```
#|

 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 $\text{append}(\text{from-to}(i, j-1), \text{list}(j))$ **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** $1 + (x + y)$

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
$((a + b) = 0) = ((a \simeq 0) \wedge (b \simeq 0))$

THEOREM: difference-x-x
$(x - x) = 0$

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
$(y \not< x) \rightarrow ((x - y) = 0)$

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

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

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

DEFINITION:
delete(x, y)
$=$ **if** listp(y)
then if $x = \text{car}(y)$ **then** cdr(y)
else cons$(\text{car}(y), \text{delete}(x, \text{cdr}(y)))$ **endif**
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 $(x) \wedge$ (car $(x) =$ `'plus`)
then append (plus-fringe (cadr (x)), plus-fringe (caddr (x)))
else cons $(x,$ **nil**$)$ **endif**

DEFINITION:
plus-tree (l)
$=$ **if** $l \simeq$ **nil then** `''0`
elseif cdr $(l) \simeq$ **nil then** list (`'fix`, car (l))
elseif cddr $(l) \simeq$ **nil then** list (`'plus`, car (l), cadr (l))
else list (`'plus`, car (l), plus-tree (cdr (l))) **endif**

DEFINITION:
cancel (x)
$=$ **if** listp $(x) \wedge$ (car $(x) =$ `'equal`)
then if listp (cadr (x))
$\wedge$ (caadr $(x) =$ `'plus`)
$\wedge$ listp (caddr (x))
$\wedge$ (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))
$\wedge$ (caadr $(x) =$ `'plus`)
$\wedge$ (caddr $(x) \in$ 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))
$\wedge$ (caaddr $(x) =$ `'plus`)
$\wedge$ (cadr $(x) \in$ plus-fringe (caddr (x)))
then list (`'if`,
list (`'numberp`, cadr (x)),
list (`'equal`,
`''0`,
plus-tree (delete (cadr (x), plus-fringe (caddr (x))))),

list (’quote, **f**))
else x **endif**
else x **endif**

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

THEOREM: numberp-eval$-plus
(listp (x) $\wedge$ (car (x) = ’plus)) $\rightarrow$ (eval$ (**t**, x, a) $\in$ **N**)

THEOREM: numberp-eval$-plus-tree
eval$ (**t**, plus-tree (l), a) $\in$ **N**

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

THEOREM: plus-tree-delete
eval$ (**t**, plus-tree (delete (x, y)), a)
= **if** $x \in y$ **then** eval$ (**t**, plus-tree (y), a) $-$ eval$ (**t**, x, a)
else eval$ (**t**, plus-tree (y), a) **endif**

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

THEOREM: plus-tree-bagdiff
subbagp (x, y)
$\rightarrow$ (eval$ (**t**, plus-tree (bagdiff (y, x)), a)
= (eval$ (**t**, plus-tree (y), a) $-$ eval$ (**t**, plus-tree (x), a)))

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

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

THEOREM: eval$-plus-tree-append
eval$ (**t**, plus-tree (append (x, y)), a)
= (eval$ (**t**, plus-tree (x), a) + eval$ (**t**, plus-tree (y), a))

THEOREM: plus-tree-plus-fringe
eval$ (**t**, plus-tree (plus-fringe (x)), a) = fix (eval$ (**t**, x, a))

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

THEOREM: cadr-eval$-list
(car (eval$ ('list, x, a)) = eval$ (**t**, car (x), a))
$\wedge$ (cdr (eval$ ('list, x, a))
= **if** listp (x) **then** eval$ ('list, cdr (x), a)
else 0 **endif**)

THEOREM: eval$-quote
eval$ (**t**, cons ('quote, *args*), a) = car (*args*)

THEOREM: listp-eval$
listp (eval$ ('list, x, a)) = listp (x)

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

DEFINITION:
reverse (x)
= **if** listp (x) **then** append (reverse (cdr (x)), cons (car (x), **nil**))
else nil endif

THEOREM: associativity-of-append
append (append (x, y), z) = append (x, append (y, z))

DEFINITION:
plistp (x)
= **if** $x \simeq$ **nil** **then** x = **nil**
else plistp (cdr (x)) **endif**

THEOREM: append-right-id
plistp (x) $\rightarrow$ (append (x, **nil**) = x)

THEOREM: plistp-reverse
plistp (reverse (x))

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

THEOREM: times-zero2
($y \notin \mathbf{N}$) $\rightarrow$ ((x * y) = 0)

THEOREM: distributivity-of-times-over-plus
(x * (y + z)) = ((x * y) + (x * z))

THEOREM: times-add1
(x * (1 + y))
= **if** $y \in \mathbf{N}$ **then** x + (x * y)
else fix (x) **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
listp (cddr (x)) $\rightarrow$ (count (cadr (x)) $<$ 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*) → expressionp (optimize (*x*))

THEOREM: correctness-of-optimize
expressionp (*x*) → (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
exec (append (*x*, *y*), *pds*, *envrn*) = exec (*y*, exec (*x*, *pds*, *envrn*), *envrn*)

THEOREM: correctness-of-codegen
expressionp (*x*)
→ (exec (reverse (codegen (*x*, *ins*)), *pds*, *envrn*)
= push (term-eval (*x*, *envrn*), exec (reverse (*ins*), *pds*, *envrn*)))

THEOREM: correctness-of-optimizing-compiler
expressionp (*x*)
→ (exec (compile (*x*), *pds*, *envrn*) = push (term-eval (*x*, *envrn*), *pds*))

THEOREM: transitivity-of-lessp
((*x* < *y*) ∧ (*y* < *z*)) → (*x* < *z*)

THEOREM: lessp-not-reflexive
x ≮ *x*

DEFINITION: eqp (*x*, *y*) = (fix (*x*) = fix (*y*))

THEOREM: trichotomy-of-lessp
((¬ eqp (*x*, *y*)) ∧ (*y* ≮ *x*)) → (*x* < *y*)

THEOREM: reverse-reverse
plistp (*x*) → (reverse (reverse (*x*)) = *x*)

DEFINITION:
flatten (*x*)
= **if** listp (*x*) **then** append (flatten (car (*x*)), flatten (cdr (*x*)))
else cons (*x*, **nil**) **endif**

DEFINITION:
mc-flatten (*x*, *y*)
= **if** listp (*x*) **then** mc-flatten (car (*x*), mc-flatten (cdr (*x*), *y*))
else cons (*x*, *y*) **endif**

THEOREM: flatten-mc-flatten
mc-flatten (*x*, *y*) = append (flatten (*x*), *y*)

THEOREM: member-append
(*x* ∈ append (*a*, *b*)) = ((*x* ∈ *a*) ∨ (*x* ∈ *b*))

THEOREM: member-reverse
(*x* ∈ reverse (*y*)) = (*x* ∈ *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
$((x \leq y) \wedge (y \leq z)) \rightarrow (x \leq z)$

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 $(x) = ((x =$ **t**$) \vee (x =$ **f**$))$

THEOREM: iff-equal-equal
(boolean $(p) \wedge$ boolean $(q)) \rightarrow ((p \leftrightarrow q) = (p = q))$

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

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

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

THEOREM: associativity-of-equal
(boolean $(a) \wedge$ (boolean $(b) \wedge$ boolean $(c)))$
$\rightarrow \quad (((a = b) = c) = (a = (b = c)))$

DEFINITION:
odd (x)
= **if** $x \simeq$ `0` **then f**
elseif $(x - 1) \simeq$ `0` **then t**
else odd $((x - 1) - 1)$ **endif**

DEFINITION:
even1 (x)
= **if** $x \simeq$ `0` **then t**
else odd $(x - 1)$ **endif**

DEFINITION:
even2 (x)
= **if** $x \simeq$ `0` **then t**
elseif $(x - 1) \simeq$ `0` **then f**
else even2 $((x - 1) - 1)$ **endif**

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

THEOREM: even1-double
even1 (double (i))

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

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

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

THEOREM: double-times-2
double $(i) = (\mathtt{2} * i)$

THEOREM: subsetp-cons
subsetp $(x, y) \rightarrow$ subsetp $(x,$ 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
last (append (*a*, *b*))
= **if** listp (*b*) **then** last (*b*)
elseif listp (*a*) **then** cons (car (last (*a*)), *b*)
else *b* **endif**

THEOREM: last-reverse
listp (*a*) → (last (reverse (*a*)) = cons (car (*a*), **nil**))

DEFINITION:
exp (*i*, *j*)
= **if** *j* ≃ `0` **then** `1`
else *i* ∗ exp (*i*, *j* − 1) **endif**

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

THEOREM: even1-even2
even1 (*x*) = even2 (*x*)

THEOREM: leq-nth
length (nth (*l*, *i*)) ≤ length (*l*)

THEOREM: member-sort
(*a* ∈ sort (*b*)) = (*a* ∈ *b*)

THEOREM: length-sort
length (sort (*a*)) = length (*a*)

DEFINITION:
count-list (*a*, *l*)
= **for** *x* **in** *l*
count *a* = *x* **endfor**

THEOREM: count-list-sort
count-list (*a*, sort (*l*)) = count-list (*a*, *l*)

THEOREM: ordered-append
ordered (append (*a*, *b*)) → ordered (*a*)

THEOREM: leq-half
half (*i*) ≤ *i*

DEFINITION:
number-listp (*l*)
= **if** listp (*l*) **then** (car (*l*) ∈ **N**) ∧ number-listp (cdr (*l*))
else *l* = **nil endif**

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

THEOREM: addtolist-of-ordered-number-list
(ordered (*x*) ∧ number-listp (*x*) ∧ (*i* ∈ **N**) ∧ (car (*x*) ≮ *i*))
→ (addtolist (*i*, *x*) = cons (*i*, *x*))

THEOREM: sort-of-ordered-number-list
(ordered (*x*) ∧ number-listp (*x*)) → (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*)) → ordered (*x*)

THEOREM: sort-ordered
number-listp (*l*) → ((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
(¬ occur (*a*, *b*)) → (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* ∈ **N**) ∧ litatom (*x*) ∧ (*pred* ≠ `'quote`))
→ (countps-loop (*l*, *pred*, *n*)
= (*n* + for (*x*,
l,
list (`'quote`, **t**),
`'count`,
list (*pred*, *x*),
nil)))

DEFINITION:
fact (*i*)
= **if** *i* ≃ `0` **then** `1`
else *i* ∗ fact (*i* − 1) **endif**

DEFINITION:
fact-loop (*i*, *ans*)
= **if** *i* ≃ `0` **then** *ans*
else fact-loop (*i* − 1, *i* ∗ *ans*) **endif**

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

THEOREM: fact-loop-fact
(*i* ∈ **N**) → (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) → (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) → ordered (addtolist $(x,\ y)$)

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

THEOREM: numberp-count
for $(x$,
l,
p,
`'count`,
b,
$a)\in$ **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 t** = z
else 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) + $(base * \text{power-eval}(\text{cdr}(l),\ base))$
else 0 **endif**

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

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

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

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

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

THEOREM: remainder-wrt-1
$(y$ **mod** 1$) = 0$

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

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

THEOREM: remainder-x-x
$(x$ **mod** $x) = 0$

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

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

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

THEOREM: lessp-quotient1
$((i \div j) < i) = ((i \not\simeq \mathtt{0}) \wedge ((j \simeq \mathtt{0}) \vee (j \neq \mathtt{1})))$

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

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

DEFINITION:
gcd $(x,\ y)$
$=$ **if** $x \simeq \mathtt{0}$ **then** fix (y)
elseif $y \simeq \mathtt{0}$ **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
$(\text{gcd}\,(x,\ y) = \mathtt{0}) = ((x \simeq \mathtt{0}) \wedge (y \simeq \mathtt{0}))$

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

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

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

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

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

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

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

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

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

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

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

THEOREM: remainder-add1
$((y \not\simeq \mathtt{0}) \wedge (y \neq \mathtt{1})) \rightarrow (((1 + (x * y))\ \mathbf{mod}\ y) \neq \mathtt{0})$

THEOREM: divides-plus-rewrite1
$(((x\ \mathbf{mod}\ z) = \mathtt{0}) \wedge ((y\ \mathbf{mod}\ z) = \mathtt{0})) \rightarrow (((x + y)\ \mathbf{mod}\ z) = \mathtt{0})$

THEOREM: divides-plus-rewrite2
$(((x\ \mathbf{mod}\ z) = \mathtt{0}) \wedge ((y\ \mathbf{mod}\ z) \neq \mathtt{0})) \rightarrow (((x + y)\ \mathbf{mod}\ z) \neq \mathtt{0})$

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

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

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

THEOREM: euclid
$((x\ \mathbf{mod}\ z) = \mathtt{0})$
$\rightarrow\ \ ((((y - x)\ \mathbf{mod}\ z) = \mathtt{0})$
$\quad = \ \ \mathbf{if}\ x < y\ \ \mathbf{then}\ (y\ \mathbf{mod}\ z) = \mathtt{0}$
$\quad\quad\ \ \mathbf{else\ t\ endif})$

THEOREM: lessp-times-cancellation
$((x * z) < (y * z)) = ((z \not\simeq 0) \wedge (x < y))$

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

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

THEOREM: gcd-divides-both
$((x \bmod \gcd(x,\ y)) = 0) \wedge ((y \bmod \gcd(x,\ y)) = 0)$

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

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*))

```
        *  (if-complexity (left-branch (x))
           +  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 (¬ if-exprp (test (x)))
         ∧  normalized-if-exprp (left-branch (x))
         ∧  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 ≃ nil then f
    elseif var = caar (alist) then t
    else assignedp (var, cdr (alist)) endif
```

DEFINITION: assume-true $(var,\ alist)$ = cons (cons $(var,\ \mathbf{t})$, $alist$)

DEFINITION: assume-false (*var*, *alist*) = cons (cons (*var*, **f**), *alist*)

DEFINITION:
tautologyp (*x*, *alist*)
= **if** if-exprp (*x*)
then if assignedp (test (*x*), *alist*)
then if assignment (test (*x*), *alist*)
then tautologyp (left-branch (*x*), *alist*)
else tautologyp (right-branch (*x*), *alist*) **endif**
else tautologyp (left-branch (*x*), assume-true (test (*x*), *alist*))
∧ tautologyp (right-branch (*x*),
assume-false (test (*x*), *alist*)) **endif**
else assignment (*x*, *alist*) **endif**

THEOREM: assignment-append
assignment (*x*, append (*a*, *b*))
= **if** assignedp (*x*, *a*) **then** assignment (*x*, *a*)
else assignment (*x*, *b*) **endif**

THEOREM: value-can-ignore-redundant-assignments
(((*val* ↔ assignment (*var*, *a*)) ∧ value (*x*, *a*))
→ value (*x*, cons (cons (*var*, *val*), *a*)))
∧ (((*val* ↔ assignment (*var*, *a*)) ∧ (¬ value (*x*, *a*)))
→ (¬ value (*x*, cons (cons (*var*, *val*), *a*))))

THEOREM: value-short-cut
(if-exprp (*x*) ∧ normalized-if-exprp (*x*))
→ (value (test (*x*), *a*) = assignment (test (*x*), *a*))

THEOREM: assignment-implies-assigned
assignment (*x*, *a*) → assignedp (*x*, *a*)

THEOREM: tautologyp-is-sound
(normalized-if-exprp (*x*) ∧ tautologyp (*x*, *a1*)) → value (*x*, append (*a1*, *a2*))

DEFINITION: tautology-checker (*x*) = tautologyp (normalize (*x*), **nil**)

DEFINITION:
falsify1 (*x*, *alist*)
= **if** if-exprp (*x*)
then if assignedp (test (*x*), *alist*)
then if assignment (test (*x*), *alist*)
then falsify1 (left-branch (*x*), *alist*)
else falsify1 (right-branch (*x*), *alist*) **endif**
elseif falsify1 (left-branch (*x*), assume-true (test (*x*), *alist*))

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*)
→ (assignment (*x*, falsify1 (*y*, *a*))
= **if** falsify1 (*y*, *a*) **then** assignment (*x*, *a*)
else *x* = **t** **endif**)

THEOREM: falsify1-falsifies
(normalized-if-exprp (*x*) ∧ falsify1 (*x*, *a*))
→ (value (*x*, falsify1 (*x*, *a*)) = **f**)

THEOREM: tautologyp-fails-means-falsify1-wins
(normalized-if-exprp (*x*) ∧ (¬ tautologyp (*x*, *a*)) ∧ *a*) → 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*)))
∧ falsify1 (*x*, *a*)
∧ normalized-if-exprp (*x*))
→ (value (*y*, falsify1 (*x*, *a*)) = **f**)

THEOREM: tautology-checker-is-complete
(¬ tautology-checker (*x*)) → (value (*x*, falsify (*x*)) = **f**)

THEOREM: tautology-checker-soundness-bridge
(tautologyp (*y*, *a1*)
∧ normalized-if-exprp (*y*)
∧ (value (*x*, *a2*) = value (*y*, append (*a1*, *a2*))))
→ value (*x*, *a2*)

THEOREM: tautology-checker-is-sound
tautology-checker (*x*) → value (*x*, *a*)

THEOREM: flatten-singleton
(flatten (x) = cons (y, **nil**)) = ((x ≃ **nil**) ∧ (x = y))

DEFINITION:
leftcount (x)
= **if** x ≃ **nil** **then** 0
else 1 + leftcount (car (x)) **endif**

DEFINITION:
gopher (x)
= **if** (x ≃ **nil**) ∨ (car (x) ≃ **nil**) **then** x
else gopher (cons (caar (x), cons (cdar (x), cdr (x)))) **endif**

THEOREM: gopher-preserves-count
count (x) ≮ count (gopher (x))

THEOREM: listp-gopher
listp (gopher (x)) = listp (x)

DEFINITION:
samefringe (x, y)
= **if** (x ≃ **nil**) ∨ (y ≃ **nil**) **then** x = y
else (car (gopher (x)) = car (gopher (y)))
∧ samefringe (cdr (gopher (x)), cdr (gopher (y))) **endif**

THEOREM: gopher-returns-leftmost-atom
car (gopher (x))
= **if** listp (x) **then** car (flatten (x))
else 0 **endif**

THEOREM: gopher-returns-correct-state
flatten (cdr (gopher (x)))
= **if** listp (x) **then** cdr (flatten (x))
else cons (0, **nil**) **endif**

THEOREM: correctness-of-samefringe
samefringe (x, y) = (flatten (x) = flatten (y))

DEFINITION:
prime1 (x, y)
= **if** y ≃ 0 **then** **f**
elseif y = 1 **then** **t**
else (¬ divides (y, x)) ∧ prime1 (x, y − 1) **endif**

DEFINITION:
prime (x) = ((x ≄ 0) ∧ (x ≠ 1) ∧ prime1 (x, x − 1))

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

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

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

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

THEOREM: remainder-0-crock
$(\mathtt{0}$ **mod** $y) = \mathtt{0}$

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

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

DEFINITION:
prime-factors (x)
$=$ **if** $(x \simeq \mathtt{0}) \vee ((x - 1) = \mathtt{0})$ **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** $\mathtt{1}$
else car $(l) *$ times-list (cdr $(l))$ **endif**

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

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

THEOREM: prime-list-prime-factors
$\text{prime-list}\,(\text{prime-factors}\,(x))$

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

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

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

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

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

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

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

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

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

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

THEOREM: remainder-gcd
$(\text{gcd}\,(b,\, x) = y) \rightarrow ((b\ \textbf{mod}\ y) = \mathtt{0})$

THEOREM: remainder-gcd-1
$((b\ \textbf{mod}\ x) \neq \mathtt{0}) \rightarrow (\text{gcd}\,(b,\, x) \neq x)$

THEOREM: divides-times1
$(a = (z * y)) \rightarrow ((a\ \textbf{mod}\ z) = \mathtt{0})$

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

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

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

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

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

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

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

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

THEOREM: little-step
$(\text{prime}\,(x) \wedge (y \neq \mathtt{1}) \wedge (x \neq y)) \rightarrow ((x\ \textbf{mod}\ y) \neq \mathtt{0})$

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

DEFINITION:
perm $(a,\, b)$
$=$ **if** $a \simeq$ **nil then** $b \simeq$ **nil**
elseif car $(a) \in b$ **then** perm (cdr (a), delete (car (a), b))
else f endif

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

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

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

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

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

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

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

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

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

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
$((a \not\simeq \mathtt{0}) \wedge (c \in \mathbf{N}) \wedge ((a * c) = b)) \rightarrow ((c = (b \div a)) = \mathbf{t})$

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
(plistp (*y*) ∧ ordered2 (*y*) ∧ (*x* ≠ *v*))
→ (addtolist2 (*v*, delete (*x*, *y*)) = delete (*x*, addtolist2 (*v*, *y*)))

THEOREM: delete-addtolist2
plistp (*y*) → (delete (*v*, addtolist2 (*v*, *y*)) = *y*)

THEOREM: addtolist2-kludge
((*v* ≮ *w*) ∧ (addtolist2 (*v*, *y*) = cons (*v*, *y*)))
→ (addtolist2 (*v*, addtolist2 (*w*, *y*)) = cons (*v*, addtolist2 (*w*, *y*)))

THEOREM: lessp-maximum-addtolist2
(*v* ≮ maximum (*z*)) → (addtolist2 (*v*, sort2 (*z*)) = cons (*v*, sort2 (*z*)))

THEOREM: sort2-delete-cons
listp (*x*) → (cons (maximum (*x*), delete (maximum (*x*), sort2 (*x*))) = sort2 (*x*))

THEOREM: sort2-delete
sort2 (delete (*x*, *l*)) = delete (*x*, sort2 (*l*))

THEOREM: dsort-sort2
dsort (*x*) = sort2 (*x*)

THEOREM: count-list-sort2
count-list (*a*, sort2 (*l*)) = 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:
sigma (*m*, *n*)
= **if** *m* < *n* **then** *n* + sigma (*m*, *n* − 1)
else 0 **endif**

THEOREM: difference-1
(*x* − 1) = (*x* − 1)

DEFINITION:
prog-trans-of-sigma (*i*, *ac*)
= **if** *i* ≃ 0 **then** *ac*
else prog-trans-of-sigma (*i* − 1, *ac* + *i*) **endif**

THEOREM: functional-loop-invrt
(*ac* ∈ **N**) → (prog-trans-of-sigma (*i*, *ac*) = (*ac* + sigma (0, *i*)))

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

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

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

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

```
;   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** $addr \simeq \mathtt{0}$ **then** cons $(val,$ cdr $(mem))$
else cons $(\text{car}\,(mem),$ set $(addr - 1, val,$ cdr $(mem)))$ **endif**

DEFINITION:
get $(addr, mem)$
$=$ **if** $addr \simeq \mathtt{0}$ **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 $(\mathbf{f}, mem)$
elseif get $(pc, mem) =$ `'(stop)` **then** list $(\mathbf{f}, mem)$
elseif car (get $(pc, mem)) =$ `'jumpa` **then** list (cadr (get $(pc, mem)), mem)$
elseif car (get $(pc, mem)) =$ `'skipne`
then if get (cadr (get $(pc, mem)), mem) \simeq \mathtt{0}$ **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 (cddddddr (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 ≮ 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
($max \not< 6$)
→ (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
($max \not< 6$)
→ (execute1 (4,
cons (’(movei 7 0),
cons (’(skipne 6),
cons (’(stop),
cons (’(add 7 6),

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

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

THEOREM: interpreter-loop-invrt
$((clk \not< \text{cadddddddr}\,(mem))$
$\wedge$ (car $(mem) =$ '(movei 7 0))
$\wedge$ (cadr $(mem) =$ '(skipne 6))
$\wedge$ (caddr $(mem) =$ '(stop))
$\wedge$ (cadddr $(mem) =$ '(add 7 6))
$\wedge$ (caddddr $(mem) =$ '(subi 6 1))
$\wedge$ (cadddddr $(mem) =$ '(jumpa 1)))
$\rightarrow$ (cadddddddr (execute $(1,\ mem,\ clk)$)
= **if** cadddddddr $(mem) \simeq 0$ **then** cadddddddr (mem)
else cadddddddr (mem) + sigma $(0,$ cadddddddr $(mem))$ **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) = fix(x)
```

THEOREM: half-plus

```
((x + (x + y)) ÷ 2) = (x + (y ÷ 2))
```

THEOREM: sigma-is-half-product

```
sigma(0, i) = ((i * (1 + i)) ÷ 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:
h-pr $(l,\ ac)$
= **if** $l \simeq$ **nil then** ac
else h (car (l), h-pr (cdr (l), ac)) **endif**

DEFINITION:
h-ac $(l,\ ac)$
= **if** $l \simeq$ **nil then** ac
else h-ac (cdr (l), h (car (l), ac)) **endif**

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

THEOREM: h-eq
h-ac $(l,\ ac)$ = h-pr $(l,\ ac)$

DEFINITION:
f0 (x)
= **if** 100 $< x$ **then** $x - 10$
else 91 **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: $\mathrm{even}(x) = (0 = (x \bmod 2))$

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

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

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

THEOREM: exp-of-0
$\exp(0, k)$
$=$ **if** $k \simeq 0$ **then** 1
else 0 **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
$((x + (i * j)) \bmod j) = (x \bmod j)$

THEOREM: remainder-plus-times-2
$((x + (j * i)) \bmod j) = (x \bmod j)$

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

THEOREM: remainder-of-1
$(1$ **mod** $x)$
$=$ **if** $x = 1$ **then** 0
else 1 **endif**

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

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

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

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

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

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

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

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

DEFINITION:
$\text{subst2}(new, old, term)$
$=$ **if** $term \in \mathbf{N}$ **then** $term$

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

addtolist, 11, 14, 16
addtolist-of-ordered-number-list, 14
addtolist2, 29, 30
addtolist2-delete, 30
addtolist2-kludge, 30
append-cancellation, 16
append-reverse, 6
append-right-id, 6
apply2, 40
assignedp, 21–23
assignment, 20, 22, 23
assignment-append, 22
assignment-implies-assigned, 22
associativity-of-append, 6
associativity-of-equal, 11
associativity-of-plus, 2
associativity-of-times, 7
assume-false, 22, 23
assume-true, 21–23

bagdiff, 3–5
bagint, 3, 4
big-plus, 17, 18
big-plus1, 17
boolean, 11
bridge-to-subbagp-implies-plus-tree-greatereqp, 5

cadr-crock, 7
cadr-eval$-list, 6
call, 7, 8
cancel, 4, 6
codegen, 8, 9
commutativity-of-delete, 3
commutativity-of-gcd, 18
commutativity-of-plus, 2
commutativity-of-times, 7
commutativity2-of-plus, 2
commutativity2-of-times, 7
compile, 8, 9
cons-if, 20, 21
correctness-of-big-plus, 18
correctness-of-cancel, 6
correctness-of-codegen, 9
correctness-of-functional-sigma, 31
correctness-of-get-simplifier, 33
correctness-of-interpreted-sigma, 37
correctness-of-optimize, 8
correctness-of-optimizing-compiler, 9
correctness-of-samefringe, 24
correctness-of-set-simplifier, 33
count-list, 13, 16, 30
count-list-sort, 13
count-list-sort2, 30
count-sort-lp, 16
countps-, 15
countps-countps, 15
countps-loop, 14, 15
crock-due-to-lack-of-bounce, 14

delete, 2–5, 27–30, 40
delete-addtolist2, 30
delete-non-member, 3
difference-0, 2
difference-1, 30
difference-2, 37
difference-add1-cancellation, 19
difference-cancellation-0, 2
difference-cancellation-1, 2
difference-elim, 16
difference-plus, 2
difference-plus-cancelation, 19
difference-plus1, 19
difference-plus2, 19
difference-plus3, 19
difference-x-x, 2
distributivity-of-divides, 28
distributivity-of-times-over-gcd, 20
distributivity-of-times-over-pl

us, 6
divides, 19, 20, 24–28
divides-implies-times, 28
divides-plus-rewrite, 19
divides-plus-rewrite-commuted, 19
divides-plus-rewrite1, 19
divides-plus-rewrite2, 19
divides-times, 19
divides-times-list, 28
divides-times1, 27
double, 12
double-half, 12
double-times-2, 12
dsort, 29, 30
dsort-sort2, 30

enough-factors, 26
eqp, 9, 31
equal-difference-0, 2
equal-length-0, 40
equal-lessp, 16
equal-times-0, 7
euclid, 19
eval2, 40, 41
eval$-plus-tree-append, 5
eval$-quote, 6
even, 39
even1, 12, 13
even1-double, 12
even1-even2, 13
even2, 12, 13
exec, 8, 9
execute, 32, 36, 37
execute-opened-up, 36
execute1, 31–36
execute1-1, 33
execute1-3, 34
execute1-4, 34
exp, 13, 39
exp-by-0, 39
exp-exp, 39
exp-of-0, 39
exp-of-1, 39
exp-plus, 13
exp-times, 39
expressionp, 7–9

f0, 38
fact, 15
fact-, 15
fact-fact, 15
fact-from-to, 15
fact-loop, 15
fact-loop-fact, 15
falsify, 23
falsify1, 22, 23
falsify1-extends-models, 23
falsify1-falsifies, 23
flatten, 9, 24
flatten-mc-flatten, 9
flatten-singleton, 24
formp-optimize, 8
from-to, 1, 15
functional-loop-invrt, 30

gcd, 18, 20, 27
gcd-0, 18
gcd-distributes-over-an-opened-up-times, 27
gcd-divides-both, 20
gcd-equal-0, 18
gcd-is-the-greatest, 20
get, 31, 32, 37
get-set, 31
get-simplifier, 32, 33
getvalue, 7, 8
gopher, 24
gopher-preserves-count, 24
gopher-returns-correct-state, 24
gopher-returns-leftmost-atom, 24
greatereqp, 10
greatereqpr, 26
greatereqpr-lessp, 26
greatereqpr-remainder, 26
greatest-factor, 25
greatest-factor-0, 25
greatest-factor-1, 25
greatest-factor-divides, 25

greatest-factor-lessp, 25

h, 38
h-ac, 38
h-commutivity2, 38
h-eq, 38
h-lemma, 38
h-pr, 38
hack1, 27
half, 12, 13
half-double, 12
half-plus, 37

if-complexity, 20, 21
if-complexity-goes-down1, 21
if-complexity-goes-down2, 21
if-complexity-not-0, 21
if-depth, 20
if-exprp, 20–22
if-times-then-divides, 28
iff-equal-equal, 11
interpreter-input-path, 36
interpreter-loop-invrt, 36
intersect, 10

kludge-bridge, 27

last, 12, 13
last-append, 13
last-reverse, 13
left-branch, 20–23
leftcount, 24
length, 10, 13, 19, 32, 33, 36, 40
length-5, 33
length-cons6, 33
length-delete, 40
length-reverse, 10
length-sort, 13
leq-half, 13
leq-nth, 13
lessp-count-delete, 27
lessp-delete-rewrite, 29
lessp-maximum-addtolist2, 30
lessp-not-reflexive, 9
lessp-plus-cancelation, 19
lessp-plus-cancellation3, 20
lessp-quotient1, 18
lessp-remainder-divisor, 40
lessp-remainder1, 18
lessp-remainder2, 17
lessp-times-1, 18
lessp-times-2, 18
lessp-times-cancellation, 20
listp-eval$, 6
listp-gopher, 24
little-step, 27

maximum, 29, 30
mc-flatten, 9
member-append, 9
member-delete, 3
member-implies-numberp, 5
member-implies-plus-tree-greate
reqp, 5
member-intersect, 10
member-maximum, 29
member-reverse, 9
member-sort, 13
member-union, 10

normalize, 21–23
normalize-is-sound, 23
normalize-normalizes, 23
normalized-if-exprp, 21–23
nth, 10, 11, 13, 19
nth-0, 11
nth-append, 19
nth-append1, 11
nth-nil, 11
number-listp, 13, 14
numberp-call, 7
numberp-count, 16
numberp-eval$-bridge, 5
numberp-eval$-plus, 5
numberp-eval$-plus-tree, 5
numberp-gcd, 18
numberp-greatest-factor, 25

occur, 14

occur-subst, 14
odd, 12
optimize, 8
ordered, 11, 13, 14, 16
ordered-addtolist, 16
ordered-append, 13
ordered-sort, 14
ordered-sort-lp, 16
ordered2, 29, 30

perm, 27, 28
plistp, 6, 9, 10, 16, 29, 30
plistp-reverse, 6
plus-add1, 1
plus-cancellation, 2
plus-equal-0, 2
plus-fringe, 4, 5
plus-right-id2, 1
plus-tree, 4, 5
plus-tree-bagdiff, 5
plus-tree-delete, 5
plus-tree-plus-fringe, 5
pop, 8
power-eval, 17, 18
power-eval-big-plus, 17
power-eval-big-plus1, 17
power-eval-power-rep, 18
power-rep, 18
prime, 24, 25, 27, 28, 40
prime-basic, 26
prime-factorization-existence, 26
prime-factorization-uniqueness, 28
prime-factors, 25, 26
prime-gcd, 27
prime-key, 27
prime-key-rewrite, 40
prime-list, 25, 26, 28
prime-list-append, 26
prime-list-delete, 28
prime-list-prime-factors, 26
prime-list-times-list, 28
prime-member, 28
prime1, 24–27
prime1-basic, 26
prog-trans-of-sigma, 30, 31
push, 7–9

quotient-divides, 27
quotient-lessp, 26
quotient-times, 28
quotient-times1, 26

remainder-0-crock, 25
remainder-add1, 19
remainder-difference-times, 40
remainder-gcd, 27
remainder-gcd-1, 27
remainder-of-1, 40
remainder-plus-times-1, 39
remainder-plus-times-2, 39
remainder-quotient, 17
remainder-quotient-elim, 18
remainder-times, 28
remainder-times-1, 40
remainder-wrt-1, 17
remainder-wrt-12, 17
remainder-x-x, 17
reverse, 6, 8–10, 13, 16
reverse-, 15, 16
reverse-append, 16
reverse-loop, 15, 16
reverse-loop-append-reverse, 16
reverse-loop-reverse, 16
reverse-reverse, 9
reverse-reverse-, 16
right-branch, 20–23

samefringe, 24
sequential-execution, 9
set, 31, 32
set-simplifier, 33
sigma, 30, 31, 36, 37
sigma-input-path, 31
sigma-is-half-product, 37
sigma-loop-invrt, 31
sigma-output-path, 31
sort, 11, 13, 14
sort-lp, 16

sort-of-ordered-number-list, 14
sort-ordered, 14
sort2, 29, 30
sort2-delete, 30
sort2-delete-cons, 30
sort2-gen, 29
sort2-gen-1, 29
sort2-gen-2, 29
square, 39
subbagp, 3, 5
subbagp-bagint1, 3
subbagp-bagint2, 3
subbagp-cdr1, 3
subbagp-cdr2, 3
subbagp-delete, 3
subbagp-implies-plus-tree-greatereqp, 5
subsetp, 10, 12
subsetp-cons, 12
subsetp-intersect, 10
subsetp-union, 10
subst, 14
subst-a-a, 14
subst2, 40, 41
subst2-ok, 41

tautology-checker, 22, 23
tautology-checker-completeness-bridge, 23
tautology-checker-is-complete, 23
tautology-checker-is-sound, 23
tautology-checker-soundness-bridge, 23
tautologyp, 22, 23
tautologyp-fails-means-falsify1-wins, 23
tautologyp-is-sound, 22
term-eval, 7–9
test, 20–23
times-1, 39
times-2, 39
times-add1, 6
times-difference, 19
times-equal-1, 28
times-id-iff-1, 26
times-identity, 27
times-identity1, 27
times-list, 25, 26, 28, 40
times-list-append, 26
times-list-delete, 28
times-times-list-delete, 40
times-zero2, 6
top, 8
transitivity-of-leq, 11
transitivity-of-lessp, 9
trichotomy-of-lessp, 9

value, 20, 22, 23
value-can-ignore-redundant-assignments, 22
value-short-cut, 22

xor, 14