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
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

3
(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")

---

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

**Definition:**
integerp(x)
= if x ∈ N then t
elseif negativep(x) then negative-guts(x) ≠ 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)
Definition: $ilessp(i, j) = \begin{cases} 
\text{if negativep}(i) & \text{then if negativep}(j) \text{ then negative-guts}(j) < \text{negative-guts}(i) \\
& \text{elseif } i = (-0) \text{ then } 0 < j \\
& \text{else } t \text{ endif} \\
\text{elseif negativep}(j) & \text{then } f \\
& \text{else } i < j \text{ endif} 
\end{cases}$

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

Definition: $iplus(x, y) = \begin{cases} 
\text{if negativep}(x) & \text{then if negativep}(y) \\
& \text{then if (negative-guts}(x) \simeq 0) \land (\text{negative-guts}(y) \simeq 0) \text{ then } 0 \\
& \text{else } (\text{negative-guts}(x) + \text{negative-guts}(y)) \text{ endif} \\
& \text{elseif } y < \text{negative-guts}(x) \text{ then } (\text{negative-guts}(x) - y) \\
& \text{else } y - \text{negative-guts}(x) \text{ endif} \\
\text{elseif negativep}(y) & \text{then if } x < \text{negative-guts}(y) \text{ then } (\text{negative-guts}(y) - x) \\
& \text{else } x - \text{negative-guts}(y) \text{ endif} \\
\text{else } x + y \text{ endif} 
\end{cases}$

Definition: $ineg(x) = \begin{cases} 
\text{if negativep}(x) & \text{then negative-guts}(x) \\
\text{elseif } x \simeq 0 & \text{then } 0 \\
\text{else } -x \text{ endif} 
\end{cases}$

Definition: $idifference(x, y) = iplus(x, ineg(y))$

Definition: $iabs(i) = \begin{cases} 
\text{if negativep}(i) & \text{then negative-guts}(i) \\
\text{else fix}(i) \text{ endif} 
\end{cases}$

| old version: |
| (defn izerop (i) |
| (if (integerp i) |
| (equal i 0) |
| t)) |
| # |
\textbf{Definition:} \\
\texttt{itimes (i, j)} \\
\hspace{1em} = \begin{array}{l}
\quad \text{if negativep (i)} \\\n\quad \quad \text{then if negativep (j) then negative-guts (i) \times negative-guts (j)} \\\n\quad \quad \quad \quad \text{else fix-int (- (negative-guts (i) \times j)) endif} \\\n\quad \quad \text{elseif negativep (j) then fix-int (- (i \times negative-guts (j)))} \\\n\quad \quad \text{else } i \times j \text{ endif} \\
\end{array}

\textbf{Definition:} \\
\texttt{iquotient (i, j)} \\
\hspace{1em} = \begin{array}{l}
\quad \text{if fix-int (j) = 0 then 0} \\\n\quad \text{elseif negativep (i)} \\\n\quad \quad \text{then if negativep (j)} \\\n\quad \quad \quad \text{then if (negative-guts (i) \ mod negative-guts (j)) = 0} \\\n\quad \quad \quad \quad \text{then negative-guts (i) \div negative-guts (j)} \\\n\quad \quad \quad \quad \text{else 1 + (negative-guts (i) \div negative-guts (j)) endif} \\\n\quad \quad \text{elseif (negative-guts (i) \ mod j) = 0} \\\n\quad \quad \quad \text{then fix-int (- (negative-guts (i) \div j))} \\\n\quad \quad \quad \text{else fix-int (- (1 + (negative-guts (i) \div j))) endif} \\\n\quad \quad \text{elseif negativep (j)} \\\n\quad \quad \quad \text{then fix-int (- (i \div negative-guts (j)))} \\\n\quad \quad \text{else i \div j endif} \\
\end{array}

\textbf{Definition:} \\
\texttt{iremainder (i, j)} = \text{idifference (i, \texttt{itimes (j, iquotient (i, j))}}

\textbf{Definition:} \\
\texttt{idiv (i, j)} \\
\hspace{1em} = \begin{array}{l}
\quad \text{if fix-int (j) = 0 then 0} \\\n\quad \text{elseif negativep (i)} \\\n\quad \quad \text{then if negativep (j)} \\\n\quad \quad \quad \text{then if (i \ mod negative-guts (j)) = 0} \\\n\quad \quad \quad \quad \text{then fix-int (- (i \div negative-guts (j)))} \\\n\quad \quad \quad \quad \text{else fix-int (- (1 + (i \div negative-guts (j))) endif} \\\n\quad \quad \text{elseif (i \ mod j) = 0} \\\n\quad \quad \quad \text{then fix-int (- (i \div j))} \\\n\quad \quad \quad \text{else fix-int (- (1 + (i \div j))) endif} \\\n\quad \text{elseif negativep (j)} \\\n\quad \quad \text{then fix-int (- (i \div negative-guts (j)))} \\\n\quad \quad \text{else i \div j endif} \\
\end{array}

\textbf{Definition:} \\
\texttt{imod (i, j)} = \text{idifference (fix-int (i), \texttt{itimes (j, iquotient (i, j))}}

\textbf{Definition:} \\
\texttt{iquo (i, j)} \\
\hspace{1em} = \begin{array}{l}
\quad \text{if fix-int (j) = 0 then 0} \\
\end{array}
elseif 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:
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))

;; 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) → (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)
THEOREM: ineg-iplus
ineg (iplus (a, b)) = iplus (ineg (a), ineg (b))

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

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

THEOREM: ineg-of-non-integerp
(¬ integerp (x)) → (ineg (x) = 0)

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

EVENT: Disable ineg-of-non-integerp.

THEOREM: ineg-0
ineg (0) = 0

; -------- IPLUS --------

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

THEOREM: iplus-left-id
(¬ integerp (x)) → (iplus (x, y) = fix-int (y))

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

EVENT: Disable iplus-left-id.

THEOREM: iplus-right-id
(¬ integerp (y)) → (iplus (x, y) = fix-int (x))

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

EVENT: Disable iplus-right-id.

THEOREM: iplus-0-left
iplus (0, x) = fix-int (x)
Theorem: iplus-0-right
\[ \text{iplus} (x, 0) = \text{fix-int} (x) \]

Theorem: commutativity2-of-iplus
\[ \text{iplus} (x, \text{iplus} (y, z)) = \text{iplus} (y, \text{iplus} (x, z)) \]

Theorem: commutativity-of-iplus
\[ \text{iplus} (x, y) = \text{iplus} (y, x) \]

Theorem: associativity-of-iplus
\[ \text{iplus} (\text{iplus} (x, y), z) = \text{iplus} (x, \text{iplus} (y, z)) \]

Theorem: iplus-cancellation-1
\[ (\text{iplus} (a, b) = \text{iplus} (a, c)) = (\text{fix-int} (b) = \text{fix-int} (c)) \]

Theorem: iplus-cancellation-2
\[ (\text{iplus} (b, a) = \text{iplus} (c, a)) = (\text{fix-int} (b) = \text{fix-int} (c)) \]

Theorem: iplus-ineg1
\[ \text{iplus} (\text{ineg} (a), a) = 0 \]

Theorem: iplus-ineg2
\[ \text{iplus} (a, \text{ineg} (a)) = 0 \]

Theorem: iplus-fix-int1
\[ \text{iplus} (\text{fix-int} (a), b) = \text{iplus} (a, b) \]

Theorem: iplus-fix-int2
\[ \text{iplus} (a, \text{fix-int} (b)) = \text{iplus} (a, b) \]

; -------------- IDIFFERENCE --------------

;; mostly omitted, but I’ll keep a few

Theorem: idifference-fix-int1
\[ \text{idifference} (\text{fix-int} (a), b) = \text{idifference} (a, b) \]

Theorem: idifference-fix-int2
\[ \text{idifference} (a, \text{fix-int} (b)) = \text{idifference} (a, b) \]

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

; Theorem: eval$-list-cons
eval$ ("list", cons (x, y), a) = cons (eval$ (t, x, a), eval$ ('list, y, a))

; Theorem: eval$-list-nlistp
(x \cong nil) \rightarrow (eval$ ('list, x, a) = nil)

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

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

13
(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$-other
((- litatom (x)) ∧ (x ≃ nil)) → (eval$ (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),
       else 0 endif endlet endif

14
eval$\text{-cancel-ineg-aux-fn}(x, y^2, a) \) 

else 0 endif endlet endif endif 

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

Theorem: iplus-ineg3 
iplus(\text{ineg}(x), \text{iplus}(x, y)) = \text{fix-int}(y) 

Theorem: iplus-ineg4 
iplus(x, \text{iplus}(\text{ineg}(x), y)) = \text{fix-int}(y) 

Theorem: iplus-ineg-promote 
iplus(y, \text{ineg}(x)) = \text{iplus}(\text{ineg}(x), y) 

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

Event: Disable iplus-ineg-promote. 

Theorem: correctness-of-cancel-ineg-aux 
\(\text{cancel-ineg-aux}(x, y) \neq f \)
\(\rightarrow \ (\text{eval$}\text{-cancel-ineg-aux-fn}(x, y, a) = \text{eval$\text{-cancel-ineg-aux-fn}(x, y, a)}) \)

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


; --- 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)) \to (count (cdr (x)) < count (x))

Definition:
iplus-tree-rec (l) = if cdr (l) \simeq 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$-iplus-tree
eval$ (t, iplus-tree (x), a) = iplus-list (eval$ ('list, x, a))

Theorem: eval$-list-append
eval$ ('list, append (x, y), a) = append (eval$ ('list, x, a), eval$ ('list, y, a))

# The function below has no AND or OR, for efficiency
(defun 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 (caaddr 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))

DEcLI1ION:

cancel-iplus (x)
 =  if listp (x)
       then if car (x) = 'equal
               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 (cadr (x))),
                                               bagint (iplus-fringe (cadr (x))),
                                               iplus-fringe (caddr (x))))))
               elseif caddr (x) ∈ iplus-fringe (cadr (x))
       then list ('if,
                 list ('integerp, caddr (x)),
                 cons ('equal,
                 cons (iplus-tree (delete (caddr (x)),
                              iplus-fringe (caddr (x)))),
                      '('0)),)
                 list ('quote, f))
       else x endif
     elseif caddr (x) ∈ iplus-fringe (cadr (x))
     then list ('if,
               list ('integerp, caddr (x)),
               cons ('equal,
               cons (iplus-tree (delete (caddr (x)),
                              iplus-fringe (caddr (x)))),
                      '('0)),)
               list ('quote, f))
     else x endif
   elseif listp (caddr (x))
else x endif
then if \( \text{caaddr}(x) = \text{'iplus} \)
\[ \text{then if } \text{cadr}(x) \in \text{iplus-fringe(caddr}(x)) \]
\[ \text{then list('if,} \]
\[ \text{list('integerp, cadr}(x)), \]
\[ \text{list('equal,} \]
\[ \text{''0}, \]
\[ \text{iplus-tree(delete(cadr}(x),} \]
\[ \text{iplus-fringe(caddr}(x))))\), \]
\[ \text{list('quote, f))} \]
\[ \text{else } x \text{ endif} \]
\[ \text{else } x \text{ endif} \]
\[ \text{else } x \text{ endif} \]
\[ \text{else } x \text{ endif} \]
\[ \text{else } x \text{ endif} \]
\[ \text{else } x \text{ endif} \]
\[ \text{else } x \text{ endif} \]
\[ \text{else } x \text{ endif} \]
\[ \text{else } x \text{ endif} \]

**Theorem**: eval$\$-cancel-iplus
\[
\text{eval$\$(t, cancel-iplus(x), a) = if listp}(x) \land (\text{car}(x) = \text{'equal) then if listp}(\text{cadr}(x)) \land (\text{caadr}(x) = \text{'iplus) then listp}(\text{caddr}(x)) \land (\text{caaddr}(x) = \text{'iplus) then let } xs \text{ be } \text{iplus-fringe(cadr}(x)), \]
\[ ys \text{ be } \text{iplus-fringe(caddr}(x)) \]
in
\[ \text{let bagint be } \text{bagint}(xs, ys) \]
in
\[ \text{if listp}(bagint) \text{ then iplus-list}(\text{eval$\$('}list,} \]
\[ \text{bagdiff}(xs,} \]
\[ \text{bagint}(xs, ys)), \]
\[ 19 \]
\[
\text{iplus-list (eval$ ('list, bagdiff (ys, bagint (xs, ys))), a))}\\
\text{else eval$ (t, x, a) endif endlet endlet}
\]

\text{elseif listp (cadr (x))}\\
\text{∧ (caadr (x) = 'iplus)\\
∧ (caddr (x) ∈ iplus-fringe (cadr (x)))\\
then if integerp (eval$ (t, cadr (x), a))\\
\text{then iplus-list (eval$ ('list, delete (caddr (x), iplus-fringe (cadr (x))), a))}\\
\text{= 0\\
else f endif\\
else f endif\\
\text{elseif listp (caddr (x))}\\
\text{∧ (caaddr (x) = 'iplus)\\
∧ (cadr (x) ∈ iplus-fringe (caddr (x)))\\
then if integerp (eval$ (t, cadr (x), a))\\
\text{then 0 = iplus-list (eval$ ('list, delete (cadr (x), iplus-fringe (caddr (x))), a))}\\
\text{else f endif\\
else eval$ (t, x, a) endif\\
else eval$ (t, x, a) endif\\
\]

**Event:** Disable cancel-iplus.

**Theorem:** eval$-iplus-list-delete\\
\text{(z ∈ y)}\\
\rightarrow (\text{iplus-list (eval$ ('list, delete (z, y), a))}\\
\text{= idifference (iplus-list (eval$ ('list, y, a)), eval$ (t, z, a)))}

**Theorem:** eval$-iplus-list-bagdiff\\
\text{subbagp (x, y)}\\
\rightarrow (\text{iplus-list (eval$ ('list, bagdiff (y, x), a))}\\
\text{= 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.
Theorem: not-integerp-implies-not-equal-iplus
\((\neg \text{integerp}(a)) \rightarrow ((a = \text{iplus}(b, c)) = \text{f})\)

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

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

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

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

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

Event: Disable iplus-ineg5-lemma-1.

Event: Disable iplus-ineg5-lemma-2.

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

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

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

Theorem: correctness-of-cancel-iplus
\(\text{eval$}(t, x, a) = \text{eval$}(t, \text{cancel-iplus}(x), a)\)
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
(defun 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-INESSP-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)) = illessp (b, c)
Theorem: iplus-cancellation-2-for-ilessp
ilessp (iplus (b, a), iplus (c, a)) = illessp (b, c)
Theorem: correctness-of-cancel-iplus-ilessp-lemma
eval$ (t, x, a) = eval$ (t, cancel-iplus-ilessp-1 (x), a)

Definition:
iliplus-tree-no-fix-int (l)
= if listp (l) then iliplus-tree-rec (l)
else '0 endif

Theorem: eval$-ilessp-iliplus-tree-no-fix-int
ilessp (eval$ (t, iliplus-tree-no-fix-int (x), a),
      eval$ (t, iliplus-tree-no-fix-int (y), a))
= illessp (eval$ (t, iliplus-tree (x), a), eval$ (t, iliplus-tree (y), a))

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

Theorem: make-cancel-iplus-INESSP-simplifier
eval$ (t, make-cancel-iplus-INESSP-1 (x, y), a)
= eval$ (t,
    list ('ilessp,
      iliplus-tree-no-fix-int (bagdiff (x, bagint (x, y))),
      iliplus-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)
          (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$ (t, x, a) = eval$ (t, cancel-iplus-ilessp (x), a)
; ---------- Multiplication ----------

THEOREM: itimes-zero1
(fix-int \( x \) = 0) \( \rightarrow \) (itimes \( x, y \) = 0)

THEOREM: itimes-0-left
itimes \( 0, y \) = 0

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

EVENT: Disable itimes-zero1.

THEOREM: itimes-zero2
(fix-int \( y \) = 0) \( \rightarrow \) (itimes \( x, y \) = 0)

THEOREM: itimes-0-right
itimes \( x, 0 \) = 0

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

EVENT: Disable itimes-zero2.

THEOREM: itimes-fix-int1
itimes (fix-int \( a \), \( b \)) = itimes \( a, b \)

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

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

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

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

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

THEOREM: associativity-of-itimes
itimes (itimes \( x, y \), \( z \)) = itimes \( x, \text{itimes} (y, z) \)
Theorem: equal-itimes-0
\( \text{itimes}(x, y) = 0 = ((\text{fix-int}(x) = 0) \lor (\text{fix-int}(y) = 0)) \)

Theorem: equal-itimes-1
\( \text{itimes}(a, b) = 1 \\
= ((a = 1) \land (b = 1)) \lor ((a = -1) \land (b = -1)) \)

Theorem: equal-itimes-minus-1
\( \text{itimes}(a, b) = -1 \\
= ((a = -1) \land (b = 1)) \lor ((a = 1) \land (b = -1)) \)

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

; ---------- Division ----------

Theorem: quotient-remainder-uniqueness
\( ((a = (r + (b \ast q))) \land (r < b)) \\
\rightarrow ((\text{fix}(r) = (a \mod b)) \land (\text{fix}(q) = (a \div b))) \)

; 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 \( i = jq + r, \ 0 \leq r < |j|. \)
;
; 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
\( (\text{integerp}(j) \land (j \neq 0)) \rightarrow (\neg \text{ilessp}(\text{iremainder}(i, j), 0)) \)

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

Theorem: division-theorem
\( (\text{integerp}(i) \land \text{integerp}(j) \land (j \neq 0)) \\
\rightarrow ((\text{iplus}(\text{iremainder}(i, j), \text{itimes}(j, \text{iquotient}(i, j))) = i) \\
\land (\neg \text{ilessp}(\text{iremainder}(i, j), 0)) \\
\land \text{ilessp}(\text{iremainder}(i, j), \text{iabs}(j))) \)
Theorem: quotient-difference-lessp-arg2

\[(a \mod c) = 0 \land (b < c)\]
\[\rightarrow (a - b) \div c\]
\[= \begin{cases} a \div c & \text{if } b \simeq 0 \\ (a \div c) - (1 + (b \div c)) & \text{else} \end{cases}\]

Theorem: iquotient-iremainder-uniqueness

\[\text{integerp } (i) \land \text{integerp } (j) \land \text{integerp } (r) \land \text{integerp } (q) \land (j \neq 0) \land (i = \text{iplus } (r, \text{itimes } (j, q))) \land (\neg \text{ilessp } (r, 0)) \land \text{iessp } (r, \text{iabs } (j))\]
\[\rightarrow ((r = \text{iremainder } (i, j)) \land (q = \text{iquotient } (i, j)))\]

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
; \[i = jq + r, \quad 0 \leq r < j \quad (j > 0)\]
; \[j < r \leq 0 \quad (j < 0)\]
; 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

\[\text{integerp } (i) \rightarrow (\text{iplus } (\text{imod } (i, j), \text{itimes } (j, \text{idiv } (i, j)))) = i\]

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

\[\text{iessp } (0, j) \rightarrow (((\neg \text{ilessp } (\text{imod } (i, j), 0)) \land \text{iessp } (\text{imod } (i, j), j))\]

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

\[\text{integerp } (j) \land \text{iessp } (j, 0)) \rightarrow (((\neg \text{iessp } (0, \text{imod } (i, j))) \land \text{iessp } (j, \text{imod } (i, j)))\]
Theorem: division-theorem-for-truncate-to-neginf
\[(\text{integerp}(i) \land \text{integerp}(j) \land (j \neq 0)) \rightarrow ((\text{iplus}(\text{imod}(i, j), \text{itimes}(j, \text{idiv}(i, j)))) = i) \land \begin{cases} \neg \text{ilessp}(0, j) \\ \text{if} \ \text{ilessp}(\text{imod}(i, j), 0) \land \text{ilessp}(\text{imod}(i, j), j) \\ \text{else} (\neg \text{ilessp}(0, \text{imod}(i, j))) \\ \land \text{ilessp}(j, \text{imod}(i, j)) \end{cases} \text{endif} \]

Theorem: idiv-imod-uniqueness
\[(\text{integerp}(i) \land \text{integerp}(j) \land \text{integerp}(r) \land \text{integerp}(q) \land (j \neq 0) \land ((i = \text{iplus}(r, \text{itimes}(j, q))) \land \text{if} \ \text{ilessp}(0, j) \text{then} (\neg \text{ilessp}(r, 0)) \land \text{ilessp}(r, j) \text{else} (\neg \text{ilessp}(0, r)) \land \text{ilessp}(j, r) \text{endif}) \rightarrow ((r = \text{imod}(i, j)) \land (q = \text{idiv}(i, j))) \]

; Division Theorem (truncate towards zero variant):
;
; For all integers i, j, j not 0, there exist unique integers q and r
; which satisfy
;
; \[i = jq + r, \quad 0 \leq r < |j| \quad (i \geq 0)\]
;
; \[\neg |j| < r \leq 0 \quad (i < 0)\]
;
; 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
\[\text{integerp}(i) \rightarrow (\text{iplus}(\text{irem}(i, j), \text{itimes}(j, \text{iquo}(i, j)))) = i)\]

Theorem: division-theorem-for-truncate-to-zero-part2
\[(\text{integerp}(i) \land \text{integerp}(j) \land (j \neq 0) \land (\neg \text{ilessp}(i, 0))) \rightarrow ((\neg \text{ilessp}(\text{irem}(i, j), 0)) \land \text{ilessp}(\text{irem}(i, j), \text{labs}(j)))\]

Theorem: division-theorem-for-truncate-to-zero-part3
\[(\text{integerp}(i) \land \text{integerp}(j) \land (j \neq 0) \land \text{ilessp}(i, 0)) \rightarrow ((\neg \text{ilessp}(0, \text{irem}(i, j))) \land \text{ilessp}(\text{ineg}(\text{labs}(j)), \text{irem}(i, j)))\]
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:
\[
\text{itimes-fringe}(x) = 
\begin{cases} 
  \text{if listp}(x) \\
   \text{then if car}(x) = \text{'itimes} \\
   \text{then append(itimes-fringe(cadr(x)), itimes-fringe(caddr(x)))} \\
   \text{else list}(x) \text{ endif} \\
   \text{else list}(x) \text{ endif} 
\end{cases}
\]

### Definition:
\[
\text{itimes-tree-rec}(l) = 
\begin{cases} 
  \text{if cdr}(l) \approx \text{nil} \text{ then car}(l) \\
   \text{else list('itimes, car(l), itimes-tree-rec(cdr(l)))} \text{ endif} 
\end{cases}
\]

### Definition:
\[
\text{itimes-tree}(l) = 
\begin{cases} 
  \text{if listp}(l) \\
   \text{then if listp(cdr(l)) then itimes-tree-rec(l) } \\
   \text{else list('fix-int, car(l)) endif} \\
   \text{else '}'1} \text{ endif} 
\end{cases}
\]

### Definition:
\[
\text{itimes-list}(x) = 
\begin{cases} 
  \text{if listp}(x) \text{ then itimes(car(x), itimes-list(cdr(x)))} \\
   \text{else 1} \text{ endif} 
\end{cases}
\]

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

Theorem: eval$-itimes-tree-rec
listp(x)
→ (eval$ (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)

\begin{verbatim}
(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) (cadadr x))
(make-cancel-itimes-equality (list (cadr x))
(itimes-fringe (caddr x))
(list (cadr x)))
(list (quote quote) f))
x))
x))
|#

Definition:
\text{cancel-itimes}\ (x) = \begin{array}{l}
\text{if } \text{listp}\ (x)\\
\text{then if } \text{car}\ (x) = \text{’equal}\\
\text{then if } \text{listp}\ (\text{cadr}\ (x))\\
\text{then if } \text{caadr}\ (x) = \text{’itimes}\\
\text{then if } \text{listp}\ (\text{cadadr}\ (x))\\
\text{then if } \text{cadadr}\ (x) = \text{’itimes}\\
\text{then if } \text{listp}\ (\text{bagint}\ (\text{itimes-fringe} (\text{cadadr}\ (x))),\\
\text{itimes-fringe} (\text{cadadr}\ (x))))\\
\text{then } \text{make-cancel-itimes-equality}\ (\text{itimes-fringe}\ (\text{cadadr}\ (x)),\\
\text{itimes-fringe} (\text{cadadr}\ (x)),\\
\text{bagint}\ (\text{itimes-fringe}\ (\text{cadadr}\ (x))),\\
\text{itimes-fringe} (\text{cadadr}\ (x))))\\
\text{else } x \text{ endif}\\
\text{elseif } \text{cadadr}\ (x) \in \text{itimes-fringe}\ (\text{cadadr}\ (x))\\
\text{then list}(’\text{if},\\
\text{list}(’\text{integerp}, \text{cadadr}\ (x)),\\
\text{make-cancel-itimes-equality} (\text{itimes-fringe}\ (\text{cadadr}\ (x)),\\
\text{list}\ (\text{cadadr}\ (x)),\\
\text{list}\ (\text{cadadr}\ (x))),\\
\text{list}(’\text{quote}, f))\\
\text{else } x \text{ endif}
\end{array}
\end{verbatim}

34
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 (caddr (x))),
      list ('quote, f))
    else x endif
  else x endif
else x endif
else x endif
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 (caddr (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-evall$-fringe
itimes-list (evall$ ('list, itimes-fringe (x), a)) = fix-int (evall$ (t, x, a))

Theorem: integerp-evall$-itimes
(car (x) = 'itimes) → integerp (evall$ (t, x, a))

Theorem: not-integerp-implies-not-equal-itimes
(¬ integerp (a)) → ((a = itimes (b, c)) = f)

35
THEOREM: itimes-list-eval$-delete
\((z \in y)\)
\rightarrow (\text{itimes-list}(\text{eval$('list', y, a)}))
= \text{itimes(\text{eval$('t, z, a)}, \text{itimes-list(\text{eval$('list, delete(z, y), a)})})\)

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

THEOREM: equal-itimes-list-eval$-list-delete
\((c \in y) \land (\text{fix-int(\text{eval$('t, c, a)})} \neq 0))\)
\rightarrow ((x = \text{itimes-list(\text{eval$('list, delete(c, y), a)})})
\land \text{integerp}(x)
\land (\text{itimes(x, \text{eval$('t, c, a)})})
= \text{itimes-list(\text{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
\((a \in \text{append}(x, y)) = ((a \in x) \lor (a \in y))\)

THEOREM: member-izerop-itimes-fringe
\((z \in \text{itimes-fringe}(x)) \land (\text{fix-int(\text{eval$('t, z, a)}) = 0})\)
\rightarrow (\text{fix-int(\text{eval$('t, x, a)}) = 0})\)

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

EVENT: Enable eval$-equal.

THEOREM: eval$-make-cancel-itimes-equality
\text{eval$('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)
    ∧ (car (x) = 'itimes))
→ (eval$(t, x, a) = 0)

THEOREM: izerop-eval-of-member-implies-itimes-list-0
((z ∈ y) ∧ (fix-int (eval$(t, z, a)) = 0))
→ (itimes-list (eval$ ('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 \(x \simeq \text{nil}\) then \(t\)
  elseif \(\text{car}(x) \in y\) then subsetp(cdr\((x)\), y)
  else \(f\) endif

**Theorem:** subsetp-implies-itimes-list-eval$\text{-evals-0}$
\((\text{subsetp}\((x, y)) \land (\text{itimes-list}\((\text{'list}, x, a)) = 0))\)
→ (\(\text{itimes-list}\((\text{'list}, y, a)) = 0)\)

**Theorem:** subbagp-subsetp
subbagp\((x, y)\) → subsetp\((x, y)\)

**Theorem:** equal-0-itimes-list-eval$\text{-bagint-1}$
(\(\text{itimes-list}\((\text{'list}, \text{bagint}(x, y, a)) = 0)\)
→ (\(\text{itimes-list}\((\text{'list}, x, a)) = 0)\)

**Theorem:** equal-0-itimes-list-eval$\text{-bagint-2}$
(\(\text{itimes-list}\((\text{'list}, \text{bagint}(x, y, a)) = 0)\)
→ (\(\text{itimes-list}\((\text{'list}, y, a)) = 0)\)

**Theorem:** correctness-of-cancel-itimes-hack-2
\((\text{listp}(u)\)
  ∧ (\(\text{car}(u) = \text{'itimes}\))
  ∧ \(\text{listp}(v)\)
  ∧ (\(\text{car}(v) = \text{'itimes}\))
  ∧ (\(\text{eval}((t, u, a)) \neq \text{eval}((t, v, a))\))
→ (\(\text{itimes-list}\((\text{'list},
  \text{bagdiff}(\text{itimes-fringe}(u),
  \text{bagint}(\text{itimes-fringe}(u), \text{itimes-fringe}(v))),
  a)\)
  \neq \(\text{itimes-list}\((\text{'list},
  \text{bagdiff}(\text{itimes-fringe}(v),
  \text{bagint}(\text{itimes-fringe}(u),
  \text{itimes-fringe}(v))),
  a)))\))

**Theorem:** correctness-of-cancel-itimes-hack-3-lemma
\(((u = \text{itimes}(a, b)) \land (\text{fix-int}(a) \neq 0))\)
→ (\((u = \text{itimes}(a, c)) = (\text{fix-int}(b) = \text{fix-int}(c))\))

**Theorem:** correctness-of-cancel-itimes-hack-3
\((\text{listp}(u)\)
  ∧ (\(\text{car}(u) = \text{'itimes}\))
  ∧ \(\text{listp}(v)\)
∧ (\text{car}(v) = \text{'}\text{itimes}')
∧ (\text{eval}(t, u, a) = \text{eval}(t, v, a))
∧ (\neg \text{eval}(t,
      \text{list}(\text{'}\text{equal'},
      \text{itimes-tree}(\text{bagint}(\text{itimes-fringe}(u), \text{itimes-fringe}(v)),
      ',0),
   a))))
\rightarrow ((\text{itimes-list(\text{eval}(\text{'}\text{list'},
      \text{bagdiff}(\text{itimes-fringe}(u),
      \text{bagint}(\text{itimes-fringe}(u), \text{itimes-fringe}(v))),
   a))
   = \text{itimes-list(\text{eval}(\text{'}\text{list'},
      \text{bagdiff}(\text{itimes-fringe}(v),
      \text{bagint}(\text{itimes-fringe}(u), \text{itimes-fringe}(v))),
   a))))
   = t)

\textbf{EVENT:} Disable correctness-of-cancel-itimes-hack-3-lemma.

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

; ---------- Meta lemma for itimes cancellation on \text{ilesssp} terms

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

;; EVAL$\text{-EQUAL}$ is currently enabled, but that's OK.

\textbf{DEFINITION:}
\text{itimes-tree-no-fix-int}(l)
= \text{if listp}(l) \text{ then itimes-tree-rec}(l) \text{ else '1 endif}

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

\textbf{THEOREM:} eval$\text{-itimes-tree-no-fix-int-1}$
\text{ilesssp}(\text{eval}(t, \text{itimes-tree-no-fix-int}(x), a), y)
= \text{ilesssp}(\text{eval}(t, \text{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
(defun 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)),

40
Theorem: eval$-make-cancel-itimes-inequality

\[ \text{eval$ (t, \text{make-cancel-itimes-inequality} (x, y, \text{in-both}, a))} \]
\[ = \begin{cases} \text{if eval$ (t, \text{list} ('\text{ilessp}, \text{itimes-tree-no-fix-int} (in-both), '1, 0, a))} \begin{cases} \text{then} \text{ilessp (eval$ (t, \text{itimes-tree-no-fix-int} (\text{bagdiff} (y, in-both), a)), eval$ (t, \text{itimes-tree-no-fix-int} (\text{bagdiff} (x, in-both), a)))} \end{cases} \end{cases} \]
\[ \text{else eval$ (t, \text{list} ('\text{ilessp}, '0, \text{itimes-tree-no-fix-int} (in-both), a))} \begin{cases} \text{then} \text{ilessp (eval$ (t, \text{itimes-tree-no-fix-int} (\text{bagdiff} (x, in-both), a)), eval$ (t, \text{itimes-tree-no-fix-int} (\text{bagdiff} (y, in-both), a)))} \end{cases} \] \[ \text{else} \text{f endif} \]

Event: Disable make-cancel-itimes-inequality.

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

\[ \text{listp (bagint (y, list(z)))} \rightarrow (z \in y) \]

Theorem: itimes-list-eval$-list-0

\[ (0 \in x) \rightarrow (\text{itimes-list (eval$ ('\text{list}, x, a))} = 0) \]

Theorem: ilessp-itimes-right-positive

\[ \text{ilessp (0, x)} \rightarrow (\text{ilessp (y, z)} = \text{ilessp (itimes (y, x), itimes (z, x))}) \]

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

\[ \text{(subbagp (bag, x))} \land (\text{subbagp (bag, y)}) \]
\[ \land \text{ilessp (0, \text{itimes-list (eval$ ('\text{list}, bag, a)))}) \rightarrow (\text{ilessp (itimes-list (eval$ ('\text{list}, \text{bagdiff} (x, bag), a)), itimes-list (eval$ ('\text{list}, \text{bagdiff} (y, bag), a)))}) \]
\[ = \text{ilessp (itimes-list (eval$ ('\text{list}, x, a)), itimes-list (eval$ ('\text{list}, y, a))}) \]

Theorem: listp-bagint-with-singleton-member

\[ \text{listp (bagint (y, list(z)))} = (z \in y) \]

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

\[ (0 \in \text{itimes-fringe (w)}) \rightarrow (\text{eval$ (t, w, a)} = 0) \]

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

\[ (0 \in \text{itimes-fringe (w)}) \rightarrow (\neg \text{ilessp (eval$ (t, w, a)}, 0)) \]

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

THEOREM: ilessp-trichotomy
(¬ ilessp (x, y)) → (ilessp (y, x) = (fix-int (x) ≠ fix-int (y)))

THEOREM: correctness-of-cancel-itimes-ilessp-hack-3-lemma-1
((0 = itimes-list (eval$ (‘list, bag, a))) ∧ subsetp (bag, z))
→ (itimes-list (eval$ (‘list, z, a)) = 0)

THEOREM: correctness-of-cancel-itimes-ilessp-hack-3-lemma-2
((0 = itimes-list (eval$ (‘list, bag, a))) ∧ subsetp (bag, itimes-fringe (x)))
→ (fix-int (eval$ (t, x, a)) = 0)

THEOREM: same-fix-int-implies-not-ilessp
(fix-int (x) = fix-int (y)) → (¬ ilessp (x, y))

THEOREM: correctness-of-cancel-itimes-ilessp-hack-3
(¬ ilessp (itimes-list (eval$ (‘list, bag, a)), 0))
∧ (¬ ilessp (0, itimes-list (eval$ (‘list, bag, a))))
∧ subbagp (bag, itimes-fringe (w))
∧ subbagp (bag, itimes-fringe (v))
→ (¬ ilessp (eval$ (t, w, a), eval$ (t, v, a)))

THEOREM: ilessp-itimes-right-negative
ilessp (x, 0) → (ilessp (y, z) = ilessp (itimes (z, x), itimes (y, x)))

THEOREM: correctness-of-cancel-itimes-ilessp-hack-4
(subbagp (bag, x)
∧ subbagp (bag, y)
∧ ilessp (itimes-list (eval$ (‘list, bag, a)), 0))
→ (ilessp (itimes-list (eval$ (‘list, bagdiff (x, bag), a)),
itimes-list (eval$ (‘list, bagdiff (y, bag), a)))
= ilessp (itimes-list (eval$ (‘list, y, a)),
itimes-list (eval$ (‘list, x, a))))

EVENT: Disable ilessp-trichotomy.

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

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

`ilessp(x, y) → (¬ 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-3-lemma.

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


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-3-lemma-1.


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

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

\[\text{(IMPLIES (AND (NOT (IZEROP X)) (EQUAL (ITIMES A X) (INEG (ITIMES B X)))) (EQUAL (FIX-INT A) (INEG B)))} \]

\[
definition: \text{itimes-tree-ineg} (l) = \begin{cases} \text{if listp} (l) & \begin{cases} \text{if car} (l) = \text{'quote, -1} & \begin{cases} \text{if listp} (\text{cdr} (l)) & \text{list} ('\text{ineg}, \text{itimes-tree-rec} (\text{cdr} (l))) \\
\text{else} & \text{car} (l) \text{ endif}
\end{cases} \\
\text{else} & \text{itimes-tree-rec} (l) \text{ endif}
\end{cases} \\
\text{else} & '\text{1} \text{ endif}
\end{cases}
\]

\[
definition: \text{itimes-factors} (x) = \begin{cases} \text{if listp} (x) & \begin{cases} \text{if car} (x) = '\text{itimes} & \begin{cases} \text{append} (\text{itimes-factors} (\text{cadr} (x)), \text{itimes-factors} (\text{caddr} (x))) \\
\text{elseif} & \text{car} (x) = '\text{iplus} \\
\text{then let} & \text{bag1 be} \text{itimes-factors} (\text{cadr} (x)), \\
& \text{bag2 be} \text{itimes-factors} (\text{caddr} (x)) \\
& \text{in} \\
& \text{let} \text{inboth be} \text{bagint (bag1, bag2)} \\
& \text{in} \\
& \text{if listp} (\text{inboth}) & \begin{cases} \text{cons} (\text{list} ('\text{iplus}, \\
& \text{itimes-tree-ineg (bagdiff (bag1, inboth))}, \\
& \text{itimes-tree-ineg (bagdiff (bag2,} \\
\end{cases}
\end{cases} \\
\end{cases} \\
\end{cases}
\]
\section*{Theorem: \texttt{itimes-1}}

\begin{verbatim}

\texttt{itimes(-1, x) = ineg(x)}
\end{verbatim}

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

\section*{Theorem: \texttt{equal-ineg-ineg}}

\begin{verbatim}

\texttt{(ineg(x) = ineg(y)) = (fix-int(x) = fix-int(y))}
\end{verbatim}

\section*{Theorem: \texttt{ilessp-ineg-ineg}}

\begin{verbatim}

\texttt{ilessp(ineg(x), ineg(y)) = ilessp(y, x)}
\end{verbatim}

\section*{Theorem: \texttt{fix-int-evall-\texttt{itimes-tree-rec}}}

\begin{verbatim}

\texttt{listp(x) → (fix-int(evall(t, \texttt{itimes-tree-rec}(x, a)))

\text{= \texttt{itimes-list(evall(’list, x, a))})}
\end{verbatim}

\section*{Theorem: \texttt{evall-\texttt{itimes-tree-ineg}}}

\begin{verbatim}

\texttt{fix-int(evall(t, \texttt{itimes-tree-ineg}(x, a))) = \texttt{itimes-list(evall(’list, x, a))}}
\end{verbatim}

\text{;; Now I want the above lemma to apply, but it doesn’t, so the}
\text{;; following three lemmas are used instead.}

\section*{Theorem: \texttt{ineg-evall-\texttt{itimes-tree-ineg}}}

\begin{verbatim}

\texttt{ineg(evall(t, \texttt{itimes-tree-ineg}(x, a)))

\text{= \texttt{ineg(itimes-list(evall(’list, x, a))})}
\end{verbatim}

\section*{Theorem: \texttt{iplus-evall-\texttt{itimes-tree-ineg}}}

\begin{verbatim}

\texttt{(iplus(evall(t, \texttt{itimes-tree-ineg}(x, a), y)

\text{= \texttt{iplus(itimes-list(evall(’list, x, a)), y))}}
\end{verbatim}

\text{∧ \texttt{(iplus(y, evall(t, \texttt{itimes-tree-ineg}(x, a))

\text{= \texttt{iplus(y, itimes-list(evall(’list, x, a))}}))}}

\section*{Theorem: \texttt{itimes-evall-\texttt{itimes-tree-ineg}}}

\begin{verbatim}

\texttt{(itimes(evall(t, \texttt{itimes-tree-ineg}(x, a), y)

\text{= \texttt{itimes(itimes-list(evall(’list, x, a)), y))}}
\end{verbatim}

\text{∧ \texttt{(itimes(y, evall(t, \texttt{itimes-tree-ineg}(x, a))

\text{= \texttt{itimes(y, itimes-list(evall(’list, x, a))}}))}}

47
**Event:** Disable itimes-tree-ineg.

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

(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)))))
```

```lisp
;;; 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:**

48
\textbf{Definition:}

\begin{verbatim}
cancel-itimes-factors\(x) = \begin{cases} 
\text{if listp}(x) \land \text{car}(x) = 'equal \\
\text{let bagint be bagint(itimes-factors(cadr(x)), itimes-factors(caddr(x)))} \\
\text{let new-equality be make-cancel-itimes-equality(itimes-factors(cadr(x)), itimes-factors(caddr(x)), bagint)} \\
\text{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)) endif else x endif endlet endlet
\end{cases}
\end{verbatim}

\begin{verbatim}
;; The following was generated with the nqthm-macroexpand macro defined above.
\end{verbatim}
(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 (CADADDR 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 'INTEGRERP (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)))))
)
(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))))))))
(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))))
         (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 'QUOTE F)))
  
  (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)))))

53
(LIST 'IF
  (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))))
  ((LISTP (CADDR X))
    (CASE (CAR (CAR (CDR (CDR X))))
      (IPLUS (IF
        (LISTP
          (BAGINT
            (ITIMES-FACTORS (CADR X))
            (ITIMES-FACTORS (CADDR X))))
        (LIST 'QUOTE F))
        (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))
          (LISTP
            (BAGINT
              (ITIMES-FACTORS (CADR X))
              (ITIMES-FACTORS (CADDR X))))
          (LIST 'QUOTE F))
          (LISTP
            (BAGINT
              (ITIMES-FACTORS (CADR X))
              (ITIMES-FACTORS (CADDR X))))
          (LIST 'QUOTE F)))))
(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)))))
        (T 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)
(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))
        (IF
         (EQUAL (CADADDR 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))
        (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))
        (IF
         (EQUAL (CADADDR 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))
        (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))
        (IF
         (EQUAL (CADADDR 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))
        (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))
        (IF
         (EQUAL (CADADDR 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))
        (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))
        (IF
         (EQUAL (CADADDR 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))
        (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))
        (IF
         (EQUAL (CADADDR 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))
00

57
(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))
 (OTHERWISE 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))
 (ITIMES-FACTORS (CADDR X))))
 (LIST 'QUOTE F))
 X)))
 (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))
 (ITIMES-FACTORS (CADDR X))))
 (LIST 'QUOTE F))
 X))
 (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))
 (ITIMES-FACTORS (CADDR X))))
 (LIST 'QUOTE F))
 X))
 (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))
 (ITIMES-FACTORS (CADDR X))))
 (LIST 'QUOTE F))
 X))
 (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))
 (ITIMES-FACTORS (CADDR X))
(BAGINT
 (ITIMES-FACTORS (CADR X))
 (ITIMES-FACTORS (CADDR 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

\[
(\text{fix-int} (\text{eval$} (t, \text{elt}, a)) \neq 0) \rightarrow (\left(x = \text{itimes-list} (\text{eval$} ('\text{list}, \text{delete} (\text{elt}, \text{bag}), a))\right) = \begin{cases} 
\text{integerp}(x) \\
\land (\text{itimes}(x, \text{eval$} (t, \text{elt}, a)) \\
= \text{itimes-list} (\text{eval$} ('\text{list}, \text{bag}, a))) \\
\text{else} \ x = \text{itimes-list} (\text{eval$} ('\text{list}, \text{bag}, a))) \end{cases}) \end{cases}
\]

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

\[
(\text{fix-int} (\text{eval$} (t, \text{elt}, a)) \neq 0) \rightarrow (\left(\text{itimes-list} (\text{eval$} ('\text{list}, \text{delete} (\text{elt}, \text{bag}), a)) = x\right) = \begin{cases} 
\text{integerp}(x) \\
\land (\text{itimes}(x, \text{eval$} (t, \text{elt}, a)) \\
= \text{itimes-list} (\text{eval$} ('\text{list}, \text{bag}, a))) \\
\text{else} \ x = \text{itimes-list} (\text{eval$} ('\text{list}, \text{bag}, a))) \end{cases}) \end{cases}
\]

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

\[
(x \in \text{bag}) \rightarrow (\text{itimes} (\text{eval$} (t, x, a), \text{itimes-list} (\text{eval$} ('\text{list}, \text{delete} (x, \text{bag}), a))) = \text{itimes-list} (\text{eval$} ('\text{list}, \text{bag}, a)))
\]

Theorem: equal-itimes-list-eval$-list-bagdiff

(subbagp (in-both, bag1)
∧ subbagp (in-both, bag2)
∧ (itimes-list (eval$ ('list, in-both, a) ≠ 0))
→ ((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 ∈ x) → (itimes-list (x) = 0)

THEOREM: member-0-eval$-list
(0 ∈ x) → (0 ∈ eval$ ('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) → integerp (eval$ (t, x, a))

THEOREM: eval$-list-bagint-0
(itimes-list (eval$ ('list, bagint (x, y), a)) = 0)
→ ((itimes-list (eval$ ('list, x, a)) = 0)
  ∧ (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)
  ∧ integerp (eval$ (t, v, a))
  ∧ integerp (eval$ (t, w, a)))
→ ((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 $y$ $\in$ $x$ then list ($y$)
  else nil endif

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

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

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

**Theorem:** ilessp-itimes-list-eval$\$-list-delete
((z $\in$ y) $\land$ (fix-int (eval$\$ ($t$, $z$, $a$)) $\neq$ 0))

$\rightarrow$ (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
ilessp (0, $w$) $\rightarrow$ (ilessp (itimes ($w$, $u$), itimes ($x$, $w$)) = ilessp ($x$, $u$))

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

61
THEOREM: ilessp-itimes-list-eval$-list-delete-prime
((z ∈ y) \land (fix-int (eval$ (t, z, a)) \neq 0))
→ (iessp (itimes-list (eval$ ('list, delete (z, y), a)), x)
    = if iessp (0, eval$ (t, z, a))
      then iessp (itimes-list (eval$ ('list, y, a)), itimes (x, eval$ (t, z, a)))
      elseif iessp (eval$ (t, z, a), 0)
      then iessp (itimes (x, eval$ (t, z, a)), itimes-list (eval$ ('list, y, a)))
      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
;; ;;;;; similarly for ilessp.

THEOREM: ilessp-0-itimes
iessp (0, itimes (x, y))
= ((iessp (0, x) \land iessp (0, y)) \lor (iessp (x, 0) \land iessp (y, 0)))

THEOREM: ilessp-itimes-0
iessp (itimes (x, y), 0)
= ((iessp (0, x) \land iessp (y, 0)) \lor (iessp (x, 0) \land iessp (0, y)))

THEOREM: ilessp-itimes-list-eval$-list-bagdiff
(subbagp (in-both, bag1)
  \land (itimes-list (eval$ ('list, in-both, a)) \neq 0))
→ (iessp (itimes-list (eval$ ('list, bagdiff (bag1, in-both), a)),
             itimes-list (eval$ ('list, bagdiff (bag2, in-both), a)))
    = if iessp (0, itimes-list (eval$ ('list, in-both, a)))
      then iessp (itimes-list (eval$ ('list, bag1, a)),
                    itimes-list (eval$ ('list, bag2, a)))
      else iessp (itimes-list (eval$ ('list, bag2, a)),
                    itimes-list (eval$ ('list, bag1, a))) endif)

THEOREM: zero-iessp-implies-not-equal
iessp (0, x) → (0 \neq x)

THEOREM: ilessp-itimes-list-eval$-list-bagdiff-corollary-1
(subbagp (in-both, bag1)
  \land (itimes-list (eval$ ('list, in-both, a))
  \land iessp (0, itimes-list (eval$ ('list, in-both, a))))
\[ \text{Theorem: } \text{ilessp-zero-implies-not-equal} \\
\text{ilessp}(x, 0) \rightarrow (0 \neq x) \]

\[ \text{Theorem: } \text{ilessp-itimes-list-eval$-list-bagdiff-corollary-2} \\
\text{(subbagp}(in\text{-both}, \text{bag1}) \\
\land \text{(subbagp}(in\text{-both}, \text{bag2}) \\
\land \text{ilessp}(\text{eval$}('\text{list}, \text{in\text{-both}, a}), 0)) \\
\rightarrow (\text{ilessp}(\text{eval$}('\text{list}, \text{bagdiff}(\text{bag1}, \text{in\text{-both}), a)), \\
\text{times-list}(\text{eval$}('\text{list}, \text{bag2}, a))) \\
= \text{ilessp}(\text{times-list}(\text{eval$}('\text{list}, \text{bag1}, a)), \\
\text{times-list}(\text{eval$}('\text{list}, \text{bag2}, a)))) \]

\[ \text{Theorem: } \text{member-0-itimes-factors-yields-0} \\
\text{eval$}(t, w, a) \neq 0 \rightarrow (0 \not\in \text{itimes-factors}(w)) \]

\[ \text{Theorem: } \text{member-0-itimes-factors-yields-0-ilessp-consequence-1} \\
\text{ilessp}(\text{eval$}(t, w, a), 0) \rightarrow (0 \not\in \text{itimes-factors}(w)) \]

\[ \text{Theorem: } \text{member-0-itimes-factors-yields-0-ilessp-consequence-2} \\
\text{ilessp}(0, \text{eval$}(t, w, a)) \rightarrow (0 \not\in \text{itimes-factors}(w)) \]

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

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

63
(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$-list-bagint-0-for-ilessp
((¬ ilessp (itimes-list (eval$ ('list, bagint (x, y), a)), 0))
 ∧ (¬ ilessp (0, itimes-list (eval$ ('list, bagint (x, y), a))))
 → ((fix-int (itimes-list (eval$ ('list, x, a))) = 0)
   ∧ (fix-int (itimes-list (eval$ ('list, y, a))) = 0))

THEOREM: eval$-list-bagint-0-implies-equal-for-ilessp-lemma
((¬ ilessp (itimes-list (eval$ ('list, bagint (itimes-factors (v), itimes-factors (w)),
                                    a), 0))
 ∧ (¬ ilessp (0, itimes-list (eval$ ('list, bagint (itimes-factors (v), itimes-factors (w)),
                                    a))))
 → (fix-int (eval$ (t, v, a)) = fix-int (eval$ (t, w, a)))

THEOREM: equal-fix-int-to-ilessp
(fix-int (x) = fix-int (y)) → (¬ ilessp (x, y))

THEOREM: eval$-list-bagint-0-implies-equal-for-ilessp
((¬ ilessp (itimes-list (eval$ ('list, bagint (itimes-factors (v), itimes-factors (w)),
                                    a), 0))
 ∧ (¬ ilessp (0, itimes-list (eval$ ('list, bagint (itimes-factors (v), itimes-factors (w)),
                                    a)),
\[ (\neg \text{ilessp} (\text{eval} (t, v, a), \text{eval} (t, w, a))) \]
\[ \land (\neg \text{ilessp} (\text{eval} (t, w, a), \text{eval} (t, v, a))) \]

;; 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

\[ \text{eval} (t, x, a) = \text{eval} (t, \text{cancel-itimes-ilessp-factors} (x), a) \]

;; OK -- now, the zero cases.

**EVENT:** Enable lessp-count-listp-cdr.

**DEFINITION:**

disjoin-equalities-with-0 \((factors)\)

\[=\]  
\[\text{if } \text{listp} (\text{cdr} (factors))\]
\[\text{then } \text{list} ('\text{or},\]
\[\text{list} ('\text{equal}, \text{list} ('\text{fix-int}, \text{car} (factors)), '0),\]
\[\text{disjoin-equalities-with-0} (\text{cdr} (factors))\]
\[\text{else } \text{list} ('\text{equal}, \text{list} ('\text{fix-int}, \text{car} (factors)), '0)\text{ endif}\]

**EVENT:** Disable lessp-count-listp-cdr.

**DEFINITION:**
cancel-factors-0 \((x)\)

\[=\]  
\[\text{if } \text{listp} (x)\]
\[\text{then if } \text{car} (x) = '\text{equal}\]
\[\text{then if } \text{cadr} (x) = '0\]
\[\text{then let } factors \text{ be itimes-factors} (\text{caddr} (x))\]
\[\text{in}\]
\[\text{if } \text{listp} (\text{cdr} (factors))\]
\[\text{then } \text{disjoin-equalities-with-0} (factors)\]
\[\text{else } x \text{ endif}\text{ endlet}\]
\[\text{elseif } \text{caddr} (x) = '0\]
\[\text{then let } factors \text{ be itimes-factors} (\text{cadr} (x))\]
\[\text{in}\]
\[\text{if } \text{listp} (\text{cdr} (factors))\]
\[\text{then } \text{disjoin-equalities-with-0} (factors)\]
\[\text{else } x \text{ endif}\text{ endlet} \]
Definition:
some-eval$s-to-0 (x, a)
= if listp (x)
    then (fix-int (eval$ (t, car (x), a)) = 0)
        \lor some-eval$s-to-0 (cdr (x), a)
    else f endif

Theorem: eval$-disjoin-equalities-with-0
listp (lst)
\rightarrow (eval$ (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$s-to-0 (x, a) \lor some-eval$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))) \rightarrow 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 (cadr (x))
                in
                if listp (cdr (factors))
                    then conjoin-inequalities-with-0 (factors, t)
                    else x endif endlet
            elseif cadr (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
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$-list-delete-new-2.
EVENT: Disable itimes-etimes-list-eval$-list-delete.

EVENT: Disable equal-etimes-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-etimes-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-etimes-list-eval$-list-delete-helper-1.

EVENT: Disable ilessp-etimes-list-eval$-list-delete-helper-2.

EVENT: Disable ilessp-etimes-list-eval$-list-delete.

EVENT: Disable ilessp-etimes-list-eval$-list-delete-prime-helper-1.

EVENT: Disable ilessp-etimes-list-eval$-list-delete-prime-helper-2.

EVENT: Disable ilessp-etimes-list-eval$-list-delete-prime.

EVENT: Disable ilessp-0-etimes.

EVENT: Disable ilessp-etimes-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) ∧ ((car (x) = 'ineg) ∨ (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 (cadr (x))):
                  case = ineg
                  then if listp (caddr (x))
                     then case on car (car (cadr (cadr (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 (cadr (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 (x))))),
                                     iplus-tree (cadr (remove-inegs (iplus-fringe (cadr (x)),
                                                      iplus-fringe (caddr (x))))))
                                  list ('quote, f)) endcase
     else list ('if, 
                  list ('integerp, caddr (x)),
                  list ('equal, 
                    list ('integerp, caddr (x)),
                    list ('equal, 
                      list ('integerp, caddr (x)),
                      list ('equal, 
                        list ('integerp, caddr (x)),
                        list ('equal, 
                          list ('integerp, caddr (x))))))))

71
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 (x))),
            iplus-fringe (caddr (x))))),
iplus-tree (cdr (remove-inegs (iplus-fringe (cadr (x))),
            iplus-fringe (caddr (x)))))),
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 (cadr (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) endif
otherwise if listp (caddr (x))
then case on car (car (cdr (cadr (x))));
case = ineg
then list ('if,
    list ('integerp, cadr (x)),
    list ('equal,
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 (\text{iplus-list} \( (eval$ \( ('\text{list}, \text{car} \( (remove-inegs \( (x, y)\)\), a))\), \text{iplus-list} \( (eval$ \( ('\text{list}, \text{cdr} \( (split-out-ineg-terms \( (y)\)\), a))\)\)\)\))\)

**THEOREM:** eval$-iplus-list-cdr-remove-inegs

\[remove-inegs \( (x, y) \rightarrow (\text{iplus-list} \( (eval$ \( ('\text{list}, \text{cdr} \( (remove-inegs \( (x, y)\)\), a))\), \text{iplus-list} \( (eval$ \( ('\text{list}, \text{car} \( (split-out-ineg-terms \( (x)\)\), a))\)\)\)\)\))\)

**THEOREM:** minus-ineg

\[\((x \in \mathbb{N}) \land (x \neq 0)\) \rightarrow ((- x) = \text{ineg} \( (x)\))\]

**THEOREM:** iplus-list- eval$-car-split-out-ineg-terms

\[iplus-list \( (eval$ \( ('\text{list}, \text{car} \( (\text{split-out-ineg-terms} \( (x)\)\), a))\), \text{iplus-list} \( (eval$ \( ('\text{list}, \text{cdr} \( (\text{split-out-ineg-terms} \( (x)\)\), a))\)\)\)\)\)

**EVENT:** Disable remove-inegs.

**THEOREM:** correctness-of-cancel-ineg-terms-from-equality

\[\text{eval} \( (t, x, a) = \text{eval} \( (t, \text{cancel-ineg-terms-from-equality-expanded} \( (x), a)\)\)\)

**DEFINITION:**

\[\text{make-cancel-ineg-terms-inequality} \( (x)\)

\[=\] let \( \text{new-fringes} \) be remove-inegs \( (\text{iplus-fringe} \( (\text{cadr} \( (x)\)\)), \text{iplus-fringe} \( (\text{caddr} \( (x)\)\))\)

\[\text{in}\]

\[\text{if} \text{new-fringes} \]

\[\text{then} \text{list} ('\text{ilessp}, \text{iplustree} \( (\text{car} \( (\text{new-fringes} ))\)), \text{iplustree} \( (\text{cdr} \( (\text{new-fringes} ))\))\]

\[\text{else} \ x \ \text{endif} \endlet\]
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
            else x endif
        else x endif
    otherwise if listp (caddr (x))
        then case on car (car (cdr (cadr (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
            else x endif
    endif
else x endif

75
cancel-ineg-terms-from-inequality-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
\[ \text{eval} \left( t, x, a \right) = \text{eval} \left( t, \text{cancel-ineg-terms-from-inequality-expanded} \left( x, a \right) \right) \]

EVENT: Disable minus-ineg.

EVENT: Disable integerp-\text{eval} \neg \text{-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
\[ ((i \in \mathbb{N}) \land (j \in \mathbb{N})) \rightarrow ((i + j) = \text{iplus} \left( i, j \right)) \]

THEOREM: iplus-constants
\[ \text{iplus} \left( 1 + i, \text{iplus} \left( 1 + j, x \right) \right) = \text{iplus} \left( (1 + i) + (1 + j), x \right) \]

THEOREM: numberp-is-integerp
\[ (w \in \mathbb{N}) \rightarrow \text{integerp} \left( w \right) \]

THEOREM: difference-idifference
\[ ((x \in \mathbb{N}) \land (y \in \mathbb{N}) \land (x \leq y)) \rightarrow ((y - x) = \text{idifference} \left( y, x \right)) \]

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

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

THEOREM: ilessp-add1
\[ (y \in \mathbb{N}) \rightarrow (\text{ilessp} \left( x, 1 + y \right) = \left( \neg \text{ilessp} \left( y, x \right) \right)) \]
Theorem: `ilessp-add1-iplus`
\((y \in \mathbb{N}) \rightarrow (ilessp (x, iplus (1 + y, z)) = (\neg ilessp (iplus (y, z), x)))\)

Theorem: `cancel-constants-ilessp-lemma-1`
\(((m \in \mathbb{N}) \land (n \in \mathbb{N}))\)
\(\rightarrow (ilessp (iplus (m, x), iplus (n, y)) = \begin{array}{ll}
\text{if } m < n & \text{then } ilessp (x, iplus (n - m, y)) \\
\text{else} & ilessp (iplus (m - n, x), y) \end{array} \text{ endif}
\)

Theorem: `cancel-constants-ilessp-lemma-2`
\(((m \in \mathbb{N}) \land (n \in \mathbb{N}))\)
\(\rightarrow (ilessp (iplus (m, x), iplus (n, y)) = \begin{array}{ll}
\text{if } m < n & \neg ilessp (iplus ((n - m) - 1, y), x) \\
\text{else} & ilessp (iplus (m - n, x), y) \end{array} \text{ endif}
\)

Theorem: `cancel-constants-ilessp`
\(ilessp (iplus (1 + i, x), iplus (1 + j, y)) = \begin{array}{ll}
\text{if } i < j & \neg ilessp (iplus ((j - i) - 1, y), x) \\
\text{else} & ilessp (iplus (i - j, x), y) \end{array} \text{ endif}
\)

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-TRICHTOMY 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:
Event: Make the library "integers" and compile it.
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>all-integer-defns, 7</td>
</tr>
<tr>
<td>associativity-of-iplus, 10</td>
</tr>
<tr>
<td>associativity-of-itimes, 25</td>
</tr>
<tr>
<td>bagdiff, 18–20, 22–24, 33, 36–42, 46, 47, 60, 62, 63</td>
</tr>
<tr>
<td>bagint, 18–20, 22–24, 34, 38–41, 46, 49, 60, 61, 64</td>
</tr>
<tr>
<td>bagint-singleton, 61</td>
</tr>
<tr>
<td>cancel-constants-equal, 77</td>
</tr>
<tr>
<td>cancel-constants-equal-lemma, 77</td>
</tr>
<tr>
<td>cancel-constants-ilessp, 78</td>
</tr>
<tr>
<td>cancel-constants-ilessp-lemma-1, 78</td>
</tr>
<tr>
<td>cancel-constants-ilessp-lemma-2, 78</td>
</tr>
<tr>
<td>cancel-factors-0, 65, 66</td>
</tr>
<tr>
<td>cancel-factors-ilessp-0, 67</td>
</tr>
<tr>
<td>cancel-ineg, 13, 15</td>
</tr>
<tr>
<td>cancel-ineg-aux, 11–13, 15</td>
</tr>
<tr>
<td>cancel-ineg-terms-from-equality, 71, 74</td>
</tr>
<tr>
<td>-cancel-ineg-terms-from-equality-expanded, 74</td>
</tr>
<tr>
<td>-expanded, 71, 74</td>
</tr>
<tr>
<td>cancel-ineg-terms-from-inequality, 75, 76</td>
</tr>
<tr>
<td>y-cancel-ineg-terms-from-inequality-expanded, 76</td>
</tr>
<tr>
<td>y-expanded, 75–77</td>
</tr>
<tr>
<td>cancel-iplus, 18, 19, 21</td>
</tr>
<tr>
<td>cancel-iplus-ilessp, 24</td>
</tr>
<tr>
<td>cancel-iplus-ilessp-1, 23</td>
</tr>
<tr>
<td>cancel-itimes, 34, 39</td>
</tr>
<tr>
<td>cancel-itimes-factors, 49, 59</td>
</tr>
<tr>
<td>cancel-itimes-factors-expanded, 59, 60</td>
</tr>
<tr>
<td>cancel-itimes-factors-expanded-cancel-itimes-factors, 59</td>
</tr>
<tr>
<td>cancel-itimes-ilessp, 40, 42</td>
</tr>
<tr>
<td>cancel-itimes-ilessp-factors, 61, 65</td>
</tr>
<tr>
<td>commutativity-of-iplus, 10</td>
</tr>
<tr>
<td>commutativity-of-itimes, 25</td>
</tr>
<tr>
<td>commutativity2-of-iplus, 10</td>
</tr>
<tr>
<td>commutativity2-of-itimes, 25</td>
</tr>
<tr>
<td>conjoin-inequalities-with-0, 66, 67</td>
</tr>
<tr>
<td>conjoin-inequalities-with-0-eliminator, 67</td>
</tr>
<tr>
<td>correctness-of-cancel-factors-0, 66</td>
</tr>
<tr>
<td>correctness-of-cancel-factors-ilessp-0, 67</td>
</tr>
<tr>
<td>correctness-of-cancel-ineg, 15</td>
</tr>
<tr>
<td>correctness-of-cancel-ineg-aux, 15</td>
</tr>
<tr>
<td>correctness-of-cancel-ineg-term-s-from-equality, 74</td>
</tr>
<tr>
<td>s-from-inequality, 77</td>
</tr>
<tr>
<td>correctness-of-cancel-iplus-ilessp-ile-ssp, 24</td>
</tr>
<tr>
<td>ssp-lemma, 23</td>
</tr>
<tr>
<td>correctness-of-cancel-itimes, 39</td>
</tr>
<tr>
<td>correctness-of-cancel-itimes-factors, 60</td>
</tr>
<tr>
<td>correctness-of-cancel-itimes-ha</td>
</tr>
<tr>
<td>ck-1, 36</td>
</tr>
<tr>
<td>ck-2, 38</td>
</tr>
<tr>
<td>ck-3, 38</td>
</tr>
<tr>
<td>ck-3-lemma, 38</td>
</tr>
<tr>
<td>correctness-of-cancel-itimes-ilessp-hack-1, 41</td>
</tr>
<tr>
<td>ssp-hack-1, 41</td>
</tr>
<tr>
<td>ssp-hack-2, 41</td>
</tr>
<tr>
<td>ssp-hack-2-lemma, 41</td>
</tr>
<tr>
<td>ssp-hack-3, 42</td>
</tr>
<tr>
<td>ssp-hack-3-lemma-1, 42</td>
</tr>
<tr>
<td>ssp-hack-3-lemma-2, 42</td>
</tr>
<tr>
<td>ssp-hack-4, 42</td>
</tr>
<tr>
<td>delete, 18–20, 36, 37, 59, 61, 62</td>
</tr>
<tr>
<td>difference-idifference, 77</td>
</tr>
<tr>
<td>disjoin-equalities-with-0, 65, 66</td>
</tr>
<tr>
<td>division-theorem, 26</td>
</tr>
<tr>
<td>division-theorem-for-truncate-t</td>
</tr>
</tbody>
</table>
idifference, 5–8, 10, 20, 77
idifference-fix-int1, 10
idifference-fix-int2, 10
idiv, 6, 27, 28, 30, 31
idiv-fix-int1, 30
idiv-fix-int2, 30
idiv-uniquness, 28
ileq, 5
ilessp-0-itimes, 62
ilessp-add1, 77
ilessp-add1-iplus, 78
ilessp-fix-int-1, 22
ilessp-fix-int-2, 22
ilessp-ineg-ineg, 47
ilessp-itimes-0, 62
ilessp-itimes-list-eval$-list-bagdiff, 62
bagdiff-corollary-1, 62
bagdiff-corollary-2, 63
delete, 61
delete-helper-1, 61
delete-helper-2, 61
delete-prime, 62
delete-prime-helper-1, 61
delete-prime-helper-2, 61
ilessp-itimes-right-negative, 42
ilessp-itimes-right-positive, 41
ilessp-strict, 43
ilessp-trichotomy, 42
ilessp-zero-implies-not-equal, 63
imod, 6, 27, 28, 30, 31
imod-fix-int1, 30
imod-fix-int2, 30
ineg, 5, 8–10, 15, 21, 28, 29, 47, 74
ineg-0, 9
ineg-eval$-itimes-tree-ineg, 47
ineg-fix-int, 9
ineg-ineg, 9
ineg-iplus, 9
ineg-of-non-integerp, 9
integer-defns, 7
integerp, 4, 8, 9, 16, 20, 21, 26–30, 32, 35–37, 59, 60, 66, 74, 77
integerp-eval$-iplus-or-ineg-te
rm, 74
integerp-eval$-itimes, 35
integerp-fix-int, 8
integerp-labs, 8
integerp-idifference, 8
integerp-idiv, 30
integerp-imod, 30
integerp-ineg, 8
integerp-iplus, 8
integerp-iplus-list, 16
integerp-iquo, 30
integerp-iquotient, 29
integerp-irem, 30
integerp-iremaider, 30
integerp-itimes, 8
integerp-itimes-list, 32
integers, 79
iplus, 5, 8–10, 14–16, 20, 21, 23, 25–29, 47, 74, 77, 78
iplus-0-left, 9
iplus-0-right, 10
iplus-cancellation-1, 10
iplus-cancellation-1-for-ilessp, 23
iplus-cancellation-2, 10
iplus-cancellation-2-for-ilessp, 23
iplus-constants, 77
iplus-eval$-itimes-tree-ineg, 47
iplus-fix-int1, 10
iplus-fix-int2, 10
iplus-fringe, 16, 18–21, 23, 24, 70–76
iplus-ineg-promote, 15
iplus-ineg1, 10
iplus-ineg2, 10
iplus-ineg3, 15
iplus-ineg4, 15
iplus-ineg5, 21
iplus-ineg5-lemma-1, 21
iplus-ineg5-lemma-2, 21
iplus-ineg6, 21
iplus-ineg7, 21
iplus-left-id, 9
iplus-list, 16, 17, 20, 21, 74
iplus-list-append, 20
iplus-list-evaul8-car-split-out
-ineg-terms, 74
iplus-list-evaul8-fringe, 21
iplus-or-ineg-term, 70, 74, 75
iplus-or-itimes-term, 48, 49, 60
iplus-or-itimes-term-integerp-ev
al8, 60
iplus-right-id, 9
iplus-tree, 16–19, 22, 23, 70–76
iplus-tree-no-fix-int, 23, 24
iplus-tree-rec, 16, 23
iplus-x-y-ineg-x, 15
quo, 6, 7, 28–31
quo-fix-int1, 30
quo-fix-int2, 30
quo-irem-uniqueness, 29
quotient, 6, 26, 27, 29–31
quotient-fix-int1, 30
quotient-fix-int2, 30
quotient-irem-uniqueness, 27
rem, 7, 28–31
rem-fix-int1, 30
rem-fix-int2, 31
remainder, 6, 26, 27, 30, 31
remainder-fix-int1, 30
remainder-fix-int2, 30
itimes, 6–8, 25–29, 32, 35, 36, 38, 41, 42, 47, 59–62
itimes-1, 47
itimes-0-left, 25
itimes-0-right, 25
itimes-1-arg1, 26
itimes-cancellation-1, 29
itimes-cancellation-2, 29
itimes-cancellation-3, 29
itimes-distributes-over-iplus, 25
itimes-distributes-over-iplus-p
roof, 25
itimes-evaul8-itimes-tree-ineg, 47
itimes-factors, 46, 47, 49, 60, 61, 63–67
itimes-fix-int1, 25
itimes-fix-int2, 25
itimes-fringe, 32, 34–42
itimes-ineg-1, 29
itimes-ineg-2, 29
itimes-itimes-list-evaul8-list-
delete, 59
itimes-list, 32, 33, 35–39, 41, 42, 47, 59–67
itimes-list-append, 35
itimes-list-bagdiff, 36
itimes-list-evaul8-delete, 36
itimes-list-evaul8-factors, 60
itimes-list-evaul8-factors-lemma, 60
a-prime, 60
itimes-list-evaul8-fringe, 35
itimes-list-evaul8-list-0, 41
itimes-tree, 32, 33, 37, 39, 40
itimes-tree-ineg, 46, 47
itimes-tree-no-fix-int, 39–41
itimes-tree-rec, 32, 33, 39, 46, 47
itimes-zero1, 25
itimes-zero2, 25
izerop, 4
izerop-avoid-dividend-implies-itimes-
list-0, 37
izerop-ilessp-0-relationship, 61
lessp-count-listp-cdr, 16
lessp-difference-plus-arg1, 23
lessp-difference-plus-arg1-commu-
ted, 23
listp-bagint-with-singleton-imp
lies-member, 41
listp-bagint-with-singleton-member, 41
listp-cdr-factors-implies-integers,
66
make-cancel-ineg-terms-equality, 70,
71
make-cancel-ineg-terms-inequality, 74, 75
make-cancel-iplus-inequality-1, 22, 23
make-cancel-iplus-inequality-sil
mplifier, 23
make-cancel-itimes-equality, 33–37, 49
make-cancel-itimes-inequality, 40, 41, 61
member-0-eval$-list, 60
member-0-itimes-factors-yields-
0-ilessp-consequence-1, 63
0-ilessp-consequence-2, 63
0, 63
member-append, 36
member-izerop-itimes-fringe, 36
membership-of-0-implies-itimes-
list-is-0, 60
minus-ineg, 74
not-integerp-implies-not-equal-ip-
lus, 21
not-integerp-implies-not-equal-it-
mes, 35
numberp-is-integerp, 77
plus-iplus, 77
quotient-difference-lessp-arg2, 27
quotient-remainder-uniqueness, 26
remove-inegs, 70–76
same-fix-int-implies-not-ilessp, 42
some-eval$s-to-0, 66
some-eval$s-to-0-append, 66
some-eval$s-to-0-eliminator, 66
split-out-ineg-terms, 69, 70, 74
subbagp, 20, 36, 38, 41, 42, 59, 60, 62, 63
subbagp-subsetp, 38
subsetp, 38, 42
subsetp-implies-itimes-list-eval-
l$-equals-0, 38
zero-ilessp-implies-not-equal, 62