EVENT: Start with the library "bags".

```
;; Tue Sep 26 10:20:45 1989, from ~wilding/numerical/newnat.events

;; NATURALS Theory

;; Created by Bill Bevier 1988 (see CLI internal note 057)

;; Modifications by Bill Bevier and Matt Wilding (9/89) including
;; adding some new metalemmas for times, reorganizing the theories,
;; removing some extraneous lemmas, and removing dependence upon
;; other theories (by adding the pertinent lemmas).

;; This script requires the bags theory

;; This script sets up a theory for the NATURALS with the following subtheories
;; ADDITION
;; MULTIPLICATION
;; REMAINDER
;; QUOTIENT
;; EXPONENTIATION
;; LOGS
;; GCDS

;; The theories of EXPONENTIATION, LOGS, and GCDS still need a lot of work

; ------------------------------------------------------------------------------
; ARITHMETIC
; ------------------------------------------------------------------------------

; -------------------- PLUS & DIFFERENCE --------------------

; ---------- EQUAL ----------
```

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

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

EVENT: Disable plus-cancellation.

THEOREM: equal-difference-0

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

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

EVENT: Disable difference-cancellation.

```
; ---------- PLUS ----------
```

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

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

THEOREM: plus-zero-arg2
$(y \simeq 0) \rightarrow ((x + y) = \mathrm{fix}\,(x))$

THEOREM: plus-add1-arg1
$((1 + a) + b) = (1 + (a + b))$

THEOREM: plus-add1-arg2
$(x + (1 + y))$
$=$ **if** $y \in \mathbf{N}$ **then** $1 + (x + y)$
else $1 + x$ **endif**

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

THEOREM: plus-difference-arg1
$((a - b) + c)$
$=$ **if** $b < a$ **then** $(a + c) - b$
else $0 + c$ **endif**

THEOREM: plus-difference-arg2
$(a + (b - c))$
$=$ **if** $c < b$ **then** $(a + b) - c$
else $a + 0$ **endif**

```
; ---------- DIFFERENCE-PLUS cancellation rules ----------
;
; Here are the basic canonicalization rules for differences of sums. These
; are subsumed by the meta lemmas and are therefore globally disabled.
; They are here merely to prove the meta lemmas.
```

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

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

EVENT: Disable difference-plus-cancellation.

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

THEOREM: difference-plus-plus-cancellation
$(((x + y) - (x + z)) = (y - z))$
$\wedge \quad (((y + x) - (x + z)) = (y - z))$
$\wedge \quad (((x + y) - (z + x)) = (y - z))$
$\wedge \quad (((y + x) - (z + x)) = (y - z))$

EVENT: Disable difference-plus-plus-cancellation.

THEOREM: difference-plus-plus-cancellation-hack
$((w + x + a) - (y + z + a)) = ((w + x) - (y + z))$

EVENT: Disable difference-plus-plus-cancellation-hack.

```
; Here are a few more facts about difference needed to prove the meta lemmas.
; These are disabled here. We re-prove them after the proof of the meta
; lemmas so that they will fire before the meta lemmas in subsequent proofs.
```

THEOREM: diff-sub1-arg2
$(a - (b - 1))$
$=$ **if** $b \simeq 0$ **then** $\text{fix}(a)$
elseif $a < b$ **then** 0
else $1 + (a - b)$ **endif**

EVENT: Disable diff-sub1-arg2.

THEOREM: diff-diff-arg1
$((x - y) - z) = (x - (y + z))$

THEOREM: diff-diff-arg2
$(a - (b - c))$
$=$ **if** $b < c$ **then** $\text{fix}(a)$
else $(a + c) - b$ **endif**

```
; diff-diff-diff should be removed, but since the hack lemmas for
; correctness-of-cancel-difference-plus are designed for it, we'll
; keep it around.
```

THEOREM: diff-diff-diff
$((b \leq a) \wedge (d \leq c))$
$\rightarrow \quad (((a - b) - (c - d)) = ((a + d) - (b + c)))$

EVENT: Disable diff-diff-diff.

THEOREM: difference-lessp-arg1
$(a < b) \rightarrow ((a - b) = \mathtt{0})$

EVENT: Disable difference-lessp-arg1.

```
; ------------------------------------------------------------------------------
; Meta Lemmas to Cancel PLUS and DIFFERENCE expressions
; ------------------------------------------------------------------------------

; ---------- PLUS-TREE and PLUS-FRINGE ----------
```

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

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

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

EVENT: Disable numberp-eval$-plus.

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

EVENT: Disable numberp-eval$-plus-tree.

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

EVENT: Disable member-implies-plus-tree-greatereqp.

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

EVENT: Disable plus-tree-delete.

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

EVENT: Disable subbagp-implies-plus-tree-greatereqp.

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

EVENT: Disable plus-tree-bagdiff.

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

EVENT: Disable numberp-eval$-bridge.

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

EVENT: Disable bridge-to-subbagp-implies-plus-tree-greatereqp.

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

EVENT: Disable eval$-plus-tree-append.

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

EVENT: Disable plus-tree-plus-fringe.

THEOREM: member-implies-numberp
((c ∈ plus-fringe (x)) ∧ (eval$ (**t**, c, a) ∈ **N**)) → (eval$ (**t**, x, a) ∈ **N**)

EVENT: Disable member-implies-numberp.

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

EVENT: Disable cadr-eval$-list.

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

EVENT: Disable eval$-quote.

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

EVENT: Disable listp-eval$.

```
; ---------- CANCEL PLUS ----------

; CANCEL-EQUAL-PLUS cancels identical terms in a term which is the equality
; of two sums. For example,
;
;   (EQUAL (PLUS A B C) (PLUS B D E)) => (EQUAL (PLUS A C) (PLUS D E))
;
```

DEFINITION:
cancel-equal-plus (x)
= **if** listp (x) ∧ (car (x) = `'equal`)

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

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

```
; ---------- CANCEL-DIFFERENCE-PLUS ----------

; CANCEL-DIFFERENCE-PLUS cancels identical terms in a term which is the
; difference of two sums. For example,
;
;  (DIFFERENCE (PLUS A B C) (PLUS B D E)) => (DIFFERENCE (PLUS A C) (PLUS D E))
;
; Using rewrite rules, we canonicalize terms involving  PLUS and DIFFERENCE
; to be the DIFFERENCE of two sums. Then CANCEL-DIFFERENCE-PLUS cancels out
```

```
; like terms.
```

DEFINITION:
cancel-difference-plus (x)
$=$ **if** listp $(x) \wedge$ (car $(x) =$ `'difference`)
then if listp (cadr (x))
$\wedge$ (caadr $(x) =$ `'plus`)
$\wedge$ listp (caddr (x))
$\wedge$ (caaddr $(x) =$ `'plus`)
then list (`'difference`,
plus-tree (bagdiff (plus-fringe (cadr (x))),
bagint (plus-fringe (cadr (x))),
plus-fringe (caddr (x)))))),
plus-tree (bagdiff (plus-fringe (caddr (x))),
bagint (plus-fringe (cadr (x))),
plus-fringe (caddr (x)))))))
elseif listp (cadr (x))
$\wedge$ (caadr $(x) =$ `'plus`)
$\wedge$ (caddr $(x) \in$ plus-fringe (cadr (x))))
then plus-tree (delete (caddr (x), plus-fringe (cadr (x))))
elseif listp (caddr (x))
$\wedge$ (caaddr $(x) =$ `'plus`)
$\wedge$ (cadr $(x) \in$ plus-fringe (caddr (x)))) **then** `''0`
else x **endif**
else x **endif**

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

```
; ---------- DIFFERENCE ----------

; Here are the rules for difference terms which we want to try before
; the meta lemmas. They help canonicalize terms to differences of sums.
```

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

THEOREM: difference-leq-arg1
$(a \leq b) \rightarrow ((a - b) = 0)$

THEOREM: difference-add1-arg2
$(a - (1 + b))$
$=$ **if** $b < a$ **then** $(a - b) - 1$
else 0 **endif**

THEOREM: difference-sub1-arg2
$(a - (b - 1))$
$=$ **if** $b \simeq$ `0` **then** fix (a)
elseif $a < b$ **then** `0`
else $1 + (a - b)$ **endif**

THEOREM: difference-difference-arg1
$((x - y) - z) = (x - (y + z))$

THEOREM: difference-difference-arg2
$(a - (b - c))$
$=$ **if** $b < c$ **then** fix (a)
else $(a + c) - b$ **endif**

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

```
; ---------- LESSP ----------
```

THEOREM: lessp-difference-cancellation
$((a - c) < (b - c))$
$=$ **if** $c \leq a$ **then** $a < b$
else $c < b$ **endif**

EVENT: Disable lessp-difference-cancellation.

```
; CANCEL-LESSP-PLUS cancels LESSP terms whose arguments are sums.
; Examples:
; (LESSP (PLUS A B C) (PLUS A C D)) -> (LESSP (FIX B) (FIX D))
; (LESSP A (PLUS A B))  -> (NOT (ZEROP (FIX B)))
; (LESSP (PLUS A B) A)  -> F
```

DEFINITION:
cancel-lessp-plus (x)
$=$ **if** listp $(x) \wedge$ (car $(x) =$ `'lessp`)
then if listp (cadr (x))
$\wedge$ (caadr $(x) =$ `'plus`)
$\wedge$ listp (caddr (x))
$\wedge$ (caaddr $(x) =$ `'plus`)
then list (`'lessp`,
plus-tree (bagdiff (plus-fringe (cadr (x)),
bagint (plus-fringe (cadr (x)),
plus-fringe (caddr (x))))),

plus-tree (bagdiff (plus-fringe (caddr (*x*)),
bagint (plus-fringe (cadr (*x*)),
plus-fringe (caddr (*x*))))))
elseif listp (cadr (*x*))
∧ (caadr (*x*) = 'plus)
∧ (caddr (*x*) ∈ plus-fringe (cadr (*x*)))
then list ('quote, **f**)
elseif listp (caddr (*x*))
∧ (caaddr (*x*) = 'plus)
∧ (cadr (*x*) ∈ plus-fringe (caddr (*x*)))
then list ('not,
list ('zerop,
plus-tree (delete (cadr (*x*), plus-fringe (caddr (*x*))))))
else *x* **endif**
else *x* **endif**

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

```
; Define the available theory of addition.  To get the list of events to
; put in the theory, evaluate the following form in NQTHM at this point
; in the script. This form lists all  lemmas which are globally enabled,
; and which have non-null lemma type.
;
; (remove-if-not (function (lambda (x)
;     (and (member x (lemmas))
;  (not (assoc x disabled-lemmas))
;  (not (null (nth 2 (get x 'event)))))))
;         chronology)
```

EVENT: Let us define the theory *addition* to consist of the following events: equal-plus-0, equal-difference-0, commutativity-of-plus, commutativity2-of-plus, plus-zero-arg2, plus-add1-arg2, plus-add1-arg1, associativity-of-plus, plus-difference-arg1, plus-difference-arg2, diff-diff-arg1, diff-diff-arg2, correctness-of-cancel-equal-plus, correctness-of-cancel-difference-plus, difference-elim, difference-leq-arg1, difference-add1-arg2, difference-sub1-arg2, difference-difference-arg1, difference-difference-arg2, difference-x-x, correctness-of-cancel-lessp-plus.

```
; ---------- TIMES ----------
```

THEOREM: equal-times-0
((*x* ∗ *y*) = 0) = ((*x* ≃ 0) ∨ (*y* ≃ 0))

THEOREM: equal-times-1
$((a * b) = 1) = ((a = 1) \wedge (b = 1))$

```
;(lemma equal-sub1-times-0 (rewrite)
;       (equal (equal (sub1 (times a b)) 0)
;       (or (zerop a)
;   (zerop b)
;   (and (equal a 1) (equal b 1)))))
```

THEOREM: equal-sub1-0
$((x - 1) = 0) = ((x \simeq 0) \vee (x = 1))$

THEOREM: times-zero
$(y \simeq 0) \rightarrow ((x * y) = 0)$

THEOREM: times-add1
$(x * (1 + y))$
$=$ **if** $y \in \mathbf{N}$ **then** $x + (x * y)$
else $\mathrm{fix}\,(x)$ **endif**

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

THEOREM: times-distributes-over-plus-proof
$(x * (y + z)) = ((x * y) + (x * z))$

THEOREM: times-distributes-over-plus
$((x * (y + z)) = ((x * y) + (x * z)))$
$\wedge$ $(((x + y) * z) = ((x * z) + (y * z)))$

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

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

THEOREM: times-distributes-over-difference-proof
$((a - b) * c) = ((a * c) - (b * c))$

THEOREM: times-distributes-over-difference
$(((a - b) * c) = ((a * c) - (b * c)))$
$\wedge$ $((a * (b - c)) = ((a * b) - (a * c)))$

THEOREM: times-quotient-proof
$((x \not\simeq 0) \wedge ((y \bmod x) = 0)) \rightarrow (((y \div x) * x) = \mathrm{fix}\,(y))$

THEOREM: times-quotient
$((y \not\simeq \mathtt{0}) \wedge ((x \ \mathbf{mod}\ y) = \mathtt{0}))$
$\rightarrow \quad ((((x \div y) * y) = \mathrm{fix}\,(x)) \wedge ((y * (x \div y)) = \mathrm{fix}\,(x)))$

THEOREM: times-1-arg1
$(\mathtt{1} * x) = \mathrm{fix}\,(x)$

THEOREM: lessp-times1-proof
$((a < b) \wedge (c \not\simeq \mathtt{0})) \rightarrow ((a < (b * c)) = \mathbf{t})$

THEOREM: lessp-times1
$((a < b) \wedge (c \not\simeq \mathtt{0}))$
$\rightarrow \quad (((a < (b * c)) = \mathbf{t}) \wedge ((a < (c * b)) = \mathbf{t}))$

THEOREM: lessp-times2-proof
$((a \leq b) \wedge (c \not\simeq \mathtt{0})) \rightarrow (((b * c) < a) = \mathbf{f})$

THEOREM: lessp-times2
$((a \leq b) \wedge (c \not\simeq \mathtt{0}))$
$\rightarrow \quad ((((b * c) < a) = \mathbf{f}) \wedge (((c * b) < a) = \mathbf{f}))$

THEOREM: lessp-times3-proof1
$((a \not\simeq \mathtt{0}) \wedge (\mathtt{1} < b)) \rightarrow (a < (a * b))$

THEOREM: lessp-times3-proof2
$(a < (a * b)) \rightarrow ((a \not\simeq \mathtt{0}) \wedge (\mathtt{1} < b))$

THEOREM: lessp-times3
$((a < (a * b)) = ((a \not\simeq \mathtt{0}) \wedge (\mathtt{1} < b)))$
$\wedge \quad ((a < (b * a)) = ((a \not\simeq \mathtt{0}) \wedge (\mathtt{1} < b)))$

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

THEOREM: lessp-times-cancellation1
$(((x * z) < (y * z)) = ((z \not\simeq \mathtt{0}) \wedge (x < y)))$
$\wedge \quad (((z * x) < (y * z)) = ((z \not\simeq \mathtt{0}) \wedge (x < y)))$
$\wedge \quad (((x * z) < (z * y)) = ((z \not\simeq \mathtt{0}) \wedge (x < y)))$
$\wedge \quad (((z * x) < (z * y)) = ((z \not\simeq \mathtt{0}) \wedge (x < y)))$

EVENT: Disable lessp-times-cancellation1.

THEOREM: lessp-plus-times-proof
$(x < a) \rightarrow (((x + (a * b)) < (a * c)) = (b < c))$

THEOREM: lessp-plus-times1
$(((a + (b * c)) < b) = ((a < b) \wedge (c \simeq \mathtt{0})))$
$\wedge \quad (((a + (c * b)) < b) = ((a < b) \wedge (c \simeq \mathtt{0})))$
$\wedge \quad ((((c * b) + a) < b) = ((a < b) \wedge (c \simeq \mathtt{0})))$
$\wedge \quad ((((b * c) + a) < b) = ((a < b) \wedge (c \simeq \mathtt{0})))$

THEOREM: lessp-plus-times2
$((a \not\simeq \mathtt{0}) \wedge (x < a))$
$\rightarrow \quad ((((x + (a * b)) < (a * c)) = (b < c))$
$\wedge \quad (((x + (b * a)) < (a * c)) = (b < c))$
$\wedge \quad (((x + (a * b)) < (c * a)) = (b < c))$
$\wedge \quad (((x + (b * a)) < (c * a)) = (b < c))$
$\wedge \quad ((((a * b) + x) < (a * c)) = (b < c))$
$\wedge \quad ((((b * a) + x) < (a * c)) = (b < c))$
$\wedge \quad ((((a * b) + x) < (c * a)) = (b < c))$
$\wedge \quad ((((b * a) + x) < (c * a)) = (b < c)))$

THEOREM: lessp-1-times
$(\mathtt{1} < (a * b))$
$= \quad (\neg\, ((a \simeq \mathtt{0}) \vee (b \simeq \mathtt{0}) \vee ((a = \mathtt{1}) \wedge (b = \mathtt{1}))))$

```
;;; meta lemmas to cancel lessp-times and equal-times expressions
;; examples
;; (lessp (times b (times d a)) (times b (times e (times a f)))) ->
;;                                                 (and (and (not (zerop a))
;;                                                           (not (zerop b)))
;;                                                      (lessp (fix d) (times e f)))
;;
;; (equal (times b (times c d)) (times b d))    ->
;;                                               (or (or (zerop b) (zerop d))
;;                                                   (equal (fix c) 1))
```

DEFINITION:
times-tree (x)
$=$ **if** $x \simeq$ **nil** **then** `''1`
elseif cdr $(x) \simeq$ **nil** **then** list (`'fix`, car (x))
elseif cddr $(x) \simeq$ **nil** **then** list (`'times`, car (x), cadr (x))
else list (`'times`, car (x), times-tree (cdr (x))) **endif**

DEFINITION:
times-fringe (x)
$=$ **if** listp $(x) \wedge$ (car $(x) =$ `'times`)
then append (times-fringe (cadr (x)), times-fringe (caddr (x)))
else cons $(x,$ **nil**$)$ **endif**

DEFINITION:
or-zerop-tree (x)
= **if** $x \simeq$ **nil** **then** `'(false)`
elseif cdr $(x) \simeq$ **nil** **then** list (`'zerop`, car (x))
elseif cddr $(x) \simeq$ **nil**
then list (`'or`, list (`'zerop`, car (x)), list (`'zerop`, cadr (x)))
else list (`'or`, list (`'zerop`, car (x)), or-zerop-tree (cdr (x))) **endif**

DEFINITION:
and-not-zerop-tree (x)
= **if** $x \simeq$ **nil** **then** `'(true)`
elseif cdr $(x) \simeq$ **nil** **then** list (`'not`, list (`'zerop`, car (x)))
else list (`'and`,
list (`'not`, list (`'zerop`, car (x))),
and-not-zerop-tree (cdr (x))) **endif**

THEOREM: numberp-eval$-times
(car (x) = `'times`) $\rightarrow$ (eval$ (**t**, x, a) $\in$ **N**)

EVENT: Disable numberp-eval$-times.

THEOREM: eval$-times
(car (x) = `'times`)
$\rightarrow$ (eval$ (**t**, x, a) = (eval$ (**t**, cadr (x), a) $*$ eval$ (**t**, caddr (x), a)))

EVENT: Disable eval$-times.

THEOREM: eval$-or
(car (x) = `'or`)
$\rightarrow$ (eval$ (**t**, x, a) = (eval$ (**t**, cadr (x), a) $\vee$ eval$ (**t**, caddr (x), a)))

EVENT: Disable eval$-or.

THEOREM: eval$-equal
(car (x) = `'equal`)
$\rightarrow$ (eval$ (**t**, x, a) = (eval$ (**t**, cadr (x), a) = eval$ (**t**, caddr (x), a)))

EVENT: Disable eval$-equal.

THEOREM: eval$-lessp
(car (x) = `'lessp`)
$\rightarrow$ (eval$ (**t**, x, a) = (eval$ (**t**, cadr (x), a) $<$ eval$ (**t**, caddr (x), a)))

EVENT: Disable eval$-lessp.

THEOREM: eval$-quotient
$(\text{car}\,(x) = \texttt{'quotient})$
$\rightarrow \quad (\text{eval\$}\,(\mathbf{t},\ x,\ a) = (\text{eval\$}\,(\mathbf{t},\ \text{cadr}\,(x),\ a) \div \text{eval\$}\,(\mathbf{t},\ \text{caddr}\,(x),\ a)))$

EVENT: Disable eval$-quotient.

THEOREM: eval$-if
$(\text{car}\,(x) = \texttt{'if})$
$\rightarrow \quad (\text{eval\$}\,(\mathbf{t},\ x,\ a)$
$\quad = \quad \textbf{if}\ \text{eval\$}\,(\mathbf{t},\ \text{cadr}\,(x),\ a)\ \ \textbf{then}\ \text{eval\$}\,(\mathbf{t},\ \text{caddr}\,(x),\ a)$
$\quad\quad \textbf{else}\ \text{eval\$}\,(\mathbf{t},\ \text{cadddr}\,(x),\ a)\ \textbf{endif})$

EVENT: Disable eval$-if.

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

EVENT: Disable numberp-eval$-times-tree.

THEOREM: lessp-times-arg1
$(a \not\simeq \texttt{0}) \rightarrow (((a * x) \not< (a * y)) = (x \not< y))$

THEOREM: infer-equality-from-not-lessp
$((a \in \mathbf{N}) \wedge (b \in \mathbf{N})) \rightarrow (((a \not< b) \wedge (b \not< a)) = (a = b))$

THEOREM: equal-times-arg1
$(a \not\simeq \texttt{0}) \rightarrow (((a * x) = (a * y)) = (\text{fix}\,(x) = \text{fix}\,(y)))$

EVENT: Disable equal-times-arg1.

THEOREM: equal-times-bridge
$((a * b) = (c * (a * d))) = ((a \simeq \texttt{0}) \vee (\text{fix}\,(b) = (c * d)))$

EVENT: Disable equal-times-bridge.

THEOREM: eval$-times-member
$(e \in x)$
$\rightarrow \quad (\text{eval\$}\,(\mathbf{t},\ \text{times-tree}\,(x),\ a)$
$\quad = \quad (\text{eval\$}\,(\mathbf{t},\ e,\ a) * \text{eval\$}\,(\mathbf{t},\ \text{times-tree}\,(\text{delete}\,(e,\ x)),\ a)))$

EVENT: Disable eval$-times-member.

THEOREM: zerop-makes-times-tree-zero
((¬ eval$ (**t**, and-not-zerop-tree (*x*), *a*)) ∧ subbagp (*x*, *y*))
→ (eval$ (**t**, times-tree (*y*), *a*) = 0)

EVENT: Disable zerop-makes-times-tree-zero.

THEOREM: or-zerop-tree-is-not-zerop-tree
eval$ (**t**, or-zerop-tree (*x*), *a*) = (¬ eval$ (**t**, and-not-zerop-tree (*x*), *a*))

EVENT: Disable or-zerop-tree-is-not-zerop-tree.

THEOREM: zerop-makes-times-tree-zero2
(eval$ (**t**, or-zerop-tree (*x*), *a*) ∧ subbagp (*x*, *y*))
→ (eval$ (**t**, times-tree (*y*), *a*) = 0)

EVENT: Disable zerop-makes-times-tree-zero2.

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

EVENT: Disable times-tree-append.

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

EVENT: Disable times-tree-of-times-fringe.

DEFINITION:
cancel-lessp-times (*x*)
= **if** (car (*x*) = 'lessp)
∧ (caadr (*x*) = 'times)
∧ (caaddr (*x*) = 'times)
then let *inboth* **be** bagint (times-fringe (cadr (*x*)),
times-fringe (caddr (*x*)))
in
if listp (*inboth*)
then list ('and,
and-not-zerop-tree (*inboth*),
list ('lessp,
times-tree (bagdiff (times-fringe (cadr (*x*)),
inboth)),

```
                    times-tree (bagdiff (times-fringe (caddr (x)),
                                         inboth))))
        else x endif endlet
    else x endif
```

THEOREM: eval$-lessp-times-tree-bagdiff
$$\begin{array}{ll}
(\text{subbagp}(x,\ y) \wedge \text{subbagp}(x,\ z) \wedge \text{eval\$}(\mathbf{t},\ \text{and-not-zerop-tree}(x),\ a)) & \\
\rightarrow & ((\text{eval\$}(\mathbf{t},\ \text{times-tree}(\text{bagdiff}(y,\ x)),\ a) \\
& \quad < \quad \text{eval\$}(\mathbf{t},\ \text{times-tree}(\text{bagdiff}(z,\ x)),\ a)) \\
& \quad = \quad (\text{eval\$}(\mathbf{t},\ \text{times-tree}(y),\ a) < \text{eval\$}(\mathbf{t},\ \text{times-tree}(z),\ a)))
\end{array}$$

EVENT: Disable eval$-lessp-times-tree-bagdiff.

THEOREM: zerop-makes-lessp-false-bridge
$$\begin{array}{ll}
((\text{car}(x) = \texttt{'times}) & \\
\wedge & (\text{car}(y) = \texttt{'times}) \\
\wedge & (\neg\ \text{eval\$}(\mathbf{t}, \\
& \qquad \text{and-not-zerop-tree}(\text{bagint}(\text{times-fringe}(x),\ \text{times-fringe}(y))), \\
& \qquad a))) \\
\rightarrow & (((\text{eval\$}(\mathbf{t},\ \text{cadr}(x),\ a) * \text{eval\$}(\mathbf{t},\ \text{caddr}(x),\ a)) \\
& \quad < \quad (\text{eval\$}(\mathbf{t},\ \text{cadr}(y),\ a) * \text{eval\$}(\mathbf{t},\ \text{caddr}(y),\ a))) \\
& \quad = \quad \mathbf{f})
\end{array}$$

EVENT: Disable zerop-makes-lessp-false-bridge.

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

DEFINITION:
```
cancel-equal-times (x)
=  if (car (x) = 'equal)
     ∧  (caadr (x) = 'times)
     ∧  (caaddr (x) = 'times)
   then let inboth be bagint (times-fringe (cadr (x)),
                              times-fringe (caddr (x)))
        in
        if listp (inboth)
        then list ('or,
                   or-zerop-tree (inboth),
                   list ('equal,
                         times-tree (bagdiff (times-fringe (cadr (x)),
                                              inboth)),
                         times-tree (bagdiff (times-fringe (caddr (x)),
                                              inboth))))
```

else x **endif endlet**
else x **endif**

THEOREM: zerop-makes-equal-true-bridge
((car (x) = 'times)
$\wedge$ (car (y) = 'times)
$\wedge$ eval\$ (**t**, or-zerop-tree (bagint (times-fringe (x), times-fringe (y))), a))
$\rightarrow$ (((eval\$ (**t**, cadr (x), a) $*$ eval\$ (**t**, caddr (x), a))
= (eval\$ (**t**, cadr (y), a) $*$ eval\$ (**t**, caddr (y), a)))
= **t**)

EVENT: Disable zerop-makes-equal-true-bridge.

THEOREM: eval\$-equal-times-tree-bagdiff
(subbagp (x, y) $\wedge$ subbagp (x, z) $\wedge$ ($\neg$ eval\$ (**t**, or-zerop-tree (x), a)))
$\rightarrow$ ((eval\$ (**t**, times-tree (bagdiff (y, x)), a)
= eval\$ (**t**, times-tree (bagdiff (z, x)), a))
= (eval\$ (**t**, times-tree (y), a) = eval\$ (**t**, times-tree (z), a)))

EVENT: Disable eval\$-equal-times-tree-bagdiff.

THEOREM: cancel-equal-times-preserves-inequality
(subbagp (z, x)
$\wedge$ subbagp (z, y)
$\wedge$ (eval\$ (**t**, times-tree (x), a) $\neq$ eval\$ (**t**, times-tree (y), a)))
$\rightarrow$ (eval\$ (**t**, times-tree (bagdiff (x, z)), a)
$\neq$ eval\$ (**t**, times-tree (bagdiff (y, z)), a))

EVENT: Disable cancel-equal-times-preserves-inequality.

THEOREM: cancel-equal-times-preserves-inequality-bridge
((car (x) = 'times)
$\wedge$ (car (y) = 'times)
$\wedge$ ((eval\$ (**t**, cadr (x), a) $*$ eval\$ (**t**, caddr (x), a))
$\neq$ (eval\$ (**t**, cadr (y), a) $*$ eval\$ (**t**, caddr (y), a))))
$\rightarrow$ (eval\$ (**t**,
times-tree (bagdiff (times-fringe (x),
bagint (times-fringe (x), times-fringe (y)))),
a)
$\neq$ eval\$ (**t**,
times-tree (bagdiff (times-fringe (y),
bagint (times-fringe (x),
times-fringe (y)))),
a))

EVENT: Disable cancel-equal-times-preserves-inequality-bridge.

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

```
; Define the available  theory of  multiplication.   To  get the list of
; events to put in the theory,  evaluate the following  form in NQTHM at
; this point  in  the  script. This   form  lists all lemmas   which are
; globally enabled, and which have non-null lemma type.
;
; (remove-if-not (function (lambda (x)
;     (and (member x (lemmas))
;  (not (assoc x disabled-lemmas))
;  (not (null (nth 2 (get x 'event))))
;  (not (member x (nth 2 (get 'addition 'event)))))))
;         chronology)
```

EVENT: Let us define the theory *multiplication* to consist of the following events: equal-times-0, equal-times-1, equal-sub1-0, times-zero, times-add1, commutativity-of-times, times-distributes-over-plus, commutativity2-of-times, associativity-of-times, times-distributes-over-difference, times-quotient, times-1-arg1, lessp-times1, lessp-times2, lessp-times3, lessp-plus-times1, lessp-plus-times2, lessp-1-times, correctness-of-cancel-lessp-times, correctness-of-cancel-equal-times.

```
; ---------- REMAINDER ----------
```

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

THEOREM: remainder-noop
$(a < b) \rightarrow ((a\ \mathbf{mod}\ b) = \text{fix}\,(a))$

THEOREM: remainder-of-non-number
$(a \notin \mathbf{N}) \rightarrow ((a\ \mathbf{mod}\ n) = (\mathtt{0}\ \mathbf{mod}\ n))$

THEOREM: remainder-zero
$(x \simeq \mathtt{0}) \rightarrow ((y\ \mathbf{mod}\ x) = \text{fix}\,(y))$

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

EVENT: Disable plus-remainder-times-quotient.

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

```
; (lemma remainder-sub1 (rewrite)
;        (implies (and (not (zerop a))
;       (not (zerop b)))
;  (equal (remainder (sub1 a) b)
;         (if (equal (remainder a b) 0)
;     (sub1 b)
;     (sub1 (remainder a b)))))
;        ((enable lessp-remainder
;  remainder-noop
;  remainder-quotient-elim)
;  (enable-theory addition)
;  (induct (remainder a b))))
```

THEOREM: remainder-add1
$((a \ \mathbf{mod}\ b) = \mathtt{0}) \rightarrow (((1 + a) \ \mathbf{mod}\ b) = (\mathtt{1} \ \mathbf{mod}\ b))$

THEOREM: remainder-plus-proof
$((b \ \mathbf{mod}\ c) = \mathtt{0}) \rightarrow (((a + b) \ \mathbf{mod}\ c) = (a \ \mathbf{mod}\ c))$

THEOREM: remainder-plus
$((a \ \mathbf{mod}\ c) = \mathtt{0})$
$\rightarrow \quad ((((a + b) \ \mathbf{mod}\ c) = (b \ \mathbf{mod}\ c))$
$\quad \wedge \quad (((b + a) \ \mathbf{mod}\ c) = (b \ \mathbf{mod}\ c))$
$\quad \wedge \quad (((x + y + a) \ \mathbf{mod}\ c) = ((x + y) \ \mathbf{mod}\ c)))$

THEOREM: equal-remainder-plus-0-proof
$((a \ \mathbf{mod}\ c) = \mathtt{0}) \rightarrow ((((a + b) \ \mathbf{mod}\ c) = \mathtt{0}) = ((b \ \mathbf{mod}\ c) = \mathtt{0}))$

THEOREM: equal-remainder-plus-0
$((a \ \mathbf{mod}\ c) = \mathtt{0})$
$\rightarrow \quad (((((a + b) \ \mathbf{mod}\ c) = \mathtt{0}) = ((b \ \mathbf{mod}\ c) = \mathtt{0}))$
$\quad \wedge \quad ((((b + a) \ \mathbf{mod}\ c) = \mathtt{0}) = ((b \ \mathbf{mod}\ c) = \mathtt{0}))$
$\quad \wedge \quad ((((x + y + a) \ \mathbf{mod}\ c) = \mathtt{0})$
$\qquad = \quad (((x + y) \ \mathbf{mod}\ c) = \mathtt{0})))$

THEOREM: equal-remainder-plus-remainder-proof
$(a < c) \rightarrow ((((a + b) \ \mathbf{mod}\ c) = (b \ \mathbf{mod}\ c)) = (a \simeq \mathtt{0}))$

THEOREM: equal-remainder-plus-remainder
$(a < c)$
$\rightarrow \quad (((((a + b) \ \mathbf{mod}\ c) = (b \ \mathbf{mod}\ c)) = (a \simeq \mathtt{0}))$
$\quad \wedge \quad ((((b + a) \ \mathbf{mod}\ c) = (b \ \mathbf{mod}\ c)) = (a \simeq \mathtt{0})))$

EVENT: Disable equal-remainder-plus-remainder.

THEOREM: remainder-times1-proof
$((b \textbf{ mod } c) = 0) \rightarrow (((a * b) \textbf{ mod } c) = 0)$

THEOREM: remainder-times1
$((b \textbf{ mod } c) = 0)$
$\rightarrow \quad ((((a * b) \textbf{ mod } c) = 0) \wedge (((b * a) \textbf{ mod } c) = 0))$

THEOREM: remainder-times1-instance-proof
$((x * y) \textbf{ mod } y) = 0$

THEOREM: remainder-times1-instance
$(((x * y) \textbf{ mod } y) = 0) \wedge (((x * y) \textbf{ mod } x) = 0)$

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

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

EVENT: Disable remainder-times-times.

THEOREM: remainder-times2-proof
$((a \textbf{ mod } z) = 0) \rightarrow ((a \textbf{ mod } (z * y)) = (z * ((a \div z) \textbf{ mod } y)))$

THEOREM: remainder-times2
$((a \textbf{ mod } z) = 0)$
$\rightarrow \quad (((a \textbf{ mod } (y * z)) = (z * ((a \div z) \textbf{ mod } y)))$
$\quad \wedge \quad ((a \textbf{ mod } (z * y)) = (z * ((a \div z) \textbf{ mod } y))))$

THEOREM: remainder-times2-instance
$(((x * y) \textbf{ mod } (x * z)) = (x * (y \textbf{ mod } z)))$
$\wedge \quad (((x * z) \textbf{ mod } (y * z)) = ((x \textbf{ mod } y) * z))$

THEOREM: remainder-difference1
$((a \textbf{ mod } c) = (b \textbf{ mod } c))$
$\rightarrow \quad (((a - b) \textbf{ mod } c) = ((a \textbf{ mod } c) - (b \textbf{ mod } c)))$

DEFINITION:
double-remainder-induction $(a,\ b,\ c)$
$=$ **if** $c \simeq 0$ **then** 0
elseif $a < c$ **then** 0
elseif $b < c$ **then** 0
else double-remainder-induction $(a - c,\ b - c,\ c)$ **endif**

THEOREM: remainder-difference2
$(((a \textbf{ mod } c) = \texttt{0}) \wedge ((b \textbf{ mod } c) \neq \texttt{0}))$
$\rightarrow \quad (((a - b) \textbf{ mod } c)$
$\quad = \quad \textbf{if } b < a \textbf{ then } c - (b \textbf{ mod } c)$
$\quad\quad \textbf{else } \texttt{0} \textbf{ endif})$

THEOREM: remainder-difference3
$(((b \textbf{ mod } c) = \texttt{0}) \wedge ((a \textbf{ mod } c) \neq \texttt{0}))$
$\rightarrow \quad (((a - b) \textbf{ mod } c)$
$\quad = \quad \textbf{if } b < a \textbf{ then } a \textbf{ mod } c$
$\quad\quad \textbf{else } \texttt{0} \textbf{ endif})$

EVENT: Disable remainder-difference3.

THEOREM: equal-remainder-difference-0
$(((a - b) \textbf{ mod } c) = \texttt{0})$
$= \quad \textbf{if } b \leq a \textbf{ then } (a \textbf{ mod } c) = (b \textbf{ mod } c)$
$\quad \textbf{else t endif}$

EVENT: Disable equal-remainder-difference-0.

THEOREM: lessp-plus-fact
$(((b \textbf{ mod } x) = \texttt{0}) \wedge ((c \textbf{ mod } x) = \texttt{0}) \wedge (b < c) \wedge (a < x))$
$\rightarrow \quad (((a + b) < c) = \textbf{t})$

EVENT: Disable lessp-plus-fact.

THEOREM: remainder-plus-fact
$(((b \textbf{ mod } x) = \texttt{0}) \wedge ((c \textbf{ mod } x) = \texttt{0}) \wedge (a < x))$
$\rightarrow \quad (((a + b) \textbf{ mod } c) = (a + (b \textbf{ mod } c)))$

THEOREM: remainder-plus-times-times-proof
$(a < b)$
$\rightarrow \quad (((a + (b * c)) \textbf{ mod } (b * d))$
$\quad = \quad (a + ((b * c) \textbf{ mod } (b * d))))$

THEOREM: remainder-plus-times-times
$(a < b)$
$\rightarrow \quad ((((a + (b * c)) \textbf{ mod } (b * d))$
$\quad\quad = \quad (a + ((b * c) \textbf{ mod } (b * d))))$
$\quad\quad \wedge \quad (((a + (c * b)) \textbf{ mod } (d * b))$
$\quad\quad\quad = \quad (a + ((c * b) \textbf{ mod } (d * b)))))$

```
; REMAINDER-PLUS-TIMES-TIMES-INSTANCE is the completion of the rules
; TIMES-DISTRIBUTES-OVER-PLUS, REMAINDER-TIMES-TIMES and REMAINDER-PLUS-TIMES-TIMES
```

THEOREM: remainder-plus-times-times-instance
$(a < b)$
$\rightarrow \quad ((((a + (b * c)) + (b * d)) \textbf{ mod } (b * e))$
$\quad = \quad (a + (b * ((c + d) \textbf{ mod } e))))$
$\quad \wedge \quad (((a + (c * b)) + (d * b)) \textbf{ mod } (e * b))$
$\quad = \quad (a + (b * ((c + d) \textbf{ mod } e)))))$

THEOREM: remainder-remainder
$((b \textbf{ mod } a) = \texttt{0}) \rightarrow (((n \textbf{ mod } b) \textbf{ mod } a) = (n \textbf{ mod } a))$

THEOREM: remainder-1-arg1
$(\texttt{1} \textbf{ mod } x)$
$= \quad \textbf{if } x = \texttt{1} \textbf{ then } \texttt{0}$
$\quad \textbf{else } \texttt{1} \textbf{ endif}$

THEOREM: remainder-1-arg2
$(y \textbf{ mod } \texttt{1}) = \texttt{0}$

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

THEOREM: transitivity-of-divides
$(((a \textbf{ mod } b) = \texttt{0}) \wedge ((b \textbf{ mod } c) = \texttt{0})) \rightarrow ((a \textbf{ mod } c) = \texttt{0})$

```
; Define the available  theory of  remainder.   To  get the list of
; events to put in the theory,  evaluate the following  form in NQTHM at
; this point  in  the  script. This   form  lists all lemmas   which are
; globally enabled, and which have non-null lemma type.
;
;
; (let ((lemmas (lemmas)))
;   (remove-if-not (function (lambda (x)
;       (and (member x lemmas)
;    (not (assoc x disabled-lemmas))
;    (not (null (nth 2 (get x 'event))))
;    (not (member x (nth 2 (get 'addition 'event))))
;    (not (member x (nth 2 (get 'multiplication 'event)))))))
;   chronology))
```

EVENT: Let us define the theory *remainders* to consist of the following events: lessp-remainder, remainder-noop, remainder-of-non-number, remainder-zero, remainder-quotient-elim, remainder-add1, remainder-plus, equal-remainder-plus-0, remainder-times1, remainder-times1-instance, remainder-times2, remainder-times2-instance, remainder-difference1, remainder-difference2, remainder-plus-times-times, remainder-plus-times-times-instance, remainder-remainder, remainder-1-arg1, remainder-1-arg2, remainder-x-x.

```
; ---------- QUOTIENT, DIVIDES  ----------
```

THEOREM: quotient-noop
$(b = 1) \rightarrow ((a \div b) = \mathrm{fix}(a))$

THEOREM: quotient-of-non-number
$(a \notin \mathbf{N}) \rightarrow ((a \div n) = (0 \div n))$

THEOREM: quotient-zero
$(x \simeq 0) \rightarrow ((y \div x) = 0)$

THEOREM: quotient-add1
$((a \textbf{ mod } b) = 0)$
$\rightarrow \quad (((1 + a) \div b)$
$\quad = \quad \textbf{if } b = 1 \textbf{ then } 1 + (a \div b)$
$\quad\quad \textbf{else } a \div b \textbf{ endif})$

THEOREM: equal-quotient-0
$((a \div b) = 0) = ((b \simeq 0) \vee (a < b))$

THEOREM: quotient-sub1
$((a \not\simeq 0) \wedge (b \not\simeq 0))$
$\rightarrow \quad (((a - 1) \div b)$
$\quad = \quad \textbf{if } (a \textbf{ mod } b) = 0 \textbf{ then } (a \div b) - 1$
$\quad\quad \textbf{else } a \div b \textbf{ endif})$

THEOREM: quotient-plus-proof
$((b \textbf{ mod } c) = 0) \rightarrow (((a + b) \div c) = ((a \div c) + (b \div c)))$

THEOREM: quotient-plus
$((a \textbf{ mod } c) = 0)$
$\rightarrow \quad ((((a + b) \div c) = ((a \div c) + (b \div c)))$
$\quad \wedge \quad (((b + a) \div c) = ((a \div c) + (b \div c)))$
$\quad \wedge \quad (((x + y + a) \div c)$
$\quad\quad = \quad (((x + y) \div c) + (a \div c))))$

```
; I need QUOTIENT-TIMES-INSTANCE to prove the more general QUOTIENT-TIMES,
; but I want QUOTIENT-TIMES-INSTANCE to be tried first (i.e. come after
; QUOTIENT-TIMES in the event list.) So first, prove QUOTIENT-TIMES-INSTANCE-TEMP,
; then prove QUOTIENT-TIMES, and finally give QUOTIENT-TIMES-INSTANCE.
```

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

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

EVENT: Disable quotient-times-instance-temp.

THEOREM: quotient-times-proof
$((a \bmod c) = 0) \rightarrow (((a * b) \div c) = (b * (a \div c)))$

THEOREM: quotient-times
$((a \bmod c) = 0)$
$\rightarrow$ $((((a * b) \div c) = (b * (a \div c)))$
$\wedge$ $(((b * a) \div c) = (b * (a \div c))))$

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

THEOREM: quotient-times-times-proof
$((x * y) \div (x * z))$
$=$ **if** $x \simeq 0$ **then** 0
else $y \div z$ **endif**

THEOREM: quotient-times-times
$(((x * y) \div (x * z))$
$=$ **if** $x \simeq 0$ **then** 0

else $y \div z$ **endif**)
$\wedge$ (((($x * z$) $\div$ ($y * z$))
= **if** $z \simeq$ 0 **then** 0
else $x \div y$ **endif**)

EVENT: Disable quotient-times-times.

THEOREM: quotient-difference1
(((a **mod** c) = (b **mod** c))
$\rightarrow$ ((($a - b$) $\div$ c) = (($a \div c$) $-$ ($b \div c$)))

THEOREM: quotient-lessp-arg1
($a < b$) $\rightarrow$ (($a \div b$) = 0)

THEOREM: quotient-difference2
(((a **mod** c) = 0) $\wedge$ ((b **mod** c) $\neq$ 0))
$\rightarrow$ ((($a - b$) $\div$ c)
= **if** $b < a$ **then** ($a \div c$) $-$ (1 + ($b \div c$))
else 0 **endif**)

THEOREM: quotient-difference3
(((b **mod** c) = 0) $\wedge$ ((a **mod** c) $\neq$ 0))
$\rightarrow$ ((($a - b$) $\div$ c)
= **if** $b < a$ **then** ($a \div c$) $-$ ($b \div c$)
else 0 **endif**)

THEOREM: remainder-equals-its-first-argument
(a = (a **mod** b)) = (($a \in$ **N**) $\wedge$ (($b \simeq$ 0) $\vee$ ($a < b$)))

EVENT: Disable remainder-equals-its-first-argument.

THEOREM: quotient-remainder-times
((x **mod** ($a * b$)) $\div$ a) = (($x \div a$) **mod** b)

THEOREM: quotient-remainder
((c **mod** a) = 0) $\rightarrow$ (((b **mod** c) $\div$ a) = (($b \div a$) **mod** ($c \div a$)))

THEOREM: quotient-remainder-instance
((x **mod** ($a * b$)) $\div$ a) = (($x \div a$) **mod** b)

THEOREM: quotient-plus-fact
(((b **mod** x) = 0) $\wedge$ ((c **mod** x) = 0) $\wedge$ ($a < x$))
$\rightarrow$ ((($a + b$) $\div$ c) = ($b \div c$))

THEOREM: quotient-plus-times-times-proof
$(a < b)$
$\rightarrow \quad (((a + (b * c)) \div (b * d)) = ((b * c) \div (b * d)))$

THEOREM: quotient-plus-times-times
$(a < b)$
$\rightarrow \quad ((((a + (b * c)) \div (b * d)) = ((b * c) \div (b * d)))$
$\quad \wedge \quad (((a + (b * c)) \div (b * d))$
$\quad\quad = \quad ((b * c) \div (b * d))))$

```
; QUOTIENT-PLUS-TIMES-TIMES-INSTANCE is the completion of the rules
; QUOTIENT-TIMES-TIMES, QUOTIENT-PLUS-TIMES-TIMES and TIMES-DISTRIBUTES-OVER-PLUS
```

THEOREM: quotient-plus-times-times-instance
$(a < b)$
$\rightarrow \quad ((((a + (b * c) + (b * d)) \div (b * e))$
$\quad = \quad$ **if** $b \simeq 0$ **then** 0
$\quad\quad$ **else** $(c + d) \div e$ **endif**)
$\quad \wedge \quad (((a + (c * b) + (d * b)) \div (e * b))$
$\quad\quad = \quad$ **if** $b \simeq 0$ **then** 0
$\quad\quad\quad$ **else** $(d + c) \div e$ **endif**))

THEOREM: quotient-quotient
$((b \div a) \div c) = (b \div (a * c))$

THEOREM: leq-quotient
$(a < b) \rightarrow ((a \div c) \leq (b \div c))$

THEOREM: quotient-1-arg2
$(n \div 1) = \text{fix}\,(n)$

THEOREM: quotient-1-arg1-casesplit
$(n \simeq 0) \vee (n = 1) \vee (1 < n)$

THEOREM: quotient-1-arg1
$(1 \div n)$
$=$ **if** $n = 1$ **then** 1
else 0 **endif**

THEOREM: quotient-x-x
$(x \not\simeq 0) \rightarrow ((x \div x) = 1)$

THEOREM: lessp-quotient
$((i \div j) < i) = ((i \not\simeq 0) \wedge (j \neq 1))$

```
;; Metalemma to cancel quotient-times expressions

;; ex.
;; (quotient (times a b) (times c (times d a)))  ->
;;                                                (if (not (zerop a))
;;                                                    (quotient (fix b) (times c d))
;;                                                 (zero))
;;
```

DEFINITION:
cancel-quotient-times (x)
$=$ **if** $(\text{car}\,(x) = \texttt{'quotient})$
$\wedge$ $(\text{caadr}\,(x) = \texttt{'times})$
$\wedge$ $(\text{caaddr}\,(x) = \texttt{'times})$
then let *inboth* **be** bagint (times-fringe (cadr (x)),
times-fringe (caddr (x)))
in
if listp (*inboth*)
then list (`'if`,
and-not-zerop-tree (*inboth*),
list (`'quotient`,
times-tree (bagdiff (times-fringe (cadr (x)),
inboth)),
times-tree (bagdiff (times-fringe (caddr (x)),
inboth))),
`'(zero)`)
else x **endif endlet**
else x **endif**

THEOREM: zerop-makes-quotient-zero-bridge
$((\text{car}\,(x) = \texttt{'times})$
$\wedge$ $(\text{car}\,(y) = \texttt{'times})$
$\wedge$ $(\neg\ \text{eval\$}\,(\mathbf{t},$
and-not-zerop-tree (bagint (times-fringe (x), times-fringe (y))),
$a)))$
$\rightarrow$ $(((\text{eval\$}\,(\mathbf{t},\ \text{cadr}\,(x),\ a) * \text{eval\$}\,(\mathbf{t},\ \text{caddr}\,(x),\ a))$
$\div$ $(\text{eval\$}\,(\mathbf{t},\ \text{cadr}\,(y),\ a) * \text{eval\$}\,(\mathbf{t},\ \text{caddr}\,(y),\ a)))$
$=$ `0`)

EVENT: Disable zerop-makes-quotient-zero-bridge.

THEOREM: eval$-quotient-times-tree-bagdiff
$(\text{subbagp}\,(x,\ y) \wedge \text{subbagp}\,(x,\ z) \wedge \text{eval\$}\,(\mathbf{t},\ \text{and-not-zerop-tree}\,(x),\ a))$

$\rightarrow$ ((eval\$ (**t**, times-tree (bagdiff (*y*, *x*)), *a*)
$\div$ eval\$ (**t**, times-tree (bagdiff (*z*, *x*)), *a*))
= (eval\$ (**t**, times-tree (*y*), *a*) $\div$ eval\$ (**t**, times-tree (*z*), *a*)))

EVENT: Disable eval\$-quotient-times-tree-bagdiff.

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

```
; Define the available theory of quotient.  To get the list of events to
; put in the theory, evaluate the following form in NQTHM at  this point
; in the script. This form lists all  lemmas which are globally enabled,
; and which have non-null lemma type.
;
;
; (let ((lemmas (lemmas)))
;   (remove-if-not (function (lambda (x)
;       (and (member x lemmas)
;    (not (assoc x disabled-lemmas))
;    (not (null (nth 2 (get x 'event))))
;    (not (member x (nth 2 (get 'addition 'event))))
;    (not (member x (nth 2 (get 'multiplication 'event))))
;    (not (member x (nth 2 (get 'remainders 'event)))))))
;   chronology))
```

EVENT: Let us define the theory *quotients* to consist of the following events: quotient-noop, quotient-of-non-number, quotient-zero, quotient-add1, equal-quotient-0, quotient-sub1, quotient-plus, quotient-times, quotient-times-instance, quotient-difference1, quotient-lessp-arg1, quotient-difference2, quotient-difference3, quotient-remainder-times, quotient-remainder, quotient-remainder-instance, quotient-plus-times-times, quotient-plus-times-times-instance, quotient-quotient, quotient-1-arg2, quotient-1-arg1, quotient-x-x, lessp-quotient, correctness-of-cancel-quotient-times.

```
;;; exp, log, and gcd
```

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

DEFINITION:
log (*base*, *n*)
= **if** *base* < 2 **then** 0
elseif *n* ≃ 0 **then** 0
else 1 + log (*base*, *n* ÷ *base*) **endif**

DEFINITION:
gcd (*x*, *y*)
= **if** *x* ≃ 0 **then** fix (*y*)
elseif *y* ≃ 0 **then** *x*
elseif *x* < *y* **then** gcd (*x*, *y* − *x*)
else gcd (*x* − *y*, *y*) **endif**

THEOREM: remainder-exp
(*k* ≄ 0) → ((exp (*n*, *k*) **mod** *n*) = 0)

DEFINITION:
double-number-induction (*i*, *j*)
= **if** *i* ≃ 0 **then** 0
elseif *j* ≃ 0 **then** 0
else double-number-induction (*i* − 1, *j* − 1) **endif**

THEOREM: remainder-exp-exp
(*i* ≤ *j*) → ((exp (*a*, *j*) **mod** exp (*a*, *i*)) = 0)

THEOREM: quotient-exp
(*k* ≄ 0)
→ ((exp (*n*, *k*) ÷ *n*)
= **if** *n* ≃ 0 **then** 0
else exp (*n*, *k* − 1) **endif**)

THEOREM: exp-zero
(*k* ≃ 0) → (exp (*n*, *k*) = 1)

THEOREM: exp-add1
exp (*n*, 1 + *k*) = (*n* ∗ exp (*n*, *k*))

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

THEOREM: exp-0-arg1
exp (0, *k*)
= **if** *k* ≃ 0 **then** 1
else 0 **endif**

THEOREM: exp-1-arg1
exp (1, *k*) = 1

THEOREM: exp-0-arg2
$\exp(n, 0) = 1$

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

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

THEOREM: equal-exp-0
$(\exp(n, k) = 0) = ((n \simeq 0) \wedge (k \not\simeq 0))$

THEOREM: equal-exp-1
$(\exp(n, k) = 1)$
$=$ **if** $k \simeq 0$ **then t**
else $n = 1$ **endif**

THEOREM: exp-difference
$((c \leq b) \wedge (a \not\simeq 0)) \rightarrow (\exp(a, b - c) = (\exp(a, b) \div \exp(a, c)))$

EVENT: Let us define the theory *exponentiation* to consist of the following events: equal-exp-0, equal-exp-1, exp-exp, exp-add1, exp-times, exp-1-arg1, exp-zero, exp-0-arg2, exp-0-arg1, exp-difference, exp-plus, quotient-exp, remainder-exp-exp, remainder-exp.

THEOREM: equal-log-0
$(\log(base, n) = 0) = ((base < 2) \vee (n \simeq 0))$

THEOREM: log-0
$(n \simeq 0) \rightarrow (\log(base, n) = 0)$

THEOREM: log-1
$(1 < base) \rightarrow (\log(base, 1) = 1)$

DEFINITION:
double-log-induction $(base, a, b)$
$=$ **if** $base < 2$ **then** 0
elseif $a \simeq 0$ **then** 0
elseif $b \simeq 0$ **then** 0
else double-log-induction $(base, a \div base, b \div base)$ **endif**

THEOREM: leq-log-log
$(n \leq m) \rightarrow (\log(c, n) \leq \log(c, m))$

THEOREM: log-quotient
$(1 < c) \rightarrow (\log(c, n \div c) = (\log(c, n) - 1))$

THEOREM: log-quotient-times-proof
$(1 < c) \rightarrow (\log(c,\ n \div (c * m)) = (\log(c,\ n \div m) - 1))$

THEOREM: log-quotient-times
$(1 < c)$
$\rightarrow \quad ((\log(c,\ n \div (c * m)) = (\log(c,\ n \div m) - 1))$
$\quad \wedge \quad (\log(c,\ n \div (m * c)) = (\log(c,\ n \div m) - 1)))$

THEOREM: log-quotient-exp
$(1 < c) \rightarrow (\log(c,\ n \div \exp(c,\ m)) = (\log(c,\ n) - m))$

THEOREM: log-times-proof
$((1 < c) \wedge (n \not\simeq 0)) \rightarrow (\log(c,\ c * n) = (1 + \log(c,\ n)))$

THEOREM: log-times
$((1 < c) \wedge (n \not\simeq 0))$
$\rightarrow \quad ((\log(c,\ c * n) = (1 + \log(c,\ n)))$
$\quad \wedge \quad (\log(c,\ n * c) = (1 + \log(c,\ n))))$

THEOREM: log-times-exp-proof
$((1 < c) \wedge (n \not\simeq 0)) \rightarrow (\log(c,\ n * \exp(c,\ m)) = (\log(c,\ n) + m))$

THEOREM: log-times-exp
$((1 < c) \wedge (n \not\simeq 0))$
$\rightarrow \quad ((\log(c,\ n * \exp(c,\ m)) = (\log(c,\ n) + m))$
$\quad \wedge \quad (\log(c,\ \exp(c,\ m) * n) = (\log(c,\ n) + m)))$

THEOREM: log-exp
$(1 < c) \rightarrow (\log(c,\ \exp(c,\ n)) = (1 + n))$

EVENT: Let us define the theory *logs* to consist of the following events: log-exp, log-times-exp, log-times, log-quotient-exp, log-quotient-times, log-quotient, log-1, log-0, equal-log-0, exp-exp.

THEOREM: commutativity-of-gcd
$\gcd(b,\ a) = \gcd(a,\ b)$

DEFINITION:
single-number-induction(n)
$=$ **if** $n \simeq 0$ **then** 0
else single-number-induction$(n - 1)$ **endif**

THEOREM: gcd-0
$(\gcd(0,\ x) = \mathrm{fix}(x)) \wedge (\gcd(x,\ 0) = \mathrm{fix}(x))$

THEOREM: gcd-1
$(\gcd(\mathtt{1},\ x) = \mathtt{1}) \wedge (\gcd(x,\ \mathtt{1}) = \mathtt{1})$

THEOREM: equal-gcd-0
$(\gcd(a,\ b) = \mathtt{0}) = ((a \simeq \mathtt{0}) \wedge (b \simeq \mathtt{0}))$

THEOREM: lessp-gcd
$(b \not\simeq \mathtt{0}) \rightarrow (((b < \gcd(a,\ b)) = \mathbf{f}) \wedge ((b < \gcd(b,\ a)) = \mathbf{f}))$

THEOREM: gcd-plus-instance-temp-proof
$\gcd(a,\ a + b) = \gcd(a,\ b)$

THEOREM: gcd-plus-instance-temp
$(\gcd(a,\ a + b) = \gcd(a,\ b)) \wedge (\gcd(a,\ b + a) = \gcd(a,\ b))$

THEOREM: gcd-plus-proof
$((b\ \mathbf{mod}\ a) = \mathtt{0}) \rightarrow (\gcd(a,\ b + c) = \gcd(a,\ c))$

THEOREM: gcd-plus
$((b\ \mathbf{mod}\ a) = \mathtt{0})$
$\rightarrow\quad ((\gcd(a,\ b + c) = \gcd(a,\ c))$
$\quad\wedge\quad (\gcd(a,\ c + b) = \gcd(a,\ c))$
$\quad\wedge\quad (\gcd(b + c,\ a) = \gcd(a,\ c))$
$\quad\wedge\quad (\gcd(c + b,\ a) = \gcd(a,\ c)))$

THEOREM: gcd-plus-instance
$(\gcd(a,\ a + b) = \gcd(a,\ b)) \wedge (\gcd(a,\ b + a) = \gcd(a,\ b))$

THEOREM: remainder-gcd
$((a\ \mathbf{mod}\ \gcd(a,\ b)) = \mathtt{0}) \wedge ((b\ \mathbf{mod}\ \gcd(a,\ b)) = \mathtt{0})$

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

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

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

THEOREM: common-divisor-divides-gcd
$(((x\ \mathbf{mod}\ z) = \mathtt{0}) \wedge ((y\ \mathbf{mod}\ z) = \mathtt{0})) \rightarrow ((\gcd(x,\ y)\ \mathbf{mod}\ z) = \mathtt{0})$

```
; We prove ASSOCIATIVITY-OF-GCD and COMMUTATIVITY2-OF-GCD roughly the same way.
; Use GCD-IS-THE-GREATEST twice to show that each side of the equality is
; less than or equal to the other side.
```

THEOREM: associativity-of-gcd-zero-case
$((a \simeq 0) \vee (b \simeq 0) \vee (c \simeq 0)) \rightarrow (\gcd(\gcd(a,\ b),\ c) = \gcd(a,\ \gcd(b,\ c)))$

THEOREM: associativity-of-gcd
$\gcd(\gcd(a,\ b),\ c) = \gcd(a,\ \gcd(b,\ c))$

THEOREM: commutativity2-of-gcd-zero-case
$((a \simeq 0) \vee (b \simeq 0) \vee (c \simeq 0)) \rightarrow (\gcd(b,\ \gcd(a,\ c)) = \gcd(a,\ \gcd(b,\ c)))$

THEOREM: commutativity2-of-gcd
$\gcd(b,\ \gcd(a,\ c)) = \gcd(a,\ \gcd(b,\ c))$

THEOREM: gcd-x-x
$\gcd(x,\ x) = \mathrm{fix}(x)$

THEOREM: gcd-idempotence
$(\gcd(x,\ \gcd(x,\ y)) = \gcd(x,\ y)) \wedge (\gcd(y,\ \gcd(x,\ y)) = \gcd(x,\ y))$

EVENT: Let us define the theory *gcds* to consist of the following events: commutativity2-of-gcd, associativity-of-gcd, common-divisor-divides-gcd, distributivity-of-times-over-gcd, lessp-gcd, equal-gcd-0, gcd-0, gcd-idempotence, gcd-x-x, remainder-gcd, gcd-plus, gcd-plus-instance, gcd-1, commutativity-of-gcd.

EVENT: Let us define the theory *naturals* to consist of the following events: addition, multiplication, remainders, quotients, exponentiation, logs, gcds.

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

Index