```
#|

 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.

DEFINITION:
from-to (i, j)
$=$ **if** $j < i$ **then nil**
elseif $\text{fix}(i) = \text{fix}(j)$ **then** $\text{list}(\text{fix}(j))$
else append (from-to $(i, j - 1)$, $\text{list}(j)$) **endif**

THEOREM: plus-right-id2
$(y \notin \mathbf{N}) \rightarrow ((x + y) = \text{fix}(x))$

THEOREM: plus-add1
$(x + (1 + y))$
$=$ **if** $y \in \mathbf{N}$ **then** $1 + (x + y)$

$\quad$ **else** $1 + x$ **endif**

THEOREM: commutativity2-of-plus
$(x + (y + z)) = (y + (x + z))$

THEOREM: commutativity-of-plus
$(x + y) = (y + x)$

THEOREM: associativity-of-plus
$((x + y) + z) = (x + (y + z))$

THEOREM: plus-equal-0
$((a + b) = \mathsf{0}) = ((a \simeq \mathsf{0}) \wedge (b \simeq \mathsf{0}))$

THEOREM: difference-x-x
$(x - x) = \mathsf{0}$

THEOREM: difference-plus
$(((x + y) - x) = \text{fix}\,(y)) \wedge (((y + x) - x) = \text{fix}\,(y))$

THEOREM: plus-cancellation
$((a + b) = (a + c)) = (\text{fix}\,(b) = \text{fix}\,(c))$

THEOREM: difference-0
$(y \not< x) \rightarrow ((x - y) = \mathsf{0})$

THEOREM: equal-difference-0
$(\mathsf{0} = (x - y)) = (y \not< x)$

THEOREM: difference-cancellation-0
$(x = (x - y)) = ((x \in \mathbf{N}) \wedge ((x = \mathsf{0}) \vee (y \simeq \mathsf{0})))$

THEOREM: difference-cancellation-1
$((x - y) = (z - y))$
$=$ **if** $x < y$ **then** $y \not< z$
$\quad$ **elseif** $z < y$ **then** $y \not< x$
$\quad$ **else** $\text{fix}\,(x) = \text{fix}\,(z)$ **endif**

DEFINITION:
$\text{delete}\,(x,\ y)$
$=$ **if** $\text{listp}\,(y)$
$\quad$ **then if** $x = \text{car}\,(y)$ **then** $\text{cdr}\,(y)$
$\quad\quad$ **else** $\text{cons}\,(\text{car}\,(y),\ \text{delete}\,(x,\ \text{cdr}\,(y)))$ **endif**
$\quad$ **else** y **endif**

DEFINITION:
subbagp $(x,\ y)$
$=$ **if** listp (x)
then if car $(x) \in y$ **then** subbagp (cdr (x), delete (car (x), y))
else f endif
else t endif

DEFINITION:
bagdiff $(x,\ y)$
$=$ **if** listp (y)
then if car $(y) \in x$ **then** bagdiff (delete (car (y), x), cdr (y))
else bagdiff (x, cdr (y)) **endif**
else x **endif**

DEFINITION:
bagint $(x,\ y)$
$=$ **if** listp (x)
then if car $(x) \in y$
then cons (car (x), bagint (cdr (x), delete (car (x), y)))
else bagint (cdr (x), y) **endif**
else nil endif

THEOREM: delete-non-member
$(x \notin y) \rightarrow$ (delete $(x,\ y) = y$)

THEOREM: member-delete
($x \in$ delete $(u,\ v)$) $\rightarrow (x \in v)$

THEOREM: commutativity-of-delete
delete (x, delete $(y,\ z)$) $=$ delete (y, delete $(x,\ z)$)

THEOREM: subbagp-delete
subbagp (x, delete $(u,\ y)$) $\rightarrow$ subbagp $(x,\ y)$

THEOREM: subbagp-cdr1
subbagp $(x,\ y) \rightarrow$ subbagp (cdr (x), y)

THEOREM: subbagp-cdr2
subbagp (x, cdr (y)) $\rightarrow$ subbagp $(x,\ y)$

THEOREM: subbagp-bagint1
subbagp (bagint $(x,\ y)$, x)

THEOREM: subbagp-bagint2
subbagp (bagint $(x,\ y)$, y)

DEFINITION:

```
plus-fringe (x)
=   if listp (x) ∧ (car (x) = 'plus)
    then append (plus-fringe (cadr (x)), plus-fringe (caddr (x)))
    else cons (x, nil) endif
```

DEFINITION:

```
plus-tree (l)
=   if l ≃ nil then ''0
    elseif cdr (l) ≃ nil then list ('fix, car (l))
    elseif cddr (l) ≃ nil then list ('plus, car (l), cadr (l))
    else list ('plus, car (l), plus-tree (cdr (l))) endif
```

DEFINITION:

```
cancel (x)
=   if listp (x) ∧ (car (x) = 'equal)
    then if listp (cadr (x))
            ∧  (caadr (x) = 'plus)
            ∧  listp (caddr (x))
            ∧  (caaddr (x) = 'plus)
         then list ('equal,
                   plus-tree (bagdiff (plus-fringe (cadr (x)),
                                       bagint (plus-fringe (cadr (x)),
                                               plus-fringe (caddr (x))))),
                   plus-tree (bagdiff (plus-fringe (caddr (x)),
                                       bagint (plus-fringe (cadr (x)),
                                               plus-fringe (caddr (x))))))
         elseif listp (cadr (x))
                ∧  (caadr (x) = 'plus)
                ∧  (caddr (x) ∈ plus-fringe (cadr (x)))
         then list ('if,
                   list ('numberp, caddr (x)),
                   list ('equal,
                        plus-tree (delete (caddr (x), plus-fringe (cadr (x)))),
                        ''0),
                   list ('quote, f))
         elseif listp (caddr (x))
                ∧  (caaddr (x) = 'plus)
                ∧  (cadr (x) ∈ plus-fringe (caddr (x)))
         then list ('if,
                   list ('numberp, cadr (x)),
                   list ('equal,
                        ''0,
                        plus-tree (delete (cadr (x), plus-fringe (caddr (x))))),
```

$\text{list}\,(\texttt{'quote}, \mathbf{f}))$
else x **endif**
else x **endif**

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

THEOREM: numberp-eval\$-plus
$(\text{listp}\,(x) \wedge (\text{car}\,(x) = \texttt{'plus})) \rightarrow (\text{eval\$}\,(\mathbf{t},\ x,\ a) \in \mathbf{N})$

THEOREM: numberp-eval\$-plus-tree
$\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(l),\ a) \in \mathbf{N}$

THEOREM: member-implies-plus-tree-greatereqp
$(x \in y) \rightarrow (\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(y),\ a) \not< \text{eval\$}\,(\mathbf{t},\ x,\ a))$

THEOREM: plus-tree-delete
$\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(\text{delete}\,(x,\ y)),\ a)$
$= \quad \textbf{if}\ x \in y\ \ \textbf{then}\ \text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(y),\ a) - \text{eval\$}\,(\mathbf{t},\ x,\ a)$
$\quad\ \ \textbf{else}\ \text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(y),\ a)\ \textbf{endif}$

THEOREM: subbagp-implies-plus-tree-greatereqp
$\text{subbagp}\,(x,\ y) \rightarrow (\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(y),\ a) \not< \text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(x),\ a))$

THEOREM: plus-tree-bagdiff
$\text{subbagp}\,(x,\ y)$
$\rightarrow \quad (\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(\text{bagdiff}\,(y,\ x)),\ a)$
$\qquad\quad = \quad (\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(y),\ a) - \text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(x),\ a)))$

THEOREM: numberp-eval\$-bridge
$(\text{eval\$}\,(\mathbf{t},\ z,\ a) = \text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(x),\ a)) \rightarrow (\text{eval\$}\,(\mathbf{t},\ z,\ a) \in \mathbf{N})$

THEOREM: bridge-to-subbagp-implies-plus-tree-greatereqp
$(\text{subbagp}\,(y,\ \text{plus-fringe}\,(z))$
$\ \wedge \quad (\text{eval\$}\,(\mathbf{t},\ z,\ a) = \text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(\text{plus-fringe}\,(z)),\ a)))$
$\rightarrow \quad ((\text{eval\$}\,(\mathbf{t},\ z,\ a) < \text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(y),\ a)) = \mathbf{f})$

THEOREM: eval\$-plus-tree-append
$\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(\text{append}\,(x,\ y)),\ a)$
$= \quad (\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(x),\ a) + \text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(y),\ a))$

THEOREM: plus-tree-plus-fringe
$\text{eval\$}\,(\mathbf{t},\ \text{plus-tree}\,(\text{plus-fringe}\,(x)),\ a) = \text{fix}\,(\text{eval\$}\,(\mathbf{t},\ x,\ a))$

THEOREM: member-implies-numberp
$((c \in \text{plus-fringe}\,(x)) \wedge (\text{eval\$}\,(\mathbf{t},\ c,\ a) \in \mathbf{N})) \rightarrow (\text{eval\$}\,(\mathbf{t},\ x,\ a) \in \mathbf{N})$

THEOREM: cadr-eval$-list

$$(\mathrm{car}\,(\mathrm{eval\$}\,(\texttt{'list},\ x,\ a)) = \mathrm{eval\$}\,(\mathbf{t},\ \mathrm{car}\,(x),\ a)) \land (\mathrm{cdr}\,(\mathrm{eval\$}\,(\texttt{'list},\ x,\ a)) = \textbf{if}\ \mathrm{listp}\,(x)\ \textbf{then}\ \mathrm{eval\$}\,(\texttt{'list},\ \mathrm{cdr}\,(x),\ a)\ \textbf{else}\ \texttt{0}\ \textbf{endif})$$

THEOREM: eval$-quote

$$\mathrm{eval\$}\,(\mathbf{t},\ \mathrm{cons}\,(\texttt{'quote},\ \mathit{args}),\ a) = \mathrm{car}\,(\mathit{args})$$

THEOREM: listp-eval$

$$\mathrm{listp}\,(\mathrm{eval\$}\,(\texttt{'list},\ x,\ a)) = \mathrm{listp}\,(x)$$

THEOREM: correctness-of-cancel

$$\mathrm{eval\$}\,(\mathbf{t},\ x,\ a) = \mathrm{eval\$}\,(\mathbf{t},\ \mathrm{cancel}\,(x),\ a)$$

DEFINITION:

$$\mathrm{reverse}\,(x) = \textbf{if}\ \mathrm{listp}\,(x)\ \textbf{then}\ \mathrm{append}\,(\mathrm{reverse}\,(\mathrm{cdr}\,(x)),\ \mathrm{cons}\,(\mathrm{car}\,(x),\ \textbf{nil}))\ \textbf{else nil endif}$$

THEOREM: associativity-of-append

$$\mathrm{append}\,(\mathrm{append}\,(x,\ y),\ z) = \mathrm{append}\,(x,\ \mathrm{append}\,(y,\ z))$$

DEFINITION:

$$\mathrm{plistp}\,(x) = \textbf{if}\ x \simeq \textbf{nil}\ \textbf{then}\ x = \textbf{nil}\ \textbf{else}\ \mathrm{plistp}\,(\mathrm{cdr}\,(x))\ \textbf{endif}$$

THEOREM: append-right-id

$$\mathrm{plistp}\,(x) \to (\mathrm{append}\,(x,\ \textbf{nil}) = x)$$

THEOREM: plistp-reverse

$$\mathrm{plistp}\,(\mathrm{reverse}\,(x))$$

THEOREM: append-reverse

$$\mathrm{reverse}\,(\mathrm{append}\,(a,\ b)) = \mathrm{append}\,(\mathrm{reverse}\,(b),\ \mathrm{reverse}\,(a))$$

THEOREM: times-zero2

$$(y \notin \mathbf{N}) \to ((x * y) = \texttt{0})$$

THEOREM: distributivity-of-times-over-plus

$$(x * (y + z)) = ((x * y) + (x * z))$$

THEOREM: times-add1

$$(x * (1 + y)) = \textbf{if}\ y \in \mathbf{N}\ \textbf{then}\ x + (x * y)\ \textbf{else}\ \mathrm{fix}\,(x)\ \textbf{endif}$$

THEOREM: commutativity-of-times
$(x * y) = (y * x)$

THEOREM: commutativity2-of-times
$(x * (y * z)) = (y * (x * z))$

THEOREM: associativity-of-times
$((x * y) * z) = (x * (y * z))$

THEOREM: equal-times-0
$((x * y) = 0) = ((x \simeq 0) \vee (y \simeq 0))$

EVENT: Add the shell *push*, with recognizer function symbol *stackp* and 2 accessors: *top*, with type restriction (none-of) and default value zero; *pop*, with type restriction (none-of) and default value zero.

CONSERVATIVE AXIOM: numberp-call
$\text{call}\,(fn,\ x,\ y) \in \mathbf{N}$

Simultaneously, we introduce the new function symbols *call* and *getvalue*.

DEFINITION:
expressionp (x)
= **if** listp (x)
    then if listp (car (x)) **then f**
        elseif listp (cdr (x))
        then if listp (cddr (x))
            then if expressionp (cadr (x)) **then** expressionp (caddr (x))
                else f endif
            else f endif
        else f endif
    else t endif

THEOREM: cadr-crock
$\text{listp}\,(\text{cddr}\,(x)) \rightarrow (\text{count}\,(\text{cadr}\,(x)) < \text{count}\,(x))$

DEFINITION:
term-eval (*form*, *envrn*)
= **if** *form* $\in \mathbf{N}$ **then** *form*
    elseif listp (cddr (*form*))
    then call (car (*form*),
            term-eval (cadr (*form*), *envrn*),
            term-eval (caddr (*form*), *envrn*))
    else getvalue (*form*, *envrn*) **endif**

DEFINITION:
```
optimize (form)
=  if listp (cddr (form))
   then if optimize (cadr (form)) ∈ N
        then if optimize (caddr (form)) ∈ N
             then call (car (form),
                        optimize (cadr (form)),
                        optimize (caddr (form)))
             else list (car (form),
                        optimize (cadr (form)),
                        optimize (caddr (form))) endif
        else list (car (form),
                   optimize (cadr (form)),
                   optimize (caddr (form))) endif
   else form endif
```

DEFINITION:
```
codegen (form, ins)
=  if form ∈ N then cons (list ('pushi, form), ins)
   elseif listp (cddr (form))
   then cons (car (form), codegen (caddr (form), codegen (cadr (form), ins)))
   else cons (list ('pushv, form), ins) endif
```

DEFINITION:
compile (*form*) = reverse (codegen (optimize (*form*), **nil**))

THEOREM: formp-optimize
expressionp (x) $\rightarrow$ expressionp (optimize (x))

THEOREM: correctness-of-optimize
expressionp (x) $\rightarrow$ (term-eval (optimize (x), *envrn*) = term-eval (x, *envrn*))

DEFINITION:
```
exec (pc, pds, envrn)
=  if pc ≃ nil then pds
   elseif listp (car (pc))
   then if caar (pc) = 'pushi
        then exec (cdr (pc), push (cadar (pc), pds), envrn)
        else exec (cdr (pc),
                   push (getvalue (cadar (pc), envrn), pds),
                   envrn) endif
   else exec (cdr (pc),
              push (call (car (pc), top (pop (pds)), top (pds)),
                    pop (pop (pds))),
              envrn) endif
```

THEOREM: sequential-execution
$\mathrm{exec}\,(\mathrm{append}\,(x,\ y),\ pds,\ envrn) = \mathrm{exec}\,(y,\ \mathrm{exec}\,(x,\ pds,\ envrn),\ envrn)$

THEOREM: correctness-of-codegen
$\mathrm{expressionp}\,(x)$
$\rightarrow\quad (\mathrm{exec}\,(\mathrm{reverse}\,(\mathrm{codegen}\,(x,\ ins)),\ pds,\ envrn)$
$\quad\quad = \quad \mathrm{push}\,(\text{term-eval}\,(x,\ envrn),\ \mathrm{exec}\,(\mathrm{reverse}\,(ins),\ pds,\ envrn)))$

THEOREM: correctness-of-optimizing-compiler
$\mathrm{expressionp}\,(x)$
$\rightarrow\quad (\mathrm{exec}\,(\mathrm{compile}\,(x),\ pds,\ envrn) = \mathrm{push}\,(\text{term-eval}\,(x,\ envrn),\ pds))$

THEOREM: transitivity-of-lessp
$((x < y) \land (y < z)) \rightarrow (x < z)$

THEOREM: lessp-not-reflexive
$x \not< x$

DEFINITION: $\mathrm{eqp}\,(x,\ y) = (\mathrm{fix}\,(x) = \mathrm{fix}\,(y))$

THEOREM: trichotomy-of-lessp
$((\neg\ \mathrm{eqp}\,(x,\ y)) \land (y \not< x)) \rightarrow (x < y)$

THEOREM: reverse-reverse
$\mathrm{plistp}\,(x) \rightarrow (\mathrm{reverse}\,(\mathrm{reverse}\,(x)) = x)$

DEFINITION:
$\mathrm{flatten}\,(x)$
$= \quad \textbf{if}\ \mathrm{listp}\,(x)\ \ \textbf{then}\ \mathrm{append}\,(\mathrm{flatten}\,(\mathrm{car}\,(x)),\ \mathrm{flatten}\,(\mathrm{cdr}\,(x)))$
$\quad\quad \textbf{else}\ \mathrm{cons}\,(x,\ \textbf{nil})\ \textbf{endif}$

DEFINITION:
$\text{mc-flatten}\,(x,\ y)$
$= \quad \textbf{if}\ \mathrm{listp}\,(x)\ \ \textbf{then}\ \text{mc-flatten}\,(\mathrm{car}\,(x),\ \text{mc-flatten}\,(\mathrm{cdr}\,(x),\ y))$
$\quad\quad \textbf{else}\ \mathrm{cons}\,(x,\ y)\ \textbf{endif}$

THEOREM: flatten-mc-flatten
$\text{mc-flatten}\,(x,\ y) = \mathrm{append}\,(\mathrm{flatten}\,(x),\ y)$

THEOREM: member-append
$(x \in \mathrm{append}\,(a,\ b)) = ((x \in a) \lor (x \in b))$

THEOREM: member-reverse
$(x \in \mathrm{reverse}\,(y)) = (x \in y)$

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

THEOREM: length-reverse
length (reverse (x)) $=$ length (x)

DEFINITION:
intersect (x, y)
$=$ **for** *x1* **in** x
when *x1* $\in y$
collect *x1* **endfor**

THEOREM: member-intersect
litatom (v)
$\rightarrow$ (($a \in$ for (v,
lst,
p,
'collect,
v,
alist))
$=$ (($a \in$ *lst*) $\wedge$ eval\$ (**t**, p, cons (cons (v, a), *alist*))))

THEOREM: member-union
$(a \in (b \cup c)) = ((a \in b) \vee (a \in c))$

DEFINITION:
subsetp (x, y)
$=$ **if** listp (x)
then if car $(x) \in y$ **then** subsetp (cdr (x), y)
else f endif
else t endif

THEOREM: subsetp-union
subsetp $(a, b) \rightarrow ((a \cup b) = b)$

THEOREM: subsetp-intersect
(plistp (a) $\wedge$ subsetp (a, b)) $\rightarrow$ (intersect $(a, b) = a$)

DEFINITION:
nth (x, n)
$=$ **if** $n \simeq 0$ **then** x
else nth (cdr (x), $n - 1$) **endif**

DEFINITION: greatereqp $(x, y) = (x \not< y)$

THEOREM: transitivity-of-leq
$((x \leq y) \land (y \leq z)) \rightarrow (x \leq z)$

DEFINITION:
ordered (l)
$=$ **if** listp (l)
then if listp (cdr (l))
then if cadr $(l) <$ car (l) **then f**
else ordered (cdr (l)) **endif**
else t endif
else t endif

DEFINITION:
addtolist (x, l)
$=$ **if** listp (l)
then if $x <$ car (l) **then** cons (x, l)
else cons (car (l), addtolist $(x$, cdr (l))) **endif**
else cons $(x$, **nil**$)$ **endif**

DEFINITION:
sort (l)
$=$ **if** listp (l) **then** addtolist (car (l), sort (cdr (l)))
else nil endif

DEFINITION: boolean $(x) = ((x = \mathbf{t}) \lor (x = \mathbf{f}))$

THEOREM: iff-equal-equal
$(\mathrm{boolean}(p) \land \mathrm{boolean}(q)) \rightarrow ((p \leftrightarrow q) = (p = q))$

THEOREM: nth-0
$\mathrm{nth}(0, i) = 0$

THEOREM: nth-nil
nth (**nil**, i)
$=$ **if** $i \simeq 0$ **then nil**
else 0 **endif**

THEOREM: nth-append1
$\mathrm{nth}(a, i + j) = \mathrm{nth}(\mathrm{nth}(a, i), j)$

THEOREM: associativity-of-equal
$(\mathrm{boolean}(a) \land (\mathrm{boolean}(b) \land \mathrm{boolean}(c)))$
$\rightarrow \quad (((a = b) = c) = (a = (b = c)))$

Definition:
odd (x)
$= \ $ **if** $x \simeq \mathtt{0}$ **then f**
elseif $(x - 1) \simeq \mathtt{0}$ **then t**
else odd $((x - 1) - 1)$ **endif**

Definition:
even1 (x)
$= \ $ **if** $x \simeq \mathtt{0}$ **then t**
else odd $(x - 1)$ **endif**

Definition:
even2 (x)
$= \ $ **if** $x \simeq \mathtt{0}$ **then t**
elseif $(x - 1) \simeq \mathtt{0}$ **then f**
else even2 $((x - 1) - 1)$ **endif**

Definition:
double (i)
$= \ $ **if** $i \simeq \mathtt{0}$ **then** $\mathtt{0}$
else $1 + (1 + \text{double}\,(i - 1))$ **endif**

Theorem: even1-double
even1 (double (i))

Definition:
half (i)
$= \ $ **if** $i \simeq \mathtt{0}$ **then** $\mathtt{0}$
elseif $(i - 1) \simeq \mathtt{0}$ **then** $\mathtt{0}$
else $1 + \text{half}\,((i - 1) - 1)$ **endif**

Theorem: half-double
$(i \in \mathbf{N}) \rightarrow (\text{half}\,(\text{double}\,(i)) = i)$

Theorem: double-half
$((i \in \mathbf{N}) \wedge \text{even1}\,(i)) \rightarrow (\text{double}\,(\text{half}\,(i)) = i)$

Theorem: double-times-2
$\text{double}\,(i) = (\mathtt{2} * i)$

Theorem: subsetp-cons
$\text{subsetp}\,(x,\ y) \rightarrow \text{subsetp}\,(x,\ \text{cons}\,(z,\ y))$

Definition:
last (x)
$= \ $ **if** listp (x)
then if listp (cdr (x)) **then** last (cdr (x))
else x **endif**
else x **endif**

THEOREM: last-append
$\text{last}(\text{append}(a,\ b))$
$= \ \textbf{if}\ \text{listp}(b)\ \textbf{then}\ \text{last}(b)$
$\quad \textbf{elseif}\ \text{listp}(a)\ \textbf{then}\ \text{cons}(\text{car}(\text{last}(a)),\ b)$
$\quad \textbf{else}\ b\ \textbf{endif}$

THEOREM: last-reverse
$\text{listp}(a) \rightarrow (\text{last}(\text{reverse}(a)) = \text{cons}(\text{car}(a),\ \textbf{nil}))$

DEFINITION:
$\exp(i,\ j)$
$= \ \textbf{if}\ j \simeq \mathtt{0}\ \textbf{then}\ \mathtt{1}$
$\quad \textbf{else}\ i * \exp(i,\ j - 1)\ \textbf{endif}$

THEOREM: exp-plus
$\exp(i,\ j + k) = (\exp(i,\ j) * \exp(i,\ k))$

THEOREM: even1-even2
$\text{even1}(x) = \text{even2}(x)$

THEOREM: leq-nth
$\text{length}(\text{nth}(l,\ i)) \leq \text{length}(l)$

THEOREM: member-sort
$(a \in \text{sort}(b)) = (a \in b)$

THEOREM: length-sort
$\text{length}(\text{sort}(a)) = \text{length}(a)$

DEFINITION:
$\text{count-list}(a,\ l)$
$= \ \textbf{for}\ x\ \textbf{in}\ l$
$\quad \textbf{count}\ a = x\ \textbf{endfor}$

THEOREM: count-list-sort
$\text{count-list}(a,\ \text{sort}(l)) = \text{count-list}(a,\ l)$

THEOREM: ordered-append
$\text{ordered}(\text{append}(a,\ b)) \rightarrow \text{ordered}(a)$

THEOREM: leq-half
$\text{half}(i) \leq i$

DEFINITION:
$\text{number-listp}(l)$
$= \ \textbf{if}\ \text{listp}(l)\ \textbf{then}\ (\text{car}(l) \in \mathbf{N}) \wedge \text{number-listp}(\text{cdr}(l))$
$\quad \textbf{else}\ l = \textbf{nil}\ \textbf{endif}$

THEOREM: ordered-sort
ordered (sort (x))

THEOREM: addtolist-of-ordered-number-list
(ordered (x) $\wedge$ number-listp (x) $\wedge$ ($i \in \mathbf{N}$) $\wedge$ (car (x) $\not<$ i))
$\rightarrow$ (addtolist (i, x) $=$ cons (i, x))

THEOREM: sort-of-ordered-number-list
(ordered (x) $\wedge$ number-listp (x)) $\rightarrow$ (sort (x) $=$ x)

DEFINITION:
xor (p, q)
$=$ **if** q
then if p **then f**
else t endif
else p $=$ **t endif**

THEOREM: crock-due-to-lack-of-bounce
(x $=$ sort (l)) $\rightarrow$ ordered (x)

THEOREM: sort-ordered
number-listp (l) $\rightarrow$ ((sort (l) $=$ l) $=$ ordered (l))

DEFINITION:
subst (x, y, z)
$=$ **if** $y = z$ **then** x
elseif listp (z) **then** cons (subst (x, y, car (z)), subst (x, y, cdr (z)))
else z **endif**

THEOREM: subst-a-a
subst (a, a, b) $=$ b

DEFINITION:
occur (x, y)
$=$ **if** $x = y$ **then t**
elseif listp (y)
then if occur (x, car (y)) **then t**
else occur (x, cdr (y)) **endif**
else f endif

THEOREM: occur-subst
($\neg$ occur (a, b)) $\rightarrow$ (subst (c, a, b) $=$ b)

DEFINITION:
countps-loop (l, $pred$, ans)
$=$ **if** listp (l)

then if apply\$ $(pred,$ list (car $(l)))$
  then countps-loop (cdr (l), $pred$, $1 + ans$)
  else countps-loop (cdr (l), $pred$, ans) **endif**
else ans **endif**

DEFINITION: countps- $(l, pred)$ = countps-loop $(l, pred,$ `0`$)$

THEOREM: countps-countps
$((n \in \mathbf{N}) \wedge$ litatom $(x) \wedge (pred \neq$ `'quote`$))$
$\rightarrow$ (countps-loop $(l, pred, n)$
  $=$ $(n +$ for $(x,$
    $l,$
    list (`'quote`, **t**),
    `'count`,
    list $(pred, x)$,
    nil)))

DEFINITION:
fact (i)
$=$ **if** $i \simeq$ `0` **then** `1`
  else $i *$ fact $(i - 1)$ **endif**

DEFINITION:
fact-loop (i, ans)
$=$ **if** $i \simeq$ `0` **then** ans
  else fact-loop $(i - 1, i * ans)$ **endif**

DEFINITION: fact- (i) = fact-loop $(i,$ `1`$)$

THEOREM: fact-loop-fact
$(i \in \mathbf{N}) \rightarrow$ (fact-loop $(j, i) = (i *$ fact $(j)))$

THEOREM: fact-fact
fact- (i) = fact (i)

THEOREM: fact-from-to
fact (n)
$=$ **for** i **in** from-to (`1`, n)
  multiply i **endfor**

DEFINITION:
reverse-loop (x, ans)
$=$ **if** listp (x) **then** reverse-loop (cdr (x), cons (car (x), ans))
  else ans **endif**

DEFINITION: reverse- (x) = reverse-loop $(x$, **nil**$)$

THEOREM: reverse-loop-append-reverse
reverse-loop $(x,\ y)$ = append (reverse $(x),\ y)$

THEOREM: reverse-loop-reverse
reverse-loop $(x,$ **nil**$)$ = reverse (x)

THEOREM: reverse-append
reverse- (append $(a,\ b))$ = append (reverse- (b), reverse- $(a))$

THEOREM: reverse-reverse-
plistp $(x) \rightarrow$ (reverse- (reverse- $(x)) = x)$

DEFINITION:
sort-lp $(x,\ y)$
= **if** listp (x) **then** sort-lp (cdr (x), addtolist (car $(x),\ y))$
else y **endif**

THEOREM: ordered-addtolist
ordered $(y) \rightarrow$ ordered (addtolist $(x,\ y))$

THEOREM: ordered-sort-lp
ordered $(y) \rightarrow$ ordered (sort-lp $(x,\ y))$

THEOREM: numberp-count
for $(x$,
l,
p,
'count,
b,
$a) \in \mathbf{N}$

THEOREM: count-sort-lp
count-list $(z$, sort-lp $(x,\ y))$ = (count-list $(z,\ x)$ + count-list $(z,\ y))$

THEOREM: append-cancellation
(append $(a,\ b)$ = append $(a,\ c)) = (b = c)$

THEOREM: equal-lessp
$((x < y) = z)$
= **if** $x < y$ **then** $\mathbf{t} = z$
else $\mathbf{f} = z$ **endif**

THEOREM: difference-elim
$((y \in \mathbf{N}) \wedge (y \not< x)) \rightarrow ((x + (y - x)) = y)$

DEFINITION:
power-eval $(l,\ base)$
$=$ **if** listp (l) **then** car (l) + $(base *$ power-eval (cdr $(l),\ base))$
else 0 **endif**

DEFINITION:
big-plus1 $(l,\ i,\ base)$
$=$ **if** listp (l)
then if $i \simeq 0$ **then** l
else cons ((car $(l) + i)$ **mod** $base$,
big-plus1 (cdr (l), (car $(l) + i) \div base,\ base)$) **endif**
else cons $(i,$ **nil**$)$ **endif**

THEOREM: remainder-quotient
$((x \textbf{ mod } y) + (y * (x \div y))) = \text{fix}(x)$

THEOREM: power-eval-big-plus1
power-eval (big-plus1 $(l,\ i,\ base),\ base) =$ (power-eval $(l,\ base) + i)$

DEFINITION:
big-plus $(x,\ y,\ i,\ base)$
$=$ **if** listp (x)
then if listp (y)
then cons $((i +$ (car (x) + car $(y)))$ **mod** $base$,
big-plus (cdr (x),
cdr (y),
$(i +$ (car (x) + car $(y))) \div base$,
$base$))
else big-plus1 $(x,\ i,\ base)$ **endif**
else big-plus1 $(y,\ i,\ base)$ **endif**

THEOREM: power-eval-big-plus
power-eval (big-plus $(x,\ y,\ i,\ base),\ base)$
$=$ $(i +$ (power-eval $(x,\ base)$ + power-eval $(y,\ base)))$

THEOREM: remainder-wrt-1
$(y \textbf{ mod } 1) = 0$

THEOREM: remainder-wrt-12
$(x \notin \mathbf{N}) \rightarrow ((y \textbf{ mod } x) = \text{fix}(y))$

THEOREM: lessp-remainder2
$((x \textbf{ mod } y) < y) = (y \not\simeq 0)$

THEOREM: remainder-x-x
$(x \textbf{ mod } x) = 0$

THEOREM: remainder-quotient-elim
$((y \not\simeq 0) \land (x \in \mathbf{N})) \rightarrow (((x \mathbin{\mathbf{mod}} y) + (y * (x \div y))) = x)$

THEOREM: lessp-times-1
$(i \not\simeq 0) \rightarrow ((i * j) \not< j)$

THEOREM: lessp-times-2
$(i \not\simeq 0) \rightarrow ((j * i) \not< j)$

THEOREM: lessp-quotient1
$((i \div j) < i) = ((i \not\simeq 0) \land ((j \simeq 0) \lor (j \neq 1)))$

THEOREM: lessp-remainder1
$((x \mathbin{\mathbf{mod}} y) < x) = ((y \not\simeq 0) \land (x \not\simeq 0) \land (x \not< y))$

DEFINITION:
power-rep $(i,\ base)$
$=$ **if** $i \simeq 0$ **then nil**
elseif $base \simeq 0$ **then** cons $(i,$ **nil**$)$
elseif $base = 1$ **then** cons $(i,$ **nil**$)$
else cons $(i \mathbin{\mathbf{mod}} base,$ power-rep $(i \div base,\ base))$ **endif**

THEOREM: power-eval-power-rep
power-eval (power-rep $(i,\ base),\ base) =$ fix (i)

THEOREM: correctness-of-big-plus
power-eval (big-plus (power-rep $(i,\ base)$, power-rep $(j,\ base),\ 0,\ base),\ base)$
$= \quad (i + j)$

DEFINITION:
gcd $(x,\ y)$
$=$ **if** $x \simeq 0$ **then** fix (y)
elseif $y \simeq 0$ **then** x
elseif $x < y$ **then** gcd $(x,\ y - x)$
else gcd $(x - y,\ y)$ **endif**

THEOREM: numberp-gcd
gcd $(x,\ y) \in \mathbf{N}$

THEOREM: gcd-equal-0
(gcd $(x,\ y) = 0) = ((x \simeq 0) \land (y \simeq 0))$

THEOREM: gcd-0
gcd $(0,\ y) =$ fix (y)

THEOREM: commutativity-of-gcd
gcd $(x,\ y) =$ gcd $(y,\ x)$

THEOREM: nth-append
$\text{nth}\,(\text{append}\,(a,\ b),\ i) = \text{append}\,(\text{nth}\,(a,\ i),\ \text{nth}\,(b,\ i\ -\ \text{length}\,(a)))$

THEOREM: difference-plus1
$((x\ +\ y)\ -\ x) = \text{fix}\,(y)$

THEOREM: difference-plus2
$((y\ +\ x)\ -\ x) = \text{fix}\,(y)$

THEOREM: difference-plus-cancelation
$((x\ +\ y)\ -\ (x\ +\ z)) = (y\ -\ z)$

THEOREM: times-difference
$(x * (c\ -\ w)) = ((c * x)\ -\ (w * x))$

DEFINITION: $\text{divides}\,(x,\ y) = ((y\ \mathbf{mod}\ x) \simeq \mathsf{0})$

THEOREM: divides-times
$((x * z)\ \mathbf{mod}\ z) = \mathsf{0}$

THEOREM: difference-plus3
$((b\ +\ (a\ +\ c))\ -\ a) = (b\ +\ c)$

THEOREM: difference-add1-cancellation
$((1\ +\ (y\ +\ z))\ -\ z) = (1\ +\ y)$

THEOREM: remainder-add1
$((y \not\simeq \mathsf{0}) \land (y \neq \mathsf{1})) \to (((1\ +\ (x * y))\ \mathbf{mod}\ y) \neq \mathsf{0})$

THEOREM: divides-plus-rewrite1
$(((x\ \mathbf{mod}\ z) = \mathsf{0}) \land ((y\ \mathbf{mod}\ z) = \mathsf{0})) \to (((x\ +\ y)\ \mathbf{mod}\ z) = \mathsf{0})$

THEOREM: divides-plus-rewrite2
$(((x\ \mathbf{mod}\ z) = \mathsf{0}) \land ((y\ \mathbf{mod}\ z) \neq \mathsf{0})) \to (((x\ +\ y)\ \mathbf{mod}\ z) \neq \mathsf{0})$

THEOREM: divides-plus-rewrite
$((x\ \mathbf{mod}\ z) = \mathsf{0}) \to ((((x\ +\ y)\ \mathbf{mod}\ z) = \mathsf{0}) = ((y\ \mathbf{mod}\ z) = \mathsf{0}))$

THEOREM: lessp-plus-cancelation
$((x\ +\ y) < (x\ +\ z)) = (y < z)$

THEOREM: divides-plus-rewrite-commuted
$((x\ \mathbf{mod}\ z) = \mathsf{0}) \to ((((y\ +\ x)\ \mathbf{mod}\ z) = \mathsf{0}) = ((y\ \mathbf{mod}\ z) = \mathsf{0}))$

THEOREM: euclid
$$
\begin{array}{l}
((x\ \mathbf{mod}\ z) = \mathsf{0}) \\
\to \quad ((((y\ -\ x)\ \mathbf{mod}\ z) = \mathsf{0}) \\
\qquad\quad = \quad \mathbf{if}\ x < y\ \ \mathbf{then}\ (y\ \mathbf{mod}\ z) = \mathsf{0} \\
\qquad\qquad\quad\ \mathbf{else}\ \mathbf{t}\ \mathbf{endif})
\end{array}
$$

THEOREM: lessp-times-cancellation
$((x * z) < (y * z)) = ((z \not\simeq \mathsf{0}) \land (x < y))$

THEOREM: lessp-plus-cancellation3
$(y < (x + y)) = (x \not\simeq \mathsf{0})$

THEOREM: distributivity-of-times-over-gcd
$\mathrm{gcd}\,(x * z,\ y * z) = (z * \mathrm{gcd}\,(x,\ y))$

THEOREM: gcd-divides-both
$((x \ \mathbf{mod}\ \mathrm{gcd}\,(x,\ y)) = \mathsf{0}) \land ((y\ \mathbf{mod}\ \mathrm{gcd}\,(x,\ y)) = \mathsf{0})$

THEOREM: gcd-is-the-greatest
$((x \not\simeq \mathsf{0}) \land (y \not\simeq \mathsf{0}) \land \mathrm{divides}\,(z,\ x) \land \mathrm{divides}\,(z,\ y))$
$\rightarrow \quad (z \leq \mathrm{gcd}\,(x,\ y))$

EVENT: Add the shell *cons-if*, with recognizer function symbol *if-exprp* and 3 accessors: *test*, with type restriction (none-of) and default value zero; *left-branch*, with type restriction (none-of) and default value zero; *right-branch*, with type restriction (none-of) and default value zero.

DEFINITION:
assignment (*var*, *alist*)
= **if** *var* = **t** **then t**
  elseif *var* = **f** **then f**
  elseif *alist* $\simeq$ **nil** **then f**
  elseif *var* = caar (*alist*) **then** cdar (*alist*)
  else assignment (*var*, cdr (*alist*)) **endif**

DEFINITION:
value (x, *alist*)
= **if** if-exprp (x)
  then if value (test (x), *alist*) **then** value (left-branch (x), *alist*)
    else value (right-branch (x), *alist*) **endif**
  else assignment (x, *alist*) **endif**

DEFINITION:
if-depth (x)
= **if** if-exprp (x) **then** 1 + if-depth (test (x))
  else 0 **endif**

DEFINITION:
if-complexity (x)
= **if** if-exprp (x)
  then if-complexity (test (x))

$*\quad$ (if-complexity (left-branch (x))
$\quad +\quad$ if-complexity (right-branch (x)))
else 1 **endif**

THEOREM: if-complexity-not-0
if-complexity (x) $\neq$ 0

THEOREM: if-complexity-goes-down1
if-exprp (x) $\rightarrow$ (if-complexity (left-branch (x)) $<$ if-complexity (x))

THEOREM: if-complexity-goes-down2
if-exprp (x) $\rightarrow$ (if-complexity (right-branch (x)) $<$ if-complexity (x))

DEFINITION:
normalize (x)
= **if** if-exprp (x)
then **if** if-exprp (test (x))
then normalize (cons-if (test (test (x)),
cons-if (left-branch (test (x)),
left-branch (x),
right-branch (x)),
cons-if (right-branch (test (x)),
left-branch (x),
right-branch (x))))
else cons-if (test (x),
normalize (left-branch (x)),
normalize (right-branch (x))) **endif**
else x **endif**

DEFINITION:
normalized-if-exprp (x)
= **if** if-exprp (x)
then ($\neg$ if-exprp (test (x)))
$\wedge$ normalized-if-exprp (left-branch (x))
$\wedge$ normalized-if-exprp (right-branch (x))
else **t** **endif**

DEFINITION:
assignedp (var, $alist$)
= **if** var = **t** **then** **t**
elseif var = **f** **then** **t**
elseif $alist \simeq$ **nil** **then** **f**
elseif var = caar ($alist$) **then** **t**
else assignedp (var, cdr ($alist$)) **endif**

DEFINITION: assume-true (var, $alist$) = cons (cons (var, **t**), $alist$)

DEFINITION: assume-false $(var,\ alist) = \text{cons}(\text{cons}(var,\ \mathbf{f}),\ alist)$

DEFINITION:

$$
\begin{array}{l}
\text{tautologyp}(x,\ alist) \\
= \ \ \mathbf{if}\ \text{if-exprp}(x) \\
\quad\ \ \mathbf{then}\ \mathbf{if}\ \text{assignedp}(\text{test}(x),\ alist) \\
\quad\quad\quad \mathbf{then}\ \mathbf{if}\ \text{assignment}(\text{test}(x),\ alist) \\
\quad\quad\quad\quad\quad \mathbf{then}\ \text{tautologyp}(\text{left-branch}(x),\ alist) \\
\quad\quad\quad\quad\quad \mathbf{else}\ \text{tautologyp}(\text{right-branch}(x),\ alist)\ \mathbf{endif} \\
\quad\quad\quad \mathbf{else}\ \text{tautologyp}(\text{left-branch}(x),\ \text{assume-true}(\text{test}(x),\ alist)) \\
\quad\quad\quad\quad \wedge\ \ \text{tautologyp}(\text{right-branch}(x), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \text{assume-false}(\text{test}(x),\ alist))\ \mathbf{endif} \\
\quad\ \ \mathbf{else}\ \text{assignment}(x,\ alist)\ \mathbf{endif}
\end{array}
$$

THEOREM: assignment-append

$$
\begin{array}{l}
\text{assignment}(x,\ \text{append}(a,\ b)) \\
= \ \ \mathbf{if}\ \text{assignedp}(x,\ a)\ \ \mathbf{then}\ \text{assignment}(x,\ a) \\
\quad\ \ \mathbf{else}\ \text{assignment}(x,\ b)\ \mathbf{endif}
\end{array}
$$

THEOREM: value-can-ignore-redundant-assignments

$$
\begin{array}{l}
(((val \leftrightarrow \text{assignment}(var,\ a)) \wedge \text{value}(x,\ a)) \\
\ \ \rightarrow\ \ \text{value}(x,\ \text{cons}(\text{cons}(var,\ val),\ a))) \\
\wedge\ \ (((val \leftrightarrow \text{assignment}(var,\ a)) \wedge (\neg\ \text{value}(x,\ a))) \\
\quad\quad \rightarrow\ \ (\neg\ \text{value}(x,\ \text{cons}(\text{cons}(var,\ val),\ a))))
\end{array}
$$

THEOREM: value-short-cut

$$
\begin{array}{l}
(\text{if-exprp}(x) \wedge \text{normalized-if-exprp}(x)) \\
\rightarrow\ \ (\text{value}(\text{test}(x),\ a) = \text{assignment}(\text{test}(x),\ a))
\end{array}
$$

THEOREM: assignment-implies-assigned

$$\text{assignment}(x,\ a) \rightarrow \text{assignedp}(x,\ a)$$

THEOREM: tautologyp-is-sound

$$(\text{normalized-if-exprp}(x) \wedge \text{tautologyp}(x,\ a1)) \rightarrow \text{value}(x,\ \text{append}(a1,\ a2))$$

DEFINITION: tautology-checker $(x) = \text{tautologyp}(\text{normalize}(x),\ \mathbf{nil})$

DEFINITION:

$$
\begin{array}{l}
\text{falsify1}(x,\ alist) \\
= \ \ \mathbf{if}\ \text{if-exprp}(x) \\
\quad\ \ \mathbf{then}\ \mathbf{if}\ \text{assignedp}(\text{test}(x),\ alist) \\
\quad\quad\quad \mathbf{then}\ \mathbf{if}\ \text{assignment}(\text{test}(x),\ alist) \\
\quad\quad\quad\quad\quad \mathbf{then}\ \text{falsify1}(\text{left-branch}(x),\ alist) \\
\quad\quad\quad\quad\quad \mathbf{else}\ \text{falsify1}(\text{right-branch}(x),\ alist)\ \mathbf{endif} \\
\quad\quad\quad \mathbf{elseif}\ \text{falsify1}(\text{left-branch}(x),\ \text{assume-true}(\text{test}(x),\ alist))
\end{array}
$$

then falsify1 (left-branch (x), assume-true (test (x), *alist*))
else falsify1 (right-branch (x), assume-false (test (x), *alist*)) **endif**
elseif assignedp $(x,\ alist)$
then if assignment $(x,\ alist)$ **then f**
else *alist* **endif**
else cons (cons $(x,$ **f**$)$, *alist*) **endif**

DEFINITION: falsify (x) = falsify1 (normalize (x), **nil**)

THEOREM: falsify1-extends-models
assignedp $(x,\ a)$
$\rightarrow$ (assignment $(x$, falsify1 $(y,\ a))$
= **if** falsify1 $(y,\ a)$ **then** assignment $(x,\ a)$
else x = **t** **endif**)

THEOREM: falsify1-falsifies
(normalized-if-exprp (x) $\wedge$ falsify1 $(x,\ a)$)
$\rightarrow$ (value $(x$, falsify1 $(x,\ a))$ = **f**)

THEOREM: tautologyp-fails-means-falsify1-wins
(normalized-if-exprp (x) $\wedge$ ($\neg$ tautologyp $(x,\ a)$) $\wedge$ a) $\rightarrow$ falsify1 $(x,\ a)$

THEOREM: normalize-is-sound
value (normalize (x), a) = value $(x,\ a)$

THEOREM: normalize-normalizes
normalized-if-exprp (normalize (x))

THEOREM: tautology-checker-completeness-bridge
((value $(y$, falsify1 $(x,\ a))$ = value $(x$, falsify1 $(x,\ a))$)
$\wedge$ falsify1 $(x,\ a)$
$\wedge$ normalized-if-exprp (x))
$\rightarrow$ (value $(y$, falsify1 $(x,\ a))$ = **f**)

THEOREM: tautology-checker-is-complete
($\neg$ tautology-checker (x)) $\rightarrow$ (value $(x$, falsify $(x))$ = **f**)

THEOREM: tautology-checker-soundness-bridge
(tautologyp $(y,\ a1)$
$\wedge$ normalized-if-exprp (y)
$\wedge$ (value $(x,\ a2)$ = value $(y$, append $(a1,\ a2))$))
$\rightarrow$ value $(x,\ a2)$

THEOREM: tautology-checker-is-sound
tautology-checker (x) $\rightarrow$ value $(x,\ a)$

THEOREM: flatten-singleton
$(\text{flatten}(x) = \text{cons}(y, \textbf{nil})) = ((x \simeq \textbf{nil}) \land (x = y))$

DEFINITION:
leftcount (x)
$= \textbf{if } x \simeq \textbf{nil} \textbf{ then } \mathtt{0}$
$\quad \textbf{else } 1 + \text{leftcount}(\text{car}(x)) \textbf{ endif}$

DEFINITION:
gopher (x)
$= \textbf{if } (x \simeq \textbf{nil}) \lor (\text{car}(x) \simeq \textbf{nil}) \textbf{ then } x$
$\quad \textbf{else } \text{gopher}(\text{cons}(\text{caar}(x), \text{cons}(\text{cdar}(x), \text{cdr}(x)))) \textbf{ endif}$

THEOREM: gopher-preserves-count
$\text{count}(x) \not< \text{count}(\text{gopher}(x))$

THEOREM: listp-gopher
$\text{listp}(\text{gopher}(x)) = \text{listp}(x)$

DEFINITION:
samefringe (x, y)
$= \textbf{if } (x \simeq \textbf{nil}) \lor (y \simeq \textbf{nil}) \textbf{ then } x = y$
$\quad \textbf{else } (\text{car}(\text{gopher}(x)) = \text{car}(\text{gopher}(y)))$
$\qquad \land \quad \text{samefringe}(\text{cdr}(\text{gopher}(x)), \text{cdr}(\text{gopher}(y))) \textbf{ endif}$

THEOREM: gopher-returns-leftmost-atom
$\text{car}(\text{gopher}(x))$
$= \textbf{if } \text{listp}(x) \textbf{ then } \text{car}(\text{flatten}(x))$
$\quad \textbf{else } \mathtt{0} \textbf{ endif}$

THEOREM: gopher-returns-correct-state
$\text{flatten}(\text{cdr}(\text{gopher}(x)))$
$= \textbf{if } \text{listp}(x) \textbf{ then } \text{cdr}(\text{flatten}(x))$
$\quad \textbf{else } \text{cons}(\mathtt{0}, \textbf{nil}) \textbf{ endif}$

THEOREM: correctness-of-samefringe
$\text{samefringe}(x, y) = (\text{flatten}(x) = \text{flatten}(y))$

DEFINITION:
prime1 (x, y)
$= \textbf{if } y \simeq \mathtt{0} \textbf{ then } \textbf{f}$
$\quad \textbf{elseif } y = \mathtt{1} \textbf{ then } \textbf{t}$
$\quad \textbf{else } (\neg \text{divides}(y, x)) \land \text{prime1}(x, y - 1) \textbf{ endif}$

DEFINITION:
$\text{prime}(x) = ((x \not\simeq \mathtt{0}) \land (x \neq \mathtt{1}) \land \text{prime1}(x, x - 1))$

DEFINITION:
greatest-factor $(x,\ y)$
$=$ **if** $(y \simeq 0) \vee (y = 1)$ **then** x
elseif divides $(y,\ x)$ **then** y
else greatest-factor $(x,\ y - 1)$ **endif**

THEOREM: greatest-factor-lessp
$((y < x)$
$\wedge$ $(\neg$ prime1 $(x,\ y))$
$\wedge$ $(x \not\simeq 0)$
$\wedge$ $((x - 1) \neq 0)$
$\wedge$ $(y \not\simeq 0))$
$\rightarrow$ (greatest-factor $(x,\ y) < x)$

THEOREM: greatest-factor-divides
$((y < x) \wedge (\neg$ prime1 $(x,\ y)) \wedge (x \not\simeq 0) \wedge (x \neq 1) \wedge (y \not\simeq 0))$
$\rightarrow$ $((x$ **mod** greatest-factor $(x,\ y)) = 0)$

THEOREM: greatest-factor-0
(greatest-factor $(x,\ y) = 0) = (((y \simeq 0) \vee (y = 1)) \wedge (x = 0))$

THEOREM: remainder-0-crock
$(0$ **mod** $y) = 0$

THEOREM: greatest-factor-1
(greatest-factor $(x,\ y) = 1) = (x = 1)$

THEOREM: numberp-greatest-factor
(greatest-factor $(x,\ y) \in \mathbf{N}) = (\neg\ (((y \simeq 0) \vee (y = 1)) \wedge (x \notin \mathbf{N})))$

DEFINITION:
prime-factors (x)
$=$ **if** $(x \simeq 0) \vee ((x - 1) = 0)$ **then nil**
elseif prime1 $(x,\ x - 1)$ **then** cons $(x,$ **nil**$)$
else append (prime-factors (greatest-factor $(x,\ x - 1)$),
prime-factors $(x \div$ greatest-factor $(x,\ x - 1)))$ **endif**

DEFINITION:
prime-list (l)
$=$ **if** $l \simeq$ **nil** **then t**
else prime (car (l)) $\wedge$ prime-list (cdr (l)) **endif**

DEFINITION:
times-list (l)
$=$ **if** $l \simeq$ **nil** **then** 1
else car $(l) *$ times-list (cdr (l)) **endif**

THEOREM: times-list-append
times-list (append $(x,\ y)$) $=$ (times-list $(x) *$ times-list (y))

THEOREM: prime-list-append
prime-list (append $(x,\ y)$) $=$ (prime-list $(x) \wedge$ prime-list (y))

THEOREM: prime-list-prime-factors
prime-list (prime-factors (x))

THEOREM: quotient-times1
$((y \in \mathbf{N}) \wedge (x \in \mathbf{N}) \wedge (x \neq \mathsf{0}) \wedge$ divides $(x,\ y))$
$\rightarrow \quad ((x * (y \div x)) = y)$

THEOREM: quotient-lessp
$((x \not\simeq \mathsf{0}) \wedge (x < y)) \rightarrow ((y \div x) \neq \mathsf{0})$

THEOREM: enough-factors
$(x \not\simeq \mathsf{0}) \rightarrow$ (times-list (prime-factors (x)) $= x$)

THEOREM: prime-factorization-existence
$(x \not\simeq \mathsf{0})$
$\rightarrow$ ((times-list (prime-factors (x)) $= x$)
$\wedge$ prime-list (prime-factors (x)))

DEFINITION:
greatereqpr $(w,\ z)$
$=$ **if** $w \simeq \mathsf{0}$ **then** $z \simeq \mathsf{0}$
elseif $w = z$ **then t**
else greatereqpr $(w - 1,\ z)$ **endif**

THEOREM: times-id-iff-1
$(z = (w * z)) = ((z \in \mathbf{N}) \wedge ((z = \mathsf{0}) \vee (w = \mathsf{1})))$

THEOREM: prime1-basic
$((z \neq \mathsf{1}) \wedge (z \neq \mathsf{0}) \wedge (z \in \mathbf{N}) \wedge$ greatereqpr $(u,\ z))$
$\rightarrow \quad (\neg$ prime1 $(w * z,\ u))$

THEOREM: greatereqpr-lessp
greatereqpr $(x,\ y) = (x \not< y)$

THEOREM: greatereqpr-remainder
$((z \neq (1 + v)) \wedge$ divides $(z,\ 1 + v)) \rightarrow$ greatereqpr $(v,\ z)$

THEOREM: prime-basic
$((z \neq \mathsf{1}) \wedge (z \neq x) \wedge (x \not\simeq \mathsf{0}) \wedge (x \neq \mathsf{1}) \wedge$ divides $(z,\ x))$
$\rightarrow \quad (\neg$ prime1 $(x,\ x - 1))$

Theorem: remainder-gcd
$(\mathrm{gcd}(b,\ x) = y) \rightarrow ((b \ \mathbf{mod}\ y) = \mathsf{0})$

Theorem: remainder-gcd-1
$((b \ \mathbf{mod}\ x) \neq \mathsf{0}) \rightarrow (\mathrm{gcd}(b,\ x) \neq x)$

Theorem: divides-times1
$(a = (z * y)) \rightarrow ((a \ \mathbf{mod}\ z) = \mathsf{0})$

Theorem: times-identity1
$((y \in \mathbf{N}) \wedge (y \neq \mathsf{1}) \wedge (y \neq \mathsf{0}) \wedge (x \neq \mathsf{0})) \rightarrow (x \neq (x * y))$

Theorem: times-identity
$(x = (x * y)) = ((x = \mathsf{0}) \vee ((x \in \mathbf{N}) \wedge (y = \mathsf{1})))$

Theorem: kludge-bridge
$(y = (k * x)) \rightarrow (\mathrm{gcd}(y,\ a * x) = (x * \mathrm{gcd}(a,\ k)))$

Theorem: hack1
$((\neg\ \mathrm{divides}(x,\ a)) \wedge (a = \mathrm{gcd}(x * a,\ b * a)))$
$\rightarrow \quad ((k * x) \neq (b * a))$

Theorem: prime-gcd
$((\neg\ \mathrm{divides}(x,\ b)) \wedge (x \not\simeq \mathsf{0}) \wedge ((x - 1) \neq \mathsf{0}) \wedge \mathrm{prime1}(x,\ x - 1))$
$\rightarrow \quad ((\mathrm{gcd}(b,\ x) = \mathsf{1}) = \mathbf{t})$

Theorem: gcd-distributes-over-an-opened-up-times
$((x \in \mathbf{N}) \wedge (x \neq \mathsf{0}) \wedge (\mathit{free} = (x * z)))$
$\rightarrow \quad (\mathrm{gcd}(b * z,\ \mathit{free}) = (z * \mathrm{gcd}(b,\ x)))$

Theorem: prime-key
$((z \in \mathbf{N}) \wedge \mathrm{prime}(x) \wedge (\neg\ \mathrm{divides}(x,\ z)) \wedge (\neg\ \mathrm{divides}(x,\ b)))$
$\rightarrow \quad ((x * k) \neq (b * z))$

Theorem: quotient-divides
$((y \in \mathbf{N}) \wedge ((x * (y \div x)) \neq y)) \rightarrow ((y \ \mathbf{mod}\ x) \neq \mathsf{0})$

Theorem: little-step
$(\mathrm{prime}(x) \wedge (y \neq \mathsf{1}) \wedge (x \neq y)) \rightarrow ((x \ \mathbf{mod}\ y) \neq \mathsf{0})$

Theorem: lessp-count-delete
$(n \in l) \rightarrow (\mathrm{count}(\mathrm{delete}(n,\ l)) < \mathrm{count}(l))$

Definition:
$\mathrm{perm}(a,\ b)$
$= \quad \mathbf{if}\ a \simeq \mathbf{nil}\ \ \mathbf{then}\ b \simeq \mathbf{nil}$
$\quad\ \ \mathbf{elseif}\ \mathrm{car}(a) \in b\ \ \mathbf{then}\ \mathrm{perm}(\mathrm{cdr}(a),\ \mathrm{delete}(\mathrm{car}(a),\ b))$
$\quad\ \ \mathbf{else}\ \mathbf{f}\ \mathbf{endif}$

THEOREM: remainder-times
$((y * x) \textbf{ mod } y) = 0$

THEOREM: prime-list-delete
$\text{prime-list}(l2) \rightarrow \text{prime-list}(\text{delete}(x, l2))$

THEOREM: divides-times-list
$((c \not\simeq 0) \wedge (c \in l)) \rightarrow ((\text{times-list}(l) \textbf{ mod } c) = 0)$

THEOREM: quotient-times
$((y * x) \div y)$
$= \textbf{if } y \simeq 0 \textbf{ then } 0$
$\textbf{else } \text{fix}(x) \textbf{ endif}$

THEOREM: distributivity-of-divides
$((a \not\simeq 0) \wedge \text{divides}(a, w)) \rightarrow ((c * (w \div a)) = ((c * w) \div a))$

THEOREM: times-list-delete
$((c \not\simeq 0) \wedge (c \in l))$
$\rightarrow (\text{times-list}(\text{delete}(c, l)) = (\text{times-list}(l) \div c))$

THEOREM: prime-list-times-list
$(\text{prime}(c) \wedge \text{prime-list}(l2) \wedge (c \notin l2))$
$\rightarrow ((\text{times-list}(l2) \textbf{ mod } c) \neq 0)$

THEOREM: if-times-then-divides
$((c \not\simeq 0) \wedge (\neg\, \text{divides}(c, x))) \rightarrow ((c * y) \neq x)$

THEOREM: times-equal-1
$((a * b) = 1)$
$= ((a \neq 0)$
$\wedge (b \neq 0)$
$\wedge (a \in \mathbf{N})$
$\wedge (b \in \mathbf{N})$
$\wedge ((a - 1) = 0)$
$\wedge ((b - 1) = 0))$

THEOREM: prime-member
$(((c * \text{times-list}(l1)) = \text{times-list}(l2)) \wedge \text{prime}(c) \wedge \text{prime-list}(l2))$
$\rightarrow (c \in l2)$

THEOREM: divides-implies-times
$((a \not\simeq 0) \wedge (c \in \mathbf{N}) \wedge ((a * c) = b)) \rightarrow ((c = (b \div a)) = \mathbf{t})$

THEOREM: prime-factorization-uniqueness
$(\text{prime-list}(l1) \wedge \text{prime-list}(l2) \wedge (\text{times-list}(l1) = \text{times-list}(l2)))$
$\rightarrow \text{perm}(l1, l2)$

DEFINITION:
maximum (l)
$=$ **if** $l \simeq$ **nil** **then** 0
elseif car $(l) <$ maximum (cdr (l)) **then** maximum (cdr (l))
else car (l) **endif**

THEOREM: member-maximum
listp $(x) \rightarrow$ (maximum $(x) \in x$)

THEOREM: lessp-delete-rewrite
(count (delete (x, l)) $<$ count (l)) $=$ $(x \in l)$

DEFINITION:
ordered2 (l)
$=$ **if** listp (l)
then if listp (cdr (l))
then if car $(l) <$ cadr (l) **then f**
else ordered2 (cdr (l)) **endif**
else t endif
else t endif

DEFINITION:
dsort (l)
$=$ **if** $l \simeq$ **nil** **then nil**
else cons (maximum (l), dsort (delete (maximum (l), l))) **endif**

DEFINITION:
addtolist2 (x, l)
$=$ **if** listp (l)
then if $x <$ car (l) **then** cons (car (l), addtolist2 $(x$, cdr (l)))
else cons (x, l) **endif**
else cons $(x$, **nil**) **endif**

DEFINITION:
sort2 (l)
$=$ **if** $l \simeq$ **nil** **then nil**
else addtolist2 (car (l), sort2 (cdr (l))) **endif**

THEOREM: sort2-gen-1
plistp (sort2 (x))

THEOREM: sort2-gen-2
ordered2 (sort2 (x))

THEOREM: sort2-gen
plistp (sort2 (x)) $\wedge$ ordered2 (sort2 (x))

THEOREM: addtolist2-delete
$(\text{plistp}(y) \land \text{ordered2}(y) \land (x \neq v))$
$\rightarrow \quad (\text{addtolist2}(v, \text{delete}(x, y)) = \text{delete}(x, \text{addtolist2}(v, y)))$

THEOREM: delete-addtolist2
$\text{plistp}(y) \rightarrow (\text{delete}(v, \text{addtolist2}(v, y)) = y)$

THEOREM: addtolist2-kludge
$((v \not< w) \land (\text{addtolist2}(v, y) = \text{cons}(v, y)))$
$\rightarrow \quad (\text{addtolist2}(v, \text{addtolist2}(w, y)) = \text{cons}(v, \text{addtolist2}(w, y)))$

THEOREM: lessp-maximum-addtolist2
$(v \not< \text{maximum}(z)) \rightarrow (\text{addtolist2}(v, \text{sort2}(z)) = \text{cons}(v, \text{sort2}(z)))$

THEOREM: sort2-delete-cons
$\text{listp}(x) \rightarrow (\text{cons}(\text{maximum}(x), \text{delete}(\text{maximum}(x), \text{sort2}(x))) = \text{sort2}(x))$

THEOREM: sort2-delete
$\text{sort2}(\text{delete}(x, l)) = \text{delete}(x, \text{sort2}(l))$

THEOREM: dsort-sort2
$\text{dsort}(x) = \text{sort2}(x)$

THEOREM: count-list-sort2
$\text{count-list}(a, \text{sort2}(l)) = \text{count-list}(a, l)$

```
;   The next segment of this XXX illustrates three different program
;   verification methods:  the functional approach, the inductive assertion
;   approach, and the interpreter approach.  The program considered is a simple
;   loop for summing the numbers from I down to 0
```

DEFINITION:
$\text{sigma}(m, n)$
$= \quad$ **if** $m < n$ **then** $n + \text{sigma}(m, n - 1)$
else 0 **endif**

THEOREM: difference-1
$(x - \mathtt{1}) = (x - 1)$

DEFINITION:
$\text{prog-trans-of-sigma}(i, ac)$
$= \quad$ **if** $i \simeq \mathtt{0}$ **then** ac
else $\text{prog-trans-of-sigma}(i - \mathtt{1}, ac + i)$ **endif**

THEOREM: functional-loop-invrt
$(ac \in \mathbf{N}) \rightarrow (\text{prog-trans-of-sigma}(i, ac) = (ac + \text{sigma}(\mathtt{0}, i)))$

THEOREM: correctness-of-functional-sigma
$\text{prog-trans-of-sigma}(i,\ 0) = \text{sigma}(0,\ i)$

THEOREM: sigma-input-path
$(0 = \text{sigma}(k,\ k)) \wedge (k \leq k)$

THEOREM: sigma-loop-invrt
$((i \not\simeq 0) \wedge (i \leq k))$
$\rightarrow \quad (((\text{sigma}(i,\ k) + i) = \text{sigma}(i - 1,\ k)) \wedge ((i - 1) \leq k))$

THEOREM: sigma-output-path
$((i \simeq 0) \wedge (i \leq k)) \rightarrow (\text{sigma}(i,\ k) = \text{sigma}(0,\ k))$

```
;   The interpreter we consider fetches instructions out of the same memory
;   being modified by the execution.  Earlier we proved a simpler version in
;   which the program was in read-only memory.  The new approach is almost
;   identical but we have to force the opening up of certain functions because
;   instead of doing CDR recursion the interpreter EXECUTE1 has to count the PC
;   up and the theorem prover doesn't handle counting up very well yet.
```

DEFINITION:
set (*addr*, *val*, *mem*)
= **if** $addr \simeq 0$ **then** cons (*val*, cdr (*mem*))
else cons (car (*mem*), set (*addr* − 1, *val*, cdr (*mem*))) **endif**

DEFINITION:
get (*addr*, *mem*)
= **if** $addr \simeq 0$ **then** car (*mem*)
else get (*addr* − 1, cdr (*mem*)) **endif**

THEOREM: get-set
get (j, set (i, *val*, *mem*))
= **if** eqp (j, i) **then** *val*
else get (j, *mem*) **endif**

DEFINITION:
execute1 (*pc*, *mem*, *max*)
= **if** $pc \not< max$ **then** list (**f**, *mem*)
elseif get (*pc*, *mem*) = `'(stop)` **then** list (**f**, *mem*)
elseif car (get (*pc*, *mem*)) = `'jumpa` **then** list (cadr (get (*pc*, *mem*)), *mem*)
elseif car (get (*pc*, *mem*)) = `'skipne`
then if get (cadr (get (*pc*, *mem*)), *mem*) $\simeq 0$ **then** execute1 (1 + *pc*, *mem*, *max*)
else execute1 (1 + (1 + *pc*), *mem*, *max*) **endif**
elseif car (get (*pc*, *mem*)) = `'subi`

```
    then execute1 (1 + pc,
                   set (cadr (get (pc, mem)),
                        get (cadr (get (pc, mem)), mem) − caddr (get (pc, mem)),
                        mem),
                   max)
    elseif car (get (pc, mem)) = 'addi
    then execute1 (1 + pc,
                   set (cadr (get (pc, mem)),
                        caddr (get (pc, mem)) + get (cadr (get (pc, mem)), mem),
                        mem),
                   max)
    elseif car (get (pc, mem)) = 'add
    then execute1 (1 + pc,
                   set (cadr (get (pc, mem)),
                        get (caddr (get (pc, mem)), mem)
                        +    get (cadr (get (pc, mem)), mem),
                        mem),
                   max)
    elseif car (get (pc, mem)) = 'movei
    then execute1 (1 + pc,
                   set (cadr (get (pc, mem)), caddr (get (pc, mem)), mem),
                   max)
    else list (f, mem) endif
```

DEFINITION:

```
execute (pc, mem, clk)
=   if clk ≃ 0 then mem
    elseif pc ∈ N
    then execute (car (execute1 (pc, mem, length (mem))),
                  cadr (execute1 (pc, mem, length (mem))),
                  clk − 1)
    else mem endif
```

DEFINITION:

```
get-simplifier (x)
=   if listp (x)
       ∧   (car (x) = 'get)
       ∧   listp (cadr (x))
       ∧   (car (cadr (x)) = 'quote)
    then if cadr (cadr (x)) ≃ 0 then list ('car, caddr (x))
         else list ('get,
                    list ('quote, cadr (cadr (x)) − 1),
                    list ('cdr, caddr (x))) endif
    else x endif
```

THEOREM: correctness-of-get-simplifier
eval$ (**t**, x, a) = eval$ (**t**, get-simplifier (x), a)

DEFINITION:
set-simplifier (x)
= **if** listp (x)
  ∧ (car (x) = `'set`)
  ∧ listp (cadr (x))
  ∧ (car (cadr (x)) = `'quote`)
 then if cadr (cadr (x)) ≃ `0`
  then list (`'cons`, caddr (x), list (`'cdr`, cadddr (x)))
  else list (`'cons`,
    list (`'car`, cadddr (x)),
    list (`'set`,
     list (`'quote`, cadr (cadr (x)) − 1),
     caddr (x),
     list (`'cdr`, cadddr (x)))) **endif**
 else x **endif**

THEOREM: correctness-of-set-simplifier
eval$ (**t**, x, a) = eval$ (**t**, set-simplifier (x), a)

THEOREM: length-5
(caddddr (x) = `'(jumpa 1)`)
→ (length (x) = (`5` + length (cdddddr (x))))

THEOREM: length-cons6
length (cons ($x1$, cons ($x2$, cons ($x3$, cons ($x4$, cons ($x5$, cons ($x6$, $x7$)))))))
= (`6` + length ($x7$))

THEOREM: execute1-1
($max \not< 6$)
→ (execute1 (`1`,
   cons (`'(movei 7 0)`,
    cons (`'(skipne 6)`,
     cons (`'(stop)`,
      cons (`'(add 7 6)`,
       cons (`'(subi 6 1)`,
        cons (`'(jumpa 1)`, l)))))),
   max)
 = **if** car (l) ≃ `0`
  then execute1 (`2`,
    cons (`'(movei 7 0)`,
     cons (`'(skipne 6)`,
      cons (`'(stop)`,

```
                                      cons ('(add 7 6),
                                        cons ('(subi 6 1),
                                          cons ('(jumpa 1), l))))))),
                        max)
          else execute1 (3,
                        cons ('(movei 7 0),
                          cons ('(skipne 6),
                            cons ('(stop),
                              cons ('(add 7 6),
                                cons ('(subi 6 1),
                                  cons ('(jumpa 1),
                                    l))))))),
                        max) endif)
```

THEOREM: execute1-3

```
 (max ≮ 6)
 →   (execute1 (3,
                cons ('(movei 7 0),
                  cons ('(skipne 6),
                    cons ('(stop),
                      cons ('(add 7 6),
                        cons ('(subi 6 1),
                          cons ('(jumpa 1), l)))))),
                max)
      =   execute1 (4,
                cons ('(movei 7 0),
                  cons ('(skipne 6),
                    cons ('(stop),
                      cons ('(add 7 6),
                        cons ('(subi 6 1),
                          cons ('(jumpa 1),
                            cons (car (l),
                              cons (car (l)
                                    +   cadr (l),
                                    cddr (l))))))))),
                max))
```

THEOREM: execute1-4

```
 (max ≮ 6)
 →   (execute1 (4,
                cons ('(movei 7 0),
                  cons ('(skipne 6),
                    cons ('(stop),
                      cons ('(add 7 6),
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad \mathrm{cons}\,(\texttt{'(subi 6 1)}, \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathrm{cons}\,(\texttt{'(jumpa 1)}, l)))))), \\
\qquad\quad max) \\
= \quad \mathrm{execute1}\,(\texttt{5}, \\
\qquad\qquad \mathrm{cons}\,(\texttt{'(movei 7 0)}, \\
\qquad\qquad\quad \mathrm{cons}\,(\texttt{'(skipne 6)}, \\
\qquad\qquad\qquad \mathrm{cons}\,(\texttt{'(stop)}, \\
\qquad\qquad\qquad\quad \mathrm{cons}\,(\texttt{'(add 7 6)}, \\
\qquad\qquad\qquad\qquad \mathrm{cons}\,(\texttt{'(subi 6 1)}, \\
\qquad\qquad\qquad\qquad\quad \mathrm{cons}\,(\texttt{'(jumpa 1)}, \\
\qquad\qquad\qquad\qquad\qquad \mathrm{cons}\,(\mathrm{car}\,(l) - \texttt{1}, \\
\qquad\qquad\qquad\qquad\qquad\quad \mathrm{cdr}\,(l)))))))), \\
\qquad\qquad max))
\end{array}
$$

```
(PROVE-LEMMA
  EXECUTE1-OPENED-UP
  (REWRITE)
  (IMPLIES
    (AND (NOT (LESSP MAX 6))
         (EQUAL (CAR MEM)
                (QUOTE (MOVEI 7 0)))
         (EQUAL (CADR MEM)
                (QUOTE (SKIPNE 6)))
         (EQUAL (CADDR MEM)
                (QUOTE (STOP)))
         (EQUAL (CADDDR MEM)
                (QUOTE (ADD 7 6)))
         (EQUAL (CADDDDR MEM)
                (QUOTE (SUBI 6 1)))
         (EQUAL (CADDDDDR MEM)
                (QUOTE (JUMPA 1))))
    (EQUAL
      (EXECUTE1 1 MEM MAX)
      (IF
        (ZEROP (CADDDDDDR MEM))
        (LIST
          F
          (CONS
            (QUOTE (MOVEI 7 0))
            (CONS
              (QUOTE (SKIPNE 6))
              (CONS (QUOTE (STOP))
                    (CONS (QUOTE (ADD 7 6))
```

```
                        (CONS (QUOTE (SUBI 6 1))
                              (CONS (QUOTE (JUMPA 1))
                                    (CDDDDDDR MEM)))))))
  (LIST
    1
    (CONS
      (QUOTE (MOVEI 7 0))
      (CONS
        (QUOTE (SKIPNE 6))
        (CONS
          (QUOTE (STOP))
          (CONS
            (QUOTE (ADD 7 6))
            (CONS
              (QUOTE (SUBI 6 1))
              (CONS
                (QUOTE (JUMPA 1))
                (CONS (SUB1 (CADDDDDDR MEM))
                      (CONS (PLUS (CADDDDDDR MEM)
                                  (CADDDDDDDR MEM))
                            (CDDDDDDDDR MEM))))))))))))
```

THEOREM: execute-opened-up

$((pc \in \mathbf{N}) \wedge (clk \not\simeq 0))$
$\rightarrow \quad (\text{execute}\,(pc,\ mem,\ clk)$
$\quad = \quad \text{execute}\,(\text{car}\,(\text{execute1}\,(pc,\ mem,\ \text{length}\,(mem))),$
$\qquad\qquad \text{cadr}\,(\text{execute1}\,(pc,\ mem,\ \text{length}\,(mem))),$
$\qquad\qquad clk - 1))$

THEOREM: interpreter-loop-invrt

$((clk \not< \text{caddddddr}\,(mem))$
$\wedge \quad (\text{car}\,(mem) = \texttt{'(movei 7 0)})$
$\wedge \quad (\text{cadr}\,(mem) = \texttt{'(skipne 6)})$
$\wedge \quad (\text{caddr}\,(mem) = \texttt{'(stop)})$
$\wedge \quad (\text{cadddr}\,(mem) = \texttt{'(add 7 6)})$
$\wedge \quad (\text{caddddr}\,(mem) = \texttt{'(subi 6 1)})$
$\wedge \quad (\text{cadddddr}\,(mem) = \texttt{'(jumpa 1)}))$
$\rightarrow \quad (\text{cadddddddr}\,(\text{execute}\,(1,\ mem,\ clk))$
$\quad = \quad \textbf{if}\ \text{caddddddr}\,(mem) \simeq 0\ \textbf{then}\ \text{cadddddddr}\,(mem)$
$\qquad \textbf{else}\ \text{cadddddddr}\,(mem) + \text{sigma}\,(0,\ \text{caddddddr}\,(mem))\ \textbf{endif})$

THEOREM: interpreter-input-path

execute (0,
    cons (`'(movei 7 0)`,
        cons (`'(skipne 6)`,

```
                    cons ('(stop),
                         cons ('(add 7 6),
                              cons ('(subi 6 1), cons ('(jumpa 1), mem)))))),
          clk)
=   if clk ≃ 0
    then cons ('(movei 7 0),
              cons ('(skipne 6),
                   cons ('(stop),
                        cons ('(add 7 6),
                             cons ('(subi 6 1), cons ('(jumpa 1), mem))))))
    else execute (1,
                 cons ('(movei 7 0),
                      cons ('(skipne 6),
                           cons ('(stop),
                                cons ('(add 7 6),
                                     cons ('(subi 6 1),
                                          cons ('(jumpa 1),
                                               cons (car (mem),
                                                    cons (0,
                                                         cddr (mem))))))))),
                 clk) endif
```

THEOREM: correctness-of-interpreted-sigma

```
((mem = append ('((movei 7 0)
                  (skipne 6)
                  (stop)
                  (add 7 6)
                  (subi 6 1)
                  (jumpa 1)),
                tl))
 ∧   (i = get (6, mem))
 ∧   (clk ≮ i))
→   (get (7, execute (0, mem, clk))
      =   if clk ≃ 0 then get (7, mem)
          else sigma (0, i) endif)
```

THEOREM: difference-2

$((1 + (1 + x)) - 2) = \mathrm{fix}\,(x)$

THEOREM: half-plus

$((x + (x + y)) \div 2) = (x + (y \div 2))$

THEOREM: sigma-is-half-product

$\mathrm{sigma}\,(0,\ i) = ((i * (1 + i)) \div 2)$

Conservative Axiom: h-commutivity2
$\mathrm{h}(x, \mathrm{h}(y, z)) = \mathrm{h}(y, \mathrm{h}(x, z))$

Simultaneously, we introduce the new function symbol h.

Definition:
$\text{h-pr}(l, ac)$
$= \ \ \textbf{if } l \simeq \textbf{nil} \textbf{ then } ac$
$\quad \textbf{else } \mathrm{h}(\mathrm{car}(l), \text{h-pr}(\mathrm{cdr}(l), ac)) \textbf{ endif}$

Definition:
$\text{h-ac}(l, ac)$
$= \ \ \textbf{if } l \simeq \textbf{nil} \textbf{ then } ac$
$\quad \textbf{else } \text{h-ac}(\mathrm{cdr}(l), \mathrm{h}(\mathrm{car}(l), ac)) \textbf{ endif}$

Theorem: h-lemma
$\text{h-pr}(x, \mathrm{h}(z, a)) = \mathrm{h}(z, \text{h-pr}(x, a))$

Theorem: h-eq
$\text{h-ac}(l, ac) = \text{h-pr}(l, ac)$

Definition:
$\mathrm{f0}(x)$
$= \ \ \textbf{if } 100 < x \textbf{ then } x - 10$
$\quad \textbf{else } 91 \textbf{ endif}$

```
#|
(COMMENT
 (DEFN F91 (X)
       (EVAL$ T '(IF (LESSP 100 X)
                    (DIFFERENCE X 10)
                    (F91 (F91 (PLUS X 11))))
              (LIST (CONS 'X X))))

 (REFLECT F91 F0-SATISFIES-F91-EQUATION
          ((LESSP (DIFFERENCE 101 X))))

 (DEFN FOO (A) (IF (LESSP 100 (CDR (ASSOC 'X A)))
                  T
                  (FOO A)))

 (DEFN F91-INDUCTION-HINT (X A)
       (IF (LESSP 100 (CDR (ASSOC 'X A)))
         T
         (AND (F91-INDUCTION-HINT '(PLUS X 11) A)
```

```
              (F91-INDUCTION-HINT '(F0 (PLUS X 11)) A)))
       ((LESSP (DIFFERENCE 101 (EVAL$ T X A)))))

 (PROVE-LEMMA F91-TERMINATES ()
              (IMPLIES (V&C$ T X A)
                       (AND  (V&C$ T (LIST 'F91 X) A)
                             (EQUAL (CAR (V&C$ T (LIST 'F91 X) A))
                                    (F0 (CAR (V&C$ T X A))))))
              ((INDUCT (F91-INDUCTION-HINT X A))))
 (PROVE-LEMMA F91-IS-F0 ()
              (EQUAL (F91 X) (F0 X))))

|#
```

DEFINITION: $\text{even}\,(x) = (0 = (x \textbf{ mod } 2))$

DEFINITION: $\text{square}\,(x) = (x * x)$

THEOREM: times-1
$(1 * x) = \text{fix}\,(x)$

THEOREM: times-2
$(2 * x) = (x + x)$

THEOREM: exp-of-0
$\exp\,(0,\, k)$
$= \quad \textbf{if } k \simeq 0 \textbf{ then } 1$
$\quad\;\; \textbf{else } 0 \textbf{ endif}$

THEOREM: exp-of-1
$\exp\,(1,\, k) = 1$

THEOREM: exp-by-0
$\exp\,(x,\, 0) = 1$

THEOREM: exp-times
$\exp\,(i * j,\, k) = (\exp\,(i,\, k) * \exp\,(j,\, k))$

THEOREM: exp-exp
$\exp\,(\exp\,(i,\, j),\, k) = \exp\,(i,\, j * k)$

THEOREM: remainder-plus-times-1
$((x + (i * j)) \textbf{ mod } j) = (x \textbf{ mod } j)$

THEOREM: remainder-plus-times-2
$((x + (j * i)) \textbf{ mod } j) = (x \textbf{ mod } j)$

THEOREM: remainder-times-1
$((b * (a * c)) \textbf{ mod } a) = 0$

THEOREM: remainder-of-1
$(1 \textbf{ mod } x)$
$= \ \textbf{if } x = 1 \textbf{ then } 0$
$\quad \textbf{else } 1 \textbf{ endif}$

THEOREM: equal-length-0
$(\mathrm{length}(x) = 0) = (x \simeq \textbf{nil})$

THEOREM: length-delete
$\mathrm{length}(\mathrm{delete}(x,\ l))$
$= \ \textbf{if } x \in l \textbf{ then } \mathrm{length}(\mathrm{cdr}(l))$
$\quad \textbf{else } \mathrm{length}(l) \textbf{ endif}$

THEOREM: remainder-difference-times
$(((p * x) - (p * y)) \textbf{ mod } p) = 0$

THEOREM: prime-key-rewrite
$\mathrm{prime}(p)$
$\rightarrow \ ((((a * b) \textbf{ mod } p) = 0)$
$\quad = \ (((a \textbf{ mod } p) = 0) \vee ((b \textbf{ mod } p) = 0)))$

THEOREM: times-times-list-delete
$(x \in l) \rightarrow ((x * \mathrm{times\text{-}list}(\mathrm{delete}(x,\ l))) = \mathrm{times\text{-}list}(l))$

THEOREM: lessp-remainder-divisor
$(y \not\simeq 0) \rightarrow ((x \textbf{ mod } y) < y)$

EVENT: Introduce the function symbol *apply2* of 3 arguments.

DEFINITION:
$\mathrm{eval2}(\mathit{form},\ \mathit{envrn})$
$= \ \textbf{if } \mathit{form} \in \mathbf{N} \textbf{ then } \mathit{form}$
$\quad \textbf{elseif } \mathrm{litatom}(\mathit{form}) \textbf{ then } \mathrm{cdr}(\mathrm{assoc}(\mathit{form},\ \mathit{envrn}))$
$\quad \textbf{elseif } \mathrm{listp}(\mathit{form})$
$\quad \textbf{then } \mathrm{apply2}(\mathrm{car}(\mathit{form}),$
$\qquad\qquad \mathrm{eval2}(\mathrm{cadr}(\mathit{form}),\ \mathit{envrn}),$
$\qquad\qquad \mathrm{eval2}(\mathrm{caddr}(\mathit{form}),\ \mathit{envrn}))$
$\quad \textbf{else } \mathit{form} \textbf{ endif}$

DEFINITION:
$\mathrm{subst2}(\mathit{new},\ \mathit{old},\ \mathit{term})$
$= \ \textbf{if } \mathit{term} \in \mathbf{N} \textbf{ then } \mathit{term}$

elseif litatom ($term$)
then if $old = term$ **then** new
  else $term$ **endif**
elseif listp ($term$)
then list (car ($term$),
  subst2 (new, old, cadr ($term$)),
  subst2 (new, old, caddr ($term$)))
else $term$ **endif**

THEOREM: subst2-ok
eval2 (subst2 (new, old, $term$), a)
= eval2 ($term$, cons (cons (old, eval2 (new, a)), a))

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

Index