```
#|

 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* $\simeq$ **nil** **then** BTM
  elseif x = caar (*alist*) **then** cdar (*alist*)
  else get (x, cdr (*alist*)) **endif**

DEFINITION:
unsolv-subrp (*fn*)

```
=   (fn ∈ '(zero true false add1 sub1 numberp cons car cdr
            listp pack unpack litatom equal list))
```

DEFINITION:

```
unsolv-apply-subr (fn, lst)
=   if fn = 'zero then ZERO
    elseif fn = 'true then TRUE
    elseif fn = 'false then FALSE
    elseif fn = 'add1 then 1 + car (lst)
    elseif fn = 'sub1 then car (lst) − 1
    elseif fn = 'numberp then car (lst) ∈ N
    elseif fn = 'cons then cons (car (lst), cadr (lst))
    elseif fn = 'list then lst
    elseif fn = 'car then car (car (lst))
    elseif fn = 'cdr then cdr (car (lst))
    elseif fn = 'listp then listp (car (lst))
    elseif fn = 'pack then pack (car (lst))
    elseif fn = 'unpack then unpack (car (lst))
    elseif fn = 'litatom then litatom (car (lst))
    elseif fn = 'equal then car (lst) = cadr (lst)
    else 0 endif
```

DEFINITION:

```
ev (flg, x, va, fa, n)
=   if flg = 'al
    then if x ≃ nil
         then if x ∈ N then x
              elseif x = 't then t
              elseif x = 'f then f
              elseif x = nil then nil
              else get (x, va) endif
         elseif car (x) = 'quote then cadr (x)
         elseif car (x) = 'if
         then if btmp (ev ('al, cadr (x), va, fa, n)) then BTM
              elseif ev ('al, cadr (x), va, fa, n)
              then ev ('al, caddr (x), va, fa, n)
              else ev ('al, cadddr (x), va, fa, n) endif
         elseif btmp (ev ('list, cdr (x), va, fa, n)) then BTM
         elseif unsolv-subrp (car (x))
         then unsolv-apply-subr (car (x), ev ('list, cdr (x), va, fa, n))
         elseif btmp (get (car (x), fa)) then BTM
         elseif n ≃ 0 then BTM
         else ev ('al,
                  cadr (get (car (x), fa)),
```

```
                        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*)))

DEFINITION: $\mathrm{k}(n) = (1 + n)$

```
;   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:
occur (x, y)
= **if** $x = y$ **then t**
elseif $y \simeq$ **nil then f**
else occur $(x, \mathrm{car}(y)) \vee$ occur $(x, \mathrm{cdr}(y))$ **endif**

DEFINITION:
occur-in-defns (x, lst)
= **if** $lst \simeq$ **nil then f**
else occur $(x, \mathrm{caddr}(\mathrm{car}(lst))) \vee$ occur-in-defns $(x, \mathrm{cdr}(lst))$ **endif**

THEOREM: occur-occur-in-defns
$((\neg$ occur-in-defns $(fn, fa)) \wedge (\neg$ btmp $(\mathrm{get}(x, fa))))$
$\rightarrow$ $(\neg$ occur $(fn, \mathrm{cadr}(\mathrm{get}(x, fa))))$

THEOREM: lemma1
$((\neg$ occur $(fn, x)) \wedge (\neg$ occur-in-defns $(fn, fa)))$
$\rightarrow$ $(\mathrm{ev}(flg, x, va, \mathrm{cons}(\mathrm{cons}(fn, def), fa), n) = \mathrm{ev}(flg, x, va, fa, n))$

THEOREM: count-occur
$(\mathrm{count}(y) < \mathrm{count}(x)) \rightarrow (\neg$ occur $(x, y))$

THEOREM: count-get
$\mathrm{count}(\mathrm{cadr}(\mathrm{get}(fn, fa))) < (1 + \mathrm{count}(fa))$

THEOREM: count-occur-in-defns
$(\mathrm{count}(fa) < \mathrm{count}(x)) \rightarrow (\neg$ occur-in-defns $(x, fa))$

THEOREM: corollary1
ev (`'al`,
cadr (get (`'halts`, fa)),
va,
cons (cons (cons (fa, `0`), $def0$),
cons (list (cons (fa, `1`), **nil**, list (cons (fa, `1`))), fa)),
n)
= ev (`'al`, cadr (get (`'halts`, fa)), va, fa, n)

EVENT: Disable lemma1.

THEOREM: lemma2
$((\neg\ \text{btmp}\,(\text{ev}\,(\mathit{flg},\ x,\ \mathit{va},\ \mathit{fa},\ n))) \wedge (\neg\ \text{btmp}\,(\text{ev}\,(\mathit{flg},\ x,\ \mathit{va},\ \mathit{fa},\ k))))$
$\rightarrow\quad (\text{ev}\,(\mathit{flg},\ x,\ \mathit{va},\ \mathit{fa},\ n) = \text{ev}\,(\mathit{flg},\ x,\ \mathit{va},\ \mathit{fa},\ k))$

THEOREM: corollary2
$(\text{ev}\,(\mathit{flg},\ x,\ \mathit{va},\ \mathit{fa},\ n) = \mathtt{t}) \rightarrow \text{ev}\,(\mathit{flg},\ x,\ \mathit{va},\ \mathit{fa},\ k)$

THEOREM: lemma3
(listp (x)
$\wedge$ listp (car (x))
$\wedge$ (cdr $(x) \simeq$ **nil**)
$\wedge$ listp (get (car (x), fa))
$\wedge$ (car (get (car (x), fa)) = **nil**)
$\wedge$ (cadr (get (car (x), fa)) = x))
$\rightarrow$ btmp (ev (`'al`, x, va, fa, n))

THEOREM: expand-circ
((¬ btmp (val)) ∧ (¬ btmp (get (cons (fn, `0`), fa))))
$\rightarrow$ (ev (`'al`, cons (cons (fn, `0`), `'(a)`), list (cons (`'a`, val)), fa, j)
= **if** $j \simeq$ `0` **then** BTM
else ev (`'al`,
cadr (get (cons (fn, `0`), fa)),
pairlist (car (get (cons (fn, `0`), fa)),
ev (`'list`,
`'(a)`,
list (cons (`'a`, val)),
fa,
j)),
fa,
$j - 1$) **endif**)

```
;   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:
sexp (x)

$$
\begin{array}{ll}
= & \textbf{if } x = \mathbf{t} \textbf{ then } \mathbf{t} \\
 & \textbf{elseif } x = \mathbf{f} \textbf{ then } \mathbf{t} \\
 & \textbf{elseif } x \in \mathbf{N} \textbf{ then } \mathbf{t} \\
 & \textbf{elseif } \mathrm{listp}\,(x) \textbf{ then } \mathrm{sexp}\,(\mathrm{car}\,(x)) \land \mathrm{sexp}\,(\mathrm{cdr}\,(x)) \\
 & \textbf{elseif } \mathrm{litatom}\,(x) \textbf{ then } \mathrm{sexp}\,(\mathrm{unpack}\,(x)) \\
 & \textbf{else } \mathbf{f} \textbf{ endif}
\end{array}
$$

THEOREM: unsolvability-of-the-halting-problem

$$
\begin{array}{l}
((h = \text{pr-eval}\,(\mathrm{list}\,(\texttt{'halts}, \\
\qquad\qquad\qquad \mathrm{list}\,(\texttt{'quote}, \mathrm{x}\,(fa)), \\
\qquad\qquad\qquad \mathrm{list}\,(\texttt{'quote}, \mathrm{va}\,(fa)), \\
\qquad\qquad\qquad \mathrm{list}\,(\texttt{'quote}, \mathrm{fa}\,(fa))), \\
\qquad\qquad \mathbf{nil}, \\
\qquad\qquad fa, \\
\qquad\qquad n)) \\
\quad \rightarrow \quad (((h = \mathbf{f}) \rightarrow (\neg\, \mathrm{btmp}\,(\text{pr-eval}\,(\mathrm{x}\,(fa), \mathrm{va}\,(fa), \mathrm{fa}\,(fa), \mathrm{k}\,(n))))) \\
\qquad\quad \land \quad ((h = \mathbf{t}) \rightarrow \mathrm{btmp}\,(\text{pr-eval}\,(\mathrm{x}\,(fa), \mathrm{va}\,(fa), \mathrm{fa}\,(fa), k))))) \\
\land \quad (\mathrm{sexp}\,(fa) \rightarrow (\mathrm{sexp}\,(\mathrm{x}\,(fa)) \land \mathrm{sexp}\,(\mathrm{va}\,(fa)) \land \mathrm{sexp}\,(\mathrm{fa}\,(fa))))
\end{array}
$$

EVENT: Make the library `"unsolv"` and compile it.

Index