Copyright (C) 1994 by Robert S. Boyer and J Strother Moore. All Rights Reserved.

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

NO WARRANTY

Robert S. Boyer and J Strother Moore PROVIDE 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 Robert S. Boyer or J Strother Moore 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 initial nqthm theory.

EVENT: For efficiency, compile those definitions not yet compiled.

EVENT: Add the shell btm, with recognizer function symbol btmp and no accessors.

DEFINITION:
get (x, alist)
    = if alist ∼ nil then btm
      elseif x = caar (alist) then cdar (alist)
      else get (x, cdr (alist)) endif

DEFINITION:
unsolv-subrp (fn)
\[
= (fn \in \{\text{zero true false add1 sub1 numberp cons car cdr listp pack unpack litatom equal list}\})
\]

**Definition:**
not-solve-apply-subr \((fn, lst)\)

\[
= \begin{cases} 
\text{if } fn = \text{zero} \text{ then } \text{ZERO} \\
\text{elseif } fn = \text{true} \text{ then } \text{TRUE} \\
\text{elseif } fn = \text{false} \text{ then } \text{FALSE} \\
\text{elseif } fn = \text{add1} \text{ then } 1 + \text{car}(lst) \\
\text{elseif } fn = \text{sub1} \text{ then } \text{car}(lst) - 1 \\
\text{elseif } fn = \text{numberp} \text{ then } \text{car}(lst) \in \mathbb{N} \\
\text{elseif } fn = \text{cons} \text{ then } \text{cons}(\text{car}(lst), \text{cadr}(lst)) \\
\text{elseif } fn = \text{list} \text{ then } lst \\
\text{elseif } fn = \text{car} \text{ then } \text{car}(\text{car}(lst)) \\
\text{elseif } fn = \text{cdr} \text{ then } \text{cdr}(\text{car}(lst)) \\
\text{elseif } fn = \text{listp} \text{ then } \text{listp}(\text{car}(lst)) \\
\text{elseif } fn = \text{pack} \text{ then } \text{pack}(\text{car}(lst)) \\
\text{elseif } fn = \text{unpack} \text{ then } \text{unpack}(\text{car}(lst)) \\
\text{elseif } fn = \text{litatom} \text{ then } \text{litatom}(\text{car}(lst)) \\
\text{elseif } fn = \text{equal} \text{ then } \text{car}(lst) = \text{cdr}(lst) \\
\text{else } 0 
\end{cases}
\]

**Definition:**
\(ev(flq, x, va, fa, n)\)

\[
= \begin{cases} 
\text{if } flq = \text{al} \text{ then if } x \simeq \text{nil} \\
\text{then if } x \in \mathbb{N} \text{ then } x \\
\text{elseif } x = \text{t} \text{ then } t \\
\text{elseif } x = \text{f} \text{ then } f \\
\text{elseif } x = \text{nil} \text{ then } \text{nil} \\
\text{else } \text{get}(x, va) \text{ endif} \\
\text{elseif } \text{car}(x) = \text{quote} \text{ then } \text{cadr}(x) \\
\text{elseif } \text{car}(x) = \text{if} \\
\text{then if } \text{btmp}(\text{ev}(\text{al}, \text{cadr}(x), va, fa, n)) \text{ then } \text{BTM} \\
\text{elseif } \text{ev}(\text{al}, \text{cadr}(x), va, fa, n) \\
\text{then } \text{ev}(\text{al}, \text{caddr}(x), va, fa, n) \\
\text{else } \text{ev}(\text{al}, \text{caddr}(x), va, fa, n) \text{ endif} \\
\text{elseif } \text{btmp}(\text{ev}(\text{list}, \text{cadr}(x), va, fa, n)) \text{ then } \text{BTM} \\
\text{elseif } \text{unsolv-subrp}(\text{car}(x)) \\
\text{then } \text{unsolv-apply-subr}(\text{car}(x), \text{ev}(\text{list}, \text{cadr}(x), va, fa, n)) \\
\text{elseif } \text{btmp}(\text{get}(\text{car}(x), fa)) \text{ then } \text{BTM} \\
\text{elseif } n \simeq 0 \text{ then } \text{BTM} \\
\text{else } \text{ev}(\text{al}, \text{cadr}(\text{get}(\text{car}(x), fa)));
\end{cases}
\]
pairlist (car (get (car x), fa)),
ev('list, cdr x, va, fa, n)),
fa,
n - 1) endif
elseif listp (x)
then if btmp (ev ('al, car x, va, fa, n)) then BTM
elseif btmp (ev ('list, cdr x, va, fa, n)) then BTM
else cons (ev ('al, car x, va, fa, n),
ev ('list, cdr x, va, fa, n)) endif
else nil endif

**DEFINITION:** pr-eval (x, va, fa, n) = ev ('al, x, va, fa, n)

**DEFINITION:** evlist (x, va, fa, n) = ev ('list, x, va, fa, n)

; We now define the functions x, va, fa, and k. To do so we first define
; SUBLIS, which applies a substitution to an s-expression. Then we use the
; names CIRC and LOOP in the definitions of x and fa and use SUBLIS to
; replace those names with "new" names. It is not important whether we have
; defined this notion of substitution correctly, since all that is required
; is that we exhibit some x, va, fa, and k with the desired properties.

**DEFINITION:**
sublis (alist, x) =
  if x ≃ nil
    then if assoc (x, alist) then cdr (assoc (x, alist))
    else x endif
  else cons (sublis (alist, car x), sublis (alist, cdr x)) endif

**DEFINITION:**
x (fa) = sublis (list (cons ('circ, cons (fa, 0))), ' (circ a))

**DEFINITION:**
fa (fa) = append (sublis (list (cons ('circ, cons (fa, 0))), cons ('loop, cons (fa, 1))),
  ' ((circ (a)
    (if
      (halts (circ a) (list (cons 'a a)) a)
      (loop)
    t))
  (loop nil (loop)))},
fa)

**DEFINITION:** va (fa) = list (cons ('a, fa (fa)))

3
Definition: \( k(n) = (1 + n) \)

; We wish to prove that having "new" program names in the function
; environment does not effect the computation of the body of HALTS. To state
; this we must first define formally what we mean by "new". Then we will
; prove the general result we need and then we will instantiate it for the
; particular "new" program names we choose.

Definition:
\[
\text{occur}(x, y) = \begin{cases} \text{t} & \text{if } x = y \\ \text{f} & \text{if } y \simeq \text{nil} \\ \text{occur}(x, \text{car}(y)) \lor \text{occur}(x, \text{cdr}(y)) & \text{else} \end{cases}
\]

Definition:
\[
\text{occur-in-defns}(x, \text{lst}) = \begin{cases} \text{f} & \text{if } \text{lst} \simeq \text{nil} \\ \text{occur}(x, \text{caddr}(\text{car}(\text{lst}))) \lor \text{occur-in-defns}(x, \text{cdr}(\text{lst})) & \text{else} \end{cases}
\]

Theorem: occur-occur-in-defns
\[
((\neg \text{occur-in-defns}(\text{fn}, \text{fa})) \land (\neg \text{btmp} (\text{get}(x, \text{fa})))) \rightarrow (\neg \text{occur}(\text{fn}, \text{cadr(\text{get}(x, \text{fa}))}))
\]

Theorem: lemma1
\[
((\neg \text{occur}(\text{fn}, x)) \land (\neg \text{occur-in-defns}(\text{fn}, \text{fa}))) \rightarrow (\text{ev}(\text{flg}, x, \text{va}, \text{cons(cons(\text{fn}, \text{def}), \text{fa}), n}) = \text{ev}(\text{flg}, x, \text{va}, \text{fa}, n))
\]

Theorem: count-occur
\[
(\text{count}(y) < \text{count}(x)) \rightarrow (\neg \text{occur}(x, y))
\]

Theorem: count-get
\[
\text{count(\text{cadr(\text{get}(\text{fn}, \text{fa}))}) < (1 + \text{count(\text{fa}))}}
\]

Theorem: count-occur-in-defns
\[
(\text{count}(\text{fa}) < \text{count}(x)) \rightarrow (\neg \text{occur-in-defns}(x, \text{fa}))
\]

Theorem: corollary1
\[
\text{ev('al, cadr(get('halts, fa)), va, cons(cons(cons(\text{fa}, 0), \text{def0}), cons(list(cons(\text{fa}, 1), \text{nil, list(cons(\text{fa}, 1)})), \text{fa})), n}) = \text{ev('al, cadr(get('halts, fa)), va, fa, n)}
\]

4
Event: Disable lemma 1.

**Theorem:** lemma 2
\[ ((\neg \text{btmp}(\text{ev} (\text{flg}, x, \text{va}, \text{fa}, n))) \land (\neg \text{btmp}(\text{ev}(\text{flg}, x, \text{va}, \text{fa}, k)))) \rightarrow (\text{ev}(\text{flg}, x, \text{va}, \text{fa}, n) = \text{ev}(\text{flg}, x, \text{va}, \text{fa}, k)) \]

**Theorem:** corollary 2
\[ (\text{ev}(\text{flg}, x, \text{va}, \text{fa}, n) = \text{t}) \rightarrow \text{ev}(\text{flg}, x, \text{va}, \text{fa}, k) \]

**Theorem:** lemma 3
\[ (\text{listp}(x) \land (\text{cadr}(x) \simeq \text{nil}) \land \text{listp}(\text{get}(\text{car}(x), \text{fa})) \land (\text{car}(\text{get}(\text{car}(x), \text{fa})) = \text{nil}) \land (\text{cadr}(\text{get}(\text{car}(x), \text{fa})) = x)) \rightarrow \text{btmp}(\text{ev}(\text{al}, x, \text{va}, \text{fa}, n)) \]

**Theorem:** expand-circ
\[ ((\neg \text{btmp}(\text{val})) \land (\neg \text{btmp}(\text{get}(\text{cons}(\text{fn}, 0), \text{fa})))) \rightarrow (\text{ev}(\text{al}, \text{cons}(\text{cons}(\text{fn}, 0), \text{a}), \text{list}(\text{cons}(\text{a}, \text{val})), \text{fa}, \text{j}) = \text{if} \text{j} \simeq 0 \text{ then } \text{btm} \text{ else } \text{ev}(\text{al}, \text{cadr}(\text{get}(\text{cons}(\text{fn}, 0), \text{fa}))), \text{pairlist}(\text{car}(\text{get}(\text{cons}(\text{fn}, 0), \text{fa}))), \text{ev}(\text{list}, \text{a}, \text{list}(\text{cons}(\text{a}, \text{val})), \text{fa}, \text{j})))) \]

; After we published a proof of the unsolvability of the halting problem in
; the JACM, a student in one of our classes named Jonathan Bellin observed
; that one could get a trivial proof by defining (x FA) = (BTM). However,
; the "idea" is that the frustrating values (x FA), (va FA), and (fa FA) are
; supposed to be objects on which EVAL behaves normally. This class consists
; of those objects for which SEXP, defined below is, true. So we added the
; second conjunct to our statement of UNSOLVABILITY-OF-THE-HALTING-PROBLEM.

**Definition:**
\[ \text{sexp}(x) \]
if $x = t$ then $t$
elseif $x = f$ then $t$
elseif $x \in N$ then $t$
elseif listp ($x$) then sexp (car ($x$)) \& sexp (cdr ($x$))
else f endif

Theorem: unsolvability-of-the-halting-problem

$((h = \text{pr-eval (list ('halts, list ('quote, x (fa)), list ('quote, va (fa)), list ('quote, fa (fa))), nil, fa, n))) \rightarrow (((h = f) \rightarrow (\neg \text{btmp (pr-eval (x (fa), va (fa), va (fa), k (n))))) \land ((h = t) \rightarrow \text{btmp (pr-eval (x (fa), va (fa), va (fa), k (n)))))) \land (\text{sexp (fa)} \rightarrow (\text{sexp (x (fa))} \land \text{sexp (va (fa))} \land \text{sexp (fa (fa))))))$
Index

btm, 1–3, 5
btmp, 2–6

corollary1, 4
corollary2, 5
count-get, 4
count-occur, 4
count-occur-in-defns, 4

ev, 2–5
evlist, 3
expand-circ, 5

fa, 3, 6
get, 1–5
k, 4, 6

lemma1, 4
lemma2, 5
lemma3, 5

occur, 4
occur-in-defns, 4
occur-occur-in-defns, 4

pr-eval, 3, 6
sexp, 5, 6
sublis, 3

unsolv-apply-subr, 2
unsolv-subrp, 1, 2
unsolvability-of-the-halting-problem, 6

va, 3, 6

x, 3, 6