

#|

Copyright (C) 1994 by Computational Logic, Inc. All Rights Reserved.

This script is hereby placed in the public domain, and therefore unlimited editing and redistribution is permitted.

NO WARRANTY

Computational Logic, Inc. PROVIDES ABSOLUTELY NO WARRANTY. THE EVENT SCRIPT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE SCRIPT IS WITH YOU. SHOULD THE SCRIPT PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT WILL Computational Logic, Inc. BE LIABLE TO YOU FOR ANY DAMAGES, ANY LOST PROFITS, LOST MONIES, OR OTHER SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THIS SCRIPT (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY THIRD PARTIES), EVEN IF YOU HAVE ADVISED US OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY.

|#

; William R. Bevier, Matt Kaufmann, and Matt Wilding

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

;; Jan 91 - some additional facts mostly having to do with GCD
;; added by MMW

;; Oct 90 - modified by MMW to eliminate theories

;; 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
 $((a + b) = (a + c)) = (\text{fix}(b) = \text{fix}(c))$ 

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  $\text{fix}(x) = \text{fix}(z)$  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

$$(y \simeq 0) \rightarrow ((x + y) = \text{fix}(x))$$

THEOREM: plus-add1-arg1

$$((1 + a) + b) = (1 + (a + b))$$

THEOREM: plus-add1-arg2

$$\begin{aligned} & (x + (1 + y)) \\ &= \text{if } y \in \mathbf{N} \text{ then } 1 + (x + y) \\ & \quad \text{else } 1 + x \text{ endif} \end{aligned}$$

THEOREM: associativity-of-plus

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

THEOREM: plus-difference-arg1

$$\begin{aligned} & ((a - b) + c) \\ &= \text{if } b < a \text{ then } (a + c) - b \\ & \quad \text{else } 0 + c \text{ endif} \end{aligned}$$

THEOREM: plus-difference-arg2

$$\begin{aligned} & (a + (b - c)) \\ &= \text{if } c < b \text{ then } (a + b) - c \\ & \quad \text{else } a + 0 \text{ endif} \end{aligned}$$

; ----- 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

$$\begin{aligned} &(((x + y) - (x + z)) = (y - z)) \\ \wedge &(((y + x) - (x + z)) = (y - z)) \\ \wedge &(((x + y) - (z + x)) = (y - z)) \\ \wedge &(((y + x) - (z + x)) = (y - z)) \end{aligned}$$

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

$$\begin{aligned} &(a - (b - 1)) \\ = &\text{if } b \simeq 0 \text{ then fix}(a) \\ &\text{elseif } a < b \text{ then } 0 \\ &\text{else } 1 + (a - b) \text{ endif} \end{aligned}$$

EVENT: Disable diff-sub1-arg2.

THEOREM: diff-diff-arg1

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

THEOREM: diff-diff-arg2

$$\begin{aligned} &(a - (b - c)) \\ = &\text{if } b < c \text{ then fix}(a) \\ &\text{else } (a + c) - b \text{ endif} \end{aligned}$$

; 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

$$\begin{aligned} &((b \leq a) \wedge (d \leq c)) \\ \rightarrow &(((a - b) - (c - d)) = ((a + d) - (b + c))) \end{aligned}$$

EVENT: Disable diff-diff-diff.

THEOREM: difference-lessp-arg1

$$(a < b) \rightarrow ((a - b) = 0)$$

EVENT: Disable difference-lessp-arg1.

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

; ----- PLUS-TREE and PLUS-FRIDGE -----
```

DEFINITION:

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

DEFINITION:

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

THEOREM: numberp-eval\$-plus

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

EVENT: Disable numberp-eval\$-plus.

THEOREM: numberp-eval\$-plus-tree

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

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

THEOREM: member-implies-plus-tree-greatereqp

$$(x \in y) \rightarrow (\text{eval}\$(\mathbf{t}, \text{plus-tree}(y), a) \not\prec \text{eval}\$(\mathbf{t}, x, a))$$

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

THEOREM: plus-tree-delete

$$\begin{aligned} & \text{eval}\$(\mathbf{t}, \text{plus-tree}(\text{delete}(x, y)), a) \\ &= \text{if } x \in y \text{ then eval}\$(\mathbf{t}, \text{plus-tree}(y), a) - \text{eval}\$(\mathbf{t}, x, a) \\ & \quad \text{else eval}\$(\mathbf{t}, \text{plus-tree}(y), a) \text{ endif} \end{aligned}$$

EVENT: Disable plus-tree-delete.

THEOREM: subbagp-implies-plus-tree-greatereqp

$$\text{subbagp}(x, y) \rightarrow (\text{eval\$}(\mathbf{t}, \text{plus-tree}(y), a) \not\leq \text{eval\$}(\mathbf{t}, \text{plus-tree}(x), a))$$

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

THEOREM: plus-tree-bagdiff

$$\begin{aligned} &\text{subbagp}(x, y) \\ &\rightarrow (\text{eval\$}(\mathbf{t}, \text{plus-tree}(\text{bagdiff}(y, x)), a) \\ &\quad = (\text{eval\$}(\mathbf{t}, \text{plus-tree}(y), a) - \text{eval\$}(\mathbf{t}, \text{plus-tree}(x), a))) \end{aligned}$$

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

$$\begin{aligned} &(\text{subbagp}(y, \text{plus-fringe}(z)) \\ &\quad \wedge (\text{eval\$}(\mathbf{t}, z, a) = \text{eval\$}(\mathbf{t}, \text{plus-tree}(\text{plus-fringe}(z)), a))) \\ &\rightarrow ((\text{eval\$}(\mathbf{t}, z, a) < \text{eval\$}(\mathbf{t}, \text{plus-tree}(y), a)) = \mathbf{f}) \end{aligned}$$

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

THEOREM: eval\$-plus-tree-append

$$\begin{aligned} &\text{eval\$}(\mathbf{t}, \text{plus-tree}(\text{append}(x, y)), a) \\ &= (\text{eval\$}(\mathbf{t}, \text{plus-tree}(x), a) + \text{eval\$}(\mathbf{t}, \text{plus-tree}(y), a)) \end{aligned}$$

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

THEOREM: plus-tree-plus-fringe

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

EVENT: Disable plus-tree-plus-fringe.

THEOREM: member-implies-numberp

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

EVENT: Disable member-implies-numberp.

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

EVENT: Disable cadr-eval\$-list.

THEOREM: eval\$-quote
 $\text{eval\$}(\mathbf{t}, \text{cons}('quote, args), a) = \text{car}(args)$

EVENT: Disable eval\$-quote.

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

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)
 $= \text{if listp}(x) \wedge (\text{car}(x) = 'equal)$
 $\text{then if listp}(\text{cadr}(x))$
 $\wedge (\text{caadr}(x) = 'plus)$
 $\wedge \text{listp}(\text{caddr}(x))$
 $\wedge (\text{caaddr}(x) = 'plus)$
 $\text{then list}('equal,$
 $\text{plus-tree}(\text{bagdiff}(\text{plus-fringe}(\text{cadr}(x)),$
 $\text{bagint}(\text{plus-fringe}(\text{cadr}(x)),$
 $\text{plus-fringe}(\text{caddr}(x))))),$
 $\text{plus-tree}(\text{bagdiff}(\text{plus-fringe}(\text{caddr}(x)),$
 $\text{bagint}(\text{plus-fringe}(\text{cadr}(x)),$
 $\text{plus-fringe}(\text{caaddr}(x))))))$
 $\text{elseif listp}(\text{cadr}(x))$
 $\wedge (\text{caadr}(x) = 'plus)$

```

       $\wedge$  (caddr( $x$ )  $\in$  plus-fringe(cadr( $x$ )))
then list('if,
      list('numberp, cadr( $x$ )),
      list('equal,
      plus-tree(delete(cadr( $x$ ), plus-fringe(cadr( $x$ )))),
      '0),
      list('quote, f))
elseif listp(cadr( $x$ ))
       $\wedge$  (caaddr( $x$ ) = 'plus)
       $\wedge$  (cadr( $x$ )  $\in$  plus-fringe(cadr( $x$ )))
then list('if,
      list('numberp, cadr( $x$ )),
      list('equal,
      '0,
      plus-tree(delete(cadr( $x$ ), plus-fringe(cadr( $x$ ))))),
      list('quote, f))
else  $x$  endif
else  $x$  endif

```

THEOREM: correctness-of-cancel-equal-plus

$\text{eval}\$(\mathbf{t}, x, a) = \text{eval}\$(\mathbf{t}, \text{cancel-equal-plus}(x), a)$

; ----- 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))
        ∧ (caadr (x) = 'plus)
        ∧ (caddr (x) ∈ plus-fringe (cadr (x)))
    then plus-tree (delete (caddr (x), plus-fringe (cadr (x))))
    elseif listp (caddr (x))
        ∧ (caaddr (x) = 'plus)
        ∧ (cadr (x) ∈ plus-fringe (caddr (x))) then '0
    else x endif
else x endif

```

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

; ----- 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
 $((y \in \mathbf{N}) \wedge (y \not\prec x)) \rightarrow ((x + (y - x)) = y)$

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

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

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

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

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

THEOREM: difference-x-x

$$(x - x) = 0$$

; ----- LESSP -----

THEOREM: lessp-difference-cancellation

$$\begin{aligned} & ((a - c) < (b - c)) \\ = & \text{ if } c \leq a \text{ then } a < b \\ & \text{ else } c < b \text{ endif} \end{aligned}$$

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( $x$ ) ∧ (car( $x$ ) = 'lessp)
   then if listp(cadr( $x$ ))
        ∧ (caadr( $x$ ) = 'plus)
        ∧ listp(caddr( $x$ ))
        ∧ (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
 $\text{eval}\$(\mathbf{t}, x, a) = \text{eval}\$(\mathbf{t}, \text{cancel-lessp-plus}(x), a)$

; ----- TIMES -----

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

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

```
; (prove-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
 $(x * (1 + y))$
 $= \text{if } y \in \mathbf{N} \text{ then } x + (x * y)$
else 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

$$\begin{aligned} &(((a - b) * c) = ((a * c) - (b * c))) \\ \wedge \quad &((a * (b - c)) = ((a * b) - (a * c))) \end{aligned}$$

THEOREM: times-quotient-proof

$$((x \neq 0) \wedge ((y \mathbf{mod} x) = 0)) \rightarrow (((y \div x) * x) = \mathbf{fix}(y))$$

THEOREM: times-quotient

$$\begin{aligned} &((y \neq 0) \wedge ((x \mathbf{mod} y) = 0)) \\ \rightarrow \quad &(((x \div y) * y) = \mathbf{fix}(x)) \wedge ((y * (x \div y)) = \mathbf{fix}(x)) \end{aligned}$$

THEOREM: times-1-arg1

$$(1 * x) = \mathbf{fix}(x)$$

THEOREM: lessp-times1-proof

$$((a < b) \wedge (c \neq 0)) \rightarrow ((a < (b * c)) = \mathbf{t})$$

THEOREM: lessp-times1

$$\begin{aligned} &((a < b) \wedge (c \neq 0)) \\ \rightarrow \quad &(((a < (b * c)) = \mathbf{t}) \wedge ((a < (c * b)) = \mathbf{t})) \end{aligned}$$

THEOREM: lessp-times2-proof

$$((a \leq b) \wedge (c \neq 0)) \rightarrow (((b * c) < a) = \mathbf{f})$$

THEOREM: lessp-times2

$$\begin{aligned} &((a \leq b) \wedge (c \neq 0)) \\ \rightarrow \quad &(((b * c) < a) = \mathbf{f}) \wedge (((c * b) < a) = \mathbf{f})) \end{aligned}$$

THEOREM: lessp-times3-proof1

$$((a \neq 0) \wedge (1 < b)) \rightarrow (a < (a * b))$$

THEOREM: lessp-times3-proof2

$$(a < (a * b)) \rightarrow ((a \neq 0) \wedge (1 < b))$$

THEOREM: lessp-times3

$$\begin{aligned} &((a < (a * b)) = ((a \neq 0) \wedge (1 < b))) \\ \wedge \quad &((a < (b * a)) = ((a \neq 0) \wedge (1 < b))) \end{aligned}$$

THEOREM: lessp-times-cancellation-proof

$$((x * z) < (y * z)) = ((z \neq 0) \wedge (x < y))$$

THEOREM: lessp-times-cancellation1

$$\begin{aligned}
& (((x * z) < (y * z)) = ((z \neq 0) \wedge (x < y))) \\
& \wedge (((z * x) < (y * z)) = ((z \neq 0) \wedge (x < y))) \\
& \wedge (((x * z) < (z * y)) = ((z \neq 0) \wedge (x < y))) \\
& \wedge (((z * x) < (z * y)) = ((z \neq 0) \wedge (x < y)))
\end{aligned}$$

EVENT: Disable lessp-times-cancellation1.

THEOREM: lessp-plus-times-proof

$$(x < a) \rightarrow (((x + (a * b)) < (a * c)) = (b < c))$$

THEOREM: lessp-plus-times1

$$\begin{aligned}
& (((a + (b * c)) < b) = ((a < b) \wedge (c \simeq 0))) \\
& \wedge (((a + (c * b)) < b) = ((a < b) \wedge (c \simeq 0))) \\
& \wedge (((c * b) + a) < b) = ((a < b) \wedge (c \simeq 0))) \\
& \wedge (((b * c) + a) < b) = ((a < b) \wedge (c \simeq 0)))
\end{aligned}$$

THEOREM: lessp-plus-times2

$$\begin{aligned}
& ((a \neq 0) \wedge (x < a)) \\
& \rightarrow (((x + (a * b)) < (a * c)) = (b < c)) \\
& \quad \wedge (((x + (b * a)) < (a * c)) = (b < c)) \\
& \quad \wedge (((x + (a * b)) < (c * a)) = (b < c)) \\
& \quad \wedge (((x + (b * a)) < (c * a)) = (b < c)) \\
& \quad \wedge (((a * b) + x) < (a * c)) = (b < c)) \\
& \quad \wedge (((b * a) + x) < (a * c)) = (b < c)) \\
& \quad \wedge (((a * b) + x) < (c * a)) = (b < c)) \\
& \quad \wedge (((b * a) + x) < (c * a)) = (b < c))
\end{aligned}$$

THEOREM: lessp-1-times

$$\begin{aligned}
& (1 < (a * b)) \\
& = (\neg ((a \simeq 0) \vee (b \simeq 0) \vee ((a = 1) \wedge (b = 1))))
\end{aligned}$$

```

;;; 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  $x \simeq \mathbf{nil}$  then ''1
    elseif  $\text{cdr}(x) \simeq \mathbf{nil}$  then list('fix, car( $x$ ))
    elseif  $\text{caddr}(x) \simeq \mathbf{nil}$  then list('times, car( $x$ ), caddr( $x$ ))
    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-zero-tree(x)

```
=  if  $x \simeq \mathbf{nil}$  then '(false)
    elseif  $\text{cdr}(x) \simeq \mathbf{nil}$  then list('zerop, car( $x$ ))
    elseif  $\text{caddr}(x) \simeq \mathbf{nil}$ 
    then list('or, list('zerop, car( $x$ )), list('zerop, caddr( $x$ )))
    else list('or, list('zerop, car( $x$ )), or-zero-tree(cdr( $x$ ))) endif
```

DEFINITION:

and-not-zero-tree(x)

```
=  if  $x \simeq \mathbf{nil}$  then '(true)
    elseif  $\text{cdr}(x) \simeq \mathbf{nil}$  then list('not, list('zerop, car( $x$ )))
    else list('and,
              list('not, list('zerop, car( $x$ ))),
              and-not-zero-tree(cdr( $x$ ))) endif
```

THEOREM: numberp-eval\$-times

$(\text{car}(x) = \text{'times}) \rightarrow (\text{eval}\$(\mathbf{t}, x, a) \in \mathbf{N})$

EVENT: Disable numberp-eval\$-times.

THEOREM: eval\$-times

$(\text{car}(x) = \text{'times})$
 $\rightarrow (\text{eval}\$(\mathbf{t}, x, a) = (\text{eval}\$(\mathbf{t}, \text{cadr}(x), a) * \text{eval}\$(\mathbf{t}, \text{caddr}(x), a)))$

EVENT: Disable eval\$-times.

THEOREM: eval\$-or

$(\text{car}(x) = \text{'or})$
 $\rightarrow (\text{eval}\$(\mathbf{t}, x, a) = (\text{eval}\$(\mathbf{t}, \text{cadr}(x), a) \vee \text{eval}\$(\mathbf{t}, \text{caddr}(x), a)))$

EVENT: Disable eval\$-or.

THEOREM: eval\$-equal

$(\text{car}(x) = \text{'equal})$

$\rightarrow (\text{eval\$}(\mathbf{t}, x, a) = (\text{eval\$}(\mathbf{t}, \text{cadr}(x), a) = \text{eval\$}(\mathbf{t}, \text{caddr}(x), a)))$

EVENT: Disable eval\$-equal.

THEOREM: eval\$-lessp

$(\text{car}(x) = \text{'lessp})$

$\rightarrow (\text{eval\$}(\mathbf{t}, x, a) = (\text{eval\$}(\mathbf{t}, \text{cadr}(x), a) < \text{eval\$}(\mathbf{t}, \text{caddr}(x), a)))$

EVENT: Disable eval\$-lessp.

THEOREM: eval\$-quotient

$(\text{car}(x) = \text{'quotient})$

$\rightarrow (\text{eval\$}(\mathbf{t}, x, a) = (\text{eval\$}(\mathbf{t}, \text{cadr}(x), a) \div \text{eval\$}(\mathbf{t}, \text{caddr}(x), a)))$

EVENT: Disable eval\$-quotient.

THEOREM: eval\$-if

$(\text{car}(x) = \text{'if})$

$\rightarrow (\text{eval\$}(\mathbf{t}, x, a)$

$= \text{if eval\$}(\mathbf{t}, \text{cadr}(x), a) \text{ then eval\$}(\mathbf{t}, \text{caddr}(x), a)$
 $\text{else eval\$}(\mathbf{t}, \text{caddr}(x), a) \text{ endif})$

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

$(a \neq 0) \rightarrow (((a * x) \neq (a * y)) = (x \neq y))$

THEOREM: infer-equality-from-not-lessp

$((a \in \mathbf{N}) \wedge (b \in \mathbf{N})) \rightarrow (((a \neq b) \wedge (b \neq a)) = (a = b))$

THEOREM: equal-times-arg1

$(a \neq 0) \rightarrow (((a * x) = (a * y)) = (\text{fix}(x) = \text{fix}(y)))$

EVENT: Disable equal-times-arg1.

THEOREM: equal-times-bridge

$((a * b) = (c * (a * d))) = ((a \simeq 0) \vee (\text{fix}(b) = (c * d)))$

EVENT: Disable equal-times-bridge.

THEOREM: eval\$-times-member

$$\begin{aligned} & (e \in x) \\ \rightarrow & \text{eval\$}(\mathbf{t}, \text{times-tree}(x), a) \\ & = \text{eval\$}(\mathbf{t}, e, a) * \text{eval\$}(\mathbf{t}, \text{times-tree}(\text{delete}(e, x)), a)) \end{aligned}$$

EVENT: Disable eval\$-times-member.

THEOREM: zerop-makes-times-tree-zero

$$\begin{aligned} & ((\neg \text{eval\$}(\mathbf{t}, \text{and-not-zerop-tree}(x), a)) \wedge \text{subbagp}(x, y)) \\ \rightarrow & \text{eval\$}(\mathbf{t}, \text{times-tree}(y), a) = 0 \end{aligned}$$

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

THEOREM: or-zerop-tree-is-not-zerop-tree

$$\text{eval\$}(\mathbf{t}, \text{or-zerop-tree}(x), a) = (\neg \text{eval\$}(\mathbf{t}, \text{and-not-zerop-tree}(x), a))$$

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

THEOREM: zerop-makes-times-tree-zero2

$$\begin{aligned} & (\text{eval\$}(\mathbf{t}, \text{or-zerop-tree}(x), a) \wedge \text{subbagp}(x, y)) \\ \rightarrow & \text{eval\$}(\mathbf{t}, \text{times-tree}(y), a) = 0 \end{aligned}$$

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

THEOREM: times-tree-append

$$\begin{aligned} & \text{eval\$}(\mathbf{t}, \text{times-tree}(\text{append}(x, y)), a) \\ & = \text{eval\$}(\mathbf{t}, \text{times-tree}(x), a) * \text{eval\$}(\mathbf{t}, \text{times-tree}(y), a) \end{aligned}$$

EVENT: Disable times-tree-append.

THEOREM: times-tree-of-times-fringe

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

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

DEFINITION:

$$\begin{aligned} & \text{cancel-lessp-times}(x) \\ = & \text{if}(\text{car}(x) = \text{'lessp}) \\ & \wedge (\text{caadr}(x) = \text{'times}) \\ & \wedge (\text{caaddr}(x) = \text{'times}) \end{aligned}$$


```

then if listp (bagint (times-fringe (cadr (x)), times-fringe (caddr (x))))
  then list ('and,
    and-not-zerop-tree (bagint (times-fringe (cadr (x)),
      times-fringe (caddr (x)))),
    list ('lessp,
      times-tree (bagdiff (times-fringe (cadr (x)),
        bagint (times-fringe (cadr (x)),
          times-fringe (caddr (x))))),
      times-tree (bagdiff (times-fringe (caddr (x)),
        bagint (times-fringe (cadr (x)),
          times-fringe (caddr (x)))))))
  else x endif
else x endif

```

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

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

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

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** ($\text{car}(x) = \text{'equal}$)
 $\quad \wedge (\text{caadr}(x) = \text{'times})$
 $\quad \wedge (\text{caaddr}(x) = \text{'times})$
then if listp (bagint (times-fringe (cadr (x)), times-fringe (caddr (x))))
 $\quad$ **then** list ('**or**,

```

or-zero-tree (bagint (times-fringe (cadr (x)),
                      times-fringe (caddr (x)))))
list ('equal,
      times-tree (bagdiff (times-fringe (cadr (x)),
                          bagint (times-fringe (cadr (x)),
                                times-fringe (caddr (x)))))
      times-tree (bagdiff (times-fringe (caddr (x)),
                          bagint (times-fringe (cadr (x)),
                                times-fringe (caddr (x)))))
      else x endif
else x endif

```

THEOREM: zero-tree-makes-equal-true-bridge

```

((car (x) = 'times)
 & (car (y) = 'times)
 & eval$ (t, or-zero-tree (bagint (times-fringe (x), times-fringe (y))), a))
→ ((eval$ (t, cadr (x), a) * eval$ (t, caddr (x), a))
    = (eval$ (t, cadr (y), a) * eval$ (t, caddr (y), a)))
    = t)

```

EVENT: Disable zero-tree-makes-equal-true-bridge.

THEOREM: eval-equal-times-tree-bagdiff

```

(subbagp (x, y) & subbagp (x, z) & (¬ eval$ (t, or-zero-tree (x), a)))
→ ((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)
 & subbagp (z, y)
 & (eval$ (t, times-tree (x), a) ≠ eval$ (t, times-tree (y), a)))
→ (eval$ (t, times-tree (bagdiff (x, z)), a)
    ≠ 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)
 & (car (y) = 'times)
 & ((eval$ (t, cadr (x), a) * eval$ (t, caddr (x), a))
    ≠ eval$ (t, cadr (y), a) * eval$ (t, caddr (y), a)))

```

$$\begin{aligned}
& \rightarrow \begin{aligned} & \neq \text{eval\$}(\mathbf{t}, \text{cadr}(y), a) * \text{eval\$}(\mathbf{t}, \text{caddr}(y), a))) \\ & (\text{eval\$}(\mathbf{t}, \\ & \quad \text{times-tree}(\text{bagdiff}(\text{times-fringe}(x), \\ & \quad \quad \text{bagint}(\text{times-fringe}(x), \text{times-fringe}(y))), \\ & \quad a) \\ & \neq \text{eval\$}(\mathbf{t}, \\ & \quad \text{times-tree}(\text{bagdiff}(\text{times-fringe}(y), \\ & \quad \quad \text{bagint}(\text{times-fringe}(x), \\ & \quad \quad \quad \text{times-fringe}(y))), \\ & \quad a)) \end{aligned}
\end{aligned}$$

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

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

; ----- REMAINDER -----

THEOREM: lessp-remainder
 $((x \mathbf{mod} y) < y) = (y \neq 0)$

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

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

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

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
 $((y \neq 0) \wedge (x \in \mathbf{N})) \rightarrow (((x \mathbf{mod} y) + (y * (x \div y))) = x)$

```

; (prove-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

$$((a \bmod b) = 0) \rightarrow (((1 + a) \bmod b) = (1 \bmod b))$$

THEOREM: remainder-plus-proof

$$((b \bmod c) = 0) \rightarrow ((a + b) \bmod c) = (a \bmod c)$$

THEOREM: remainder-plus

$$\begin{aligned}
& ((a \bmod c) = 0) \\
\rightarrow & (((a + b) \bmod c) = (b \bmod c)) \\
& \wedge (((b + a) \bmod c) = (b \bmod c)) \\
& \wedge (((x + y + a) \bmod c) = ((x + y) \bmod c))
\end{aligned}$$

THEOREM: equal-remainder-plus-0-proof

$$((a \bmod c) = 0) \rightarrow (((a + b) \bmod c) = 0) = ((b \bmod c) = 0)$$

THEOREM: equal-remainder-plus-0

$$\begin{aligned}
& ((a \bmod c) = 0) \\
\rightarrow & (((((a + b) \bmod c) = 0) = ((b \bmod c) = 0)) \\
& \wedge (((((b + a) \bmod c) = 0) = ((b \bmod c) = 0)) \\
& \wedge (((((x + y + a) \bmod c) = 0) \\
& \quad = (((x + y) \bmod c) = 0)))
\end{aligned}$$

THEOREM: equal-remainder-plus-remainder-proof

$$(a < c) \rightarrow (((a + b) \bmod c) = (b \bmod c)) = (a \simeq 0)$$

THEOREM: equal-remainder-plus-remainder

$$\begin{aligned}
& (a < c) \\
\rightarrow & (((((a + b) \bmod c) = (b \bmod c)) = (a \simeq 0)) \\
& \wedge (((((b + a) \bmod c) = (b \bmod c)) = (a \simeq 0)))
\end{aligned}$$

EVENT: Disable equal-remainder-plus-remainder.

THEOREM: remainder-times1-proof

$$((b \bmod c) = 0) \rightarrow ((a * b) \bmod c) = 0$$

THEOREM: remainder-times1

$$\begin{aligned}
& ((b \bmod c) = 0) \\
\rightarrow & (((a * b) \bmod c) = 0) \wedge (((b * a) \bmod c) = 0)
\end{aligned}$$

THEOREM: remainder-times1-instance-proof

$$((x * y) \bmod y) = 0$$

THEOREM: remainder-times1-instance

$$(((x * y) \bmod y) = 0) \wedge (((x * y) \bmod x) = 0)$$

THEOREM: remainder-times-times-proof

$$((x * y) \bmod (x * z)) = (x * (y \bmod z))$$

THEOREM: remainder-times-times

$$\begin{aligned} &(((x * y) \bmod (x * z)) = (x * (y \bmod z))) \\ \wedge &(((x * z) \bmod (y * z)) = ((x \bmod y) * z)) \end{aligned}$$

EVENT: Disable remainder-times-times.

THEOREM: remainder-times2-proof

$$((a \bmod z) = 0) \rightarrow ((a \bmod (z * y)) = (z * ((a \div z) \bmod y)))$$

THEOREM: remainder-times2

$$\begin{aligned} &((a \bmod z) = 0) \\ \rightarrow &(((a \bmod (y * z)) = (z * ((a \div z) \bmod y))) \\ &\wedge ((a \bmod (z * y)) = (z * ((a \div z) \bmod y)))) \end{aligned}$$

THEOREM: remainder-times2-instance

$$\begin{aligned} &(((x * y) \bmod (x * z)) = (x * (y \bmod z))) \\ \wedge &(((x * z) \bmod (y * z)) = ((x \bmod y) * z)) \end{aligned}$$

THEOREM: remainder-difference1

$$\begin{aligned} &((a \bmod c) = (b \bmod c)) \\ \rightarrow &(((a - b) \bmod c) = ((a \bmod c) - (b \bmod c))) \end{aligned}$$

DEFINITION:

double-remainder-induction(a, b, c)

```
=  if c ≃ 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

$$\begin{aligned} &(((a \bmod c) = 0) \wedge ((b \bmod c) \neq 0)) \\ \rightarrow &(((a - b) \bmod c) \\ &= \text{if } b < a \text{ then } c - (b \bmod c) \\ &\text{else } 0 \text{ endif}) \end{aligned}$$

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

EVENT: Disable remainder-difference3.

THEOREM: equal-remainder-difference-0
 $((a - b) \bmod c) = 0$
 $= \text{if } b \leq a \text{ then } (a \bmod c) = (b \bmod c)$
 else t endif

EVENT: Disable equal-remainder-difference-0.

THEOREM: lessp-plus-fact
 $((b \bmod x) = 0) \wedge ((c \bmod x) = 0) \wedge (b < c) \wedge (a < x)$
 $\rightarrow ((a + b) < c) = \mathbf{t}$

EVENT: Disable lessp-plus-fact.

THEOREM: remainder-plus-fact
 $((b \bmod x) = 0) \wedge ((c \bmod x) = 0) \wedge (a < x)$
 $\rightarrow ((a + b) \bmod c) = (a + (b \bmod c))$

THEOREM: remainder-plus-times-times-proof
 $(a < b)$
 $\rightarrow (((a + (b * c)) \bmod (b * d))$
 $= (a + ((b * c) \bmod (b * d))))$

THEOREM: remainder-plus-times-times
 $(a < b)$
 $\rightarrow (((a + (b * c)) \bmod (b * d))$
 $= (a + ((b * c) \bmod (b * d))))$
 $\wedge (((a + (c * b)) \bmod (d * b))$
 $= (a + ((c * b) \bmod (d * b))))$

; 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
 $(a < b)$
 $\rightarrow (((a + (b * c) + (b * d)) \bmod (b * e))$

$$\begin{aligned}
&= (a + (b * ((c + d) \bmod e))) \\
\wedge \quad &(((a + (c * b) + (d * b)) \bmod (e * b)) \\
&= (a + (b * ((c + d) \bmod e))))
\end{aligned}$$

THEOREM: remainder-remainder

$$((b \bmod a) = 0) \rightarrow (((n \bmod b) \bmod a) = (n \bmod a))$$

THEOREM: remainder-1-arg1

$$\begin{aligned}
&(1 \bmod x) \\
&= \text{if } x = 1 \text{ then } 0 \\
&\quad \text{else } 1 \text{ endif}
\end{aligned}$$

THEOREM: remainder-1-arg2

$$(y \bmod 1) = 0$$

THEOREM: remainder-x-x

$$(x \bmod x) = 0$$

THEOREM: transitivity-of-divides

$$(((a \bmod b) = 0) \wedge ((b \bmod c) = 0)) \rightarrow ((a \bmod c) = 0)$$

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

THEOREM: quotient-noop

$$(b = 1) \rightarrow ((a \div b) = \text{fix}(a))$$

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

$$\begin{aligned}
&((a \bmod b) = 0) \\
&\rightarrow (((1 + a) \div b) \\
&\quad = \text{if } b = 1 \text{ then } 1 + (a \div b) \\
&\quad \text{else } a \div b \text{ endif})
\end{aligned}$$

THEOREM: equal-quotient-0

$$((a \div b) = 0) = ((b \simeq 0) \vee (a < b))$$

THEOREM: quotient-sub1

$$\begin{aligned}
&((a \neq 0) \wedge (b \neq 0)) \\
&\rightarrow (((a - 1) \div b) \\
&\quad = \text{if } (a \bmod b) = 0 \text{ then } (a \div b) - 1 \\
&\quad \text{else } a \div b \text{ endif})
\end{aligned}$$

THEOREM: quotient-plus-proof

$$((b \bmod c) = 0) \rightarrow (((a + b) \div c) = ((a \div c) + (b \div c)))$$

THEOREM: quotient-plus

$$\begin{aligned} & ((a \bmod c) = 0) \\ \rightarrow & (((a + b) \div c) = ((a \div c) + (b \div c))) \\ & \wedge (((b + a) \div c) = ((a \div c) + (b \div c))) \\ & \wedge (((x + y + a) \div c) \\ & \quad = (((x + y) \div c) + (a \div c))) \end{aligned}$$

; 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

$$\begin{aligned} & ((y * x) \div y) \\ = & \text{ if } y \simeq 0 \text{ then } 0 \\ & \text{ else fix}(x) \text{ endif} \end{aligned}$$

THEOREM: quotient-times-instance-temp

$$\begin{aligned} & (((y * x) \div y) \\ = & \text{ if } y \simeq 0 \text{ then } 0 \\ & \text{ else fix}(x) \text{ endif}) \\ \wedge & (((x * y) \div y) \\ = & \text{ if } y \simeq 0 \text{ then } 0 \\ & \text{ else fix}(x) \text{ endif}) \end{aligned}$$

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

$$\begin{aligned} & ((a \bmod c) = 0) \\ \rightarrow & (((a * b) \div c) = (b * (a \div c))) \\ & \wedge (((b * a) \div c) = (b * (a \div c))) \end{aligned}$$

THEOREM: quotient-times-instance

$$\begin{aligned} & (((y * x) \div y) \\ = & \text{ if } y \simeq 0 \text{ then } 0 \\ & \text{ else fix}(x) \text{ endif}) \\ \wedge & (((x * y) \div y) \\ = & \text{ if } y \simeq 0 \text{ then } 0 \\ & \text{ else fix}(x) \text{ endif}) \end{aligned}$$

THEOREM: quotient-times-times-proof

$$\begin{aligned} & ((x * y) \div (x * z)) \\ &= \text{if } x \simeq 0 \text{ then } 0 \\ & \quad \text{else } y \div z \text{ endif} \end{aligned}$$

THEOREM: quotient-times-times

$$\begin{aligned} & (((x * y) \div (x * z)) \\ &= \text{if } x \simeq 0 \text{ then } 0 \\ & \quad \text{else } y \div z \text{ endif}) \\ \wedge \quad & (((x * z) \div (y * z)) \\ &= \text{if } z \simeq 0 \text{ then } 0 \\ & \quad \text{else } x \div y \text{ endif}) \end{aligned}$$

EVENT: Disable quotient-times-times.

THEOREM: quotient-difference1

$$\begin{aligned} & ((a \bmod c) = (b \bmod c)) \\ \rightarrow \quad & (((a - b) \div c) = ((a \div c) - (b \div c))) \end{aligned}$$

THEOREM: quotient-lessp-arg1

$$(a < b) \rightarrow ((a \div b) = 0)$$

THEOREM: quotient-difference2

$$\begin{aligned} & (((a \bmod c) = 0) \wedge ((b \bmod c) \neq 0)) \\ \rightarrow \quad & (((a - b) \div c) \\ &= \text{if } b < a \text{ then } (a \div c) - (1 + (b \div c)) \\ & \quad \text{else } 0 \text{ endif}) \end{aligned}$$

THEOREM: quotient-difference3

$$\begin{aligned} & (((b \bmod c) = 0) \wedge ((a \bmod c) \neq 0)) \\ \rightarrow \quad & (((a - b) \div c) \\ &= \text{if } b < a \text{ then } (a \div c) - (b \div c) \\ & \quad \text{else } 0 \text{ endif}) \end{aligned}$$

THEOREM: remainder-equals-its-first-argument

$$(a = (a \bmod b)) = ((a \in \mathbf{N}) \wedge ((b \simeq 0) \vee (a < b)))$$

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

THEOREM: quotient-remainder-times

$$((x \bmod (a * b)) \div a) = ((x \div a) \bmod b)$$

THEOREM: quotient-remainder

$$((c \bmod a) = 0) \rightarrow (((b \bmod c) \div a) = ((b \div a) \bmod (c \div a)))$$

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

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

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

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

; 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
 $(a < b)$
 $\rightarrow (((a + (b * c)) + (b * d)) \div (b * e))$
 $= \text{if } b \simeq 0 \text{ then } 0$
 $\text{else } (c + d) \div e \text{ endif}$
 $\wedge (((a + (c * b)) + (d * b)) \div (e * b))$
 $= \text{if } b \simeq 0 \text{ then } 0$
 $\text{else } (d + c) \div e \text{ endif})$

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
 $(1 \div n)$
 $= \text{if } n = 1 \text{ then } 1$
 $\text{else } 0 \text{ endif}$

THEOREM: quotient-x-x
 $(x \neq 0) \rightarrow ((x \div x) = 1)$

THEOREM: lessp-quotient
 $((i \div j) < i) = ((i \neq 0) \wedge (j \neq 1))$

;; 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 (car (x) = 'quotient)
    ^  (caadr (x) = 'times)
    ^  (caaddr (x) = 'times)
  then if listp (bagint (times-fringe (cadr (x)), times-fringe (caddr (x))))
    then list ('if,
               and-not-zerop-tree (bagint (times-fringe (cadr (x)),
                                             times-fringe (caddr (x)))),
               list ('quotient,
                     times-tree (bagdiff (times-fringe (cadr (x)),
                                           bagint (times-fringe (cadr (x)),
                                                  times-fringe (caddr (x))))),
                     times-tree (bagdiff (times-fringe (caddr (x)),
                                           bagint (times-fringe (cadr (x)),
                                                  times-fringe (caddr (x))))),
                     ' (zero)))
    else x endif
else x endif
```

THEOREM: zerop-makes-quotient-zero-bridge

```
((car (x) = 'times)
 ^  (car (y) = 'times)
 ^  (¬ eval$ (t,
              and-not-zerop-tree (bagint (times-fringe (x), times-fringe (y)),
                                         a)))
→  (((eval$ (t, cadr (x), a) * eval$ (t, caddr (x), a))
      ÷  (eval$ (t, cadr (y), a) * eval$ (t, caddr (y), a)))
     =  0)
```

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

THEOREM: eval\$-quotient-times-tree-bagdiff

$$\begin{aligned} & (\text{subbagp}(x, y) \wedge \text{subbagp}(x, z) \wedge \text{eval}\$(\mathbf{t}, \text{and-not-zerop-tree}(x), a)) \\ \rightarrow & ((\text{eval}\$(\mathbf{t}, \text{times-tree}(\text{bagdiff}(y, x)), a) \\ & \div \text{eval}\$(\mathbf{t}, \text{times-tree}(\text{bagdiff}(z, x)), a)) \\ = & (\text{eval}\$(\mathbf{t}, \text{times-tree}(y), a) \div \text{eval}\$(\mathbf{t}, \text{times-tree}(z), a))) \end{aligned}$$

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

THEOREM: correctness-of-cancel-quotient-times

$$\text{eval}\$(\mathbf{t}, x, a) = \text{eval}\$(\mathbf{t}, \text{cancel-quotient-times}(x), a)$$

;;; exp, log, and gcd

DEFINITION:

$\text{exp}(i, j)$

$$= \text{if } j \simeq 0 \text{ then } 1 \\ \text{else } i * \text{exp}(i, j - 1) \text{ endif}$$

DEFINITION:

$\text{log}(\text{base}, n)$

$$= \text{if } \text{base} < 2 \text{ then } 0 \\ \text{elseif } n \simeq 0 \text{ then } 0 \\ \text{else } 1 + \text{log}(\text{base}, n \div \text{base}) \text{ endif}$$

DEFINITION:

$\text{gcd}(x, y)$

$$= \text{if } x \simeq 0 \text{ then } \text{fix}(y) \\ \text{elseif } y \simeq 0 \text{ then } x \\ \text{elseif } x < y \text{ then } \text{gcd}(x, y - x) \\ \text{else } \text{gcd}(x - y, y) \text{ endif}$$

THEOREM: remainder-exp

$(k \not\simeq 0) \rightarrow ((\text{exp}(n, k) \bmod n) = 0)$

DEFINITION:

$\text{double-number-induction}(i, j)$

$$= \text{if } i \simeq 0 \text{ then } 0 \\ \text{elseif } j \simeq 0 \text{ then } 0 \\ \text{else } \text{double-number-induction}(i - 1, j - 1) \text{ endif}$$

THEOREM: remainder-exp-exp

$(i \leq j) \rightarrow ((\text{exp}(a, j) \bmod \text{exp}(a, i)) = 0)$

THEOREM: quotient-exp
 $(k \neq 0)$
 $\rightarrow ((\exp(n, k) \div n)$
 $\quad = \text{if } n \simeq 0 \text{ then } 0$
 $\quad \text{else } \exp(n, k - 1) \text{ endif})$

THEOREM: exp-zero
 $(k \simeq 0) \rightarrow (\exp(n, k) = 1)$

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
 $\exp(0, k)$
 $= \text{if } k \simeq 0 \text{ then } 1$
 $\text{else } 0 \text{ endif}$

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
 $(\exp(n, k) = 0) = ((n \simeq 0) \wedge (k \neq 0))$

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

THEOREM: exp-difference
 $((c \leq b) \wedge (a \neq 0)) \rightarrow (\exp(a, b - c) = (\exp(a, b) \div \exp(a, c)))$

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

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

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

DEFINITION:
double-log-induction(base, a, b)
= **if** $\text{base} < 2$ **then** 0
 elseif $a \simeq 0$ **then** 0
 elseif $b \simeq 0$ **then** 0
 else double-log-induction($\text{base}, a \div \text{base}, b \div \text{base}$) **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
 $(1 < c)$
 $\rightarrow ((\log(c, n \div (c * m)) = (\log(c, n \div m) - 1))$
 $\quad \wedge (\log(c, n \div (m * c)) = (\log(c, n \div m) - 1)))$

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

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

THEOREM: log-exp

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

THEOREM: commutativity-of-gcd

$$\gcd(b, a) = \gcd(a, b)$$

DEFINITION:

single-number-induction(n)

$$= \text{if } n \simeq 0 \text{ then } 0 \\ \text{else single-number-induction}(n - 1) \text{ endif}$$

THEOREM: gcd-0

$$(\gcd(0, x) = \text{fix}(x)) \wedge (\gcd(x, 0) = \text{fix}(x))$$

THEOREM: gcd-1

$$(\gcd(1, x) = 1) \wedge (\gcd(x, 1) = 1)$$

THEOREM: equal-gcd-0

$$(\gcd(a, b) = 0) = ((a \simeq 0) \wedge (b \simeq 0))$$

THEOREM: lessp-gcd

$$(b \not\simeq 0) \rightarrow (((b < \gcd(a, b)) = \mathbf{f}) \wedge ((b < \gcd(b, a)) = \mathbf{f}))$$

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

$$((b \mathbf{mod} a) = 0) \rightarrow (\gcd(a, b + c) = \gcd(a, c))$$

THEOREM: gcd-plus

$$((b \mathbf{mod} a) = 0) \\ \rightarrow ((\gcd(a, b + c) = \gcd(a, c)) \\ \wedge (\gcd(a, c + b) = \gcd(a, c)) \\ \wedge (\gcd(b + c, a) = \gcd(a, c)) \\ \wedge (\gcd(c + b, a) = \gcd(a, c)))$$

THEOREM: gcd-plus-instance

$$(\gcd(a, a + b) = \gcd(a, b)) \wedge (\gcd(a, b + a) = \gcd(a, b))$$

THEOREM: remainder-gcd

$$((a \mathbf{mod} \gcd(a, b)) = 0) \wedge ((b \mathbf{mod} \gcd(a, b)) = 0)$$

THEOREM: distributivity-of-times-over-gcd-proof

$$\gcd(x * z, y * z) = (z * \gcd(x, y))$$

THEOREM: distributivity-of-times-over-gcd

$$\begin{aligned} & (\gcd(x * z, y * z) = (z * \gcd(x, y))) \\ \wedge & \quad (\gcd(z * x, y * z) = (z * \gcd(x, y))) \\ \wedge & \quad (\gcd(x * z, z * y) = (z * \gcd(x, y))) \\ \wedge & \quad (\gcd(z * x, z * y) = (z * \gcd(x, y))) \end{aligned}$$

THEOREM: gcd-is-the-greatest

$$\begin{aligned} & ((x \neq 0) \wedge (y \neq 0) \wedge ((x \bmod z) = 0) \wedge ((y \bmod z) = 0)) \\ \rightarrow & \quad (z \leq \gcd(x, y)) \end{aligned}$$

THEOREM: common-divisor-divides-gcd

$$(((x \bmod z) = 0) \wedge ((y \bmod z) = 0)) \rightarrow ((\gcd(x, y) \bmod z) = 0)$$

; 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

$$\gcd(x, x) = \text{fix}(x)$$

THEOREM: gcd-idempotence

$$(\gcd(x, \gcd(x, y)) = \gcd(x, y)) \wedge (\gcd(y, \gcd(x, y)) = \gcd(x, y))$$

;;; new stuff

THEOREM: gcd-zero

$$(x \simeq 0) \rightarrow ((\gcd(x, y) = \text{fix}(y)) \wedge (\gcd(y, x) = \text{fix}(y)))$$

THEOREM: remainder-quotient-gcd-0

$$((x \bmod a) = 0) \rightarrow ((x \bmod (x \div a)) = 0)$$

THEOREM: remainder-gcd-0

$$\begin{aligned} & ((x \bmod a) = 0) \\ \rightarrow & \quad (((x \bmod \gcd(a, b)) = 0) \wedge ((x \bmod \gcd(b, a)) = 0)) \end{aligned}$$


```

;I think this is not important
;(prove-lemma gcd-of-reduce-numbers-helper-helper (rewrite)
;  (equal
;    (remainder a (gcd (quotient a (gcd a b)) (quotient b (gcd a b))))
;    0))

; I believe that the following works just as well as this one because
; of commutativity-of-gcd
;(prove-lemma gcd-of-reduced-numbers-helper (rewrite)
;  (implies
;    (not (zerop a))
;    (and
;      (equal (gcd (quotient a (gcd a b))
;                  (quotient b (gcd a b)))
;            1)
;      (equal (gcd (quotient b (gcd a b))
;                  (quotient a (gcd a b)))
;            1)
;      (equal (gcd (quotient b (gcd b a))
;                  (quotient a (gcd b a)))
;            1)
;      (equal (gcd (quotient a (gcd b a))
;                  (quotient b (gcd b a)))
;            1))))))

THEOREM: gcd-quotient-gcd
 $(a \neq 0)$ 
 $\rightarrow ((\gcd(a \div \gcd(a, b), b \div \gcd(a, b)) = 1)$ 
 $\wedge (\gcd(b \div \gcd(b, a), a \div \gcd(b, a)) = 1))$ 

THEOREM: lessp-gcd2
 $((b < \gcd(a, b)) = ((b \simeq 0) \wedge (a \neq 0)))$ 
 $\wedge ((b < \gcd(b, a)) = ((b \simeq 0) \wedge (a \neq 0)))$ 

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

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

;; replaced by gcd-zero
;(lemma gcd-zerop (rewrite)
;  (implies
;    (zerop x)

```

```

;      (and
;      (equal (gcd x z) (fix z))
;      (equal (gcd z x) (fix z)))
;      ((enable-theory naturals)
;      (enable gcd)))

; (disable gcd-zerop) ; subsumed by gcd-remainder

```

THEOREM: gcd-remainder

$$((x \text{ mod } z) = 0) \rightarrow ((\text{gcd}(x, z) = \text{fix}(z)) \wedge (\text{gcd}(z, x) = \text{fix}(z)))$$

EVENT: Disable gcd-zero.

```

; subsumed by gcd-remainder

; no longer needed
;(lemma equal-times-x-x-fact (rewrite)
;  (implies (and (not (equal x 0))
;                (not (equal y 1)))
;            (and
;              (equal (equal (times x y) x) f)
;              (equal (equal (times y x) x) f))))
;  ((enable-theory naturals)
;  (induct (times y x))))

```

THEOREM: equal-times-x-x

$$(((x * y) = x) = (((x \in \mathbf{N}) \wedge (y = 1)) \vee (x = 0)))$$

$$\wedge (((y * x) = x) = (((x \in \mathbf{N}) \wedge (y = 1)) \vee (x = 0)))$$

THEOREM: equal-x-gcd-x-y

$$(y \in \mathbf{N})$$

$$\rightarrow (((x = \text{gcd}(x, y)) = ((x \in \mathbf{N}) \wedge ((y \text{ mod } x) = 0)))$$

$$\wedge ((x = \text{gcd}(y, x)) = ((x \in \mathbf{N}) \wedge ((y \text{ mod } x) = 0))))$$

THEOREM: quotient-difference

$$((x - y) \div z)$$

$$= \text{if } (x \text{ mod } z) < (y \text{ mod } z) \text{ then } ((x \div z) - (y \div z)) - 1$$

$$\text{else } (x \div z) - (y \div z) \text{ endif}$$

THEOREM: equal-as-remainder

$$(((x \text{ mod } y) = 0) \wedge ((y \text{ mod } x) = 0) \wedge (x \in \mathbf{N}) \wedge (y \in \mathbf{N}))$$

$$\rightarrow ((x = y) = \mathbf{t})$$

EVENT: Disable equal-as-remainder.

:: add so as to control rewriting equal better

THEOREM: equal-gcd-gcd-as-remainder

$$\begin{aligned} & (((\text{gcd}(x, y) \bmod \text{gcd}(a, b)) = 0) \wedge ((\text{gcd}(a, b) \bmod \text{gcd}(x, y)) = 0)) \\ & \rightarrow ((\text{gcd}(a, b) = \text{gcd}(x, y)) = \mathbf{t}) \end{aligned}$$

THEOREM: remainder-gcd-0-means

$$\begin{aligned} & (((\text{gcd}(a, b) \bmod c) = 0) \rightarrow ((b \bmod c) = 0)) \\ & \wedge (((\text{gcd}(b, a) \bmod c) = 0) \rightarrow ((b \bmod c) = 0)) \end{aligned}$$

THEOREM: remainder-gcd-arg1

$$\begin{aligned} & ((\text{gcd}(a, b) \bmod c) = 0) \\ & = \text{if } (a \bmod c) = 0 \\ & \quad \text{then if } (b \bmod c) = 0 \text{ then } \mathbf{t} \\ & \quad \quad \text{else } \mathbf{f} \text{ endif} \\ & \quad \text{else } \mathbf{f} \text{ endif} \end{aligned}$$

THEOREM: remainder-diff1

$$\begin{aligned} & (((x \bmod d) = 0) \wedge ((w * x) \not\leq z)) \\ & \rightarrow (((((w * x) - z) \bmod d) = 0) = ((z \bmod d) = 0)) \end{aligned}$$

THEOREM: remainder-diff2

$$\begin{aligned} & (((x \bmod d) = 0) \\ & \wedge ((w * x) \not\leq z) \\ & \wedge (((((w * x) - z) \bmod d) = 0)) \\ & \rightarrow ((z \bmod d) = 0) \end{aligned}$$

THEOREM: remainder-gcd-difference-times-hack

$$((w * x) \not\leq z) \rightarrow ((z \bmod \text{gcd}(x, (w * x) - z)) = 0)$$

THEOREM: gcd-difference1

$$\begin{aligned} & ((y \bmod x) = 0) \\ & \rightarrow (\text{gcd}(x, y - z) \\ & \quad = \text{if } y < z \text{ then } \text{fix}(x) \\ & \quad \quad \text{else } \text{gcd}(x, z) \text{ endif}) \end{aligned}$$

THEOREM: gcd-difference2

$$\begin{aligned} & ((y \bmod x) = 0) \\ & \rightarrow (\text{gcd}(x, z - y) \\ & \quad = \text{if } z < y \text{ then } \text{fix}(x) \\ & \quad \quad \text{else } \text{gcd}(x, z) \text{ endif}) \end{aligned}$$

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.

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, equal-times-x-x.

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, remainder-quotient-gcd-0, remainder-gcd-0, remainder-gcd-arg1, remainder-diff1, remainder-diff2.

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, quotient-difference.

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.

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.

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-idempotence, gcd-x-x, remainder-gcd, gcd-plus, gcd-plus-instance, gcd-1, commutativity-of-gcd, gcd-zero, lessp-gcd2, gcd-times, gcd-remainder, equal-x-gcd-x-y, equal-gcd-gcd-as-remainder, remainder-gcd-0-means, gcd-quotient-gcd, gcd-difference1, gcd-difference2.

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

- addition, 36
- and-not-zero-tree, 14, 16, 17, 27, 28
- associativity-of-gcd, 32
- associativity-of-gcd-zero-case, 32
- associativity-of-plus, 3
- associativity-of-times, 12
- bagdiff, 6–10, 17–19, 27, 28
- bagint, 7–10, 17–19, 27
- bridge-to-subbagp-implies-plus-tree-greatereqp, 6
- cadr-eval\$-list, 7
- cancel-difference-plus, 8, 9
- cancel-equal-plus, 7, 8
- cancel-equal-times, 17, 19
- cancel-equal-times-preserves-inequality, 18
- quality-bridge, 18
- cancel-lessp-plus, 10, 11
- cancel-lessp-times, 16, 17
- cancel-quotient-times, 27, 28
- common-divisor-divides-gcd, 32
- commutativity-of-gcd, 31
- commutativity-of-plus, 3
- commutativity-of-times, 11
- commutativity2-of-gcd, 32
- commutativity2-of-gcd-zero-case, 32
- commutativity2-of-plus, 3
- commutativity2-of-times, 11
- correctness-of-cancel-difference-plus, 9
- correctness-of-cancel-equal-plus, 8
- correctness-of-cancel-equal-times, 19
- correctness-of-cancel-lessp-plus, 11
- correctness-of-cancel-lessp-times, 17
- correctness-of-cancel-quotient-times, 28
- delete, 5, 8–10, 16
- diff-diff-arg1, 4
- diff-diff-arg2, 4
- diff-diff-diff, 4
- diff-sub1-arg2, 4
- difference-add1-arg2, 9
- difference-cancellation, 2
- difference-difference-arg1, 9
- difference-difference-arg2, 9
- difference-elim, 9
- difference-leq-arg1, 9
- difference-lessp-arg1, 5
- difference-plus-cancellation, 3
- difference-plus-cancellation-proof, 3
- difference-plus-plus-cancellation-on, 4
- on-hack, 4
- on-proof, 3
- difference-sub1-arg2, 9
- difference-x-x, 10
- distributivity-of-times-over-gcd, 32
- d-proof, 31
- double-log-induction, 30
- double-number-induction, 28
- double-remainder-induction, 21
- equal-as-remainder, 34
- equal-difference-0, 2
- equal-exp-0, 29
- equal-exp-1, 29
- equal-gcd-0, 31
- equal-gcd-gcd-as-remainder, 35
- equal-log-0, 29
- equal-plus-0, 2
- equal-quotient-0, 23
- equal-remainder-difference-0, 22

- equal-remainder-plus-0, 20
- equal-remainder-plus-0-proof, 20
- equal-remainder-plus-remainder, 20
- equal-remainder-plus-remainder-p
 - roof, 20
- equal-sub1-0, 11
- equal-times-0, 11
- equal-times-1, 11
- equal-times-arg1, 15
- equal-times-bridge, 15
- equal-times-x-x, 34
- equal-x-gcd-x-y, 34
- eval\$-equal, 15
- eval\$-equal-times-tree-bagdiff, 18
- eval\$-if, 15
- eval\$-lessp, 15
- eval\$-lessp-times-tree-bagdiff, 17
- eval\$-or, 14
- eval\$-plus-tree-append, 6
- eval\$-quote, 7
- eval\$-quotient, 15
- eval\$-quotient-times-tree-bagdi
 - ff, 28
- eval\$-times, 14
- eval\$-times-member, 16
- exp, 28–31
- exp-0-arg1, 29
- exp-0-arg2, 29
- exp-1-arg1, 29
- exp-add1, 29
- exp-difference, 29
- exp-exp, 29
- exp-plus, 29
- exp-times, 29
- exp-zero, 29
- exponentiation, 36
- gcd, 28, 31–35
- gcd-0, 31
- gcd-1, 31
- gcd-difference1, 35
- gcd-difference2, 35
- gcd-idempotence, 32
- gcd-is-the-greatest, 32
- gcd-plus, 31
- gcd-plus-instance, 31
- gcd-plus-instance-temp, 31
- gcd-plus-instance-temp-proof, 31
- gcd-plus-proof, 31
- gcd-quotient-gcd, 33
- gcd-remainder, 34
- gcd-times, 33
- gcd-times-proof, 33
- gcd-x-x, 32
- gcd-zero, 32
- gcds, 37
- infer-equality-from-not-lessp, 15
- leq-log-log, 30
- leq-quotient, 26
- lessp-1-times, 13
- lessp-difference-cancellation, 10
- lessp-gcd, 31
- lessp-gcd2, 33
- lessp-plus-fact, 22
- lessp-plus-times-proof, 13
- lessp-plus-times1, 13
- lessp-plus-times2, 13
- lessp-quotient, 27
- lessp-remainder, 19
- lessp-times-arg1, 15
- lessp-times-cancellation-proof, 12
- lessp-times-cancellation1, 13
- lessp-times1, 12
- lessp-times1-proof, 12
- lessp-times2, 12
- lessp-times2-proof, 12
- lessp-times3, 12
- lessp-times3-proof1, 12
- lessp-times3-proof2, 12
- listp-eval\$, 7
- log, 28–31
- log-0, 30
- log-1, 30
- log-exp, 31
- log-quotient, 30
- log-quotient-exp, 30

- log-quotient-times, 30
- log-quotient-times-proof, 30
- log-times, 30
- log-times-exp, 30
- log-times-exp-proof, 30
- log-times-proof, 30
- logs, 37
- member-implies-numberp, 6
- member-implies-plus-tree-greater-than, 5
- multiplication, 36
- naturals, 37
- numberp-eval\$-bridge, 6
- numberp-eval\$-plus, 5
- numberp-eval\$-plus-tree, 5
- numberp-eval\$-times, 14
- numberp-eval\$-times-tree, 15
- or-zero-tree, 14, 16, 18
- or-zero-tree-is-not-zero-tree, 16
- plus-add1-arg1, 3
- plus-add1-arg2, 3
- plus-cancellation, 2
- plus-difference-arg1, 3
- plus-difference-arg2, 3
- plus-fringe, 5–10
- plus-remainder-times-quotient, 19
- plus-tree, 5–10
- plus-tree-bagdiff, 6
- plus-tree-delete, 5
- plus-tree-plus-fringe, 6
- plus-zero-arg2, 3
- quotient-1-arg1, 26
- quotient-1-arg1-casesplit, 26
- quotient-1-arg2, 26
- quotient-add1, 23
- quotient-difference, 34
- quotient-difference1, 25
- quotient-difference2, 25
- quotient-difference3, 25
- quotient-exp, 29
- quotient-lessp-arg1, 25
- quotient-noop, 23
- quotient-of-non-number, 23
- quotient-plus, 24
- quotient-plus-fact, 26
- quotient-plus-proof, 24
- quotient-plus-times-times, 26
- quotient-plus-times-times-instance, 26
- quotient-plus-times-times-proof, 26
- quotient-quotient, 26
- quotient-remainder, 25
- quotient-remainder-instance, 26
- quotient-remainder-times, 25
- quotient-sub1, 23
- quotient-times, 24
- quotient-times-instance, 24
- quotient-times-instance-temp, 24
- quotient-times-instance-temp-proof, 24
- quotient-times-proof, 24
- quotient-times-times, 25
- quotient-times-times-proof, 25
- quotient-x-x, 27
- quotient-zero, 23
- quotients, 36
- remainder-1-arg1, 23
- remainder-1-arg2, 23
- remainder-add1, 20
- remainder-diff1, 35
- remainder-diff2, 35
- remainder-difference1, 21
- remainder-difference2, 21
- remainder-difference3, 22
- remainder-equals-its-first-argument, 25
- remainder-exp, 28
- remainder-exp-exp, 28
- remainder-gcd, 31
- remainder-gcd-0, 32
- remainder-gcd-0-means, 35
- remainder-gcd-arg1, 35
- remainder-gcd-difference-times-

- hack, 35
- remainder-noop, 19
- remainder-of-non-number, 19
- remainder-plus, 20
- remainder-plus-fact, 22
- remainder-plus-proof, 20
- remainder-plus-times-times, 22
- remainder-plus-times-times-instance, 22
- remainder-plus-times-times-proof, 22
- remainder-quotient-elim, 19
- remainder-quotient-gcd-0, 32
- remainder-remainder, 23
- remainder-times-times, 21
- remainder-times-times-proof, 21
- remainder-times1, 20
- remainder-times1-instance, 21
- remainder-times1-instance-proof, 21
- remainder-times1-proof, 20
- remainder-times2, 21
- remainder-times2-instance, 21
- remainder-times2-proof, 21
- remainder-x-x, 23
- remainder-zero, 19
- remainders, 36
- single-number-induction, 31
- subbagp, 6, 16–18, 28
- subbagp-implies-plus-tree-greater-or-equal, 6
- times-1-arg1, 12
- times-add1, 11
- times-distributes-over-difference, 12
- times-distributes-over-difference-proof, 12
- times-distributes-over-plus, 11
- times-distributes-over-plus-proof, 11
- times-fringe, 14, 16–19, 27
- times-quotient, 12
- times-quotient-proof, 12
- times-tree, 14–19, 27, 28
- times-tree-append, 16
- times-tree-of-times-fringe, 16
- times-zero, 11
- transitivity-of-divides, 23
- zerop-makes-equal-true-bridge, 18
- zerop-makes-lessp-false-bridge, 17
- zerop-makes-quotient-zero-bridge, 27
- zerop-makes-times-tree-zero, 16
- zerop-makes-times-tree-zero2, 16