-obj*, with recognizer function symbol *obj?* and 2 accessors: *lock-tid*, with type restriction (one-of numberp falsep) and default value zero; *waiters*, with type restriction (none-of) and default value zero.

```
;; oops -- another basic lemma needed
```

THEOREM: length-append
length (append (x, y)) = (length (x) + length (y))

DEFINITION:
server (*tranq*, x)
= **let** *current* **be** car (*tranq*)
in
if *tranq* $\simeq$ **nil** **then nil**
elseif (op-type (*current*) = `'write`)
$\wedge$ (tid (*current*) = lock-tid (x))
then cons (*current*,
server (append (waiters (x), cdr (*tranq*)), make-obj (**f**, **nil**)))
elseif falsep (lock-tid (x))
then if op-type (*current*) = `'read`
then cons (*current*,
server (cdr (*tranq*),
make-obj (tid (*current*), waiters (x))))
else cons (*current*, server (cdr (*tranq*), x)) **endif**
else server (cdr (*tranq*),
make-obj (lock-tid (x),
append (waiters (x), list (*current*)))) **endif endlet**

DEFINITION:
find-write (*tid*, *tranq*)
= **if** listp (*tranq*)
then (op? (car (*tranq*))
$\wedge$ (op-type (car (*tranq*)) = `'write`)
$\wedge$ (*tid* = tid (car (*tranq*))))
$\vee$ find-write (*tid*, cdr (*tranq*))
else f endif

```
#|
I was trying to prove a lemma (later not needed) which generated
the following goal:

(IMPLIES (AND (LISTP TRANQ)
              (OP? (CAR TRANQ))
              (EQUAL TID (TID (CAR TRANQ))))
```

```
    (IMPLIES (FIND-WRITE TID TRANQ)
             (EQUAL (GOOD-TRANQ (REMOVE-WRITE TID TRANQ))
                    (GOOD-TRANQ TRANQ))))

Inspection of this goal lead me to realize that when removing
or finding a WRITE operation, one needs to check not only the
TID but also that the operation is really a WRITE!  I then fixed
the definition of REMOVE-WRITE (as indicated below), and later
realized I needed to make a similar change to the definition of
FIND-WRITE (see above).
|#
```

DEFINITION:
remove-write (*tid*, *tranq*)
= **if** listp (*tranq*)
  then if op? (car (*tranq*))
      $\land$ (op-type (car (*tranq*)) = `'write`)
      $\land$ (*tid* = tid (car (*tranq*))) **then** cdr (*tranq*)
    else cons (car (*tranq*), remove-write (*tid*, cdr (*tranq*))) **endif**
  else nil endif

THEOREM: good-tranq-helper
length (*tranq*) $\not<$ length (remove-write (*tid*, *tranq*))

DEFINITION:
good-tranq (*tranq*)
= **if** listp (*tranq*)
  then op? (car (*tranq*))
    $\land$ **if** op-type (car (*tranq*)) = `'read`
      then find-write (tid (car (*tranq*)), cdr (*tranq*))
        $\land$ good-tranq (remove-write (tid (car (*tranq*)),
            cdr (*tranq*)))
      else good-tranq (cdr (*tranq*)) **endif**
  else t endif

DEFINITION:
good-trace (*trace*, *tid*)
= **if** listp (*trace*)
  then op? (car (*trace*))
    $\land$ **if** *tid*
      then case on op-type (car (*trace*)):
        case = *read*
        then f
        case = *write*

then (tid (car (*trace*)) = *tid*)
$\wedge$ good-trace (cdr (*trace*), **f**)
otherwise good-trace (cdr (*trace*), *tid*) **endcase**
else case on op-type (car (*trace*)):
case = *read*
then good-trace (cdr (*trace*), tid (car (*trace*)))
otherwise good-trace (cdr (*trace*), **f**) **endcase endif**
elseif *tid* **then f**
else t endif

```
;; *** The simplification in the case below where tid is F is perhaps
;; the key to the whole proof.  Once I made this change, the only
;; remaining fix was the one in the definition shown above, and that
;; was easy to locate by inspection of the output of the main lemma,
;; SERVER-SAFETY-MAIN-LEMMA.
```

DEFINITION:
good-tranq-tid (x, y, *tid*)
= **if** *tid*
then find-write (*tid*, y) $\wedge$ good-tranq (append (x, remove-write (*tid*, y)))
else (x = **nil**) $\wedge$ good-tranq (y) **endif**

THEOREM: server-safety-main-lemma
(obj? (x) $\wedge$ good-tranq-tid (waiters (x), *tranq*, lock-tid (x)))
$\rightarrow$ good-trace (server (*tranq*, x), lock-tid (x))

THEOREM: server-safety
good-tranq (*tranq*) $\rightarrow$ good-trace (server (*tranq*, make-obj (**f**, **nil**)), **f**)

Index