EVENT: Start with the library "naturals" using the compiled version.

```
;; By Matt Kaufmann, modified from earlier integer library of Bill
;; Bevier and Matt Wilding.  A few functions (even ILESSP) have
;; been changed, but I expect the functionality of this library to
;; include all the functionality of the old one in most or even all
;; cases.

;; Modified from /local/src/nqthm-libs/integers.events to get ILEQ
;; expressed in terms of ILESSP and IDIFFERENCE in terms of INEG and
;; IPLUS.  There are other changes too.  The highlights are the new
;; metalemmas.

;; I'm going to leave the eval$ rules on that are proved here, and
;; leave eval$ off.

;; My intention is that this library be used in a mode in which ILEQ
;; and IDIFFERENCE are left enabled.  Otherwise, the aforementioned
;; meta lemmas may not be very useful, and also a number of additional
;; replacement rules may be needed.

;; There are three theories created by this library.  INTEGER-DEFNS is
;; a list of definitions of all integer functions (not including the
;; cancellation metafunctions and their auxiliaries, though), except
;; that ILEQ and IDIFFERENCE have been omitted.  This is a useful
;; theory for an ENABLE-THEORY hint when one simply wants to blast all
;; integer functions open, and it's also useful if one wants to close
;; them down with a DISABLE-THEORY hint (perhaps to go with an
;; (ENABLE-THEORY T) hint).  Second, ALL-INTEGER-DEFNS is the same as
;; INTEGER-DEFNS except that ILEQ and IDIFFERENCE are included in this
;; one.  Finally, INTEGERS is a list of all events to be "exported as
;; enabled" from this file when working in a mode where everything not
;; enabled by an ENABLE-THEORY hint is to be disabled.  Notice that
;; some rewrite rules have been included that might appear to be
;; unnecessary in light of the metalemmas; that's because metalemmas
;; only work on tame terms.  However, there's no guarantee that the
;; rewrite rules alone will prove very useful (on non-tame terms).
;; Also notice that INTEGER-DEFNS is disjoint from INTEGERS, since we
;; expect the basic definitions (other than ILEQ and IDIFFERENCE) to
;; remain disabled.

;; It's easy to see what I have and haven't placed in INTEGERS, since
;; I'll simply comment out the event names that I want to exclude (see
;; end of this file).

;; One might wish to consider changing (fix-int (minus ...)) in some
;; of the definitions below to (ineg ...).

;; The following meta rules are in this library.
;; (A little documentation added by Matt Wilding July 90)
;;
;; CORRECTNESS-OF-CANCEL-INEG
;;  cancel the first argument of an iplus term with a member of the second
;;  argument.
;;
;;  ex:  (iplus (ineg y) (iplus (ineg x) (iplus y z)))
;;     -->
;;       (iplus (ineg x) (fix-int z))
;;
;; CORRECTNESS-OF-CANCEL-IPLUS
;;  cancel the sides of an equality of iplus sums
;;
;;  ex:  (equal (iplus x (iplus y z)) (iplus a (iplus z x)))
;;     -->
;;       (equal (fix-int y) (fix-int a))
;;
;; CORRECTNESS-OF-CANCEL-IPLUS-ILESSP
;;  cancel the sides of an ilessp inequality of sums
;;
;;  ex:  (ilessp (iplus x (iplus y z)) (iplus a (iplus z x)))
;;     -->
;;       (ilessp y a)
;;
;; CORRECTNESS-OF-CANCEL-ITIMES
;;  cancel the sides of an equality of itimes products
;;
;;  ex: (equal (itimes x (itimes y z)) (itimes a (itimes z x)))
;;     -->
;;      (if (equal (itimes x z) '0)
;;        t
;;        (equal (fix-int y) (fix-int a)))
;;
;; CORRECTNESS-OF-CANCEL-ITIMES-ILESSP
;;  cancel the sides of an inequality of itimes products
;;
;;  ex: (ilessp (itimes x (itimes y z)) (itimes a (itimes z x)))
;;     -->
;;      (if (ilessp (itimes x z) '0)
;;         (ilessp a y)
;;        (if (ilessp 0 (itimes x z))
;;           (ilessp y a)
;;           f))
;;
;; CORRECTNESS-OF-CANCEL-ITIMES-FACTORS
;;  cancel factors in equality terms
;;  ex: (equal (iplus (itimes x y) x) (itimes z x))
;;     -->
;;      (if (equal (fix-int x) '0)
;;         t
;;        (equal (fix-int (plus y 1)) (fix-int z)))
;;
;; CORRECTNESS-OF-CANCEL-ITIMES-ILESSP-FACTORS
;;  cancel factors in ilessp terms
;;  ex: (equal (iplus (itimes x y) x) (itimes z x))
;;     -->
;;      (if (ilessp x '0)
;;         (ilessp z (iplus y 1))
;;        (if (ilessp '0 x)
;;           (ilessp (iplus y '1) z)
;;          f))
;;
;; CORRECTNESS-OF-CANCEL-FACTORS-0
;;  factor one side of equality when other side is constant 0
;;
;;  ex: (equal (iplus x (itimes x y)) '0)
;;     -->
;;      (or (equal (fix-int (iplus '1 y)) '0)
;;          (equal (fix-int x) '0))
;;
;; CORRECTNESS-OF-CANCEL-FACTORS-ILESSP-0
;;  factor one side of inequality when other side is constant 0
;;
;;  ex: (ilessp (iplus x (itimes x y)) '0)
;;     -->
;;      (or (and (ilessp (iplus '1 y) '0)
;;               (ilessp '0 x))
;;          (and (ilessp '0 (iplus '1 y))
;;               (ilessp x '0)))
;;
;; CORRECTNESS-OF-CANCEL-INEG-TERMS-FROM-EQUALITY
;;  rewrite equality to remove ineg terms
```

```
;;
;;  ex: (equal (iplus (ineg x) (ineg y)) (iplus (ineg z) w))
;;     -->
;;      (equal (fix-int z) (iplus x (iplus y w)))
;;
;; CORRECTNESS-OF-CANCEL-INEG-TERMS-FROM-INEQUALITY
;;  rewrite inequalities to remove ineg terms
;;
;;  ex: (ilessp (iplus (ineg x) (ineg y)) (iplus (ineg z) w))
;;     -->
;;      (ilessp (fix-int z) (iplus x (iplus y w)))


;(note-lib "/local/src/nqthm-libs/naturals")

;(compile-uncompiled-defns "xxx")

; ---------------------------------------------------------------------------------
; Integers
; ---------------------------------------------------------------------------------

#| The function below has no AND or OR, for efficiency
(defn integerp (x)
  (or (numberp x)
      (and (negativep x)
   (not (zerop (negative-guts x))))))
|#
```

DEFINITION:
integerp (x)
$=$ **if** $x \in \mathbf{N}$ **then t**
elseif negativep (x) **then** negative-guts $(x) \not\simeq 0$
else f endif

DEFINITION:
fix-int (x)
$=$ **if** integerp (x) **then** x
else 0 **endif**

```
;; Even though I'll include a definition for izerop here, I'll
;; often avoid using it.
```

DEFINITION: izerop (i) = (fix-int $(i) = 0$)

```
#| old version:
(defn izerop (i)
  (if (integerp i)
      (equal i 0)
      t))
|#
```

DEFINITION:
ilessp (i, j)

```
=   if negativep ($i$)
    then if negativep ($j$)  then negative-guts ($j$) $<$ negative-guts ($i$)
         elseif $i = (- 0)$  then $0 < j$
         else t endif
    elseif negativep ($j$)  then f
    else $i < j$ endif
```

DEFINITION: ileq (i, j) = ($\neg$ ilessp (j, i))

DEFINITION:
iplus (x, y)

```
=   if negativep ($x$)
    then if negativep ($y$)
         then if (negative-guts ($x$) $\simeq 0$) $\land$ (negative-guts ($y$) $\simeq 0$)  then 0
              else $-$ (negative-guts ($x$) + negative-guts ($y$)) endif
         elseif $y <$ negative-guts ($x$)  then $-$ (negative-guts ($x$) $- y$)
         else $y -$ negative-guts ($x$) endif
    elseif negativep ($y$)
    then if $x <$ negative-guts ($y$)  then $-$ (negative-guts ($y$) $- x$)
         else $x -$ negative-guts ($y$) endif
    else $x + y$ endif
```

DEFINITION:
ineg (x)

```
=   if negativep ($x$)  then negative-guts ($x$)
    elseif $x \simeq 0$  then 0
    else $- x$ endif
```

DEFINITION: idifference (x, y) = iplus (x, ineg (y))

DEFINITION:
iabs (i)

```
=   if negativep ($i$)  then negative-guts ($i$)
    else fix ($i$) endif
```

Definition:

```
itimes (i, j)
=   if negativep (i)
    then if negativep (j) then negative-guts (i) ∗ negative-guts (j)
         else fix-int (− (negative-guts (i) ∗ j)) endif
    elseif negativep (j) then fix-int (− (i ∗ negative-guts (j)))
    else i ∗ j endif
```

Definition:

```
iquotient (i, j)
=   if fix-int (j) = 0 then 0
    elseif negativep (i)
    then if negativep (j)
         then if (negative-guts (i) mod negative-guts (j)) = 0
              then negative-guts (i) ÷ negative-guts (j)
              else 1 + (negative-guts (i) ÷ negative-guts (j)) endif
         elseif (negative-guts (i) mod j) = 0
         then fix-int (− (negative-guts (i) ÷ j))
         else fix-int (− (1 + (negative-guts (i) ÷ j))) endif
    elseif negativep (j) then fix-int (− (i ÷ negative-guts (j)))
    else i ÷ j endif
```

Definition:

```
iremainder (i, j) = idifference (i, itimes (j, iquotient (i, j)))
```

Definition:

```
idiv (i, j)
=   if fix-int (j) = 0 then 0
    elseif negativep (i)
    then if negativep (j) then negative-guts (i) ÷ negative-guts (j)
         elseif (negative-guts (i) mod j) = 0
         then fix-int (− (negative-guts (i) ÷ j))
         else fix-int (− (1 + (negative-guts (i) ÷ j))) endif
    elseif negativep (j)
    then if (i mod negative-guts (j)) = 0
         then fix-int (− (i ÷ negative-guts (j)))
         else fix-int (− (1 + (i ÷ negative-guts (j)))) endif
    else i ÷ j endif
```

Definition:

```
imod (i, j) = idifference (fix-int (i), itimes (j, idiv (i, j)))
```

Definition:

```
iquo (i, j)
=   if fix-int (j) = 0 then 0
```

```
elseif negativep $(i)$
then if negativep $(j)$  then negative-guts $(i) \div$ negative-guts $(j)$
        else fix-int $(-$ (negative-guts $(i) \div j))$ endif
elseif negativep $(j)$  then fix-int $(-$ $(i \div$ negative-guts $(j)))$
else $i \div j$ endif
```

DEFINITION:
irem (i, j) = idifference (fix-int (i), itimes $(j$, iquo $(i, j)))$

```
; ---------- DEFTHEORY events for definitions ----------
```

EVENT: Let us define the theory *integer-defns* to consist of the following events: integerp, fix-int, ilessp, iplus, ineg, iabs, itimes, iquotient, iremainder, idiv, imod, iquo, irem.

EVENT: Let us define the theory *all-integer-defns* to consist of the following events: integerp, fix-int, izerop, ilessp, ileq, iplus, ineg, idifference, iabs, itimes, iquotient, iremainder, idiv, imod, iquo, irem.

EVENT: Disable integerp.

EVENT: Disable fix-int.

EVENT: Disable ilessp.

EVENT: Disable iplus.

EVENT: Disable ineg.

EVENT: Disable iabs.

EVENT: Disable itimes.

```
;; I've disabled the rest later in the file, just because the lemmas
;; about division were (re-)proved with the remaining functions enabled.

; ---------- INTEGERP ----------
```

THEOREM: integerp-fix-int
integerp (fix-int (x))

THEOREM: integerp-iplus
integerp (iplus $(x,\ y)$)

THEOREM: integerp-idifference
integerp (idifference $(x,\ y)$)

THEOREM: integerp-ineg
integerp (ineg (x))

THEOREM: integerp-iabs
integerp (iabs (x))

THEOREM: integerp-itimes
integerp (itimes $(x,\ y)$)

```
; ---------- FIX-INT ----------

;; The first of these, FIX-INT-REMOVER, is potentially dangerous from
;; a backchaining point of view, but I believe it's necessary.  At least
;; the lemmas below it should go a long way toward preventing its application.
```

THEOREM: fix-int-remover
integerp $(x) \rightarrow$ (fix-int $(x) = x$)

THEOREM: fix-int-fix-int
fix-int (fix-int (x)) = fix-int (x)

THEOREM: fix-int-iplus
fix-int (iplus $(a,\ b)$) = iplus $(a,\ b)$

THEOREM: fix-int-idifference
fix-int (idifference $(a,\ b)$) = idifference $(a,\ b)$

THEOREM: fix-int-ineg
fix-int (ineg (x)) = ineg (x)

THEOREM: fix-int-iabs
fix-int (iabs (x)) = iabs (x)

THEOREM: fix-int-itimes
fix-int (itimes $(x,\ y)$) = itimes $(x,\ y)$

```
; ---------- INEG ----------
```

THEOREM: ineg-iplus
$\text{ineg}\,(\text{iplus}\,(a,\ b)) = \text{iplus}\,(\text{ineg}\,(a),\ \text{ineg}\,(b))$

THEOREM: ineg-ineg
$\text{ineg}\,(\text{ineg}\,(x)) = \text{fix-int}\,(x)$

THEOREM: ineg-fix-int
$\text{ineg}\,(\text{fix-int}\,(x)) = \text{ineg}\,(x)$

THEOREM: ineg-of-non-integerp
$(\neg\ \text{integerp}\,(x)) \rightarrow (\text{ineg}\,(x) = \mathtt{0})$

```
;; I don't want the backchaining to slow down the prover.
```

EVENT: Disable ineg-of-non-integerp.

THEOREM: ineg-0
$\text{ineg}\,(\mathtt{0}) = \mathtt{0}$

```
; ---------- IPLUS ----------

;; The first two of these really aren't necessary, in light
;; of the cancellation metalemma.
```

THEOREM: iplus-left-id
$(\neg\ \text{integerp}\,(x)) \rightarrow (\text{iplus}\,(x,\ y) = \text{fix-int}\,(y))$

```
;; I don't want the backchaining to slow down the prover.
```

EVENT: Disable iplus-left-id.

THEOREM: iplus-right-id
$(\neg\ \text{integerp}\,(y)) \rightarrow (\text{iplus}\,(x,\ y) = \text{fix-int}\,(x))$

```
;; I don't want the backchaining to slow down the prover.
```

EVENT: Disable iplus-right-id.

THEOREM: iplus-0-left
$\text{iplus}\,(\mathtt{0},\ x) = \text{fix-int}\,(x)$

Theorem: iplus-0-right
iplus $(x,\ \mathtt{0})$ = fix-int (x)

Theorem: commutativity2-of-iplus
iplus $(x,$ iplus $(y,\ z))$ = iplus $(y,$ iplus $(x,\ z))$

Theorem: commutativity-of-iplus
iplus $(x,\ y)$ = iplus $(y,\ x)$

Theorem: associativity-of-iplus
iplus (iplus $(x,\ y),\ z)$ = iplus $(x,$ iplus $(y,\ z))$

Theorem: iplus-cancellation-1
(iplus $(a,\ b)$ = iplus $(a,\ c)$) = (fix-int (b) = fix-int (c))

Theorem: iplus-cancellation-2
(iplus $(b,\ a)$ = iplus $(c,\ a)$) = (fix-int (b) = fix-int (c))

Theorem: iplus-ineg1
iplus (ineg $(a),\ a)$ = $\mathtt{0}$

Theorem: iplus-ineg2
iplus $(a,$ ineg $(a))$ = $\mathtt{0}$

Theorem: iplus-fix-int1
iplus (fix-int $(a),\ b)$ = iplus $(a,\ b)$

Theorem: iplus-fix-int2
iplus $(a,$ fix-int $(b))$ = iplus $(a,\ b)$

```
; ---------- IDIFFERENCE ----------

;; mostly omitted, but I'll keep a few
```

Theorem: idifference-fix-int1
idifference (fix-int $(a),\ b)$ = idifference $(a,\ b)$

Theorem: idifference-fix-int2
idifference $(a,$ fix-int $(b))$ = idifference $(a,\ b)$

```
; ------------------------------------------------------------------------------
; Cancel INEG
; ------------------------------------------------------------------------------

;; We assume that the given term (IPLUS x y) has the property that y has already
;; been reduced and x is not an iplus-term.  So, the only question is whether
;; or not the formal negative of x appears in the fringe of y.

#| The function below has no AND or OR, for efficiency
(defn cancel-ineg-aux (x y)
  ;; returns nil or else a new term provably equal to (IPLUS x y)
  (if (and (listp x)
   (equal (car x) 'ineg))
      (cond
       ((equal y (cadr x))
''0)
       ((and (listp y)
     (equal (car y) 'iplus))
(let ((y1 (cadr y)) (y2 (caddr y)))
  (if (equal y1 (cadr x))
      (list 'fix-int y2)
    (let ((z (cancel-ineg-aux x y2)))
      (if z
  (list 'iplus y1 z)
f)))))
       (t f))
    (cond
     ((nlistp y)
      f)
     ((equal (car y) 'ineg)
      (if (equal x (cadr y))
  ''0
f))
     ((equal (car y) 'iplus)
      (let ((y1 (cadr y)) (y2 (caddr y)))
(if (and (listp y1)
 (equal (car y1) 'ineg)
 (equal x (cadr y1)))
    (list 'fix-int y2)
  (let ((z (cancel-ineg-aux x y2)))
    (if z
(list 'iplus y1 z)
      f)))))
     (t f))))
|#
```

Definition:
cancel-ineg-aux (x, y)
$=$ **if** listp (x)

```
then if car ($x$) = 'ineg
     then if $y$ = cadr ($x$)  then ''0
          elseif listp ($y$)
          then if car ($y$) = 'iplus
               then if cadr ($y$) = cadr ($x$)
                    then list ('fix-int, caddr ($y$))
                    elseif cancel-ineg-aux ($x$, caddr ($y$))
                    then list ('iplus,
                               cadr ($y$),
                               cancel-ineg-aux ($x$, caddr ($y$)))
                    else f endif
               else f endif
          else f endif
     elseif $y \simeq$ nil  then f
     elseif car ($y$) = 'ineg
     then if $x$ = cadr ($y$)  then ''0
          else f endif
     elseif car ($y$) = 'iplus
     then if listp (cadr ($y$))
          then if caadr ($y$) = 'ineg
               then if $x$ = cadadr ($y$)
                    then list ('fix-int, caddr ($y$))
                    elseif cancel-ineg-aux ($x$, caddr ($y$))
                    then list ('iplus,
                               cadr ($y$),
                               cancel-ineg-aux ($x$, caddr ($y$)))
                    else f endif
               elseif cancel-ineg-aux ($x$, caddr ($y$))
               then list ('iplus, cadr ($y$), cancel-ineg-aux ($x$, caddr ($y$)))
               else f endif
          elseif cancel-ineg-aux ($x$, caddr ($y$))
          then list ('iplus, cadr ($y$), cancel-ineg-aux ($x$, caddr ($y$)))
          else f endif
     else f endif
elseif $y \simeq$ nil  then f
elseif car ($y$) = 'ineg
then if $x$ = cadr ($y$)  then ''0
     else f endif
elseif car ($y$) = 'iplus
then if listp (cadr ($y$))
     then if caadr ($y$) = 'ineg
          then if $x$ = cadadr ($y$)  then list ('fix-int, caddr ($y$))
               elseif cancel-ineg-aux ($x$, caddr ($y$))
               then list ('iplus, cadr ($y$), cancel-ineg-aux ($x$, caddr ($y$)))
```

```
                    else f endif
                elseif cancel-ineg-aux ($x$, caddr ($y$))
                then list ('iplus, cadr ($y$), cancel-ineg-aux ($x$, caddr ($y$)))
                else f endif
            elseif cancel-ineg-aux ($x$, caddr ($y$))
            then list ('iplus, cadr ($y$), cancel-ineg-aux ($x$, caddr ($y$)))
            else f endif
    else f endif
```

```
#| The function below has no AND or OR, for efficiency
(defn cancel-ineg (x)
  (if (and (listp x)
   (equal (car x) 'iplus))
      (let ((temp (cancel-ineg-aux (cadr x) (caddr x))))
(if temp
    temp
  x))
    x))
|#
```

DEFINITION:
cancel-ineg (x)

```
=   if listp ($x$)
    then if car ($x$) = 'iplus
         then if cancel-ineg-aux (cadr ($x$), caddr ($x$))
              then cancel-ineg-aux (cadr ($x$), caddr ($x$))
              else $x$ endif
         else $x$ endif
    else $x$ endif
```

```
;; It seems a big win to turn off eval$.  I'll leave the recursive step out in
;; hopes that rewrite-eval$ handles it OK.
```

THEOREM: eval\$-list-cons
eval\$ ('list, cons (x, y), a) = cons (eval\$ (**t**, x, a), eval\$ ('list, y, a))

THEOREM: eval\$-list-nlistp
($x \simeq$ **nil**) $\rightarrow$ (eval\$ ('list, x, a) = **nil**)

THEOREM: eval\$-litatom
litatom (x) $\rightarrow$ (eval\$ (**t**, x, a) = cdr (assoc (x, a)))

```
#|
(prove-lemma eval$-quotep (rewrite)
  (equal (eval$ t (list 'quote x) a)
 x))
|#

;; In place of the above I'll do the following, from
;; the naturals library.
```

EVENT: Enable eval$-quote.

THEOREM: eval$-other
$((\neg \text{litatom}(x)) \land (x \simeq \textbf{nil})) \rightarrow (\text{eval\$}(\textbf{t}, x, a) = x)$

EVENT: Disable eval$.

```
;; What I'd like to do is say what (eval$ t (cancel-ineg-aux x y) a),
;; but a rewrite rule will loop because of the recursion.  So I
;; introduce a silly auxiliary function so that the opening-up
;; heuristics can help me.  The function body has (listp y) tests
;; so that it can be accepted.
```

DEFINITION:
eval$-cancel-ineg-aux-fn (x, y, a)

```
=   if listp (x) ∧ (car (x) = 'ineg)
    then if y = cadr (x) then 0
         else let y1 be cadr (y),
                  y2 be caddr (y)
              in
              if y1 = cadr (x) then fix-int (eval$ (t, y2, a))
              elseif listp (y)
              then iplus (eval$ (t, y1, a),
                          eval$-cancel-ineg-aux-fn (x, y2, a))
              else 0 endif endlet endif
    else if car (y) = 'ineg then 0
         else let y1 be cadr (y),
                  y2 be caddr (y)
              in
              if listp (y1)
                 ∧  (car (y1) = 'ineg)
                 ∧  (x = cadr (y1))
              then fix-int (eval$ (t, y2, a))
              elseif listp (y)
              then iplus (eval$ (t, y1, a),
```

eval\$-cancel-ineg-aux-fn (x,
$y2$,
a))
else 0 **endif endlet endif endif**

THEOREM: eval\$-cancel-ineg-aux-is-its-fn
(cancel-ineg-aux $(x,\ y) \neq \mathbf{f}$)
$\rightarrow$ (eval\$ ($\mathbf{t}$, cancel-ineg-aux $(x,\ y),\ a) =$ eval\$-cancel-ineg-aux-fn $(x,\ y,\ a)$)

THEOREM: iplus-ineg3
iplus (ineg (x), iplus $(x,\ y)) =$ fix-int (y)

THEOREM: iplus-ineg4
iplus $(x$, iplus (ineg $(x),\ y)) =$ fix-int (y)

THEOREM: iplus-ineg-promote
iplus $(y$, ineg $(x)) =$ iplus (ineg $(x),\ y)$

THEOREM: iplus-x-y-ineg-x
iplus $(x$, iplus $(y$, ineg $(x))) =$ fix-int (y)

EVENT: Disable iplus-ineg-promote.

THEOREM: correctness-of-cancel-ineg-aux
(cancel-ineg-aux $(x,\ y) \neq \mathbf{f}$)
$\rightarrow$ (eval\$-cancel-ineg-aux-fn $(x,\ y,\ a) =$ iplus (eval\$ ($\mathbf{t}$, x, a), eval\$ ($\mathbf{t}$, y, a)))

THEOREM: correctness-of-cancel-ineg
eval\$ ($\mathbf{t}$, x, a) = eval\$ ($\mathbf{t}$, cancel-ineg (x), a)

EVENT: Disable correctness-of-cancel-ineg-aux.

```
; ------------------------------------------------------------------------------
; Cancel IPLUS
; ------------------------------------------------------------------------------

;; All I do here is cancel like terms from both sides.  The problem of handling
;; INEG cancellation IS handled completely separately above.  That hasn't always
;; been the case -- in my first try I attempted to integrate the operations.
;; But now I see that for things like (equal z (iplus x (iplus y (ineg x))))
;; the integrated approach will fail.  Also, thanks to Matt Wilding, for pointing
;; out that the "four squares" example that Bill Pase sent me ran faster with
;; the newer approach (on his previously-implemented version for the rationals).
```

```
#| The function below has no AND or OR, for efficiency
(defn iplus-fringe (x)
  (if (and (listp x)
   (equal (car x)
  (quote iplus)))
      (append (iplus-fringe (cadr x))
      (iplus-fringe (caddr x)))
      (cons x nil)))
|#
```

DEFINITION:
iplus-fringe (x)

```
=   if listp (x)
    then if car (x) = 'iplus
         then append (iplus-fringe (cadr (x)), iplus-fringe (caddr (x)))
         else list (x) endif
    else list (x) endif
```

THEOREM: lessp-count-listp-cdr
listp (cdr (x)) $\rightarrow$ (count (cdr (x)) $<$ count (x))

DEFINITION:
iplus-tree-rec (l)

```
=   if cdr (l) ≃ nil then car (l)
    else list ('iplus, car (l), iplus-tree-rec (cdr (l))) endif
```

DEFINITION:
iplus-tree (l)

```
=   if listp (l)
    then if listp (cdr (l)) then iplus-tree-rec (l)
         else list ('fix-int, car (l)) endif
    else ''0 endif
```

DEFINITION:
iplus-list (x)

```
=   if listp (x) then iplus (car (x), iplus-list (cdr (x)))
    else 0 endif
```

THEOREM: integerp-iplus-list
integerp (iplus-list (x))

THEOREM: eval\$-iplus-tree-rec
eval\$ (**t**, iplus-tree-rec (x), a)

```
=   if listp (x)
```

```
    then if listp (cdr (x))  then iplus-list (eval$ ('list, x, a))
         else eval$ (t, car (x), a) endif
    else 0 endif
```

THEOREM: eval$-iplus-tree
$\text{eval\$}(\mathbf{t}, \text{iplus-tree}(x), a) = \text{iplus-list}(\text{eval\$}(\texttt{'list}, x, a))$

THEOREM: eval$-list-append
$\text{eval\$}(\texttt{'list}, \text{append}(x, y), a)$
$= \text{append}(\text{eval\$}(\texttt{'list}, x, a), \text{eval\$}(\texttt{'list}, y, a))$

```
#| The function below has no AND or OR, for efficiency
(defn cancel-iplus (x)
  (if (and (listp x)
   (equal (car x) (quote equal)))
      (if (and (listp (cadr x))
       (equal (caadr x) (quote iplus))
       (listp (caddr x))
       (equal (caaddr x) (quote iplus)))
  (let ((xs (iplus-fringe (cadr x)))
(ys (iplus-fringe (caddr x))))
    (let ((bagint (bagint xs ys)))
      (if (listp bagint)
  (list (quote equal)
(iplus-tree (bagdiff xs bagint))
(iplus-tree (bagdiff ys bagint)))
x)))
(if (and (listp (cadr x))
 (equal (caadr x) (quote iplus))
 ;; We don't want to introduce the IF below unless something
 ;; is "gained", or else we may get into an infinite rewriting loop.
 (member (caddr x) (iplus-fringe (cadr x))))
    (list (quote if)
  (list (quote integerp) (caddr x))
  (list (quote equal)
(iplus-tree (delete (caddr x) (iplus-fringe (cadr x))))
''0)
  (list (quote quote) f))
  (if (and (listp (caddr x))
   (equal (caaddr x) (quote iplus))
   (member (cadr x) (iplus-fringe (caddr x))))
      (list (quote if)
    (list (quote integerp) (cadr x))
    (list (quote equal)
  ''0
  (iplus-tree (delete (cadr x) (iplus-fringe (caddr x)))))
    (list (quote quote) f))
    x)))
    x))
|#
```

Definition:

```
cancel-iplus ($x$)
=   if listp ($x$)
    then if car ($x$) = 'equal
         then if listp (cadr ($x$))
              then if caadr ($x$) = 'iplus
                   then if listp (caddr ($x$))
                        then if caaddr ($x$) = 'iplus
                             then if listp (bagint (iplus-fringe (cadr ($x$)),
                                                    iplus-fringe (caddr ($x$))))
                                  then list ('equal,
                                             iplus-tree (bagdiff (iplus-fringe (cadr ($x$)),
                                                                  bagint (iplus-fringe (cadr ($x$)),
                                                                          iplus-fringe (caddr ($x$))))),
                                             iplus-tree (bagdiff (iplus-fringe (caddr ($x$)),
                                                                  bagint (iplus-fringe (cadr ($x$)),
                                                                          iplus-fringe (caddr ($x$))))))
                                  else $x$ endif
                             elseif caddr ($x$) $\in$ iplus-fringe (cadr ($x$))
                             then list ('if,
                                        list ('integerp, caddr ($x$)),
                                        cons ('equal,
                                              cons (iplus-tree (delete (caddr ($x$),
                                                                        iplus-fringe (cadr ($x$)))),
                                                    '('0))),
                                        list ('quote, f))
                             else $x$ endif
                        elseif caddr ($x$) $\in$ iplus-fringe (cadr ($x$))
                        then list ('if,
                                   list ('integerp, caddr ($x$)),
                                   cons ('equal,
                                         cons (iplus-tree (delete (caddr ($x$),
                                                                   iplus-fringe (cadr ($x$)))),
                                               '('0))),
                                   list ('quote, f))
                        else $x$ endif
                   elseif listp (caddr ($x$))
```

```
                    then if caaddr (x) = 'iplus
                         then if cadr (x) ∈ iplus-fringe (caddr (x))
                              then list ('if,
                                         list ('integerp, cadr (x)),
                                         list ('equal,
                                              ''0,
                                              iplus-tree (delete (cadr (x),
                                                                  iplus-fringe (caddr (x))))),
                                         list ('quote, f))
                              else x endif
                         else x endif
                    else x endif
               elseif listp (caddr (x))
               then if caaddr (x) = 'iplus
                    then if cadr (x) ∈ iplus-fringe (caddr (x))
                         then list ('if,
                                    list ('integerp, cadr (x)),
                                    list ('equal,
                                         ''0,
                                         iplus-tree (delete (cadr (x),
                                                             iplus-fringe (caddr (x))))),
                                    list ('quote, f))
                         else x endif
                    else x endif
               else x endif
          else x endif
     else x endif
```

THEOREM: eval$-cancel-iplus

```
eval$ (t, cancel-iplus (x), a)
=   if listp (x) ∧ (car (x) = 'equal)
    then if listp (cadr (x))
              ∧ (caadr (x) = 'iplus)
              ∧ listp (caddr (x))
              ∧ (caaddr (x) = 'iplus)
         then let xs be iplus-fringe (cadr (x)),
                  ys be iplus-fringe (caddr (x))
              in
              let bagint be bagint (xs, ys)
              in
              if listp (bagint)
              then iplus-list (eval$ ('list,
                                      bagdiff (xs,
                                               bagint (xs, ys)),
                                                   $a$))
                                   =    iplus-list (eval$ ('list,
                                                          bagdiff ($ys$,
                                                                   bagint ($xs$,
                                                                           $ys$)),
                                                          $a$))
                            else eval$ (t, $x$, $a$) endif endlet endlet
                 elseif listp (cadr ($x$))
                        ∧   (caadr ($x$) = 'iplus)
                        ∧   (caddr ($x$) ∈ iplus-fringe (cadr ($x$)))
                 then if integerp (eval$ (t, caddr ($x$), $a$))
                      then iplus-list (eval$ ('list,
                                              delete (caddr ($x$), iplus-fringe (cadr ($x$))),
                                              $a$))
                           =   0
                      else f endif
                 elseif listp (caddr ($x$))
                        ∧   (caaddr ($x$) = 'iplus)
                        ∧   (cadr ($x$) ∈ iplus-fringe (caddr ($x$)))
                 then if integerp (eval$ (t, cadr ($x$), $a$))
                      then 0 = iplus-list (eval$ ('list,
                                                  delete (cadr ($x$),
                                                          iplus-fringe (caddr ($x$))),
                                                  $a$))
                      else f endif
                 else eval$ (t, $x$, $a$) endif
          else eval$ (t, $x$, $a$) endif
```

EVENT: Disable cancel-iplus.

THEOREM: eval$-iplus-list-delete
($z \in y$)
→ (iplus-list (eval$ ('list, delete (z, y), a))
    = idifference (iplus-list (eval$ ('list, y, a)), eval$ (t, z, a)))

THEOREM: eval$-iplus-list-bagdiff
subbagp (x, y)
→ (iplus-list (eval$ ('list, bagdiff (y, x), a))
    = idifference (iplus-list (eval$ ('list, y, a)),
                   iplus-list (eval$ ('list, x, a))))

THEOREM: iplus-list-append
iplus-list (append (x, y)) = iplus (iplus-list (x), iplus-list (y))

EVENT: Disable iplus-tree.

```
;; because we want to use EVAL$-IPLUS-TREE for now
```

Theorem: not-integerp-implies-not-equal-iplus
$(\neg \text{ integerp}(a)) \rightarrow ((a = \text{iplus}(b,\ c)) = \mathbf{f})$

Theorem: iplus-list-eval\$-fringe
$\text{iplus-list}(\text{eval\$}(\texttt{'list}, \text{iplus-fringe}(x),\ a)) = \text{fix-int}(\text{eval\$}(\mathbf{t},\ x,\ a))$

```
;; The following two lemmas aren't needed but they sure do
;; shorten the total proof time!!!
```

Theorem: iplus-ineg5-lemma-1
$\text{integerp}(x)$
$\rightarrow \quad ((x = \text{iplus}(y, \text{iplus}(\text{ineg}(z),\ w)))$
$\qquad = \quad (x = \text{iplus}(\text{ineg}(z), \text{iplus}(y,\ w))))$

Theorem: iplus-ineg5-lemma-2
$(\text{integerp}(x) \wedge \text{integerp}(v))$
$\rightarrow \quad ((x = \text{iplus}(\text{ineg}(z),\ v)) = (\text{iplus}(x,\ z) = v))$

Theorem: iplus-ineg5
$\text{integerp}(x)$
$\rightarrow \quad ((x = \text{iplus}(y, \text{iplus}(\text{ineg}(z),\ w))) = (\text{iplus}(x,\ z) = \text{iplus}(y,\ w)))$

Event: Disable iplus-ineg5-lemma-1.

Event: Disable iplus-ineg5-lemma-2.

Theorem: iplus-ineg6
$\text{integerp}(x)$
$\rightarrow \quad ((x = \text{iplus}(y, \text{iplus}(w, \text{ineg}(z)))) = (\text{iplus}(x,\ z) = \text{iplus}(y,\ w)))$

Theorem: eval\$-iplus
$(\text{listp}(x) \wedge (\text{car}(x) = \texttt{'iplus}))$
$\rightarrow \quad (\text{eval\$}(\mathbf{t},\ x,\ a) = \text{iplus}(\text{eval\$}(\mathbf{t}, \text{cadr}(x),\ a), \text{eval\$}(\mathbf{t}, \text{caddr}(x),\ a)))$

Theorem: iplus-ineg7
$(\texttt{0} = \text{iplus}(x, \text{ineg}(y))) = (\text{fix-int}(y) = \text{fix-int}(x))$

Theorem: correctness-of-cancel-iplus
$\text{eval\$}(\mathbf{t},\ x,\ a) = \text{eval\$}(\mathbf{t}, \text{cancel-iplus}(x),\ a)$

EVENT: Disable iplus-ineg5.

EVENT: Disable iplus-ineg6.

```
; ------------------------------------------------------------------------------
; Cancel IPLUS from ILESSP
; ------------------------------------------------------------------------------

;; This is similar to the cancellation of IPLUS terms from equalities,
;; handled above, and uses many of the same lemmas.  A small but definite
;; difference however is that for ILESSP we don't have to fix integers.

;; By luck we have that iplus-tree-rec is appropriate here, since
;; the lemma eval$-iplus-tree-rec shows that it (accidentally) behaves
;; properly on the empty list.
```

THEOREM: ilessp-fix-int-1
ilessp (fix-int (x), y) $=$ ilessp $(x,\ y)$

THEOREM: ilessp-fix-int-2
ilessp $(x,$ fix-int $(y))$ $=$ ilessp $(x,\ y)$

```
;; Perhaps the easiest approach is to do everything with respect to the
;; same IPLUS-TREE function that we used before, and then once the
;; supposed meta-lemma is proved, go back and show that we get the
;; same answer if we use a version that doesn't fix-int singleton fringes.
```

DEFINITION:
make-cancel-iplus-inequality-1 $(x,\ y)$
$=$ list (`'ilessp`,
iplus-tree (bagdiff $(x,$ bagint $(x,\ y))$),
iplus-tree (bagdiff $(y,$ bagint $(x,\ y))$))

```
#| The function below has no AND or OR, for efficiency
(defn cancel-iplus-ilessp-1 (x)
  (if (and (listp x)
   (equal (car x) (quote ilessp)))
      (make-cancel-iplus-inequality-1 (iplus-fringe (cadr x))
      (iplus-fringe (caddr x)))
    x))
|#
```

Definition:
cancel-iplus-ilessp-1 (x)

```
=   if listp (x)
    then if car (x) = 'ilessp
         then make-cancel-iplus-inequality-1 (iplus-fringe (cadr (x)),
                                              iplus-fringe (caddr (x)))
         else x endif
    else x endif
```

```
;; Notice that IPLUS-TREE-NO-FIX-INT is currently enabled, which is
;; good since we want to use EVAL$-IPLUS-TREE-NO-FIX-INT for now.
```

Theorem: lessp-difference-plus-arg1
$(w < ((w + y) - x)) = (x < y)$

Theorem: lessp-difference-plus-arg1-commuted
$(w < ((y + w) - x)) = (x < y)$

Theorem: iplus-cancellation-1-for-ilessp
ilessp (iplus $(a,\ b)$, iplus $(a,\ c)$) = ilessp $(b,\ c)$

Theorem: iplus-cancellation-2-for-ilessp
ilessp (iplus $(b,\ a)$, iplus $(c,\ a)$) = ilessp $(b,\ c)$

Theorem: correctness-of-cancel-iplus-ilessp-lemma
eval\$ (**t**, x, a) = eval\$ (**t**, cancel-iplus-ilessp-1 (x), a)

Definition:
iplus-tree-no-fix-int (l)

```
=   if listp (l) then iplus-tree-rec (l)
    else ''0 endif
```

Theorem: eval\$-ilessp-iplus-tree-no-fix-int
ilessp (eval\$ (**t**, iplus-tree-no-fix-int (x), a),
        eval\$ (**t**, iplus-tree-no-fix-int (y), a))
= ilessp (eval\$ (**t**, iplus-tree (x), a), eval\$ (**t**, iplus-tree (y), a))

Event: Disable iplus-tree-no-fix-int.

Theorem: make-cancel-iplus-inequality-simplifier
eval\$ (**t**, make-cancel-iplus-inequality-1 $(x,\ y)$, a)

```
=   eval$ (t,
           list ('ilessp,
                 iplus-tree-no-fix-int (bagdiff (x, bagint (x, y))),
                 iplus-tree-no-fix-int (bagdiff (y, bagint (x, y)))),
           a)
```

```
#| The function below has no AND or OR, for efficiency
(defn cancel-iplus-ilessp (x)
  (if (and (listp x)
   (equal (car x) (quote ilessp)))
      (let ((x1 (iplus-fringe (cadr x)))
    (y1 (iplus-fringe (caddr x))))
(let ((bagint (bagint x1 y1)))
  (if (listp bagint)
      ;; I check (listp bagint) only for efficiency
      (list (quote ilessp)
    (iplus-tree-no-fix-int (bagdiff x1 bagint))
    (iplus-tree-no-fix-int (bagdiff y1 bagint)))
    x)))
    x))
|#

;; **** Should perhaps check that some argument of the ILESSP has function
;; symbol IPLUS, or else we may wind up dealing with (ILESSP 0 0).  That should
;; be harmless enough, though, even if *1*IPLUS is disabled; we'll just get the
;; same term back, the hard way.
```

Definition:
cancel-iplus-ilessp (x)

```
=   if listp (x)
    then if car (x) = 'ilessp
         then if listp (bagint (iplus-fringe (cadr (x)),
                                iplus-fringe (caddr (x))))
              then list ('ilessp,
                         iplus-tree-no-fix-int (bagdiff (iplus-fringe (cadr (x)),
                                                         bagint (iplus-fringe (cadr (x)),
                                                                 iplus-fringe (caddr (x))))),
                         iplus-tree-no-fix-int (bagdiff (iplus-fringe (caddr (x)),
                                                         bagint (iplus-fringe (cadr (x)),
                                                                 iplus-fringe (caddr (x))))))
              else x endif
         else x endif
    else x endif
```

Event: Disable make-cancel-iplus-inequality-1.

Theorem: correctness-of-cancel-iplus-ilessp
eval\$ ($\mathbf{t}$, x, a) = eval\$ ($\mathbf{t}$, cancel-iplus-ilessp (x), a)

```
; ---------- Multiplication ----------
```

THEOREM: itimes-zero1
$(\text{fix-int}\,(x) = \texttt{0}) \rightarrow (\text{itimes}\,(x,\ y) = \texttt{0})$

THEOREM: itimes-0-left
$\text{itimes}\,(\texttt{0},\ y) = \texttt{0}$

```
;; I don't want the backchaining to slow down the prover.
```

EVENT: Disable itimes-zero1.

THEOREM: itimes-zero2
$(\text{fix-int}\,(y) = \texttt{0}) \rightarrow (\text{itimes}\,(x,\ y) = \texttt{0})$

THEOREM: itimes-0-right
$\text{itimes}\,(x,\ \texttt{0}) = \texttt{0}$

```
;; I don't want the backchaining to slow down the prover.
```

EVENT: Disable itimes-zero2.

THEOREM: itimes-fix-int1
$\text{itimes}\,(\text{fix-int}\,(a),\ b) = \text{itimes}\,(a,\ b)$

THEOREM: itimes-fix-int2
$\text{itimes}\,(a,\ \text{fix-int}\,(b)) = \text{itimes}\,(a,\ b)$

THEOREM: commutativity-of-itimes
$\text{itimes}\,(x,\ y) = \text{itimes}\,(y,\ x)$

THEOREM: itimes-distributes-over-iplus-proof
$\text{itimes}\,(x,\ \text{iplus}\,(y,\ z)) = \text{iplus}\,(\text{itimes}\,(x,\ y),\ \text{itimes}\,(x,\ z))$

THEOREM: itimes-distributes-over-iplus
$(\text{itimes}\,(x,\ \text{iplus}\,(y,\ z)) = \text{iplus}\,(\text{itimes}\,(x,\ y),\ \text{itimes}\,(x,\ z)))$
$\land \quad (\text{itimes}\,(\text{iplus}\,(x,\ y),\ z) = \text{iplus}\,(\text{itimes}\,(x,\ z),\ \text{itimes}\,(y,\ z)))$

THEOREM: commutativity2-of-itimes
$\text{itimes}\,(x,\ \text{itimes}\,(y,\ z)) = \text{itimes}\,(y,\ \text{itimes}\,(x,\ z))$

THEOREM: associativity-of-itimes
$\text{itimes}\,(\text{itimes}\,(x,\ y),\ z) = \text{itimes}\,(x,\ \text{itimes}\,(y,\ z))$

THEOREM: equal-itimes-0
$(\text{itimes}\,(x,\ y) = \texttt{0}) = ((\text{fix-int}\,(x) = \texttt{0}) \lor (\text{fix-int}\,(y) = \texttt{0}))$

THEOREM: equal-itimes-1
$(\text{itimes}\,(a,\ b) = \texttt{1})$
$= \quad (((a = \texttt{1}) \land (b = \texttt{1})) \lor ((a = \texttt{-1}) \land (b = \texttt{-1})))$

THEOREM: equal-itimes-minus-1
$(\text{itimes}\,(a,\ b) = \texttt{-1})$
$= \quad (((a = \texttt{-1}) \land (b = \texttt{1})) \lor ((a = \texttt{1}) \land (b = \texttt{-1})))$

THEOREM: itimes-1-arg1
$\text{itimes}\,(\texttt{1},\ x) = \text{fix-int}\,(x)$

```
; ---------- Division ----------
```

THEOREM: quotient-remainder-uniqueness
$((a = (r + (b * q))) \land (r < b))$
$\rightarrow \quad ((\text{fix}\,(r) = (a\ \mathbf{mod}\ b)) \land (\text{fix}\,(q) = (a \div b)))$

```
; We want to define IQUOTIENT and IREMAINDER. The standard approach to
; integer division derives from from the following theorem.
;
; Division Theorem:
; For all integers i,j, j not 0, there exist unique integers q and r
; which satisfy  i = jq + r,  0 <= r < |j|.
;
; The functions IQUOTIENT and IREMAINDER are intended to compute q and r.
; Therefore, to be satisfied that we have the right definitions, we must
; prove the above theorem.
```

THEOREM: division-theorem-part1
$\text{integerp}\,(i) \rightarrow (\text{iplus}\,(\text{iremainder}\,(i,\ j),\ \text{itimes}\,(j,\ \text{iquotient}\,(i,\ j))) = i)$

THEOREM: division-theorem-part2
$(\text{integerp}\,(j) \land (j \neq \texttt{0})) \rightarrow (\neg\ \text{ilessp}\,(\text{iremainder}\,(i,\ j),\ \texttt{0}))$

THEOREM: division-theorem-part3
$(\text{integerp}\,(j) \land (j \neq \texttt{0})) \rightarrow \text{ilessp}\,(\text{iremainder}\,(i,\ j),\ \text{iabs}\,(j))$

THEOREM: division-theorem
$(\text{integerp}\,(i) \land \text{integerp}\,(j) \land (j \neq \texttt{0}))$
$\rightarrow \quad ((\text{iplus}\,(\text{iremainder}\,(i,\ j),\ \text{itimes}\,(j,\ \text{iquotient}\,(i,\ j))) = i)$
$\quad\quad \land \quad (\neg\ \text{ilessp}\,(\text{iremainder}\,(i,\ j),\ \texttt{0}))$
$\quad\quad \land \quad \text{ilessp}\,(\text{iremainder}\,(i,\ j),\ \text{iabs}\,(j)))$

THEOREM: quotient-difference-lessp-arg2

$$
\begin{array}{l}
(((a \ \mathbf{mod}\ c) = 0) \land (b < c)) \\
\to \quad (((a - b) \div c) \\
\qquad = \ \ \mathbf{if}\ b \simeq 0\ \ \mathbf{then}\ a \div c \\
\qquad\qquad \mathbf{elseif}\ b < a\ \ \mathbf{then}\ (a \div c) - (1 + (b \div c)) \\
\qquad\qquad \mathbf{else}\ 0\ \mathbf{endif})
\end{array}
$$

THEOREM: iquotient-iremainder-uniqueness

$$
\begin{array}{l}
(\text{integerp}\,(i) \\
\ \land \quad \text{integerp}\,(j) \\
\ \land \quad \text{integerp}\,(r) \\
\ \land \quad \text{integerp}\,(q) \\
\ \land \quad (j \neq 0) \\
\ \land \quad (i = \text{iplus}\,(r,\ \text{itimes}\,(j,\ q))) \\
\ \land \quad (\neg\ \text{ilessp}\,(r,\ 0)) \\
\ \land \quad \text{ilessp}\,(r,\ \text{iabs}\,(j))) \\
\to \quad ((r = \text{iremainder}\,(i,\ j)) \land (q = \text{iquotient}\,(i,\ j)))
\end{array}
$$

```
; It turns out that in computer arithmetic, notions of division other than that
; given by the division theorem are used. Two in particular, called
; "truncate towards negative infinity" and "truncate towards zero" are common.
; We present their definitions here.

; Division Theorem (truncate towards negative infinity variant):
;
; For all integers i,j, j not 0, there exist unique integers q and r
; which satisfy
;               i = jq + r,  0 <= r <  j  (j > 0)
;                            j <  r <= 0  (j < 0)
;
; In this version the integer quotient of two integers is the integer floor
; of the real quotient of the integers. The remainder has the sign of the
; divisor. The functions IDIV and IMOD are intended to compute q and r.
; Therefore, to be satisfied that we have the right definitions, we must
; prove the above theorem.
```

THEOREM: division-theorem-for-truncate-to-neginf-part1

$\text{integerp}\,(i) \to (\text{iplus}\,(\text{imod}\,(i,\ j),\ \text{itimes}\,(j,\ \text{idiv}\,(i,\ j))) = i)$

THEOREM: division-theorem-for-truncate-to-neginf-part2

$\text{ilessp}\,(0,\ j) \to ((\neg\ \text{ilessp}\,(\text{imod}\,(i,\ j),\ 0)) \land \text{ilessp}\,(\text{imod}\,(i,\ j),\ j))$

THEOREM: division-theorem-for-truncate-to-neginf-part3

$$
\begin{array}{l}
(\text{integerp}\,(j) \land \text{ilessp}\,(j,\ 0)) \\
\to \quad ((\neg\ \text{ilessp}\,(0,\ \text{imod}\,(i,\ j))) \land \text{ilessp}\,(j,\ \text{imod}\,(i,\ j)))
\end{array}
$$

THEOREM: division-theorem-for-truncate-to-neginf

$$\begin{array}{l}
(\text{integerp}\,(i) \land \text{integerp}\,(j) \land (j \neq 0)) \\
\to \quad ((\text{iplus}\,(\text{imod}\,(i,\, j),\, \text{itimes}\,(j,\, \text{idiv}\,(i,\, j))) = i) \\
\qquad \land \quad \textbf{if}\ \text{ilessp}\,(0,\, j) \\
\qquad\qquad \textbf{then}\ (\neg\ \text{ilessp}\,(\text{imod}\,(i,\, j),\, 0)) \land \text{ilessp}\,(\text{imod}\,(i,\, j),\, j) \\
\qquad\qquad \textbf{else}\ (\neg\ \text{ilessp}\,(0,\, \text{imod}\,(i,\, j))) \\
\qquad\qquad\qquad \land \quad \text{ilessp}\,(j,\, \text{imod}\,(i,\, j))\ \textbf{endif})
\end{array}$$

THEOREM: idiv-imod-uniqueness

$$\begin{array}{l}
(\text{integerp}\,(i) \\
\ \land \quad \text{integerp}\,(j) \\
\ \land \quad \text{integerp}\,(r) \\
\ \land \quad \text{integerp}\,(q) \\
\ \land \quad (j \neq 0) \\
\ \land \quad (i = \text{iplus}\,(r,\, \text{itimes}\,(j,\, q))) \\
\ \land \quad \textbf{if}\ \text{ilessp}\,(0,\, j)\ \ \textbf{then}\ (\neg\ \text{ilessp}\,(r,\, 0)) \land \text{ilessp}\,(r,\, j) \\
\qquad\ \ \textbf{else}\ (\neg\ \text{ilessp}\,(0,\, r)) \land \text{ilessp}\,(j,\, r)\ \textbf{endif}) \\
\to \quad ((r = \text{imod}\,(i,\, j)) \land (q = \text{idiv}\,(i,\, j)))
\end{array}$$

```
; Division Theorem (truncate towards zero variant):
;
; For all integers i,j, j not 0, there exist unique integers q and r
; which satisfy
;                i = jq + r,      0 <=  r  <  |j|  (i => 0)
;                          -|j| <   r  <= 0    (i < 0)
;
; In this version (iquo, irem), the integer quotient of two integers is the integer floor
; of the real quotient of the integers, if the real quotient is positive. If the
; real quotient is negative, the integer quotient is the integer ceiling of the
; real quotient. The remainder has the sign of the dividend. The functions IQUO
; and IREM are intended to compute q and r. Therefore, to be satisfied that we
; have the right definitions, we must prove the above theorem.
```

THEOREM: division-theorem-for-truncate-to-zero-part1

$$\text{integerp}\,(i) \to (\text{iplus}\,(\text{irem}\,(i,\, j),\, \text{itimes}\,(j,\, \text{iquo}\,(i,\, j))) = i)$$

THEOREM: division-theorem-for-truncate-to-zero-part2

$$\begin{array}{l}
(\text{integerp}\,(i) \land \text{integerp}\,(j) \land (j \neq 0) \land (\neg\ \text{ilessp}\,(i,\, 0))) \\
\to \quad ((\neg\ \text{ilessp}\,(\text{irem}\,(i,\, j),\, 0)) \land \text{ilessp}\,(\text{irem}\,(i,\, j),\, \text{iabs}\,(j)))
\end{array}$$

THEOREM: division-theorem-for-truncate-to-zero-part3

$$\begin{array}{l}
(\text{integerp}\,(i) \land \text{integerp}\,(j) \land (j \neq 0) \land \text{ilessp}\,(i,\, 0)) \\
\to \quad ((\neg\ \text{ilessp}\,(0,\, \text{irem}\,(i,\, j))) \land \text{ilessp}\,(\text{ineg}\,(\text{iabs}\,(j)),\, \text{irem}\,(i,\, j)))
\end{array}$$

THEOREM: division-theorem-for-truncate-to-zero

```
 (integerp (i) ∧ integerp (j) ∧ (j ≠ 0))
 →    ((iplus (irem (i, j), itimes (j, iquo (i, j))) = i)
        ∧   if ¬ ilessp (i, 0)
            then (¬ ilessp (irem (i, j), 0))
                   ∧   ilessp (irem (i, j), iabs (j))
            else (¬ ilessp (0, irem (i, j)))
                  ∧   ilessp (ineg (iabs (j)), irem (i, j)) endif)
```

THEOREM: iquo-irem-uniqueness

```
 (integerp (i)
  ∧   integerp (j)
  ∧   integerp (r)
  ∧   integerp (q)
  ∧   (j ≠ 0)
  ∧   (i = iplus (r, itimes (j, q)))
  ∧   if ¬ ilessp (i, 0) then (¬ ilessp (r, 0)) ∧ ilessp (r, iabs (j))
      else (¬ ilessp (0, r)) ∧ ilessp (ineg (iabs (j)), r) endif)
 →    ((r = irem (i, j)) ∧ (q = iquo (i, j)))
```

```
; ---------- Multiplication Facts
```

THEOREM: itimes-ineg-1
itimes (ineg (x), y) = ineg (itimes (x, y))

THEOREM: itimes-ineg-2
itimes (x, ineg (y)) = ineg (itimes (x, y))

THEOREM: itimes-cancellation-1
(itimes (a, b) = itimes (a, c))
= ((fix-int (a) = 0) ∨ (fix-int (b) = fix-int (c)))

THEOREM: itimes-cancellation-2
(itimes (b, a) = itimes (c, a))
= ((fix-int (a) = 0) ∨ (fix-int (b) = fix-int (c)))

THEOREM: itimes-cancellation-3
(itimes (a, b) = itimes (c, a))
= ((fix-int (a) = 0) ∨ (fix-int (b) = fix-int (c)))

```
; ---------- Division Facts
```

THEOREM: integerp-iquotient
integerp (iquotient (i, j))

Theorem: integerp-iremainder
integerp (iremainder $(i,\ j)$)

Theorem: integerp-idiv
integerp (idiv $(i,\ j)$)

Theorem: integerp-imod
integerp (imod $(i,\ j)$)

Theorem: integerp-iquo
integerp (iquo $(i,\ j)$)

Theorem: integerp-irem
integerp (irem $(i,\ j)$)

Theorem: iquotient-fix-int1
iquotient (fix-int (i), j) = iquotient $(i,\ j)$

Theorem: iquotient-fix-int2
iquotient (i, fix-int (j)) = iquotient $(i,\ j)$

Theorem: iremainder-fix-int1
iremainder (fix-int (i), j) = iremainder $(i,\ j)$

Theorem: iremainder-fix-int2
iremainder (i, fix-int (j)) = iremainder $(i,\ j)$

Theorem: idiv-fix-int1
idiv (fix-int (i), j) = idiv $(i,\ j)$

Theorem: idiv-fix-int2
idiv (i, fix-int (j)) = idiv $(i,\ j)$

Theorem: imod-fix-int1
imod (fix-int (i), j) = imod $(i,\ j)$

Theorem: imod-fix-int2
imod (i, fix-int (j)) = imod $(i,\ j)$

Theorem: iquo-fix-int1
iquo (fix-int (i), j) = iquo $(i,\ j)$

Theorem: iquo-fix-int2
iquo (i, fix-int (j)) = iquo $(i,\ j)$

Theorem: irem-fix-int1
irem (fix-int (i), j) = irem $(i,\ j)$

THEOREM: irem-fix-int2
irem $(i,$ fix-int $(j))$ = irem (i, j)

THEOREM: fix-int-iquotient
fix-int (iquotient $(i, j))$ = iquotient (i, j)

THEOREM: fix-int-iremainder
fix-int (iremainder $(i, j))$ = iremainder (i, j)

THEOREM: fix-int-idiv
fix-int (idiv $(i, j))$ = idiv (i, j)

THEOREM: fix-int-imod
fix-int (imod $(i, j))$ = imod (i, j)

THEOREM: fix-int-iquo
fix-int (iquo $(i, j))$ = iquo (i, j)

THEOREM: fix-int-irem
fix-int (irem $(i, j))$ = irem (i, j)

EVENT: Disable iquotient.

EVENT: Disable iremainder.

EVENT: Disable idiv.

EVENT: Disable imod.

EVENT: Disable iquo.

EVENT: Disable irem.

```
; ---------- Meta lemma for itimes cancellation

;; I tried to adapt this somewhat from corresponding meta lemmas in
;; naturals library, but it seemed to get hairy.  So instead I'll try
;; to parallel the development I gave for IPLUS.  I'll be lazier here
;; about efficiency, so I'll use a completely analogous definition of
;; itimes-tree.  Notice that I've avoided the IZEROP-TREE approach
;; from the naturals version, in that I simply create the appropriate
;; common fringe into a product and say that this product is non-zero
;; when dividing both sides by it.  It can then be up to the user whether
;; or not to enable the (meta or rewrite) rule that says that izerop of a product reduces
;; to the disjunction of izerop of the factors.

#| The function below has no AND or OR, for efficiency
(defn itimes-fringe (x)
  (if (and (listp x)
   (equal (car x)
  (quote itimes)))
      (append (itimes-fringe (cadr x))
      (itimes-fringe (caddr x)))
      (cons x nil)))
|#
```

DEFINITION:
itimes-fringe (x)

```
=  if listp (x)
   then if car (x) = 'itimes
        then append (itimes-fringe (cadr (x)), itimes-fringe (caddr (x)))
        else list (x) endif
   else list (x) endif
```

DEFINITION:
itimes-tree-rec (l)

```
=  if cdr (l) ≃ nil then car (l)
   else list ('itimes, car (l), itimes-tree-rec (cdr (l))) endif
```

DEFINITION:
itimes-tree (l)

```
=  if listp (l)
   then if listp (cdr (l)) then itimes-tree-rec (l)
        else list ('fix-int, car (l)) endif
   else ''1 endif
```

DEFINITION:
itimes-list (x)

```
=  if listp (x) then itimes (car (x), itimes-list (cdr (x)))
   else 1 endif
```

THEOREM: integerp-itimes-list
integerp (itimes-list (x))

THEOREM: eval\$-itimes-tree-rec
listp (x)

$\rightarrow$ (eval\$ (**t**, itimes-tree-rec (x), a)
= **if** listp (cdr (x)) **then** itimes-list (eval\$ (`'list`, x, a))
else eval\$ (**t**, car (x), a) **endif**)

```
;; The following allows us to pretty much ignore itimes-tree forever.  (Notice
;; that it is disabled immediately below.)
```

THEOREM: eval\$-itimes-tree
eval\$ (**t**, itimes-tree (x), a) = itimes-list (eval\$ (`'list`, x, a))

EVENT: Disable itimes-tree.

```
;; because we want to use EVAL$-ITIMES-TREE for now
```

DEFINITION:
make-cancel-itimes-equality (x, y, *in-both*)
= list (`'if`,
list (`'equal`, itimes-tree (*in-both*), `''0`),
list (`'quote`, **t**),
list (`'equal`,
itimes-tree (bagdiff (x, *in-both*)),
itimes-tree (bagdiff (y, *in-both*))))

```
#| The function below has no AND or OR, for efficiency
(defn cancel-itimes (x)
  (if (and (listp x)
   (equal (car x) (quote equal)))
      (if (and (listp (cadr x))
       (equal (caadr x) (quote itimes))
       (listp (caddr x))
       (equal (caaddr x) (quote itimes)))
  (if (listp (bagint (itimes-fringe (cadr x)) (itimes-fringe (caddr x))))
      (make-cancel-itimes-equality (itimes-fringe (cadr x))
   (itimes-fringe (caddr x))
   (bagint (itimes-fringe (cadr x)) (itimes-fringe (caddr x))))
    x)
(if (and (listp (cadr x))
 (equal (caadr x) (quote itimes)))
    ;; We don't want to introduce the IF below unless something
    ;; is "gained", or else we may get into an infinite rewriting loop.
    (if (member (caddr x) (itimes-fringe (cadr x)))
(list (quote if)
      (list (quote integerp) (caddr x))
      (make-cancel-itimes-equality (itimes-fringe (cadr x))
   (list (caddr x))
   (list (caddr x)))
      (list (quote quote) f))
      x)
  (if (and (listp (caddr x))
   (equal (caaddr x) (quote itimes)))
      (if (member (cadr x) (itimes-fringe (caddr x)))
  (list (quote if)
(list (quote integerp) (cadr x))
(make-cancel-itimes-equality (list (cadr x))
     (itimes-fringe (caddr x))
     (list (cadr x)))
(list (quote quote) f))
x)
    x)))
    x))
|#
```

Definition:

```
cancel-itimes $(x)$
=   if listp $(x)$
    then if car $(x)$ = 'equal
         then if listp (cadr $(x)$)
              then if caadr $(x)$ = 'itimes
                   then if listp (caddr $(x)$)
                        then if caaddr $(x)$ = 'itimes
                             then if listp (bagint (itimes-fringe (cadr $(x)$),
                                                    itimes-fringe (caddr $(x)$)))
                                  then make-cancel-itimes-equality (itimes-fringe (cadr $(x)$),
                                                                    itimes-fringe (caddr $(x)$),
                                                                    bagint (itimes-fringe (cadr $(x)$),
                                                                            itimes-fringe (caddr $(x)$)))
                                  else $x$ endif
                             elseif caddr $(x)$ $\in$ itimes-fringe (cadr $(x)$)
                             then list ('if,
                                        list ('integerp, caddr $(x)$),
                                        make-cancel-itimes-equality (itimes-fringe (cadr $(x)$),
                                                                     list (caddr $(x)$),
                                                                     list (caddr $(x)$)),
                                        list ('quote, f))
                             else $x$ endif
```

```
                  elseif caddr (x) ∈ itimes-fringe (cadr (x))
                  then list ('if,
                             list ('integerp, caddr (x)),
                             make-cancel-itimes-equality (itimes-fringe (cadr (x)),
                                                         list (caddr (x)),
                                                         list (caddr (x))),
                             list ('quote, f))
                  else x endif
             elseif listp (caddr (x))
             then if caaddr (x) = 'itimes
                  then if cadr (x) ∈ itimes-fringe (caddr (x))
                       then list ('if,
                                  list ('integerp, cadr (x)),
                                  make-cancel-itimes-equality (list (cadr (x)),
                                                              itimes-fringe (caddr (x)),
                                                              list (cadr (x))),
                                  list ('quote, f))
                       else x endif
                  else x endif
             else x endif
        elseif listp (caddr (x))
        then if caaddr (x) = 'itimes
             then if cadr (x) ∈ itimes-fringe (caddr (x))
                  then list ('if,
                             list ('integerp, cadr (x)),
                             make-cancel-itimes-equality (list (cadr (x)),
                                                         itimes-fringe (caddr (x)),
                                                         list (cadr (x))),
                             list ('quote, f))
                  else x endif
             else x endif
        else x endif
   else x endif
else x endif
```

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

THEOREM: itimes-list-eval\$-fringe
itimes-list (eval\$ ('list, itimes-fringe (x), a)) = fix-int (eval\$ (**t**, x, a))

THEOREM: integerp-eval\$-itimes
(car (x) = 'itimes) $\rightarrow$ integerp (eval\$ (**t**, x, a))

THEOREM: not-integerp-implies-not-equal-itimes
($\neg$ integerp (a)) $\rightarrow$ ((a = itimes $(b,\ c)$) = **f**)

THEOREM: itimes-list-eval$-delete
$(z \in y)$
$\rightarrow$ (itimes-list (eval\$ (`'list`, y, a))
  = itimes (eval\$ (**t**, z, a), itimes-list (eval\$ (`'list`, delete (z, y), a))))

THEOREM: itimes-list-bagdiff
subbagp (x, y)
$\rightarrow$ (itimes-list (eval\$ (`'list`, y, a))
  = itimes (itimes-list (eval\$ (`'list`, bagdiff (y, x), a)),
    itimes-list (eval\$ (`'list`, x, a))))

THEOREM: equal-itimes-list-eval$-list-delete
$((c \in y) \wedge$ (fix-int (eval\$ (**t**, c, a)) $\neq$ 0))
$\rightarrow$ ((x = itimes-list (eval\$ (`'list`, delete (c, y), a)))
  = (integerp (x)
   $\wedge$ (itimes (x, eval\$ (**t**, c, a))
    = itimes-list (eval\$ (`'list`, y, a)))))

EVENT: Disable itimes-list-eval$-delete.

```
;; I had trouble with the clausifier (thanks, J, for pointing that out
;; as the source of my trouble) in the proof of the meta lemma -- it's
;; getting rid of a case split.  So, I'll proceed by reducing
;; cancel-itimes in each case; see lemma eval$-make-cancel-itimes-equality
;; (and its -1 and -2 versions).
```

THEOREM: member-append
$(a \in$ append $(x, y)) = ((a \in x) \vee (a \in y))$

THEOREM: member-izerop-itimes-fringe
$((z \in$ itimes-fringe $(x)) \wedge$ (fix-int (eval\$ (**t**, z, a)) = 0))
$\rightarrow$ (fix-int (eval\$ (**t**, x, a)) = 0)

THEOREM: correctness-of-cancel-itimes-hack-1
$((w \in$ itimes-fringe (cons (`'itimes`, *x1*)))
 $\wedge$ (fix-int (eval\$ (**t**, w, a)) = 0)
 $\wedge$ (fix-int (eval\$ (**t**, car (*x1*), a)) $\neq$ 0))
$\rightarrow$ (fix-int (eval\$ (**t**, cadr (*x1*), a)) = 0)

EVENT: Enable eval$-equal.

THEOREM: eval$-make-cancel-itimes-equality
eval\$ (**t**, make-cancel-itimes-equality (x, y, *in-both*), a)

$=$ **if** eval\$ (**t**, list (`'equal`, itimes-tree (*in-both*), `''0`), a) **then t**
else itimes-list (eval\$ (`'list`, bagdiff (x, *in-both*), a))
$=$ itimes-list (eval\$ (`'list`, bagdiff (y, *in-both*), a)) **endif**

EVENT: Disable make-cancel-itimes-equality.

```
;; Here's a special case that I hope helps with the clausifier problem.
;; The lemma above seems necessary for its proof.
```

THEOREM: eval\$-make-cancel-itimes-equality-1
eval\$ (**t**, make-cancel-itimes-equality (list (x), y, list (x)), a)
$=$ **if** fix-int (eval\$ (**t**, x, a)) $=$ `0` **then t**
else `1` $=$ itimes-list (eval\$ (`'list`, delete (x, y), a)) **endif**

THEOREM: equal-fix-int
(fix-int (x) $= x$) $=$ integerp (x)

```
;; Here's another special case that I hope helps with the clausifier problem.
```

THEOREM: eval\$-make-cancel-itimes-equality-2
eval\$ (**t**, make-cancel-itimes-equality (x, list (y), list (y)), a)
$=$ **if** fix-int (eval\$ (**t**, y, a)) $=$ `0` **then t**
else `1` $=$ itimes-list (eval\$ (`'list`, delete (y, x), a)) **endif**

THEOREM: eval\$-equal-itimes-tree-itimes-fringe-0
(eval\$ (**t**, list (`'equal`, itimes-tree (itimes-fringe (x)), `''0`), a)
$\land$ (car (x) $=$ `'itimes`))
$\rightarrow$ (eval\$ (**t**, x, a) $=$ `0`)

THEOREM: izerop-eval-of-member-implies-itimes-list-0
(($z \in y$) $\land$ (fix-int (eval\$ (**t**, z, a)) $=$ `0`))
$\rightarrow$ (itimes-list (eval\$ (`'list`, y, a)) $=$ `0`)

```
#| The function below has no AND or OR, for efficiency
(defn subsetp (x y)
  (if (nlistp x)
      t
    (and (member (car x) y)
 (subsetp (cdr x) y))))
|#
```

DEFINITION:
subsetp $(x,\ y)$

```
=   if $x \simeq$ nil then t
    elseif car $(x) \in y$ then subsetp (cdr $(x),\ y)$
    else f endif
```

THEOREM: subsetp-implies-itimes-list-eval$-equals-0
(subsetp $(x,\ y)$ $\wedge$ (itimes-list (eval\$ (`'list`, $x,\ a$)) = `0`))
$\rightarrow$ (itimes-list (eval\$ (`'list`, $y,\ a$)) = `0`)

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

THEOREM: equal-0-itimes-list-eval$-bagint-1
(itimes-list (eval\$ (`'list`, bagint $(x,\ y),\ a$)) = `0`)
$\rightarrow$ (itimes-list (eval\$ (`'list`, $x,\ a$)) = `0`)

THEOREM: equal-0-itimes-list-eval$-bagint-2
(itimes-list (eval\$ (`'list`, bagint $(x,\ y),\ a$)) = `0`)
$\rightarrow$ (itimes-list (eval\$ (`'list`, $y,\ a$)) = `0`)

THEOREM: correctness-of-cancel-itimes-hack-2

```
(listp (u)
 ∧  (car (u) = 'itimes)
 ∧  listp (v)
 ∧  (car (v) = 'itimes)
 ∧  (eval$ (t, u, a) ≠ eval$ (t, v, a)))
→  (itimes-list (eval$ ('list,
                        bagdiff (itimes-fringe (u),
                                 bagint (itimes-fringe (u), itimes-fringe (v))),
                        a))
       ≠  itimes-list (eval$ ('list,
                               bagdiff (itimes-fringe (v),
                                        bagint (itimes-fringe (u),
                                                itimes-fringe (v))),
                               a)))
```

THEOREM: correctness-of-cancel-itimes-hack-3-lemma
((u = itimes $(a,\ b)$) $\wedge$ (fix-int $(a) \neq$ `0`))
$\rightarrow$ ((u = itimes $(a,\ c)$) = (fix-int (b) = fix-int (c)))

THEOREM: correctness-of-cancel-itimes-hack-3

```
(listp (u)
 ∧  (car (u) = 'itimes)
 ∧  listp (v)
 ∧   (car (v) = 'itimes)
 ∧   (eval$ (t, u, a) = eval$ (t, v, a))
 ∧   (¬ eval$ (t,
               list ('equal,
                     itimes-tree (bagint (itimes-fringe (u), itimes-fringe (v))),
                     ''0),
               a)))
→   ((itimes-list (eval$ ('list,
                          bagdiff (itimes-fringe (u),
                                   bagint (itimes-fringe (u), itimes-fringe (v))),
                          a))
       =   itimes-list (eval$ ('list,
                               bagdiff (itimes-fringe (v),
                                        bagint (itimes-fringe (u),
                                                itimes-fringe (v))),
                               a)))
      =   t)
```

EVENT: Disable correctness-of-cancel-itimes-hack-3-lemma.

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

```
; ---------- Meta lemma for itimes cancellation on ilessp terms

;; I'll try to keep this similar to the approach for equalities above,
;; modified as in the iplus case (i.e. no fix-int is necessary).

;; EVAL$-EQUAL is currently enabled, but that's OK.
```

DEFINITION:
itimes-tree-no-fix-int (l)
= **if** listp (l) **then** itimes-tree-rec (l)
  else ''1 **endif**

```
;; The following allows us to pretty much ignore
;; itimes-tree-no-fix-int forever.  (Notice that it is disabled
;; immediately below.)
```

THEOREM: eval$-itimes-tree-no-fix-int-1
ilessp (eval$ (**t**, itimes-tree-no-fix-int (x), a), y)
= ilessp (eval$ (**t**, itimes-tree (x), a), y)

THEOREM: eval$-itimes-tree-no-fix-int-2
ilessp (y, eval$ (**t**, itimes-tree-no-fix-int (x), a))
= ilessp (y, eval$ (**t**, itimes-tree (x), a))

EVENT: Disable itimes-tree-no-fix-int.

```
;; We want to use EVAL$-ITIMES-TREE, and ITIMES-TREE is still disabled
;; so we're in good shape.
```

DEFINITION:
```
make-cancel-itimes-inequality (x, y, in-both)
=   list ('if,
          list ('ilessp, itimes-tree-no-fix-int (in-both), ''0),
          list ('ilessp,
                itimes-tree-no-fix-int (bagdiff (y, in-both)),
                itimes-tree-no-fix-int (bagdiff (x, in-both))),
          list ('if,
                list ('ilessp, ''0, itimes-tree-no-fix-int (in-both)),
                list ('ilessp,
                      itimes-tree-no-fix-int (bagdiff (x, in-both)),
                      itimes-tree-no-fix-int (bagdiff (y, in-both))),
                '(false)))
```

```
#| The function below has no AND or OR, for efficiency
(defn cancel-itimes-ilessp (x)
  (if (and (listp x)
   (equal (car x) (quote ilessp))
   (listp (bagint (itimes-fringe (cadr x)) (itimes-fringe (caddr x)))))
      (make-cancel-itimes-inequality (itimes-fringe (cadr x))
     (itimes-fringe (caddr x))
     (bagint (itimes-fringe (cadr x))
     (itimes-fringe (caddr x))))
    x))
|#
```

DEFINITION:
```
cancel-itimes-ilessp (x)
=   if listp (x)
    then if car (x) = 'ilessp
         then if listp (bagint (itimes-fringe (cadr (x)),
                                itimes-fringe (caddr (x))))
              then make-cancel-itimes-inequality (itimes-fringe (cadr (x)),
                                        itimes-fringe (caddr (x)),
                                        bagint (itimes-fringe (cadr (x)),
                                                itimes-fringe (caddr (x))))
             else x endif
         else x endif
     else x endif
```

THEOREM: eval$-make-cancel-itimes-inequality

```
 eval$ (t, make-cancel-itimes-inequality (x, y, in-both), a)
 =   if eval$ (t, list ('ilessp, itimes-tree-no-fix-int (in-both), ''0), a)
     then ilessp (eval$ (t, itimes-tree-no-fix-int (bagdiff (y, in-both)), a),
                  eval$ (t, itimes-tree-no-fix-int (bagdiff (x, in-both)), a))
     elseif eval$ (t, list ('ilessp, ''0, itimes-tree-no-fix-int (in-both)), a)
     then ilessp (eval$ (t, itimes-tree-no-fix-int (bagdiff (x, in-both)), a),
                  eval$ (t, itimes-tree-no-fix-int (bagdiff (y, in-both)), a))
     else f endif
```

EVENT: Disable make-cancel-itimes-inequality.

THEOREM: listp-bagint-with-singleton-implies-member

listp (bagint (y, list (z))) $\rightarrow$ ($z \in y$)

THEOREM: itimes-list-eval$-list-0

(0 $\in x$) $\rightarrow$ (itimes-list (eval$ ('list, x, a)) = 0)

THEOREM: ilessp-itimes-right-positive

ilessp (0, x) $\rightarrow$ (ilessp (y, z) = ilessp (itimes (y, x), itimes (z, x)))

THEOREM: correctness-of-cancel-itimes-ilessp-hack-1

```
 (subbagp (bag, x)
  ∧   subbagp (bag, y)
  ∧   ilessp (0, itimes-list (eval$ ('list, bag, a))))
 →    (ilessp (itimes-list (eval$ ('list, bagdiff (x, bag), a)),
               itimes-list (eval$ ('list, bagdiff (y, bag), a)))
        =   ilessp (itimes-list (eval$ ('list, x, a)),
                    itimes-list (eval$ ('list, y, a))))
```

THEOREM: listp-bagint-with-singleton-member

listp (bagint (y, list (z))) = ($z \in y$)

THEOREM: correctness-of-cancel-itimes-ilessp-hack-2-lemma

(0 $\in$ itimes-fringe (w)) $\rightarrow$ (eval$ (**t**, w, a) = 0)

THEOREM: correctness-of-cancel-itimes-ilessp-hack-2

(0 $\in$ itimes-fringe (w)) $\rightarrow$ ($\neg$ ilessp (eval$ (**t**, w, a), 0))

EVENT: Disable correctness-of-cancel-itimes-ilessp-hack-2-lemma.

```
;;; Now hack-3 and hack-4 below are all that's left to prove before the
;;; main result.
```

THEOREM: ilessp-trichotomy
$(\neg\ \text{ilessp}\,(x,\ y)) \rightarrow (\text{ilessp}\,(y,\ x) = (\text{fix-int}\,(x) \neq \text{fix-int}\,(y)))$

THEOREM: correctness-of-cancel-itimes-ilessp-hack-3-lemma-1
$((\texttt{0} = \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ bag,\ a))) \wedge \text{subsetp}\,(bag,\ z))$
$\rightarrow \quad (\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ z,\ a)) = \texttt{0})$

THEOREM: correctness-of-cancel-itimes-ilessp-hack-3-lemma-2
$((\texttt{0} = \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ bag,\ a))) \wedge \text{subsetp}\,(bag,\ \text{itimes-fringe}\,(x)))$
$\rightarrow \quad (\text{fix-int}\,(\text{eval\$}\,(\mathbf{t},\ x,\ a)) = \texttt{0})$

THEOREM: same-fix-int-implies-not-ilessp
$(\text{fix-int}\,(x) = \text{fix-int}\,(y)) \rightarrow (\neg\ \text{ilessp}\,(x,\ y))$

THEOREM: correctness-of-cancel-itimes-ilessp-hack-3
$((\neg\ \text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ bag,\ a)),\ \texttt{0}))$
$\wedge \quad (\neg\ \text{ilessp}\,(\texttt{0},\ \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ bag,\ a))))$
$\wedge \quad \text{subbagp}\,(bag,\ \text{itimes-fringe}\,(w))$
$\wedge \quad \text{subbagp}\,(bag,\ \text{itimes-fringe}\,(v)))$
$\rightarrow \quad (\neg\ \text{ilessp}\,(\text{eval\$}\,(\mathbf{t},\ w,\ a),\ \text{eval\$}\,(\mathbf{t},\ v,\ a)))$

THEOREM: ilessp-itimes-right-negative
$\text{ilessp}\,(x,\ \texttt{0}) \rightarrow (\text{ilessp}\,(y,\ z) = \text{ilessp}\,(\text{itimes}\,(z,\ x),\ \text{itimes}\,(y,\ x)))$

THEOREM: correctness-of-cancel-itimes-ilessp-hack-4
$(\text{subbagp}\,(bag,\ x)$
$\wedge \quad \text{subbagp}\,(bag,\ y)$
$\wedge \quad \text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ bag,\ a)),\ \texttt{0}))$
$\rightarrow \quad (\text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \text{bagdiff}\,(x,\ bag),\ a)),$
$\qquad\qquad \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \text{bagdiff}\,(y,\ bag),\ a)))$
$\qquad = \quad \text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ y,\ a)),$
$\qquad\qquad\qquad \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ x,\ a))))$

EVENT: Disable ilessp-trichotomy.

EVENT: Disable same-fix-int-implies-not-ilessp.

THEOREM: correctness-of-cancel-itimes-ilessp
$\text{eval\$}\,(\mathbf{t},\ x,\ a) = \text{eval\$}\,(\mathbf{t},\ \text{cancel-itimes-ilessp}\,(x),\ a)$

```
;; I think that the following lemma is safe because it won't be
;; called at all during relieve-hyps.
```

THEOREM: ilessp-strict
$\text{ilessp}\,(x,\ y) \rightarrow (\neg\ \text{ilessp}\,(y,\ x))$

```
; ---------- Setting up the State ----------

;; I'll close by disabling (or enabling) those rules and definitions
;; whose status as left over from above isn't quite what I'd like.
;; I'm going to leave the eval$ rules on and eval$ off.
```

EVENT: Disable eval$-cancel-iplus.

EVENT: Disable eval$-iplus.

EVENT: Disable lessp-count-listp-cdr.

EVENT: Disable eval$-iplus-tree-rec.

EVENT: Disable eval$-iplus-tree.

```
;;(disable eval$-list-append) ;; Nice rule -- I'll keep it enabled
```

EVENT: Disable iplus-list-eval$-fringe.

EVENT: Disable eval$-iplus-list-bagdiff.

EVENT: Disable lessp-difference-plus-arg1.

EVENT: Disable lessp-difference-plus-arg1-commuted.

EVENT: Disable correctness-of-cancel-iplus-ilessp-lemma.

EVENT: Disable eval$-ilessp-iplus-tree-no-fix-int.

EVENT: Disable make-cancel-iplus-inequality-simplifier.

EVENT: Disable quotient-difference-lessp-arg2.

EVENT: Disable eval$-itimes-tree-rec.

EVENT: Disable eval$-itimes-tree.

EVENT: Disable itimes-list-eval$-fringe.

EVENT: Disable integerp-eval$-itimes.

EVENT: Disable itimes-list-bagdiff.

EVENT: Disable equal-itimes-list-eval$-list-delete.

EVENT: Disable member-izerop-itimes-fringe.

EVENT: Disable correctness-of-cancel-itimes-hack-1.

EVENT: Disable eval$-make-cancel-itimes-equality.

EVENT: Disable eval$-make-cancel-itimes-equality-1.

EVENT: Disable eval$-make-cancel-itimes-equality-2.

EVENT: Disable eval$-equal-itimes-tree-itimes-fringe-0.

EVENT: Disable izerop-eval-of-member-implies-itimes-list-0.

EVENT: Disable subsetp-implies-itimes-list-eval$-equals-0.

EVENT: Disable equal-0-itimes-list-eval$-bagint-1.

EVENT: Disable equal-0-itimes-list-eval$-bagint-2.

EVENT: Disable correctness-of-cancel-itimes-hack-2.

EVENT: Disable correctness-of-cancel-itimes-hack-3-lemma.

EVENT: Disable correctness-of-cancel-itimes-hack-3.

EVENT: Disable eval$-itimes-tree-no-fix-int-1.

EVENT: Disable eval$-itimes-tree-no-fix-int-2.

EVENT: Disable eval$-make-cancel-itimes-inequality.

EVENT: Disable listp-bagint-with-singleton-implies-member.

EVENT: Disable itimes-list-eval$-list-0.

EVENT: Disable correctness-of-cancel-itimes-ilessp-hack-1.

EVENT: Disable listp-bagint-with-singleton-member.

EVENT: Disable correctness-of-cancel-itimes-ilessp-hack-2.

EVENT: Disable correctness-of-cancel-itimes-ilessp-hack-3-lemma-1.

EVENT: Disable correctness-of-cancel-itimes-ilessp-hack-3-lemma-2.

EVENT: Disable correctness-of-cancel-itimes-ilessp-hack-3.

EVENT: Disable correctness-of-cancel-itimes-ilessp-hack-4.

```
;; The last one is a tough call, but I think it's OK.
;; (disable ilessp-strict)

;;;;;; ***** EXTRA META STUFF ***** ;;;;;;

;; The next goal is to improve itimes cancellation so that it looks
;; for common factors, and hence works on equations like
;;    x*y + x = x*z
;; and, for that matter,
;; a*x + -b*x = 0.

;; Rather than changing the existing cancel-itimes function, I'll
;; leave that one but disable its metalemma at the end.  Then if the
;; new version, which I'll call cancel-itimes-factors, is found to be
;; too slow, one can always disable its metalemma and re-enable the
;; metalemma for cancel-itimes.

;; Notice, by the way, that the existing cancel-itimes function is
;; useless for something like the following, since there's no special
;; treatment for INEG.  I'll remedy that in this version.

#|
(IMPLIES (AND (NOT (IZEROP X))
      (EQUAL (ITIMES A X) (INEG (ITIMES B X))))
 (EQUAL (FIX-INT A) (INEG B)))
|#
```

DEFINITION:
itimes-tree-ineg (l)

```
=   if listp (l)
    then if car (l) = list ('quote, -1)
         then if listp (cdr (l))  then list ('ineg, itimes-tree-rec (cdr (l)))
              else car (l) endif
         else itimes-tree-rec (l) endif
    else ''1 endif
```

DEFINITION:
itimes-factors (x)

```
=   if listp (x)
    then if car (x) = 'itimes
         then append (itimes-factors (cadr (x)), itimes-factors (caddr (x)))
         elseif car (x) = 'iplus
         then let bag1  be  itimes-factors (cadr (x)),
                  bag2  be  itimes-factors (caddr (x))
              in
              let inboth  be  bagint (bag1, bag2)
              in
              if listp (inboth)
              then cons (list ('iplus,
                              itimes-tree-ineg (bagdiff (bag1,
                                                         inboth)),
                              itimes-tree-ineg (bagdiff (bag2,
                                                  inboth))),
                              inboth)
              else list (x) endif endlet endlet
          elseif car (x) = 'ineg
          then cons (list ('quote, -1), itimes-factors (cadr (x)))
          else list (x) endif
    else list (x) endif
```

THEOREM: itimes–1
itimes (`-1`, x) = ineg (x)

```
;; I'll need the following lemma because it's simplest not to deal with
;; e.g. (equal x x), where x is a variable, in the meta thing.  I'll do
;; the one after it too, simply because I'm thinking of it now.
```

THEOREM: equal-ineg-ineg
(ineg (x) = ineg (y)) = (fix-int (x) = fix-int (y))

THEOREM: ilessp-ineg-ineg
ilessp (ineg (x), ineg (y)) = ilessp (y, x)

THEOREM: fix-int-eval$-itimes-tree-rec
```
listp (x)
→   (fix-int (eval$ (t, itimes-tree-rec (x), a))
      =   itimes-list (eval$ ('list, x, a)))
```

THEOREM: eval$-itimes-tree-ineg
fix-int (eval$ (**t**, itimes-tree-ineg (x), a)) = itimes-list (eval$ (`'list`, x, a))

```
;; Now I want the above lemma to apply, but it doesn't, so the
;; following three lemmas are used instead.
```

THEOREM: ineg-eval$-itimes-tree-ineg
```
ineg (eval$ (t, itimes-tree-ineg (x), a))
=   ineg (itimes-list (eval$ ('list, x, a)))
```

THEOREM: iplus-eval$-itimes-tree-ineg
```
(iplus (eval$ (t, itimes-tree-ineg (x), a), y)
  =   iplus (itimes-list (eval$ ('list, x, a)), y))
∧   (iplus (y, eval$ (t, itimes-tree-ineg (x), a))
      =   iplus (y, itimes-list (eval$ ('list, x, a))))
```

THEOREM: itimes-eval$-itimes-tree-ineg
```
(itimes (eval$ (t, itimes-tree-ineg (x), a), y)
  =   itimes (itimes-list (eval$ ('list, x, a)), y))
∧   (itimes (y, eval$ (t, itimes-tree-ineg (x), a))
      =   itimes (y, itimes-list (eval$ ('list, x, a))))
```

EVENT: Disable itimes-tree-ineg.

```
#| ****** The following definitions are for efficient execution of
   metafunctions.  They should probably be applied to all the metafunctions
   with fns arguments AND and OR.

(defmacro nqthm-macroexpand (defn &rest fns)
  `(nqthm-macroexpand-fn ',defn ',fns))

(defun nqthm-macroexpand-fn (defn fns)
  (iterate for fn in fns
   when (not (get fn 'sdefn))
   do (er soft (fn) |Sorry| |,| |but| |there| |is| |no| SDEFN
  |for| (!ppr fn (quote |.|))))
  (let (name args body)
    (match! defn (defn name args body))
    (let ((arity-alist (cons (cons name (length args)) arity-alist)))
      (list 'defn name args
    (untranslate (normalize-ifs
  (nqthm-macroexpand-term (translate body) fns)
  nil nil nil))))))

(defun nqthm-macroexpand-term (term fns)
  (cond
   ((or (variablep term) (fquotep term))
    term)
   ((member-eq (ffn-symb term) fns)
    (let ((sdefn (get (ffn-symb term) 'sdefn)))
      (sub-pair-var (cadr sdefn)
    (iterate for arg in (fargs term)
     collect (nqthm-macroexpand-term arg fns))
    (caddr sdefn))))
   (t (fcons-term (ffn-symb term)
  (iterate for arg in (fargs term)
   collect (nqthm-macroexpand-term arg fns))))))

|#

;; I "macroexpand" away the following below, so it's not really needed except
;; for the proof.  That is, I use it in the definition of cancel-itimes-factors,
;; but then get rid of it for cancel-itimes-factors-expanded, and although I
;; reason about the former, I USE the latter, for efficiency.
```

DEFINITION:

```
iplus-or-itimes-term ($x$)
=   if listp ($x$)
    then case on car ($x$):
            case = *iplus*
            then t
            case = *itimes*
             then t
            case = *ineg*
             then if listp (cadr ($x$))  then car (cadr ($x$)) = 'itimes
                   else f endif
            otherwise f endcase
    else f endif
```

Definition:

```
cancel-itimes-factors ($x$)
=   if listp ($x$) $\wedge$ (car ($x$) = 'equal)
    then let *bagint*  be  bagint (itimes-factors (cadr ($x$)),
                                   itimes-factors (caddr ($x$)))
            in
            let *new-equality*  be  make-cancel-itimes-equality (itimes-factors (cadr ($x$)),
                                                                 itimes-factors (caddr ($x$)),
                                                                 *bagint*)
            in
            if iplus-or-itimes-term (cadr ($x$))
            then if listp (*bagint*)
                  then if iplus-or-itimes-term (caddr ($x$))
                        then *new-equality*
                        else list ('if,
                                   list ('integerp,
                                         caddr ($x$)),
                                   *new-equality*,
                                   list ('quote, f)) endif
                  else $x$ endif
            elseif iplus-or-itimes-term (caddr ($x$))
            then if listp (*bagint*)
                  then list ('if,
                             list ('integerp, cadr ($x$)),
                             *new-equality*,
                             list ('quote, f))
                  else $x$ endif
            else $x$ endif endlet endlet
    else $x$ endif
```

```
;; The following was generated with the nqthm-macroexpand macro defined above.
;; The following was generated with the nqthm-macroexpand macro defined above.
(DEFN CANCEL-ITIMES-FACTORS-expanded (X)
  (IF (LISTP X)
      (IF (EQUAL (CAR X) 'EQUAL)
  (COND
   ((LISTP (CADR X))
    (CASE (CAR (CAR (CDR X)))
  (IPLUS (IF (LISTP (BAGINT (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))))
     (IF (LISTP (CADDR X))
 (CASE (CAR (CAR (CDR (CDR X))))
       (IPLUS
(MAKE-CANCEL-ITIMES-EQUALITY
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))
 (BAGINT (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X)))))
       (ITIMES
(MAKE-CANCEL-ITIMES-EQUALITY
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))
 (BAGINT (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X)))))
       (INEG
(IF (LISTP (CADADDR X))
    (IF (EQUAL (CAADADDR X) 'ITIMES)
(MAKE-CANCEL-ITIMES-EQUALITY
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))
 (BAGINT
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))))
(LIST 'IF
      (LIST 'INTEGERP (CADDR X))
      (MAKE-CANCEL-ITIMES-EQUALITY
       (ITIMES-FACTORS (CADR X))
       (ITIMES-FACTORS (CADDR X))
       (BAGINT
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))))
      (LIST 'QUOTE F)))
    (LIST 'IF
  (LIST 'INTEGERP (CADDR X))
  (MAKE-CANCEL-ITIMES-EQUALITY
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))
   (BAGINT
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))))
  (LIST 'QUOTE F))))
       (OTHERWISE
(LIST 'IF
      (LIST 'INTEGERP (CADDR X))
      (MAKE-CANCEL-ITIMES-EQUALITY
       (ITIMES-FACTORS (CADR X))
       (ITIMES-FACTORS (CADDR X))
       (BAGINT
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))))
      (LIST 'QUOTE F))))
 (LIST 'IF (LIST 'INTEGERP (CADDR X))
       (MAKE-CANCEL-ITIMES-EQUALITY
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))
(BAGINT
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))))
       (LIST 'QUOTE F)))
     X))
  (ITIMES (IF (LISTP (BAGINT (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))))
      (IF (LISTP (CADDR X))
  (CASE (CAR (CAR (CDR (CDR X))))
(IPLUS
 (MAKE-CANCEL-ITIMES-EQUALITY
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))
  (BAGINT
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X)))))
(ITIMES
 (MAKE-CANCEL-ITIMES-EQUALITY
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))
  (BAGINT
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X)))))
(INEG
 (IF (LISTP (CADADDR X))
     (IF (EQUAL (CAADADDR X) 'ITIMES)
 (MAKE-CANCEL-ITIMES-EQUALITY
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))
  (BAGINT
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))))
 (LIST 'IF
       (LIST 'INTEGERP (CADDR X))
       (MAKE-CANCEL-ITIMES-EQUALITY
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))
(BAGINT
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))))
       (LIST 'QUOTE F)))
     (LIST 'IF
   (LIST 'INTEGERP (CADDR X))
   (MAKE-CANCEL-ITIMES-EQUALITY
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))
    (BAGINT
     (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))))
   (LIST 'QUOTE F))))
(OTHERWISE
 (LIST 'IF
       (LIST 'INTEGERP (CADDR X))
       (MAKE-CANCEL-ITIMES-EQUALITY
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))
(BAGINT
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))))
       (LIST 'QUOTE F))))
  (LIST 'IF (LIST 'INTEGERP (CADDR X))
(MAKE-CANCEL-ITIMES-EQUALITY
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))
 (BAGINT (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))))
```

```
(LIST 'QUOTE F)))
      X))
  (INEG (COND
 ((LISTP (CADADR X))
  (COND
   ((EQUAL (CAADADR X) 'ITIMES)
    (IF (LISTP
 (BAGINT (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))))
(IF (LISTP (CADDR X))
                                    (CASE (CAR (CAR (CDR (CDR X))))
  (IPLUS
   (MAKE-CANCEL-ITIMES-EQUALITY
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))
    (BAGINT
     (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X)))))
  (ITIMES
   (MAKE-CANCEL-ITIMES-EQUALITY
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))
    (BAGINT
     (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X)))))
  (INEG
   (IF (LISTP (CADADDR X))
       (IF
(EQUAL (CAADADDR X) 'ITIMES)
(MAKE-CANCEL-ITIMES-EQUALITY
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))
 (BAGINT
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))))
(LIST 'IF
      (LIST 'INTEGERP (CADDR X))
      (MAKE-CANCEL-ITIMES-EQUALITY
       (ITIMES-FACTORS (CADR X))
       (ITIMES-FACTORS (CADDR X))
       (BAGINT
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))))
      (LIST 'QUOTE F)))
```

```
       (LIST 'IF
     (LIST 'INTEGERP (CADDR X))
     (MAKE-CANCEL-ITIMES-EQUALITY
      (ITIMES-FACTORS (CADR X))
      (ITIMES-FACTORS (CADDR X))
      (BAGINT
       (ITIMES-FACTORS (CADR X))
       (ITIMES-FACTORS (CADDR X))))
     (LIST 'QUOTE F))))
  (OTHERWISE
   (LIST 'IF
 (LIST 'INTEGERP (CADDR X))
 (MAKE-CANCEL-ITIMES-EQUALITY
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))
  (BAGINT
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))))
 (LIST 'QUOTE F))))
                                    (LIST 'IF
  (LIST 'INTEGERP (CADDR X))
  (MAKE-CANCEL-ITIMES-EQUALITY
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))
   (BAGINT (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))))
  (LIST 'QUOTE F)))
X))
   ((LISTP (CADDR X))
    (CASE (CAR (CAR (CDR (CDR X))))
  (IPLUS (IF
  (LISTP
   (BAGINT
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))))
  (LIST 'IF
(LIST 'INTEGERP (CADR X))
(MAKE-CANCEL-ITIMES-EQUALITY
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))
 (BAGINT
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))))
(LIST 'QUOTE F))
```

```
  X))
  (ITIMES (IF
   (LISTP
    (BAGINT
     (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))))
   (LIST 'IF
 (LIST 'INTEGERP (CADR X))
 (MAKE-CANCEL-ITIMES-EQUALITY
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))
  (BAGINT
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))))
 (LIST 'QUOTE F))
   X))
  (INEG (IF (LISTP (CADADDR X))
    (IF
     (EQUAL (CAADADDR X) 'ITIMES)
     (IF
      (LISTP
       (BAGINT
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))))
      (LIST 'IF
    (LIST 'INTEGERP (CADR X))
    (MAKE-CANCEL-ITIMES-EQUALITY
     (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))
     (BAGINT
      (ITIMES-FACTORS (CADR X))
      (ITIMES-FACTORS (CADDR X))))
    (LIST 'QUOTE F))
      X)
     X)
    X))
  (OTHERWISE X)))
   (T X)))
 ((LISTP (CADDR X))
  (CASE (CAR (CAR (CDR (CDR X))))
(IPLUS (IF
(LISTP
 (BAGINT
  (ITIMES-FACTORS (CADR X))
  (ITIMES-FACTORS (CADDR X))))
(LIST 'IF
      (LIST 'INTEGERP (CADR X))
      (MAKE-CANCEL-ITIMES-EQUALITY
       (ITIMES-FACTORS (CADR X))
       (ITIMES-FACTORS (CADDR X))
       (BAGINT
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))))
      (LIST 'QUOTE F))
X))
(ITIMES (IF
 (LISTP
  (BAGINT
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))))
 (LIST 'IF
       (LIST 'INTEGERP (CADR X))
       (MAKE-CANCEL-ITIMES-EQUALITY
(ITIMES-FACTORS (CADR X))
(ITIMES-FACTORS (CADDR X))
(BAGINT
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))))
       (LIST 'QUOTE F))
 X))
(INEG (IF (LISTP (CADADDR X))
  (IF (EQUAL (CAADADDR X) 'ITIMES)
      (IF
       (LISTP
(BAGINT
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR X))))
       (LIST 'IF
     (LIST 'INTEGERP (CADR X))
     (MAKE-CANCEL-ITIMES-EQUALITY
      (ITIMES-FACTORS (CADR X))
      (ITIMES-FACTORS (CADDR X))
      (BAGINT
       (ITIMES-FACTORS (CADR X))
       (ITIMES-FACTORS (CADDR X))))
     (LIST 'QUOTE F))
       X)
      X)
  X))
(OTHERWISE X)))
 (T X)))
  (OTHERWISE
   (IF (LISTP (CADDR X))
       (CASE (CAR (CAR (CDR (CDR X))))
                             (IPLUS (IF
                                     (LISTP
                                      (BAGINT (ITIMES-FACTORS (CADR X))
      (ITIMES-FACTORS (CADDR X))))
                                     (LIST 'IF
   (LIST 'INTEGERP (CADR X))
   (MAKE-CANCEL-ITIMES-EQUALITY
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))
    (BAGINT
     (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))))
   (LIST 'QUOTE F))
                                     X))
                             (ITIMES (IF
                                      (LISTP
                                       (BAGINT
                                        (ITIMES-FACTORS (CADR X))
                                        (ITIMES-FACTORS (CADDR X))))
                                      (LIST 'IF
    (LIST 'INTEGERP (CADR X))
    (MAKE-CANCEL-ITIMES-EQUALITY
     (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))
     (BAGINT
      (ITIMES-FACTORS (CADR X))
      (ITIMES-FACTORS (CADDR X))))
    (LIST 'QUOTE F))
                                      X))
                             (INEG (IF (LISTP (CADADDR X))
       (IF (EQUAL (CAADADDR X) 'ITIMES)
   (IF
    (LISTP
     (BAGINT
      (ITIMES-FACTORS (CADR X))
      (ITIMES-FACTORS (CADDR X))))
    (LIST 'IF
  (LIST 'INTEGERP (CADR X))
  (MAKE-CANCEL-ITIMES-EQUALITY
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))
   (BAGINT
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))))
  (LIST 'QUOTE F))
    X)
   X)
       X))
                             (OTHERWISE X))
       X))))
   ((LISTP (CADDR X))
    (CASE (CAR (CAR (CDR (CDR X))))
  (IPLUS (IF (LISTP (BAGINT (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))))
     (LIST 'IF (LIST 'INTEGERP (CADR X))
   (MAKE-CANCEL-ITIMES-EQUALITY
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))
    (BAGINT (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))))
   (LIST 'QUOTE F))
     X))
  (ITIMES (IF (LISTP (BAGINT (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))))
      (LIST 'IF (LIST 'INTEGERP (CADR X))
    (MAKE-CANCEL-ITIMES-EQUALITY
     (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))
     (BAGINT (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))))
    (LIST 'QUOTE F))
      X))
  (INEG (IF (LISTP (CADADDR X))
    (IF (EQUAL (CAADADDR X) 'ITIMES)
(IF (LISTP
     (BAGINT (ITIMES-FACTORS (CADR X))
     (ITIMES-FACTORS (CADDR X))))
    (LIST 'IF
  (LIST 'INTEGERP (CADR X))
  (MAKE-CANCEL-ITIMES-EQUALITY
   (ITIMES-FACTORS (CADR X))
   (ITIMES-FACTORS (CADDR X))
   (BAGINT
    (ITIMES-FACTORS (CADR X))
    (ITIMES-FACTORS (CADDR X))))
  (LIST 'QUOTE F))
    X)
X)
    X))
  (OTHERWISE X)))
   (T X))
  X)
      X))
```

Theorem: cancel-itimes-factors-expanded-cancel-itimes-factors
cancel-itimes-factors-expanded (x) = cancel-itimes-factors (x)

Event: Disable cancel-itimes-factors-expanded.

Event: Disable iplus-or-itimes-term.

Theorem: equal-itimes-list-eval\$-list-delete-new-1

```
(fix-int (eval$ (t, elt, a)) ≠ 0)
→   ((x = itimes-list (eval$ ('list, delete (elt, bag), a)))
      =   if elt ∈ bag
          then integerp (x)
                ∧   (itimes (x, eval$ (t, elt, a))
                      =   itimes-list (eval$ ('list, bag, a)))
          else x = itimes-list (eval$ ('list, bag, a)) endif)
```

Theorem: equal-itimes-list-eval\$-list-delete-new-2

```
(fix-int (eval$ (t, elt, a)) ≠ 0)
→   ((itimes-list (eval$ ('list, delete (elt, bag), a)) = x)
      =   if elt ∈ bag
          then integerp (x)
                ∧   (itimes (x, eval$ (t, elt, a))
                      =   itimes-list (eval$ ('list, bag, a)))
          else x = itimes-list (eval$ ('list, bag, a)) endif)
```

Theorem: itimes-itimes-list-eval\$-list-delete

```
(x ∈ bag)
→   (itimes (eval$ (t, x, a), itimes-list (eval$ ('list, delete (x, bag), a)))
      =   itimes-list (eval$ ('list, bag, a)))
```

Theorem: equal-itimes-list-eval\$-list-bagdiff
(subbagp (*in-both*, *bag1*)

$\wedge$ subbagp (*in-both*, *bag2*)
$\wedge$ (itimes-list (eval\$ (`'list`, *in-both*, a)) $\neq$ `0`))
$\rightarrow$ ((itimes-list (eval\$ (`'list`, bagdiff (*bag1*, *in-both*), a))
  $=$ itimes-list (eval\$ (`'list`, bagdiff (*bag2*, *in-both*), a)))
  $=$ (itimes-list (eval\$ (`'list`, *bag1*, a))
    $=$ itimes-list (eval\$ (`'list`, *bag2*, a))))

Theorem: membership-of-0-implies-itimes-list-is-0
(`0` $\in x$) $\rightarrow$ (itimes-list (x) $=$ `0`)

Theorem: member-0-eval\$-list
(`0` $\in x$) $\rightarrow$ (`0` $\in$ eval\$ (`'list`, x, a))

Theorem: itimes-list-eval\$-factors-lemma
itimes (itimes-list (eval\$ (`'list`, bagint (*bag1*, *bag2*), a)),
  itimes-list (eval\$ (`'list`, bagdiff (*bag2*, bagint (*bag1*, *bag2*)), a)))
$=$ itimes-list (eval\$ (`'list`, *bag2*, a))

Theorem: itimes-list-eval\$-factors-lemma-prime
itimes (itimes-list (eval\$ (`'list`, bagint (*bag1*, *bag2*), a)),
  itimes-list (eval\$ (`'list`, bagdiff (*bag1*, bagint (*bag1*, *bag2*)), a)))
$=$ itimes-list (eval\$ (`'list`, *bag1*, a))

Theorem: itimes-list-eval\$-factors
itimes-list (eval\$ (`'list`, itimes-factors (x), a)) $=$ fix-int (eval\$ (**t**, x, a))

Theorem: iplus-or-itimes-term-integerp-eval\$
iplus-or-itimes-term (x) $\rightarrow$ integerp (eval\$ (**t**, x, a))

Theorem: eval\$-list-bagint-0
(itimes-list (eval\$ (`'list`, bagint (x, y), a)) $=$ `0`)
$\rightarrow$ ((itimes-list (eval\$ (`'list`, x, a)) $=$ `0`)
  $\wedge$ (itimes-list (eval\$ (`'list`, y, a)) $=$ `0`))

Theorem: eval\$-list-bagint-0-implies-equal
((itimes-list (eval\$ (`'list`, bagint (itimes-factors (v), itimes-factors (w)), a))
 $=$ `0`)
$\wedge$ integerp (eval\$ (**t**, v, a))
$\wedge$ integerp (eval\$ (**t**, w, a)))
$\rightarrow$ ((eval\$ (**t**, v, a) $=$ eval\$ (**t**, w, a)) $=$ **t**)

Theorem: correctness-of-cancel-itimes-factors
eval\$ (**t**, x, a) $=$ eval\$ (**t**, cancel-itimes-factors-expanded (x), a)

```
;; OK -- now, the lessp case, finally.  Ugh!
```

Definition:
cancel-itimes-ilessp-factors (x)

```
=   if listp (x)
    then if car (x) = 'ilessp
         then if listp (bagint (itimes-factors (cadr (x)),
                                itimes-factors (caddr (x))))
              then make-cancel-itimes-inequality (itimes-factors (cadr (x)),
                                                  itimes-factors (caddr (x)),
                                                  bagint (itimes-factors (cadr (x)),
                                                          itimes-factors (caddr (x))))
              else x endif
         else x endif
    else x endif
```

Theorem: bagint-singleton

```
 bagint (x, list (y))
 =   if y ∈ x  then list (y)
     else nil endif
```

Theorem: izerop-ilessp-0-relationship
$(\text{fix-int}(x) = 0) = ((\neg\ \text{ilessp}(x,\ 0)) \land (\neg\ \text{ilessp}(0,\ x)))$

Theorem: ilessp-itimes-list-eval\$-list-delete-helper-1
$\text{ilessp}(0,\ w) \rightarrow (\text{ilessp}(\text{itimes}(x,\ w),\ \text{itimes}(w,\ u)) = \text{ilessp}(x,\ u))$

Theorem: ilessp-itimes-list-eval\$-list-delete-helper-2
$\text{ilessp}(w,\ 0) \rightarrow (\text{ilessp}(\text{itimes}(w,\ u),\ \text{itimes}(x,\ w)) = \text{ilessp}(x,\ u))$

Theorem: ilessp-itimes-list-eval\$-list-delete

```
 ((z ∈ y) ∧ (fix-int (eval$ (t, z, a)) ≠ 0))
 →   (ilessp (x, itimes-list (eval$ ('list, delete (z, y), a)))
        =   if ilessp (0, eval$ (t, z, a))
            then ilessp (itimes (x, eval$ (t, z, a)),
                         itimes-list (eval$ ('list, y, a)))
            elseif ilessp (eval$ (t, z, a), 0)
            then ilessp (itimes-list (eval$ ('list, y, a)),
                         itimes (x, eval$ (t, z, a)))
            else f endif)
```

Theorem: ilessp-itimes-list-eval\$-list-delete-prime-helper-1
$\text{ilessp}(0,\ w) \rightarrow (\text{ilessp}(\text{itimes}(w,\ u),\ \text{itimes}(x,\ w)) = \text{ilessp}(u,\ x))$

Theorem: ilessp-itimes-list-eval\$-list-delete-prime-helper-2
$\text{ilessp}(w,\ 0) \rightarrow (\text{ilessp}(\text{itimes}(x,\ w),\ \text{itimes}(w,\ u)) = \text{ilessp}(u,\ x))$

THEOREM: ilessp-itimes-list-eval$-list-delete-prime
$((z \in y) \land (\text{fix-int}\,(\text{eval\$}\,(\mathbf{t},\ z,\ a)) \neq \texttt{0}))$
$\rightarrow\ (\text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \text{delete}\,(z,\ y),\ a)),\ x)$
$=\ \textbf{if}\ \text{ilessp}\,(\texttt{0},\ \text{eval\$}\,(\mathbf{t},\ z,\ a))$
$\quad \textbf{then}\ \text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ y,\ a)),$
$\qquad \text{itimes}\,(x,\ \text{eval\$}\,(\mathbf{t},\ z,\ a)))$
$\quad \textbf{elseif}\ \text{ilessp}\,(\text{eval\$}\,(\mathbf{t},\ z,\ a),\ \texttt{0})$
$\quad \textbf{then}\ \text{ilessp}\,(\text{itimes}\,(x,\ \text{eval\$}\,(\mathbf{t},\ z,\ a)),$
$\qquad \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ y,\ a)))$
$\quad \textbf{else f endif})$

```
;; **** Do I have anything like the following two lemmas for the equality case?
;; Should I?

;;;***** I should also consider if I've dealt with things like 0 = a*x + b*x, and
;;; simlilarly for ilessp.
```

THEOREM: ilessp-0-itimes
$\text{ilessp}\,(\texttt{0},\ \text{itimes}\,(x,\ y))$
$=\ ((\text{ilessp}\,(\texttt{0},\ x) \land \text{ilessp}\,(\texttt{0},\ y)) \lor (\text{ilessp}\,(x,\ \texttt{0}) \land \text{ilessp}\,(y,\ \texttt{0})))$

THEOREM: ilessp-itimes-0
$\text{ilessp}\,(\text{itimes}\,(x,\ y),\ \texttt{0})$
$=\ ((\text{ilessp}\,(\texttt{0},\ x) \land \text{ilessp}\,(y,\ \texttt{0})) \lor (\text{ilessp}\,(x,\ \texttt{0}) \land \text{ilessp}\,(\texttt{0},\ y)))$

THEOREM: ilessp-itimes-list-eval$-list-bagdiff
$(\text{subbagp}\,(\textit{in-both},\ \textit{bag1})$
$\land\ \text{subbagp}\,(\textit{in-both},\ \textit{bag2})$
$\land\ (\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \textit{in-both},\ a)) \neq \texttt{0}))$
$\rightarrow\ (\text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \text{bagdiff}\,(\textit{bag1},\ \textit{in-both}),\ a)),$
$\qquad \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \text{bagdiff}\,(\textit{bag2},\ \textit{in-both}),\ a)))$
$=\ \textbf{if}\ \text{ilessp}\,(\texttt{0},\ \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \textit{in-both},\ a)))$
$\quad \textbf{then}\ \text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \textit{bag1},\ a)),$
$\qquad \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \textit{bag2},\ a)))$
$\quad \textbf{else}\ \text{ilessp}\,(\text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \textit{bag2},\ a)),$
$\qquad \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \textit{bag1},\ a)))\ \textbf{endif})$

THEOREM: zero-ilessp-implies-not-equal
$\text{ilessp}\,(\texttt{0},\ x) \rightarrow (\texttt{0} \neq x)$

THEOREM: ilessp-itimes-list-eval$-list-bagdiff-corollary-1
$(\text{subbagp}\,(\textit{in-both},\ \textit{bag1})$
$\land\ \text{subbagp}\,(\textit{in-both},\ \textit{bag2})$
$\land\ \text{ilessp}\,(\texttt{0},\ \text{itimes-list}\,(\text{eval\$}\,(\texttt{'list},\ \textit{in-both},\ a))))$

$\rightarrow$ (ilessp (itimes-list (eval\$ (`'list`, bagdiff (*bag1*, *in-both*), *a*)),
 itimes-list (eval\$ (`'list`, bagdiff (*bag2*, *in-both*), *a*)))
 $=$ ilessp (itimes-list (eval\$ (`'list`, *bag1*, *a*)),
 itimes-list (eval\$ (`'list`, *bag2*, *a*))))

THEOREM: ilessp-zero-implies-not-equal
ilessp (x, 0) $\rightarrow$ ($0 \neq x$)

THEOREM: ilessp-itimes-list-eval\$-list-bagdiff-corollary-2
(subbagp (*in-both*, *bag1*)
$\wedge$ subbagp (*in-both*, *bag2*)
$\wedge$ ilessp (itimes-list (eval\$ (`'list`, *in-both*, *a*)), 0))
$\rightarrow$ (ilessp (itimes-list (eval\$ (`'list`, bagdiff (*bag1*, *in-both*), *a*)),
 itimes-list (eval\$ (`'list`, bagdiff (*bag2*, *in-both*), *a*)))
 $=$ ilessp (itimes-list (eval\$ (`'list`, *bag2*, *a*)),
 itimes-list (eval\$ (`'list`, *bag1*, *a*))))

THEOREM: member-0-itimes-factors-yields-0
(eval\$ (**t**, w, a) $\neq$ 0) $\rightarrow$ (0 $\notin$ itimes-factors (w))

THEOREM: member-0-itimes-factors-yields-0-ilessp-consequence-1
ilessp (eval\$ (**t**, w, a), 0) $\rightarrow$ (0 $\notin$ itimes-factors (w))

THEOREM: member-0-itimes-factors-yields-0-ilessp-consequence-2
ilessp (0, eval\$ (**t**, w, a)) $\rightarrow$ (0 $\notin$ itimes-factors (w))

```
#|
(prove-lemma eval$-list-bagint-0 nil
  (implies (equal (itimes-list (eval$ 'list (bagint x y) a)) 0)
   (and (equal (itimes-list (eval$ 'list x a)) 0)
(equal (itimes-list (eval$ 'list y a)) 0)))
  ((use (subsetp-implies-itimes-list-eval$-equals-0
 (x (bagint x y))
 (y x))
(subsetp-implies-itimes-list-eval$-equals-0
 (x (bagint x y))
 (y y)))))
|#

#|
(prove-lemma eval$-list-bagint-0-implies-equal (rewrite)
  (implies (and (equal (itimes-list (eval$ 'list (bagint (itimes-factors v) (itimes-factors
      0)
(integerp (eval$ t v a))
(integerp (eval$ t w a)))
```

```
   (equal (equal (eval$ t v a) (eval$ t w a))
  t))
  ((use (eval$-list-bagint-0 (x (itimes-factors v))
     (y (itimes-factors w))))))
|#

;; At this point I'm going to switch the states of ilessp-trichotomy and
;; izerop-ilessp-0-relationship, for good (or till I change my mind again!).
```

EVENT: Enable ilessp-trichotomy.

EVENT: Disable izerop-ilessp-0-relationship.

THEOREM: eval$-list-bagint-0-for-ilessp
$((\neg \text{ ilessp}(\text{itimes-list}(\text{eval\$}(\texttt{'list}, \text{bagint}(x, y), a)), \texttt{0})) \wedge (\neg \text{ ilessp}(\texttt{0}, \text{itimes-list}(\text{eval\$}(\texttt{'list}, \text{bagint}(x, y), a))))) \rightarrow ((\text{fix-int}(\text{itimes-list}(\text{eval\$}(\texttt{'list}, x, a))) = \texttt{0}) \wedge (\text{fix-int}(\text{itimes-list}(\text{eval\$}(\texttt{'list}, y, a))) = \texttt{0}))$

THEOREM: eval$-list-bagint-0-implies-equal-for-ilessp-lemma
$((\neg \text{ ilessp}(\text{itimes-list}(\text{eval\$}(\texttt{'list}, \text{bagint}(\text{itimes-factors}(v), \text{itimes-factors}(w)), a)), \texttt{0})) \wedge (\neg \text{ ilessp}(\texttt{0}, \text{itimes-list}(\text{eval\$}(\texttt{'list}, \text{bagint}(\text{itimes-factors}(v), \text{itimes-factors}(w)), a))))) \rightarrow (\text{fix-int}(\text{eval\$}(\mathbf{t}, v, a)) = \text{fix-int}(\text{eval\$}(\mathbf{t}, w, a)))$

THEOREM: equal-fix-int-to-ilessp
$(\text{fix-int}(x) = \text{fix-int}(y)) \rightarrow (\neg \text{ ilessp}(x, y))$

THEOREM: eval$-list-bagint-0-implies-equal-for-ilessp
$((\neg \text{ ilessp}(\text{itimes-list}(\text{eval\$}(\texttt{'list}, \text{bagint}(\text{itimes-factors}(v), \text{itimes-factors}(w)), a)), \texttt{0})) \wedge (\neg \text{ ilessp}(\texttt{0}, \text{itimes-list}(\text{eval\$}(\texttt{'list}, \text{bagint}(\text{itimes-factors}(v), \text{itimes-factors}(w)),$

```
                                        a)))))
 →    ((¬ ilessp (eval$ (t, v, a), eval$ (t, w, a)))
        ∧    (¬ ilessp (eval$ (t, w, a), eval$ (t, v, a))))
```

```
;; The rewrite rule ILESSP-TRICHOTOMY seemed to mess up the proof of the following,
;; so I'm just going to leave it disabled.
```

EVENT: Disable ilessp-trichotomy.

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

```
;; OK -- now, the zero cases.
```

EVENT: Enable lessp-count-listp-cdr.

DEFINITION:
disjoin-equalities-with-0 (*factors*)

```
=    if listp (cdr (factors))
     then list ('or,
                list ('equal, list ('fix-int, car (factors)), ''0),
                disjoin-equalities-with-0 (cdr (factors)))
     else list ('equal, list ('fix-int, car (factors)), ''0) endif
```

EVENT: Disable lessp-count-listp-cdr.

DEFINITION:
cancel-factors-0 (x)

```
=    if listp (x)
     then if car (x) = 'equal
          then if cadr (x) = ''0
               then let factors be itimes-factors (caddr (x))
                    in
                    if listp (cdr (factors))
                    then disjoin-equalities-with-0 (factors)
                    else x endif endlet
               elseif caddr (x) = ''0
               then let factors be itimes-factors (cadr (x))
                    in
                    if listp (cdr (factors))
                    then disjoin-equalities-with-0 (factors)
                    else x endif endlet
```

```
                else $x$ **endif**
            else $x$ **endif**
        else $x$ **endif**
```

DEFINITION:
```
some-eval$s-to-0 ($x$, $a$)
=   if listp ($x$)
    then (fix-int (eval$ (t, car ($x$), $a$)) = 0)
            ∨  some-eval$s-to-0 (cdr ($x$), $a$)
    else f endif
```

THEOREM: eval$-disjoin-equalities-with-0
```
 listp ($lst$)
 →   (eval$ (t, disjoin-equalities-with-0 ($lst$), $a$)
       =   some-eval$s-to-0 ($lst$, $a$))
```

THEOREM: some-eval$s-to-0-append
```
 some-eval$s-to-0 (append ($x$, $y$), $a$)
 =   (some-eval$s-to-0 ($x$, $a$) ∨ some-eval$s-to-0 ($y$, $a$))
```

THEOREM: some-eval$s-to-0-eliminator
some-eval$s-to-0 ($x$, a) = (itimes-list (eval$ ('list, x, a)) = 0)

THEOREM: listp-cdr-factors-implies-integerp
listp (cdr (itimes-factors (v))) → integerp (eval$ (**t**, v, a))

THEOREM: correctness-of-cancel-factors-0
eval$ (**t**, x, a) = eval$ (**t**, cancel-factors-0 (x), a)

```
;; and now for inequalities...
```

EVENT: Enable lessp-count-listp-cdr.

DEFINITION:
```
conjoin-inequalities-with-0 ($factors$, $parity$)
=   if listp (cdr ($factors$))
    then if $parity$
         then list ('or,
                    list ('and,
                          list ('ilessp, ''0, car ($factors$)),
                          conjoin-inequalities-with-0 (cdr ($factors$), t)),
                    list ('and,
                          list ('ilessp, car ($factors$), ''0),
                          conjoin-inequalities-with-0 (cdr ($factors$), f)))
```

```
          else list ('or,
                     list ('and,
                           list ('ilessp, car (factors), ''0),
                           conjoin-inequalities-with-0 (cdr (factors), t)),
                     list ('and,
                           list ('ilessp, ''0, car (factors)),
                           conjoin-inequalities-with-0 (cdr (factors), f))) endif
    elseif parity then list ('ilessp, ''0, car (factors))
    else list ('ilessp, car (factors), ''0) endif
```

Event: Disable lessp-count-listp-cdr.

Definition:

```
cancel-factors-ilessp-0 (x)
=   if listp (x)
    then if car (x) = 'ilessp
         then if cadr (x) = ''0
              then let factors be itimes-factors (caddr (x))
                   in
                   if listp (cdr (factors))
                   then conjoin-inequalities-with-0 (factors, t)
                   else x endif endlet
              elseif caddr (x) = ''0
              then let factors be itimes-factors (cadr (x))
                   in
                   if listp (cdr (factors))
                   then conjoin-inequalities-with-0 (factors, f)
                   else x endif endlet
              else x endif
         else x endif
    else x endif
```

Theorem: conjoin-inequalities-with-0-eliminator

```
listp (x)
→   (eval$ (t, conjoin-inequalities-with-0 (x, parity), a)
      =  if parity then ilessp (0, itimes-list (eval$ ('list, x, a)))
         else ilessp (itimes-list (eval$ ('list, x, a)), 0) endif)
```

Theorem: correctness-of-cancel-factors-ilessp-0

eval$ (**t**, x, a) = eval$ (**t**, cancel-factors-ilessp-0 (x), a)

Event: Disable equal-itimes-list-eval$-list-delete-new-1.

Event: Disable equal-itimes-list-eval$-list-delete-new-2.

Event: Disable itimes-itimes-list-eval$-list-delete.

Event: Disable equal-itimes-list-eval$-list-bagdiff.

Event: Disable itimes-list-eval$-factors-lemma.

Event: Disable itimes-list-eval$-factors-lemma-prime.

Event: Disable itimes-list-eval$-factors.

Event: Disable iplus-or-itimes-term-integerp-eval$.

Event: Disable eval$-list-bagint-0.

Event: Disable eval$-list-bagint-0-implies-equal.

Event: Disable izerop-ilessp-0-relationship.

Event: Disable ilessp-itimes-list-eval$-list-delete-helper-1.

Event: Disable ilessp-itimes-list-eval$-list-delete-helper-2.

Event: Disable ilessp-itimes-list-eval$-list-delete.

Event: Disable ilessp-itimes-list-eval$-list-delete-prime-helper-1.

Event: Disable ilessp-itimes-list-eval$-list-delete-prime-helper-2.

Event: Disable ilessp-itimes-list-eval$-list-delete-prime.

Event: Disable ilessp-0-itimes.

Event: Disable ilessp-itimes-0.

Event: Disable listp-cdr-factors-implies-integerp.

```
;; We presumably have better meta-lemmas now, but if we want we
;; can disable those (i.e., correctness-of-cancel-itimes-factors,
;; correctness-of-cancel-itimes-ilessp-factors,
;; correctness-of-cancel-factors-0, and
;; correctness-of-cancel-factors-ilessp-0) and enable the two
;; mentioned below:
```

EVENT: Disable correctness-of-cancel-itimes.

EVENT: Disable correctness-of-cancel-itimes-ilessp.

```
;; I'll disable some rules now, finally, that I'd previously thought
;; would be OK but now fear because of potential nasty backchaining.
```

EVENT: Disable not-integerp-implies-not-equal-iplus.

EVENT: Disable not-integerp-implies-not-equal-itimes.

EVENT: Disable subbagp-subsetp.

EVENT: Disable eval$-list-bagint-0-implies-equal-for-ilessp.

```
; ---------- Cancel ineg terms from equalities and inequalities ----------
```

DEFINITION:

```
split-out-ineg-terms ($x$)
=   if listp ($x$)
    then let pair be split-out-ineg-terms (cdr ($x$)),
             $a$ be car ($x$)
         in
         if listp ($a$)
         then if car ($a$) = 'ineg
              then cons (car (pair), cons (cadr ($a$), cdr (pair)))
              elseif (car ($a$) = 'quote)
                     ∧ negativep (cadr ($a$))
                     ∧ (negative-guts (cadr ($a$)) ≠ 0)
              then cons (car (pair),
                         cons (list ('quote, negative-guts (cadr ($a$))),
                              cdr (pair)))
              else cons (cons (a, car (pair)), cdr (pair)) endif
         else cons (cons (a, car (pair)), cdr (pair)) endif endlet
    else cons (nil, nil) endif
```

DEFINITION:
remove-inegs (x, y)

```
=    let xpair  be  split-out-ineg-terms (x),
         ypair  be  split-out-ineg-terms (y)
     in
     if listp (cdr (xpair)) ∨ listp (cdr (ypair))
     then cons (append (cdr (ypair), car (xpair)),
                append (cdr (xpair), car (ypair)))
     else f endif endlet
```

DEFINITION:
iplus-or-ineg-term (x)
= (listp (x) $\land$ ((car (x) = `'ineg`) $\lor$ (car (x) = `'iplus`)))

DEFINITION:
make-cancel-ineg-terms-equality (x)

```
=    let new-fringes  be  remove-inegs (iplus-fringe (cadr (x)),
                                        iplus-fringe (caddr (x)))
     in
     if new-fringes
     then if iplus-or-ineg-term (cadr (x))
          then if iplus-or-ineg-term (caddr (x))
               then list ('equal,
                          iplus-tree (car (new-fringes)),
                          iplus-tree (cdr (new-fringes)))
               else list ('if,
                         list ('integerp, caddr (x)),
                         list ('equal,
                               iplus-tree (car (new-fringes)),
                               iplus-tree (cdr (new-fringes))),
                         list ('quote, f)) endif
          elseif iplus-or-ineg-term (caddr (x))
          then list ('if,
                    list ('integerp, cadr (x)),
                    list ('equal,
                          iplus-tree (car (new-fringes)),
                          iplus-tree (cdr (new-fringes))),
                    list ('quote, f))
          else x endif
     else x endif endlet
```

```
DEFINITION:
cancel-ineg-terms-from-equality (x)
=    if listp (x) ∧ (car (x) = 'equal)
     then make-cancel-ineg-terms-equality (x)
     else x endif

;; The following was created from nqthm-macroexpand with arguments
;; and or make-cancel-ineg-terms-equality iplus-or-ineg-term

DEFINITION:
cancel-ineg-terms-from-equality-expanded (x)
=    if listp (x)
     then if car (x) = 'equal
          then if remove-inegs (iplus-fringe (cadr (x)),
                                iplus-fringe (caddr (x)))
               then if listp (cadr (x))
                    then case on car (car (cdr (x))):
                         case = ineg
                         then if listp (caddr (x))
                              then case on car (car (cdr (cdr (x)))):
                                   case = ineg
                                   then list ('equal,
                                              iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                                             iplus-fringe (caddr (x))))),
                                              iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                                             iplus-fringe (caddr (x))))))
                                   case = iplus
                                    then list ('equal,
                                               iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                                              iplus-fringe (caddr (x))))),
                                               iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                                              iplus-fringe (caddr (x))))))
                                   otherwise list ('if,
                                                   list ('integerp,
                                                         caddr (x)),
                                                   list ('equal,
                                                         iplus-tree (car (remove-inegs (iplus-fringe (cadr (x
                                                                                        iplus-fringe (caddr (
                                                         iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x
                                                                                        iplus-fringe (caddr (
                                                   list ('quote, f)) endcase
                              else list ('if,
                                         list ('integerp, caddr (x)),
                                         list ('equal,
                    iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                 iplus-fringe (caddr (x))))),
                    iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                 iplus-fringe (caddr (x)))))),
               list ('quote, f)) endif
case = iplus
 then if listp (caddr (x))
       then case on car (car (cdr (cdr (x)))):
             case = ineg
             then list ('equal,
                        iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                     iplus-fringe (caddr (x))))),
                        iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                     iplus-fringe (caddr (x))))))
             case = iplus
              then list ('equal,
                         iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                      iplus-fringe (caddr (x))))),
                         iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                      iplus-fringe (caddr (x))))))
             otherwise list ('if,
                              list ('integerp,
                                  caddr (x)),
                              list ('equal,
                                  iplus-tree (car (remove-inegs (iplus-fringe (cadr (
                                                               iplus-fringe (caddr
                                  iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (
                                                               iplus-fringe (caddr
                              list ('quote, f)) endcase
       else list ('if,
                 list ('integerp, caddr (x)),
                 list ('equal,
                     iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                  iplus-fringe (caddr (x))))),
                     iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                  iplus-fringe (caddr (x)))))),
                 list ('quote, f)) endif
otherwise if listp (caddr (x))
            then case on car (car (cdr (cdr (x)))):
                  case = ineg
                  then list ('if,
                            list ('integerp,
                                cadr (x)),
                            list ('equal,
                                               iplus-tree (car (remove-inegs (iplus-fringe (cadr (x
                                                                       iplus-fringe (caddr (
                                               iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x
                                                                       iplus-fringe (caddr (
                                           list ('quote, f))
                                   case = iplus
                                    then list ('if,
                                            list ('integerp,
                                                cadr (x)),
                                            list ('equal,
                                                iplus-tree (car (remove-inegs (iplus-fringe (cadr (
                                                                        iplus-fringe (caddr
                                                iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (
                                                                        iplus-fringe (caddr
                                            list ('quote, f))
                                   otherwise x endcase
                              else x endif endcase
               elseif listp (caddr (x))
               then case on car (car (cdr (cdr (x)))):
                    case = ineg
                    then list ('if,
                             list ('integerp, cadr (x)),
                             list ('equal,
                                iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                        iplus-fringe (caddr (x))))),
                                iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                        iplus-fringe (caddr (x)))))),
                             list ('quote, f))
                    case = iplus
                     then list ('if,
                              list ('integerp, cadr (x)),
                              list ('equal,
                                 iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                         iplus-fringe (caddr (x))))),
                                 iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                         iplus-fringe (caddr (x)))))),
                              list ('quote, f))
                    otherwise x endcase
               else x endif
          else x endif
     else x endif
else x endif
```

Theorem: cancel-ineg-terms-from-equality-cancel-ineg-terms-from-equality-expanded

cancel-ineg-terms-from-equality-expanded (x)
$=$ cancel-ineg-terms-from-equality (x)

Event: Disable cancel-ineg-terms-from-equality-expanded.

Theorem: integerp-eval$-iplus-or-ineg-term
iplus-or-ineg-term (x) $\rightarrow$ integerp (eval$ (**t**, x, a))

Event: Disable iplus-or-ineg-term.

Theorem: eval$-iplus-list-car-remove-inegs
remove-inegs (x, y)
$\rightarrow$ (iplus-list (eval$ (`'list`, car (remove-inegs (x, y)), a))
    $=$ iplus (iplus-list (eval$ (`'list`, car (split-out-ineg-terms (x)), a)),
        iplus-list (eval$ (`'list`, cdr (split-out-ineg-terms (y)), a))))

Theorem: eval$-iplus-list-cdr-remove-inegs
remove-inegs (x, y)
$\rightarrow$ (iplus-list (eval$ (`'list`, cdr (remove-inegs (x, y)), a))
    $=$ iplus (iplus-list (eval$ (`'list`, car (split-out-ineg-terms (y)), a)),
        iplus-list (eval$ (`'list`, cdr (split-out-ineg-terms (x)), a))))

Theorem: minus-ineg
$((x \in \mathbf{N}) \land (x \neq \mathtt{0})) \rightarrow ((-\ x) = \text{ineg}\,(x))$

Theorem: iplus-list-eval$-car-split-out-ineg-terms
iplus-list (eval$ (`'list`, car (split-out-ineg-terms (x)), a))
$=$ iplus (iplus-list (eval$ (`'list`, x, a)),
    iplus-list (eval$ (`'list`, cdr (split-out-ineg-terms (x)), a)))

Event: Disable remove-inegs.

Theorem: correctness-of-cancel-ineg-terms-from-equality
eval$ (**t**, x, a) $=$ eval$ (**t**, cancel-ineg-terms-from-equality-expanded (x), a)

Definition:
make-cancel-ineg-terms-inequality (x)

```
=   let new-fringes be remove-inegs (iplus-fringe (cadr (x)),
                                     iplus-fringe (caddr (x)))
    in
    if new-fringes
    then list ('ilessp,
               iplus-tree (car (new-fringes)),
               iplus-tree (cdr (new-fringes)))
    else x endif endlet
```

DEFINITION:

```
cancel-ineg-terms-from-inequality (x)
=   if listp (x) ∧ (car (x) = 'ilessp)
    then if iplus-or-ineg-term (cadr (x))
         then make-cancel-ineg-terms-inequality (x)
         elseif iplus-or-ineg-term (caddr (x))
         then make-cancel-ineg-terms-inequality (x)
         else x endif
    else x endif
```

```
;; The following was created from nqthm-macroexpand with arguments
;; and or make-cancel-ineg-terms-inequality iplus-or-ineg-term
```

DEFINITION:

```
cancel-ineg-terms-from-inequality-expanded (x)
=   if listp (x)
    then if car (x) = 'ilessp
         then if listp (cadr (x))
              then case on car (car (cdr (x))):
                   case = ineg
                   then if remove-inegs (iplus-fringe (cadr (x)),
                                         iplus-fringe (caddr (x)))
                        then list ('ilessp,
                                   iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                                  iplus-fringe (caddr (x))))),
                                   iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                                  iplus-fringe (caddr (x))))))
                        else x endif
                   case = iplus
                    then if remove-inegs (iplus-fringe (cadr (x)),
                                          iplus-fringe (caddr (x)))
                         then list ('ilessp,
                                    iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                                   iplus-fringe (caddr (x))))),
                                    iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                                   iplus-fringe (caddr (x))))))
                         else x endif
                   otherwise if listp (caddr (x))
                             then case on car (car (cdr (cdr (x)))):
                                  case = ineg
                                  then if remove-inegs (iplus-fringe (cadr (x)),
                                                        iplus-fringe (caddr (x)))
                                       then list ('ilessp,
                                                    iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                                                     iplus-fringe (caddr (x))))
                                                    iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                                                     iplus-fringe (caddr (x))))
                                          else x endif
                                    case = iplus
                                     then if remove-inegs (iplus-fringe (cadr (x)),
                                                            iplus-fringe (caddr (x)))
                                           then list ('ilessp,
                                                      iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                                                       iplus-fringe (caddr (x))
                                                      iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                                                       iplus-fringe (caddr (x))
                                           else x endif
                                    otherwise x endcase
                              else x endif endcase
            elseif listp (caddr (x))
            then case on car (car (cdr (cdr (x)))):
                  case = ineg
                  then if remove-inegs (iplus-fringe (cadr (x)),
                                         iplus-fringe (caddr (x)))
                        then list ('ilessp,
                                   iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                                   iplus-fringe (caddr (x))))),
                                   iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                                   iplus-fringe (caddr (x))))))
                        else x endif
                  case = iplus
                   then if remove-inegs (iplus-fringe (cadr (x)),
                                          iplus-fringe (caddr (x)))
                         then list ('ilessp,
                                    iplus-tree (car (remove-inegs (iplus-fringe (cadr (x)),
                                                                    iplus-fringe (caddr (x))))),
                                    iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x)),
                                                                    iplus-fringe (caddr (x))))))
                         else x endif
                  otherwise x endcase
            else x endif
      else x endif
else x endif
```

Theorem: cancel-ineg-terms-from-inequality-cancel-ineg-terms-from-inequality-expanded
cancel-ineg-terms-from-inequality-expanded (x)
= cancel-ineg-terms-from-inequality (x)

EVENT: Disable cancel-ineg-terms-from-inequality-expanded.

THEOREM: correctness-of-cancel-ineg-terms-from-inequality
eval$ ($\mathbf{t}$, x, a) = eval$ ($\mathbf{t}$, cancel-ineg-terms-from-inequality-expanded (x), a)

EVENT: Disable minus-ineg.

EVENT: Disable integerp-eval$-iplus-or-ineg-term.

```
; ---------- Eliminating constants ----------

;; We want to combine in terms like (iplus 3 (iplus x 7)).  Also, when
;; two iplus terms are equated or in-equated, there should only be a
;; natural number summand on at most one side.  Finally, if one adds 1
;; to the right side of a strict inequality, a stronger inequality (in
;; a certain sense) is obtained by removing the 1 and making a non-strict
;; inequality in the other direction.
```

THEOREM: plus-iplus
$((i \in \mathbf{N}) \land (j \in \mathbf{N})) \rightarrow ((i + j) = \text{iplus}\,(i,\, j))$

THEOREM: iplus-constants
$\text{iplus}\,(1 + i,\, \text{iplus}\,(1 + j,\, x)) = \text{iplus}\,((1 + i) + (1 + j),\, x)$

THEOREM: numberp-is-integerp
$(w \in \mathbf{N}) \rightarrow \text{integerp}\,(w)$

THEOREM: difference-idifference
$((x \in \mathbf{N}) \land (y \in \mathbf{N}) \land (x \leq y)) \rightarrow ((y - x) = \text{idifference}\,(y,\, x))$

THEOREM: cancel-constants-equal-lemma
$((m \in \mathbf{N}) \land (n \in \mathbf{N}))$
$\rightarrow \quad ((\text{iplus}\,(m,\, x) = \text{iplus}\,(n,\, y))$
$\quad\quad = \quad \textbf{if } m < n \textbf{ then } \text{fix-int}\,(x) = \text{iplus}\,(n - m,\, y)$
$\quad\quad\quad\quad \textbf{else } \text{iplus}\,(m - n,\, x) = \text{fix-int}\,(y) \textbf{ endif})$

THEOREM: cancel-constants-equal
$(\text{iplus}\,(1 + i,\, x) = \text{iplus}\,(1 + j,\, y))$
$= \quad \textbf{if } i < j \textbf{ then } \text{fix-int}\,(x) = \text{iplus}\,(j - i,\, y)$
$\quad\quad \textbf{else } \text{iplus}\,(i - j,\, x) = \text{fix-int}\,(y) \textbf{ endif}$

THEOREM: ilessp-add1
$(y \in \mathbf{N}) \rightarrow (\text{ilessp}\,(x,\, 1 + y) = (\neg\, \text{ilessp}\,(y,\, x)))$

THEOREM: ilessp-add1-iplus
$(y \in \mathbf{N}) \rightarrow (\text{ilessp}\,(x,\ \text{iplus}\,(1 + y,\ z)) = (\neg\ \text{ilessp}\,(\text{iplus}\,(y,\ z),\ x)))$

THEOREM: cancel-constants-ilessp-lemma-1
$((m \in \mathbf{N}) \wedge (n \in \mathbf{N}))$
$\rightarrow \quad (\text{ilessp}\,(\text{iplus}\,(m,\ x),\ \text{iplus}\,(n,\ y))$
$\quad = \quad \textbf{if } m < n \textbf{ then } \text{ilessp}\,(x,\ \text{iplus}\,(n - m,\ y))$
$\quad\quad \textbf{else } \text{ilessp}\,(\text{iplus}\,(m - n,\ x),\ y) \textbf{ endif})$

THEOREM: cancel-constants-ilessp-lemma-2
$((m \in \mathbf{N}) \wedge (n \in \mathbf{N}))$
$\rightarrow \quad (\text{ilessp}\,(\text{iplus}\,(m,\ x),\ \text{iplus}\,(n,\ y))$
$\quad = \quad \textbf{if } m < n \textbf{ then } \neg\ \text{ilessp}\,(\text{iplus}\,((n - m) - 1,\ y),\ x)$
$\quad\quad \textbf{else } \text{ilessp}\,(\text{iplus}\,(m - n,\ x),\ y) \textbf{ endif})$

THEOREM: cancel-constants-ilessp
$\text{ilessp}\,(\text{iplus}\,(1 + i,\ x),\ \text{iplus}\,(1 + j,\ y))$
$= \quad \textbf{if } i < j \textbf{ then } \neg\ \text{ilessp}\,(\text{iplus}\,((j - i) - 1,\ y),\ x)$
$\quad \textbf{else } \text{ilessp}\,(\text{iplus}\,(i - j,\ x),\ y) \textbf{ endif}$

EVENT: Disable plus-iplus.

EVENT: Disable numberp-is-integerp.

EVENT: Disable difference-idifference.

```
; ---------- Final DEFTHEORY event ----------

;; I'll go ahead and include iplus-list and itimes-list and lemmas
;; about them that were developed.

;; I've left out ILESSP-TRICHOTOMY because I'm scared it will slow
;; things down too much.  But it certainly represents useful
;; information.
```

EVENT: Let us define the theory *integers* to consist of the following events: ileq, idifference, integerp-fix-int, integerp-iplus, integerp-idifference, integerp-ineg, integerp-iabs, integerp-itimes, fix-int-remover, fix-int-fix-int, fix-int-iplus, fix-int-idifference, fix-int-ineg, fix-int-iabs, fix-int-itimes, ineg-iplus, ineg-ineg, ineg-fix-int, ineg-of-non-integerp, ineg-0, iplus-left-id, iplus-right-id, iplus-0-left, iplus-0-right, commutativity2-of-iplus, commutativity-of-iplus, associativity-of-iplus, iplus-cancellation-1, iplus-cancellation-2, iplus-ineg1, iplus-ineg2, iplus-fix-int1, iplus-fix-int2, idifference-fix-int1, idifference-fix-int2, iplus-list, eval$-list-append, iplus-list-append, iplus-ineg3, iplus-ineg4, correctness-of-cancel-iplus, ilessp-fix-int-1, ilessp-fix-int-2, iplus-cancellation-1-for-ilessp, iplus-cancellation-2-for-ilessp, correctness-of-cancel-iplus-ilessp, itimes-0-left, itimes-0-right, itimes-fix-int1, itimes-fix-int2, commutativity-of-itimes, itimes-distributes-over-iplus-proof, itimes-distributes-over-iplus, commutativity2-of-itimes, associativity-of-itimes, equal-itimes-0, equal-itimes-1, equal-itimes-minus-1, itimes-1-arg1, quotient-remainder-uniqueness, division-theorem, itimes-ineg-1, itimes-ineg-2, itimes-cancellation-1, itimes-cancellation-2, itimes-cancellation-3, integerp-iquotient, integerp-iremainder, integerp-idiv, integerp-imod, integerp-iquo, integerp-irem, iquotient-fix-int1, iquotient-fix-int2, iremainder-fix-int1, iremainder-fix-int2, idiv-fix-int1, idiv-fix-int2, imod-fix-int1, imod-fix-int2, iquo-fix-int1, iquo-fix-int2, irem-fix-int1, irem-fix-int2, fix-int-iquotient, fix-int-iremainder, fix-int-idiv, fix-int-imod, fix-int-iquo, fix-int-irem, itimes-list, itimes-list-append, member-append, equal-fix-int, subsetp, correctness-of-cancel-itimes, correctness-of-cancel-itimes-ilessp, ilessp-strict, eval$-list-cons, eval$-list-nlistp, eval$-litatom, eval$-quote, eval$-other, iplus-x-y-ineg-x, correctness-of-cancel-ineg, integerp-iplus-list, eval$-iplus-list-delete, eval$-iplus-list-bagdiff, itimes-tree-ineg, itimes-factors, itimes–1, equal-ineg-ineg, ilessp-ineg-ineg, fix-int-eval$-itimes-tree-rec, eval$-itimes-tree-ineg, ineg-eval$-itimes-tree-ineg, iplus-eval$-itimes-tree-ineg, itimes-eval$-itimes-tree-ineg, iplus-or-itimes-term, cancel-itimes-factors, cancel-itimes-factors-expanded, cancel-itimes-factors-expanded-cancel-itimes-factors, membership-of-0-implies-itimes-list-is-0, member-0-eval$-list, correctness-of-cancel-itimes-factors, cancel-itimes-ilessp-factors, bagint-singleton, ilessp-itimes-list-eval$-list-bagdiff, ilessp-itimes-list-eval$-list-bagdiff-corollary-1, member-0-itimes-factors-yields-0, member-0-itimes-factors-yields-0-ilessp-consequence-1, member-0-itimes-factors-yields-0-ilessp-consequence-2, ilessp-itimes-list-eval$-list-bagdiff-corollary-2, correctness-of-cancel-itimes-ilessp-factors, disjoin-equalities-with-0, cancel-factors-0, some-eval$s-to-0, eval$-disjoin-equalities-with-0, some-eval$s-to-0-append, some-eval$s-to-0-eliminator, correctness-of-cancel-factors-0, conjoin-inequalities-with-0, cancel-factors-ilessp-0, split-out-ineg-terms, correctness-of-cancel-ineg-terms-from-equality, correctness-of-cancel-ineg-terms-from-inequality, iplus-constants, cancel-constants-equal, ilessp-add1, ilessp-add1-iplus, cancel-constants-ilessp.

Event: Make the library `"integers"` and compile it.

Index