```
#|

 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 library `"unsolv"` using the compiled version.

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

DEFINITION: $\text{symbol}(x) = (x \in \texttt{'(0 1)})$

DEFINITION:
$\text{half-tape}(x)$
$=$ **if** $x \simeq$ **nil then** $x = \texttt{0}$
else $\text{symbol}(\text{car}(x)) \wedge \text{half-tape}(\text{cdr}(x))$ **endif**

DEFINITION:
$\text{tape}(x) = (\text{listp}(x) \wedge \text{half-tape}(\text{car}(x)) \wedge \text{half-tape}(\text{cdr}(x)))$

DEFINITION: $\text{operation}(x) = (x \in \texttt{'(l r 0 1)})$

DEFINITION: state (x) = litatom (x)

DEFINITION:
turing-4tuple (x)
= (listp (x)
$\wedge$ state (car (x))
$\wedge$ symbol (cadr (x))
$\wedge$ operation (caddr (x))
$\wedge$ state (cadddr (x))
$\wedge$ (cddddr (x) = **nil**))

DEFINITION:
turing-machine (x)
= **if** $x \simeq$ **nil** **then** x = **nil**
else turing-4tuple (car (x)) $\wedge$ turing-machine (cdr (x)) **endif**

DEFINITION:
instr $(st,\ sym,\ tm)$
= **if** listp (tm)
then if st = car (car (tm))
then if sym = car (cdr (car (tm))) **then** cdr (cdr (car (tm)))
else instr $(st,\ sym,$ cdr $(tm))$ **endif**
else instr $(st,\ sym,$ cdr $(tm))$ **endif**
else f endif

DEFINITION:
new-tape $(op,\ tape)$
= **if** op = `'l`
then cons (cdr (car $(tape)$), cons (car (car $(tape)$), cdr $(tape)$))
elseif op = `'r`
then cons (cons (car (cdr $(tape)$), car $(tape)$), cdr (cdr $(tape)$))
else cons (car $(tape)$, cons $(op,$ cdr (cdr $(tape)$))) **endif**

DEFINITION:
tmi $(st,\ tape,\ tm,\ n)$
= **if** $n \simeq$ `0` **then** BTM
elseif instr $(st,$ car (cdr $(tape)$), $tm)$
then tmi (car (cdr (instr $(st,$ car (cdr $(tape)$), $tm)$)),
new-tape (car (instr $(st,$ car (cdr $(tape)$), $tm)$), $tape$),
tm,
$n - 1$)
else $tape$ **endif**

DEFINITION:
INSTR-DEFN

```
=  '((st sym tm)
     (if
      (listp tm)
      (if
       (equal st (car (car tm)))
       (if
        (equal sym (car (cdr (car tm))))
        (cdr (cdr (car tm)))
        (instr st sym (cdr tm)))
       (instr st sym (cdr tm)))
      f))
```

DEFINITION:
NEW-TAPE-DEFN

```
=  '((op tape)
     (if
      (equal op 'l)
      (cons
       (cdr (car tape))
       (cons (car (car tape)) (cdr tape)))
      (if
       (equal op 'r)
       (cons
        (cons (car (cdr tape)) (car tape))
        (cdr (cdr tape)))
       (cons (car tape) (cons op (cdr (cdr tape)))))))
```

DEFINITION:
TMI-DEFN

```
=  '((st tape tm)
     (if
      (instr st (car (cdr tape)) tm)
      (tmi
       (car (cdr (instr st (car (cdr tape)) tm)))
       (new-tape
        (car (instr st (car (cdr tape)) tm))
        tape)
       tm)
      tape))
```

DEFINITION: kwote (x) = list ('quote, x)

DEFINITION:
tmi-fa (tm)
= list (list ('tm, **nil**, kwote (tm)),

cons (`'instr`, INSTR-DEFN),
cons (`'new-tape`, NEW-TAPE-DEFN),
cons (`'tmi`, TMI-DEFN))

DEFINITION:
tmi-x (st, $tape$) = list (`'tmi`, kwote (st), kwote ($tape$), `'(tm)`)

DEFINITION:
length (x)
= **if** $x \simeq$ **nil** **then** `0`
else 1 + length (cdr (x)) **endif**

DEFINITION: tmi-k (st, $tape$, tm, n) = (n − (1 + length (tm)))

DEFINITION: tmi-n (st, $tape$, tm, k) = (k + (1 + length (tm)))

THEOREM: length-0
(length (x) = `0`) = ($x \simeq$ **nil**)

THEOREM: plus-equal-0
((i + j) = `0`) = (($i \simeq$ `0`) ∧ ($j \simeq$ `0`))

THEOREM: plus-difference
((i − j) + j)
= **if** $i \leq j$ **then** fix (j)
else i **endif**

EVENT: Enable difference; name this event 'difference-off'.

THEOREM: pr-eval-fn-0
(($n \simeq$ `0`)
∧ ($fn \neq$ `'quote`)
∧ ($fn \neq$ `'if`)
∧ (¬ unsolv-subrp (fn))
∧ ($vargs$ = ev (`'list`, $args$, va, fa, n)))
→ (ev (`'al`, cons (fn, $args$), va, fa, n) = BTM)

THEOREM: pr-eval-fn-1
(($n \not\simeq$ `0`)
∧ ($fn \neq$ `'quote`)
∧ ($fn \neq$ `'if`)
∧ (¬ unsolv-subrp (fn))
∧ ($vargs$ = ev (`'list`, $args$, va, fa, n)))
→ (ev (`'al`, cons (fn, $args$), va, fa, n)
= **if** btmp ($vargs$) **then** BTM

elseif btmp (get (*fn*, *fa*)) **then** BTM
else ev ('al,
cadr (get (*fn*, *fa*)),
pairlist (car (get (*fn*, *fa*)), *vargs*),
fa,
$n - 1$) **endif**)

THEOREM: pr-eval-unsolv-subrp
(unsolv-subrp (*fn*) $\wedge$ (*vargs* = ev ('list, *args*, *va*, *fa*, *n*)))
$\rightarrow$ (ev ('al, cons (*fn*, *args*), *va*, *fa*, *n*)
= **if** btmp (*vargs*) **then** BTM
else unsolv-apply-subr (*fn*, *vargs*) **endif**)

THEOREM: pr-eval-if
(*vx1* = ev ('al, *x1*, *va*, *fa*, *n*))
$\rightarrow$ (ev ('al, list ('if, *x1*, *x2*, *x3*), *va*, *fa*, *n*)
= **if** btmp (*vx1*) **then** BTM
elseif *vx1* **then** ev ('al, *x2*, *va*, *fa*, *n*)
else ev ('al, *x3*, *va*, *fa*, *n*) **endif**)

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

THEOREM: pr-eval-nlistp
(ev ('al, 0, *va*, *fa*, *n*) = 0)
$\wedge$ (ev ('al, $1 + n$, *va*, *fa*, *n*) = $(1 + n)$)
$\wedge$ (ev ('al, pack (*x*), *va*, *fa*, *n*)
= **if** pack (*x*) = 't **then t**
elseif pack (*x*) = 'f **then f**
elseif pack (*x*) = 'nil **then nil**
else get (pack (*x*), *va*) **endif**)

THEOREM: evlist-nil
ev ('list, **nil**, *va*, *fa*, *n*) = **nil**

THEOREM: evlist-cons
((*vx* = ev ('al, *x*, *va*, *fa*, *n*)) $\wedge$ (*vl* = ev ('list, *l*, *va*, *fa*, *n*)))
$\rightarrow$ (ev ('list, cons (*x*, *l*), *va*, *fa*, *n*)
= **if** btmp (*vx*) **then** BTM
elseif btmp (*vl*) **then** BTM
else cons (*vx*, *vl*) **endif**)

EVENT: Enable unsolv-subrp; name this event 'unsolv-subrp-off'.

EVENT: Enable ev; name this event 'ev-off'.

DEFINITION:
cnb (x)
= **if** listp (x) **then** cnb (car (x)) $\wedge$ cnb (cdr (x))
else $\neg$ btmp (x) **endif**

THEOREM: cnb-nbtm
cnb $(x) \rightarrow$ (btmp $(x) =$ **f**)

THEOREM: cnb-cons
(cnb (cons $(a,\ b)$) = (cnb (a) $\wedge$ cnb (b)))
$\wedge$ (cnb $(x) \rightarrow$ cnb (car (x)))
$\wedge$ (cnb $(x) \rightarrow$ cnb (cdr (x)))

THEOREM: cnb-litatom
litatom $(x) \rightarrow$ cnb (x)

THEOREM: cnb-numberp
$(x \in \mathbf{N}) \rightarrow$ cnb (x)

EVENT: Enable cnb; name this event 'cnb-off'.

THEOREM: get-tmi-fa
(get (`'tm`, tmi-fa (tm)) = list (**nil**, kwote (tm)))
$\wedge$ (get (`'instr`, tmi-fa (tm)) = INSTR-DEFN)
$\wedge$ (get (`'new-tape`, tmi-fa (tm)) = NEW-TAPE-DEFN)
$\wedge$ (get (`'tmi`, tmi-fa (tm)) = TMI-DEFN)

EVENT: Enable tmi-fa; name this event 'tmi-fa-off'.

DEFINITION:
instrn $(st,\ sym,\ tm,\ n)$
= **if** $n \simeq$ `0` **then** BTM
elseif listp (tm)
then if $st =$ car (car (tm))
then if $sym =$ car (cdr (car (tm))) **then** cdr (cdr (car (tm)))
else instrn $(st,\ sym,$ cdr $(tm),\ n - 1)$ **endif**
else instrn $(st,\ sym,$ cdr $(tm),\ n - 1)$ **endif**
else f endif

DEFINITION:
pr-eval-instr-induction-scheme $(st,\ sym,\ tm\text{–},\ va,\ tm,\ n)$
= **if** $n \simeq$ `0` **then t**
else pr-eval-instr-induction-scheme (`'st`,
`'sym`,

```
'(cdr tm),
list (cons ('st,
           pr-eval (st,
                    va,
                    tmi-fa (tm),
                    n)),
     cons ('sym,
           pr-eval (sym,
                    va,
                    tmi-fa (tm),
                    n)),
     cons ('tm,
           pr-eval (tm–,
                    va,
                    tmi-fa (tm),
                    n))),
tm,
n − 1) endif
```

THEOREM: pr-eval-instr
(cnb (ev ('al, *st*, *va*, tmi-fa (*tm*), *n*))
 ∧ cnb (ev ('al, *sym*, *va*, tmi-fa (*tm*), *n*))
 ∧ cnb (ev ('al, *tm*–, *va*, tmi-fa (*tm*), *n*)))
→ (ev ('al, list ('instr, *st*, *sym*, *tm*–), *va*, tmi-fa (*tm*), *n*)
 = instrn (ev ('al, *st*, *va*, tmi-fa (*tm*), *n*),
           ev ('al, *sym*, *va*, tmi-fa (*tm*), *n*),
           ev ('al, *tm*–, *va*, tmi-fa (*tm*), *n*),
           n))

THEOREM: pr-eval-new-tape
(cnb (ev ('al, *op*, *va*, tmi-fa (*tm*), *n*)) ∧ cnb (ev ('al, *tape*, *va*, tmi-fa (*tm*), *n*)))
→ (ev ('al, list ('new-tape, *op*, *tape*), *va*, tmi-fa (*tm*), *n*)
 = **if** *n* ≃ 0 **then** BTM
   else new-tape (ev ('al, *op*, *va*, tmi-fa (*tm*), *n*),
                      ev ('al, *tape*, *va*, tmi-fa (*tm*), *n*)) **endif**)

THEOREM: cnb-instrn
((¬ btmp (instrn (*st*, *sym*, *tm*, *n*))) ∧ cnb (*tm*)) → cnb (instrn (*st*, *sym*, *tm*, *n*))

THEOREM: cnb-new-tape
(cnb (*op*) ∧ cnb (*tape*)) → cnb (new-tape (*op*, *tape*))

EVENT: Enable new-tape; name this event 'new-tape-off'.

DEFINITION:

```
tmin (st, tape, tm, n)
=  if n ≃ 0 then BTM
   elseif btmp (instrn (st, car (cdr (tape)), tm, n − 1)) then BTM
   elseif instrn (st, car (cdr (tape)), tm, n − 1)
   then tmin (car (cdr (instrn (st, car (cdr (tape)), tm, n − 1))),
              new-tape (car (instrn (st, car (cdr (tape)), tm, n − 1)), tape),
              tm,
              n − 1)
   else tape endif
```

DEFINITION:

```
pr-eval-tmi-induction-scheme (st, tape, tm–, va, tm, n)
=  if n ≃ 0 then t
   else pr-eval-tmi-induction-scheme ('(car
                                        (cdr
                                         (instr st
                                          (car (cdr tape))
                                          tm))),
                                      '(new-tape
                                        (car
                                         (instr st
                                          (car (cdr tape))
                                          tm))
                                        tape),
                                      'tm,
                                      list (cons ('st,
                                                  ev ('al,
                                                      st,
                                                      va,
                                                      tmi-fa (tm),
                                                      n)),
                                           cons ('tape,
                                                 ev ('al,
                                                     tape,
                                                     va,
                                                     tmi-fa (tm),
                                                     n)),
                                           cons ('tm,
                                                 ev ('al,
                                                     tm–,
                                                     va,
                                                     tmi-fa (tm),
                                                     n))),
                                      tm,
```

$n - 1)$ **endif**

THEOREM: pr-eval-tmi
(cnb (ev ('al, *st*, *va*, tmi-fa (*tm*), *n*))
∧ cnb (ev ('al, *tape*, *va*, tmi-fa (*tm*), *n*))
∧ cnb (ev ('al, *tm*–, *va*, tmi-fa (*tm*), *n*)))
→ (ev ('al, list ('tmi, *st*, *tape*, *tm*–), *va*, tmi-fa (*tm*), *n*)
= tmin (ev ('al, *st*, *va*, tmi-fa (*tm*), *n*),
ev ('al, *tape*, *va*, tmi-fa (*tm*), *n*),
ev ('al, *tm*–, *va*, tmi-fa (*tm*), *n*),
n))

THEOREM: pr-eval-tmi-x
(cnb (*st*) ∧ cnb (*tape*) ∧ cnb (*tm*))
→ (ev ('al, tmi-x (*st*, *tape*), **nil**, tmi-fa (*tm*), *n*)
= **if** *n* ≃ 0 **then** BTM
else tmin (*st*, *tape*, *tm*, *n*) **endif**)

EVENT: Enable tmi-x; name this event 'tmi-x-off'.

THEOREM: instrn-instr
(length (*tm*) < *n*) → (instrn (*st*, *sym*, *tm*, *n*) = instr (*st*, *sym*, *tm*))

THEOREM: nbtmp-instr
turing-machine (*tm*) → (¬ btmp (instr (*st*, *sym*, *tm*)))

THEOREM: instrn-non-f
(turing-machine (*tm*) ∧ (*n* ≤ length (*tm*))) → instrn (*st*, *sym*, *tm*, *n*)

THEOREM: tmin-btm
(turing-machine (*tm*) ∧ (*n* ≤ length (*tm*)))
→ (tmin (*st*, *tape*, *tm*, *n*) = BTM)

THEOREM: tmin-tmi
turing-machine (*tm*)
→ (tmi (*st*, *tape*, *tm*, *k*) = tmin (*st*, *tape*, *tm*, *k* + (1 + length (*tm*))))

THEOREM: symbol-cnb
symbol (*sym*) → cnb (*sym*)

EVENT: Enable symbol; name this event 'symbol-off'.

THEOREM: half-tape-cnb
half-tape (*x*) → cnb (*x*)

THEOREM: tape-cnb
$\text{tape}\,(x) \rightarrow \text{cnb}\,(x)$

EVENT: Enable tape; name this event 'tape-off'.

THEOREM: operation-cnb
$\text{operation}\,(op) \rightarrow \text{cnb}\,(op)$

EVENT: Enable operation; name this event 'operation-off'.

THEOREM: turing-machine-cnb
$\text{turing-machine}\,(tm) \rightarrow \text{cnb}\,(tm)$

EVENT: Enable turing-machine; name this event 'turing-machine-off'.

THEOREM: turing-completeness-of-lisp

$$
\begin{array}{l}
(\text{state}\,(st) \wedge \text{tape}\,(tape) \wedge \text{turing-machine}\,(tm)) \\
\rightarrow \quad (((\neg\ \text{btmp}\,(\text{pr-eval}\,(\text{tmi-x}\,(st,\ tape),\ \textbf{nil},\ \text{tmi-fa}\,(tm),\ n))) \\
\qquad \rightarrow \quad (\neg\ \text{btmp}\,(\text{tmi}\,(st,\ tape,\ tm,\ \text{tmi-k}\,(st,\ tape,\ tm,\ n))))) \\
\qquad \wedge \quad ((\neg\ \text{btmp}\,(\text{tmi}\,(st,\ tape,\ tm,\ k))) \\
\qquad\qquad \rightarrow \quad (\text{tmi}\,(st,\ tape,\ tm,\ k) \\
\qquad\qquad\qquad = \quad \text{pr-eval}\,(\text{tmi-x}\,(st,\ tape), \\
\qquad\qquad\qquad\qquad\qquad\qquad \textbf{nil}, \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{tmi-fa}\,(tm), \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{tmi-n}\,(st,\ tape,\ tm,\ k)))))
\end{array}
$$

Index