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

\[ \text{length}(\text{append}(x, y)) = (\text{length}(x) + \text{length}(y)) \]

Definition:

\[
\text{server}(\text{tranq}, x) = \begin{cases} 
\text{let } \text{current} \text{ be } \text{car}(\text{tranq}) 
\text{ in } \\
\text{if } \text{tranq} \not\equiv \text{nil} \text{ then } \text{nil} \\
\text{elseif } \left( \text{op-type}(\text{current}) = '\text{write} \right) \\
\quad \land \left( \text{tid}(\text{current}) = \text{lock-tid}(x) \right) \\
\quad \text{then } \text{cons}(\text{current}, \text{server}(\text{append}(\text{waiters}(x), \text{cdr}(\text{tranq})), \text{make-obj}(\text{f, nil}))) \\
\text{elseif } \text{falsep}(\text{lock-tid}(x)) \\
\quad \text{then if } \text{op-type}(\text{current}) = '\text{read} \\
\qquad \text{then } \text{cons}(\text{current}, \text{server}(\text{cdr}(\text{tranq}), \text{make-obj}(\text{tid}(\text{current}), \text{waiters}(x)))) \\
\qquad \text{else cons}(\text{current}, \text{server}(\text{cdr}(\text{tranq}), x)) \text{ endif} \\
\text{else } \text{server}(\text{cdr}(\text{tranq}), \text{make-obj}(\text{lock-tid}(x), \text{append}(\text{waiters}(x), \text{list}(\text{current})))) \text{ endif} \end{cases} \]

Definition:

\[
\text{find-write}(\text{tid}, \text{tranq}) = \begin{cases} 
\text{if } \text{listp}(\text{tranq}) \\
\quad \text{then } \left( \text{op?}(\text{car}(\text{tranq})) \\
\quad \land \left( \text{op-type}(\text{car}(\text{tranq})) = '\text{write} \right) \\
\quad \land \left( \text{tid} = \text{tid}(\text{car}(\text{tranq})) \right) \right) \\
\quad \lor \left( \text{find-write}(\text{tid}, \text{cdr}(\text{tranq})) \right) \\
\text{else } \text{f} \text{ endif} \end{cases} \]

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

\[
(\text{IMPLIES} \ (\text{AND} \ (\text{LISTP} \ \text{TRANQ}) \\
(\text{OP?} \ \text{(CAR} \ \text{TRANQ})) \\
(\text{EQUAL} \ \text{TID} \ \text{(TID} \ \text{(CAR} \ \text{TRANQ}))))))
\]
\[(\text{IMPLIES} \ (\text{FIND-WRITE} \ \text{TID} \ \text{TRANQ})
\qquad (\text{EQUAL} \ (\text{GOOD-TRANQ} \ \text{REMOVE-WRITE} \ \text{TID} \ \text{TRANQ}))
\qquad (\text{GOOD-TRANQ} \ \text{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 \((\text{tid, tranq})\)
\[= \begin{cases} 
\text{if listp(}\text{tranq}) \\
\quad \text{then if op? (}\text{car(}\text{tranq})) \\
\quad \quad \wedge \ (\text{op-type(}\text{car(}\text{tranq})) = \‘\text{write}) \\
\quad \quad \wedge \ (\text{tid} = \text{tid (}\text{car(}\text{tranq})) \quad \text{then} \ \text{cdr(}\text{tranq}) \\
\quad \quad \text{else} \ \text{cons(}\text{car(}\text{tranq}), \ \text{remove-write(}\text{tid, cdr(}\text{tranq)})) \ \text{endif} \\
\quad \text{else nil endif} 
\end{cases}\]

**Theorem:** good-tranq-helper
\[\text{length(}\text{tranq}) \neq \text{length(}\text{remove-write(}\text{tid, tranq)}))\]

**Definition:**
good-tranq \((\text{tranq})\)
\[= \begin{cases} 
\text{if listp(}\text{tranq}) \\
\quad \text{then op? (}\text{car(}\text{tranq})) \\
\quad \quad \wedge \ \text{if op-type(}\text{car(}\text{tranq})) = \‘\text{read} \\
\quad \quad \quad \text{then} \ \text{find-write(}\text{tid (}\text{car(}\text{tranq)}), \ \text{cdr(}\text{tranq})) \\
\quad \quad \quad \wedge \ \text{good-tranq(}\text{remove-write(}\text{tid (}\text{car(}\text{tranq)}), \ \text{cdr(}\text{tranq)})) \\
\quad \quad \text{else} \ \text{good-tranq(}\text{cdr(}\text{tranq)}) \ \text{endif} \\
\quad \text{else t endif} 
\end{cases}\]

**Definition:**
good-trace \((\text{trace, tid})\)
\[= \begin{cases} 
\text{if listp(}\text{trace}) \\
\quad \text{then op? (}\text{car(}\text{trace})) \\
\quad \quad \wedge \ \text{if tid} \\
\quad \quad \quad \text{then case on op-type(}\text{car(}\text{trace)}): \\
\quad \quad \quad \quad \text{case} = \ ‘\text{read} \\
\quad \quad \quad \quad \text{then f} \\
\quad \quad \quad \text{case} = \ ‘\text{write} \\
\end{cases}\]

3
then \((\text{tid}(\text{car}(\text{trace})) = \text{tid})\)  
\&  \text{good-trace}(\text{cdr}(\text{trace}), \text{f})\)  
otherwise \text{good-trace}(\text{cdr}(\text{trace}), \text{tid}) \)  
endcase

else case on \text{op-type}(\text{car}(\text{trace})):
  
case = \text{read}
  then \text{good-trace}(\text{cdr}(\text{trace}), \text{tid}(\text{car}(\text{trace})))
  otherwise \text{good-trace}(\text{cdr}(\text{trace}), \text{f}) \)  
endcase endif
  
eelseif \text{tid} \) then \text{f}
  else \text{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\( (x, y, \text{tid}) \)
  =  if \text{tid}  
   then \text{find-write}(\text{tid}, y) \& \text{good-tranq}(\text{append}(x, \text{remove-write}(\text{tid}, y)))
   else \(x = \text{nil}\) \& \text{good-tranq}(y) \)  
endif

**Theorem:** server-safety-main-lemma
  
(obj?\((x) \& \text{good-tranq-tid}(\text{waiters}(x), \text{tranq}, \text{lock-tid}(x)))
  \rightarrow  \text{good-trace}(\text{server}(\text{tranq}, x), \text{lock-tid}(x))

**Theorem:** server-safety
  
good-tranq\((\text{tranq}) \rightarrow \text{good-trace}(\text{server}(\text{tranq}, \text{make-obj}(f, \text{nil})), f)\)
Index

append-associativity, 1

find-write, 2–4

good-trace, 3, 4
good-tranq, 3, 4
good-tranq-helper, 3
good-tranq-tid, 4

length, 1–3
length-append, 2
lock-tid, 2, 4

make-obj, 2, 4
make-op, 1

obj?, 4
op-type, 2–4
op?, 2, 3

remove-write, 3, 4

server, 2, 4
server-safety, 4
server-safety-main-lemma, 4

tid, 2–4

waiters, 2, 4