Miscellaneous HOL-Complex Examples

November 22, 2007

Contents

1 Binary arithmetic examples

theory *BinEx*
imports *Complex-Main*
begin

Examples of performing binary arithmetic by simplification. This time we use the reals, though the representation is just of integers.

1.1 Real Arithmetic

1.1.1 Addition

lemma $(1359::real) + -2468 = -1109$
by *simp*

lemma $(93746::real) + -46375 = 47371$
by *simp*

1.1.2 Negation

lemma $- (65745::real) = -65745$
by *simp*

lemma $- (-54321::real) = 54321$
by *simp*

1.1.3 Multiplication

lemma $(-84::real) * 51 = -4284$
by *simp*

lemma $(255::real) * 255 = 65025$
by *simp*

lemma $(1359::real) * -2468 = -3354012$
by *simp*

1.1.4 Inequalities

lemma $(89::real) * 10 \neq 889$
by *simp*

lemma $(13::real) < 18 - 4$
by *simp*

lemma $(-345::real) < -242 + -100$
by *simp*

lemma $(13557456::real) < 18678654$
by *simp*

lemma $(999999::real) \leq (1000001 + 1) - 2$
by *simp*

lemma $(1234567::real) \leq 1234567$
by *simp*

1.1.5 Powers

lemma $2 \hat{} 15 = (32768::real)$
by *simp*

lemma $-3 \hat{} 7 = (-2187::real)$
by *simp*

lemma $13 \hat{} 7 = (62748517::real)$
by *simp*

lemma $3 \hat{} 15 = (14348907::real)$
by *simp*

lemma $-5 \hat{} 11 = (-48828125::real)$
by *simp*

1.1.6 Tests

lemma $(x + y = x) = (y = (0::real))$
by *arith*

lemma $(x + y = y) = (x = (0::real))$
by *arith*

lemma $(x + y = (0::real)) = (x = -y)$
by *arith*

lemma $(x + y = (0::real)) = (y = -x)$
by *arith*

lemma $((x + y) < (x + z)) = (y < (z::real))$
by *arith*

lemma $((x + z) < (y + z)) = (x < (y::real))$
by *arith*

lemma $(\neg x < y) = (y \leq (x::real))$
by *arith*

lemma $\neg (x < y \wedge y < (x::real))$
by *arith*

lemma $(x::real) < y ==> \neg y < x$
by *arith*

lemma $((x::real) \neq y) = (x < y \vee y < x)$
by *arith*

lemma $(\neg x \leq y) = (y < (x::real))$
by *arith*

lemma $x \leq y \vee y \leq (x::real)$
by *arith*

lemma $x \leq y \vee y < (x::real)$
by *arith*

lemma $x < y \vee y \leq (x::real)$
by *arith*

lemma $x \le (x{::}real)$
by *arith*

lemma $((x{::}real) \le y) = (x < y \lor x = y)$
by *arith*

lemma $((x{::}real) \le y \land y \le x) = (x = y)$
by *arith*

lemma $\neg(x < y \land y \le (x{::}real))$
by *arith*

lemma $\neg(x \le y \land y < (x{::}real))$
by *arith*

lemma $(-x < (0{::}real)) = (0 < x)$
by *arith*

lemma $((0{::}real) < -x) = (x < 0)$
by *arith*

lemma $(-x \le (0{::}real)) = (0 \le x)$
by *arith*

lemma $((0{::}real) \le -x) = (x \le 0)$
by *arith*

lemma $(x{::}real) = y \lor x < y \lor y < x$
by *arith*

lemma $(x{::}real) = 0 \lor 0 < x \lor 0 < -x$
by *arith*

lemma $(0{::}real) \le x \lor 0 \le -x$
by *arith*

lemma $((x{::}real) + y \le x + z) = (y \le z)$
by *arith*

lemma $((x{::}real) + z \le y + z) = (x \le y)$
by *arith*

lemma $(w{::}real) < x \land y < z ==> w + y < x + z$
by *arith*

lemma $(w{::}real) \le x \land y \le z ==> w + y \le x + z$
by *arith*

lemma $(0::real) \leq x \wedge 0 \leq y ==> 0 \leq x + y$
by *arith*

lemma $(0::real) < x \wedge 0 < y ==> 0 < x + y$
by *arith*

lemma $(-x < y) = (0 < x + (y::real))$
by *arith*

lemma $(x < -y) = (x + y < (0::real))$
by *arith*

lemma $(y < x + -z) = (y + z < (x::real))$
by *arith*

lemma $(x + -y < z) = (x < z + (y::real))$
by *arith*

lemma $x \leq y ==> x < y + (1::real)$
by *arith*

lemma $(x - y) + y = (x::real)$
by *arith*

lemma $y + (x - y) = (x::real)$
by *arith*

lemma $x - x = (0::real)$
by *arith*

lemma $(x - y = 0) = (x = (y::real))$
by *arith*

lemma $((0::real) \leq x + x) = (0 \leq x)$
by *arith*

lemma $(-x \leq x) = ((0::real) \leq x)$
by *arith*

lemma $(x \leq -x) = (x \leq (0::real))$
by *arith*

lemma $(-x = (0::real)) = (x = 0)$
by *arith*

lemma $-(x - y) = y - (x::real)$
by *arith*

lemma $((0::real) < x - y) = (y < x)$

by *arith*

lemma $((0::real) \leq x - y) = (y \leq x)$
by *arith*

lemma $(x + y) - x = (y::real)$
by *arith*

lemma $(-x = y) = (x = (-y::real))$
by *arith*

lemma $x < (y::real) ==> \neg(x = y)$
by *arith*

lemma $(x \leq x + y) = ((0::real) \leq y)$
by *arith*

lemma $(y \leq x + y) = ((0::real) \leq x)$
by *arith*

lemma $(x < x + y) = ((0::real) < y)$
by *arith*

lemma $(y < x + y) = ((0::real) < x)$
by *arith*

lemma $(x - y) - x = (-y::real)$
by *arith*

lemma $(x + y < z) = (x < z - (y::real))$
by *arith*

lemma $(x - y < z) = (x < z + (y::real))$
by *arith*

lemma $(x < y - z) = (x + z < (y::real))$
by *arith*

lemma $(x \leq y - z) = (x + z \leq (y::real))$
by *arith*

lemma $(x - y \leq z) = (x \leq z + (y::real))$
by *arith*

lemma $(-x < -y) = (y < (x::real))$
by *arith*

lemma $(-x \leq -y) = (y \leq (x::real))$
by *arith*

lemma $(a + b) - (c + d) = (a - c) + (b - (d::real))$
by *arith*

lemma $(0::real) - x = -x$
by *arith*

lemma $x - (0::real) = x$
by *arith*

lemma $w \leq x \wedge y < z ==> w + y < x + (z::real)$
by *arith*

lemma $w < x \wedge y \leq z ==> w + y < x + (z::real)$
by *arith*

lemma $(0::real) \leq x \wedge 0 < y ==> 0 < x + (y::real)$
by *arith*

lemma $(0::real) < x \wedge 0 \leq y ==> 0 < x + y$
by *arith*

lemma $-x - y = -(x + (y::real))$
by *arith*

lemma $x - (-y) = x + (y::real)$
by *arith*

lemma $-x - -y = y - (x::real)$
by *arith*

lemma $(a - b) + (b - c) = a - (c::real)$
by *arith*

lemma $(x = y - z) = (x + z = (y::real))$
by *arith*

lemma $(x - y = z) = (x = z + (y::real))$
by *arith*

lemma $x - (x - y) = (y::real)$
by *arith*

lemma $x - (x + y) = -(y::real)$
by *arith*

lemma $x = y ==> x \leq (y::real)$
by *arith*

lemma $(0::real) < x ==> \neg(x = 0)$
by *arith*

lemma $(x + y) * (x - y) = (x * x) - (y * y)$
oops

lemma $(-x = -y) = (x = (y::real))$
by *arith*

lemma $(-x < -y) = (y < (x::real))$
by *arith*

lemma $!!a::real.\ a \leq b ==> c \leq d ==> x + y < z ==> a + c \leq b + d$
by (*tactic fast-arith-tac* @{*context*} *1*)

lemma $!!a::real.\ a < b ==> c < d ==> a - d \leq b + (-c)$
by (*tactic fast-arith-tac* @{*context*} *1*)

lemma $!!a::real.\ a \leq b ==> b + b \leq c ==> a + a \leq c$
by (*tactic fast-arith-tac* @{*context*} *1*)

lemma $!!a::real.\ a + b \leq i + j ==> a \leq b ==> i \leq j ==> a + a \leq j + j$
by (*tactic fast-arith-tac* @{*context*} *1*)

lemma $!!a::real.\ a + b < i + j ==> a < b ==> i < j ==> a + a < j + j$
by (*tactic fast-arith-tac* @{*context*} *1*)

lemma $!!a::real.\ a + b + c \leq i + j + k \wedge a \leq b \wedge b \leq c \wedge i \leq j \wedge j \leq k -->$
$a + a + a \leq k + k + k$
by *arith*

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l ==> a \leq b ==> b \leq c$
$==> c \leq d ==> i \leq j ==> j \leq k ==> k \leq l ==> a \leq l$
by (*tactic fast-arith-tac* @{*context*} *1*)

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l ==> a \leq b ==> b \leq c$
$==> c \leq d ==> i \leq j ==> j \leq k ==> k \leq l ==> a + a + a + a \leq l +$
$l + l + l$
by (*tactic fast-arith-tac* @{*context*} *1*)

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l ==> a \leq b ==> b \leq c$
$==> c \leq d ==> i \leq j ==> j \leq k ==> k \leq l ==> a + a + a + a + a \leq$
$l + l + l + l + i$
by (*tactic fast-arith-tac* @{*context*} *1*)

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l ==> a \leq b ==> b \leq c$
$==> c \leq d ==> i \leq j ==> j \leq k ==> k \leq l ==> a + a + a + a + a +$
$a \leq l + l + l + l + i + l$
by (*tactic fast-arith-tac* @{*context*} *1*)

1.2 Complex Arithmetic

lemma $(1359 + 93746*ii) - (2468 + 46375*ii) = -1109 + 47371*ii$
by *simp*

lemma $- (65745 + -47371*ii) = -65745 + 47371*ii$
by *simp*

Multiplication requires distributive laws. Perhaps versions instantiated to literal constants should be added to the simpset.

lemma $(1 + ii) * (1 - ii) = 2$
by (*simp add*: *ring-distribs*)

lemma $(1 + 2*ii) * (1 + 3*ii) = -5 + 5*ii$
by (*simp add*: *ring-distribs*)

lemma $(-84 + 255*ii) + (51 * 255*ii) = -84 + 13260 * ii$
by (*simp add*: *ring-distribs*)

No inequalities or linear arithmetic: the complex numbers are unordered!

No powers (not supported yet)

end

2 Square roots of primes are irrational

theory *Sqrt*
imports *Primes Complex-Main*
begin

2.1 Preliminaries

The set of rational numbers, including the key representation theorem.

definition
  rationals (ℚ) **where**
  $\mathbb{Q} = \{x.\ \exists\, m\ n.\ n \neq 0 \wedge |x| = real\ (m::nat)\ /\ real\ (n::nat)\}$

theorem *rationals-rep* [*elim?*]:
  assumes $x \in \mathbb{Q}$
  obtains *m n* **where** $n \neq 0$ **and** $|x| = real\ m\ /\ real\ n$ **and** $gcd\ (m, n) = 1$
proof −
  from ‹$x \in \mathbb{Q}$› **obtain** $m\ n :: nat$ **where**
    n: $n \neq 0$ **and** *x-rat*: $|x| = real\ m\ /\ real\ n$
   unfolding *rationals-def* **by** *blast*
  let $?gcd = gcd\ (m, n)$
  from *n* **have** *gcd*: $?gcd \neq 0$ **by** (*simp add*: *gcd-zero*)
  let $?k = m\ div\ ?gcd$

```
  let ?l = n div ?gcd
  let ?gcd' = gcd (?k, ?l)
  have ?gcd dvd m .. then have gcd-k: ?gcd * ?k = m
    by (rule dvd-mult-div-cancel)
  have ?gcd dvd n .. then have gcd-l: ?gcd * ?l = n
    by (rule dvd-mult-div-cancel)

  from n and gcd-l have ?l ≠ 0
    by (auto iff del: neq0-conv)
  moreover
  have |x| = real ?k / real ?l
  proof −
    from gcd have real ?k / real ?l =
        real (?gcd * ?k) / real (?gcd * ?l) by simp
    also from gcd-k and gcd-l have ... = real m / real n by simp
    also from x-rat have ... = |x| ..
    finally show ?thesis ..
  qed
  moreover
  have ?gcd' = 1
  proof −
    have ?gcd * ?gcd' = gcd (?gcd * ?k, ?gcd * ?l)
      by (rule gcd-mult-distrib2)
    with gcd-k gcd-l have ?gcd * ?gcd' = ?gcd by simp
    with gcd show ?thesis by simp
  qed
  ultimately show ?thesis ..
qed
```

2.2 Main theorem

The square root of any prime number (including *2*) is irrational.

```
theorem sqrt-prime-irrational:
  assumes prime p
  shows sqrt (real p) ∉ ℚ
proof
  from ‹prime p› have p: 1 < p by (simp add: prime-def)
  assume sqrt (real p) ∈ ℚ
  then obtain m n where
      n: n ≠ 0 and sqrt-rat: |sqrt (real p)| = real m / real n
    and gcd: gcd (m, n) = 1 ..
  have eq: m^2 = p * n^2
  proof −
    from n and sqrt-rat have real m = |sqrt (real p)| * real n by simp
    then have real (m^2) = (sqrt (real p))^2 * real (n^2)
      by (auto simp add: power2-eq-square)
    also have (sqrt (real p))^2 = real p by simp
    also have ... * real (n^2) = real (p * n^2) by simp
    finally show ?thesis ..
```

qed
have $p\ \mathit{dvd}\ m \wedge p\ \mathit{dvd}\ n$
proof
from *eq* **have** $p\ \mathit{dvd}\ m^2$ **..**
with ‹*prime p*› **show** $p\ \mathit{dvd}\ m$ **by** (*rule prime-dvd-power-two*)
then obtain k **where** $m = p * k$ **..**
with *eq* **have** $p * n^2 = p^2 * k^2$ **by** (*auto simp add*: *power2-eq-square mult-ac*)
with p **have** $n^2 = p * k^2$ **by** (*simp add*: *power2-eq-square*)
then have $p\ \mathit{dvd}\ n^2$ **..**
with ‹*prime p*› **show** $p\ \mathit{dvd}\ n$ **by** (*rule prime-dvd-power-two*)
qed
then have $p\ \mathit{dvd}\ \mathit{gcd}\ (m,\ n)$ **..**
with *gcd* **have** $p\ \mathit{dvd}\ 1$ **by** *simp*
then have $p \leq 1$ **by** (*simp add*: *dvd-imp-le*)
with p **show** *False* **by** *simp*
qed

corollary $\mathit{sqrt}\ (\mathit{real}\ (2{::}\mathit{nat})) \notin \mathbb{Q}$
by (*rule sqrt-prime-irrational*) (*rule two-is-prime*)

2.3 Variations

Here is an alternative version of the main proof, using mostly linear forward-reasoning. While this results in less top-down structure, it is probably closer to proofs seen in mathematics.

theorem
assumes *prime p*
shows $\mathit{sqrt}\ (\mathit{real}\ p) \notin \mathbb{Q}$
proof
from ‹*prime p*› **have** p: $1 < p$ **by** (*simp add*: *prime-def*)
assume $\mathit{sqrt}\ (\mathit{real}\ p) \in \mathbb{Q}$
then obtain $m\ n$ **where**
n: $n \neq 0$ **and** *sqrt-rat*: $|\mathit{sqrt}\ (\mathit{real}\ p)| = \mathit{real}\ m\ /\ \mathit{real}\ n$
and *gcd*: $\mathit{gcd}\ (m,\ n) = 1$ **..**
from n **and** *sqrt-rat* **have** $\mathit{real}\ m = |\mathit{sqrt}\ (\mathit{real}\ p)| * \mathit{real}\ n$ **by** *simp*
then have $\mathit{real}\ (m^2) = (\mathit{sqrt}\ (\mathit{real}\ p))^2 * \mathit{real}\ (n^2)$
by (*auto simp add*: *power2-eq-square*)
also have $(\mathit{sqrt}\ (\mathit{real}\ p))^2 = \mathit{real}\ p$ **by** *simp*
also have $\ldots * \mathit{real}\ (n^2) = \mathit{real}\ (p * n^2)$ **by** *simp*
finally have *eq*: $m^2 = p * n^2$ **..**
then have $p\ \mathit{dvd}\ m^2$ **..**
with ‹*prime p*› **have** *dvd-m*: $p\ \mathit{dvd}\ m$ **by** (*rule prime-dvd-power-two*)
then obtain k **where** $m = p * k$ **..**
with *eq* **have** $p * n^2 = p^2 * k^2$ **by** (*auto simp add*: *power2-eq-square mult-ac*)
with p **have** $n^2 = p * k^2$ **by** (*simp add*: *power2-eq-square*)
then have $p\ \mathit{dvd}\ n^2$ **..**
with ‹*prime p*› **have** $p\ \mathit{dvd}\ n$ **by** (*rule prime-dvd-power-two*)
with *dvd-m* **have** $p\ \mathit{dvd}\ \mathit{gcd}\ (m,\ n)$ **by** (*rule gcd-greatest*)
with *gcd* **have** $p\ \mathit{dvd}\ 1$ **by** *simp*

then have $p \le 1$ **by** (*simp add*: *dvd-imp-le*)
with p **show** *False* **by** *simp*
qed

end

3 Square roots of primes are irrational (script version)

theory *Sqrt-Script*
imports *Primes Complex-Main*
begin

Contrast this linear Isabelle/Isar script with Markus Wenzel's more mathematical version.

3.1 Preliminaries

lemma *prime-nonzero*: $prime\ p \Longrightarrow p \neq 0$
by (*force simp add*: *prime-def*)

lemma *prime-dvd-other-side*:
$n * n = p * (k * k) \Longrightarrow prime\ p \Longrightarrow p\ dvd\ n$
apply (*subgoal-tac* $p\ dvd\ n * n$, *blast dest*: *prime-dvd-mult*)
apply (*rule-tac* $j = k * k$ **in** *dvd-mult-left*, *simp*)
done

lemma *reduction*: $prime\ p \Longrightarrow$
$0 < k \Longrightarrow k * k = p * (j * j) \Longrightarrow k < p * j \wedge 0 < j$
apply (*rule ccontr*)
apply (*simp add*: *linorder-not-less*)
apply (*erule disjE*)
apply (*frule mult-le-mono*, *assumption*)
apply *auto*
apply (*force simp add*: *prime-def*)
done

lemma *rearrange*: $(j::nat) * (p * j) = k * k \Longrightarrow k * k = p * (j * j)$
by (*simp add*: *mult-ac*)

lemma *prime-not-square*:
$prime\ p \Longrightarrow (\bigwedge k.\ 0 < k \Longrightarrow m * m \neq p * (k * k))$
apply (*induct m rule*: *nat-less-induct*)
apply *clarify*
apply (*frule prime-dvd-other-side*, *assumption*)
apply (*erule dvdE*)
apply (*simp add*: *nat-mult-eq-cancel-disj prime-nonzero*)

apply (*blast dest*: *rearrange reduction*)
done

3.2 The set of rational numbers

definition
rationals :: *real set* $\quad$ ($\mathbb{Q}$) **where**
$\mathbb{Q} = \{x.\ \exists\, m\ n.\ n \neq 0 \wedge |x| = \mathit{real}\ (m{::}\mathit{nat})\ /\ \mathit{real}\ (n{::}\mathit{nat})\}$

3.3 Main theorem

The square root of any prime number (including *2*) is irrational.

theorem *prime-sqrt-irrational*:
$\mathit{prime}\ p \Longrightarrow x * x = \mathit{real}\ p \Longrightarrow 0 \leq x \Longrightarrow x \notin \mathbb{Q}$
apply (*simp add*: *rationals-def real-abs-def*)
apply *clarify*
apply (*erule-tac P* = *real m* / *real n* ∗ *?x* = *?y* **in** *rev-mp*)
apply (*simp del*: *real-of-nat-mult*
add: *divide-eq-eq prime-not-square real-of-nat-mult* [*symmetric*])
done

lemmas *two-sqrt-irrational* =
prime-sqrt-irrational [*OF two-is-prime*]

end

4 The Nonstandard Primes as an Extension of the Prime Numbers

theory *NSPrimes*
imports *~~/src/HOL/NumberTheory/Factorization Complex-Main*
begin

These can be used to derive an alternative proof of the infinitude of primes by considering a property of nonstandard sets.

definition
hdvd :: [*hypnat*, *hypnat*] => *bool* $\quad$ (**infixl** *hdvd 50*) **where**
[*transfer-unfold*]: (*M*::*hypnat*) *hdvd N* = (∗*p2*∗ (*op dvd*)) *M N*

definition
starprime :: *hypnat set* **where**
[*transfer-unfold*]: *starprime* = (∗*s*∗ {*p*. *prime p*})

definition
choicefun :: *'a set* => *'a* **where**
choicefun E = (@*x*. ∃ *X* ∈ *Pow*(*E*) −{{}}. *x* : *X*)

consts *injf-max* :: *nat* => (*'a*::{*order*} *set*) => *'a*
primrec
  injf-max-zero: *injf-max 0 E* = *choicefun E*
  injf-max-Suc: *injf-max* (*Suc n*) *E* = *choicefun*({*e*. *e*:*E* & *injf-max n E* < *e*})

lemma *dvd-by-all*: ∀ *M*. ∃ *N*. *0* < *N* & (∀ *m*. *0* < *m* & (*m*::*nat*) <= *M* --> *m dvd N*)
apply (*rule allI*)
apply (*induct-tac M*, *auto*)
apply (*rule-tac x* = *N* * (*Suc n*) **in** *exI*)
apply (*safe*, *force*)
apply (*drule le-imp-less-or-eq*, *erule disjE*)
apply (*force intro*!: *dvd-mult2*)
apply (*force intro*!: *dvd-mult*)
done

lemmas *dvd-by-all2* = *dvd-by-all* [*THEN spec*, *standard*]

lemma *hypnat-of-nat-le-zero-iff*: (*hypnat-of-nat n* <= *0*) = (*n* = *0*)
by (*transfer*, *simp*)
declare *hypnat-of-nat-le-zero-iff* [*simp*]

lemma *hdvd-by-all*: ∀ *M*. ∃ *N*. *0* < *N* & (∀ *m*. *0* < *m* & (*m*::*hypnat*) <= *M* --> *m hdvd N*)
by (*transfer*, *rule dvd-by-all*)

lemmas *hdvd-by-all2* = *hdvd-by-all* [*THEN spec*, *standard*]

lemma *hypnat-dvd-all-hypnat-of-nat*:
    ∃ (*N*::*hypnat*). *0* < *N* & (∀ *n* ∈ −{*0*::*nat*}. *hypnat-of-nat*(*n*) *hdvd N*)
apply (*cut-tac hdvd-by-all*)
apply (*drule-tac x* = *whn* **in** *spec*, *auto*)
apply (*rule exI*, *auto*)
apply (*drule-tac x* = *hypnat-of-nat n* **in** *spec*)
apply (*auto simp add*: *linorder-not-less star-of-eq-0*)
done

The nonstandard extension of the set prime numbers consists of precisely those hypernaturals exceeding 1 that have no nontrivial factors

lemma *starprime*:
  starprime = {*p*. *1* < *p* & (∀ *m*. *m hdvd p* --> *m* = *1* | *m* = *p*)}
by (*transfer*, *auto simp add*: *prime-def*)

lemma *prime-two*: *prime 2*

apply (*unfold prime-def*, *auto*)
apply (*frule dvd-imp-le*)
apply (*auto dest*: *dvd-0-left*)
apply (*case-tac m*, *simp*, *arith*)
done
declare *prime-two* [*simp*]

lemma *prime-factor-exists* [*rule-format*]: *Suc 0* < *n* −−> (∃ *k*. *prime k* & *k dvd n*)
apply (*rule-tac n* = *n* **in** *nat-less-induct*, *auto*)
apply (*case-tac prime n*)
apply (*rule-tac x* = *n* **in** *exI*, *auto*)
apply (*drule conjI* [*THEN not-prime-ex-mk*], *auto*)
apply (*drule-tac x* = *m* **in** *spec*, *auto*)
apply (*rule-tac x* = *ka* **in** *exI*)
apply (*auto intro*: *dvd-mult2*)
done

lemma *hyperprime-factor-exists* [*rule-format*]:
!!*n*. *1* < *n* ==> (∃ *k* ∈ *starprime*. *k hdvd n*)
by (*transfer*, *simp add*: *prime-factor-exists*)

lemma *NatStar-hypnat-of-nat*: *finite A* ==> ∗*s*∗ *A* = *hypnat-of-nat* ' *A*
by (*rule starset-finite*)

4.1 Another characterization of infinite set of natural numbers

lemma *finite-nat-set-bounded*: *finite N* ==> ∃ *n*. (∀ *i* ∈ *N*. *i*<(*n*::*nat*))
apply (*erule-tac F* = *N* **in** *finite-induct*, *auto*)
apply (*rule-tac x* = *Suc n* + *x* **in** *exI*, *auto*)
done

lemma *finite-nat-set-bounded-iff*: *finite N* = (∃ *n*. (∀ *i* ∈ *N*. *i*<(*n*::*nat*)))
by (*blast intro*: *finite-nat-set-bounded bounded-nat-set-is-finite*)

lemma *not-finite-nat-set-iff*: (~ *finite N*) = (∀ *n*. ∃ *i* ∈ *N*. *n* <= (*i*::*nat*))
by (*auto simp add*: *finite-nat-set-bounded-iff le-def*)

lemma *bounded-nat-set-is-finite2*: (∀ *i* ∈ *N*. *i*<=(*n*::*nat*)) ==> *finite N*
apply (*rule finite-subset*)
apply (*rule-tac* [*2*] *finite-atMost*, *auto*)
done

lemma *finite-nat-set-bounded2*: *finite N* ==> ∃ *n*. (∀ *i* ∈ *N*. *i*<=(*n*::*nat*))
apply (*erule-tac F* = *N* **in** *finite-induct*, *auto*)

apply (*rule-tac* $x = n + x$ **in** *exI*, *auto*)
done

lemma *finite-nat-set-bounded-iff2*: *finite* $N = (\exists\, n.\ (\forall\, i \in N.\ i{<=}(n{::}nat)))$
by (*blast intro*: *finite-nat-set-bounded2 bounded-nat-set-is-finite2*)

lemma *not-finite-nat-set-iff2*: ($\sim$ *finite* N) $= (\forall\, n.\ \exists\, i \in N.\ n < (i{::}nat))$
by (*auto simp add*: *finite-nat-set-bounded-iff2 le-def*)

4.2 An injective function cannot define an embedded natural number

lemma *lemma-infinite-set-singleton*: $\forall\, m\ n.\ m \neq n \ {-}{-}{>}\ f\, n \neq f\, m$
$==>\ \{n.\ f\, n = N\} = \{\} \mid\ (\exists\, m.\ \{n.\ f\, n = N\} = \{m\})$
apply *auto*
apply (*drule-tac* $x = x$ **in** *spec*, *auto*)
apply (*subgoal-tac* $\forall\, n.\ (f\, n = f\, x) = (x = n)$)
apply *auto*
done

lemma *inj-fun-not-hypnat-in-SHNat*:
assumes *inj-f*: *inj* (f::*nat=>nat*)
shows *starfun f whn* $\notin$ *Nats*
proof
from *inj-f* **have** *inj-f'*: *inj* (*starfun f*)
by (*transfer inj-on-def Ball-def UNIV-def*)
assume *starfun f whn* $\in$ *Nats*
then obtain N **where** N: *starfun f whn* = *hypnat-of-nat* N
by (*auto simp add*: *Nats-def*)
hence $\exists\, n.$ *starfun f n* = *hypnat-of-nat* N **..**
hence $\exists\, n.\ f\, n = N$ **by** *transfer*
then obtain n **where** n: $f\, n = N$ **..**
hence *starfun f* (*hypnat-of-nat n*) = *hypnat-of-nat* N
by *transfer*
with N **have** *starfun f whn* = *starfun f* (*hypnat-of-nat n*)
by *simp*
with *inj-f'* **have** *whn* = *hypnat-of-nat n*
by (*rule injD*)
thus *False*
by (*simp add*: *whn-neq-hypnat-of-nat*)
qed

lemma *range-subset-mem-starsetNat*:
range f <= A ==> *starfun f whn* $\in$ *s* A
apply (*rule-tac x=whn* **in** *spec*)
apply (*transfer*, *auto*)
done

lemma *lemmaPow3*: $E \neq \{\} ==> \exists x.\ \exists X \in (Pow\ E - \{\{\}\}).\ x: X$
by *auto*

lemma *choicefun-mem-set*: $E \neq \{\} ==> choicefun\ E \in E$
apply (*unfold choicefun-def*)
apply (*rule lemmaPow3* [*THEN someI2-ex*], *auto*)
done
declare *choicefun-mem-set* [*simp*]

lemma *injf-max-mem-set*: $[|\ E \neq \{\};\ \forall x.\ \exists y \in E.\ x < y\ |] ==> injf\text{-}max\ n\ E \in E$
apply (*induct-tac n*, *force*)
apply (*simp* (*no-asm*) *add*: *choicefun-def*)
apply (*rule lemmaPow3* [*THEN someI2-ex*], *auto*)
done

lemma *injf-max-order-preserving*: $\forall x.\ \exists y \in E.\ x < y ==> injf\text{-}max\ n\ E < injf\text{-}max\ (Suc\ n)\ E$
apply (*simp* (*no-asm*) *add*: *choicefun-def*)
apply (*rule lemmaPow3* [*THEN someI2-ex*], *auto*)
done

lemma *injf-max-order-preserving2*: $\forall x.\ \exists y \in E.\ x < y$
$==> \forall n\ m.\ m < n --> injf\text{-}max\ m\ E < injf\text{-}max\ n\ E$
apply (*rule allI*)
apply (*induct-tac n*, *auto*)
apply (*simp* (*no-asm*) *add*: *choicefun-def*)
apply (*rule lemmaPow3* [*THEN someI2-ex*])
apply (*auto simp add*: *less-Suc-eq*)
apply (*drule-tac* $x = m$ **in** *spec*)
apply (*drule subsetD*, *auto*)
apply (*drule-tac* $x = injf\text{-}max\ m\ E$ **in** *order-less-trans*, *auto*)
done

lemma *inj-injf-max*: $\forall x.\ \exists y \in E.\ x < y ==> inj\ (\%n.\ injf\text{-}max\ n\ E)$
apply (*rule inj-onI*)
apply (*rule ccontr*, *auto*)
apply (*drule injf-max-order-preserving2*)
apply (*metis linorder-antisym-conv3 order-less-le*)
done

lemma *infinite-set-has-order-preserving-inj*:
  $[| \ (E::('a::\{order\}\ set)) \neq \{\}; \forall x. \ \exists y \in E. \ x < y \ |]$
  $==> \exists f. \ range\ f <= E \ \& \ inj\ (f::nat => 'a) \ \& \ (\forall m. \ f\ m < f(Suc\ m))$
apply (*rule-tac x* = %*n*. *injf-max n E* **in** *exI*, *safe*)
apply (*rule injf-max-mem-set*)
apply (*rule-tac* [*3*] *inj-injf-max*)
apply (*rule-tac* [*4*] *injf-max-order-preserving*, *auto*)
done

Only need the existence of an injective function from N to A for proof

lemma *hypnat-infinite-has-nonstandard*:
  $\sim$ *finite A* ==> *hypnat-of-nat* ' *A* < (*s* *A*)
apply *auto*
apply (*subgoal-tac* $A \neq \{\}$)
prefer *2* **apply** *force*
apply (*drule infinite-set-has-order-preserving-inj*)
apply (*erule not-finite-nat-set-iff2* [*THEN iffD1*], *auto*)
apply (*drule inj-fun-not-hypnat-in-SHNat*)
apply (*drule range-subset-mem-starsetNat*)
apply (*auto simp add*: *SHNat-eq*)
done

lemma *starsetNat-eq-hypnat-of-nat-image-finite*: *s* *A* = *hypnat-of-nat* ' *A* ==> *finite A*
apply (*rule ccontr*)
apply (*auto dest*: *hypnat-infinite-has-nonstandard*)
done

lemma *finite-starsetNat-iff*: (*s* *A* = *hypnat-of-nat* ' *A*) = (*finite A*)
by (*blast intro*!: *starsetNat-eq-hypnat-of-nat-image-finite NatStar-hypnat-of-nat*)

lemma *hypnat-infinite-has-nonstandard-iff*: ($\sim$ *finite A*) = (*hypnat-of-nat* ' *A* < *s* *A*)
apply (*rule iffI*)
apply (*blast intro*!: *hypnat-infinite-has-nonstandard*)
apply (*auto simp add*: *finite-starsetNat-iff* [*symmetric*])
done

4.3 Existence of Infinitely Many Primes: a Nonstandard Proof

lemma *lemma-not-dvd-hypnat-one*: $\sim (\forall n \in -\{0\}.$ *hypnat-of-nat n hdvd 1*)
apply *auto*
apply (*rule-tac x* = *2* **in** *bexI*)
apply (*transfer*, *auto*)
done
declare *lemma-not-dvd-hypnat-one* [*simp*]

lemma *lemma-not-dvd-hypnat-one2*: $\exists n \in -\{0\}. \sim$ *hypnat-of-nat n hdvd 1*

apply (*cut-tac lemma-not-dvd-hypnat-one*)
apply (*auto simp del*: *lemma-not-dvd-hypnat-one*)
done
declare *lemma-not-dvd-hypnat-one2* [*simp*]

lemma *hypnat-gt-zero-gt-one*:
!!N. [| $0 < (N::hypnat)$; $N \neq 1$ |] ==> $1 < N$
by (*transfer*, *simp*)

lemma *hypnat-add-one-gt-one*:
!!N. $0 < N$ ==> $1 < (N::hypnat) + 1$
by (*transfer*, *simp*)

lemma *zero-not-prime*: $\neg$ *prime* *0*
apply *safe*
apply (*drule prime-g-zero*, *auto*)
done
declare *zero-not-prime* [*simp*]

lemma *hypnat-of-nat-zero-not-prime*: *hypnat-of-nat* *0* $\notin$ *starprime*
by (*transfer*, *simp*)
declare *hypnat-of-nat-zero-not-prime* [*simp*]

lemma *hypnat-zero-not-prime*:
0 $\notin$ *starprime*
by (*cut-tac hypnat-of-nat-zero-not-prime*, *simp*)
declare *hypnat-zero-not-prime* [*simp*]

lemma *one-not-prime*: $\neg$ *prime* *1*
apply *safe*
apply (*drule prime-g-one*, *auto*)
done
declare *one-not-prime* [*simp*]

lemma *one-not-prime2*: $\neg$ *prime*(*Suc* *0*)
apply *safe*
apply (*drule prime-g-one*, *auto*)
done
declare *one-not-prime2* [*simp*]

lemma *hypnat-of-nat-one-not-prime*: *hypnat-of-nat* *1* $\notin$ *starprime*
by (*transfer*, *simp*)
declare *hypnat-of-nat-one-not-prime* [*simp*]

lemma *hypnat-one-not-prime*: *1* $\notin$ *starprime*
by (*cut-tac hypnat-of-nat-one-not-prime*, *simp*)
declare *hypnat-one-not-prime* [*simp*]

lemma *hdvd-diff*: !!*k m n*. [| *k hdvd m*; *k hdvd n* |] ==> *k hdvd* (*m* − *n*)
by (*transfer*, *rule dvd-diff*)

lemma *dvd-one-eq-one*: *x dvd* (*1*::*nat*) ==> *x* = *1*
by (*unfold dvd-def*, *auto*)

lemma *hdvd-one-eq-one*: !!*x*. *x hdvd 1* ==> *x* = *1*
by (*transfer*, *rule dvd-one-eq-one*)

theorem *not-finite-prime*: ~ *finite* {*p*. *prime p*}
apply (*rule hypnat-infinite-has-nonstandard-iff* [*THEN iffD2*])
apply (*cut-tac hypnat-dvd-all-hypnat-of-nat*)
apply (*erule exE*)
apply (*erule conjE*)
apply (*subgoal-tac 1* < *N* + *1*)
prefer *2* **apply** (*blast intro*: *hypnat-add-one-gt-one*)
apply (*drule hyperprime-factor-exists*)
apply *auto*
apply (*subgoal-tac k* ∉ *hypnat-of-nat* ' {*p*. *prime p*})
apply (*force simp add*: *starprime-def*, *safe*)
apply (*drule-tac x* = *x* **in** *bspec*)
apply (*rule ccontr*, *simp*)
apply (*drule hdvd-diff*, *assumption*)
apply (*auto dest*: *hdvd-one-eq-one*)
done

end

5 Big O notation – continued

theory *BigO-Complex*
imports *BigO Complex*
begin

Additional lemmas that require the `HOL-Complex` logic image.

lemma *bigo-LIMSEQ1*: *f* =*o O*(*g*) ==> *g* −−−−> *0* ==> *f* −−−−> (*0*::*real*)
  apply (*simp add*: *LIMSEQ-def bigo-alt-def*)
  apply *clarify*
  apply (*drule-tac x* = *r* / *c* **in** *spec*)
  apply (*drule mp*)
  apply (*erule divide-pos-pos*)
  apply *assumption*
  apply *clarify*
  apply (*rule-tac x* = *no* **in** *exI*)
  apply (*rule allI*)
  apply (*drule-tac x* = *n* **in** *spec*)+
  apply (*rule impI*)
  apply (*drule mp*)

```
  apply assumption
  apply (rule order-le-less-trans)
  apply assumption
  apply (rule order-less-le-trans)
  apply (subgoal-tac c * abs(g n) < c * (r / c))
  apply assumption
  apply (erule mult-strict-left-mono)
  apply assumption
  apply simp
done

lemma bigo-LIMSEQ2: f =o g +o O(h) ==> h ----> 0 ==> f ----> a
    ==> g ----> (a::real)
  apply (drule set-plus-imp-minus)
  apply (drule bigo-LIMSEQ1)
  apply assumption
  apply (simp only: func-diff)
  apply (erule LIMSEQ-diff-approach-zero2)
  apply assumption
done

end
```

6 Arithmetic Series for Reals

```
theory Arithmetic-Series-Complex
imports Complex-Main
begin

lemma arith-series-real:
  (2::real) * (∑ i∈{..<n}. a + of-nat i * d) =
  of-nat n * (a + (a + of-nat(n - 1)*d))
proof -
  have
    ((1::real) + 1) * (∑ i∈{..<n}. a + of-nat(i)*d) =
    of-nat(n) * (a + (a + of-nat(n - 1)*d))
    by (rule arith-series-general)
  thus ?thesis by simp
qed

end
```

7 Divergence of the Harmonic Series

```
theory HarmonicSeries
imports Complex-Main
```

begin

8 Abstract

The following document presents a proof of the Divergence of Harmonic Series theorem formalised in the Isabelle/Isar theorem proving system.

Theorem: The series $\sum_{n=1}^{\infty} \frac{1}{n}$ does not converge to any number.

Informal Proof: The informal proof is based on the following auxillary lemmas:

- aux: $\sum_{n=2^m-1}^{2^m} \frac{1}{n} \geq \frac{1}{2}$
- aux2: $\sum_{n=1}^{2^M} \frac{1}{n} = 1 + \sum_{m=1}^{M} \sum_{n=2^m-1}^{2^m} \frac{1}{n}$

From *aux* and *aux2* we can deduce that $\sum_{n=1}^{2^M} \frac{1}{n} \geq 1 + \frac{M}{2}$ for all M. Now for contradiction, assume that $\sum_{n=1}^{\infty} \frac{1}{n} = s$ for some s. Because $\forall n. \frac{1}{n} > 0$ all the partial sums in the series must be less than s. However with our deduction above we can choose $N > 2 * s - 2$ and thus $\sum_{n=1}^{2^N} \frac{1}{n} > s$. This leads to a contradiction and hence $\sum_{n=1}^{\infty} \frac{1}{n}$ is not summable. QED.

9 Formal Proof

lemma *two-pow-sub*:
  0 < m ⟹ (2::nat) ^m − 2 ^(m − 1) = 2 ^(m − 1)
  by (*induct m*) *auto*

We first prove the following auxillary lemma. This lemma simply states that the finite sums: $\frac{1}{2}$, $\frac{1}{3} + \frac{1}{4}$, $\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8}$ etc. are all greater than or equal to $\frac{1}{2}$. We do this by observing that each term in the sum is greater than or equal to the last term, e.g. $\frac{1}{3} > \frac{1}{4}$ and thus $\frac{1}{3} + \frac{1}{4} > \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$.

lemma *harmonic-aux*:
  ∀ m>0. (∑ n∈{(2::nat) ^(m − 1)+1..2 ^m}. 1/real n) ≥ 1/2
  (**is** *∀ m>0. (∑ n∈(?S m). 1/real n) ≥ 1/2*)
proof
  fix *m::nat*
  obtain *tm* **where** *tmdef*: *tm = (2::nat) ^m* **by** *simp*
  {
    assume *mgt0*: *0 < m*
    have *⋀x. x∈(?S m) ⟹ 1/(real x) ≥ 1/(real tm)*
    proof −
      fix *x::nat*
      assume *xs*: *x∈(?S m)*
      have *xgt0*: *x>0*
      proof −
        from *xs* **have**

```
      x ≥ 2^(m − 1) + 1 by auto
    moreover with mgt0 have
      2^(m − 1) + 1 ≥ (1::nat) by auto
    ultimately have
      x ≥ 1 by (rule xtrans)
    thus ?thesis by simp
  qed
  moreover from xs have x ≤ 2^m by auto
  ultimately have
    inverse (real x) ≥ inverse (real ((2::nat)^m)) by simp
  moreover
  from xgt0 have real x ≠ 0 by simp
  then have
    inverse (real x) = 1 / (real x)
    by (rule nonzero-inverse-eq-divide)
  moreover from mgt0 have real tm ≠ 0 by (simp add: tmdef)
  then have
    inverse (real tm) = 1 / (real tm)
    by (rule nonzero-inverse-eq-divide)
  ultimately show
    1/(real x) ≥ 1/(real tm) by (auto simp add: tmdef)
qed
then have
  (∑ n∈(?S m). 1 / real n) ≥ (∑ n∈(?S m). 1/(real tm))
  by (rule setsum-mono)
moreover have
  (∑ n∈(?S m). 1/(real tm)) = 1/2
proof −
  have
    (∑ n∈(?S m). 1/(real tm)) =
     (1/(real tm))∗(∑ n∈(?S m). 1)
    by simp
  also have
    . . . = ((1/(real tm)) ∗ real (card (?S m)))
    by (simp add: real-of-card real-of-nat-def)
  also have
    . . . = ((1/(real tm)) ∗ real (tm − (2^(m − 1))))
    by (simp add: tmdef)
  also from mgt0 have
    . . . = ((1/(real tm)) ∗ real ((2::nat)^(m − 1)))
    by (auto simp: tmdef dest: two-pow-sub)
  also have
    . . . = (real (2::nat))^(m − 1) / (real (2::nat))^m
    by (simp add: tmdef realpow-real-of-nat [symmetric])
  also from mgt0 have
    . . . = (real (2::nat))^(m − 1) / (real (2::nat))^((m − 1) + 1)
    by auto
  also have . . . = 1/2 by simp
  finally show ?thesis .
```

```
    qed
    ultimately have
      (∑ n∈(?S m). 1 / real n) ≥ 1/2
      by − (erule subst)
  }
  thus 0 < m ⟶ 1 / 2 ≤ (∑ n∈(?S m). 1 / real n) by simp
qed
```

We then show that the sum of a finite number of terms from the harmonic series can be regrouped in increasing powers of 2. For example: $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} = 1 + (\frac{1}{2}) + (\frac{1}{3} + \frac{1}{4}) + (\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8})$.

```
lemma harmonic-aux2 [rule-format]:
  0<M ⟹ (∑ n∈{1..(2::nat)^M}. 1/real n) =
   (1 + (∑ m∈{1..M}. ∑ n∈{(2::nat)^(m − 1)+1..2^m}. 1/real n))
  (is 0<M ⟹ ?LHS M = ?RHS M)
proof (induct M)
  case 0 show ?case by simp
next
  case (Suc M)
  have ant: 0 < Suc M by fact
  {
    have suc: ?LHS (Suc M) = ?RHS (Suc M)
    proof cases — show that LHS = c and RHS = c, and thus LHS = RHS
      assume mz: M=0
      {
        then have
          ?LHS (Suc M) = ?LHS 1 by simp
        also have
          ... = (∑ n∈{(1::nat)..2}. 1/real n) by simp
        also have
          ... = ((∑ n∈{Suc 1..2}. 1/real n) + 1/(real (1::nat)))
          by (subst setsum-head)
             (auto simp: atLeastSucAtMost-greaterThanAtMost)
        also have
          ... = ((∑ n∈{2..2::nat}. 1/real n) + 1/(real (1::nat)))
          by (simp add: nat-number)
        also have
          ... =  1/(real (2::nat)) + 1/(real (1::nat)) by simp
        finally have
          ?LHS (Suc M) = 1/2 + 1 by simp
      }
      moreover
      {
        from mz have
          ?RHS (Suc M) = ?RHS 1 by simp
        also have
          ... = (∑ n∈{((2::nat)^0)+1..2^1}. 1/real n) + 1
          by simp
        also have
```

```
      ... = (∑ n∈{2::nat..2}. 1/real n) + 1
    proof -
      have (2::nat)^0 = 1 by simp
      then have (2::nat)^0+1 = 2 by simp
      moreover have (2::nat)^1 = 2 by simp
      ultimately have {((2::nat)^0)+1..2^1} = {2::nat..2} by auto
      thus ?thesis by simp
    qed
    also have
      ... = 1/2 + 1
      by simp
    finally have
      ?RHS (Suc M) = 1/2 + 1 by simp
  }
  ultimately show ?LHS (Suc M) = ?RHS (Suc M) by simp
next
  assume mnz: M≠0
  then have mgtz: M>0 by simp
  with Suc have suc:
    (?LHS M) = (?RHS M) by blast
  have
    (?LHS (Suc M)) =
     ((?LHS M) + (∑ n∈{(2::nat)^M+1..2^(Suc M)}. 1 / real n))
  proof -
    have
      {1..(2::nat)^(Suc M)} =
       {1..(2::nat)^M}∪{(2::nat)^M+1..(2::nat)^(Suc M)}
      by auto
    moreover have
      {1..(2::nat)^M}∩{(2::nat)^M+1..(2::nat)^(Suc M)} = {}
      by auto
    moreover have
      finite {1..(2::nat)^M} and finite {(2::nat)^M+1..(2::nat)^(Suc M)}
      by auto
    ultimately show ?thesis
      by (auto intro: setsum-Un-disjoint)
  qed
  moreover
  {
    have
      (?RHS (Suc M)) =
       (1 + (∑ m∈{1..M}. ∑ n∈{(2::nat)^(m - 1)+1..2^m}. 1/real n) +
       (∑ n∈{(2::nat)^(Suc M - 1)+1..2^(Suc M)}. 1/real n)) by simp
    also have
      ... = (?RHS M) + (∑ n∈{(2::nat)^M+1..2^(Suc M)}. 1/real n)
      by simp
    also from suc have
      ... = (?LHS M) + (∑ n∈{(2::nat)^M+1..2^(Suc M)}. 1/real n)
      by simp
```

```
      finally have
        (?RHS (Suc M)) = ... by simp
    }
    ultimately show ?LHS (Suc M) = ?RHS (Suc M) by simp
  qed
 }
 thus ?case by simp
qed
```

Using *harmonic-aux* and *harmonic-aux2* we now show that each group sum is greater than or equal to $\frac{1}{2}$ and thus the finite sum is bounded below by a value proportional to the number of elements we choose.

```
lemma harmonic-aux3 [rule-format]:
  shows ∀ (M::nat). (∑ n∈{1..(2::nat)^M}. 1 / real n) ≥ 1 + (real M)/2
  (is ∀ M. ?P M ≥ -)
proof (rule allI, cases)
  fix M::nat
  assume M=0
  then show ?P M ≥ 1 + (real M)/2 by simp
next
  fix M::nat
  assume M≠0
  then have M > 0 by simp
  then have
    (?P M) =
     (1 + (∑ m∈{1..M}. ∑ n∈{(2::nat)^(m - 1)+1..2^m}. 1/real n))
    by (rule harmonic-aux2)
  also have
    ... ≥ (1 + (∑ m∈{1..M}. 1/2))
  proof -
    let ?f = (λx. 1/2)
    let ?g = (λx. (∑ n∈{(2::nat)^(x - 1)+1..2^x}. 1/real n))
    from harmonic-aux have ⋀x. x∈{1..M} ⟹ ?f x ≤ ?g x by simp
    then have (∑ m∈{1..M}. ?g m) ≥ (∑ m∈{1..M}. ?f m) by (rule setsum-mono)
    thus ?thesis by simp
  qed
  finally have (?P M) ≥ (1 + (∑ m∈{1..M}. 1/2)) .
  moreover
  {
    have
      (∑ m∈{1..M}. (1::real)/2) = 1/2 * (∑ m∈{1..M}. 1)
      by auto
    also have
      ... = 1/2*(real (card {1..M}))
      by (simp only: real-of-card[symmetric])
    also have
      ... = 1/2*(real M) by simp
    also have
      ... = (real M)/2 by simp
```

```
  finally have (∑ m∈{1..M}. (1::real)/2) = (real M)/2 .
 }
 ultimately show (?P M) ≥ (1 + (real M)/2) by simp
qed
```

The final theorem shows that as we take more and more elements (see *harmonic-aux3*) we get an ever increasing sum. By assuming the sum converges, the lemma *series-pos-less* (⟦*summable ?f*; ∀ *m*≥*?n*. *0* < *?f m*⟧ ⟹ *setsum ?f* {*0*..<*?n*} < *suminf ?f*) states that each sum is bounded above by the series' limit. This contradicts our first statement and thus we prove that the harmonic series is divergent.

```
theorem DivergenceOfHarmonicSeries:
 shows ¬summable (λn. 1/real (Suc n))
 (is ¬summable ?f)
proof — by contradiction
 let ?s = suminf ?f — let ?s equal the sum of the harmonic series
 assume sf: summable ?f
 then obtain n::nat where ndef: n = nat ⌈2 * ?s⌉ by simp
 then have ngt: 1 + real n/2 > ?s
 proof −
  have ∀ n. 0 ≤ ?f n by simp
  with sf have ?s ≥ 0
   by − (rule suminf-0-le, simp-all)
  then have cgt0: ⌈2*?s⌉ ≥ 0 by simp

  from ndef have n = nat ⌈(2*?s)⌉ .
  then have real n = real (nat ⌈2*?s⌉) by simp
  with cgt0 have real n = real ⌈2*?s⌉
   by (auto dest: real-nat-eq-real)
  then have real n ≥ 2*(?s) by simp
  then have real n/2 ≥ (?s) by simp
  then show 1 + real n/2 > (?s) by simp
 qed

 obtain j where jdef: j = (2::nat)^n by simp
 have ∀ m≥j. 0 < ?f m by simp
 with sf have (∑ i∈{0..<j}. ?f i) < ?s by (rule series-pos-less)
 then have (∑ i∈{1..<Suc j}. 1/(real i)) < ?s
  apply −
  apply (subst(asm) setsum-shift-bounds-Suc-ivl [symmetric])
  by simp
 with jdef have
  (∑ i∈{1..< Suc ((2::nat)^n)}. 1 / (real i)) < ?s by simp
 then have
  (∑ i∈{1..(2::nat)^n}. 1 / (real i)) < ?s
  by (simp only: atLeastLessThanSuc-atLeastAtMost)
 moreover from harmonic-aux3 have
  (∑ i∈{1..(2::nat)^n}. 1 / (real i)) ≥ 1 + real n/2 by simp
 moreover from ngt have 1 + real n/2 > ?s by simp
```

ultimately show *False* **by** *simp*
qed

end

10 Denumerability of the Rationals

theory *DenumRat*
imports
Complex-Main NatPair
begin

lemma *nat-to-int-surj*: $\exists f{::}nat \Rightarrow int.\ surj\ f$
proof
let $?f = \lambda n.\ if\ (n\ mod\ 2 = 0)\ then\ -\ int\ (n\ div\ 2)\ else\ int\ ((n - 1)\ div\ 2 + 1)$
have $\forall y.\ \exists x.\ y = ?f\ x$
proof
fix $y{::}int$
{
assume *yl0*: $y \le 0$
then obtain n **where** *ndef*: $n = nat\ (-\ y * 2)$ **by** *simp*
from *yl0* **have** *g0*: $-\ y * 2 \ge 0$ **by** *simp*
hence $nat\ (-\ y * 2)\ mod\ (nat\ 2) = nat\ ((-y * 2)\ mod\ 2)$ **by** (*subst nat-mod-distrib*, *auto*)
moreover have $(-y * 2)\ mod\ 2 = 0$ **by** *arith*
ultimately have $nat\ (-\ y * 2)\ mod\ 2 = 0$ **by** *simp*
with *ndef* **have** $n\ mod\ 2 = 0$ **by** *simp*
hence $?f\ n = -\ int\ (n\ div\ 2)$ **by** *simp*
also with *ndef* **have** $\ldots = -\ int\ (nat\ (-y * 2)\ div\ 2)$ **by** *simp*
also with *g0* **have** $\ldots = -\ int\ (nat\ (((-y) * 2)\ div\ 2))$ **using** *nat-div-distrib* **by** *auto*
also have $\ldots = -\ int\ (nat\ (-y))$ **using** *zdiv-zmult-self1* [*of* $2\ -\ y$]
by *simp*
also from *yl0* **have** $\ldots = y$ **using** *nat-0-le* **by** *auto*
finally have $?f\ n = y$ **.**
hence $\exists x.\ y = ?f\ x$ **by** *blast*
}
moreover
{
assume $\neg(y \le 0)$
hence *yg0*: $y > 0$ **by** *simp*
hence *yn0*: $y \ne 0$ **by** *simp*
from *yg0* **have** *g0*: $y{*}2 + -2 \ge 0$ **by** *arith*
from *yg0* **obtain** n **where** *ndef*: $n = nat\ (y * 2 - 1)$ **by** *simp*
from *yg0* **have** $nat\ (y{*}2 - 1)\ mod\ 2 = nat\ ((y{*}2 - 1)\ mod\ 2)$ **using** *nat-mod-distrib* **by** *auto*
also have $\ldots = nat\ ((y{*}2 + -\ 1)\ mod\ 2)$ **by** (*auto simp add*: *diff-int-def*)
also have $\ldots = nat\ (1)$ **by** (*auto simp add*: *zmod-zadd-left-eq*)

finally have *n mod 2 = 1* using *ndef* by *auto*
hence *?f n = int ((n − 1) div 2 + 1)* by *simp*
also with *ndef* have … = *int ((nat (y*2 − 1) − 1) div 2 + 1)* by *simp*
also with *yg0* have … = *int (nat (y*2 − 2) div 2 + 1)* by *arith*
also have … = *int (nat (y*2 + −2) div 2 + 1)* by *(simp add: diff-int-def)*
also have … = *int (nat (y*2 + −2) div (nat 2) + 1)* by *auto*
also from *g0* have … = *int (nat ((y*2 + −2) div 2) + 1)*
using *nat-div-distrib* by *auto*
also have … = *int (nat ((y*2) div 2 + (−2) div 2 + ((y*2) mod 2 + (−2) mod 2) div 2) + 1)*
by *(auto simp add: zdiv-zadd1-eq)*
also from *yg0 g0* have … = *int (nat (y))*
by *(auto)*
finally have *?f n = y* using *yg0* by *auto*
hence *∃x. y = ?f x* by *blast*
}
ultimately show *∃x. y = ?f x* by *(rule case-split)*
qed
thus *surj ?f* by *(fold surj-def)*
qed

lemma *nat2-to-int2-surj*: *∃f::(nat*nat)⇒(int*int). surj f*
proof −
from *nat-to-int-surj* obtain *g::nat⇒int* where *surj g* ..
hence *aux*: *∀y. ∃x. y = g x* by *(unfold surj-def)*
let *?f = λn. (g (fst n), g (snd n))*
{
fix *y::(int*int)*
from *aux* have *∃x1 x2. fst y = g x1 ∧ snd y = g x2* by *auto*
hence *∃x. fst y = g (fst x) ∧ snd y = g (snd x)* by *auto*
hence *∃x. (fst y, snd y) = (g (fst x), g (snd x))* by *blast*
hence *∃x. y = ?f x* by *auto*
}
hence *∀y. ∃x. y = ?f x* by *auto*
hence *surj ?f* by *(fold surj-def)*
thus *?thesis* by *auto*
qed

lemma *rat-denum*:
∃f::nat⇒rat. surj f
proof −
have *inj nat2-to-nat* by *(rule nat2-to-nat-inj)*
hence *surj (inv nat2-to-nat)* by *(rule inj-imp-surj-inv)*
moreover from *nat2-to-int2-surj* obtain *h::(nat*nat)⇒(int*int)* where *surj h*
..
ultimately have *surj (h o (inv nat2-to-nat))* by *(rule comp-surj)*
hence *∃f::nat⇒(int*int). surj f* by *auto*
then obtain *g::nat⇒(int*int)* where *surj g* by *auto*
hence *gdef*: *∀y. ∃x. y = g x* by *(unfold surj-def)*

```
  {
    fix y
    obtain a b where y: y = Fract a b by (cases y)
    from gdef
    obtain x where (a,b) = g x by blast
    hence g x = (a,b) ..
    with y have y = (split Fract o g) x by simp
    hence ∃x. y = (split Fract o g) x ..
  }
  hence surj (split Fract o g)
    by (simp add: surj-def)
  thus ?thesis by blast
qed


end
```

11 Type of indices

```
theory Code-Index
imports PreList
begin
```

Indices are isomorphic to HOL *int* but mapped to target-language builtin integers

11.1 Datatype of indices

```
datatype index = index-of-int int

lemmas [code func del] = index.recs index.cases

fun
  int-of-index :: index ⇒ int
where
  int-of-index (index-of-int k) = k
lemmas [code func del] = int-of-index.simps

lemma index-id [simp]:
  index-of-int (int-of-index k) = k
  by (cases k) simp-all

lemma index:
  (⋀k::index. PROP P k) ≡ (⋀k::int. PROP P (index-of-int k))
proof
  fix k :: int
  assume ⋀k::index. PROP P k
  then show PROP P (index-of-int k) .
next
```

fix *k* :: *index*
assume $\bigwedge$*k*::*int*. *PROP P* (*index-of-int k*)
then have *PROP P* (*index-of-int* (*int-of-index k*)) **.**
then show *PROP P k* **by** *simp*
qed

lemma [*code func*]: *size* (*k*::*index*) = *0*
by (*cases k*) *simp-all*

11.2 Built-in integers as datatype on numerals

instance *index* :: *number*
number-of ≡ *index-of-int* **..**

code-datatype *number-of* :: *int* ⇒ *index*

lemma *number-of-index-id* [*simp*]:
number-of (*int-of-index k*) = *k*
unfolding *number-of-index-def* **by** *simp*

lemma *number-of-index-shift*:
number-of k = *index-of-int* (*number-of k*)
by (*simp add*: *number-of-is-id number-of-index-def*)

lemma *int-of-index-number-of* [*simp*]:
int-of-index (*number-of k*) = *number-of k*
unfolding *number-of-index-def number-of-is-id* **by** *simp*

11.3 Basic arithmetic

instance *index* :: *zero*
[*simp*]: *0* ≡ *index-of-int 0* **..**
lemmas [*code func del*] = *zero-index-def*

instance *index* :: *one*
[*simp*]: *1* ≡ *index-of-int 1* **..**
lemmas [*code func del*] = *one-index-def*

instance *index* :: *plus*
[*simp*]: *k* + *l* ≡ *index-of-int* (*int-of-index k* + *int-of-index l*) **..**
lemmas [*code func del*] = *plus-index-def*
lemma *plus-index-code* [*code func*]:
index-of-int k + *index-of-int l* = *index-of-int* (*k* + *l*)
unfolding *plus-index-def* **by** *simp*

instance *index* :: *minus*
[*simp*]: − *k* ≡ *index-of-int* (− *int-of-index k*)
[*simp*]: *k* − *l* ≡ *index-of-int* (*int-of-index k* − *int-of-index l*) **..**
lemmas [*code func del*] = *uminus-index-def minus-index-def*
lemma *uminus-index-code* [*code func*]:

$-$ *index-of-int k* $\equiv$ *index-of-int* $(-\ k)$
unfolding *uminus-index-def* **by** *simp*
lemma *minus-index-code* [*code func*]:
index-of-int k $-$ *index-of-int l* $=$ *index-of-int* $(k - l)$
unfolding *minus-index-def* **by** *simp*

instance *index* :: *times*
[*simp*]: $k * l \equiv$ *index-of-int* (*int-of-index k* $*$ *int-of-index l*) **..**
lemmas [*code func del*] = *times-index-def*
lemma *times-index-code* [*code func*]:
index-of-int k $*$ *index-of-int l* $=$ *index-of-int* $(k * l)$
unfolding *times-index-def* **by** *simp*

instance *index* :: *ord*
[*simp*]: $k \leq l \equiv$ *int-of-index k* $\leq$ *int-of-index l*
[*simp*]: $k < l \equiv$ *int-of-index k* $<$ *int-of-index l* **..**
lemmas [*code func del*] = *less-eq-index-def less-index-def*
lemma *less-eq-index-code* [*code func*]:
index-of-int k $\leq$ *index-of-int l* $\longleftrightarrow k \leq l$
unfolding *less-eq-index-def* **by** *simp*
lemma *less-index-code* [*code func*]:
index-of-int k $<$ *index-of-int l* $\longleftrightarrow k < l$
unfolding *less-index-def* **by** *simp*

instance *index* :: *Divides.div*
[*simp*]: *k div l* $\equiv$ *index-of-int* (*int-of-index k div int-of-index l*)
[*simp*]: *k mod l* $\equiv$ *index-of-int* (*int-of-index k mod int-of-index l*) **..**

instance *index* :: *ring-1*
by *default* (*auto simp add*: *left-distrib right-distrib*)

lemma *of-nat-index*: *of-nat n* $=$ *index-of-int* (*of-nat n*)
proof (*induct n*)
case *0* **show** *?case* **by** *simp*
next
case (*Suc n*)
then have *int-of-index* (*index-of-int* (*int n*))
$=$ *int-of-index* (*of-nat n*) **by** *simp*
then have *int n* $=$ *int-of-index* (*of-nat n*) **by** *simp*
then show *?case* **by** *simp*
qed

instance *index* :: *number-ring*
by *default*
(*simp-all add*: *left-distrib number-of-index-def of-int-of-nat of-nat-index*)

lemma *zero-index-code* [*code inline*, *code func*]:
(*0*::*index*) $=$ *Numeral0*
by *simp*

lemma *one-index-code* [*code inline*, *code func*]:
$(1::index) = Numeral1$
by *simp*

instance *index* :: *abs*
$|k| \equiv$ *if* $k < 0$ *then* $-k$ *else* k **..**

lemma *index-of-int* [*code func*]:
index-of-int $k =$ (*if* $k = 0$ *then* 0
else if $k = -1$ *then* -1
else let $(l, m) =$ *divAlg* $(k, 2)$ *in* $2 *$ *index-of-int* $l +$
(*if* $m = 0$ *then* 0 *else* 1))
by (*simp add*: *number-of-index-shift Let-def split-def divAlg-mod-div*) *arith*

lemma *int-of-index* [*code func*]:
int-of-index $k =$ (*if* $k = 0$ *then* 0
else if $k = -1$ *then* -1
else let $l = k$ *div* 2; $m = k$ *mod* 2 *in* $2 *$ *int-of-index* $l +$
(*if* $m = 0$ *then* 0 *else* 1))
by (*auto simp add*: *number-of-index-shift Let-def split-def*) *arith*

11.4 Conversion to and from *nat*

definition
nat-of-index :: *index* $\Rightarrow$ *nat*
where
[*code func del*]: *nat-of-index* = *nat o int-of-index*

definition
nat-of-index-aux :: *index* $\Rightarrow$ *nat* $\Rightarrow$ *nat* **where**
[*code func del*]: *nat-of-index-aux* i n = *nat-of-index* $i + n$

lemma *nat-of-index-aux-code* [*code*]:
nat-of-index-aux i n = (*if* $i \leq 0$ *then* n *else* *nat-of-index-aux* $(i - 1)$ (*Suc* n))
by (*auto simp add*: *nat-of-index-aux-def nat-of-index-def*)

lemma *nat-of-index-code* [*code*]:
nat-of-index i = *nat-of-index-aux* i 0
by (*simp add*: *nat-of-index-aux-def*)

definition
index-of-nat :: *nat* $\Rightarrow$ *index*
where
[*code func del*]: *index-of-nat* = *index-of-int o of-nat*

lemma *index-of-nat* [*code func*]:
index-of-nat $0 = 0$
index-of-nat (*Suc* n) = *index-of-nat* $n + 1$

unfolding *index-of-nat-def* **by** *simp-all*

lemma *index-nat-id* [*simp*]:
nat-of-index (*index-of-nat n*) = *n*
index-of-nat (*nat-of-index i*) = (*if i* ≤ *0 then 0 else i*)
unfolding *index-of-nat-def nat-of-index-def* **by** *simp-all*

11.5 ML interface

ML ⟪
structure Index =
struct

fun mk k = @{*term index-of-int*} $ *HOLogic.mk-number* @{*typ index*} *k*;

end;
⟫

11.6 Code serialization

code-type *index*
(*SML int*)
(*OCaml int*)
(*Haskell Integer*)

code-instance *index* :: *eq*
(*Haskell* −)

setup ⟪
fold (*fn target* => *CodeTarget.add-pretty-numeral target true*
@{*const-name number-index-inst.number-of-index*}
@{*const-name Numeral.B0*} @{*const-name Numeral.B1*}
@{*const-name Numeral.Pls*} @{*const-name Numeral.Min*}
@{*const-name Numeral.Bit*}
) [*SML*, *OCaml*, *Haskell*]
⟫

code-reserved *SML int*
code-reserved *OCaml int*

code-const *op* + :: *index* ⇒ *index* ⇒ *index*
(*SML Int.+* ((-), (-)))
(*OCaml Pervasives.+*)
(*Haskell* **infixl** *6* +)

code-const *uminus* :: *index* ⇒ *index*
(*SML Int.~*)
(*OCaml Pervasives.~−*)
(*Haskell negate*)

code-const *op* − :: *index* ⇒ *index* ⇒ *index*
(*SML Int.*− ((-), (-)))
(*OCaml Pervasives.*−)
(*Haskell* **infixl** *6* −)

code-const *op* ∗ :: *index* ⇒ *index* ⇒ *index*
(*SML Int.*∗ ((-), (-)))
(*OCaml Pervasives.*∗)
(*Haskell* **infixl** *7* ∗)

code-const *op* = :: *index* ⇒ *index* ⇒ *bool*
(*SML* !((- : *Int.int*) = -))
(*OCaml* !((- : *Pervasives.int*) = -))
(*Haskell* **infixl** *4* ==)

code-const *op* ≤ :: *index* ⇒ *index* ⇒ *bool*
(*SML Int.*<= ((-), (-)))
(*OCaml* !((- : *Pervasives.int*) <= -))
(*Haskell* **infix** *4* <=)

code-const *op* < :: *index* ⇒ *index* ⇒ *bool*
(*SML Int.*< ((-), (-)))
(*OCaml* !((- : *Pervasives.int*) < -))
(*Haskell* **infix** *4* <)

code-reserved *SML Int*
code-reserved *OCaml Pervasives*

end

12 Pretty integer literals for code generation

theory *Code-Integer*
imports *IntArith Code-Index*
begin

HOL numeral expressions are mapped to integer literals in target languages, using predefined target language operations for abstract integer operations.

code-type *int*
(*SML IntInf.int*)
(*OCaml Big′-int.big′-int*)
(*Haskell Integer*)

code-instance *int* :: *eq*
(*Haskell* −)

setup ⟪
fold (*fn target* => *CodeTarget.add-pretty-numeral target true*

@{*const-name number-int-inst.number-of-int*}
@{*const-name Numeral.B0*} @{*const-name Numeral.B1*}
@{*const-name Numeral.Pls*} @{*const-name Numeral.Min*}
@{*const-name Numeral.Bit*}
) [*SML*, *OCaml*, *Haskell*]
⟫

code-const *Numeral.Pls* **and** *Numeral.Min* **and** *Numeral.Bit*
(*SML raise/ Fail/ Pls*
and *raise/ Fail/ Min*
and !((-);/ (-);/ *raise/ Fail/ Bit*))
(*OCaml failwith/ Pls*
and *failwith/ Min*
and !((-);/ (-);/ *failwith/ Bit*))
(*Haskell error/ Pls*
and *error/ Min*
and *error/ Bit*)

code-const *Numeral.pred*
(*SML IntInf*.– ((-), *1*))
(*OCaml Big'-int.pred'-big'-int*)
(*Haskell* !(-/ –/ *1*))

code-const *Numeral.succ*
(*SML IntInf*.+ ((-), *1*))
(*OCaml Big'-int.succ'-big'-int*)
(*Haskell* !(-/ +/ *1*))

code-const *op* + :: *int* ⇒ *int* ⇒ *int*
(*SML IntInf*.+ ((-), (-)))
(*OCaml Big'-int.add'-big'-int*)
(*Haskell* **infixl** *6* +)

code-const *uminus* :: *int* ⇒ *int*
(*SML IntInf*.~)
(*OCaml Big'-int.minus'-big'-int*)
(*Haskell negate*)

code-const *op* – :: *int* ⇒ *int* ⇒ *int*
(*SML IntInf*.– ((-), (-)))
(*OCaml Big'-int.sub'-big'-int*)
(*Haskell* **infixl** *6* –)

code-const *op* ∗ :: *int* ⇒ *int* ⇒ *int*
(*SML IntInf*.∗ ((-), (-)))
(*OCaml Big'-int.mult'-big'-int*)
(*Haskell* **infixl** *7* ∗)

code-const *op* = :: *int* ⇒ *int* ⇒ *bool*

(*SML* !((- : *IntInf.int*) = -))
(*OCaml Big′-int.eq′-big′-int*)
(*Haskell* **infixl** *4* ==)

code-const *op* ≤ :: *int* ⇒ *int* ⇒ *bool*
(*SML IntInf.*<= ((-), (-)))
(*OCaml Big′-int.le′-big′-int*)
(*Haskell* **infix** *4* <=)

code-const *op* < :: *int* ⇒ *int* ⇒ *bool*
(*SML IntInf.*< ((-), (-)))
(*OCaml Big′-int.lt′-big′-int*)
(*Haskell* **infix** *4* <)

code-const *index-of-int* **and** *int-of-index*
(*SML IntInf.toInt* **and** *IntInf.fromInt*)
(*OCaml Big′-int.int′-of′-big′-int* **and** *Big′-int.big′-int′-of′-int*)
(*Haskell* - **and** -)

code-reserved *SML IntInf*
code-reserved *OCaml Big-int*

end

13 Quatifier elimination for R(0,1,+,floor,¡)

theory *MIR*
imports *Real GCD Code-Integer*
uses (*mireif.ML*) (*mirtac.ML*)
begin

declare *real-of-int-floor-cancel* [*simp del*]

fun *alluopairs*:: ′*a list* ⇒ (′*a* × ′*a*) *list* **where**
alluopairs [] = []
| *alluopairs* (*x*#*xs*) = (*map* (*Pair x*) (*x*#*xs*))@(*alluopairs xs*)

lemma *alluopairs-set1*: *set* (*alluopairs xs*) ≤ {(*x*,*y*). *x*∈ *set xs* ∧ *y*∈ *set xs*}
by (*induct xs*, *auto*)

lemma *alluopairs-set*:
⟦*x*∈ *set xs* ; *y* ∈ *set xs*⟧ ⟹ (*x*,*y*) ∈ *set* (*alluopairs xs*) ∨ (*y*,*x*) ∈ *set* (*alluopairs xs*)
by (*induct xs*, *auto*)

lemma *alluopairs-ex*:
assumes *Pc*: ∀ *x y*. *P x y* = *P y x*
shows (∃ *x* ∈ *set xs*. ∃ *y* ∈ *set xs*. *P x y*) = (∃ (*x*,*y*) ∈ *set* (*alluopairs xs*). *P x y*)

```
proof
  assume ∃x∈set xs. ∃y∈set xs. P x y
  then obtain x y where x: x ∈ set xs and y:y ∈ set xs and P: P x y  by blast
  from alluopairs-set[OF x y] P Pc show∃(x, y)∈set (alluopairs xs). P x y
    by auto
next
  assume ∃(x, y)∈set (alluopairs xs). P x y
  then obtain x and y  where xy:(x,y) ∈ set (alluopairs xs) and P: P x y by
blast+
  from xy have x ∈ set xs ∧ y∈ set xs using alluopairs-set1 by blast
  with P show ∃x∈set xs. ∃y∈set xs. P x y by blast
qed


consts iupt :: int × int ⇒ int list
recdef iupt measure (λ (i,j). nat (j−i +1))
  iupt (i,j) = (if j <i then [] else (i# iupt(i+1, j)))

lemma iupt-set: set (iupt(i,j)) = {i .. j}
proof(induct rule: iupt.induct)
  case (1 a b)
  show ?case
    using prems by (simp add: simp-from-to)
qed

lemma nth-pos2: 0 < n ⟹ (x#xs) ! n = xs ! (n − 1)
using Nat.gr0-conv-Suc
by clarsimp


lemma myl: ∀ (a::'a::{pordered-ab-group-add}) (b::'a). (a ≤ b) = (0 ≤ b − a)
proof(clarify)
  fix x y ::'a
   have (x ≤ y) = (x − y ≤ 0) by (simp only: le-iff-diff-le-0[where a=x and
b=y])
  also have ... = (− (y − x) ≤ 0) by simp
  also have ... = (0 ≤ y − x) by (simp only: neg-le-0-iff-le[where a=y−x])
  finally show (x ≤ y) = (0 ≤ y − x) .
qed

lemma myless: ∀ (a::'a::{pordered-ab-group-add}) (b::'a). (a < b) = (0 < b − a)

proof(clarify)
  fix x y ::'a
  have (x < y) = (x − y < 0) by (simp only: less-iff-diff-less-0[where a=x and
b=y])
  also have ... = (− (y − x) < 0) by simp
  also have ... = (0 < y − x) by (simp only: neg-less-0-iff-less[where a=y−x])
  finally show (x < y) = (0 < y − x) .
```

qed

lemma *myeq*: $\forall$ (a::$'a$::{*pordered-ab-group-add*}) (b::$'a$). (a = b) = (*0* = b − a)
  by *auto*

lemma *floor-int-eq*: (*real* n≤ x ∧ x < *real* (n+*1*)) = (*floor* x = n)
proof(*auto*)
  assume *lb*: *real* n ≤ x
    and *ub*: x < *real* n + *1*
  have *real* (*floor* x) ≤ x **by** *simp*
  hence *real* (*floor* x) < *real* (n + *1*) **using** *ub* **by** *arith*
  hence *floor* x < n+*1* **by** *simp*
  moreover from *lb* **have** n ≤ *floor* x **using** *floor-mono2*[**where** x=*real* n **and** y=x]
    by *simp* **ultimately show** *floor* x = n **by** *simp*
qed

lemma *dvd-period*:
  assumes *advdd*: (a::*int*) *dvd* d
  shows (a *dvd* (x + t)) = (a *dvd* ((x+ c∗d) + t))
  using *advdd*
proof−
  {**fix** x k
    from *inf-period*(*3*)[*OF advdd*, *rule-format*, **where** x=x **and** k=−k]
    have ((a::*int*) *dvd* (x + t)) = (a *dvd* (x+k∗d + t)) **by** *simp*}
  hence ∀ x.∀ k. ((a::*int*) *dvd* (x + t)) = (a *dvd* (x+k∗d + t)) **by** *simp*
  then show *?thesis* **by** *simp*
qed

definition
  rdvd:: *real* ⇒ *real* ⇒ *bool* (**infixl** *rdvd 50*)
where
  rdvd-def: x *rdvd* y ⟷ (∃ k::*int*. y = x ∗ *real* k)

lemma *int-rdvd-real*:
  shows *real* (i::*int*) *rdvd* x = (i *dvd* (*floor* x) ∧ *real* (*floor* x) = x) (**is** *?l* = *?r*)
proof
  assume *?l*
  hence *th*: ∃ k. x=*real* (i∗k) **by** (*simp add*: *rdvd-def*)
  hence *th'*: *real* (*floor* x) = x **by** (*auto simp del*: *real-of-int-mult*)
  with *th* **have** ∃ k. *real* (*floor* x) = *real* (i∗k) **by** *simp*
  hence ∃ k. *floor* x = i∗k **by** (*simp only*: *real-of-int-inject*)
  thus *?r* **using** *th'* **by** (*simp add*: *dvd-def*)
next
  assume *?r* **hence** (i::*int*) *dvd* ⌊x::*real*⌋ **..**
  hence ∃ k. *real* (*floor* x) = *real* (i∗k)

by (*simp only*: *real-of-int-inject*) (*simp add*: *dvd-def*)
thus *?l* **using** *prems* **by** (*simp add*: *rdvd-def*)
qed

lemma *int-rdvd-iff*: (*real* (*i*::*int*) *rdvd real t*) = (*i dvd t*)
by (*auto simp add*: *rdvd-def dvd-def*) (*rule-tac x*=*k* **in** *exI*, *simp only* :*real-of-int-mult*[*symmetric*])

lemma *rdvd-abs1*:
(*abs* (*real d*) *rdvd t*) = (*real* (*d* ::*int*) *rdvd t*)
proof
assume *d*: *real d rdvd t*
from *d int-rdvd-real* **have** *d2*: *d dvd* (*floor t*) **and** *ti*: *real* (*floor t*) = *t* **by** *auto*

from *iffD2*[*OF zdvd-abs1*] *d2* **have** (*abs d*) *dvd* (*floor t*) **by** *blast*
with *ti int-rdvd-real*[*symmetric*] **have** *real* (*abs d*) *rdvd t* **by** *blast*
thus *abs* (*real d*) *rdvd t* **by** *simp*
next
assume *abs* (*real d*) *rdvd t* **hence** *real* (*abs d*) *rdvd t* **by** *simp*
with *int-rdvd-real*[**where** *i*=*abs d* **and** *x*=*t*] **have** *d2*: *abs d dvd floor t* **and** *ti*:
real (*floor t*) =*t* **by** *auto*
from *iffD1*[*OF zdvd-abs1*] *d2* **have** *d dvd floor t* **by** *blast*
with *ti int-rdvd-real*[*symmetric*] **show** *real d rdvd t* **by** *blast*
qed

lemma *rdvd-minus*: (*real* (*d*::*int*) *rdvd t*) = (*real d rdvd* −*t*)
apply (*auto simp add*: *rdvd-def*)
apply (*rule-tac x*=−*k* **in** *exI*, *simp*)
apply (*rule-tac x*=−*k* **in** *exI*, *simp*)
done

lemma *rdvd-left-0-eq*: (*0 rdvd t*) = (*t*=*0*)
by (*auto simp add*: *rdvd-def*)

lemma *rdvd-mult*:
assumes *knz*: *k*≠*0*
shows (*real* (*n*::*int*) ∗ *real* (*k*::*int*) *rdvd x* ∗ *real k*) = (*real n rdvd x*)
using *knz* **by** (*simp add*:*rdvd-def*)

lemma *rdvd-trans*: **assumes** *mn*:*m rdvd n* **and** *nk*:*n rdvd k*
shows *m rdvd k*
proof−
from *rdvd-def mn* **obtain** *c* **where** *nmc*:*n* = *m* ∗ *real* (*c*::*int*) **by** *auto*
from *rdvd-def nk* **obtain** *c′* **where** *nkc*:*k* = *n* ∗ *real* (*c′*::*int*) **by** *auto*
hence *k* = *m* ∗ *real* (*c* ∗ *c′*) **using** *nmc* **by** *simp*
thus *?thesis* **using** *rdvd-def* **by** *blast*
qed

datatype *num* = *C int* | *Bound nat* | *CN nat int num* | *Neg num* | *Add num num*| *Sub num num*
| *Mul int num* | *Floor num*| *CF int num num*

fun *num-size* :: *num* ⇒ *nat* **where**
num-size (*C c*) = *1*
| *num-size* (*Bound n*) = *1*
| *num-size* (*Neg a*) = *1* + *num-size a*
| *num-size* (*Add a b*) = *1* + *num-size a* + *num-size b*
| *num-size* (*Sub a b*) = *3* + *num-size a* + *num-size b*
| *num-size* (*CN n c a*) = *4* + *num-size a*
| *num-size* (*CF c a b*) = *4* + *num-size a* + *num-size b*
| *num-size* (*Mul c a*) = *1* + *num-size a*
| *num-size* (*Floor a*) = *1* + *num-size a*

fun *Inum* :: *real list* ⇒ *num* ⇒ *real* **where**
Inum bs (*C c*) = (*real c*)
| *Inum bs* (*Bound n*) = *bs*!*n*
| *Inum bs* (*CN n c a*) = (*real c*) ∗ (*bs*!*n*) + (*Inum bs a*)
| *Inum bs* (*Neg a*) = −(*Inum bs a*)
| *Inum bs* (*Add a b*) = *Inum bs a* + *Inum bs b*
| *Inum bs* (*Sub a b*) = *Inum bs a* − *Inum bs b*
| *Inum bs* (*Mul c a*) = (*real c*) ∗ *Inum bs a*
| *Inum bs* (*Floor a*) = *real* (*floor* (*Inum bs a*))
| *Inum bs* (*CF c a b*) = *real c* ∗ *real* (*floor* (*Inum bs a*)) + *Inum bs b*
definition *isint t bs* ≡ *real* (*floor* (*Inum bs t*)) = *Inum bs t*

lemma *isint-iff*: *isint n bs* = (*real* (*floor* (*Inum bs n*)) = *Inum bs n*)
by (*simp add*: *isint-def*)

lemma *isint-Floor*: *isint* (*Floor n*) *bs*
by (*simp add*: *isint-iff*)

lemma *isint-Mul*: *isint e bs* ⟹ *isint* (*Mul c e*) *bs*
proof−
let *?e* = *Inum bs e*
let *?fe* = *floor ?e*
assume *be*: *isint e bs* **hence** *efe*:*real ?fe* = *?e* **by** (*simp add*: *isint-iff*)
have *real* ((*floor* (*Inum bs* (*Mul c e*)))) = *real* (*floor* (*real* (*c* ∗ *?fe*))) **using** *efe* **by** *simp*
also have ... = *real* (*c*∗ *?fe*) **by** (*simp only*: *floor-real-of-int*)
also have ... = *real c* ∗ *?e* **using** *efe* **by** *simp*
finally show *?thesis* **using** *isint-iff* **by** *simp*
qed

lemma *isint-neg*: *isint e bs* $\Longrightarrow$ *isint* (*Neg e*) *bs*
proof−
  let *?I* = λ *t*. *Inum bs t*
  assume *ie*: *isint e bs*
  hence *th*: *real* (*floor* (*?I e*)) = *?I e* **by** (*simp add*: *isint-def*)
  have *real* (*floor* (*?I* (*Neg e*))) = *real* (*floor* (− (*real* (*floor* (*?I e*))))) **by** (*simp add*: *th*)
  also have ... = − *real* (*floor* (*?I e*)) **by**(*simp add*: *floor-minus-real-of-int*)
  finally show *isint* (*Neg e*) *bs* **by** (*simp add*: *isint-def th*)
qed

lemma *isint-sub*:
  assumes *ie*: *isint e bs* **shows** *isint* (*Sub* (*C c*) *e*) *bs*
proof−
  let *?I* = λ *t*. *Inum bs t*
  from *ie* **have** *th*: *real* (*floor* (*?I e*)) = *?I e* **by** (*simp add*: *isint-def*)
  have *real* (*floor* (*?I* (*Sub* (*C c*) *e*))) = *real* (*floor* ((*real* (*c* −*floor* (*?I e*))))) **by** (*simp add*: *th*)
  also have ... = *real* (*c*− *floor* (*?I e*)) **by**(*simp add*: *floor-minus-real-of-int*)
  finally show *isint* (*Sub* (*C c*) *e*) *bs* **by** (*simp add*: *isint-def th*)
qed

lemma *isint-add*: **assumes**
  ai:*isint a bs* **and** *bi*: *isint b bs* **shows** *isint* (*Add a b*) *bs*
proof−
  let *?a* = *Inum bs a*
  let *?b* = *Inum bs b*
  from *ai bi isint-iff* **have** *real* (*floor* (*?a* + *?b*)) = *real* (*floor* (*real* (*floor ?a*) + *real* (*floor ?b*))) **by** *simp*
  also have ... = *real* (*floor ?a*) + *real* (*floor ?b*) **by** *simp*
  also have ... = *?a* + *?b* **using** *ai bi isint-iff* **by** *simp*
  finally show *isint* (*Add a b*) *bs* **by** (*simp add*: *isint-iff*)
qed

lemma *isint-c*: *isint* (*C j*) *bs*
  by (*simp add*: *isint-iff*)

datatype *fm* =
  T| *F*| *Lt num*| *Le num*| *Gt num*| *Ge num*| *Eq num*| *NEq num*| *Dvd int num*| *NDvd int num*|
  NOT fm| *And fm fm*| *Or fm fm*| *Imp fm fm*| *Iff fm fm*| *E fm*| *A fm*

fun *fmsize* :: *fm* $\Rightarrow$ *nat* **where**
 fmsize (*NOT p*) = *1* + *fmsize p*

| *fmsize* (*And p q*) = *1* + *fmsize p* + *fmsize q*
| *fmsize* (*Or p q*) = *1* + *fmsize p* + *fmsize q*
| *fmsize* (*Imp p q*) = *3* + *fmsize p* + *fmsize q*
| *fmsize* (*Iff p q*) = *3* + *2*∗(*fmsize p* + *fmsize q*)
| *fmsize* (*E p*) = *1* + *fmsize p*
| *fmsize* (*A p*) = *4*+ *fmsize p*
| *fmsize* (*Dvd i t*) = *2*
| *fmsize* (*NDvd i t*) = *2*
| *fmsize p* = *1*

lemma *fmsize-pos*: *fmsize p* > *0*
by (*induct p rule*: *fmsize.induct*) *simp-all*

fun *Ifm* ::*real list* ⇒ *fm* ⇒ *bool* **where**
Ifm bs T = *True*
| *Ifm bs F* = *False*
| *Ifm bs* (*Lt a*) = (*Inum bs a* < *0*)
| *Ifm bs* (*Gt a*) = (*Inum bs a* > *0*)
| *Ifm bs* (*Le a*) = (*Inum bs a* ≤ *0*)
| *Ifm bs* (*Ge a*) = (*Inum bs a* ≥ *0*)
| *Ifm bs* (*Eq a*) = (*Inum bs a* = *0*)
| *Ifm bs* (*NEq a*) = (*Inum bs a* ≠ *0*)
| *Ifm bs* (*Dvd i b*) = (*real i rdvd Inum bs b*)
| *Ifm bs* (*NDvd i b*) = (¬(*real i rdvd Inum bs b*))
| *Ifm bs* (*NOT p*) = (¬ (*Ifm bs p*))
| *Ifm bs* (*And p q*) = (*Ifm bs p* ∧ *Ifm bs q*)
| *Ifm bs* (*Or p q*) = (*Ifm bs p* ∨ *Ifm bs q*)
| *Ifm bs* (*Imp p q*) = ((*Ifm bs p*) ⟶ (*Ifm bs q*))
| *Ifm bs* (*Iff p q*) = (*Ifm bs p* = *Ifm bs q*)
| *Ifm bs* (*E p*) = (∃ *x*. *Ifm* (*x*#*bs*) *p*)
| *Ifm bs* (*A p*) = (∀ *x*. *Ifm* (*x*#*bs*) *p*)

consts *prep* :: *fm* ⇒ *fm*
recdef *prep measure fmsize*
prep (*E T*) = *T*
prep (*E F*) = *F*
prep (*E* (*Or p q*)) = *Or* (*prep* (*E p*)) (*prep* (*E q*))
prep (*E* (*Imp p q*)) = *Or* (*prep* (*E* (*NOT p*))) (*prep* (*E q*))
prep (*E* (*Iff p q*)) = *Or* (*prep* (*E* (*And p q*))) (*prep* (*E* (*And* (*NOT p*) (*NOT q*))))
prep (*E* (*NOT* (*And p q*))) = *Or* (*prep* (*E* (*NOT p*))) (*prep* (*E*(*NOT q*)))
prep (*E* (*NOT* (*Imp p q*))) = *prep* (*E* (*And p* (*NOT q*)))
prep (*E* (*NOT* (*Iff p q*))) = *Or* (*prep* (*E* (*And p* (*NOT q*)))) (*prep* (*E*(*And* (*NOT p*) *q*)))
prep (*E p*) = *E* (*prep p*)
prep (*A* (*And p q*)) = *And* (*prep* (*A p*)) (*prep* (*A q*))
prep (*A p*) = *prep* (*NOT* (*E* (*NOT p*)))
prep (*NOT* (*NOT p*)) = *prep p*

prep (*NOT* (*And p q*)) = *Or* (*prep* (*NOT p*)) (*prep* (*NOT q*))
prep (*NOT* (*A p*)) = *prep* (*E* (*NOT p*))
prep (*NOT* (*Or p q*)) = *And* (*prep* (*NOT p*)) (*prep* (*NOT q*))
prep (*NOT* (*Imp p q*)) = *And* (*prep p*) (*prep* (*NOT q*))
prep (*NOT* (*Iff p q*)) = *Or* (*prep* (*And p* (*NOT q*))) (*prep* (*And* (*NOT p*) *q*))
prep (*NOT p*) = *NOT* (*prep p*)
prep (*Or p q*) = *Or* (*prep p*) (*prep q*)
prep (*And p q*) = *And* (*prep p*) (*prep q*)
prep (*Imp p q*) = *prep* (*Or* (*NOT p*) *q*)
prep (*Iff p q*) = *Or* (*prep* (*And p q*)) (*prep* (*And* (*NOT p*) (*NOT q*)))
prep p = *p*
(**hints** *simp add*: *fmsize-pos*)
lemma *prep*: $\bigwedge$ *bs*. *Ifm bs* (*prep p*) = *Ifm bs p*
by (*induct p rule*: *prep.induct*, *auto*)

consts *qfree*:: *fm* ⇒ *bool*
recdef *qfree measure size*
qfree (*E p*) = *False*
qfree (*A p*) = *False*
qfree (*NOT p*) = *qfree p*
qfree (*And p q*) = (*qfree p* ∧ *qfree q*)
qfree (*Or p q*) = (*qfree p* ∧ *qfree q*)
qfree (*Imp p q*) = (*qfree p* ∧ *qfree q*)
qfree (*Iff p q*) = (*qfree p* ∧ *qfree q*)
qfree p = *True*

consts
numbound0:: *num* ⇒ *bool*
bound0:: *fm* ⇒ *bool*
numsubst0:: *num* ⇒ *num* ⇒ *num*
subst0:: *num* ⇒ *fm* ⇒ *fm*
primrec
numbound0 (*C c*) = *True*
numbound0 (*Bound n*) = (*n*>*0*)
numbound0 (*CN n i a*) = (*n* > *0* ∧ *numbound0 a*)
numbound0 (*Neg a*) = *numbound0 a*
numbound0 (*Add a b*) = (*numbound0 a* ∧ *numbound0 b*)
numbound0 (*Sub a b*) = (*numbound0 a* ∧ *numbound0 b*)
numbound0 (*Mul i a*) = *numbound0 a*
numbound0 (*Floor a*) = *numbound0 a*
numbound0 (*CF c a b*) = (*numbound0 a* ∧ *numbound0 b*)
lemma *numbound0-I*:
assumes *nb*: *numbound0 a*
shows *Inum* (*b*#*bs*) *a* = *Inum* (*b'*#*bs*) *a*
using *nb*
by (*induct a rule*: *numbound0.induct*) (*auto simp add*: *nth-pos2*)

lemma *numbound0-gen*:
  assumes *nb*: *numbound0 t* **and** *ti*: *isint t* $(x\#bs)$
  shows $\forall$ *y*. *isint t* $(y\#bs)$
using *nb ti*
proof(*clarify*)
  fix *y*
  from *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*y* **and** *b'*=*x*] *ti*[*simplified isint-def*]
  show *isint t* $(y\#bs)$
    by (*simp add*: *isint-def*)
qed

primrec
  bound0 T = *True*
  bound0 F = *True*
  bound0 (*Lt a*) = *numbound0 a*
  bound0 (*Le a*) = *numbound0 a*
  bound0 (*Gt a*) = *numbound0 a*
  bound0 (*Ge a*) = *numbound0 a*
  bound0 (*Eq a*) = *numbound0 a*
  bound0 (*NEq a*) = *numbound0 a*
  bound0 (*Dvd i a*) = *numbound0 a*
  bound0 (*NDvd i a*) = *numbound0 a*
  bound0 (*NOT p*) = *bound0 p*
  bound0 (*And p q*) = (*bound0 p* $\wedge$ *bound0 q*)
  bound0 (*Or p q*) = (*bound0 p* $\wedge$ *bound0 q*)
  bound0 (*Imp p q*) = ((*bound0 p*) $\wedge$ (*bound0 q*))
  bound0 (*Iff p q*) = (*bound0 p* $\wedge$ *bound0 q*)
  bound0 (*E p*) = *False*
  bound0 (*A p*) = *False*

lemma *bound0-I*:
  assumes *bp*: *bound0 p*
  shows *Ifm* $(b\#bs)$ *p* = *Ifm* $(b'\#bs)$ *p*
using *bp numbound0-I*[**where** *b*=*b* **and** *bs*=*bs* **and** *b'*=*b'*]
by (*induct p rule*: *bound0.induct*) (*auto simp add*: *nth-pos2*)

primrec
  numsubst0 t (*C c*) = (*C c*)
  numsubst0 t (*Bound n*) = (*if n*=*0 then t else Bound n*)
  numsubst0 t (*CN n i a*) = (*if n*=*0 then Add* (*Mul i t*) (*numsubst0 t a*) *else CN n i* (*numsubst0 t a*))
  numsubst0 t (*CF i a b*) = *CF i* (*numsubst0 t a*) (*numsubst0 t b*)
  numsubst0 t (*Neg a*) = *Neg* (*numsubst0 t a*)
  numsubst0 t (*Add a b*) = *Add* (*numsubst0 t a*) (*numsubst0 t b*)
  numsubst0 t (*Sub a b*) = *Sub* (*numsubst0 t a*) (*numsubst0 t b*)
  numsubst0 t (*Mul i a*) = *Mul i* (*numsubst0 t a*)

numsubst0 t (*Floor a*) = *Floor* (*numsubst0 t a*)

lemma *numsubst0-I*:
shows *Inum* (*b*#*bs*) (*numsubst0 a t*) = *Inum* ((*Inum* (*b*#*bs*) *a*)#*bs*) *t*
by (*induct t*) (*simp-all add*: *nth-pos2*)

lemma *numsubst0-I'*:
assumes *nb*: *numbound0 a*
shows *Inum* (*b*#*bs*) (*numsubst0 a t*) = *Inum* ((*Inum* (*b'*#*bs*) *a*)#*bs*) *t*
by (*induct t*) (*simp-all add*: *nth-pos2 numbound0-I*[*OF nb*, **where** *b*=*b* **and** *b'*=*b'*])

primrec
subst0 t T = *T*
subst0 t F = *F*
subst0 t (*Lt a*) = *Lt* (*numsubst0 t a*)
subst0 t (*Le a*) = *Le* (*numsubst0 t a*)
subst0 t (*Gt a*) = *Gt* (*numsubst0 t a*)
subst0 t (*Ge a*) = *Ge* (*numsubst0 t a*)
subst0 t (*Eq a*) = *Eq* (*numsubst0 t a*)
subst0 t (*NEq a*) = *NEq* (*numsubst0 t a*)
subst0 t (*Dvd i a*) = *Dvd i* (*numsubst0 t a*)
subst0 t (*NDvd i a*) = *NDvd i* (*numsubst0 t a*)
subst0 t (*NOT p*) = *NOT* (*subst0 t p*)
subst0 t (*And p q*) = *And* (*subst0 t p*) (*subst0 t q*)
subst0 t (*Or p q*) = *Or* (*subst0 t p*) (*subst0 t q*)
subst0 t (*Imp p q*) = *Imp* (*subst0 t p*) (*subst0 t q*)
subst0 t (*Iff p q*) = *Iff* (*subst0 t p*) (*subst0 t q*)

lemma *subst0-I*: **assumes** *qfp*: *qfree p*
shows *Ifm* (*b*#*bs*) (*subst0 a p*) = *Ifm* ((*Inum* (*b*#*bs*) *a*)#*bs*) *p*
using *qfp numsubst0-I*[**where** *b*=*b* **and** *bs*=*bs* **and** *a*=*a*]
by (*induct p*) (*simp-all add*: *nth-pos2*)

consts
decrnum:: *num* ⇒ *num*
decr :: *fm* ⇒ *fm*

recdef *decrnum measure size*
decrnum (*Bound n*) = *Bound* (*n* − *1*)
decrnum (*Neg a*) = *Neg* (*decrnum a*)
decrnum (*Add a b*) = *Add* (*decrnum a*) (*decrnum b*)
decrnum (*Sub a b*) = *Sub* (*decrnum a*) (*decrnum b*)
decrnum (*Mul c a*) = *Mul c* (*decrnum a*)
decrnum (*Floor a*) = *Floor* (*decrnum a*)
decrnum (*CN n c a*) = *CN* (*n* − *1*) *c* (*decrnum a*)
decrnum (*CF c a b*) = *CF c* (*decrnum a*) (*decrnum b*)
decrnum a = *a*

recdef *decr measure size*
  decr (*Lt a*) = *Lt* (*decrnum a*)
  decr (*Le a*) = *Le* (*decrnum a*)
  decr (*Gt a*) = *Gt* (*decrnum a*)
  decr (*Ge a*) = *Ge* (*decrnum a*)
  decr (*Eq a*) = *Eq* (*decrnum a*)
  decr (*NEq a*) = *NEq* (*decrnum a*)
  decr (*Dvd i a*) = *Dvd i* (*decrnum a*)
  decr (*NDvd i a*) = *NDvd i* (*decrnum a*)
  decr (*NOT p*) = *NOT* (*decr p*)
  decr (*And p q*) = *And* (*decr p*) (*decr q*)
  decr (*Or p q*) = *Or* (*decr p*) (*decr q*)
  decr (*Imp p q*) = *Imp* (*decr p*) (*decr q*)
  decr (*Iff p q*) = *Iff* (*decr p*) (*decr q*)
  decr p = *p*

lemma *decrnum*: **assumes** *nb*: *numbound0 t*
  shows *Inum* (*x#bs*) *t* = *Inum bs* (*decrnum t*)
  using *nb* **by** (*induct t rule*: *decrnum.induct*, *simp-all add*: *nth-pos2*)

lemma *decr*: **assumes** *nb*: *bound0 p*
  shows *Ifm* (*x#bs*) *p* = *Ifm bs* (*decr p*)
  using *nb*
  by (*induct p rule*: *decr.induct*, *simp-all add*: *nth-pos2 decrnum*)

lemma *decr-qf*: *bound0 p* ⟹ *qfree* (*decr p*)
by (*induct p*, *simp-all*)

consts
  isatom :: *fm* ⇒ *bool*
recdef *isatom measure size*
  isatom T = *True*
  isatom F = *True*
  isatom (*Lt a*) = *True*
  isatom (*Le a*) = *True*
  isatom (*Gt a*) = *True*
  isatom (*Ge a*) = *True*
  isatom (*Eq a*) = *True*
  isatom (*NEq a*) = *True*
  isatom (*Dvd i b*) = *True*
  isatom (*NDvd i b*) = *True*
  isatom p = *False*

lemma *numsubst0-numbound0*: **assumes** *nb*: *numbound0 t*
  shows *numbound0* (*numsubst0 t a*)
using *nb* **by** (*induct a rule*: *numsubst0.induct*, *auto*)

lemma *subst0-bound0*: **assumes** *qf*: *qfree p* **and** *nb*: *numbound0 t*

shows *bound0* (*subst0 t p*)
using *qf numsubst0-numbound0*[*OF nb*] **by** (*induct p rule*: *subst0.induct*, *auto*)

lemma *bound0-qf*: *bound0 p* $\Longrightarrow$ *qfree p*
by (*induct p*, *simp-all*)

constdefs *djf* :: ($'a \Rightarrow fm$) $\Rightarrow$ $'a \Rightarrow fm \Rightarrow fm$
djf f p q $\equiv$ (*if q*=*T then T else if q*=*F then f p else*
(*let fp* = *f p in case fp of T* $\Rightarrow$ *T* | *F* $\Rightarrow$ *q* | - $\Rightarrow$ *Or fp q*))
constdefs *evaldjf* :: ($'a \Rightarrow fm$) $\Rightarrow$ $'a$ *list* $\Rightarrow fm$
evaldjf f ps $\equiv$ *foldr* (*djf f*) *ps F*

lemma *djf-Or*: *Ifm bs* (*djf f p q*) = *Ifm bs* (*Or* (*f p*) *q*)
by (*cases q*=*T*, *simp add*: *djf-def*,*cases q*=*F*,*simp add*: *djf-def*)
(*cases f p*, *simp-all add*: *Let-def djf-def*)

lemma *evaldjf-ex*: *Ifm bs* (*evaldjf f ps*) = ($\exists$ *p* $\in$ *set ps*. *Ifm bs* (*f p*))
by(*induct ps*, *simp-all add*: *evaldjf-def djf-Or*)

lemma *evaldjf-bound0*:
assumes *nb*: $\forall$ *x*$\in$ *set xs*. *bound0* (*f x*)
shows *bound0* (*evaldjf f xs*)
using *nb* **by** (*induct xs*, *auto simp add*: *evaldjf-def djf-def Let-def*) (*case-tac f a*, *auto*)

lemma *evaldjf-qf*:
assumes *nb*: $\forall$ *x*$\in$ *set xs*. *qfree* (*f x*)
shows *qfree* (*evaldjf f xs*)
using *nb* **by** (*induct xs*, *auto simp add*: *evaldjf-def djf-def Let-def*) (*case-tac f a*, *auto*)

consts
disjuncts :: *fm* $\Rightarrow$ *fm list*
conjuncts :: *fm* $\Rightarrow$ *fm list*
recdef *disjuncts measure size*
disjuncts (*Or p q*) = (*disjuncts p*) @ (*disjuncts q*)
disjuncts F = []
disjuncts p = [*p*]

recdef *conjuncts measure size*
conjuncts (*And p q*) = (*conjuncts p*) @ (*conjuncts q*)
conjuncts T = []
conjuncts p = [*p*]
lemma *disjuncts*: ($\exists$ *q*$\in$ *set* (*disjuncts p*). *Ifm bs q*) = *Ifm bs p*
by(*induct p rule*: *disjuncts.induct*, *auto*)
lemma *conjuncts*: ($\forall$ *q*$\in$ *set* (*conjuncts p*). *Ifm bs q*) = *Ifm bs p*
by(*induct p rule*: *conjuncts.induct*, *auto*)

lemma *disjuncts-nb*: *bound0 p* $\Longrightarrow \forall\ q\in$ *set* (*disjuncts p*). *bound0 q*
proof−
  assume *nb*: *bound0 p*
  hence *list-all bound0* (*disjuncts p*) **by** (*induct p rule*:*disjuncts.induct*,*auto*)
  thus *?thesis* **by** (*simp only*: *list-all-iff*)
qed
lemma *conjuncts-nb*: *bound0 p* $\Longrightarrow \forall\ q\in$ *set* (*conjuncts p*). *bound0 q*
proof−
  assume *nb*: *bound0 p*
  hence *list-all bound0* (*conjuncts p*) **by** (*induct p rule*:*conjuncts.induct*,*auto*)
  thus *?thesis* **by** (*simp only*: *list-all-iff*)
qed

lemma *disjuncts-qf*: *qfree p* $\Longrightarrow \forall\ q\in$ *set* (*disjuncts p*). *qfree q*
proof−
  assume *qf*: *qfree p*
  hence *list-all qfree* (*disjuncts p*)
    by (*induct p rule*: *disjuncts.induct*, *auto*)
  thus *?thesis* **by** (*simp only*: *list-all-iff*)
qed
lemma *conjuncts-qf*: *qfree p* $\Longrightarrow \forall\ q\in$ *set* (*conjuncts p*). *qfree q*
proof−
  assume *qf*: *qfree p*
  hence *list-all qfree* (*conjuncts p*)
    by (*induct p rule*: *conjuncts.induct*, *auto*)
  thus *?thesis* **by** (*simp only*: *list-all-iff*)
qed

constdefs *DJ* :: (*fm* $\Rightarrow$ *fm*) $\Rightarrow$ *fm* $\Rightarrow$ *fm*
  DJ f p $\equiv$ *evaldjf f* (*disjuncts p*)

lemma *DJ*: **assumes** *fdj*: $\forall$ *p q*. *f* (*Or p q*) = *Or* (*f p*) (*f q*)
  and *fF*: *f F* = *F*
  shows *Ifm bs* (*DJ f p*) = *Ifm bs* (*f p*)
proof−
  have *Ifm bs* (*DJ f p*) = ($\exists$ *q* $\in$ *set* (*disjuncts p*). *Ifm bs* (*f q*))
    by (*simp add*: *DJ-def evaldjf-ex*)
  also have ... = *Ifm bs* (*f p*) **using** *fdj fF* **by** (*induct p rule*: *disjuncts.induct*, *auto*)
  finally show *?thesis* **.**
qed

lemma *DJ-qf*: **assumes**
  fqf: $\forall$ *p*. *qfree p* $\longrightarrow$ *qfree* (*f p*)
  shows $\forall$ *p*. *qfree p* $\longrightarrow$ *qfree* (*DJ f p*)
proof(*clarify*)
  fix *p* **assume** *qf*: *qfree p*
  have *th*: *DJ f p* = *evaldjf f* (*disjuncts p*) **by** (*simp add*: *DJ-def*)
  from *disjuncts-qf*[*OF qf*] **have** $\forall$ *q*$\in$ *set* (*disjuncts p*). *qfree q* **.**

with *fqf* **have** *th'*:∀ *q*∈ *set* (*disjuncts p*). *qfree* (*f q*) **by** *blast*

from *evaldjf-qf*[*OF th'*] *th* **show** *qfree* (*DJ f p*) **by** *simp*
qed

lemma *DJ-qe*: **assumes** *qe*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ∀ *bs p*. *qfree p* ⟶ *qfree* (*DJ qe p*) ∧ (*Ifm bs* ((*DJ qe p*)) = *Ifm bs* (*E p*))
proof(*clarify*)
fix *p*::*fm* **and** *bs*
assume *qf*: *qfree p*
from *qe* **have** *qth*: ∀ *p*. *qfree p* ⟶ *qfree* (*qe p*) **by** *blast*
from *DJ-qf*[*OF qth*] *qf* **have** *qfth*:*qfree* (*DJ qe p*) **by** *auto*
have *Ifm bs* (*DJ qe p*) = (∃ *q*∈ *set* (*disjuncts p*). *Ifm bs* (*qe q*))
by (*simp add*: *DJ-def evaldjf-ex*)
also have ... = (∃ *q* ∈ *set*(*disjuncts p*). *Ifm bs* (*E q*)) **using** *qe disjuncts-qf*[*OF qf*] **by** *auto*
also have ... = *Ifm bs* (*E p*) **by** (*induct p rule*: *disjuncts.induct*, *auto*)
finally show *qfree* (*DJ qe p*) ∧ *Ifm bs* (*DJ qe p*) = *Ifm bs* (*E p*) **using** *qfth* **by** *blast*
qed

consts *bnds*:: *num* ⇒ *nat list*
lex-ns:: *nat list* × *nat list* ⇒ *bool*
recdef *bnds measure size*
bnds (*Bound n*) = [*n*]
bnds (*CN n c a*) = *n*#(*bnds a*)
bnds (*Neg a*) = *bnds a*
bnds (*Add a b*) = (*bnds a*)@(*bnds b*)
bnds (*Sub a b*) = (*bnds a*)@(*bnds b*)
bnds (*Mul i a*) = *bnds a*
bnds (*Floor a*) = *bnds a*
bnds (*CF c a b*) = (*bnds a*)@(*bnds b*)
bnds a = []
recdef *lex-ns measure* (λ (*xs*,*ys*). *length xs* + *length ys*)
lex-ns ([], *ms*) = *True*
lex-ns (*ns*, []) = *False*
lex-ns (*n*#*ns*, *m*#*ms*) = (*n*<*m* ∨ ((*n* = *m*) ∧ *lex-ns* (*ns*,*ms*)))
constdefs *lex-bnd* :: *num* ⇒ *num* ⇒ *bool*
lex-bnd t s ≡ *lex-ns* (*bnds t*, *bnds s*)

consts
numgcdh:: *num* ⇒ *int* ⇒ *int*
reducecoeffh:: *num* ⇒ *int* ⇒ *num*
dvdnumcoeff:: *num* ⇒ *int* ⇒ *bool*
consts *maxcoeff*:: *num* ⇒ *int*

recdef *maxcoeff measure size*
  maxcoeff (*C i*) = *abs i*
  maxcoeff (*CN n c t*) = *max* (*abs c*) (*maxcoeff t*)
  maxcoeff (*CF c t s*) = *max* (*abs c*) (*maxcoeff s*)
  maxcoeff t = *1*

lemma *maxcoeff-pos*: *maxcoeff t* $\geq$ *0*
  apply (*induct t rule*: *maxcoeff.induct*, *auto*)
  done

recdef *numgcdh measure size*
  numgcdh (*C i*) = (λ*g. igcd i g*)
  numgcdh (*CN n c t*) = (λ*g. igcd c* (*numgcdh t g*))
  numgcdh (*CF c s t*) = (λ*g. igcd c* (*numgcdh t g*))
  numgcdh t = (λ*g. 1*)

definition
  numgcd :: *num* $\Rightarrow$ *int*
where
  numgcd-def: *numgcd t* = *numgcdh t* (*maxcoeff t*)

recdef *reducecoeffh measure size*
  reducecoeffh (*C i*) = (λ *g. C* (*i div g*))
  reducecoeffh (*CN n c t*) = (λ *g. CN n* (*c div g*) (*reducecoeffh t g*))
  reducecoeffh (*CF c s t*) = (λ *g. CF* (*c div g*) *s* (*reducecoeffh t g*))
  reducecoeffh t = (λ*g. t*)

definition
  reducecoeff :: *num* $\Rightarrow$ *num*
where
  reducecoeff-def: *reducecoeff t* =
  (*let g* = *numgcd t in*
  if g = *0 then C 0 else if g*=*1 then t else reducecoeffh t g*)

recdef *dvdnumcoeff measure size*
  dvdnumcoeff (*C i*) = (λ *g. g dvd i*)
  dvdnumcoeff (*CN n c t*) = (λ *g. g dvd c* $\wedge$ (*dvdnumcoeff t g*))
  dvdnumcoeff (*CF c s t*) = (λ *g. g dvd c* $\wedge$ (*dvdnumcoeff t g*))
  dvdnumcoeff t = (λ*g. False*)

lemma *dvdnumcoeff-trans*:
  assumes *gdg*: *g dvd g'* **and** *dgt'*:*dvdnumcoeff t g'*
  shows *dvdnumcoeff t g*
  using *dgt' gdg*
  by (*induct t rule*: *dvdnumcoeff.induct*, *simp-all add*: *gdg zdvd-trans*[*OF gdg*])

declare *zdvd-trans* [*trans add*]

lemma *natabs0*: (*nat* (*abs x*) = *0*) = (*x* = *0*)

by *arith*

lemma *numgcd0*:
  assumes *g0*: *numgcd t = 0*
  shows *Inum bs t = 0*
proof−
  have $\bigwedge x$. *numgcdh t x= 0* $\Longrightarrow$ *Inum bs t = 0*
    by (*induct t rule*: *numgcdh.induct*, *auto simp add*: *igcd-def gcd-zero natabs0 max-def maxcoeff-pos*)
  thus *?thesis* **using** *g0*[*simplified numgcd-def*] **by** *blast*
qed

lemma *numgcdh-pos*: **assumes** *gp*: $g \geq 0$ **shows** *numgcdh t g* ≥ 0
  using *gp*
  by (*induct t rule*: *numgcdh.induct*, *auto simp add*: *igcd-def*)

lemma *numgcd-pos*: *numgcd t* $\geq$*0*
  by (*simp add*: *numgcd-def numgcdh-pos maxcoeff-pos*)

lemma *reducecoeffh*:
  assumes *gt*: *dvdnumcoeff t g* **and** *gp*: $g > 0$
  shows *real g* ∗(*Inum bs* (*reducecoeffh t g*)) = *Inum bs t*
  using *gt*
proof(*induct t rule*: *reducecoeffh.induct*)
  case (*1 i*) **hence** *gd*: *g dvd i* **by** *simp*
  from *gp* **have** *gnz*: $g \neq 0$ **by** *simp*
  from *prems* **show** *?case* **by** (*simp add*: *real-of-int-div*[*OF gnz gd*])
next
  case (*2 n c t*) **hence** *gd*: *g dvd c* **by** *simp*
  from *gp* **have** *gnz*: $g \neq 0$ **by** *simp*
  from *prems* **show** *?case* **by** (*simp add*: *real-of-int-div*[*OF gnz gd*] *ring-simps*)
next
  case (*3 c s t*) **hence** *gd*: *g dvd c* **by** *simp*
  from *gp* **have** *gnz*: $g \neq 0$ **by** *simp*
  from *prems* **show** *?case* **by** (*simp add*: *real-of-int-div*[*OF gnz gd*] *ring-simps*)
qed (*auto simp add*: *numgcd-def gp*)
consts *ismaxcoeff*:: *num* ⇒ *int* ⇒ *bool*
recdef *ismaxcoeff measure size*
  ismaxcoeff (*C i*) = (λ *x*. *abs i* ≤ *x*)
  ismaxcoeff (*CN n c t*) = (λ*x*. *abs c* ≤ *x* ∧ (*ismaxcoeff t x*))
  ismaxcoeff (*CF c s t*) = (λ*x*. *abs c* ≤ *x* ∧ (*ismaxcoeff t x*))
  ismaxcoeff t = (λ*x*. *True*)

lemma *ismaxcoeff-mono*: *ismaxcoeff t c* $\Longrightarrow$ $c \leq c'$ $\Longrightarrow$ *ismaxcoeff t c'*
by (*induct t rule*: *ismaxcoeff.induct*, *auto*)

lemma *maxcoeff-ismaxcoeff*: *ismaxcoeff t* (*maxcoeff t*)
proof (*induct t rule*: *maxcoeff.induct*)
  case (*2 n c t*)

```
  hence H:ismaxcoeff t (maxcoeff t) .
  have thh: maxcoeff t ≤ max (abs c) (maxcoeff t) by (simp add: le-maxI2)
  from ismaxcoeff-mono[OF H thh] show ?case by (simp add: le-maxI1)
next
  case (3 c t s)
  hence H1:ismaxcoeff s (maxcoeff s) by auto
  have thh1: maxcoeff s ≤ max |c| (maxcoeff s) by (simp add: max-def)
  from ismaxcoeff-mono[OF H1 thh1] show ?case by (simp add: le-maxI1)
qed simp-all

lemma igcd-gt1: igcd i j > 1 ⟹ ((abs i > 1 ∧ abs j > 1) ∨ (abs i = 0 ∧ abs j
> 1) ∨ (abs i > 1 ∧ abs j = 0))
  apply (unfold igcd-def)
  apply (cases i = 0, simp-all)
  apply (cases j = 0, simp-all)
  apply (cases abs i = 1, simp-all)
  apply (cases abs j = 1, simp-all)
  apply auto
  done
lemma numgcdh0:numgcdh t m = 0 ⟹  m =0
  by (induct t rule: numgcdh.induct, auto simp add:igcd0)

lemma dvdnumcoeff-aux:
  assumes ismaxcoeff t m and mp:m ≥ 0 and numgcdh t m > 1
  shows dvdnumcoeff t (numgcdh t m)
using prems
proof(induct t rule: numgcdh.induct)
  case (2 n c t)
  let ?g = numgcdh t m
  from prems have th:igcd c ?g > 1 by simp
  from igcd-gt1[OF th] numgcdh-pos[OF mp, where t=t]
  have (abs c > 1 ∧ ?g > 1) ∨ (abs c = 0 ∧ ?g > 1) ∨ (abs c > 1 ∧ ?g = 0)
by simp
  moreover {assume abs c > 1 and gp: ?g > 1 with prems
    have th: dvdnumcoeff t ?g by simp
    have th': igcd c ?g dvd ?g by (simp add:igcd-dvd2)
    from dvdnumcoeff-trans[OF th' th] have ?case by (simp add: igcd-dvd1)}
  moreover {assume abs c = 0 ∧ ?g > 1
    with prems have th: dvdnumcoeff t ?g by simp
    have th': igcd c ?g dvd ?g by (simp add:igcd-dvd2)
    from dvdnumcoeff-trans[OF th' th] have ?case by (simp add: igcd-dvd1)
    hence ?case by simp }
  moreover {assume abs c > 1 and g0:?g = 0
    from numgcdh0[OF g0] have m=0. with prems   have ?case by simp }
  ultimately show ?case by blast
next
  case (3 c s t)
  let ?g = numgcdh t m
  from prems have th:igcd c ?g > 1 by simp
```

from *igcd-gt1*[*OF th*] *numgcdh-pos*[*OF mp*, **where** *t*=*t*]
have (*abs c* > *1* ∧ *?g* > *1*) ∨ (*abs c* = *0* ∧ *?g* > *1*) ∨ (*abs c* > *1* ∧ *?g* = *0*)
by *simp*
moreover {**assume** *abs c* > *1* **and** *gp*: *?g* > *1* **with** *prems*
have *th*: *dvdnumcoeff t ?g* **by** *simp*
have *th'*: *igcd c ?g dvd ?g* **by** (*simp add*:*igcd-dvd2*)
from *dvdnumcoeff-trans*[*OF th' th*] **have** *?case* **by** (*simp add*: *igcd-dvd1*)}
moreover {**assume** *abs c* = *0* ∧ *?g* > *1*
with *prems* **have** *th*: *dvdnumcoeff t ?g* **by** *simp*
have *th'*: *igcd c ?g dvd ?g* **by** (*simp add*:*igcd-dvd2*)
from *dvdnumcoeff-trans*[*OF th' th*] **have** *?case* **by** (*simp add*: *igcd-dvd1*)
hence *?case* **by** *simp* }
moreover {**assume** *abs c* > *1* **and** *g0*:*?g* = *0*
from *numgcdh0*[*OF g0*] **have** *m*=*0*. **with** *prems* **have** *?case* **by** *simp* }
ultimately show *?case* **by** *blast*
qed(*auto simp add*: *igcd-dvd1*)

lemma *dvdnumcoeff-aux2*:
assumes *numgcd t* > *1* **shows** *dvdnumcoeff t* (*numgcd t*) ∧ *numgcd t* > *0*
using *prems*
proof (*simp add*: *numgcd-def*)
let *?mc* = *maxcoeff t*
let *?g* = *numgcdh t ?mc*
have *th1*: *ismaxcoeff t ?mc* **by** (*rule maxcoeff-ismaxcoeff*)
have *th2*: *?mc* ≥ *0* **by** (*rule maxcoeff-pos*)
assume *H*: *numgcdh t ?mc* > *1*
from *dvdnumcoeff-aux*[*OF th1 th2 H*] **show** *dvdnumcoeff t ?g* **.**
qed

lemma *reducecoeff*: *real* (*numgcd t*) ∗ (*Inum bs* (*reducecoeff t*)) = *Inum bs t*
proof−
let *?g* = *numgcd t*
have *?g* ≥ *0* **by** (*simp add*: *numgcd-pos*)
hence *?g* = *0* ∨ *?g* = *1* ∨ *?g* > *1* **by** *auto*
moreover {**assume** *?g* = *0* **hence** *?thesis* **by** (*simp add*: *numgcd0*)}
moreover {**assume** *?g* = *1* **hence** *?thesis* **by** (*simp add*: *reducecoeff-def*)}
moreover { **assume** *g1*:*?g* > *1*
from *dvdnumcoeff-aux2*[*OF g1*] **have** *th1*:*dvdnumcoeff t ?g* **and** *g0*: *?g* > *0* **by** *blast*+
from *reducecoeffh*[*OF th1 g0*, **where** *bs*=*bs*] *g1* **have** *?thesis*
by (*simp add*: *reducecoeff-def Let-def*)}
ultimately show *?thesis* **by** *blast*
qed

lemma *reducecoeffh-numbound0*: *numbound0 t* ⟹ *numbound0* (*reducecoeffh t g*)
by (*induct t rule*: *reducecoeffh.induct*, *auto*)

lemma *reducecoeff-numbound0*: *numbound0 t* ⟹ *numbound0* (*reducecoeff t*)
using *reducecoeffh-numbound0* **by** (*simp add*: *reducecoeff-def Let-def*)

consts
  simpnum:: *num* ⇒ *num*
  numadd:: *num* × *num* ⇒ *num*
  nummul:: *num* ⇒ *int* ⇒ *num*

recdef *numadd measure* (λ (*t*,*s*). *size t* + *size s*)
  numadd (*CN n1 c1 r1*,*CN n2 c2 r2*) =
  (*if n1*=*n2 then*
  (*let c* = *c1* + *c2*
  in (*if c*=*0 then numadd*(*r1*,*r2*) *else CN n1 c* (*numadd* (*r1*,*r2*))))
  else if n1 ≤ *n2 then CN n1 c1* (*numadd* (*r1*,*CN n2 c2 r2*))
  else (*CN n2 c2* (*numadd* (*CN n1 c1 r1*,*r2*))))
  numadd (*CN n1 c1 r1*,*t*) = *CN n1 c1* (*numadd* (*r1*, *t*))
  numadd (*t*,*CN n2 c2 r2*) = *CN n2 c2* (*numadd* (*t*,*r2*))
  numadd (*CF c1 t1 r1*,*CF c2 t2 r2*) =
   (*if t1* = *t2 then*
    (*let c*=*c1*+*c2*; *s*= *numadd*(*r1*,*r2*) *in* (*if c*=*0 then s else CF c t1 s*))
   else if lex-bnd t1 t2 then CF c1 t1 (*numadd*(*r1*,*CF c2 t2 r2*))
   else CF c2 t2 (*numadd*(*CF c1 t1 r1*,*r2*)))
  numadd (*CF c1 t1 r1*,*C c*) = *CF c1 t1* (*numadd* (*r1*, *C c*))
  numadd (*C c*,*CF c1 t1 r1*) = *CF c1 t1* (*numadd* (*r1*, *C c*))
  numadd (*C b1*, *C b2*) = *C* (*b1*+*b2*)
  numadd (*a*,*b*) = *Add a b*

lemma *numadd*[*simp*]: *Inum bs* (*numadd* (*t*,*s*)) = *Inum bs* (*Add t s*)
apply (*induct t s rule*: *numadd.induct*, *simp-all add*: *Let-def*)
 apply (*case-tac c1*+*c2* = *0*,*case-tac n1* ≤ *n2*, *simp-all*)
  apply (*case-tac n1* = *n2*, *simp-all add*: *ring-simps*)
  apply (*simp only*: *left-distrib*[*symmetric*])
 apply *simp*
apply (*case-tac lex-bnd t1 t2*, *simp-all*)
 apply (*case-tac c1*+*c2* = *0*)
  by (*case-tac t1* = *t2*, *simp-all add*: *ring-simps left-distrib*[*symmetric*] *real-of-int-mult*[*symmetric*] *real-of-int-add*[*symmetric*]*del*: *real-of-int-mult real-of-int-add left-distrib*)

lemma *numadd-nb*[*simp*]: ⟦ *numbound0 t* ; *numbound0 s*⟧ ⟹ *numbound0* (*numadd* (*t*,*s*))
by (*induct t s rule*: *numadd.induct*, *auto simp add*: *Let-def*)

recdef *nummul measure size*
  nummul (*C j*) = (λ *i*. *C* (*i*∗*j*))
  nummul (*CN n c t*) = (λ *i*. *CN n* (*c*∗*i*) (*nummul t i*))
  nummul (*CF c t s*) = (λ *i*. *CF* (*c*∗*i*) *t* (*nummul s i*))
  nummul (*Mul c t*) = (λ *i*. *nummul t* (*i*∗*c*))
  nummul t = (λ *i*. *Mul i t*)

lemma *nummul*[*simp*]: ⋀ *i*. *Inum bs* (*nummul t i*) = *Inum bs* (*Mul i t*)
by (*induct t rule*: *nummul.induct*, *auto simp add*: *ring-simps*)

lemma *nummul-nb*[*simp*]: $\bigwedge$ *i*. *numbound0 t* $\Longrightarrow$ *numbound0* (*nummul t i*)
by (*induct t rule*: *nummul.induct*, *auto*)

constdefs *numneg* :: *num* $\Rightarrow$ *num*
numneg t $\equiv$ *nummul t* (− *1*)

constdefs *numsub* :: *num* $\Rightarrow$ *num* $\Rightarrow$ *num*
numsub s t $\equiv$ (*if s* = *t then C 0 else numadd* (*s*,*numneg t*))

lemma *numneg*[*simp*]: *Inum bs* (*numneg t*) = *Inum bs* (*Neg t*)
using *numneg-def nummul* **by** *simp*

lemma *numneg-nb*[*simp*]: *numbound0 t* $\Longrightarrow$ *numbound0* (*numneg t*)
using *numneg-def* **by** *simp*

lemma *numsub*[*simp*]: *Inum bs* (*numsub a b*) = *Inum bs* (*Sub a b*)
using *numsub-def* **by** *simp*

lemma *numsub-nb*[*simp*]: $[\![$ *numbound0 t* ; *numbound0 s*$]\!]$ $\Longrightarrow$ *numbound0* (*numsub t s*)
using *numsub-def* **by** *simp*

lemma *isint-CF*: **assumes** *si*: *isint s bs* **shows** *isint* (*CF c t s*) *bs*
proof−
have *cti*: *isint* (*Mul c* (*Floor t*)) *bs* **by** (*simp add*: *isint-Mul isint-Floor*)

have *?thesis* = *isint* (*Add* (*Mul c* (*Floor t*)) *s*) *bs* **by** (*simp add*: *isint-def*)
also have ... **by** (*simp add*: *isint-add cti si*)
finally show *?thesis* **.**
qed

consts *split-int*:: *num* $\Rightarrow$ *num*×*num*
recdef *split-int measure num-size*
split-int (*C c*) = (*C 0*, *C c*)
split-int (*CN n c b*) =
(*let* (*bv*,*bi*) = *split-int b*
in (*CN n c bv*, *bi*))
split-int (*CF c a b*) =
(*let* (*bv*,*bi*) = *split-int b*
in (*bv*, *CF c a bi*))
split-int a = (*a*,*C 0*)

lemma *split-int*:$\bigwedge$ *tv ti*. *split-int t* = (*tv*,*ti*) $\Longrightarrow$ (*Inum bs* (*Add tv ti*) = *Inum bs t*) $\wedge$ *isint ti bs*
proof (*induct t rule*: *split-int.induct*)
case (*2 c n b tv ti*)
let *?bv* = *fst* (*split-int b*)
let *?bi* = *snd* (*split-int b*)

have *split-int b* = (*?bv*,*?bi*) **by** *simp*
with *prems*(*1*) **have** *b*:*Inum bs* (*Add ?bv ?bi*) = *Inum bs b* **and** *bii*: *isint ?bi bs*
by *blast+*
from *prems*(*2*) **have** *tibi*: *ti* = *?bi* **by** (*simp add*: *Let-def split-def*)
from *prems*(*2*) *b*[*symmetric*] *bii* **show** *?case* **by** (*auto simp add*: *Let-def split-def*)
next
case (*3 c a b tv ti*)
let *?bv* = *fst* (*split-int b*)
let *?bi* = *snd* (*split-int b*)
have *split-int b* = (*?bv*,*?bi*) **by** *simp*
with *prems*(*1*) **have** *b*:*Inum bs* (*Add ?bv ?bi*) = *Inum bs b* **and** *bii*: *isint ?bi bs*
by *blast+*
from *prems*(*2*) **have** *tibi*: *ti* = *CF c a ?bi* **by** (*simp add*: *Let-def split-def*)
from *prems*(*2*) *b*[*symmetric*] *bii* **show** *?case* **by** (*auto simp add*: *Let-def split-def*
isint-Floor isint-add isint-Mul isint-CF)
qed (*auto simp add*: *Let-def isint-iff isint-Floor isint-add isint-Mul split-def ring-simps*)

lemma *split-int-nb*: *numbound0 t* $\Longrightarrow$ *numbound0* (*fst* (*split-int t*)) $\wedge$ *numbound0*
(*snd* (*split-int t*))
by (*induct t rule*: *split-int.induct*, *auto simp add*: *Let-def split-def*)

definition
numfloor:: *num* $\Rightarrow$ *num*
where
numfloor-def: *numfloor t* = (*let* (*tv*,*ti*) = *split-int t in*
(*case tv of C i* $\Rightarrow$ *numadd* (*tv*,*ti*)
| - $\Rightarrow$ *numadd*(*CF 1 tv* (*C 0*),*ti*)))

lemma *numfloor*[*simp*]: *Inum bs* (*numfloor t*) = *Inum bs* (*Floor t*) (**is** *?n t* = *?N*
(*Floor t*))
proof−
let *?tv* = *fst* (*split-int t*)
let *?ti* = *snd* (*split-int t*)
have *tvti*:*split-int t* = (*?tv*,*?ti*) **by** *simp*
{assume *H*: $\forall$ *v*. *?tv* $\neq$ *C v*
hence *th1*: *?n t* = *?N* (*Add* (*Floor ?tv*) *?ti*)
by (*cases ?tv*, *auto simp add*: *numfloor-def Let-def split-def numadd*)
from *split-int*[*OF tvti*] **have** *?N* (*Floor t*) = *?N* (*Floor*(*Add ?tv ?ti*)) **and**
tii:*isint ?ti bs* **by** *simp+*
hence *?N* (*Floor t*) = *real* (*floor* (*?N* (*Add ?tv ?ti*))) **by** *simp*
also have ... = *real* (*floor* (*?N ?tv*) + (*floor* (*?N ?ti*)))
by (*simp*,*subst tii*[*simplified isint-iff*, *symmetric*]) *simp*
also have ... = *?N* (*Add* (*Floor ?tv*) *?ti*) **by** (*simp add*: *tii*[*simplified isint-iff*])
finally have *?thesis* **using** *th1* **by** *simp***}**
moreover {fix *v* **assume** *H*:*?tv* = *C v*
from *split-int*[*OF tvti*] **have** *?N* (*Floor t*) = *?N* (*Floor*(*Add ?tv ?ti*)) **and**
tii:*isint ?ti bs* **by** *simp+*
hence *?N* (*Floor t*) = *real* (*floor* (*?N* (*Add ?tv ?ti*))) **by** *simp*
also have ... = *real* (*floor* (*?N ?tv*) + (*floor* (*?N ?ti*)))

```
    by (simp,subst tii[simplified isint-iff, symmetric]) simp
   also have ... = ?N (Add (Floor ?tv) ?ti) by (simp add: tii[simplified isint-iff])
    finally have ?thesis by (simp add: H numfloor-def Let-def split-def numadd)
}
  ultimately show ?thesis by auto
qed

lemma numfloor-nb[simp]: numbound0 t ⟹ numbound0 (numfloor t)
  using split-int-nb[where t=t]
 by (cases fst(split-int t) , auto simp add: numfloor-def Let-def split-def  numadd-nb)

recdef simpnum measure num-size
  simpnum (C j) = C j
  simpnum (Bound n) = CN n 1 (C 0)
  simpnum (Neg t) = numneg (simpnum t)
  simpnum (Add t s) = numadd (simpnum t,simpnum s)
  simpnum (Sub t s) = numsub (simpnum t) (simpnum s)
  simpnum (Mul i t) = (if i = 0 then (C 0) else nummul (simpnum t) i)
  simpnum (Floor t) = numfloor (simpnum t)
  simpnum (CN n c t) = (if c=0 then simpnum t else CN n c (simpnum t))
  simpnum (CF c t s) = simpnum(Add (Mul c (Floor t)) s)

lemma simpnum-ci[simp]: Inum bs (simpnum t) = Inum bs t
by (induct t rule: simpnum.induct, auto)

lemma simpnum-numbound0[simp]:
  numbound0 t ⟹ numbound0 (simpnum t)
by (induct t rule: simpnum.induct, auto)

consts nozerocoeff:: num ⇒ bool
recdef nozerocoeff measure size
  nozerocoeff (C c) = True
  nozerocoeff (CN n c t) = (c≠0 ∧ nozerocoeff t)
  nozerocoeff (CF c s t) = (c ≠ 0 ∧ nozerocoeff t)
  nozerocoeff (Mul c t) = (c≠0 ∧ nozerocoeff t)
  nozerocoeff t = True

lemma numadd-nz : nozerocoeff a ⟹ nozerocoeff b ⟹ nozerocoeff (numadd
(a,b))
by (induct a b rule: numadd.induct,auto simp add: Let-def)

lemma nummul-nz : ⋀ i. i≠0 ⟹ nozerocoeff a ⟹ nozerocoeff (nummul a i)
  by (induct a rule: nummul.induct,auto simp add: Let-def numadd-nz)

lemma numneg-nz : nozerocoeff a ⟹ nozerocoeff (numneg a)
by (simp add: numneg-def nummul-nz)

lemma numsub-nz: nozerocoeff a ⟹ nozerocoeff b ⟹ nozerocoeff (numsub a b)
by (simp add: numsub-def numneg-nz numadd-nz)
```

lemma *split-int-nz*: *nozerocoeff t* $\Longrightarrow$ *nozerocoeff* (*fst* (*split-int t*)) $\wedge$ *nozerocoeff* (*snd* (*split-int t*))
by (*induct t rule*: *split-int.induct*,*auto simp add*: *Let-def split-def*)

lemma *numfloor-nz*: *nozerocoeff t* $\Longrightarrow$ *nozerocoeff* (*numfloor t*)
by (*simp add*: *numfloor-def Let-def split-def*)
(*cases fst* (*split-int t*), *simp-all add*: *split-int-nz numadd-nz*)

lemma *simpnum-nz*: *nozerocoeff* (*simpnum t*)
by(*induct t rule*: *simpnum.induct*, *auto simp add*: *numadd-nz numneg-nz numsub-nz nummul-nz numfloor-nz*)

lemma *maxcoeff-nz*: *nozerocoeff t* $\Longrightarrow$ *maxcoeff t* = *0* $\Longrightarrow$ *t* = *C 0*
proof (*induct t rule*: *maxcoeff.induct*)
  case (*2 n c t*)
  hence *cnz*: *c* $\neq$*0* **and** *mx*: *max* (*abs c*) (*maxcoeff t*) = *0* **by** *simp*+
  have *max* (*abs c*) (*maxcoeff t*) $\geq$ *abs c* **by** (*simp add*: *le-maxI1*)
  with *cnz* **have** *max* (*abs c*) (*maxcoeff t*) > *0* **by** *arith*
  with *prems* **show** *?case* **by** *simp*
next
  case (*3 c s t*)
  hence *cnz*: *c* $\neq$*0* **and** *mx*: *max* (*abs c*) (*maxcoeff t*) = *0* **by** *simp*+
  have *max* (*abs c*) (*maxcoeff t*) $\geq$ *abs c* **by** (*simp add*: *le-maxI1*)
  with *cnz* **have** *max* (*abs c*) (*maxcoeff t*) > *0* **by** *arith*
  with *prems* **show** *?case* **by** *simp*
qed *auto*

lemma *numgcd-nz*: **assumes** *nz*: *nozerocoeff t* **and** *g0*: *numgcd t* = *0* **shows** *t* = *C 0*
proof−
  from *g0* **have** *th*:*numgcdh t* (*maxcoeff t*) = *0* **by** (*simp add*: *numgcd-def*)
  from *numgcdh0*[*OF th*] **have** *th*:*maxcoeff t* = *0* **.**
  from *maxcoeff-nz*[*OF nz th*] **show** *?thesis* **.**
qed

constdefs *simp-num-pair*:: (*num* × *int*) ⇒ *num* × *int*
  simp-num-pair ≡ (λ (*t*,*n*). (*if n* = *0 then* (*C 0*, *0*) *else*
   (*let t′* = *simpnum t* ; *g* = *numgcd t′ in*
      if g > *1 then* (*let g′* = *igcd n g in*
        if g′ = *1 then* (*t′*,*n*)
        else (*reducecoeffh t′ g′*, *n div g′*))
      else (*t′*,*n*))))

lemma *simp-num-pair-ci*:
  shows ((λ (*t*,*n*). *Inum bs t* / *real n*) (*simp-num-pair* (*t*,*n*))) = ((λ (*t*,*n*). *Inum bs t* / *real n*) (*t*,*n*))
  (**is** *?lhs* = *?rhs*)
proof−

let *?t′ = simpnum t*
let *?g = numgcd ?t′*
let *?g′ = igcd n ?g*
{assume *nz*: *n = 0* **hence** *?thesis* **by** (*simp add*: *Let-def simp-num-pair-def*)**}**
moreover
{ assume *nnz*: *n ≠ 0*
{assume ¬ *?g > 1* **hence** *?thesis* **by** (*simp add*: *Let-def simp-num-pair-def*)**}**
moreover
{assume *g1*:*?g>1* **hence** *g0*: *?g > 0* **by** *simp*
from *igcd0 g1 nnz* **have** *gp0*: *?g′ ≠ 0* **by** *simp*
hence *g′p*: *?g′ > 0* **using** *igcd-pos*[**where** *i=n* **and** *j=numgcd ?t′*] **by** *arith*
hence *?g′= 1 ∨ ?g′ > 1* **by** *arith*
moreover {assume *?g′=1* **hence** *?thesis* **by** (*simp add*: *Let-def simp-num-pair-def*)**}**
moreover {assume *g′1*:*?g′>1*
from *dvdnumcoeff-aux2*[*OF g1*] **have** *th1*:*dvdnumcoeff ?t′ ?g* **..**
let *?tt = reducecoeffh ?t′ ?g′*
let *?t = Inum bs ?tt*
have *gpdg*: *?g′ dvd ?g* **by** (*simp add*: *igcd-dvd2*)
have *gpdd*: *?g′ dvd n* **by** (*simp add*: *igcd-dvd1*)
have *gpdgp*: *?g′ dvd ?g′* **by** *simp*
from *reducecoeffh*[*OF dvdnumcoeff-trans*[*OF gpdg th1*] *g′p*]
have *th2*:*real ?g′ ∗ ?t = Inum bs ?t′* **by** *simp*
from *prems* **have** *?lhs = ?t / real* (*n div ?g′*) **by** (*simp add*: *simp-num-pair-def Let-def*)
also have ... = (*real ?g′ ∗ ?t*) / (*real ?g′ ∗* (*real* (*n div ?g′*))) **by** *simp*
also have ... = (*Inum bs ?t′ / real n*)
using *real-of-int-div*[*OF gp0 gpdd*] *th2 gp0* **by** *simp*
finally have *?lhs = Inum bs t / real n* **by** *simp*
then have *?thesis* **using** *prems* **by** (*simp add*: *simp-num-pair-def*)**}**
ultimately have *?thesis* **by** *blast***}**
ultimately have *?thesis* **by** *blast***}**
ultimately show *?thesis* **by** *blast*
qed

lemma *simp-num-pair-l*: **assumes** *tnb*: *numbound0 t* **and** *np*: *n >0* **and** *tn*: *simp-num-pair* (*t*,*n*) = (*t′*,*n′*)
shows *numbound0 t′ ∧ n′ >0*
proof−
let *?t′ = simpnum t*
let *?g = numgcd ?t′*
let *?g′ = igcd n ?g*
{assume *nz*: *n = 0* **hence** *?thesis* **using** *prems* **by** (*simp add*: *Let-def simp-num-pair-def*)**}**
moreover
{ assume *nnz*: *n ≠ 0*
{assume ¬ *?g > 1* **hence** *?thesis* **using** *prems* **by** (*auto simp add*: *Let-def simp-num-pair-def*)**}**
moreover
{assume *g1*:*?g>1* **hence** *g0*: *?g > 0* **by** *simp*
from *igcd0 g1 nnz* **have** *gp0*: *?g′ ≠ 0* **by** *simp*

hence $g'p$: $?g' > 0$ **using** *igcd-pos*[**where** $i=n$ **and** $j=numgcd\ ?t'$] **by** *arith*
hence $?g'= 1 \lor ?g' > 1$ **by** *arith*
moreover {**assume** $?g'=1$ **hence** *?thesis* **using** *prems*
by (*auto simp add*: *Let-def simp-num-pair-def*)}
moreover {**assume** $g'1$:$?g'>1$
have *gpdg*: $?g'$ *dvd* $?g$ **by** (*simp add*: *igcd-dvd2*)
have *gpdd*: $?g'$ *dvd* n **by** (*simp add*: *igcd-dvd1*)
have *gpdgp*: $?g'$ *dvd* $?g'$ **by** *simp*
from *zdvd-imp-le*[*OF gpdd np*] **have** $g'n$: $?g' \leq n$ **.**
from *zdiv-mono1*[*OF* $g'n$ $g'p$, *simplified zdiv-self*[*OF gp0*]]
have n *div* $?g' > 0$ **by** *simp*
hence *?thesis* **using** *prems*
by(*auto simp add*: *simp-num-pair-def Let-def reducecoeffh-numbound0*)}
ultimately have *?thesis* **by** *blast*}
ultimately have *?thesis* **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

consts *not*:: $fm \Rightarrow fm$
recdef *not measure size*
not (*NOT* p) = p
not $T = F$
not $F = T$
not (*Lt* t) = *Ge* t
not (*Le* t) = *Gt* t
not (*Gt* t) = *Le* t
not (*Ge* t) = *Lt* t
not (*Eq* t) = *NEq* t
not (*NEq* t) = *Eq* t
not (*Dvd* i t) = *NDvd* i t
not (*NDvd* i t) = *Dvd* i t
not (*And* p q) = *Or* (*not* p) (*not* q)
not (*Or* p q) = *And* (*not* p) (*not* q)
not p = *NOT* p
lemma *not*[*simp*]: *Ifm bs* (*not* p) = *Ifm bs* (*NOT* p)
by (*induct* p) *auto*
lemma *not-qf*[*simp*]: *qfree* $p \Longrightarrow$ *qfree* (*not* p)
by (*induct* p, *auto*)
lemma *not-nb*[*simp*]: *bound0* $p \Longrightarrow$ *bound0* (*not* p)
by (*induct* p, *auto*)

constdefs *conj* :: $fm \Rightarrow fm \Rightarrow fm$
conj p q $\equiv$ (*if* ($p = F \lor q=F$) *then* F *else if* $p=T$ *then* q *else if* $q=T$ *then* p *else if* $p = q$ *then* p *else And* p q)
lemma *conj*[*simp*]: *Ifm bs* (*conj* p q) = *Ifm bs* (*And* p q)
by (*cases* $p=F \lor q=F$,*simp-all add*: *conj-def*) (*cases* p,*simp-all*)

lemma *conj-qf*[*simp*]: $[\![$*qfree* p ; *qfree* $q]\!] \Longrightarrow$ *qfree* (*conj* p q)
using *conj-def* **by** *auto*

lemma *conj-nb*[*simp*]: $[\![bound0\ p\ ;\ bound0\ q]\!] \Longrightarrow bound0\ (conj\ p\ q)$
using *conj-def* **by** *auto*

constdefs *disj* :: *fm* ⇒ *fm* ⇒ *fm*
disj p q ≡ (*if* (*p* = *T* ∨ *q*=*T*) *then T else if p*=*F then q else if q*=*F then p else if p*=*q then p else Or p q*)

lemma *disj*[*simp*]: *Ifm bs* (*disj p q*) = *Ifm bs* (*Or p q*)
by (*cases p*=*T* ∨ *q*=*T*,*simp-all add*: *disj-def*) (*cases p*,*simp-all*)
lemma *disj-qf*[*simp*]: $[\![qfree\ p\ ;\ qfree\ q]\!] \Longrightarrow qfree\ (disj\ p\ q)$
using *disj-def* **by** *auto*
lemma *disj-nb*[*simp*]: $[\![bound0\ p\ ;\ bound0\ q]\!] \Longrightarrow bound0\ (disj\ p\ q)$
using *disj-def* **by** *auto*

constdefs *imp* :: *fm* ⇒ *fm* ⇒ *fm*
imp p q ≡ (*if* (*p* = *F* ∨ *q*=*T* ∨ *p*=*q*) *then T else if p*=*T then q else if q*=*F then not p*
else Imp p q)
lemma *imp*[*simp*]: *Ifm bs* (*imp p q*) = *Ifm bs* (*Imp p q*)
by (*cases p*=*F* ∨ *q*=*T*,*simp-all add*: *imp-def*)
lemma *imp-qf*[*simp*]: $[\![qfree\ p\ ;\ qfree\ q]\!] \Longrightarrow qfree\ (imp\ p\ q)$
using *imp-def* **by** (*cases p*=*F* ∨ *q*=*T*,*simp-all add*: *imp-def*)
lemma *imp-nb*[*simp*]: $[\![bound0\ p\ ;\ bound0\ q]\!] \Longrightarrow bound0\ (imp\ p\ q)$
using *imp-def* **by** (*cases p*=*F* ∨ *q*=*T* ∨ *p*=*q*,*simp-all add*: *imp-def*)

constdefs *iff* :: *fm* ⇒ *fm* ⇒ *fm*
iff p q ≡ (*if* (*p* = *q*) *then T else if* (*p* = *not q* ∨ *not p* = *q*) *then F else*
if p=*F then not q else if q*=*F then not p else if p*=*T then q else if q*=*T then p else*
Iff p q)
lemma *iff*[*simp*]: *Ifm bs* (*iff p q*) = *Ifm bs* (*Iff p q*)
by (*unfold iff-def*,*cases p*=*q*, *simp*,*cases p*=*not q*, *simp add*:*not*)
(*cases not p*= *q*, *auto simp add*:*not*)
lemma *iff-qf*[*simp*]: $[\![qfree\ p\ ;\ qfree\ q]\!] \Longrightarrow qfree\ (iff\ p\ q)$
by (*unfold iff-def*,*cases p*=*q*, *auto*)
lemma *iff-nb*[*simp*]: $[\![bound0\ p\ ;\ bound0\ q]\!] \Longrightarrow bound0\ (iff\ p\ q)$
using *iff-def* **by** (*unfold iff-def*,*cases p*=*q*, *auto*)

consts *check-int*:: *num* ⇒ *bool*
recdef *check-int measure size*
check-int (*C i*) = *True*
check-int (*Floor t*) = *True*
check-int (*Mul i t*) = *check-int t*
check-int (*Add t s*) = (*check-int t* ∧ *check-int s*)
check-int (*Neg t*) = *check-int t*
check-int (*CF c t s*) = *check-int s*
check-int t = *False*
lemma *check-int*: *check-int t* ⟹ *isint t bs*
by (*induct t*, *auto simp add*: *isint-add isint-Floor isint-Mul isint-neg isint-c isint-CF*)

lemma *rdvd-left1-int*: *real* $\lfloor t \rfloor$ = *t* $\Longrightarrow$ *1 rdvd t*
  by (*simp add*: *rdvd-def*,*rule-tac x*=$\lfloor t \rfloor$ **in** *exI*) *simp*

lemma *rdvd-reduce*:
  assumes *gd*:*g dvd d* **and** *gc*:*g dvd c* **and** *gp*: *g* > *0*
  shows *real* (*d*::*int*) *rdvd real* (*c*::*int*)∗*t* = (*real* (*d div g*) *rdvd real* (*c div g*)∗*t*)
proof
  assume *d*: *real d rdvd real c* ∗ *t*
  from *d rdvd-def* **obtain** *k* **where** *k-def*: *real c* ∗ *t* = *real d*∗ *real* (*k*::*int*) **by** *auto*
  from *gd dvd-def* **obtain** *kd* **where** *kd-def*: *d* = *g* ∗ *kd* **by** *auto*
  from *gc dvd-def* **obtain** *kc* **where** *kc-def*: *c* = *g* ∗ *kc* **by** *auto*
  from *k-def kd-def kc-def* **have** *real g* ∗ *real kc* ∗ *t* = *real g* ∗ *real kd* ∗ *real k* **by** *simp*
  hence *real kc* ∗ *t* = *real kd* ∗ *real k* **using** *gp* **by** *simp*
  hence *th*:*real kd rdvd real kc* ∗ *t* **using** *rdvd-def* **by** *blast*
  from *kd-def gp* **have** *th′*:*kd* = *d div g* **by** *simp*
  from *kc-def gp* **have** *kc* = *c div g* **by** *simp*
  with *th th′* **show** *real* (*d div g*) *rdvd real* (*c div g*) ∗ *t* **by** *simp*
next
  assume *d*: *real* (*d div g*) *rdvd real* (*c div g*) ∗ *t*
  from *gp* **have** *gnz*: *g* ≠ *0* **by** *simp*
  thus *real d rdvd real c* ∗ *t* **using** *d rdvd-mult*[*OF gnz*, **where** *n*=*d div g* **and** *x*=*real* (*c div g*) ∗ *t*] *real-of-int-div*[*OF gnz gd*] *real-of-int-div*[*OF gnz gc*] **by** *simp*
qed

constdefs *simpdvd*:: *int* ⇒ *num* ⇒ (*int* × *num*)
  simpdvd d t ≡
   (*let g* = *numgcd t in*
      if g > *1 then* (*let g′* = *igcd d g in*
        if g′ = *1 then* (*d*, *t*)
        else (*d div g′*,*reducecoeffh t g′*))
      else (*d*, *t*))
lemma *simpdvd*:
  assumes *tnz*: *nozerocoeff t* **and** *dnz*: *d* ≠ *0*
  shows *Ifm bs* (*Dvd* (*fst* (*simpdvd d t*)) (*snd* (*simpdvd d t*))) = *Ifm bs* (*Dvd d t*)
proof−
  let *?g* = *numgcd t*
  let *?g′* = *igcd d ?g*
  {**assume** ¬ *?g* > *1* **hence** *?thesis* **by** (*simp add*: *Let-def simpdvd-def*)}
  moreover
  {**assume** *g1*:*?g*>*1* **hence** *g0*: *?g* > *0* **by** *simp*
    from *igcd0 g1 dnz* **have** *gp0*: *?g′* ≠ *0* **by** *simp*
    hence *g′p*: *?g′* > *0* **using** *igcd-pos*[**where** *i*=*d* **and** *j*=*numgcd t*] **by** *arith*
    hence *?g′*= *1* ∨ *?g′* > *1* **by** *arith*
   moreover {**assume** *?g′*=*1* **hence** *?thesis* **by** (*simp add*: *Let-def simpdvd-def*)}
    moreover {**assume** *g′1*:*?g′*>*1*
      from *dvdnumcoeff-aux2*[*OF g1*] **have** *th1*:*dvdnumcoeff t ?g* **..**

let *?tt* = *reducecoeffh t ?g′*
let *?t* = *Inum bs ?tt*
have *gpdg*: *?g′ dvd ?g* **by** (*simp add*: *igcd-dvd2*)
have *gpdd*: *?g′ dvd d* **by** (*simp add*: *igcd-dvd1*)
have *gpdgp*: *?g′ dvd ?g′* **by** *simp*
from *reducecoeffh*[*OF dvdnumcoeff-trans*[*OF gpdg th1*] *g′p*]
have *th2*:*real ?g′* ∗ *?t* = *Inum bs t* **by** *simp*
from *prems* **have** *Ifm bs* (*Dvd* (*fst* (*simpdvd d t*)) (*snd*(*simpdvd d t*))) = *Ifm bs* (*Dvd* (*d div ?g′*) *?tt*)
by (*simp add*: *simpdvd-def Let-def*)
also have . . . = (*real d rdvd* (*Inum bs t*))
using *rdvd-reduce*[*OF gpdd gpdgp g′p*, **where** *t*=*?t*, *simplified zdiv-self*[*OF gp0*]]
th2[*symmetric*] **by** *simp*
finally have *?thesis* **by** *simp* **}**
ultimately have *?thesis* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed

consts *simpfm* :: *fm* ⇒ *fm*
recdef *simpfm measure fmsize*
simpfm (*And p q*) = *conj* (*simpfm p*) (*simpfm q*)
simpfm (*Or p q*) = *disj* (*simpfm p*) (*simpfm q*)
simpfm (*Imp p q*) = *imp* (*simpfm p*) (*simpfm q*)
simpfm (*Iff p q*) = *iff* (*simpfm p*) (*simpfm q*)
simpfm (*NOT p*) = *not* (*simpfm p*)
simpfm (*Lt a*) = (*let a′* = *simpnum a in case a′ of C v* ⇒ *if* (*v* < *0*) *then T else F*
| - ⇒ *Lt* (*reducecoeff a′*))
simpfm (*Le a*) = (*let a′* = *simpnum a in case a′ of C v* ⇒ *if* (*v* ≤ *0*) *then T else F* | - ⇒ *Le* (*reducecoeff a′*))
simpfm (*Gt a*) = (*let a′* = *simpnum a in case a′ of C v* ⇒ *if* (*v* > *0*) *then T else F* | - ⇒ *Gt* (*reducecoeff a′*))
simpfm (*Ge a*) = (*let a′* = *simpnum a in case a′ of C v* ⇒ *if* (*v* ≥ *0*) *then T else F* | - ⇒ *Ge* (*reducecoeff a′*))
simpfm (*Eq a*) = (*let a′* = *simpnum a in case a′ of C v* ⇒ *if* (*v* = *0*) *then T else F* | - ⇒ *Eq* (*reducecoeff a′*))
simpfm (*NEq a*) = (*let a′* = *simpnum a in case a′ of C v* ⇒ *if* (*v* ≠ *0*) *then T else F* | - ⇒ *NEq* (*reducecoeff a′*))
simpfm (*Dvd i a*) = (*if i*=*0 then simpfm* (*Eq a*)
else if (*abs i* = *1*) ∧ *check-int a then T*
else let a′ = *simpnum a in case a′ of C v* ⇒ *if* (*i dvd v*) *then T else F* | - ⇒ (*let* (*d*,*t*) = *simpdvd i a′ in Dvd d t*))
simpfm (*NDvd i a*) = (*if i*=*0 then simpfm* (*NEq a*)
else if (*abs i* = *1*) ∧ *check-int a then F*
else let a′ = *simpnum a in case a′ of C v* ⇒ *if* (¬(*i dvd v*)) *then T else F* | - ⇒ (*let* (*d*,*t*) = *simpdvd i a′ in NDvd d t*))
simpfm p = *p*

```
lemma simpfm[simp]: Ifm bs (simpfm p) = Ifm bs p
proof(induct p rule: simpfm.induct)
  case (6 a) let ?sa = simpnum a have sa: Inum bs ?sa = Inum bs a by simp
  {fix v assume ?sa = C v hence ?case using sa by simp }
  moreover {assume H:¬ (∃ v. ?sa = C v)
    let ?g = numgcd ?sa
    let ?rsa = reducecoeff ?sa
    let ?r = Inum bs ?rsa
    have sa-nz: nozerocoeff ?sa by (rule simpnum-nz)
    {assume gz: ?g=0 from numgcd-nz[OF sa-nz gz] H have False by auto}
    with numgcd-pos[where t=?sa] have ?g > 0 by (cases ?g=0, auto)
    hence gp: real ?g > 0 by simp
    have Inum bs ?sa = real ?g∗ ?r by (simp add: reducecoeff)
    with sa have Inum bs a < 0 = (real ?g ∗ ?r < real ?g ∗ 0) by simp
    also have . . . = (?r < 0) using gp
      by (simp only: mult-less-cancel-left) simp
    finally have ?case using H by (cases ?sa , simp-all add: Let-def)}
  ultimately show ?case by blast
next
  case (7 a)  let ?sa = simpnum a have sa: Inum bs ?sa = Inum bs a by simp
  {fix v assume ?sa = C v hence ?case using sa by simp }
  moreover {assume H:¬ (∃ v. ?sa = C v)
    let ?g = numgcd ?sa
    let ?rsa = reducecoeff ?sa
    let ?r = Inum bs ?rsa
    have sa-nz: nozerocoeff ?sa by (rule simpnum-nz)
    {assume gz: ?g=0 from numgcd-nz[OF sa-nz gz] H have False by auto}
    with numgcd-pos[where t=?sa] have ?g > 0 by (cases ?g=0, auto)
    hence gp: real ?g > 0 by simp
    have Inum bs ?sa = real ?g∗ ?r by (simp add: reducecoeff)
    with sa have Inum bs a ≤ 0 = (real ?g ∗ ?r ≤ real ?g ∗ 0) by simp
    also have . . . = (?r ≤ 0) using gp
      by (simp only: mult-le-cancel-left) simp
    finally have ?case using H by (cases ?sa , simp-all add: Let-def)}
  ultimately show ?case by blast
next
  case (8 a)  let ?sa = simpnum a have sa: Inum bs ?sa = Inum bs a by simp
  {fix v assume ?sa = C v hence ?case using sa by simp }
  moreover {assume H:¬ (∃ v. ?sa = C v)
    let ?g = numgcd ?sa
    let ?rsa = reducecoeff ?sa
    let ?r = Inum bs ?rsa
    have sa-nz: nozerocoeff ?sa by (rule simpnum-nz)
    {assume gz: ?g=0 from numgcd-nz[OF sa-nz gz] H have False by auto}
    with numgcd-pos[where t=?sa] have ?g > 0 by (cases ?g=0, auto)
    hence gp: real ?g > 0 by simp
    have Inum bs ?sa = real ?g∗ ?r by (simp add: reducecoeff)
    with sa have Inum bs a > 0 = (real ?g ∗ ?r > real ?g ∗ 0) by simp
```

also have ... = (*?r* > *0*) **using** *gp*
  by (*simp only*: *mult-less-cancel-left*) *simp*
finally have *?case* **using** *H* **by** (*cases ?sa* , *simp-all add*: *Let-def*)}
ultimately show *?case* **by** *blast*
next
case (*9 a*) **let** *?sa* = *simpnum a* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
{**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
moreover {**assume** *H*:¬ (∃ *v*. *?sa* = *C v*)
let *?g* = *numgcd ?sa*
let *?rsa* = *reducecoeff ?sa*
let *?r* = *Inum bs ?rsa*
have *sa-nz*: *nozerocoeff ?sa* **by** (*rule simpnum-nz*)
{**assume** *gz*: *?g=0* **from** *numgcd-nz*[*OF sa-nz gz*] *H* **have** *False* **by** *auto*}
with *numgcd-pos*[**where** *t=?sa*] **have** *?g* > *0* **by** (*cases ?g=0*, *auto*)
hence *gp*: *real ?g* > *0* **by** *simp*
have *Inum bs ?sa* = *real ?g*∗ *?r* **by** (*simp add*: *reducecoeff*)
with *sa* **have** *Inum bs a* ≥ *0* = (*real ?g* ∗ *?r* ≥ *real ?g* ∗ *0*) **by** *simp*
also have ... = (*?r* ≥ *0*) **using** *gp*
  by (*simp only*: *mult-le-cancel-left*) *simp*
finally have *?case* **using** *H* **by** (*cases ?sa* , *simp-all add*: *Let-def*)}
ultimately show *?case* **by** *blast*
next
case (*10 a*) **let** *?sa* = *simpnum a* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
{**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
moreover {**assume** *H*:¬ (∃ *v*. *?sa* = *C v*)
let *?g* = *numgcd ?sa*
let *?rsa* = *reducecoeff ?sa*
let *?r* = *Inum bs ?rsa*
have *sa-nz*: *nozerocoeff ?sa* **by** (*rule simpnum-nz*)
{**assume** *gz*: *?g=0* **from** *numgcd-nz*[*OF sa-nz gz*] *H* **have** *False* **by** *auto*}
with *numgcd-pos*[**where** *t=?sa*] **have** *?g* > *0* **by** (*cases ?g=0*, *auto*)
hence *gp*: *real ?g* > *0* **by** *simp*
have *Inum bs ?sa* = *real ?g*∗ *?r* **by** (*simp add*: *reducecoeff*)
with *sa* **have** *Inum bs a* = *0* = (*real ?g* ∗ *?r* = *0*) **by** *simp*
also have ... = (*?r* = *0*) **using** *gp*
  by (*simp add*: *mult-eq-0-iff*)
finally have *?case* **using** *H* **by** (*cases ?sa* , *simp-all add*: *Let-def*)}
ultimately show *?case* **by** *blast*
next
case (*11 a*) **let** *?sa* = *simpnum a* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
{**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
moreover {**assume** *H*:¬ (∃ *v*. *?sa* = *C v*)
let *?g* = *numgcd ?sa*
let *?rsa* = *reducecoeff ?sa*
let *?r* = *Inum bs ?rsa*
have *sa-nz*: *nozerocoeff ?sa* **by** (*rule simpnum-nz*)
{**assume** *gz*: *?g=0* **from** *numgcd-nz*[*OF sa-nz gz*] *H* **have** *False* **by** *auto*}
with *numgcd-pos*[**where** *t=?sa*] **have** *?g* > *0* **by** (*cases ?g=0*, *auto*)
hence *gp*: *real ?g* > *0* **by** *simp*

have *Inum bs ?sa* = *real ?g*∗ *?r* **by** (*simp add*: *reducecoeff*)
with *sa* **have** *Inum bs a* ≠ *0* = (*real ?g* ∗ *?r* ≠ *0*) **by** *simp*
also have ... = (*?r* ≠ *0*) **using** *gp*
by (*simp add*: *mult-eq-0-iff*)
finally have *?case* **using** *H* **by** (*cases ?sa* , *simp-all add*: *Let-def*)}
ultimately show *?case* **by** *blast*
next
case (*12 i a*) **let** *?sa* = *simpnum a* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
have *i=0* ∨ (*abs i* = *1* ∧ *check-int a*) ∨ (*i≠0* ∧ ((*abs i* ≠ *1*) ∨ (¬ *check-int a*))) **by** *auto*
{**assume** *i=0* **hence** *?case* **using** *12.hyps* **by** (*simp add*: *rdvd-left-0-eq Let-def*)}
moreover
{**assume** *ai1*: *abs i* = *1* **and** *ai*: *check-int a*
hence *i=1* ∨ *i=* − *1* **by** *arith*
moreover {**assume** *i1*: *i* = *1*
from *rdvd-left1-int*[*OF check-int*[*OF ai*, *simplified isint-iff*]]
have *?case* **using** *i1 ai* **by** *simp* }
moreover {**assume** *i1*: *i* = − *1*
from *rdvd-left1-int*[*OF check-int*[*OF ai*, *simplified isint-iff*]]
rdvd-abs1[**where** *d=*− *1* **and** *t=Inum bs a*]
have *?case* **using** *i1 ai* **by** *simp* }
ultimately have *?case* **by** *blast*}
moreover
{**assume** *inz*: *i≠0* **and** *cond*: (*abs i* ≠ *1*) ∨ (¬ *check-int a*)
{**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa*[*symmetric*] *inz cond*
by (*cases abs i* = *1*, *auto simp add*: *int-rdvd-iff*) }
moreover {**assume** *H*:¬ (∃ *v*. *?sa* = *C v*)
hence *th*: *simpfm* (*Dvd i a*) = *Dvd* (*fst* (*simpdvd i ?sa*)) (*snd* (*simpdvd i ?sa*)) **using** *inz cond* **by** (*cases ?sa*, *auto simp add*: *Let-def split-def*)
from *simpnum-nz* **have** *nz*:*nozerocoeff ?sa* **by** *simp*
from *simpdvd* [*OF nz inz*] *th* **have** *?case* **using** *sa* **by** *simp*}
ultimately have *?case* **by** *blast*}
ultimately show *?case* **by** *blast*
next
case (*13 i a*) **let** *?sa* = *simpnum a* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
have *i=0* ∨ (*abs i* = *1* ∧ *check-int a*) ∨ (*i≠0* ∧ ((*abs i* ≠ *1*) ∨ (¬ *check-int a*))) **by** *auto*
{**assume** *i=0* **hence** *?case* **using** *13.hyps* **by** (*simp add*: *rdvd-left-0-eq Let-def*)}
moreover
{**assume** *ai1*: *abs i* = *1* **and** *ai*: *check-int a*
hence *i=1* ∨ *i=* − *1* **by** *arith*
moreover {**assume** *i1*: *i* = *1*
from *rdvd-left1-int*[*OF check-int*[*OF ai*, *simplified isint-iff*]]
have *?case* **using** *i1 ai* **by** *simp* }
moreover {**assume** *i1*: *i* = − *1*
from *rdvd-left1-int*[*OF check-int*[*OF ai*, *simplified isint-iff*]]
rdvd-abs1[**where** *d=*− *1* **and** *t=Inum bs a*]

have *?case* **using** *i1 ai* **by** *simp* **}**
ultimately have *?case* **by** *blast***}**
moreover
{assume *inz*: $i \neq 0$ **and** *cond*: (*abs i* $\neq$ *1*) $\vee$ ($\neg$ *check-int a*)
{fix *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa*[*symmetric*] *inz cond*
by (*cases abs i* = *1*, *auto simp add*: *int-rdvd-iff*) **}**
moreover {assume *H*:$\neg$ ($\exists$ *v*. *?sa* = *C v*)
hence *th*: *simpfm* (*NDvd i a*) = *NDvd* (*fst* (*simpdvd i ?sa*)) (*snd* (*simpdvd i ?sa*)) **using** *inz cond*
by (*cases ?sa*, *auto simp add*: *Let-def split-def*)
from *simpnum-nz* **have** *nz*:*nozerocoeff ?sa* **by** *simp*
from *simpdvd* [*OF nz inz*] *th* **have** *?case* **using** *sa* **by** *simp***}**
ultimately have *?case* **by** *blast***}**
ultimately show *?case* **by** *blast*
qed (*induct p rule*: *simpfm.induct*, *simp-all*)

lemma *simpdvd-numbound0*: *numbound0 t* $\Longrightarrow$ *numbound0* (*snd* (*simpdvd d t*))
by (*simp add*: *simpdvd-def Let-def split-def reducecoeffh-numbound0*)

lemma *simpfm-bound0*[*simp*]: *bound0 p* $\Longrightarrow$ *bound0* (*simpfm p*)
proof(*induct p rule*: *simpfm.induct*)
case (*6 a*) **hence** *nb*: *numbound0 a* **by** *simp*
hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def reducecoeff-numbound0*)
next
case (*7 a*) **hence** *nb*: *numbound0 a* **by** *simp*
hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def reducecoeff-numbound0*)
next
case (*8 a*) **hence** *nb*: *numbound0 a* **by** *simp*
hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def reducecoeff-numbound0*)
next
case (*9 a*) **hence** *nb*: *numbound0 a* **by** *simp*
hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def reducecoeff-numbound0*)
next
case (*10 a*) **hence** *nb*: *numbound0 a* **by** *simp*
hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def reducecoeff-numbound0*)
next
case (*11 a*) **hence** *nb*: *numbound0 a* **by** *simp*
hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def reducecoeff-numbound0*)
next
case (*12 i a*) **hence** *nb*: *numbound0 a* **by** *simp*
hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def reducecoeff-numbound0 simpdvd-numbound0 split-def*)

```
next
  case (13 i a) hence nb: numbound0 a by simp
  hence numbound0 (simpnum a) by (simp only: simpnum-numbound0[OF nb])
  thus ?case by (cases simpnum a, auto simp add: Let-def reducecoeff-numbound0
simpdvd-numbound0 split-def)
qed(auto simp add: disj-def imp-def iff-def conj-def)

lemma simpfm-qf[simp]: qfree p ⟹ qfree (simpfm p)
by (induct p rule: simpfm.induct, auto simp add: Let-def)
(case-tac simpnum a,auto simp add: split-def Let-def)+




constdefs list-conj :: fm list ⇒ fm
  list-conj ps ≡ foldr conj ps T
lemma list-conj: Ifm bs (list-conj ps) = (∀ p∈ set ps. Ifm bs p)
  by (induct ps, auto simp add: list-conj-def)
lemma list-conj-qf:  ∀ p∈ set ps. qfree p ⟹ qfree (list-conj ps)
  by (induct ps, auto simp add: list-conj-def)
lemma list-conj-nb:  ∀ p∈ set ps. bound0 p ⟹ bound0 (list-conj ps)
  by (induct ps, auto simp add: list-conj-def)
constdefs CJNB:: (fm ⇒ fm) ⇒ fm ⇒ fm
  CJNB f p ≡ (let cjs = conjuncts p ; (yes,no) = partition bound0 cjs
                 in conj (decr (list-conj yes)) (f (list-conj no)))

lemma CJNB-qe:
  assumes qe: ∀ bs p. qfree p ⟶ qfree (qe p) ∧ (Ifm bs (qe p) = Ifm bs (E p))
  shows ∀ bs p. qfree p ⟶ qfree (CJNB qe p) ∧ (Ifm bs ((CJNB qe p)) = Ifm bs
(E p))
proof(clarify)
  fix bs p
  assume qfp: qfree p
  let ?cjs = conjuncts p
  let ?yes = fst (partition bound0 ?cjs)
  let ?no = snd (partition bound0 ?cjs)
  let ?cno = list-conj ?no
  let ?cyes = list-conj ?yes
  have part: partition bound0 ?cjs = (?yes,?no) by simp
  from partition-P[OF part] have ∀ q∈ set ?yes. bound0 q by blast
  hence yes-nb: bound0 ?cyes by (simp add: list-conj-nb)
  hence yes-qf: qfree (decr ?cyes ) by (simp add: decr-qf)
  from conjuncts-qf[OF qfp] partition-set[OF part]
  have ∀q∈ set ?no. qfree q by auto
  hence no-qf: qfree ?cnoby (simp add: list-conj-qf)
  with qe have cno-qf:qfree (qe ?cno )
    and noE: Ifm bs (qe ?cno) = Ifm bs (E ?cno) by blast+
  from cno-qf yes-qf have qf: qfree (CJNB qe p)
    by (simp add: CJNB-def Let-def conj-qf split-def)
```

{**fix** *bs*
from *conjuncts* **have** *Ifm bs p* = (∀ *q*∈ *set ?cjs*. *Ifm bs q*) **by** *blast*
also have ... = ((∀ *q*∈ *set ?yes*. *Ifm bs q*) ∧ (∀ *q*∈ *set ?no*. *Ifm bs q*))
using *partition-set*[*OF part*] **by** *auto*
finally have *Ifm bs p* = ((*Ifm bs ?cyes*) ∧ (*Ifm bs ?cno*)) **using** *list-conj* **by** *simp*}
hence *Ifm bs* (*E p*) = (∃ *x*. (*Ifm* (*x*#*bs*) *?cyes*) ∧ (*Ifm* (*x*#*bs*) *?cno*)) **by** *simp*
also have ... = (∃ *x*. (*Ifm* (*y*#*bs*) *?cyes*) ∧ (*Ifm* (*x*#*bs*) *?cno*))
using *bound0-I*[*OF yes-nb*, **where** *bs*=*bs* **and** *b'*=*y*] **by** *blast*
also have ... = (*Ifm bs* (*decr ?cyes*) ∧ *Ifm bs* (*E ?cno*))
by (*auto simp add*: *decr*[*OF yes-nb*])
also have ... = (*Ifm bs* (*conj* (*decr ?cyes*) (*qe ?cno*)))
using *qe*[*rule-format*, *OF no-qf*] **by** *auto*
finally have *Ifm bs* (*E p*) = *Ifm bs* (*CJNB qe p*)
by (*simp add*: *Let-def CJNB-def split-def*)
with *qf* **show** *qfree* (*CJNB qe p*) ∧ *Ifm bs* (*CJNB qe p*) = *Ifm bs* (*E p*) **by** *blast*
qed

consts *qelim* :: *fm* ⇒ (*fm* ⇒ *fm*) ⇒ *fm*
recdef *qelim measure fmsize*
qelim (*E p*) = (λ *qe*. *DJ* (*CJNB qe*) (*qelim p qe*))
qelim (*A p*) = (λ *qe*. *not* (*qe* ((*qelim* (*NOT p*) *qe*))))
qelim (*NOT p*) = (λ *qe*. *not* (*qelim p qe*))
qelim (*And p q*) = (λ *qe*. *conj* (*qelim p qe*) (*qelim q qe*))
qelim (*Or p q*) = (λ *qe*. *disj* (*qelim p qe*) (*qelim q qe*))
qelim (*Imp p q*) = (λ *qe*. *disj* (*qelim* (*NOT p*) *qe*) (*qelim q qe*))
qelim (*Iff p q*) = (λ *qe*. *iff* (*qelim p qe*) (*qelim q qe*))
qelim p = (λ *y*. *simpfm p*)

lemma *qelim-ci*:
assumes *qe-inv*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ⋀ *bs*. *qfree* (*qelim p qe*) ∧ (*Ifm bs* (*qelim p qe*) = *Ifm bs p*)
using *qe-inv DJ-qe*[*OF CJNB-qe*[*OF qe-inv*]]
by(*induct p rule*: *qelim.induct*)
(*auto simp del*: *simpfm.simps*)

The ℤ Part

Linearity for fm where Bound 0 ranges over ℤ

consts
zsplit0 :: *num* ⇒ *int* × *num*
recdef *zsplit0 measure num-size*
zsplit0 (*C c*) = (*0*,*C c*)
zsplit0 (*Bound n*) = (*if n*=*0 then* (*1*, *C 0*) *else* (*0*,*Bound n*))
zsplit0 (*CN n c a*) = *zsplit0* (*Add* (*Mul c* (*Bound n*)) *a*)
zsplit0 (*CF c a b*) = *zsplit0* (*Add* (*Mul c* (*Floor a*)) *b*)
zsplit0 (*Neg a*) = (*let* (*i'*,*a'*) = *zsplit0 a in* (−*i'*, *Neg a'*))
zsplit0 (*Add a b*) = (*let* (*ia*,*a'*) = *zsplit0 a* ;
(*ib*,*b'*) = *zsplit0 b*

```
                    in (ia+ib, Add a' b'))
  zsplit0 (Sub a b) = (let (ia,a') = zsplit0 a ;
                    (ib,b') = zsplit0 b
                    in (ia−ib, Sub a' b'))
  zsplit0 (Mul i a) = (let (i',a') = zsplit0 a in (i∗i', Mul i a'))
  zsplit0 (Floor a) = (let (i',a') = zsplit0 a in (i',Floor a'))
(hints simp add: Let-def)

lemma zsplit0-I:
  shows ⋀ n a. zsplit0 t = (n,a) ⟹ (Inum ((real (x::int)) #bs) (CN 0 n a) =
Inum (real x #bs) t) ∧ numbound0 a
  (is ⋀ n a. ?S t = (n,a) ⟹ (?I x (CN 0 n a) = ?I x t) ∧ ?N a)
proof(induct t rule: zsplit0.induct)
  case (1 c n a) thus ?case by auto
next
  case (2 m n a) thus ?case by (cases m=0) auto
next
  case (3 n i a n a') thus ?case by auto
next
  case (4 c a b n a') thus ?case by auto
next
  case (5 t n a)
  let ?nt = fst (zsplit0 t)
  let ?at = snd (zsplit0 t)
  have abj: zsplit0 t = (?nt,?at) by simp hence th: a=Neg ?at ∧ n=−?nt using
prems
    by (simp add: Let-def split-def)
  from abj prems have th2: (?I x (CN 0 ?nt ?at) = ?I x t) ∧ ?N ?at by blast
  from th2[simplified] th[simplified] show ?case by simp
next
  case (6 s t n a)
  let ?ns = fst (zsplit0 s)
  let ?as = snd (zsplit0 s)
  let ?nt = fst (zsplit0 t)
  let ?at = snd (zsplit0 t)
  have abjs: zsplit0 s = (?ns,?as) by simp
  moreover have abjt: zsplit0 t = (?nt,?at) by simp
  ultimately have th: a=Add ?as ?at ∧ n=?ns + ?nt using prems
    by (simp add: Let-def split-def)
  from abjs[symmetric] have bluddy: ∃ x y. (x,y) = zsplit0 s by blast
  from prems have (∃ x y. (x,y) = zsplit0 s) ⟶ (∀ xa xb. zsplit0 t = (xa, xb)
⟶ Inum (real x # bs) (CN 0 xa xb) = Inum (real x # bs) t ∧ numbound0 xb)
by simp
  with bluddy abjt have th3: (?I x (CN 0 ?nt ?at) = ?I x t) ∧ ?N ?at by blast
  from abjs prems have th2: (?I x (CN 0 ?ns ?as) = ?I x s) ∧ ?N ?as by blast
  from th3[simplified] th2[simplified] th[simplified] show ?case
    by (simp add: left-distrib)
next
  case (7 s t n a)
```

let *?ns* = *fst* (*zsplit0 s*)
let *?as* = *snd* (*zsplit0 s*)
let *?nt* = *fst* (*zsplit0 t*)
let *?at* = *snd* (*zsplit0 t*)
have *abjs*: *zsplit0 s* = (*?ns*,*?as*) **by** *simp*
moreover have *abjt*: *zsplit0 t* = (*?nt*,*?at*) **by** *simp*
ultimately have *th*: *a*=*Sub ?as ?at* ∧ *n*=*?ns* − *?nt* **using** *prems*
by (*simp add*: *Let-def split-def*)
from *abjs*[*symmetric*] **have** *bluddy*: ∃ *x y*. (*x*,*y*) = *zsplit0 s* **by** *blast*
from *prems* **have** (∃ *x y*. (*x*,*y*) = *zsplit0 s*) ⟶ (∀ *xa xb*. *zsplit0 t* = (*xa*, *xb*) ⟶ *Inum* (*real x* # *bs*) (*CN 0 xa xb*) = *Inum* (*real x* # *bs*) *t* ∧ *numbound0 xb*) **by** *simp*
with *bluddy abjt* **have** *th3*: (*?I x* (*CN 0 ?nt ?at*) = *?I x t*) ∧ *?N ?at* **by** *blast*
from *abjs prems* **have** *th2*: (*?I x* (*CN 0 ?ns ?as*) = *?I x s*) ∧ *?N ?as* **by** *blast*
from *th3*[*simplified*] *th2*[*simplified*] *th*[*simplified*] **show** *?case*
by (*simp add*: *left-diff-distrib*)
next
case (*8 i t n a*)
let *?nt* = *fst* (*zsplit0 t*)
let *?at* = *snd* (*zsplit0 t*)
have *abj*: *zsplit0 t* = (*?nt*,*?at*) **by** *simp* **hence** *th*: *a*=*Mul i ?at* ∧ *n*=*i*∗*?nt* **using** *prems*
by (*simp add*: *Let-def split-def*)
from *abj prems* **have** *th2*: (*?I x* (*CN 0 ?nt ?at*) = *?I x t*) ∧ *?N ?at* **by** *blast*
hence *?I x* (*Mul i t*) = (*real i*) ∗ *?I x* (*CN 0 ?nt ?at*) **by** *simp*
also have ... = *?I x* (*CN 0* (*i*∗*?nt*) (*Mul i ?at*)) **by** (*simp add*: *right-distrib*)
finally show *?case* **using** *th th2* **by** *simp*
next
case (*9 t n a*)
let *?nt* = *fst* (*zsplit0 t*)
let *?at* = *snd* (*zsplit0 t*)
have *abj*: *zsplit0 t* = (*?nt*,*?at*) **by** *simp* **hence** *th*: *a*= *Floor ?at* ∧ *n*=*?nt* **using** *prems*
by (*simp add*: *Let-def split-def*)
from *abj prems* **have** *th2*: (*?I x* (*CN 0 ?nt ?at*) = *?I x t*) ∧ *?N ?at* **by** *blast*
hence *na*: *?N a* **using** *th* **by** *simp*
have *th'*: (*real ?nt*)∗(*real x*) = *real* (*?nt* ∗ *x*) **by** *simp*
have *?I x* (*Floor t*) = *?I x* (*Floor* (*CN 0 ?nt ?at*)) **using** *th2* **by** *simp*
also have ... = *real* (*floor* ((*real ?nt*)∗ *real*(*x*) + *?I x ?at*)) **by** *simp*
also have ... = *real* (*floor* (*?I x ?at* + *real* (*?nt*∗ *x*))) **by** (*simp add*: *add-ac*)
also have ... = *real* (*floor* (*?I x ?at*) + (*?nt*∗ *x*))
using *floor-add*[**where** *x*=*?I x ?at* **and** *a*=*?nt*∗ *x*] **by** *simp*
also have ... = *real* (*?nt*)∗(*real x*) + *real* (*floor* (*?I x ?at*)) **by** (*simp add*: *add-ac*)
finally have *?I x* (*Floor t*) = *?I x* (*CN 0 n a*) **using** *th* **by** *simp*
with *na* **show** *?case* **by** *simp*
qed

consts

iszlfm :: *fm* ⇒ *real list* ⇒ *bool*
zlfm :: *fm* ⇒ *fm*
recdef *iszlfm measure size*
iszlfm (*And p q*) = (λ *bs*. *iszlfm p bs* ∧ *iszlfm q bs*)
iszlfm (*Or p q*) = (λ *bs*. *iszlfm p bs* ∧ *iszlfm q bs*)
iszlfm (*Eq* (*CN 0 c e*)) = (λ *bs*. *c*>*0* ∧ *numbound0 e* ∧ *isint e bs*)
iszlfm (*NEq* (*CN 0 c e*)) = (λ *bs*. *c*>*0* ∧ *numbound0 e* ∧ *isint e bs*)
iszlfm (*Lt* (*CN 0 c e*)) = (λ *bs*. *c*>*0* ∧ *numbound0 e* ∧ *isint e bs*)
iszlfm (*Le* (*CN 0 c e*)) = (λ *bs*. *c*>*0* ∧ *numbound0 e* ∧ *isint e bs*)
iszlfm (*Gt* (*CN 0 c e*)) = (λ *bs*. *c*>*0* ∧ *numbound0 e* ∧ *isint e bs*)
iszlfm (*Ge* (*CN 0 c e*)) = (λ *bs*. *c*>*0* ∧ *numbound0 e* ∧ *isint e bs*)
iszlfm (*Dvd i* (*CN 0 c e*)) =
(λ *bs*. *c*>*0* ∧ *i*>*0* ∧ *numbound0 e* ∧ *isint e bs*)
iszlfm (*NDvd i* (*CN 0 c e*))=
(λ *bs*. *c*>*0* ∧ *i*>*0* ∧ *numbound0 e* ∧ *isint e bs*)
iszlfm p = (λ *bs*. *isatom p* ∧ (*bound0 p*))

lemma *zlin-qfree*: *iszlfm p bs* ⟹ *qfree p*
by (*induct p rule*: *iszlfm.induct*) *auto*

lemma *iszlfm-gen*:
assumes *lp*: *iszlfm p* (*x*#*bs*)
shows ∀ *y*. *iszlfm p* (*y*#*bs*)
proof
fix *y*
show *iszlfm p* (*y*#*bs*)
using *lp*
by(*induct p rule*: *iszlfm.induct*, *simp-all add*: *numbound0-gen*[*rule-format*, **where** *x*=*x* **and** *y*=*y*])
qed

lemma *conj-zl*[*simp*]: *iszlfm p bs* ⟹ *iszlfm q bs* ⟹ *iszlfm* (*conj p q*) *bs*
using *conj-def* **by** (*cases p*,*auto*)
lemma *disj-zl*[*simp*]: *iszlfm p bs* ⟹ *iszlfm q bs* ⟹ *iszlfm* (*disj p q*) *bs*
using *disj-def* **by** (*cases p*,*auto*)
lemma *not-zl*[*simp*]: *iszlfm p bs* ⟹ *iszlfm* (*not p*) *bs*
by (*induct p rule*:*iszlfm.induct* ,*auto*)

recdef *zlfm measure fmsize*
zlfm (*And p q*) = *conj* (*zlfm p*) (*zlfm q*)
zlfm (*Or p q*) = *disj* (*zlfm p*) (*zlfm q*)
zlfm (*Imp p q*) = *disj* (*zlfm* (*NOT p*)) (*zlfm q*)
zlfm (*Iff p q*) = *disj* (*conj* (*zlfm p*) (*zlfm q*)) (*conj* (*zlfm* (*NOT p*)) (*zlfm* (*NOT q*)))
zlfm (*Lt a*) = (*let* (*c*,*r*) = *zsplit0 a in*
if c=*0 then Lt r else*
if c>*0 then Or* (*Lt* (*CN 0 c* (*Neg* (*Floor* (*Neg r*))))) (*And* (*Eq* (*CN 0 c* (*Neg* (*Floor* (*Neg r*))))) (*Lt* (*Add* (*Floor* (*Neg r*)) *r*)))
else Or (*Gt* (*CN 0* (−*c*) (*Floor*(*Neg r*)))) (*And* (*Eq*(*CN 0* (−*c*) (*Floor*(*Neg*

```
r)))) (Lt (Add (Floor (Neg r)) r))))
  zlfm (Le a) = (let (c,r) = zsplit0 a in
     if c=0 then Le r else
     if c>0 then Or (Le (CN 0 c (Neg (Floor (Neg r))))) (And (Eq (CN 0 c (Neg
(Floor (Neg r))))) (Lt (Add (Floor (Neg r)) r)))
      else Or (Ge (CN 0 (-c) (Floor(Neg r)))) (And (Eq(CN 0 (-c) (Floor(Neg
r)))) (Lt (Add (Floor (Neg r)) r))))
  zlfm (Gt a) = (let (c,r) = zsplit0 a in
     if c=0 then Gt r else
      if c>0 then Or (Gt (CN 0 c (Floor r))) (And (Eq (CN 0 c (Floor r))) (Lt
(Sub (Floor r) r)))
     else Or (Lt (CN 0 (-c) (Neg (Floor r)))) (And (Eq(CN 0 (-c) (Neg (Floor
r)))) (Lt (Sub (Floor r) r))))
  zlfm (Ge a) = (let (c,r) = zsplit0 a in
     if c=0 then Ge r else
      if c>0 then Or (Ge (CN 0 c (Floor r))) (And (Eq (CN 0 c (Floor r))) (Lt
(Sub (Floor r) r)))
     else Or (Le (CN 0 (-c) (Neg (Floor r)))) (And (Eq(CN 0 (-c) (Neg (Floor
r)))) (Lt (Sub (Floor r) r))))
  zlfm (Eq a) = (let (c,r) = zsplit0 a in
              if c=0 then Eq r else
     if c>0 then (And (Eq (CN 0 c (Neg (Floor (Neg r))))) (Eq (Add (Floor (Neg
r)) r)))
      else (And (Eq (CN 0 (-c) (Floor (Neg r)))) (Eq (Add (Floor (Neg r)) r))))
  zlfm (NEq a) = (let (c,r) = zsplit0 a in
              if c=0 then NEq r else
       if c>0 then (Or (NEq (CN 0 c (Neg (Floor (Neg r))))) (NEq (Add (Floor
(Neg r)) r)))
       else (Or (NEq (CN 0 (-c) (Floor (Neg r)))) (NEq (Add (Floor (Neg r))
r))))
  zlfm (Dvd i a) = (if i=0 then zlfm (Eq a)
  else (let (c,r) = zsplit0 a in
              if c=0 then Dvd (abs i) r else
      if c>0 then And (Eq (Sub (Floor r) r)) (Dvd (abs i) (CN 0 c (Floor r)))
     else And (Eq (Sub (Floor r) r)) (Dvd (abs i) (CN 0 (-c) (Neg (Floor r))))))
  zlfm (NDvd i a) = (if i=0 then zlfm (NEq a)
  else (let (c,r) = zsplit0 a in
              if c=0 then NDvd (abs i) r else
      if c>0 then Or (NEq (Sub (Floor r) r)) (NDvd (abs i) (CN 0 c (Floor r)))
       else Or (NEq (Sub (Floor r) r)) (NDvd (abs i) (CN 0 (-c) (Neg (Floor
r))))))
  zlfm (NOT (And p q)) = disj (zlfm (NOT p)) (zlfm (NOT q))
  zlfm (NOT (Or p q)) = conj (zlfm (NOT p)) (zlfm (NOT q))
  zlfm (NOT (Imp p q)) = conj (zlfm p) (zlfm (NOT q))
  zlfm (NOT (Iff p q)) = disj (conj(zlfm p) (zlfm(NOT q))) (conj (zlfm(NOT p))
(zlfm q))
  zlfm (NOT (NOT p)) = zlfm p
  zlfm (NOT T) = F
  zlfm (NOT F) = T
```

zlfm (*NOT* (*Lt a*)) = *zlfm* (*Ge a*)
zlfm (*NOT* (*Le a*)) = *zlfm* (*Gt a*)
zlfm (*NOT* (*Gt a*)) = *zlfm* (*Le a*)
zlfm (*NOT* (*Ge a*)) = *zlfm* (*Lt a*)
zlfm (*NOT* (*Eq a*)) = *zlfm* (*NEq a*)
zlfm (*NOT* (*NEq a*)) = *zlfm* (*Eq a*)
zlfm (*NOT* (*Dvd i a*)) = *zlfm* (*NDvd i a*)
zlfm (*NOT* (*NDvd i a*)) = *zlfm* (*Dvd i a*)
zlfm p = *p* (**hints** *simp add*: *fmsize-pos*)

lemma *split-int-less-real*:
(*real* (*a::int*) < *b*) = (*a* < *floor b* ∨ (*a* = *floor b* ∧ *real* (*floor b*) < *b*))
proof(*auto*)
assume *alb*: *real a* < *b* **and** *agb*: ¬ *a* < *floor b*
from *agb* **have** *floor b* ≤ *a* **by** *simp* **hence** *th*: *b* < *real a* + *1* **by** (*simp only*: *floor-le-eq*)
from *floor-eq*[*OF alb th*] **show** *a*= *floor b* **by** *simp*
next
assume *alb*: *a* < *floor b*
hence *real a* < *real* (*floor b*) **by** *simp*
moreover have *real* (*floor b*) ≤ *b* **by** *simp* **ultimately show** *real a* < *b* **by** *arith*
qed

lemma *split-int-less-real'*:
(*real* (*a::int*) + *b* < *0*) = (*real a* − *real* (*floor*(−*b*)) < *0* ∨ (*real a* − *real* (*floor* (−*b*)) = *0* ∧ *real* (*floor* (−*b*)) + *b* < *0*))
proof−
have (*real a* + *b* <*0*) = (*real a* < −*b*) **by** *arith*
with *split-int-less-real*[**where** *a*=*a* **and** *b*=−*b*] **show** *?thesis* **by** *arith*
qed

lemma *split-int-gt-real'*:
(*real* (*a::int*) + *b* > *0*) = (*real a* + *real* (*floor b*) > *0* ∨ (*real a* + *real* (*floor b*) = *0* ∧ *real* (*floor b*) − *b* < *0*))
proof−
have *th*: (*real a* + *b* >*0*) = (*real* (−*a*) + (−*b*)< *0*) **by** *arith*
show *?thesis* **using** *myless*[*rule-format*, **where** *b*=*real* (*floor b*)]
by (*simp only*:*th split-int-less-real'*[**where** *a*=−*a* **and** *b*=−*b*])
(*simp add*: *ring-simps diff-def*[*symmetric*],*arith*)
qed

lemma *split-int-le-real*:
(*real* (*a::int*) ≤ *b*) = (*a* ≤ *floor b* ∨ (*a* = *floor b* ∧ *real* (*floor b*) < *b*))
proof(*auto*)
assume *alb*: *real a* ≤ *b* **and** *agb*: ¬ *a* ≤ *floor b*
from *alb* **have** *floor* (*real a*) ≤ *floor b* **by** (*simp only*: *floor-mono2*)
hence *a* ≤ *floor b* **by** *simp* **with** *agb* **show** *False* **by** *simp*
next

assume *alb*: $a \le$ *floor b*
hence *real a* $\le$ *real* (*floor b*) **by** (*simp only*: *floor-mono2*)
also have $\ldots \le b$ **by** *simp* **finally show** *real a* $\le b$ **.**
qed

lemma *split-int-le-real'*:
(*real* (*a*::*int*) + $b \le 0$) = (*real a* $-$ *real* (*floor*($-b$)) $\le 0 \vee$ (*real a* $-$ *real* (*floor* ($-b$)) = $0 \wedge$ *real* (*floor* ($-b$)) + $b < 0$))
proof$-$
have (*real a* + $b \le 0$) = (*real a* $\le -b$) **by** *arith*
with *split-int-le-real*[**where** a=a **and** b=$-b$] **show** *?thesis* **by** *arith*
qed

lemma *split-int-ge-real'*:
(*real* (*a*::*int*) + $b \ge 0$) = (*real a* + *real* (*floor b*) $\ge 0 \vee$ (*real a* + *real* (*floor b*) = $0 \wedge$ *real* (*floor b*) $- b < 0$))
proof$-$
have *th*: (*real a* + $b \ge 0$) = (*real* ($-a$) + ($-b$) ≤ 0) **by** *arith*
show *?thesis* **by** (*simp only*: *th split-int-le-real'*[**where** a=$-a$ **and** b=$-b$])
(*simp add*: *ring-simps diff-def*[*symmetric*],*arith*)
qed

lemma *split-int-eq-real*: (*real* (*a*::*int*) = *b*) = (*a* = *floor b* $\wedge$ *b* = *real* (*floor b*))
(**is** *?l* = *?r*)
by *auto*

lemma *split-int-eq-real'*: (*real* (*a*::*int*) + *b* = *0*) = (*a* $-$ *floor* ($-b$) = *0* $\wedge$ *real* (*floor* ($-b$)) + *b* = *0*) (**is** *?l* = *?r*)
proof$-$
have *?l* = (*real a* = $-b$) **by** *arith*
with *split-int-eq-real*[**where** a=a **and** b=$-b$] **show** *?thesis* **by** *simp arith*
qed

lemma *zlfm-I*:
assumes *qfp*: *qfree p*
shows (*Ifm* (*real i* #*bs*) (*zlfm p*) = *Ifm* (*real i*# *bs*) *p*) $\wedge$ *iszlfm* (*zlfm p*) (*real* (*i*::*int*) #*bs*)
(**is** (*?I* (*?l p*) = *?I p*) $\wedge$ *?L* (*?l p*))
using *qfp*
proof(*induct p rule*: *zlfm.induct*)
case (*5 a*)
let *?c* = *fst* (*zsplit0 a*)
let *?r* = *snd* (*zsplit0 a*)
have *spl*: *zsplit0 a* = (*?c*,*?r*) **by** *simp*
from *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
have *Ia*:*Inum* (*real i* # *bs*) *a* = *Inum* (*real i* #*bs*) (*CN 0 ?c ?r*) **and** *nb*: *numbound0 ?r* **by** *auto*
let *?N* = λ *t*. *Inum* (*real i*#*bs*) *t*
have *?c* = *0* $\vee$ (*?c* >*0* $\wedge$ *?c*$\neq$*0*) $\vee$ (*?c*<*0* $\wedge$ *?c*$\neq$*0*) **by** *arith*

moreover
{assume *?c=0* **hence** *?case* **using** *zsplit0-I*[*OF spl*, **where** *x=i* **and** *bs=bs*]
by (*cases ?r*, *simp-all add*: *Let-def split-def*,*case-tac nat*, *simp-all*)**}**
moreover
{assume *cp*: *?c > 0* **and** *cnz*: *?c≠0* **hence** *l*: *?L* (*?l* (*Lt a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Lt a*) = (*real* (*?c ∗ i*) + (*?N ?r*) < *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also have ... = (*?I* (*?l* (*Lt a*))) **apply** (*simp only*: *split-int-less-real′*[**where** *a=?c∗i* **and** *b=?N ?r*]) **by** (*simp add*: *Ia cp cnz Let-def split-def diff-def*)
finally have *?case* **using** *l* **by** *simp***}**
moreover
{assume *cn*: *?c < 0* **and** *cnz*: *?c≠0* **hence** *l*: *?L* (*?l* (*Lt a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Lt a*) = (*real* (*?c ∗ i*) + (*?N ?r*) < *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also from *cn cnz* **have** ... = (*?I* (*?l* (*Lt a*))) **by** (*simp only*: *split-int-less-real′*[**where** *a=?c∗i* **and** *b=?N ?r*]) (*simp add*: *Ia Let-def split-def diff-def*[*symmetric*] *add-ac*, *arith*)
finally have *?case* **using** *l* **by** *simp***}**
ultimately show *?case* **by** *blast*
next
case (*6 a*)
let *?c* = *fst* (*zsplit0 a*)
let *?r* = *snd* (*zsplit0 a*)
have *spl*: *zsplit0 a* = (*?c*,*?r*) **by** *simp*
from *zsplit0-I*[*OF spl*, **where** *x=i* **and** *bs=bs*]
have *Ia*:*Inum* (*real i # bs*) *a* = *Inum* (*real i #bs*) (*CN 0 ?c ?r*) **and** *nb*: *numbound0 ?r* **by** *auto*
let *?N* = *λ t*. *Inum* (*real i#bs*) *t*
have *?c* = *0* ∨ (*?c >0* ∧ *?c≠0*) ∨ (*?c<0* ∧ *?c≠0*) **by** *arith*
moreover
{assume *?c=0* **hence** *?case* **using** *zsplit0-I*[*OF spl*, **where** *x=i* **and** *bs=bs*]
by (*cases ?r*, *simp-all add*: *Let-def split-def*, *case-tac nat*,*simp-all*)**}**
moreover
{assume *cp*: *?c > 0* **and** *cnz*: *?c≠0* **hence** *l*: *?L* (*?l* (*Le a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Le a*) = (*real* (*?c ∗ i*) + (*?N ?r*) ≤ *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also have ... = (*?I* (*?l* (*Le a*))) **by** (*simp only*: *split-int-le-real′*[**where** *a=?c∗i* **and** *b=?N ?r*]) (*simp add*: *Ia cp cnz Let-def split-def diff-def*)
finally have *?case* **using** *l* **by** *simp***}**
moreover
{assume *cn*: *?c < 0* **and** *cnz*: *?c≠0* **hence** *l*: *?L* (*?l* (*Le a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Le a*) = (*real* (*?c ∗ i*) + (*?N ?r*) ≤ *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also from *cn cnz* **have** ... = (*?I* (*?l* (*Le a*))) **by** (*simp only*: *split-int-le-real′*[**where** *a=?c∗i* **and** *b=?N ?r*]) (*simp add*: *Ia Let-def split-def diff-def*[*symmetric*] *add-ac*

,*arith*)
finally have *?case* **using** *l* **by** *simp*}
ultimately show *?case* **by** *blast*
next
case (*7 a*)
let *?c* = *fst* (*zsplit0 a*)
let *?r* = *snd* (*zsplit0 a*)
have *spl*: *zsplit0 a* = (*?c*,*?r*) **by** *simp*
from *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
have *Ia*:*Inum* (*real i* # *bs*) *a* = *Inum* (*real i* #*bs*) (*CN 0 ?c ?r*) **and** *nb*: *numbound0 ?r* **by** *auto*
let *?N* = λ *t*. *Inum* (*real i*#*bs*) *t*
have *?c* = *0* ∨ (*?c* >*0* ∧ *?c*≠*0*) ∨ (*?c*<*0* ∧ *?c*≠*0*) **by** *arith*
moreover
{**assume** *?c*=*0* **hence** *?case* **using** *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
by (*cases ?r*, *simp-all add*: *Let-def split-def*, *case-tac nat*, *simp-all*)}
moreover
{**assume** *cp*: *?c* > *0* **and** *cnz*: *?c*≠*0* **hence** *l*: *?L* (*?l* (*Gt a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Gt a*) = (*real* (*?c* ∗ *i*) + (*?N ?r*) > *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also have . . . = (*?I* (*?l* (*Gt a*))) **by** (*simp only*: *split-int-gt-real′*[**where** *a*=*?c*∗*i* **and** *b*=*?N ?r*]) (*simp add*: *Ia cp cnz Let-def split-def diff-def*)
finally have *?case* **using** *l* **by** *simp*}
moreover
{**assume** *cn*: *?c* < *0* **and** *cnz*: *?c*≠*0* **hence** *l*: *?L* (*?l* (*Gt a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Gt a*) = (*real* (*?c* ∗ *i*) + (*?N ?r*) > *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also from *cn cnz* **have** . . . = (*?I* (*?l* (*Gt a*))) **by** (*simp only*: *split-int-gt-real′*[**where** *a*=*?c*∗*i* **and** *b*=*?N ?r*]) (*simp add*: *Ia Let-def split-def diff-def*[*symmetric*] *add-ac*, *arith*)
finally have *?case* **using** *l* **by** *simp*}
ultimately show *?case* **by** *blast*
next
case (*8 a*)
let *?c* = *fst* (*zsplit0 a*)
let *?r* = *snd* (*zsplit0 a*)
have *spl*: *zsplit0 a* = (*?c*,*?r*) **by** *simp*
from *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
have *Ia*:*Inum* (*real i* # *bs*) *a* = *Inum* (*real i* #*bs*) (*CN 0 ?c ?r*) **and** *nb*: *numbound0 ?r* **by** *auto*
let *?N* = λ *t*. *Inum* (*real i*#*bs*) *t*
have *?c* = *0* ∨ (*?c* >*0* ∧ *?c*≠*0*) ∨ (*?c*<*0* ∧ *?c*≠*0*) **by** *arith*
moreover
{**assume** *?c*=*0* **hence** *?case* **using** *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
by (*cases ?r*, *simp-all add*: *Let-def split-def*, *case-tac nat*, *simp-all*)}
moreover
{**assume** *cp*: *?c* > *0* **and** *cnz*: *?c*≠*0* **hence** *l*: *?L* (*?l* (*Ge a*))

by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Ge a*) = (*real* (*?c* $*$ *i*) + (*?N ?r*) $\geq$ *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also have ... = (*?I* (*?l* (*Ge a*))) **by** (*simp only*: *split-int-ge-real'*[**where** *a*=*?c*$*$*i* **and** *b*=*?N ?r*]) (*simp add*: *Ia cp cnz Let-def split-def diff-def*)
finally have *?case* **using** *l* **by** *simp*}
moreover
{**assume** *cn*: *?c* < *0* **and** *cnz*: *?c*$\neq$*0* **hence** *l*: *?L* (*?l* (*Ge a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Ge a*) = (*real* (*?c* $*$ *i*) + (*?N ?r*) $\geq$ *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also from *cn cnz* **have** ... = (*?I* (*?l* (*Ge a*))) **by** (*simp only*: *split-int-ge-real'*[**where** *a*=*?c*$*$*i* **and** *b*=*?N ?r*]) (*simp add*: *Ia Let-def split-def diff-def*[*symmetric*] *add-ac*, *arith*)
finally have *?case* **using** *l* **by** *simp*}
ultimately show *?case* **by** *blast*
next
case (*9 a*)
let *?c* = *fst* (*zsplit0 a*)
let *?r* = *snd* (*zsplit0 a*)
have *spl*: *zsplit0 a* = (*?c*,*?r*) **by** *simp*
from *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
have *Ia*:*Inum* (*real i* # *bs*) *a* = *Inum* (*real i* #*bs*) (*CN 0 ?c ?r*) **and** *nb*: *numbound0 ?r* **by** *auto*
let *?N* = λ *t*. *Inum* (*real i*#*bs*) *t*
have *?c* = *0* $\vee$ (*?c* >*0* $\wedge$ *?c*$\neq$*0*) $\vee$ (*?c*<*0* $\wedge$ *?c*$\neq$*0*) **by** *arith*
moreover
{**assume** *?c*=*0* **hence** *?case* **using** *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
by (*cases ?r*, *simp-all add*: *Let-def split-def*, *case-tac nat*, *simp-all*)}
moreover
{**assume** *cp*: *?c* > *0* **and** *cnz*: *?c*$\neq$*0* **hence** *l*: *?L* (*?l* (*Eq a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Eq a*) = (*real* (*?c* $*$ *i*) + (*?N ?r*) = *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also have ... = (*?I* (*?l* (*Eq a*))) **using** *cp cnz* **by** (*simp only*: *split-int-eq-real'*[**where** *a*=*?c*$*$*i* **and** *b*=*?N ?r*]) (*simp add*: *Let-def split-def Ia real-of-int-mult*[*symmetric*] *del*: *real-of-int-mult*)
finally have *?case* **using** *l* **by** *simp*}
moreover
{**assume** *cn*: *?c* < *0* **and** *cnz*: *?c*$\neq$*0* **hence** *l*: *?L* (*?l* (*Eq a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Eq a*) = (*real* (*?c* $*$ *i*) + (*?N ?r*) = *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also from *cn cnz* **have** ... = (*?I* (*?l* (*Eq a*))) **by** (*simp only*: *split-int-eq-real'*[**where** *a*=*?c*$*$*i* **and** *b*=*?N ?r*]) (*simp add*: *Let-def split-def Ia real-of-int-mult*[*symmetric*] *del*: *real-of-int-mult*,*arith*)
finally have *?case* **using** *l* **by** *simp*}
ultimately show *?case* **by** *blast*
next

case (*10 a*)
let *?c* = *fst* (*zsplit0 a*)
let *?r* = *snd* (*zsplit0 a*)
have *spl*: *zsplit0 a* = (*?c*,*?r*) **by** *simp*
from *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
have *Ia*:*Inum* (*real i* # *bs*) *a* = *Inum* (*real i* #*bs*) (*CN 0 ?c ?r*) **and** *nb*: *numbound0 ?r* **by** *auto*
let *?N* = λ *t*. *Inum* (*real i*#*bs*) *t*
have *?c* = *0* ∨ (*?c* >*0* ∧ *?c*≠*0*) ∨ (*?c*<*0* ∧ *?c*≠*0*) **by** *arith*
moreover
{**assume** *?c*=*0* **hence** *?case* **using** *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
by (*cases ?r*, *simp-all add*: *Let-def split-def*, *case-tac nat*, *simp-all*)}
moreover
{**assume** *cp*: *?c* > *0* **and** *cnz*: *?c*≠*0* **hence** *l*: *?L* (*?l* (*NEq a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*NEq a*) = (*real* (*?c* ∗ *i*) + (*?N ?r*) ≠ *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also have . . . = (*?I* (*?l* (*NEq a*))) **using** *cp cnz* **by** (*simp only*: *split-int-eq-real'*[**where** *a*=*?c*∗*i* **and** *b*=*?N ?r*]) (*simp add*: *Let-def split-def Ia real-of-int-mult*[*symmetric*] *del*: *real-of-int-mult*)
finally have *?case* **using** *l* **by** *simp*}
moreover
{**assume** *cn*: *?c* < *0* **and** *cnz*: *?c*≠*0* **hence** *l*: *?L* (*?l* (*NEq a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*NEq a*) = (*real* (*?c* ∗ *i*) + (*?N ?r*) ≠ *0*) **using** *Ia* **by** (*simp add*: *Let-def split-def*)
also from *cn cnz* **have** . . . = (*?I* (*?l* (*NEq a*))) **by** (*simp only*: *split-int-eq-real'*[**where** *a*=*?c*∗*i* **and** *b*=*?N ?r*]) (*simp add*: *Let-def split-def Ia real-of-int-mult*[*symmetric*] *del*: *real-of-int-mult*,*arith*)
finally have *?case* **using** *l* **by** *simp*}
ultimately show *?case* **by** *blast*
next
case (*11 j a*)
let *?c* = *fst* (*zsplit0 a*)
let *?r* = *snd* (*zsplit0 a*)
have *spl*: *zsplit0 a* = (*?c*,*?r*) **by** *simp*
from *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
have *Ia*:*Inum* (*real i* # *bs*) *a* = *Inum* (*real i* #*bs*) (*CN 0 ?c ?r*) **and** *nb*: *numbound0 ?r* **by** *auto*
let *?N* = λ *t*. *Inum* (*real i*#*bs*) *t*
have *j*=*0* ∨ (*j*≠*0* ∧ *?c* = *0*) ∨ (*j*≠*0* ∧ *?c* >*0* ∧ *?c*≠*0*) ∨ (*j*≠ *0* ∧ *?c*<*0* ∧ *?c*≠*0*) **by** *arith*
moreover
{**assume** *j*=*0* **hence** *z*: *zlfm* (*Dvd j a*) = (*zlfm* (*Eq a*)) **by** (*simp add*: *Let-def*)
hence *?case* **using** *prems* **by** (*simp del*: *zlfm.simps add*: *rdvd-left-0-eq*)}
moreover
{**assume** *?c*=*0* **and** *j*≠*0* **hence** *?case*
using *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*] *rdvd-abs1*[**where** *d*=*j*]
by (*cases ?r*, *simp-all add*: *Let-def split-def*, *case-tac nat*, *simp-all*)}

moreover
{**assume** *cp*: *?c* > *0* **and** *cnz*: *?c*≠*0* **and** *jnz*: *j*≠*0* **hence** *l*: *?L* (*?l* (*Dvd j a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Dvd j a*) = (*real j rdvd* (*real* (*?c* ∗ *i*) + (*?N ?r*)))
using *Ia* **by** (*simp add*: *Let-def split-def*)
also have ... = (*real* (*abs j*) *rdvd real* (*?c*∗*i*) + (*?N ?r*))
by (*simp only*: *rdvd-abs1*[**where** *d*=*j* **and** *t*=*real* (*?c*∗*i*) + *?N ?r*, *symmetric*])
simp
also have ... = ((*abs j*) *dvd* (*floor* ((*?N ?r*) + *real* (*?c*∗*i*))) ∧
(*real* (*floor* ((*?N ?r*) + *real* (*?c*∗*i*))) = (*real* (*?c*∗*i*) + (*?N ?r*))))
by(*simp only*: *int-rdvd-real*[**where** *i*=*abs j* **and** *x*=*real* (*?c*∗*i*) + (*?N ?r*)])
(*simp only*: *add-ac*)
also have ... = (*?I* (*?l* (*Dvd j a*))) **using** *cp cnz jnz*
by (*simp add*: *Let-def split-def int-rdvd-iff*[*symmetric*]
del: *real-of-int-mult*) (*auto simp add*: *add-ac*)
finally have *?case* **using** *l jnz* **by** *simp* }
moreover
{**assume** *cn*: *?c* < *0* **and** *cnz*: *?c*≠*0* **and** *jnz*: *j*≠*0* **hence** *l*: *?L* (*?l* (*Dvd j a*))
by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*Dvd j a*) = (*real j rdvd* (*real* (*?c* ∗ *i*) + (*?N ?r*)))
using *Ia* **by** (*simp add*: *Let-def split-def*)
also have ... = (*real* (*abs j*) *rdvd real* (*?c*∗*i*) + (*?N ?r*))
by (*simp only*: *rdvd-abs1*[**where** *d*=*j* **and** *t*=*real* (*?c*∗*i*) + *?N ?r*, *symmetric*])
simp
also have ... = ((*abs j*) *dvd* (*floor* ((*?N ?r*) + *real* (*?c*∗*i*))) ∧
(*real* (*floor* ((*?N ?r*) + *real* (*?c*∗*i*))) = (*real* (*?c*∗*i*) + (*?N ?r*))))
by(*simp only*: *int-rdvd-real*[**where** *i*=*abs j* **and** *x*=*real* (*?c*∗*i*) + (*?N ?r*)])
(*simp only*: *add-ac*)
also have ... = (*?I* (*?l* (*Dvd j a*))) **using** *cn cnz jnz*
using *rdvd-minus* [**where** *d*=*abs j* **and** *t*=*real* (*?c*∗*i* + *floor* (*?N ?r*)),
simplified, *symmetric*]
by (*simp add*: *Let-def split-def int-rdvd-iff*[*symmetric*]
del: *real-of-int-mult*) (*auto simp add*: *add-ac*)
finally have *?case* **using** *l jnz* **by** *blast* }
ultimately show *?case* **by** *blast*
next
case (*12 j a*)
let *?c* = *fst* (*zsplit0 a*)
let *?r* = *snd* (*zsplit0 a*)
have *spl*: *zsplit0 a* = (*?c*,*?r*) **by** *simp*
from *zsplit0-I*[*OF spl*, **where** *x*=*i* **and** *bs*=*bs*]
have *Ia*:*Inum* (*real i* # *bs*) *a* = *Inum* (*real i* #*bs*) (*CN 0 ?c ?r*) **and** *nb*:
numbound0 ?r **by** *auto*
let *?N* = λ *t*. *Inum* (*real i*#*bs*) *t*
have *j*=*0* ∨ (*j*≠*0* ∧ *?c* = *0*) ∨ (*j*≠*0* ∧ *?c* >*0* ∧ *?c*≠*0*) ∨ (*j*≠ *0* ∧ *?c*<*0* ∧
?c≠*0*) **by** *arith*
moreover
{**assume** *j*=*0* **hence** *z*: *zlfm* (*NDvd j a*) = (*zlfm* (*NEq a*)) **by** (*simp add*: *Let-def*)

hence *?case* **using** *prems* **by** (*simp del*: *zlfm.simps add*: *rdvd-left-0-eq*)**}**
moreover
{assume *?c=0* **and** *j≠0* **hence** *?case*
using *zsplit0-I*[*OF spl*, **where** *x=i* **and** *bs=bs*] *rdvd-abs1*[**where** *d=j*]
by (*cases ?r*, *simp-all add*: *Let-def split-def*, *case-tac nat*, *simp-all*)**}**
moreover
{assume *cp*: *?c > 0* **and** *cnz*: *?c≠0* **and** *jnz*: *j≠0* **hence** *l*: *?L* (*?l* (*NDvd j a*))

by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*NDvd j a*) = (¬ (*real j rdvd* (*real* (*?c* ∗ *i*) + (*?N ?r*))))
using *Ia* **by** (*simp add*: *Let-def split-def*)
also have ... = (¬ (*real* (*abs j*) *rdvd real* (*?c∗i*) + (*?N ?r*)))
by (*simp only*: *rdvd-abs1*[**where** *d=j* **and** *t=real* (*?c∗i*) + *?N ?r*, *symmetric*])
simp
also have ... = (¬ ((*abs j*) *dvd* (*floor* ((*?N ?r*) + *real* (*?c∗i*))) ∧
(*real* (*floor* ((*?N ?r*) + *real* (*?c∗i*))) = (*real* (*?c∗i*) + (*?N ?r*)))))
by(*simp only*: *int-rdvd-real*[**where** *i=abs j* **and** *x=real* (*?c∗i*) + (*?N ?r*)])
(*simp only*: *add-ac*)
also have ... = (*?I* (*?l* (*NDvd j a*))) **using** *cp cnz jnz*
by (*simp add*: *Let-def split-def int-rdvd-iff*[*symmetric*]
del: *real-of-int-mult*) (*auto simp add*: *add-ac*)
finally have *?case* **using** *l jnz* **by** *simp* **}**
moreover
{assume *cn*: *?c < 0* **and** *cnz*: *?c≠0* **and** *jnz*: *j≠0* **hence** *l*: *?L* (*?l* (*NDvd j a*))

by (*simp add*: *nb Let-def split-def isint-Floor isint-neg*)
have *?I* (*NDvd j a*) = (¬ (*real j rdvd* (*real* (*?c* ∗ *i*) + (*?N ?r*))))
using *Ia* **by** (*simp add*: *Let-def split-def*)
also have ... = (¬ (*real* (*abs j*) *rdvd real* (*?c∗i*) + (*?N ?r*)))
by (*simp only*: *rdvd-abs1*[**where** *d=j* **and** *t=real* (*?c∗i*) + *?N ?r*, *symmetric*])
simp
also have ... = (¬ ((*abs j*) *dvd* (*floor* ((*?N ?r*) + *real* (*?c∗i*))) ∧
(*real* (*floor* ((*?N ?r*) + *real* (*?c∗i*))) = (*real* (*?c∗i*) + (*?N ?r*)))))
by(*simp only*: *int-rdvd-real*[**where** *i=abs j* **and** *x=real* (*?c∗i*) + (*?N ?r*)])
(*simp only*: *add-ac*)
also have ... = (*?I* (*?l* (*NDvd j a*))) **using** *cn cnz jnz*
using *rdvd-minus* [**where** *d=abs j* **and** *t=real* (*?c∗i* + *floor* (*?N ?r*)),
simplified, *symmetric*]
by (*simp add*: *Let-def split-def int-rdvd-iff*[*symmetric*]
del: *real-of-int-mult*) (*auto simp add*: *add-ac*)
finally have *?case* **using** *l jnz* **by** *blast* **}**
ultimately show *?case* **by** *blast*
qed *auto*

plusinf : Virtual substitution of $+\infty$ minusinf: Virtual substitution of $-\infty$ δ Compute lcm $d|$ $Dvd\ d\ \ c*x+t \in p\ \ d\delta$ checks if a given l divides all the ds above

consts
plusinf:: *fm* ⇒ *fm*

minusinf:: *fm* ⇒ *fm*
δ :: *fm* ⇒ *int*
$d\delta$ :: *fm* ⇒ *int* ⇒ *bool*

recdef *minusinf measure size*
minusinf (*And p q*) = *conj* (*minusinf p*) (*minusinf q*)
minusinf (*Or p q*) = *disj* (*minusinf p*) (*minusinf q*)
minusinf (*Eq* (*CN 0 c e*)) = *F*
minusinf (*NEq* (*CN 0 c e*)) = *T*
minusinf (*Lt* (*CN 0 c e*)) = *T*
minusinf (*Le* (*CN 0 c e*)) = *T*
minusinf (*Gt* (*CN 0 c e*)) = *F*
minusinf (*Ge* (*CN 0 c e*)) = *F*
minusinf p = *p*

lemma *minusinf-qfree*: *qfree p* ⟹ *qfree* (*minusinf p*)
by (*induct p rule*: *minusinf.induct*, *auto*)

recdef *plusinf measure size*
plusinf (*And p q*) = *conj* (*plusinf p*) (*plusinf q*)
plusinf (*Or p q*) = *disj* (*plusinf p*) (*plusinf q*)
plusinf (*Eq* (*CN 0 c e*)) = *F*
plusinf (*NEq* (*CN 0 c e*)) = *T*
plusinf (*Lt* (*CN 0 c e*)) = *F*
plusinf (*Le* (*CN 0 c e*)) = *F*
plusinf (*Gt* (*CN 0 c e*)) = *T*
plusinf (*Ge* (*CN 0 c e*)) = *T*
plusinf p = *p*

recdef δ *measure size*
δ (*And p q*) = *ilcm* (δ *p*) (δ *q*)
δ (*Or p q*) = *ilcm* (δ *p*) (δ *q*)
δ (*Dvd i* (*CN 0 c e*)) = *i*
δ (*NDvd i* (*CN 0 c e*)) = *i*
δ *p* = *1*

recdef $d\delta$ *measure size*
$d\delta$ (*And p q*) = (λ *d*. $d\delta$ *p d* ∧ $d\delta$ *q d*)
$d\delta$ (*Or p q*) = (λ *d*. $d\delta$ *p d* ∧ $d\delta$ *q d*)
$d\delta$ (*Dvd i* (*CN 0 c e*)) = (λ *d*. *i dvd d*)
$d\delta$ (*NDvd i* (*CN 0 c e*)) = (λ *d*. *i dvd d*)
$d\delta$ *p* = (λ *d*. *True*)

lemma *delta-mono*:
assumes *lin*: *iszlfm p bs*
and *d*: *d dvd d'*
and *ad*: $d\delta$ *p d*
shows $d\delta$ *p d'*
using *lin ad d*

proof(*induct p rule*: *iszlfm.induct*)
  case (*9 i c e*) **thus** *?case* **using** *d*
    by (*simp add*: *zdvd-trans*[**where** *m*=*i* **and** *n*=*d* **and** *k*=*d'*])
next
  case (*10 i c e*) **thus** *?case* **using** *d*
    by (*simp add*: *zdvd-trans*[**where** *m*=*i* **and** *n*=*d* **and** *k*=*d'*])
qed *simp-all*

lemma δ : **assumes** *lin*:*iszlfm p bs*
  shows *d*δ *p* (δ *p*) $\wedge$ δ *p* >*0*
using *lin*
proof (*induct p rule*: *iszlfm.induct*)
  case (*1 p q*)
  let *?d* = δ (*And p q*)
  from *prems ilcm-pos* **have** *dp*: *?d* >*0* **by** *simp*
  have *d1*: δ *p dvd* δ (*And p q*) **using** *prems* **by** *simp*
   hence *th*: *d*δ *p ?d*
     using *delta-mono prems* **by** (*auto simp del*: *dvd-ilcm-self1*)
  have δ *q dvd* δ (*And p q*) **using** *prems* **by** *simp*
  hence *th'*: *d*δ *q ?d* **using** *delta-mono prems* **by** (*auto simp del*: *dvd-ilcm-self2*)
  from *th th' dp* **show** *?case* **by** *simp*
next
  case (*2 p q*)
  let *?d* = δ (*And p q*)
  from *prems ilcm-pos* **have** *dp*: *?d* >*0* **by** *simp*
  have δ *p dvd* δ (*And p q*) **using** *prems* **by** *simp* **hence** *th*: *d*δ *p ?d* **using** *delta-mono prems*
    by (*auto simp del*: *dvd-ilcm-self1*)
  have δ *q dvd* δ (*And p q*) **using** *prems* **by** *simp* **hence** *th'*: *d*δ *q ?d* **using** *delta-mono prems* **by** (*auto simp del*: *dvd-ilcm-self2*)
  from *th th' dp* **show** *?case* **by** *simp*
qed *simp-all*

lemma *minusinf-inf*:
  assumes *linp*: *iszlfm p* (*a* # *bs*)
  shows $\exists$ (*z*::*int*). $\forall$ *x* < *z*. *Ifm* ((*real x*)#*bs*) (*minusinf p*) = *Ifm* ((*real x*)#*bs*) *p*
  (**is** *?P p* **is** $\exists$ (*z*::*int*). $\forall$ *x* < *z*. *?I x* (*?M p*) = *?I x p*)
using *linp*
proof (*induct p rule*: *minusinf.induct*)
  case (*1 f g*)
  from *prems* **have** *?P f* **by** *simp*
  then obtain *z1* **where** *z1-def*: $\forall$ *x* < *z1*. *?I x* (*?M f*) = *?I x f* **by** *blast*
  from *prems* **have** *?P g* **by** *simp*
  then obtain *z2* **where** *z2-def*: $\forall$ *x* < *z2*. *?I x* (*?M g*) = *?I x g* **by** *blast*
  let *?z* = *min z1 z2*
  from *z1-def z2-def* **have** $\forall$ *x* < *?z*. *?I x* (*?M* (*And f g*)) = *?I x* (*And f g*) **by** *simp*

```
  thus ?case by blast
next
  case (2 f g)   from prems have ?P f by simp
  then obtain z1 where z1-def: ∀ x < z1. ?I x (?M f) = ?I x f by blast
  from prems have ?P g by simp
  then obtain z2 where z2-def: ∀ x < z2. ?I x (?M g) = ?I x g by blast
  let ?z = min z1 z2
  from z1-def z2-def have ∀ x < ?z. ?I x (?M (Or f g)) = ?I x (Or f g) by simp
  thus ?case by blast
next
  case (3 c e)
  from prems have c > 0 by simp hence rcpos: real c > 0 by simp
  from prems have nbe: numbound0 e by simp
  have ∀ x < (floor (− (Inum (y#bs) e) / (real c))). ?I x (?M (Eq (CN 0 c e)))
= ?I x (Eq (CN 0 c e))
  proof (simp add: less-floor-eq , rule allI, rule impI)
    fix x
    assume A: real x + (1::real) ≤ − (Inum (y # bs) e / real c)
    hence th1:real x < − (Inum (y # bs) e / real c) by simp
    with rcpos  have (real c)∗(real  x) < (real c)∗(− (Inum (y # bs) e / real c))
      by (simp only:  real-mult-less-mono2[OF rcpos th1])
    hence real c ∗ real x + Inum (y # bs) e ≠ 0using rcpos  by simp
    thus real c ∗ real x + Inum (real x # bs) e ≠ 0
     using numbound0-I[OF nbe, where b=y and bs=bs and b′=real x]  by simp
  qed
  thus ?case by blast
next
  case (4 c e)
  from prems have c > 0 by simp hence rcpos: real c > 0 by simp
  from prems have nbe: numbound0 e by simp
   have ∀ x < (floor (− (Inum (y#bs) e) / (real c))). ?I x (?M (NEq (CN 0 c
e))) = ?I x (NEq (CN 0 c e))
  proof (simp add: less-floor-eq , rule allI, rule impI)
    fix x
    assume A: real x + (1::real) ≤ − (Inum (y # bs) e / real c)
    hence th1:real x < − (Inum (y # bs) e / real c) by simp
    with rcpos  have (real c)∗(real  x) < (real c)∗(− (Inum (y # bs) e / real c))
      by (simp only:  real-mult-less-mono2[OF rcpos th1])
    hence real c ∗ real x + Inum (y # bs) e ≠ 0using rcpos  by simp
    thus real c ∗ real x + Inum (real x # bs) e ≠ 0
     using numbound0-I[OF nbe, where b=y and bs=bs and b′=real x]  by simp
  qed
  thus ?case by blast
next
  case (5 c e)
  from prems have c > 0 by simp hence rcpos: real c > 0 by simp
  from prems have nbe: numbound0 e by simp
  have ∀ x < (floor (− (Inum (y#bs) e) / (real c))). ?I x (?M (Lt (CN 0 c e)))
= ?I x (Lt (CN 0 c e))
```

proof (*simp add*: *less-floor-eq* , *rule allI*, *rule impI*)
fix *x*
assume *A*: *real x* + (*1*::*real*) ≤ − (*Inum* (*y* # *bs*) *e* / *real c*)
hence *th1*:*real x* < − (*Inum* (*y* # *bs*) *e* / *real c*) **by** *simp*
with *rcpos* **have** (*real c*)∗(*real x*) < (*real c*)∗(− (*Inum* (*y* # *bs*) *e* / *real c*))
by (*simp only*: *real-mult-less-mono2*[*OF rcpos th1*])
thus *real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e* < *0*
using *numbound0-I*[*OF nbe*, **where** *b*=*y* **and** *bs*=*bs* **and** *b′*=*real x*] *rcpos*
by *simp*
qed
thus *?case* **by** *blast*
next
case (*6 c e*)
from *prems* **have** *c* > *0* **by** *simp* **hence** *rcpos*: *real c* > *0* **by** *simp*
from *prems* **have** *nbe*: *numbound0 e* **by** *simp*
have ∀ *x* < (*floor* (− (*Inum* (*y*#*bs*) *e*) / (*real c*))). *?I x* (*?M* (*Le* (*CN 0 c e*)))
= *?I x* (*Le* (*CN 0 c e*))
proof (*simp add*: *less-floor-eq* , *rule allI*, *rule impI*)
fix *x*
assume *A*: *real x* + (*1*::*real*) ≤ − (*Inum* (*y* # *bs*) *e* / *real c*)
hence *th1*:*real x* < − (*Inum* (*y* # *bs*) *e* / *real c*) **by** *simp*
with *rcpos* **have** (*real c*)∗(*real x*) < (*real c*)∗(− (*Inum* (*y* # *bs*) *e* / *real c*))
by (*simp only*: *real-mult-less-mono2*[*OF rcpos th1*])
thus *real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e* ≤ *0*
using *numbound0-I*[*OF nbe*, **where** *b*=*y* **and** *bs*=*bs* **and** *b′*=*real x*] *rcpos*
by *simp*
qed
thus *?case* **by** *blast*
next
case (*7 c e*)
from *prems* **have** *c* > *0* **by** *simp* **hence** *rcpos*: *real c* > *0* **by** *simp*
from *prems* **have** *nbe*: *numbound0 e* **by** *simp*
have ∀ *x* < (*floor* (− (*Inum* (*y*#*bs*) *e*) / (*real c*))). *?I x* (*?M* (*Gt* (*CN 0 c e*)))
= *?I x* (*Gt* (*CN 0 c e*))
proof (*simp add*: *less-floor-eq* , *rule allI*, *rule impI*)
fix *x*
assume *A*: *real x* + (*1*::*real*) ≤ − (*Inum* (*y* # *bs*) *e* / *real c*)
hence *th1*:*real x* < − (*Inum* (*y* # *bs*) *e* / *real c*) **by** *simp*
with *rcpos* **have** (*real c*)∗(*real x*) < (*real c*)∗(− (*Inum* (*y* # *bs*) *e* / *real c*))
by (*simp only*: *real-mult-less-mono2*[*OF rcpos th1*])
thus ¬ (*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*>*0*)
using *numbound0-I*[*OF nbe*, **where** *b*=*y* **and** *bs*=*bs* **and** *b′*=*real x*] *rcpos*
by *simp*
qed
thus *?case* **by** *blast*
next
case (*8 c e*)
from *prems* **have** *c* > *0* **by** *simp* **hence** *rcpos*: *real c* > *0* **by** *simp*
from *prems* **have** *nbe*: *numbound0 e* **by** *simp*

have ∀ *x* < (*floor* (− (*Inum* (*y*#*bs*) *e*) / (*real c*))). *?I x* (*?M* (*Ge* (*CN 0 c e*))) = *?I x* (*Ge* (*CN 0 c e*))
proof (*simp add*: *less-floor-eq* , *rule allI*, *rule impI*)
fix *x*
assume *A*: *real x* + (*1*::*real*) ≤ − (*Inum* (*y* # *bs*) *e* / *real c*)
hence *th1*:*real x* < − (*Inum* (*y* # *bs*) *e* / *real c*) **by** *simp*
with *rcpos* **have** (*real c*)∗(*real x*) < (*real c*)∗(− (*Inum* (*y* # *bs*) *e* / *real c*))
by (*simp only*: *real-mult-less-mono2*[*OF rcpos th1*])
thus ¬ *real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e* ≥ *0*
using *numbound0-I*[*OF nbe*, **where** *b*=*y* **and** *bs*=*bs* **and** *b'*=*real x*] *rcpos* **by** *simp*
qed
thus *?case* **by** *blast*
qed *simp-all*

lemma *minusinf-repeats*:
assumes *d*: *dδ p d* **and** *linp*: *iszlfm p* (*a* # *bs*)
shows *Ifm* ((*real*(*x* − *k*∗*d*))#*bs*) (*minusinf p*) = *Ifm* (*real x* #*bs*) (*minusinf p*)
using *linp d*
proof(*induct p rule*: *iszlfm.induct*)
case (*9 i c e*) **hence** *nbe*: *numbound0 e* **and** *id*: *i dvd d* **by** *simp*+
hence ∃ *k*. *d*=*i*∗*k* **by** (*simp add*: *dvd-def*)
then obtain *di* **where** *di-def*: *d*=*i*∗*di* **by** *blast*
show *?case*
proof(*simp add*: *numbound0-I*[*OF nbe*,**where** *bs*=*bs* **and** *b*=*real x* − *real k* ∗ *real d* **and** *b'*=*real x*] *right-diff-distrib*, *rule iffI*)
assume
real i rdvd real c ∗ *real x* − *real c* ∗ (*real k* ∗ *real d*) + *Inum* (*real x* # *bs*) *e*
(**is** *?ri rdvd ?rc*∗*?rx* − *?rc*∗(*?rk*∗*?rd*) + *?I x e* **is** *?ri rdvd ?rt*)
hence ∃ (*l*::*int*). *?rt* = *?ri* ∗ (*real l*) **by** (*simp add*: *rdvd-def*)
hence ∃ (*l*::*int*). *?rc*∗*?rx*+ *?I x e* = *?ri*∗(*real l*)+*?rc*∗(*?rk* ∗ (*real i*) ∗ (*real di*))
by (*simp add*: *ring-simps di-def*)
hence ∃ (*l*::*int*). *?rc*∗*?rx*+ *?I x e* = *?ri*∗(*real* (*l* + *c*∗*k*∗*di*))
by (*simp add*: *ring-simps*)
hence ∃ (*l*::*int*). *?rc*∗*?rx*+ *?I x e* = *?ri*∗ (*real l*) **by** *blast*
thus *real i rdvd real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e* **using** *rdvd-def* **by** *simp*
next
assume
real i rdvd real c ∗ *real x* + *Inum* (*real x* # *bs*) *e* (**is** *?ri rdvd ?rc*∗*?rx*+*?e*)
hence ∃ (*l*::*int*). *?rc*∗*?rx*+*?e* = *?ri* ∗ (*real l*) **by** (*simp add*: *rdvd-def*)
hence ∃ (*l*::*int*). *?rc*∗*?rx* − *real c* ∗ (*real k* ∗ *real d*) +*?e* = *?ri* ∗ (*real l*) − *real c* ∗ (*real k* ∗ *real d*) **by** *simp*
hence ∃ (*l*::*int*). *?rc*∗*?rx* − *real c* ∗ (*real k* ∗ *real d*) +*?e* = *?ri* ∗ (*real l*) − *real c* ∗ (*real k* ∗ *real i* ∗ *real di*) **by** (*simp add*: *di-def*)
hence ∃ (*l*::*int*). *?rc*∗*?rx* − *real c* ∗ (*real k* ∗ *real d*) +*?e* = *?ri* ∗ (*real* (*l* − *c*∗*k*∗*di*)) **by** (*simp add*: *ring-simps*)
hence ∃ (*l*::*int*). *?rc*∗*?rx* − *real c* ∗ (*real k* ∗ *real d*) +*?e* = *?ri* ∗ (*real l*)

by *blast*
thus *real i rdvd real c ∗ real x − real c ∗* (*real k ∗ real d*) + *Inum* (*real x #
bs*) *e* **using** *rdvd-def* **by** *simp*
qed
next
case (*10 i c e*) **hence** *nbe*: *numbound0 e* **and** *id*: *i dvd d* **by** *simp+*
hence ∃ *k*. *d=i∗k* **by** (*simp add*: *dvd-def*)
then obtain *di* **where** *di-def*: *d=i∗di* **by** *blast*
show *?case*
proof(*simp add*: *numbound0-I*[*OF nbe*,**where** *bs=bs* **and** *b=real x − real k ∗
real d* **and** *b′=real x*] *right-diff-distrib*, *rule iffI*)
assume
real i rdvd real c ∗ real x − real c ∗ (*real k ∗ real d*) + *Inum* (*real x # bs*) *e*
(**is** *?ri rdvd ?rc∗?rx − ?rc∗*(*?rk∗?rd*) + *?I x e* **is** *?ri rdvd ?rt*)
hence ∃ (*l::int*). *?rt = ?ri ∗* (*real l*) **by** (*simp add*: *rdvd-def*)
hence ∃ (*l::int*). *?rc∗?rx+ ?I x e = ?ri∗*(*real l*)+*?rc∗*(*?rk ∗* (*real i*) *∗* (*real
di*))
by (*simp add*: *ring-simps di-def*)
hence ∃ (*l::int*). *?rc∗?rx+ ?I x e = ?ri∗*(*real* (*l + c∗k∗di*))
by (*simp add*: *ring-simps*)
hence ∃ (*l::int*). *?rc∗?rx+ ?I x e = ?ri∗* (*real l*) **by** *blast*
thus *real i rdvd real c ∗ real x + Inum* (*real x # bs*) *e* **using** *rdvd-def* **by**
simp
next
assume
real i rdvd real c ∗ real x + Inum (*real x # bs*) *e* (**is** *?ri rdvd ?rc∗?rx+?e*)
hence ∃ (*l::int*). *?rc∗?rx+?e = ?ri ∗* (*real l*) **by** (*simp add*: *rdvd-def*)
hence ∃ (*l::int*). *?rc∗?rx − real c ∗* (*real k ∗ real d*) +*?e = ?ri ∗* (*real l*) *−
real c ∗* (*real k ∗ real d*) **by** *simp*
hence ∃ (*l::int*). *?rc∗?rx − real c ∗* (*real k ∗ real d*) +*?e = ?ri ∗* (*real l*) *−
real c ∗* (*real k ∗ real i ∗ real di*) **by** (*simp add*: *di-def*)
hence ∃ (*l::int*). *?rc∗?rx − real c ∗* (*real k ∗ real d*) +*?e = ?ri ∗* (*real* (*l −
c∗k∗di*)) **by** (*simp add*: *ring-simps*)
hence ∃ (*l::int*). *?rc∗?rx − real c ∗* (*real k ∗ real d*) +*?e = ?ri ∗* (*real l*)
by *blast*
thus *real i rdvd real c ∗ real x − real c ∗* (*real k ∗ real d*) + *Inum* (*real x #
bs*) *e* **using** *rdvd-def* **by** *simp*
qed
qed (*auto simp add*: *nth-pos2 numbound0-I*[**where** *bs=bs* **and** *b=real*(*x − k∗d*)
and *b′=real x*] *simp del*: *real-of-int-mult real-of-int-diff*)

lemma *minusinf-ex*:
assumes *lin*: *iszlfm p* (*real* (*a::int*) *#bs*)
and *exmi*: ∃ (*x::int*). *Ifm* (*real x#bs*) (*minusinf p*) (**is** ∃ *x*. *?P1 x*)
shows ∃ (*x::int*). *Ifm* (*real x#bs*) *p* (**is** ∃ *x*. *?P x*)
proof−
let *?d = δ p*
from *δ* [*OF lin*] **have** *dpos*: *?d >0* **by** *simp*
from *δ* [*OF lin*] **have** *alld*: *dδ p ?d* **by** *simp*

from *minusinf-repeats*[*OF alld lin*] **have** *th1*:$\forall$ *x k*. *?P1 x* = *?P1* (*x* − (*k* ∗ *?d*))
by *simp*
from *minusinf-inf*[*OF lin*] **have** *th2*:$\exists$ *z*. $\forall$ *x*. *x*<*z* $\longrightarrow$ (*?P x* = *?P1 x*) **by** *blast*
from *minusinfinity* [*OF dpos th1 th2*] *exmi* **show** *?thesis* **by** *blast*
qed

lemma *minusinf-bex*:
assumes *lin*: *iszlfm p* (*real* (*a*::*int*) #*bs*)
shows ($\exists$ (*x*::*int*). *Ifm* (*real x*#*bs*) (*minusinf p*)) =
($\exists$ (*x*::*int*)$\in$ {*1*..δ *p*}. *Ifm* (*real x*#*bs*) (*minusinf p*))
(**is** ($\exists$ *x*. *?P x*) = -)
proof−
let *?d* = δ *p*
from δ [*OF lin*] **have** *dpos*: *?d* >*0* **by** *simp*
from δ [*OF lin*] **have** *alld*: *d*δ *p ?d* **by** *simp*
from *minusinf-repeats*[*OF alld lin*] **have** *th1*:$\forall$ *x k*. *?P x* = *?P* (*x* − (*k* ∗ *?d*))
by *simp*
from *periodic-finite-ex*[*OF dpos th1*] **show** *?thesis* **by** *blast*
qed

lemma *dvd1-eq1*: *x* >*0* $\Longrightarrow$ (*x*::*int*) *dvd 1* = (*x* = *1*) **by** *auto*

consts
aβ :: *fm* $\Rightarrow$ *int* $\Rightarrow$ *fm*
dβ :: *fm* $\Rightarrow$ *int* $\Rightarrow$ *bool*
ζ :: *fm* $\Rightarrow$ *int*
β :: *fm* $\Rightarrow$ *num list*
α :: *fm* $\Rightarrow$ *num list*

recdef *a*β *measure size*
aβ (*And p q*) = (λ *k*. *And* (*a*β *p k*) (*a*β *q k*))
aβ (*Or p q*) = (λ *k*. *Or* (*a*β *p k*) (*a*β *q k*))
aβ (*Eq* (*CN 0 c e*)) = (λ *k*. *Eq* (*CN 0 1* (*Mul* (*k div c*) *e*)))
aβ (*NEq* (*CN 0 c e*)) = (λ *k*. *NEq* (*CN 0 1* (*Mul* (*k div c*) *e*)))
aβ (*Lt* (*CN 0 c e*)) = (λ *k*. *Lt* (*CN 0 1* (*Mul* (*k div c*) *e*)))
aβ (*Le* (*CN 0 c e*)) = (λ *k*. *Le* (*CN 0 1* (*Mul* (*k div c*) *e*)))
aβ (*Gt* (*CN 0 c e*)) = (λ *k*. *Gt* (*CN 0 1* (*Mul* (*k div c*) *e*)))
aβ (*Ge* (*CN 0 c e*)) = (λ *k*. *Ge* (*CN 0 1* (*Mul* (*k div c*) *e*)))
aβ (*Dvd i* (*CN 0 c e*)) =(λ *k*. *Dvd* ((*k div c*)∗*i*) (*CN 0 1* (*Mul* (*k div c*) *e*)))
aβ (*NDvd i* (*CN 0 c e*))=(λ *k*. *NDvd* ((*k div c*)∗*i*) (*CN 0 1* (*Mul* (*k div c*) *e*)))
aβ *p* = (λ *k*. *p*)

recdef *d*β *measure size*
dβ (*And p q*) = (λ *k*. (*d*β *p k*) $\wedge$ (*d*β *q k*))
dβ (*Or p q*) = (λ *k*. (*d*β *p k*) $\wedge$ (*d*β *q k*))
dβ (*Eq* (*CN 0 c e*)) = (λ *k*. *c dvd k*)
dβ (*NEq* (*CN 0 c e*)) = (λ *k*. *c dvd k*)
dβ (*Lt* (*CN 0 c e*)) = (λ *k*. *c dvd k*)
dβ (*Le* (*CN 0 c e*)) = (λ *k*. *c dvd k*)

$d\beta$ $(Gt\ \ (CN\ 0\ c\ e)) = (\lambda\ k.\ c\ dvd\ k)$
$d\beta$ $(Ge\ \ (CN\ 0\ c\ e)) = (\lambda\ k.\ c\ dvd\ k)$
$d\beta$ $(Dvd\ i\ (CN\ 0\ c\ e)) =(\lambda\ k.\ c\ dvd\ k)$
$d\beta$ $(NDvd\ i\ (CN\ 0\ c\ e))=(\lambda\ k.\ c\ dvd\ k)$
$d\beta\ p = (\lambda\ k.\ True)$

recdef ζ *measure size*
$\zeta\ (And\ p\ q) = ilcm\ (\zeta\ p)\ (\zeta\ q)$
$\zeta\ (Or\ p\ q) = ilcm\ (\zeta\ p)\ (\zeta\ q)$
$\zeta\ (Eq\ \ (CN\ 0\ c\ e)) = c$
$\zeta\ (NEq\ (CN\ 0\ c\ e)) = c$
$\zeta\ (Lt\ \ (CN\ 0\ c\ e)) = c$
$\zeta\ (Le\ \ (CN\ 0\ c\ e)) = c$
$\zeta\ (Gt\ \ (CN\ 0\ c\ e)) = c$
$\zeta\ (Ge\ \ (CN\ 0\ c\ e)) = c$
$\zeta\ (Dvd\ i\ (CN\ 0\ c\ e)) = c$
$\zeta\ (NDvd\ i\ (CN\ 0\ c\ e))= c$
$\zeta\ p = 1$

recdef β *measure size*
$\beta\ (And\ p\ q) = (\beta\ p\ @\ \beta\ q)$
$\beta\ (Or\ p\ q) = (\beta\ p\ @\ \beta\ q)$
$\beta\ (Eq\ \ (CN\ 0\ c\ e)) = [Sub\ (C\ -1)\ e]$
$\beta\ (NEq\ (CN\ 0\ c\ e)) = [Neg\ e]$
$\beta\ (Lt\ \ (CN\ 0\ c\ e)) = []$
$\beta\ (Le\ \ (CN\ 0\ c\ e)) = []$
$\beta\ (Gt\ \ (CN\ 0\ c\ e)) = [Neg\ e]$
$\beta\ (Ge\ \ (CN\ 0\ c\ e)) = [Sub\ (C\ -1)\ e]$
$\beta\ p = []$

recdef α *measure size*
$\alpha\ (And\ p\ q) = (\alpha\ p\ @\ \alpha\ q)$
$\alpha\ (Or\ p\ q) = (\alpha\ p\ @\ \alpha\ q)$
$\alpha\ (Eq\ \ (CN\ 0\ c\ e)) = [Add\ (C\ -1)\ e]$
$\alpha\ (NEq\ (CN\ 0\ c\ e)) = [e]$
$\alpha\ (Lt\ \ (CN\ 0\ c\ e)) = [e]$
$\alpha\ (Le\ \ (CN\ 0\ c\ e)) = [Add\ (C\ -1)\ e]$
$\alpha\ (Gt\ \ (CN\ 0\ c\ e)) = []$
$\alpha\ (Ge\ \ (CN\ 0\ c\ e)) = []$
$\alpha\ p = []$
consts $mirror :: fm \Rightarrow fm$
recdef *mirror measure size*
$mirror\ (And\ p\ q) = And\ (mirror\ p)\ (mirror\ q)$
$mirror\ (Or\ p\ q) = Or\ (mirror\ p)\ (mirror\ q)$
$mirror\ (Eq\ \ (CN\ 0\ c\ e)) = Eq\ (CN\ 0\ c\ (Neg\ e))$
$mirror\ (NEq\ (CN\ 0\ c\ e)) = NEq\ (CN\ 0\ c\ (Neg\ e))$
$mirror\ (Lt\ \ (CN\ 0\ c\ e)) = Gt\ (CN\ 0\ c\ (Neg\ e))$
$mirror\ (Le\ \ (CN\ 0\ c\ e)) = Ge\ (CN\ 0\ c\ (Neg\ e))$
$mirror\ (Gt\ \ (CN\ 0\ c\ e)) = Lt\ (CN\ 0\ c\ (Neg\ e))$

mirror (*Ge* (*CN 0 c e*)) = *Le* (*CN 0 c* (*Neg e*))
mirror (*Dvd i* (*CN 0 c e*)) = *Dvd i* (*CN 0 c* (*Neg e*))
mirror (*NDvd i* (*CN 0 c e*)) = *NDvd i* (*CN 0 c* (*Neg e*))
mirror p = *p*

lemma *mirror*$\alpha\beta$:
assumes *lp*: *iszlfm p* (*a*#*bs*)
shows (*Inum* (*real* (*i*::*int*)#*bs*)) ' *set* (α *p*) = (*Inum* (*real i*#*bs*)) ' *set* (β (*mirror p*))
using *lp*
by (*induct p rule*: *mirror.induct*, *auto*)

lemma *mirror*:
assumes *lp*: *iszlfm p* (*a*#*bs*)
shows *Ifm* (*real* (*x*::*int*)#*bs*) (*mirror p*) = *Ifm* (*real* (− *x*)#*bs*) *p*
using *lp*
proof(*induct p rule*: *iszlfm.induct*)
case (*9 j c e*)
have *th*: (*real j rdvd real c* ∗ *real x* − *Inum* (*real x* # *bs*) *e*) =
(*real j rdvd* − (*real c* ∗ *real x* − *Inum* (*real x* # *bs*) *e*))
by (*simp only*: *rdvd-minus*[*symmetric*])
from *prems* **show** *?case*
by (*simp add*: *ring-simps th*[*simplified ring-simps*]
numbound0-I[**where** *bs*=*bs* **and** *b'*=*real x* **and** *b*=− *real x*])
next
case (*10 j c e*)
have *th*: (*real j rdvd real c* ∗ *real x* − *Inum* (*real x* # *bs*) *e*) =
(*real j rdvd* − (*real c* ∗ *real x* − *Inum* (*real x* # *bs*) *e*))
by (*simp only*: *rdvd-minus*[*symmetric*])
from *prems* **show** *?case*
by (*simp add*: *ring-simps th*[*simplified ring-simps*]
numbound0-I[**where** *bs*=*bs* **and** *b'*=*real x* **and** *b*=− *real x*])
qed (*auto simp add*: *numbound0-I*[**where** *bs*=*bs* **and** *b*=*real x* **and** *b'*=− *real x*] *nth-pos2*)

lemma *mirror-l*: *iszlfm p* (*a*#*bs*) ⟹ *iszlfm* (*mirror p*) (*a*#*bs*)
by (*induct p rule*: *mirror.induct*, *auto simp add*: *isint-neg*)

lemma *mirror-d*β: *iszlfm p* (*a*#*bs*) ∧ *d*β *p 1*
⟹ *iszlfm* (*mirror p*) (*a*#*bs*) ∧ *d*β (*mirror p*) *1*
by (*induct p rule*: *mirror.induct*, *auto simp add*: *isint-neg*)

lemma *mirror-*δ: *iszlfm p* (*a*#*bs*) ⟹ δ (*mirror p*) = δ *p*
by (*induct p rule*: *mirror.induct*,*auto*)

lemma *mirror-ex*:
assumes *lp*: *iszlfm p* (*real* (*i*::*int*)#*bs*)
shows (∃ (*x*::*int*). *Ifm* (*real x*#*bs*) (*mirror p*)) = (∃ (*x*::*int*). *Ifm* (*real x*#*bs*)

p)
  (**is** (∃ *x*. *?I x ?mp*) = (∃ *x*. *?I x p*))
proof(*auto*)
  fix *x* **assume** *?I x ?mp* **hence** *?I* (− *x*) *p* **using** *mirror*[*OF lp*] **by** *blast*
  thus ∃ *x*. *?I x p* **by** *blast*
next
  fix *x* **assume** *?I x p* **hence** *?I* (− *x*) *?mp*
    using *mirror*[*OF lp*, **where** *x*=− *x*, *symmetric*] **by** *auto*
  thus ∃ *x*. *?I x ?mp* **by** *blast*
qed

lemma *β-numbound0*: **assumes** *lp*: *iszlfm p bs*
  shows ∀ *b*∈ *set* (*β p*). *numbound0 b*
  using *lp* **by** (*induct p rule*: *β.induct*,*auto*)

lemma *dβ-mono*:
  assumes *linp*: *iszlfm p* (*a* #*bs*)
  and *dr*: *dβ p l*
  and *d*: *l dvd l′*
  shows *dβ p l′*
using *dr linp zdvd-trans*[**where** *n*=*l* **and** *k*=*l′*, *simplified d*]
by (*induct p rule*: *iszlfm.induct*) *simp-all*

lemma *α-l*: **assumes** *lp*: *iszlfm p* (*a*#*bs*)
  shows ∀ *b*∈ *set* (*α p*). *numbound0 b* ∧ *isint b* (*a*#*bs*)
using *lp*
by(*induct p rule*: *α.induct*, *auto simp add*: *isint-add isint-c*)

lemma *ζ*:
  assumes *linp*: *iszlfm p* (*a* #*bs*)
  shows *ζ p* > *0* ∧ *dβ p* (*ζ p*)
using *linp*
proof(*induct p rule*: *iszlfm.induct*)
  case (*1 p q*)
  from *prems* **have** *dl1*: *ζ p dvd ilcm* (*ζ p*) (*ζ q*) **by** *simp*
  from *prems* **have** *dl2*: *ζ q dvd ilcm* (*ζ p*) (*ζ q*) **by** *simp*
  from *prems dβ-mono*[**where** *p* = *p* **and** *l*=*ζ p* **and** *l′*=*ilcm* (*ζ p*) (*ζ q*)]
    dβ-mono[**where** *p* = *q* **and** *l*=*ζ q* **and** *l′*=*ilcm* (*ζ p*) (*ζ q*)]
    dl1 dl2 **show** *?case* **by** (*auto simp add*: *ilcm-pos*)
next
  case (*2 p q*)
  from *prems* **have** *dl1*: *ζ p dvd ilcm* (*ζ p*) (*ζ q*) **by** *simp*
  from *prems* **have** *dl2*: *ζ q dvd ilcm* (*ζ p*) (*ζ q*) **by** *simp*
  from *prems dβ-mono*[**where** *p* = *p* **and** *l*=*ζ p* **and** *l′*=*ilcm* (*ζ p*) (*ζ q*)]
    dβ-mono[**where** *p* = *q* **and** *l*=*ζ q* **and** *l′*=*ilcm* (*ζ p*) (*ζ q*)]
    dl1 dl2 **show** *?case* **by** (*auto simp add*: *ilcm-pos*)
qed (*auto simp add*: *ilcm-pos*)

lemma *aβ*: **assumes** *linp*: *iszlfm p* (*a* #*bs*) **and** *d*: *dβ p l* **and** *lp*: *l* > *0*

shows *iszlfm* (*a*β *p l*) (*a* #*bs*) ∧ *d*β (*a*β *p l*) *1* ∧ (*Ifm* (*real* (*l* ∗ *x*) #*bs*) (*a*β *p l*) = *Ifm* ((*real x*)#*bs*) *p*)
using *linp d*
proof (*induct p rule*: *iszlfm.induct*)
case (*5 c e*) **hence** *cp*: *c*>*0* **and** *be*: *numbound0 e* **and** *ei*:*isint e* (*a*#*bs*) **and** *d'*: *c dvd l* **by** *simp*+
from *lp cp* **have** *clel*: *c*≤*l* **by** (*simp add*: *zdvd-imp-le* [*OF d' lp*])
from *cp* **have** *cnz*: *c* ≠ *0* **by** *simp*
have *c div c*≤ *l div c*
by (*simp add*: *zdiv-mono1*[*OF clel cp*])
then have *ldcp*:*0* < *l div c*
by (*simp add*: *zdiv-self*[*OF cnz*])
have *c* ∗ (*l div c*) = *c*∗ (*l div c*) + *l mod c* **using** *d' zdvd-iff-zmod-eq-0*[**where** *m*=*c* **and** *n*=*l*] **by** *simp*
hence *cl*:*c* ∗ (*l div c*) =*l* **using** *zmod-zdiv-equality*[**where** *a*=*l* **and** *b*=*c*, *symmetric*]
by *simp*
hence (*real l* ∗ *real x* + *real* (*l div c*) ∗ *Inum* (*real x* # *bs*) *e* < (*0*::*real*)) =
(*real* (*c* ∗ (*l div c*)) ∗ *real x* + *real* (*l div c*) ∗ *Inum* (*real x* # *bs*) *e* < *0*)
by *simp*
also have . . . = (*real* (*l div c*) ∗ (*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*) < (*real* (*l div c*)) ∗ *0*) **by** (*simp add*: *ring-simps*)
also have . . . = (*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e* < *0*)
using *mult-less-0-iff* [**where** *a*=*real* (*l div c*) **and** *b*=*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*] *ldcp* **by** *simp*
finally show *?case* **using** *numbound0-I*[*OF be*,**where** *b*=*real* (*l* ∗ *x*) **and** *b'*=*real x* **and** *bs*=*bs*] *be isint-Mul*[*OF ei*] **by** *simp*
next
case (*6 c e*) **hence** *cp*: *c*>*0* **and** *be*: *numbound0 e* **and** *ei*:*isint e* (*a*#*bs*) **and** *d'*: *c dvd l* **by** *simp*+
from *lp cp* **have** *clel*: *c*≤*l* **by** (*simp add*: *zdvd-imp-le* [*OF d' lp*])
from *cp* **have** *cnz*: *c* ≠ *0* **by** *simp*
have *c div c*≤ *l div c*
by (*simp add*: *zdiv-mono1*[*OF clel cp*])
then have *ldcp*:*0* < *l div c*
by (*simp add*: *zdiv-self*[*OF cnz*])
have *c* ∗ (*l div c*) = *c*∗ (*l div c*) + *l mod c* **using** *d' zdvd-iff-zmod-eq-0*[**where** *m*=*c* **and** *n*=*l*] **by** *simp*
hence *cl*:*c* ∗ (*l div c*) =*l* **using** *zmod-zdiv-equality*[**where** *a*=*l* **and** *b*=*c*, *symmetric*]
by *simp*
hence (*real l* ∗ *real x* + *real* (*l div c*) ∗ *Inum* (*real x* # *bs*) *e* ≤ (*0*::*real*)) =
(*real* (*c* ∗ (*l div c*)) ∗ *real x* + *real* (*l div c*) ∗ *Inum* (*real x* # *bs*) *e* ≤ *0*)
by *simp*
also have . . . = (*real* (*l div c*) ∗ (*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*) ≤ (*real* (*l div c*)) ∗ *0*) **by** (*simp add*: *ring-simps*)
also have . . . = (*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e* ≤ *0*)
using *mult-le-0-iff* [**where** *a*=*real* (*l div c*) **and** *b*=*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*] *ldcp* **by** *simp*

finally show *?case* **using** *numbound0-I*[*OF be*,**where** b=*real* ($l * x$) **and** b'=*real* x **and** *bs*=*bs*] *be* *isint-Mul*[*OF ei*] **by** *simp*
next
case (*7 c e*) **hence** *cp*: $c>0$ **and** *be*: *numbound0 e* **and** *ei*:*isint e* (*a*#*bs*) **and** d': *c dvd l* **by** *simp*+
from *lp cp* **have** *clel*: $c \leq l$ **by** (*simp add*: *zdvd-imp-le* [*OF* d' *lp*])
from *cp* **have** *cnz*: $c \neq 0$ **by** *simp*
have *c div c*$\leq$ *l div c*
by (*simp add*: *zdiv-mono1*[*OF clel cp*])
then have *ldcp*:$0 <$ *l div c*
by (*simp add*: *zdiv-self*[*OF cnz*])
have $c *$ (*l div c*) = $c*$ (*l div c*) + *l mod c* **using** d' *zdvd-iff-zmod-eq-0*[**where** m=c **and** n=l] **by** *simp*
hence *cl*:$c *$ (*l div c*) =l **using** *zmod-zdiv-equality*[**where** a=l **and** b=c, *symmetric*]
by *simp*
hence (*real* $l *$ *real* x + *real* (*l div c*) $*$ *Inum* (*real* x # *bs*) $e >$ (*0*::*real*)) =
(*real* ($c *$ (*l div c*)) $*$ *real* x + *real* (*l div c*) $*$ *Inum* (*real* x # *bs*) $e > 0$)
by *simp*
also have ... = (*real* (*l div c*) $*$ (*real* $c *$ *real* x + *Inum* (*real* x # *bs*) e) $>$ (*real* (*l div c*)) $* 0$) **by** (*simp add*: *ring-simps*)
also have ... = (*real* $c *$ *real* x + *Inum* (*real* x # *bs*) $e > 0$)
using *zero-less-mult-iff* [**where** a=*real* (*l div c*) **and** b=*real* $c *$ *real* x + *Inum* (*real* x # *bs*) e] *ldcp* **by** *simp*
finally show *?case* **using** *numbound0-I*[*OF be*,**where** b=*real* ($l * x$) **and** b'=*real* x **and** *bs*=*bs*] *be* *isint-Mul*[*OF ei*] **by** *simp*
next
case (*8 c e*) **hence** *cp*: $c>0$ **and** *be*: *numbound0 e* **and** *ei*:*isint e* (*a*#*bs*) **and** d': *c dvd l* **by** *simp*+
from *lp cp* **have** *clel*: $c \leq l$ **by** (*simp add*: *zdvd-imp-le* [*OF* d' *lp*])
from *cp* **have** *cnz*: $c \neq 0$ **by** *simp*
have *c div c*$\leq$ *l div c*
by (*simp add*: *zdiv-mono1*[*OF clel cp*])
then have *ldcp*:$0 <$ *l div c*
by (*simp add*: *zdiv-self*[*OF cnz*])
have $c *$ (*l div c*) = $c*$ (*l div c*) + *l mod c* **using** d' *zdvd-iff-zmod-eq-0*[**where** m=c **and** n=l] **by** *simp*
hence *cl*:$c *$ (*l div c*) =l **using** *zmod-zdiv-equality*[**where** a=l **and** b=c, *symmetric*]
by *simp*
hence (*real* $l *$ *real* x + *real* (*l div c*) $*$ *Inum* (*real* x # *bs*) $e \geq$ (*0*::*real*)) =
(*real* ($c *$ (*l div c*)) $*$ *real* x + *real* (*l div c*) $*$ *Inum* (*real* x # *bs*) $e \geq 0$)
by *simp*
also have ... = (*real* (*l div c*) $*$ (*real* $c *$ *real* x + *Inum* (*real* x # *bs*) e) $\geq$ (*real* (*l div c*)) $* 0$) **by** (*simp add*: *ring-simps*)
also have ... = (*real* $c *$ *real* x + *Inum* (*real* x # *bs*) $e \geq 0$)
using *zero-le-mult-iff* [**where** a=*real* (*l div c*) **and** b=*real* $c *$ *real* x + *Inum* (*real* x # *bs*) e] *ldcp* **by** *simp*
finally show *?case* **using** *numbound0-I*[*OF be*,**where** b=*real* ($l * x$) **and** b'=*real*

x **and** *bs=bs*] *be* *isint-Mul*[*OF ei*] **by** *simp*
next
case (*3 c e*) **hence** *cp*: *c>0* **and** *be*: *numbound0 e* **and** *ei*:*isint e* (*a#bs*) **and** *d'*: *c dvd l* **by** *simp+*
from *lp cp* **have** *clel*: *c≤l* **by** (*simp add*: *zdvd-imp-le* [*OF d' lp*])
from *cp* **have** *cnz*: *c ≠ 0* **by** *simp*
have *c div c≤ l div c*
by (*simp add*: *zdiv-mono1*[*OF clel cp*])
then have *ldcp*:*0 < l div c*
by (*simp add*: *zdiv-self*[*OF cnz*])
have *c * (l div c) = c* (l div c) + l mod c* **using** *d' zdvd-iff-zmod-eq-0*[**where** *m=c* **and** *n=l*] **by** *simp*
hence *cl*:*c * (l div c) =l* **using** *zmod-zdiv-equality*[**where** *a=l* **and** *b=c*, *symmetric*]
by *simp*
hence (*real l * real x + real (l div c) * Inum (real x # bs) e = (0::real)*) =
(*real (c * (l div c)) * real x + real (l div c) * Inum (real x # bs) e = 0*)
by *simp*
also have . . . = (*real (l div c) * (real c * real x + Inum (real x # bs) e) = (real (l div c)) * 0*) **by** (*simp add*: *ring-simps*)
also have . . . = (*real c * real x + Inum (real x # bs) e = 0*)
using *mult-eq-0-iff* [**where** *a=real (l div c)* **and** *b=real c * real x + Inum (real x # bs) e*] *ldcp* **by** *simp*
finally show *?case* **using** *numbound0-I*[*OF be*,**where** *b=real (l * x)* **and** *b'=real x* **and** *bs=bs*] *be* *isint-Mul*[*OF ei*] **by** *simp*
next
case (*4 c e*) **hence** *cp*: *c>0* **and** *be*: *numbound0 e* **and** *ei*:*isint e* (*a#bs*) **and** *d'*: *c dvd l* **by** *simp+*
from *lp cp* **have** *clel*: *c≤l* **by** (*simp add*: *zdvd-imp-le* [*OF d' lp*])
from *cp* **have** *cnz*: *c ≠ 0* **by** *simp*
have *c div c≤ l div c*
by (*simp add*: *zdiv-mono1*[*OF clel cp*])
then have *ldcp*:*0 < l div c*
by (*simp add*: *zdiv-self*[*OF cnz*])
have *c * (l div c) = c* (l div c) + l mod c* **using** *d' zdvd-iff-zmod-eq-0*[**where** *m=c* **and** *n=l*] **by** *simp*
hence *cl*:*c * (l div c) =l* **using** *zmod-zdiv-equality*[**where** *a=l* **and** *b=c*, *symmetric*]
by *simp*
hence (*real l * real x + real (l div c) * Inum (real x # bs) e ≠ (0::real)*) =
(*real (c * (l div c)) * real x + real (l div c) * Inum (real x # bs) e ≠ 0*)
by *simp*
also have . . . = (*real (l div c) * (real c * real x + Inum (real x # bs) e) ≠ (real (l div c)) * 0*) **by** (*simp add*: *ring-simps*)
also have . . . = (*real c * real x + Inum (real x # bs) e ≠ 0*)
using *zero-le-mult-iff* [**where** *a=real (l div c)* **and** *b=real c * real x + Inum (real x # bs) e*] *ldcp* **by** *simp*
finally show *?case* **using** *numbound0-I*[*OF be*,**where** *b=real (l * x)* **and** *b'=real x* **and** *bs=bs*] *be* *isint-Mul*[*OF ei*] **by** *simp*

next

case $(9\ j\ c\ e)$ **hence** *cp*: $c>0$ **and** *be*: *numbound0 e* **and** *ei*:*isint e* $(a\#bs)$ **and** *jp*: $j > 0$ **and** d': *c dvd l* **by** *simp+*

from *lp cp* **have** *clel*: $c\leq l$ **by** (*simp add*: *zdvd-imp-le* [*OF d′ lp*])

from *cp* **have** *cnz*: $c \neq 0$ **by** *simp*

have *c div c*$\leq$ *l div c*

by (*simp add*: *zdiv-mono1*[*OF clel cp*])

then have *ldcp*:$0 <$ *l div c*

by (*simp add*: *zdiv-self*[*OF cnz*])

have $c * (l\ div\ c) = c* (l\ div\ c) + l\ mod\ c$ **using** d' *zdvd-iff-zmod-eq-0*[**where** $m=c$ **and** $n=l$] **by** *simp*

hence *cl*:$c * (l\ div\ c) = l$ **using** *zmod-zdiv-equality*[**where** $a=l$ **and** $b=c$, *symmetric*]

by *simp*

hence $(\exists\ (k::int).\ real\ l * real\ x + real\ (l\ div\ c) * Inum\ (real\ x\ \#\ bs)\ e = (real\ (l\ div\ c) * real\ j) * real\ k) = (\exists\ (k::int).\ real\ (c * (l\ div\ c)) * real\ x + real\ (l\ div\ c) * Inum\ (real\ x\ \#\ bs)\ e = (real\ (l\ div\ c) * real\ j) * real\ k)$ **by** *simp*

also have $\ldots = (\exists\ (k::int).\ real\ (l\ div\ c) * (real\ c * real\ x + Inum\ (real\ x\ \#\ bs)\ e - real\ j * real\ k) = real\ (l\ div\ c)*0)$ **by** (*simp add*: *ring-simps*)

also have $\ldots = (\exists\ (k::int).\ real\ c * real\ x + Inum\ (real\ x\ \#\ bs)\ e - real\ j * real\ k = 0)$

using *zero-le-mult-iff* [**where** $a=real\ (l\ div\ c)$ **and** $b=real\ c * real\ x + Inum\ (real\ x\ \#\ bs)\ e - real\ j * real\ k$] *ldcp* **by** *simp*

also have $\ldots = (\exists\ (k::int).\ real\ c * real\ x + Inum\ (real\ x\ \#\ bs)\ e = real\ j * real\ k)$ **by** *simp*

finally show *?case* **using** *numbound0-I*[*OF be*,**where** $b=real\ (l * x)$ **and** $b'=real\ x$ **and** $bs=bs$] *rdvd-def be isint-Mul*[*OF ei*] *mult-strict-mono*[*OF ldcp jp ldcp*] **by** *simp*

next

case $(10\ j\ c\ e)$ **hence** *cp*: $c>0$ **and** *be*: *numbound0 e* **and** *ei*:*isint e* $(a\#bs)$ **and** *jp*: $j > 0$ **and** d': *c dvd l* **by** *simp+*

from *lp cp* **have** *clel*: $c\leq l$ **by** (*simp add*: *zdvd-imp-le* [*OF d′ lp*])

from *cp* **have** *cnz*: $c \neq 0$ **by** *simp*

have *c div c*$\leq$ *l div c*

by (*simp add*: *zdiv-mono1*[*OF clel cp*])

then have *ldcp*:$0 <$ *l div c*

by (*simp add*: *zdiv-self*[*OF cnz*])

have $c * (l\ div\ c) = c* (l\ div\ c) + l\ mod\ c$ **using** d' *zdvd-iff-zmod-eq-0*[**where** $m=c$ **and** $n=l$] **by** *simp*

hence *cl*:$c * (l\ div\ c) = l$ **using** *zmod-zdiv-equality*[**where** $a=l$ **and** $b=c$, *symmetric*]

by *simp*

hence $(\exists\ (k::int).\ real\ l * real\ x + real\ (l\ div\ c) * Inum\ (real\ x\ \#\ bs)\ e = (real\ (l\ div\ c) * real\ j) * real\ k) = (\exists\ (k::int).\ real\ (c * (l\ div\ c)) * real\ x + real\ (l\ div\ c) * Inum\ (real\ x\ \#\ bs)\ e = (real\ (l\ div\ c) * real\ j) * real\ k)$ **by** *simp*

also have $\ldots = (\exists\ (k::int).\ real\ (l\ div\ c) * (real\ c * real\ x + Inum\ (real\ x\ \#\ bs)\ e - real\ j * real\ k) = real\ (l\ div\ c)*0)$ **by** (*simp add*: *ring-simps*)

also have $\ldots = (\exists\ (k::int).\ real\ c * real\ x + Inum\ (real\ x\ \#\ bs)\ e - real\ j * real\ k = 0)$

using *zero-le-mult-iff* [**where** *a=real* (*l div c*) **and** *b=real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e* − *real j* ∗ *real k*] *ldcp* **by** *simp*
also have ... = (∃ (*k::int*). *real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e* = *real j* ∗ *real k*) **by** *simp*
finally show *?case* **using** *numbound0-I*[*OF be*,**where** *b=real* (*l* ∗ *x*) **and** *b′=real x* **and** *bs=bs*] *rdvd-def be isint-Mul*[*OF ei*] *mult-strict-mono*[*OF ldcp jp ldcp*] **by** *simp*
qed (*simp-all add*: *nth-pos2 numbound0-I*[**where** *bs=bs* **and** *b=real* (*l* ∗ *x*) **and** *b′=real x*] *isint-Mul del*: *real-of-int-mult*)

lemma *aβ-ex*: **assumes** *linp*: *iszlfm p* (*a#bs*) **and** *d*: *dβ p l* **and** *lp*: *l>0*
shows (∃ *x*. *l dvd x* ∧ *Ifm* (*real x #bs*) (*aβ p l*)) = (∃ (*x::int*). *Ifm* (*real x#bs*) *p*)
(**is** (∃ *x*. *l dvd x* ∧ *?P x*) = (∃ *x*. *?P′ x*))
proof−
have (∃ *x*. *l dvd x* ∧ *?P x*) = (∃ (*x::int*). *?P* (*l*∗*x*))
using *unity-coeff-ex*[**where** *l=l* **and** *P=?P*, *simplified*] **by** *simp*
also have ... = (∃ (*x::int*). *?P′ x*) **using** *aβ*[*OF linp d lp*] **by** *simp*
finally show *?thesis* .
qed

lemma *β*:
assumes *lp*: *iszlfm p* (*a#bs*)
and *u*: *dβ p 1*
and *d*: *dδ p d*
and *dp*: *d* > *0*
and *nob*: ¬(∃(*j::int*) ∈ {*1* .. *d*}. ∃ *b*∈ (*Inum* (*a#bs*)) ‘ *set*(*β p*). *real x* = *b* + *real j*)
and *p*: *Ifm* (*real x#bs*) *p* (**is** *?P x*)
shows *?P* (*x* − *d*)
using *lp u d dp nob p*
proof(*induct p rule*: *iszlfm.induct*)
case (*5 c e*) **hence** *c1*: *c=1* **and** *bn*:*numbound0 e* **using** *dvd1-eq1*[**where** *x=c*] **by** *simp+*
with *dp p c1 numbound0-I*[*OF bn*,**where** *b=real* (*x*−*d*) **and** *b′=real x* **and** *bs=bs*] *prems*
show *?case* **by** (*simp del*: *real-of-int-minus*)
next
case (*6 c e*) **hence** *c1*: *c=1* **and** *bn*:*numbound0 e* **using** *dvd1-eq1*[**where** *x=c*] **by** *simp+*
with *dp p c1 numbound0-I*[*OF bn*,**where** *b=real* (*x*−*d*) **and** *b′=real x* **and** *bs=bs*] *prems*
show *?case* **by** (*simp del*: *real-of-int-minus*)
next
case (*7 c e*) **hence** *p*: *Ifm* (*real x #bs*) (*Gt* (*CN 0 c e*)) **and** *c1*: *c=1* **and** *bn*:*numbound0 e* **and** *ie1*:*isint e* (*a#bs*) **using** *dvd1-eq1*[**where** *x=c*] **by** *simp+*
let *?e* = *Inum* (*real x* # *bs*) *e*
from *ie1* **have** *ie*: *real* (*floor ?e*) = *?e* **using** *isint-iff*[**where** *n=e* **and** *bs=a#bs*]

numbound0-I[*OF bn*,**where** *b*=*a* **and** *b*′=*real x* **and** *bs*=*bs*]
by (*simp add*: *isint-iff*)
{**assume** *real* (*x*−*d*) +*?e* > *0* **hence** *?case* **using** *c1*
numbound0-I[*OF bn*,**where** *b*=*real* (*x*−*d*) **and** *b*′=*real x* **and** *bs*=*bs*]
by (*simp del*: *real-of-int-minus*)}
moreover
{**assume** *H*: ¬ *real* (*x*−*d*) + *?e* > *0*
let *?v*=*Neg e*
have *vb*: *?v* ∈ *set* (β (*Gt* (*CN 0 c e*))) **by** *simp*
from *prems*(*11*)[*simplified simp-thms Inum.simps β.simps set.simps bex-simps numbound0-I*[*OF bn*,**where** *b*=*a* **and** *b*′=*real x* **and** *bs*=*bs*]]
have *nob*: ¬ (∃ *j*∈ {*1* ..*d*}. *real x* = − *?e* + *real j*) **by** *auto*
from *H p* **have** *real x* + *?e* > *0* ∧ *real x* + *?e* ≤ *real d* **by** (*simp add*: *c1*)
hence *real* (*x* + *floor ?e*) > *real* (*0*::*int*) ∧ *real* (*x* + *floor ?e*) ≤ *real d*
using *ie* **by** *simp*
hence *x* + *floor ?e* ≥ *1* ∧ *x* + *floor ?e* ≤ *d* **by** *simp*
hence ∃ (*j*::*int*) ∈ {*1* .. *d*}. *j* = *x* + *floor ?e* **by** *simp*
hence ∃ (*j*::*int*) ∈ {*1* .. *d*}. *real x* = *real* (− *floor ?e* + *j*)
by (*simp only*: *real-of-int-inject*) (*simp add*: *ring-simps*)
hence ∃ (*j*::*int*) ∈ {*1* .. *d*}. *real x* = − *?e* + *real j*
by (*simp add*: *ie*[*simplified isint-iff*])
with *nob* **have** *?case* **by** *auto*}
ultimately show *?case* **by** *blast*
next
case (*8 c e*) **hence** *p*: *Ifm* (*real x* #*bs*) (*Ge* (*CN 0 c e*)) **and** *c1*: *c*=*1* **and** *bn*:*numbound0 e*
and *ie1*:*isint e* (*a* #*bs*) **using** *dvd1-eq1*[**where** *x*=*c*] **by** *simp*+
let *?e* = *Inum* (*real x* # *bs*) *e*
from *ie1* **have** *ie*: *real* (*floor ?e*) = *?e* **using** *numbound0-I*[*OF bn*,**where** *b*=*real x* **and** *b*′=*a* **and** *bs*=*bs*] *isint-iff*[**where** *n*=*e* **and** *bs*=(*real x*)#*bs*]
by (*simp add*: *isint-iff*)
{**assume** *real* (*x*−*d*) +*?e* ≥ *0* **hence** *?case* **using** *c1*
numbound0-I[*OF bn*,**where** *b*=*real* (*x*−*d*) **and** *b*′=*real x* **and** *bs*=*bs*]
by (*simp del*: *real-of-int-minus*)}
moreover
{**assume** *H*: ¬ *real* (*x*−*d*) + *?e* ≥ *0*
let *?v*=*Sub* (*C* −*1*) *e*
have *vb*: *?v* ∈ *set* (β (*Ge* (*CN 0 c e*))) **by** *simp*
from *prems*(*11*)[*simplified simp-thms Inum.simps β.simps set.simps bex-simps numbound0-I*[*OF bn*,**where** *b*=*a* **and** *b*′=*real x* **and** *bs*=*bs*]]
have *nob*: ¬ (∃ *j*∈ {*1* ..*d*}. *real x* = − *?e* − *1* + *real j*) **by** *auto*
from *H p* **have** *real x* + *?e* ≥ *0* ∧ *real x* + *?e* < *real d* **by** (*simp add*: *c1*)
hence *real* (*x* + *floor ?e*) ≥ *real* (*0*::*int*) ∧ *real* (*x* + *floor ?e*) < *real d*
using *ie* **by** *simp*
hence *x* + *floor ?e* +*1* ≥ *1* ∧ *x* + *floor ?e* + *1* ≤ *d* **by** *simp*
hence ∃ (*j*::*int*) ∈ {*1* .. *d*}. *j* = *x* + *floor ?e* + *1* **by** *simp*
hence ∃ (*j*::*int*) ∈ {*1* .. *d*}. *x*= − *floor ?e* − *1* + *j* **by** (*simp add*: *ring-simps*)
hence ∃ (*j*::*int*) ∈ {*1* .. *d*}. *real x*= *real* (− *floor ?e* − *1* + *j*)
by (*simp only*: *real-of-int-inject*)

hence ∃ (*j*::*int*) ∈ {*1* .. *d*}. *real x*= − *?e* − *1* + *real j*
by (*simp add*: *ie*[*simplified isint-iff*])
with *nob* **have** *?case* **by** *simp* }
ultimately show *?case* **by** *blast*
next
case (*3 c e*) **hence** *p*: *Ifm* (*real x* #*bs*) (*Eq* (*CN 0 c e*)) (**is** *?p x*) **and** *c1*: *c*=*1*
and *bn*:*numbound0 e* **and** *ie1*: *isint e* (*a* #*bs*) **using** *dvd1-eq1*[**where** *x*=*c*] **by** *simp*+
let *?e* = *Inum* (*real x* # *bs*) *e*
let *?v*=(*Sub* (*C* −*1*) *e*)
have *vb*: *?v* ∈ *set* (*β* (*Eq* (*CN 0 c e*))) **by** *simp*
from *p* **have** *real x*= − *?e* **by** (*simp add*: *c1*) **with** *prems*(*11*) **show** *?case* **using** *dp*
by *simp* (*erule ballE*[**where** *x*=*1*],
simp-all add:*ring-simps numbound0-I*[*OF bn*,**where** *b*=*real x***and** *b'*=*a***and** *bs*=*bs*])
next
case (*4 c e*)**hence** *p*: *Ifm* (*real x* #*bs*) (*NEq* (*CN 0 c e*)) (**is** *?p x*) **and** *c1*: *c*=*1*

and *bn*:*numbound0 e* **and** *ie1*: *isint e* (*a* #*bs*) **using** *dvd1-eq1*[**where** *x*=*c*] **by** *simp*+
let *?e* = *Inum* (*real x* # *bs*) *e*
let *?v*=*Neg e*
have *vb*: *?v* ∈ *set* (*β* (*NEq* (*CN 0 c e*))) **by** *simp*
{**assume** *real x* − *real d* + *Inum* ((*real* (*x* −*d*)) # *bs*) *e* ≠ *0*
hence *?case* **by** (*simp add*: *c1*)}
moreover
{**assume** *H*: *real x* − *real d* + *Inum* ((*real* (*x* −*d*)) # *bs*) *e* = *0*
hence *real x* = − *Inum* ((*real* (*x* −*d*)) # *bs*) *e* + *real d* **by** *simp*
hence *real x* = − *Inum* (*a* # *bs*) *e* + *real d*
by (*simp add*: *numbound0-I*[*OF bn*,**where** *b*=*real x* − *real d***and** *b'*=*a***and** *bs*=*bs*])
with *prems*(*11*) **have** *?case* **using** *dp* **by** *simp*}
ultimately show *?case* **by** *blast*
next
case (*9 j c e*) **hence** *p*: *Ifm* (*real x* #*bs*) (*Dvd j* (*CN 0 c e*)) (**is** *?p x*) **and** *c1*: *c*=*1*
and *bn*:*numbound0 e* **using** *dvd1-eq1*[**where** *x*=*c*] **by** *simp*+
let *?e* = *Inum* (*real x* # *bs*) *e*
from *prems* **have** *isint e* (*a* #*bs*) **by** *simp*
hence *ie*: *real* (*floor ?e*) = *?e* **using** *isint-iff*[**where** *n*=*e* **and** *bs*=(*real x*)#*bs*] *numbound0-I*[*OF bn*,**where** *b*=*real x* **and** *b'*=*a* **and** *bs*=*bs*]
by (*simp add*: *isint-iff*)
from *prems* **have** *id*: *j dvd d* **by** *simp*
from *c1 ie*[*symmetric*] **have** *?p x* = (*real j rdvd real* (*x*+ *floor ?e*)) **by** *simp*
also have ... = (*j dvd x* + *floor ?e*)
using *int-rdvd-real*[**where** *i*=*j* **and** *x*=*real* (*x*+ *floor ?e*)] **by** *simp*
also have ... = (*j dvd x* − *d* + *floor ?e*)
using *dvd-period*[*OF id*, **where** *x*=*x* **and** *c*=−*1* **and** *t*=*floor ?e*] **by** *simp*

also have ... = (*real j rdvd real* (*x* − *d* + *floor ?e*))
using *int-rdvd-real*[**where** *i*=*j* **and** *x*=*real* (*x*−*d* + *floor ?e*),*symmetric*, *simplified*]
ie **by** *simp*
also have ... = (*real j rdvd real x* − *real d* + *?e*)
using *ie* **by** *simp*
finally show *?case*
using *numbound0-I*[*OF bn*,**where** *b*=*real* (*x*−*d*) **and** *b'*=*real x* **and** *bs*=*bs*] *c1 p* **by** *simp*
next
case (*10 j c e*) **hence** *p*: *Ifm* (*real x* #*bs*) (*NDvd j* (*CN 0 c e*)) (**is** *?p x*) **and** *c1*: *c*=*1* **and** *bn*:*numbound0 e* **using** *dvd1-eq1*[**where** *x*=*c*] **by** *simp*+
let *?e* = *Inum* (*real x* # *bs*) *e*
from *prems* **have** *isint e* (*a*#*bs*) **by** *simp*
hence *ie*: *real* (*floor ?e*) = *?e* **using** *numbound0-I*[*OF bn*,**where** *b*=*real x* **and** *b'*=*a* **and** *bs*=*bs*] *isint-iff*[**where** *n*=*e* **and** *bs*=(*real x*)#*bs*]
by (*simp add*: *isint-iff*)
from *prems* **have** *id*: *j dvd d* **by** *simp*
from *c1 ie*[*symmetric*] **have** *?p x* = (¬ *real j rdvd real* (*x*+ *floor ?e*)) **by** *simp*
also have ... = (¬ *j dvd x* + *floor ?e*)
using *int-rdvd-real*[**where** *i*=*j* **and** *x*=*real* (*x*+ *floor ?e*)] **by** *simp*
also have ... = (¬ *j dvd x* − *d* + *floor ?e*)
using *dvd-period*[*OF id*, **where** *x*=*x* **and** *c*=−*1* **and** *t*=*floor ?e*] **by** *simp*
also have ... = (¬ *real j rdvd real* (*x* − *d* + *floor ?e*))
using *int-rdvd-real*[**where** *i*=*j* **and** *x*=*real* (*x*−*d* + *floor ?e*),*symmetric*, *simplified*]
ie **by** *simp*
also have ... = (¬ *real j rdvd real x* − *real d* + *?e*)
using *ie* **by** *simp*
finally show *?case* **using** *numbound0-I*[*OF bn*,**where** *b*=*real* (*x*−*d*) **and** *b'*=*real x* **and** *bs*=*bs*] *c1 p* **by** *simp*
qed (*auto simp add*: *numbound0-I*[**where** *bs*=*bs* **and** *b*=*real* (*x* − *d*) **and** *b'*=*real x*] *nth-pos2 simp del*: *real-of-int-diff*)

lemma β':
assumes *lp*: *iszlfm p* (*a* #*bs*)
and *u*: *d*β *p 1*
and *d*: *d*δ *p d*
and *dp*: *d* > *0*
shows ∀ *x*. ¬(∃ (*j*::*int*) ∈ {*1* .. *d*}. ∃ *b*∈ *set*(β *p*). *Ifm* ((*Inum* (*a*#*bs*) *b* + *real j*) #*bs*) *p*) ⟶ *Ifm* (*real x*#*bs*) *p* ⟶ *Ifm* (*real* (*x* − *d*)#*bs*) *p* (**is** ∀ *x*. *?b* ⟶ *?P x* ⟶ *?P* (*x* − *d*))
proof(*clarify*)
fix *x*
assume *nb*:*?b* **and** *px*: *?P x*
hence *nb2*: ¬(∃ (*j*::*int*) ∈ {*1* .. *d*}. ∃ *b*∈ (*Inum* (*a*#*bs*)) ‘ *set*(β *p*). *real x* = *b* + *real j*)
by *auto*
from β[*OF lp u d dp nb2 px*] **show** *?P* (*x* −*d*) **.**

qed

lemma *β-int*: **assumes** *lp*: *iszlfm p bs*
  shows ∀ *b*∈ *set* (*β p*). *isint b bs*
using *lp* **by** (*induct p rule*: *iszlfm.induct*) (*auto simp add*: *isint-neg isint-sub*)

lemma *cpmi-eq*: *0* < *D* ⟹ (*EX z*::*int*. *ALL x*. *x* < *z* −−> (*P x* = *P1 x*))
==> *ALL x*.~(*EX* (*j*::*int*) : {*1*..*D*}. *EX* (*b*::*int*) : *B*. *P*(*b*+*j*)) −−> *P* (*x*) −−> *P* (*x* − *D*)
==> (*ALL* (*x*::*int*). *ALL* (*k*::*int*). ((*P1 x*)= (*P1* (*x*−*k*∗*D*))))
==> (*EX* (*x*::*int*). *P*(*x*)) = ((*EX* (*j*::*int*) : {*1*..*D*} . (*P1*(*j*))) | (*EX* (*j*::*int*) : {*1*..*D*}. *EX* (*b*::*int*) : *B*. *P* (*b*+*j*)))
apply(*rule iffI*)
prefer *2*
apply(*drule minusinfinity*)
apply *assumption*+
apply(*fastsimp*)
apply *clarsimp*
apply(*subgoal-tac* !!*k*. *0*<=*k* ⟹ !*x*. *P x* ⟶ *P* (*x* − *k*∗*D*))
apply(*frule-tac x* = *x* **and** *z*=*z* **in** *decr-lemma*)
apply(*subgoal-tac P1*(*x* − (|*x* − *z*| + *1*) ∗ *D*))
prefer *2*
apply(*subgoal-tac 0* <= (|*x* − *z*| + *1*))
prefer *2* **apply** *arith*
 apply *fastsimp*
apply(*drule* (*1*) *periodic-finite-ex*)
apply *blast*
apply(*blast dest*:*decr-mult-lemma*)
done

theorem *cp-thm*:
  assumes *lp*: *iszlfm p* (*a* #*bs*)
  and *u*: *dβ p 1*
  and *d*: *dδ p d*
  and *dp*: *d* > *0*
   shows (∃ (*x*::*int*). *Ifm* (*real x* #*bs*) *p*) = (∃ *j*∈ {*1*.. *d*}. *Ifm* (*real j* #*bs*) (*minusinf p*) ∨ (∃ *b* ∈ *set* (*β p*). *Ifm* ((*Inum* (*a*#*bs*) *b* + *real j*) #*bs*) *p*))
  (**is** (∃ (*x*::*int*). *?P* (*real x*)) = (∃ *j*∈ *?D*. *?M j* ∨ (∃ *b*∈ *?B*. *?P* (*?I b* + *real j*))))
proof−
  from *minusinf-inf*[*OF lp*]
  have *th*: ∃(*z*::*int*). ∀*x*<*z*. *?P* (*real x*) = *?M x* **by** *blast*
  let *?B′* = {*floor* (*?I b*) | *b*. *b*∈ *?B*}
  from *β-int*[*OF lp*] *isint-iff*[**where** *bs*=*a* # *bs*] **have** *B*: ∀ *b*∈ *?B*. *real* (*floor* (*?I b*)) = *?I b* **by** *simp*
  from *B*[*rule-format*]
  have (∃*j*∈*?D*. ∃*b*∈ *?B*. *?P* (*?I b* + *real j*)) = (∃*j*∈*?D*. ∃*b*∈ *?B*. *?P* (*real* (*floor* (*?I b*)) + *real j*))

```
    by simp
  also have ... = (∃j∈?D. ∃b∈ ?B. ?P (real (floor (?I b) + j))) by simp
  also have... = (∃ j ∈ ?D. ∃ b ∈ ?B'. ?P (real (b + j)))  by blast
  finally have BB':
    (∃j∈?D. ∃b∈ ?B. ?P (?I b + real j)) = (∃ j ∈ ?D. ∃ b ∈ ?B'. ?P (real (b + j)))
    by blast
  hence th2: ∀ x. ¬ (∃ j ∈ ?D. ∃ b ∈ ?B'. ?P (real (b + j))) ⟶ ?P (real x) ⟶ ?P (real (x − d)) using β'[OF lp u d dp] by blast
  from minusinf-repeats[OF d lp]
  have th3: ∀ x k. ?M x = ?M (x−k∗d) by simp
  from cpmi-eq[OF dp th th2 th3] BB' show ?thesis by blast
qed

consts
  ϱ :: fm ⇒ (num × int) list
  σϱ:: fm ⇒ num × int ⇒ fm
  αϱ :: fm ⇒ (num×int) list
  aϱ :: fm ⇒ int ⇒ fm
recdef ϱ measure size
  ϱ (And p q) = (ϱ p @ ϱ q)
  ϱ (Or p q) = (ϱ p @ ϱ q)
  ϱ (Eq  (CN 0 c e)) = [(Sub (C −1) e,c)]
  ϱ (NEq (CN 0 c e)) = [(Neg e,c)]
  ϱ (Lt  (CN 0 c e)) = []
  ϱ (Le  (CN 0 c e)) = []
  ϱ (Gt  (CN 0 c e)) = [(Neg e, c)]
  ϱ (Ge  (CN 0 c e)) = [(Sub (C (−1)) e, c)]
  ϱ p = []

recdef σϱ measure size
  σϱ (And p q) = (λ (t,k). And (σϱ p (t,k)) (σϱ q (t,k)))
  σϱ (Or p q) = (λ (t,k). Or (σϱ p (t,k)) (σϱ q (t,k)))
  σϱ (Eq  (CN 0 c e)) = (λ (t,k). if k dvd c then (Eq (Add (Mul (c div k) t) e))
                                       else (Eq (Add (Mul c t) (Mul k e))))
  σϱ (NEq (CN 0 c e)) = (λ (t,k). if k dvd c then (NEq (Add (Mul (c div k) t) e))
                                       else (NEq (Add (Mul c t) (Mul k e))))
  σϱ (Lt  (CN 0 c e)) = (λ (t,k). if k dvd c then (Lt (Add (Mul (c div k) t) e))
                                       else (Lt (Add (Mul c t) (Mul k e))))
  σϱ (Le  (CN 0 c e)) = (λ (t,k). if k dvd c then (Le (Add (Mul (c div k) t) e))
                                       else (Le (Add (Mul c t) (Mul k e))))
  σϱ (Gt  (CN 0 c e)) = (λ (t,k). if k dvd c then (Gt (Add (Mul (c div k) t) e))
                                       else (Gt (Add (Mul c t) (Mul k e))))
  σϱ (Ge  (CN 0 c e)) = (λ (t,k). if k dvd c then (Ge (Add (Mul (c div k) t) e))
                                       else (Ge (Add (Mul c t) (Mul k e))))
```

$\sigma\varrho$ (*Dvd i* (*CN 0 c e*)) =(λ (*t*,*k*). *if k dvd c then* (*Dvd i* (*Add* (*Mul* (*c div k*) *t*) *e*))
else (*Dvd* (*i*∗*k*) (*Add* (*Mul c t*) (*Mul k e*))))
$\sigma\varrho$ (*NDvd i* (*CN 0 c e*))=(λ (*t*,*k*). *if k dvd c then* (*NDvd i* (*Add* (*Mul* (*c div k*) *t*) *e*))
else (*NDvd* (*i*∗*k*) (*Add* (*Mul c t*) (*Mul k e*))))
$\sigma\varrho$ *p* = (λ (*t*,*k*). *p*)

recdef $\alpha\varrho$ *measure size*
$\alpha\varrho$ (*And p q*) = ($\alpha\varrho$ *p* @ $\alpha\varrho$ *q*)
$\alpha\varrho$ (*Or p q*) = ($\alpha\varrho$ *p* @ $\alpha\varrho$ *q*)
$\alpha\varrho$ (*Eq* (*CN 0 c e*)) = [(*Add* (*C* −*1*) *e*,*c*)]
$\alpha\varrho$ (*NEq* (*CN 0 c e*)) = [(*e*,*c*)]
$\alpha\varrho$ (*Lt* (*CN 0 c e*)) = [(*e*,*c*)]
$\alpha\varrho$ (*Le* (*CN 0 c e*)) = [(*Add* (*C* −*1*) *e*,*c*)]
$\alpha\varrho$ *p* = []

recdef *a*ϱ *measure size*
aϱ (*And p q*) = (λ *k*. *And* (*a*ϱ *p k*) (*a*ϱ *q k*))
aϱ (*Or p q*) = (λ *k*. *Or* (*a*ϱ *p k*) (*a*ϱ *q k*))
aϱ (*Eq* (*CN 0 c e*)) = (λ *k*. *if k dvd c then* (*Eq* (*CN 0* (*c div k*) *e*))
else (*Eq* (*CN 0 c* (*Mul k e*))))
aϱ (*NEq* (*CN 0 c e*)) = (λ *k*. *if k dvd c then* (*NEq* (*CN 0* (*c div k*) *e*))
else (*NEq* (*CN 0 c* (*Mul k e*))))
aϱ (*Lt* (*CN 0 c e*)) = (λ *k*. *if k dvd c then* (*Lt* (*CN 0* (*c div k*) *e*))
else (*Lt* (*CN 0 c* (*Mul k e*))))
aϱ (*Le* (*CN 0 c e*)) = (λ *k*. *if k dvd c then* (*Le* (*CN 0* (*c div k*) *e*))
else (*Le* (*CN 0 c* (*Mul k e*))))
aϱ (*Gt* (*CN 0 c e*)) = (λ *k*. *if k dvd c then* (*Gt* (*CN 0* (*c div k*) *e*))
else (*Gt* (*CN 0 c* (*Mul k e*))))
aϱ (*Ge* (*CN 0 c e*)) = (λ *k*. *if k dvd c then* (*Ge* (*CN 0* (*c div k*) *e*))
else (*Ge* (*CN 0 c* (*Mul k e*))))
aϱ (*Dvd i* (*CN 0 c e*)) = (λ *k*. *if k dvd c then* (*Dvd i* (*CN 0* (*c div k*) *e*))
else (*Dvd* (*i*∗*k*) (*CN 0 c* (*Mul k e*))))
aϱ (*NDvd i* (*CN 0 c e*)) = (λ *k*. *if k dvd c then* (*NDvd i* (*CN 0* (*c div k*) *e*))
else (*NDvd* (*i*∗*k*) (*CN 0 c* (*Mul k e*))))
aϱ *p* = (λ *k*. *p*)

constdefs σ :: *fm* ⇒ *int* ⇒ *num* ⇒ *fm*
σ *p k t* ≡ *And* (*Dvd k t*) ($\sigma\varrho$ *p* (*t*,*k*))

lemma $\sigma\varrho$:
assumes *linp*: *iszlfm p* (*real* (*x*::*int*)#*bs*)
and *kpos*: *real k* > *0*
and *tnb*: *numbound0 t*
and *tint*: *isint t* (*real x*#*bs*)
and *kdt*: *k dvd floor* (*Inum* (*b*′#*bs*) *t*)

```
  shows Ifm (real x#bs) (σϱ p (t,k)) =
  (Ifm ((real ((floor (Inum (b'#bs) t)) div k))#bs) p)
  (is ?I (real x) (?s p) = (?I (real ((floor (?N b' t)) div k)) p) is - = (?I ?tk p))
using linp kpos tnb
proof(induct p rule: σϱ.induct)
  case (3 c e)
  from prems have cp: c > 0 and nb: numbound0 e by auto
    {assume kdc: k dvd c
     from kpos have knz: k≠0 by simp
    from tint have ti: real (floor (?N (real x) t)) = ?N (real x) t using isint-def
by simp
     from prems have  ?case using real-of-int-div[OF knz kdc] real-of-int-div[OF
knz kdt]
       numbound0-I[OF tnb, where bs=bs and b=b' and b'=real x]
    numbound0-I[OF nb, where bs=bs and b=?tk and b'=real x] by (simp add:
ti) }
    moreover
    {assume ¬ k dvd c
     from kpos have knz: k≠0 by simp hence knz': real k ≠ 0 by simp
    from tint have ti: real (floor (?N (real x) t)) = ?N (real x) t using isint-def
by simp
     from prems have ?I (real x) (?s (Eq (CN 0 c e))) = ((real c * (?N (real x)
t / real k) + ?N (real x) e)* real k = 0)
       using real-of-int-div[OF knz kdt]
         numbound0-I[OF tnb, where bs=bs and b=b' and b'=real x]
         numbound0-I[OF nb, where bs=bs and b=?tk and b'=real x] by (simp
add: ti ring-simps)
    also have ... = (?I ?tk (Eq (CN 0 c e))) using nonzero-eq-divide-eq[OF knz',
where a=real c * (?N (real x) t / real k) + ?N (real x) e and b=0, symmetric]
real-of-int-div[OF knz kdt] numbound0-I[OF tnb, where bs=bs and b=b' and
b'=real x]
         numbound0-I[OF nb, where bs=bs and b=?tk and b'=real x]
       by (simp add: ti)
     finally have ?case . }
    ultimately show ?case by blast
next
  case (4 c e)
  from prems have cp: c > 0 and nb: numbound0 e by auto
    {assume kdc: k dvd c
     from kpos have knz: k≠0 by simp
    from tint have ti: real (floor (?N (real x) t)) = ?N (real x) t using isint-def
by simp
     from prems have  ?case using real-of-int-div[OF knz kdc] real-of-int-div[OF
knz kdt]
       numbound0-I[OF tnb, where bs=bs and b=b' and b'=real x]
    numbound0-I[OF nb, where bs=bs and b=?tk and b'=real x] by (simp add:
ti) }
    moreover
    {assume ¬ k dvd c
```

from *kpos* **have** *knz*: *k≠0* **by** *simp* **hence** *knz′*: *real k ≠ 0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?I* (*real x*) (*?s* (*NEq* (*CN 0 c e*))) = ((*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e*)∗ *real k ≠ 0*)
using *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti ring-simps*)
also have ... = (*?I ?tk* (*NEq* (*CN 0 c e*))) **using** *nonzero-eq-divide-eq*[*OF knz′*, **where** *a*=*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e* **and** *b*=*0*, *symmetric*] *real-of-int-div*[*OF knz kdt*] *numbound0-I*[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*]
by (*simp add*: *ti*)
finally have *?case* **.** **}**
ultimately show *?case* **by** *blast*
next
case (*5 c e*)
from *prems* **have** *cp*: *c > 0* **and** *nb*: *numbound0 e* **by** *auto*
{assume *kdc*: *k dvd c*
from *kpos* **have** *knz*: *k≠0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti*) **}**
moreover
{assume ¬ *k dvd c*
from *kpos* **have** *knz*: *k≠0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?I* (*real x*) (*?s* (*Lt* (*CN 0 c e*))) = ((*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e*)∗ *real k < 0*)
using *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti ring-simps*)
also have ... = (*?I ?tk* (*Lt* (*CN 0 c e*))) **using** *pos-less-divide-eq*[*OF kpos*, **where** *a*=*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e* **and** *b*=*0*, *symmetric*] *real-of-int-div*[*OF knz kdt*] *numbound0-I*[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*]
by (*simp add*: *ti*)
finally have *?case* **.** **}**
ultimately show *?case* **by** *blast*
next

case (*6 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
{**assume** *kdc*: *k dvd c*
from *kpos* **have** *knz*: *k*≠*0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti*) }
moreover
{**assume** ¬ *k dvd c*
from *kpos* **have** *knz*: *k*≠*0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?I* (*real x*) (*?s* (*Le* (*CN 0 c e*))) = ((*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e*)∗ *real k* ≤ *0*)
using *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti ring-simps*)
also have ... = (*?I ?tk* (*Le* (*CN 0 c e*))) **using** *pos-le-divide-eq*[*OF kpos*, **where** *a*=*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e* **and** *b*=*0*, *symmetric*] *real-of-int-div*[*OF knz kdt*] *numbound0-I*[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*]
by (*simp add*: *ti*)
finally have *?case* . }
ultimately show *?case* **by** *blast*
next
case (*7 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
{**assume** *kdc*: *k dvd c*
from *kpos* **have** *knz*: *k*≠*0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti*) }
moreover
{**assume** ¬ *k dvd c*
from *kpos* **have** *knz*: *k*≠*0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?I* (*real x*) (*?s* (*Gt* (*CN 0 c e*))) = ((*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e*)∗ *real k* > *0*)

using *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b'* **and** *b'*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b'*=*real x*] **by** (*simp add*: *ti ring-simps*)
also have ... = (*?I ?tk* (*Gt* (*CN 0 c e*))) **using** *pos-divide-less-eq*[*OF kpos*, **where** *a*=*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e* **and** *b*=*0*, *symmetric*] *real-of-int-div*[*OF knz kdt*] *numbound0-I*[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b'* **and** *b'*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b'*=*real x*]
by (*simp add*: *ti*)
finally have *?case* **.** }
ultimately show *?case* **by** *blast*
next
case (*8 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
{**assume** *kdc*: *k dvd c*
from *kpos* **have** *knz*: *k*≠*0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b'* **and** *b'*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b'*=*real x*] **by** (*simp add*: *ti*) }
moreover
{**assume** ¬ *k dvd c*
from *kpos* **have** *knz*: *k*≠*0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?I* (*real x*) (*?s* (*Ge* (*CN 0 c e*))) = ((*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e*)∗ *real k* ≥ *0*)
using *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b'* **and** *b'*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b'*=*real x*] **by** (*simp add*: *ti ring-simps*)
also have ... = (*?I ?tk* (*Ge* (*CN 0 c e*))) **using** *pos-divide-le-eq*[*OF kpos*, **where** *a*=*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e* **and** *b*=*0*, *symmetric*] *real-of-int-div*[*OF knz kdt*] *numbound0-I*[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b'* **and** *b'*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b'*=*real x*]
by (*simp add*: *ti*)
finally have *?case* **.** }
ultimately show *?case* **by** *blast*
next
case (*9 i c e*) **from** *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
{**assume** *kdc*: *k dvd c*
from *kpos* **have** *knz*: *k*≠*0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*

from *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti*) **}**
moreover
{assume ¬ *k dvd c*
from *kpos* **have** *knz*: $k \neq 0$ **by** *simp* **hence** *knz′*: *real k* ≠ *0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?I* (*real x*) (*?s* (*Dvd i* (*CN 0 c e*))) = (*real i* ∗ *real k rdvd* (*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e*)∗ *real k*)
using *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti ring-simps*)
also have ... = (*?I ?tk* (*Dvd i* (*CN 0 c e*))) **using** *rdvd-mult*[*OF knz*, **where** *n*=*i*] *real-of-int-div*[*OF knz kdt*] *numbound0-I*[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*]
by (*simp add*: *ti*)
finally have *?case* **. }**
ultimately show *?case* **by** *blast*
next
case (*10 i c e*) **from** *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
{assume *kdc*: *k dvd c*
from *kpos* **have** *knz*: $k \neq 0$ **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti*) **}**
moreover
{assume ¬ *k dvd c*
from *kpos* **have** *knz*: $k \neq 0$ **by** *simp* **hence** *knz′*: *real k* ≠ *0* **by** *simp*
from *tint* **have** *ti*: *real* (*floor* (*?N* (*real x*) *t*)) = *?N* (*real x*) *t* **using** *isint-def* **by** *simp*
from *prems* **have** *?I* (*real x*) (*?s* (*NDvd i* (*CN 0 c e*))) = (¬ (*real i* ∗ *real k rdvd* (*real c* ∗ (*?N* (*real x*) *t* / *real k*) + *?N* (*real x*) *e*)∗ *real k*))
using *real-of-int-div*[*OF knz kdt*]
numbound0-I[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*] **by** (*simp add*: *ti ring-simps*)
also have ... = (*?I ?tk* (*NDvd i* (*CN 0 c e*))) **using** *rdvd-mult*[*OF knz*, **where** *n*=*i*] *real-of-int-div*[*OF knz kdt*] *numbound0-I*[*OF tnb*, **where** *bs*=*bs* **and** *b*=*b′* **and** *b′*=*real x*]
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*?tk* **and** *b′*=*real x*]

by (*simp add*: *ti*)
finally have *?case* **. }**
ultimately show *?case* **by** *blast*
qed (*simp-all add*: *nth-pos2 bound0-I*[**where** *bs*=*bs* **and** *b*=*real* ((*floor* (*?N b′ t*)) *div k*) **and** *b′*=*real x*] *numbound0-I*[**where** *bs*=*bs* **and** *b*=*real* ((*floor* (*?N b′ t*)) *div k*) **and** *b′*=*real x*])

lemma *aϱ*:
assumes *lp*: *iszlfm p* (*real* (*x*::*int*)#*bs*) **and** *kp*: *real k* > *0*
shows *Ifm* (*real* (*x*∗*k*)#*bs*) (*aϱ p k*) = *Ifm* (*real x*#*bs*) *p* (**is** *?I* (*x*∗*k*) (*?f p k*) = *?I x p*)
using *lp bound0-I*[**where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b′*=*real x*] *numbound0-I*[**where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b′*=*real x*]
proof(*induct p rule*: *aϱ.induct*)
case (*3 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
from *kp* **have** *knz*: *k*≠*0* **by** *simp* **hence** *knz′*: *real k* ≠ *0* **by** *simp*
{assume *kdc*: *k dvd c* **from** *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] **by** *simp* **}**
moreover
{assume *nkdc*: ¬ *k dvd c* **hence** *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b′*=*real x*] *nonzero-eq-divide-eq*[*OF knz′*, **where** *b*=*0* **and** *a*=*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*, *symmetric*] **by** (*simp add*: *ring-simps*)**}**
ultimately show *?case* **by** *blast*
next
case (*4 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
from *kp* **have** *knz*: *k*≠*0* **by** *simp* **hence** *knz′*: *real k* ≠ *0* **by** *simp*
{assume *kdc*: *k dvd c* **from** *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] **by** *simp* **}**
moreover
{assume *nkdc*: ¬ *k dvd c* **hence** *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b′*=*real x*] *nonzero-eq-divide-eq*[*OF knz′*, **where** *b*=*0* **and** *a*=*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*, *symmetric*] **by** (*simp add*: *ring-simps*)**}**
ultimately show *?case* **by** *blast*
next
case (*5 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
from *kp* **have** *knz*: *k*≠*0* **by** *simp* **hence** *knz′*: *real k* ≠ *0* **by** *simp*
{assume *kdc*: *k dvd c* **from** *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] **by** *simp* **}**
moreover
{assume *nkdc*: ¬ *k dvd c* **hence** *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b′*=*real x*] *pos-less-divide-eq*[*OF kp*, **where** *b*=*0* **and** *a*=*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*, *symmetric*] **by** (*simp add*: *ring-simps*)**}**
ultimately show *?case* **by** *blast*

next
case (*6 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
from *kp* **have** *knz*: *k*≠*0* **by** *simp* **hence** *knz'*: *real k* ≠ *0* **by** *simp*
{**assume** *kdc*: *k dvd c* **from** *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] **by** *simp* }
moreover
{**assume** *nkdc*: ¬ *k dvd c* **hence** *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b'*=*real x*] *pos-le-divide-eq*[*OF kp*, **where** *b*=*0* **and** *a*=*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*, *symmetric*] **by** (*simp add*: *ring-simps*)}
ultimately show *?case* **by** *blast*
next
case (*7 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
from *kp* **have** *knz*: *k*≠*0* **by** *simp* **hence** *knz'*: *real k* ≠ *0* **by** *simp*
{**assume** *kdc*: *k dvd c* **from** *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] **by** *simp* }
moreover
{**assume** *nkdc*: ¬ *k dvd c* **hence** *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b'*=*real x*] *pos-divide-less-eq*[*OF kp*, **where** *b*=*0* **and** *a*=*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*, *symmetric*] **by** (*simp add*: *ring-simps*)}
ultimately show *?case* **by** *blast*
next
case (*8 c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
from *kp* **have** *knz*: *k*≠*0* **by** *simp* **hence** *knz'*: *real k* ≠ *0* **by** *simp*
{**assume** *kdc*: *k dvd c* **from** *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] **by** *simp* }
moreover
{**assume** *nkdc*: ¬ *k dvd c* **hence** *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b'*=*real x*] *pos-divide-le-eq*[*OF kp*, **where** *b*=*0* **and** *a*=*real c* ∗ *real x* + *Inum* (*real x* # *bs*) *e*, *symmetric*] **by** (*simp add*: *ring-simps*)}
ultimately show *?case* **by** *blast*
next
case (*9 i c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
from *kp* **have** *knz*: *k*≠*0* **by** *simp* **hence** *knz'*: *real k* ≠ *0* **by** *simp*
{**assume** *kdc*: *k dvd c* **from** *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] **by** *simp* }
moreover
{**assume** ¬ *k dvd c*
hence *Ifm* (*real* (*x*∗*k*)#*bs*) (*a*ϱ (*Dvd i* (*CN 0 c e*)) *k*) =
(*real i* ∗ *real k rdvd* (*real c* ∗ *real x* + *Inum* (*real x*#*bs*) *e*) ∗ *real k*)
using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b'*=*real x*]
by (*simp add*: *ring-simps*)
also have . . . = (*Ifm* (*real x*#*bs*) (*Dvd i* (*CN 0 c e*))) **by** (*simp add*: *rdvd-mult*[*OF knz*, **where** *n*=*i*])
finally have *?case* . }
ultimately show *?case* **by** *blast*

next
case (*10 i c e*)
from *prems* **have** *cp*: *c* > *0* **and** *nb*: *numbound0 e* **by** *auto*
from *kp* **have** *knz*: *k*≠*0* **by** *simp* **hence** *knz′*: *real k* ≠ *0* **by** *simp*
{**assume** *kdc*: *k dvd c* **from** *prems* **have** *?case* **using** *real-of-int-div*[*OF knz kdc*] **by** *simp* }
moreover
{**assume** ¬ *k dvd c*
hence *Ifm* (*real* (*x*∗*k*)#*bs*) (*aϱ* (*NDvd i* (*CN 0 c e*)) *k*) =
(¬ (*real i* ∗ *real k rdvd* (*real c* ∗ *real x* + *Inum* (*real x*#*bs*) *e*) ∗ *real k*))
using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real* (*x*∗*k*) **and** *b′*=*real x*]
by (*simp add*: *ring-simps*)
also have ... = (*Ifm* (*real x*#*bs*) (*NDvd i* (*CN 0 c e*))) **by** (*simp add*: *rdvd-mult*[*OF knz*, **where** *n*=*i*])
finally have *?case* . }
ultimately show *?case* **by** *blast*
qed (*simp-all add*: *nth-pos2*)

lemma *aϱ-ex*:
assumes *lp*: *iszlfm p* (*real* (*x*::*int*)#*bs*) **and** *kp*: *k* > *0*
shows (∃ (*x*::*int*). *real k rdvd real x* ∧ *Ifm* (*real x*#*bs*) (*aϱ p k*)) =
(∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) (**is** (∃ *x*. *?D x* ∧ *?P′ x*) = (∃ *x*. *?P x*))
proof−
have (∃ *x*. *?D x* ∧ *?P′ x*) = (∃ *x*. *k dvd x* ∧ *?P′ x*) **using** *int-rdvd-iff* **by** *simp*
also have ... = (∃*x*. *?P′* (*x*∗*k*)) **using** *unity-coeff-ex*[**where** *P*=*?P′* **and** *l*=*k*, *simplified*]
by (*simp add*: *ring-simps*)
also have ... = (∃ *x*. *?P x*) **using** *aϱ iszlfm-gen*[*OF lp*] *kp* **by** *auto*
finally show *?thesis* .
qed

lemma *σϱ′*: **assumes** *lp*: *iszlfm p* (*real* (*x*::*int*)#*bs*) **and** *kp*: *k* > *0* **and** *nb*: *numbound0 t*
shows *Ifm* (*real x*#*bs*) (*σϱ p* (*t*,*k*)) = *Ifm* ((*Inum* (*real x*#*bs*) *t*)#*bs*) (*aϱ p k*)
using *lp*
by(*induct p rule*: *σϱ.induct*, *simp-all add*:
numbound0-I[*OF nb*, **where** *bs*=*bs* **and** *b*=*Inum* (*real x*#*bs*) *t* **and** *b′*=*real x*]
numbound0-I[**where** *bs*=*bs* **and** *b*=*Inum* (*real x*#*bs*) *t* **and** *b′*=*real x*]
bound0-I[**where** *bs*=*bs* **and** *b*=*Inum* (*real x*#*bs*) *t* **and** *b′*=*real x*] *nth-pos2 cong*: *imp-cong*)

lemma *σϱ-nb*: **assumes** *lp*:*iszlfm p* (*a*#*bs*) **and** *nb*: *numbound0 t*
shows *bound0* (*σϱ p* (*t*,*k*))
using *lp*
by (*induct p rule*: *iszlfm.induct*, *auto simp add*: *nb*)

lemma *ϱ-l*:
assumes *lp*: *iszlfm p* (*real* (*i*::*int*)#*bs*)
shows ∀ (*b*,*k*) ∈ *set* (*ϱ p*). *k* >*0* ∧ *numbound0 b* ∧ *isint b* (*real i*#*bs*)

using *lp* **by** (*induct p rule*: *ϱ.induct*, *auto simp add*: *isint-sub isint-neg*)

lemma *αϱ-l*:
  assumes *lp*: *iszlfm p* (*real* (*i*::*int*)#*bs*)
  shows ∀ (*b*,*k*) ∈ *set* (*αϱ p*). *k* >*0* ∧ *numbound0 b* ∧ *isint b* (*real i*#*bs*)
using *lp isint-add* [*OF isint-c*[**where** *j*=− *1*],**where** *bs*=*real i*#*bs*]
 by (*induct p rule*: *αϱ.induct*, *auto*)

lemma *zminusinf-ϱ*:
  assumes *lp*: *iszlfm p* (*real* (*i*::*int*)#*bs*)
  and *nmi*: ¬ (*Ifm* (*real i*#*bs*) (*minusinf p*)) (**is** ¬ (*Ifm* (*real i*#*bs*) (*?M p*)))
  and *ex*: *Ifm* (*real i*#*bs*) *p* (**is** *?I i p*)
  shows ∃ (*e*,*c*) ∈ *set* (*ϱ p*). *real* (*c*∗*i*) > *Inum* (*real i*#*bs*) *e* (**is** ∃ (*e*,*c*) ∈ *?R p*. *real* (*c*∗*i*) > *?N i e*)
  using *lp nmi ex*
by (*induct p rule*: *minusinf.induct*, *auto*)

lemma *σ-And*: *Ifm bs* (*σ* (*And p q*) *k t*) = *Ifm bs* (*And* (*σ p k t*) (*σ q k t*))
using *σ-def* **by** *auto*
lemma *σ-Or*: *Ifm bs* (*σ* (*Or p q*) *k t*) = *Ifm bs* (*Or* (*σ p k t*) (*σ q k t*))
using *σ-def* **by** *auto*

lemma *ϱ*: **assumes** *lp*: *iszlfm p* (*real* (*i*::*int*) #*bs*)
  and *pi*: *Ifm* (*real i*#*bs*) *p*
  and *d*: *dδ p d*
  and *dp*: *d* > *0*
  and *nob*: ∀(*e*,*c*) ∈ *set* (*ϱ p*). ∀ *j*∈ {*1* .. *c*∗*d*}. *real* (*c*∗*i*) ≠ *Inum* (*real i*#*bs*) *e* + *real j*
  (**is** ∀(*e*,*c*) ∈ *set* (*ϱ p*). ∀ *j*∈ {*1* .. *c*∗*d*}. - ≠ *?N i e* + -)
  shows *Ifm* (*real*(*i* − *d*)#*bs*) *p*
  using *lp pi d nob*
proof(*induct p rule*: *iszlfm.induct*)
  case (*3 c e*) **hence** *cp*: *c* >*0* **and** *nb*: *numbound0 e* **and** *ei*: *isint e* (*real i*#*bs*)
    and *pi*: *real* (*c*∗*i*) = − *1* − *?N i e* + *real* (*1*::*int*) **and** *nob*: ∀ *j*∈ {*1* .. *c*∗*d*}. *real* (*c*∗*i*) ≠ −*1* − *?N i e* + *real j*
    by *simp*+
  from *mult-strict-left-mono*[*OF dp cp*] **have** *one*:*1* ∈ {*1* .. *c*∗*d*} **by** *auto*
  from *nob*[*rule-format*, **where** *j*=*1*, *OF one*] *pi* **show** *?case* **by** *simp*
next
  case (*4 c e*)
  hence *cp*: *c* >*0* **and** *nb*: *numbound0 e* **and** *ei*: *isint e* (*real i*#*bs*)
    and *nob*: ∀ *j*∈ {*1* .. *c*∗*d*}. *real* (*c*∗*i*) ≠ − *?N i e* + *real j*
    by *simp*+
  {**assume** *real* (*c*∗*i*) ≠ − *?N i e* + *real* (*c*∗*d*)
   with *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*real i* − *real d* **and** *b'*=*real i*]
    have *?case* **by** (*simp add*: *ring-simps*)}
  moreover
  {**assume** *pi*: *real* (*c*∗*i*) = − *?N i e* + *real* (*c*∗*d*)

```
    from mult-strict-left-mono[OF dp cp] have d: (c*d) ∈ {1 .. c*d} by simp
    from nob[rule-format, where j=c*d, OF d] pi have ?case by simp }
  ultimately show ?case by blast
next
  case (5 c e) hence cp: c > 0 by simp
 from prems mult-strict-left-mono[OF dp cp, simplified real-of-int-less-iff[symmetric]

    real-of-int-mult]
  show ?case using prems dp
     by (simp add: add: numbound0-I[where bs=bs and b=real i − real d and
b′=real i]
       ring-simps)
next
  case (6 c e)  hence cp: c > 0 by simp
 from prems mult-strict-left-mono[OF dp cp, simplified real-of-int-less-iff[symmetric]

    real-of-int-mult]
  show ?case using prems dp
     by (simp add: add: numbound0-I[where bs=bs and b=real i − real d and
b′=real i]
       ring-simps)
next
  case (7 c e) hence cp: c >0 and nb: numbound0 e and ei: isint e (real i#bs)
    and nob: ∀ j∈ {1 .. c*d}. real (c*i) ≠ − ?N i e + real j
    and pi: real (c*i) + ?N i e > 0 and cp′: real c >0
    by simp+
  let ?fe = floor (?N i e)
 from pi cp have th:(real i +?N i e / real c)*real c > 0 by (simp add: ring-simps)
 from pi ei[simplified isint-iff] have real (c*i + ?fe) > real (0::int) by simp
 hence pi′: c*i + ?fe > 0 by (simp only: real-of-int-less-iff[symmetric])
  have real (c*i) + ?N i e > real (c*d) ∨ real (c*i) + ?N i e ≤ real (c*d) by
auto
  moreover
  {assume real (c*i) + ?N i e > real (c*d) hence ?case
      by (simp add: ring-simps
        numbound0-I[OF nb,where bs=bs and b=real i − real d and b′=real i])}
  moreover
  {assume H:real (c*i) + ?N i e ≤ real (c*d)
    with ei[simplified isint-iff] have real (c*i + ?fe) ≤ real (c*d) by simp
    hence pid: c*i + ?fe ≤ c*d by (simp only: real-of-int-le-iff)
    with pi′ have ∃ j1∈ {1 .. c*d}. c*i + ?fe = j1 by auto
    hence ∃ j1∈ {1 .. c*d}. real (c*i) = − ?N i e + real j1
    by (simp only: diff-def[symmetric] real-of-int-mult real-of-int-add real-of-int-inject[symmetric]
ei[simplified isint-iff] ring-simps)
    with nob  have ?case by blast }
  ultimately show ?case by blast
next
  case (8 c e)  hence cp: c >0 and nb: numbound0 e and ei: isint e (real i#bs)
    and nob: ∀ j∈ {1 .. c*d}. real (c*i) ≠ − 1 − ?N i e + real j
```

and *pi*: *real* $(c{*}i)$ + *?N i e* $\geq$ *0* **and** *cp'*: *real c* $>$*0*
by *simp+*
let *?fe* = *floor* (*?N i e*)
from *pi cp* **have** *th*:(*real i* +*?N i e* / *real c*)$*$*real c* $\geq$ *0* **by** (*simp add*: *ring-simps*)
from *pi ei*[*simplified isint-iff*] **have** *real* $(c{*}i$ + *?fe*) $\geq$ *real* (*0*::*int*) **by** *simp*
hence *pi'*: $c{*}i$ + *1* + *?fe* $\geq$ *1* **by** (*simp only*: *real-of-int-le-iff*[*symmetric*])
have *real* $(c{*}i)$ + *?N i e* $\geq$ *real* $(c{*}d)$ $\vee$ *real* $(c{*}i)$ + *?N i e* $<$ *real* $(c{*}d)$ **by** *auto*
moreover
{**assume** *real* $(c{*}i)$ + *?N i e* $\geq$ *real* $(c{*}d)$ **hence** *?case*
by (*simp add*: *ring-simps*
numbound0-I[*OF nb*,**where** *bs*=*bs* **and** *b*=*real i* $-$ *real d* **and** *b'*=*real i*])}
moreover
{**assume** *H*:*real* $(c{*}i)$ + *?N i e* $<$ *real* $(c{*}d)$
with *ei*[*simplified isint-iff*] **have** *real* $(c{*}i$ + *?fe*) $<$ *real* $(c{*}d)$ **by** *simp*
hence *pid*: $c{*}i$ + *1* + *?fe* $\leq$ $c{*}d$ **by** (*simp only*: *real-of-int-le-iff*)
with *pi'* **have** $\exists$ *j1*$\in$ {*1* .. $c{*}d$}. $c{*}i$ + *1*+ *?fe* = *j1* **by** *auto*
hence $\exists$ *j1*$\in$ {*1* .. $c{*}d$}. *real* $(c{*}i)$ + *1*= $-$ *?N i e* + *real j1*
by (*simp only*: *diff-def*[*symmetric*] *real-of-int-mult real-of-int-add real-of-int-inject*[*symmetric*] *ei*[*simplified isint-iff*] *ring-simps real-of-one*)
hence $\exists$ *j1*$\in$ {*1* .. $c{*}d$}. *real* $(c{*}i)$ = ($-$ *?N i e* + *real j1*) $-$ *1*
by (*simp only*: *ring-simps diff-def*[*symmetric*])
hence $\exists$ *j1*$\in$ {*1* .. $c{*}d$}. *real* $(c{*}i)$ = $-$ *1* $-$ *?N i e* + *real j1*
by (*simp only*: *add-ac diff-def*)
with *nob* **have** *?case* **by** *blast* }
ultimately show *?case* **by** *blast*
next
case (*9 j c e*) **hence** *p*: *real j rdvd real* $(c{*}i)$ + *?N i e* (**is** *?p x*) **and** *cp*: *c* $>$ *0* **and** *bn*:*numbound0 e* **by** *simp+*
let *?e* = *Inum* (*real i* # *bs*) *e*
from *prems* **have** *isint e* (*real i* #*bs*) **by** *simp*
hence *ie*: *real* (*floor ?e*) = *?e* **using** *isint-iff*[**where** *n*=*e* **and** *bs*=(*real i*)#*bs*] *numbound0-I*[*OF bn*,**where** *b*=*real i* **and** *b'*=*real i* **and** *bs*=*bs*]
by (*simp add*: *isint-iff*)
from *prems* **have** *id*: *j dvd d* **by** *simp*
from *ie*[*symmetric*] **have** *?p i* = (*real j rdvd real* $(c{*}i$+ *floor ?e*)) **by** *simp*
also have ... = (*j dvd* $c{*}i$ + *floor ?e*)
using *int-rdvd-iff* [**where** *i*=*j* **and** *t*=$c{*}i$+ *floor ?e*] **by** *simp*
also have ... = (*j dvd* $c{*}i$ $-$ $c{*}d$ + *floor ?e*)
using *dvd-period*[*OF id*, **where** *x*=$c{*}i$ **and** *c*=$-$*c* **and** *t*=*floor ?e*] **by** *simp*
also have ... = (*real j rdvd real* $(c{*}i$ $-$ $c{*}d$ + *floor ?e*))
using *int-rdvd-iff*[**where** *i*=*j* **and** *t*=$(c{*}i$ $-$ $c{*}d$ + *floor ?e*),*symmetric*, *simplified*]
ie **by** *simp*
also have ... = (*real j rdvd real* $(c{*}(i$ $-$ *d*)) + *?e*)
using *ie* **by** (*simp add*:*ring-simps*)
finally show *?case*
using *numbound0-I*[*OF bn*,**where** *b*=*real i* $-$ *real d* **and** *b'*=*real i* **and** *bs*=*bs*] *p*

by (*simp add*: *ring-simps*)
next
case (*10 j c e*) **hence** *p*: ¬ (*real j rdvd real* (*c*∗*i*) + *?N i e*) (**is** *?p x*) **and** *cp*: *c* > *0* **and** *bn*:*numbound0 e* **by** *simp*+
let *?e* = *Inum* (*real i* # *bs*) *e*
from *prems* **have** *isint e* (*real i* #*bs*) **by** *simp*
hence *ie*: *real* (*floor ?e*) = *?e* **using** *isint-iff* [**where** *n*=*e* **and** *bs*=(*real i*)#*bs*] *numbound0-I*[*OF bn*,**where** *b*=*real i* **and** *b′*=*real i* **and** *bs*=*bs*]
by (*simp add*: *isint-iff*)
from *prems* **have** *id*: *j dvd d* **by** *simp*
from *ie*[*symmetric*] **have** *?p i* = (¬ (*real j rdvd real* (*c*∗*i*+ *floor ?e*))) **by** *simp*
also have ... = *Not* (*j dvd c*∗*i* + *floor ?e*)
using *int-rdvd-iff* [**where** *i*=*j* **and** *t*=*c*∗*i*+ *floor ?e*] **by** *simp*
also have ... = *Not* (*j dvd c*∗*i* − *c*∗*d* + *floor ?e*)
using *dvd-period*[*OF id*, **where** *x*=*c*∗*i* **and** *c*=−*c* **and** *t*=*floor ?e*] **by** *simp*
also have ... = *Not* (*real j rdvd real* (*c*∗*i* − *c*∗*d* + *floor ?e*))
using *int-rdvd-iff*[**where** *i*=*j* **and** *t*=(*c*∗*i* − *c*∗*d* + *floor ?e*),*symmetric*, *simplified*]
ie **by** *simp*
also have ... = *Not* (*real j rdvd real* (*c*∗(*i* − *d*)) + *?e*)
using *ie* **by** (*simp add*:*ring-simps*)
finally show *?case*
using *numbound0-I*[*OF bn*,**where** *b*=*real i* − *real d* **and** *b′*=*real i* **and** *bs*=*bs*] *p*
by (*simp add*: *ring-simps*)
qed(*auto simp add*: *numbound0-I*[**where** *bs*=*bs* **and** *b*=*real i* − *real d* **and** *b′*=*real i*] *nth-pos2*)

lemma *σ-nb*: **assumes** *lp*: *iszlfm p* (*a*#*bs*) **and** *nb*: *numbound0 t*
shows *bound0* (*σ p k t*)
using *σϱ-nb*[*OF lp nb*] *nb* **by** (*simp add*: *σ-def*)

lemma *ϱ′*: **assumes** *lp*: *iszlfm p* (*a* #*bs*)
and *d*: *dδ p d*
and *dp*: *d* > *0*
shows ∀ *x*. ¬(∃ (*e*,*c*) ∈ *set*(*ϱ p*). ∃(*j*::*int*) ∈ {*1* .. *c*∗*d*}. *Ifm* (*a* #*bs*) (*σ p c* (*Add e* (*C j*)))) ⟶ *Ifm* (*real x*#*bs*) *p* ⟶ *Ifm* (*real* (*x* − *d*)#*bs*) *p* (**is** ∀ *x*. *?b x* ⟶ *?P x* ⟶ *?P* (*x* − *d*))
proof(*clarify*)
fix *x*
assume *nob1*:*?b x* **and** *px*: *?P x*
from *iszlfm-gen*[*OF lp*, *rule-format*, **where** *y*=*real x*] **have** *lp′*: *iszlfm p* (*real x*#*bs*).
have *nob*: ∀(*e*, *c*)∈*set* (*ϱ p*). ∀*j*∈{*1*..*c* ∗ *d*}. *real* (*c* ∗ *x*) ≠ *Inum* (*real x* # *bs*) *e* + *real j*
proof(*clarify*)
fix *e c j* **assume** *ecR*: (*e*,*c*) ∈ *set* (*ϱ p*) **and** *jD*: *j*∈ {*1* .. *c*∗*d*}
and *cx*: *real* (*c*∗*x*) = *Inum* (*real x*#*bs*) *e* + *real j*
let *?e* = *Inum* (*real x*#*bs*) *e*

let *?fe = floor ?e*
from *ϱ-l[OF lp'] ecR* **have** *ei:isint e (real x#bs)* **and** *cp:c>0* **and** *nb:numbound0 e*
by *auto*
from *numbound0-gen [OF nb ei, rule-format,***where** *y=a]* **have** *isint e (a#bs)*
.
from *cx ei[simplified isint-iff]* **have** *real (c*x) = real (?fe + j)* **by** *simp*
hence *cx: c*x = ?fe + j* **by** *(simp only: real-of-int-inject)*
hence *cdej:c dvd ?fe + j* **by** *(simp add: dvd-def) (rule-tac x=x* **in** *exI, simp)*
hence *real c rdvd real (?fe + j)* **by** *(simp only: int-rdvd-iff)*
hence *rcdej: real c rdvd ?e + real j* **by** *(simp add: ei[simplified isint-iff])*
from *cx* **have** *(c*x) div c = (?fe + j) div c* **by** *simp*
with *cp* **have** *x = (?fe + j) div c* **by** *simp*
with *px* **have** *th: ?P ((?fe + j) div c)* **by** *auto*
from *cp* **have** *cp': real c > 0* **by** *simp*
from *cdej* **have** *cdej': c dvd floor (Inum (real x#bs) (Add e (C j)))* **by** *simp*
from *nb* **have** *nb': numbound0 (Add e (C j))* **by** *simp*
have *ji: isint (C j) (real x#bs)* **by** *(simp add: isint-def)*
from *isint-add[OF ei ji]* **have** *ei':isint (Add e (C j)) (real x#bs)* **.**
from *th σϱ[***where** *b'=real x, OF lp' cp' nb' ei' cdej',symmetric]*
have *Ifm (real x#bs) (σϱ p (Add e (C j), c))* **by** *simp*
with *rcdej* **have** *th: Ifm (real x#bs) (σ p c (Add e (C j)))* **by** *(simp add: σ-def)*
from *th bound0-I[OF σ-nb[OF lp nb', ***where** *k=c],***where** *bs=bs* **and** *b=real x* **and** *b'=a]*
have *Ifm (a#bs) (σ p c (Add e (C j)))* **by** *blast*
with *ecR jD nob1* **show** *False* **by** *blast*
qed
from *ϱ[OF lp' px d dp nob]* **show** *?P (x −d)* **.**
qed

lemma *rl-thm*:
assumes *lp: iszlfm p (real (i::int)#bs)*
shows *(∃ (x::int). Ifm (real x#bs) p) = ((∃ j∈ {1 .. δ p}. Ifm (real j#bs) (minusinf p)) ∨ (∃ (e,c) ∈ set (ϱ p). ∃ j∈ {1 .. c*(δ p)}. Ifm (a#bs) (σ p c (Add e (C j)))))*
(is *(∃(x::int). ?P x) = ((∃ j∈ {1.. δ p}. ?MP j)∨(∃ (e,c) ∈ ?R. ∃ j∈ -. ?SP c e j))*
is *?lhs = (?MD ∨ ?RD)* **is** *?lhs = ?rhs)*
proof−
let *?d= δ p*
from *δ[OF lp]* **have** *d:dδ p ?d* **and** *dp: ?d > 0* **by** *auto*
{ assume *H:?MD* **hence** *th:∃ (x::int). ?MP x* **by** *blast*
from *H minusinf-ex[OF lp th]* **have** *?thesis* **by** *blast***}**
moreover
{ fix *e c j* **assume** *exR:(e,c) ∈ ?R* **and** *jD:j∈ {1 .. c*?d}* **and** *spx:?SP c e j*
from *exR ϱ-l[OF lp]* **have** *nb: numbound0 e* **and** *ei:isint e (real i#bs)* **and** *cp: c > 0*

by *auto*
have *isint* (*C j*) (*real i#bs*) **by** (*simp add*: *isint-iff*)
with *isint-add*[*OF numbound0-gen*[*OF nb ei*,*rule-format*, **where** *y=real i*]]
have *eji*:*isint* (*Add e* (*C j*)) (*real i#bs*) **by** *simp*
from *nb* **have** *nb'*: *numbound0* (*Add e* (*C j*)) **by** *simp*
from *spx bound0-I*[*OF σ-nb*[*OF lp nb'*, **where** *k=c*], **where** *bs=bs* **and** *b=a* **and** *b'=real i*]
have *spx'*: *Ifm* (*real i # bs*) (*σ p c* (*Add e* (*C j*))) **by** *blast*
from *spx'* **have** *rcdej*:*real c rdvd* (*Inum* (*real i#bs*) (*Add e* (*C j*)))
and *sr*:*Ifm* (*real i#bs*) (*σϱ p* (*Add e* (*C j*),*c*)) **by** (*simp add*: *σ-def*)+
from *rcdej eji*[*simplified isint-iff*]
have *real c rdvd real* (*floor* (*Inum* (*real i#bs*) (*Add e* (*C j*)))) **by** *simp*
hence *cdej*:*c dvd floor* (*Inum* (*real i#bs*) (*Add e* (*C j*))) **by** (*simp only*: *int-rdvd-iff*)
from *cp* **have** *cp'*: *real c* > *0* **by** *simp*
from *σϱ*[*OF lp cp' nb' eji cdej*] *spx'* **have** *?P* (⌊*Inum* (*real i # bs*) (*Add e* (*C j*))⌋ *div c*)
by (*simp add*: *σ-def*)
hence *?lhs* **by** *blast*
with *exR jD spx* **have** *?thesis* **by** *blast*}
moreover
{ **fix** *x* **assume** *px*: *?P x* **and** *nob*: ¬ *?RD*
from *iszlfm-gen* [*OF lp*,*rule-format*, **where** *y=a*] **have** *lp'*:*iszlfm p* (*a#bs*) .
from *ϱ'*[*OF lp' d dp*, *rule-format*, *OF nob*] **have** *th*:∀ *x*. *?P x* ⟶ *?P* (*x* − *?d*) **by** *blast*
from *minusinf-inf*[*OF lp*] **obtain** *z* **where** *z*:∀ *x*<*z*. *?MP x* = *?P x* **by** *blast*
have *zp*: *abs* (*x* − *z*) + *1* ≥ *0* **by** *arith*
from *decr-lemma*[*OF dp*,**where** *x=x* **and** *z=z*]
decr-mult-lemma[*OF dp th zp*, *rule-format*, *OF px*] *z* **have** *th*:∃ *x*. *?MP x* **by** *auto*
with *minusinf-bex*[*OF lp*] *px nob* **have** *?thesis* **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

lemma *mirror-αϱ*: **assumes** *lp*: *iszlfm p* (*a#bs*)
shows (λ (*t*,*k*). (*Inum* (*a#bs*) *t*, *k*)) ‘ *set* (*αϱ p*) = (λ (*t*,*k*). (*Inum* (*a#bs*) *t*,*k*)) ‘ *set* (*ϱ* (*mirror p*))
using *lp*
by (*induct p rule*: *mirror.induct*, *simp-all add*: *split-def image-Un*)

The ℝ part

Linearity for fm where Bound 0 ranges over ℝ

consts
isrlfm :: *fm* ⇒ *bool*
recdef *isrlfm measure size*
isrlfm (*And p q*) = (*isrlfm p* ∧ *isrlfm q*)
isrlfm (*Or p q*) = (*isrlfm p* ∧ *isrlfm q*)
isrlfm (*Eq* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*NEq* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)

$isrlfm\ (Lt\ \ (CN\ 0\ c\ e)) = (c{>}0 \wedge numbound0\ e)$
$isrlfm\ (Le\ \ (CN\ 0\ c\ e)) = (c{>}0 \wedge numbound0\ e)$
$isrlfm\ (Gt\ \ (CN\ 0\ c\ e)) = (c{>}0 \wedge numbound0\ e)$
$isrlfm\ (Ge\ \ (CN\ 0\ c\ e)) = (c{>}0 \wedge numbound0\ e)$
$isrlfm\ p = (isatom\ p \wedge (bound0\ p))$

constdefs $fp :: fm \Rightarrow int \Rightarrow num \Rightarrow int \Rightarrow fm$
$fp\ p\ n\ s\ j \equiv (if\ n > 0\ then$
$(And\ p\ (And\ (Ge\ (CN\ 0\ n\ (Sub\ s\ (Add\ (Floor\ s)\ (C\ j)))))$
$(Lt\ (CN\ 0\ n\ (Sub\ s\ (Add\ (Floor\ s)\ (C\ (j{+}1))))))))$
$else$
$(And\ p\ (And\ (Le\ (CN\ 0\ (-n)\ (Add\ (Neg\ s)\ (Add\ (Floor\ s)\ (C\ j)))))$
$(Gt\ (CN\ 0\ (-n)\ (Add\ (Neg\ s)\ (Add\ (Floor\ s)\ (C\ (j + 1)))))))))$

consts $rsplit0 :: num \Rightarrow (fm \times int \times num)\ list$
recdef $rsplit0\ measure\ num\text{-}size$
$rsplit0\ (Bound\ 0) = [(T,1,C\ 0)]$
$rsplit0\ (Add\ a\ b) = (let\ acs = rsplit0\ a\ ;\ bcs = rsplit0\ b$
$in\ map\ (\lambda\ ((p,n,t),(q,m,s)).\ (And\ p\ q,\ n{+}m,\ Add\ t\ s))\ [(a,b).\ a{\leftarrow}acs,b{\leftarrow}bcs])$
$rsplit0\ (Sub\ a\ b) = rsplit0\ (Add\ a\ (Neg\ b))$
$rsplit0\ (Neg\ a) = map\ (\lambda\ (p,n,s).\ (p,-n,Neg\ s))\ (rsplit0\ a)$
$rsplit0\ (Floor\ a) = foldl\ (op\ @)\ []\ (map$
$(\lambda\ (p,n,s).\ if\ n{=}0\ then\ [(p,0,Floor\ s)]$
$else\ (map\ (\lambda\ j.\ (fp\ p\ n\ s\ j,\ 0,\ Add\ (Floor\ s)\ (C\ j)))\ (if\ n > 0\ then\ iupt\ (0,n)\ else\ iupt(n,0))))$
$(rsplit0\ a))$
$rsplit0\ (CN\ 0\ c\ a) = map\ (\lambda\ (p,n,s).\ (p,n{+}c,s))\ (rsplit0\ a)$
$rsplit0\ (CN\ m\ c\ a) = map\ (\lambda\ (p,n,s).\ (p,n,CN\ m\ c\ s))\ (rsplit0\ a)$
$rsplit0\ (CF\ c\ t\ s) = rsplit0\ (Add\ (Mul\ c\ (Floor\ t))\ s)$
$rsplit0\ (Mul\ c\ a) = map\ (\lambda\ (p,n,s).\ (p,c{*}n,Mul\ c\ s))\ (rsplit0\ a)$
$rsplit0\ t = [(T,0,t)]$

lemma *not-rl*[*simp*]: $isrlfm\ p \Longrightarrow isrlfm\ (not\ p)$
by (*induct p rule*: *isrlfm.induct*, *auto*)
lemma *conj-rl*[*simp*]: $isrlfm\ p \Longrightarrow isrlfm\ q \Longrightarrow isrlfm\ (conj\ p\ q)$
using *conj-def* **by** (*cases p*, *auto*)
lemma *disj-rl*[*simp*]: $isrlfm\ p \Longrightarrow isrlfm\ q \Longrightarrow isrlfm\ (disj\ p\ q)$
using *disj-def* **by** (*cases p*, *auto*)

lemma *rsplit0-cs*:
shows $\forall\ (p,n,s) \in set\ (rsplit0\ t).$
$(Ifm\ (x\#bs)\ p \longrightarrow\ (Inum\ (x\#bs)\ t = Inum\ (x\#bs)\ (CN\ 0\ n\ s))) \wedge numbound0\ s \wedge isrlfm\ p$
(**is** $\forall\ (p,n,s) \in\ ?SS\ t.\ (?I\ p \longrightarrow\ ?N\ t =\ ?N\ (CN\ 0\ n\ s)) \wedge - \wedge -$)
proof(*induct t rule*: *rsplit0.induct*)
case (*5 a*)

let *?p* = λ (*p*,*n*,*s*) *j*. *fp p n s j*
let *?f* = (λ (*p*,*n*,*s*) *j*. (*?p* (*p*,*n*,*s*) *j*, (*0*::*int*),*Add* (*Floor s*) (*C j*)))
let *?J* = λ *n*. *if n*>*0 then iupt* (*0*,*n*) *else iupt* (*n*,*0*)
let *?ff*= (λ (*p*,*n*,*s*). *if n*= *0 then* [(*p*,*0*,*Floor s*)] *else map* (*?f* (*p*,*n*,*s*)) (*?J n*))
have *int-cases*: ∀ (*i*::*int*). *i*= *0* ∨ *i* < *0* ∨ *i* > *0* **by** *arith*
have *U1*: (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) =
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). *set* [(*p*,*0*,*Floor s*)])) **by** *auto*
have *U2′*: ∀ (*p*,*n*,*s*) ∈ {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*}.
?ff (*p*,*n*,*s*) = *map* (*?f*(*p*,*n*,*s*)) (*iupt*(*0*,*n*)) **by** *auto*
hence *U2*: (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) =
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*).
set (*map* (*?f*(*p*,*n*,*s*)) (*iupt*(*0*,*n*)))))
proof−
fix *M* :: (′*a*×′*b*×′*c*) *set* **and** *f* :: (′*a*×′*b*×′*c*) ⇒ ′*d list* **and** *g*
assume ∀ (*a*,*b*,*c*) ∈ *M*. *f* (*a*,*b*,*c*) = *g a b c*
thus (*UNION M* (λ (*a*,*b*,*c*). *set* (*f* (*a*,*b*,*c*)))) = (*UNION M* (λ (*a*,*b*,*c*). *set* (*g a b c*)))
by (*auto simp add*: *split-def*)
qed
have *U3′*: ∀ (*p*,*n*,*s*) ∈ {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*}. *?ff* (*p*,*n*,*s*) = *map* (*?f*(*p*,*n*,*s*)) (*iupt*(*n*,*0*))
by *auto*
hence *U3*: (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) =
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*)∈ *?SS a*∧*n*<*0*} (λ(*p*,*n*,*s*). *set* (*map* (*?f*(*p*,*n*,*s*)) (*iupt*(*n*,*0*)))))
proof−
fix *M* :: (′*a*×′*b*×′*c*) *set* **and** *f* :: (′*a*×′*b*×′*c*) ⇒ ′*d list* **and** *g*
assume ∀ (*a*,*b*,*c*) ∈ *M*. *f* (*a*,*b*,*c*) = *g a b c*
thus (*UNION M* (λ (*a*,*b*,*c*). *set* (*f* (*a*,*b*,*c*)))) = (*UNION M* (λ (*a*,*b*,*c*). *set* (*g a b c*)))
by (*auto simp add*: *split-def*)
qed
have *?SS* (*Floor a*) = *UNION* (*?SS a*) (λ*x*. *set* (*?ff x*))
by (*auto simp add*: *foldl-conv-concat*)
also have . . . = *UNION* (*?SS a*) (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*))) **by** *auto*
also have . . . =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))))
using *int-cases*[*rule-format*] **by** *blast*
also have . . . =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). *set* [(*p*,*0*,*Floor s*)])) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*)∈ *?SS a*∧*n*>*0*} (λ(*p*,*n*,*s*). *set*(*map*(*?f*(*p*,*n*,*s*)) (*iupt*(*0*,*n*))))) *Un*

(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*).
set (*map* (*?f*(*p*,*n*,*s*)) (*iupt*(*n*,*0*)))))) **by** (*simp only*: *U1 U2 U3*)
also have ... =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). {(*p*,*0*,*Floor s*)})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). (*?f*(*p*,*n*,*s*)) ' {*0* .. *n*}))
Un
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). (*?f*(*p*,*n*,*s*)) ' {*n* .. *0*})))
by (*simp only*: *set-map iupt-set set.simps*)
also have ... =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). {(*p*,*0*,*Floor s*)})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). {*?f*(*p*,*n*,*s*) *j*| *j*. *j*∈ {*0*
.. *n*}})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). {*?f*(*p*,*n*,*s*) *j*| *j*. *j*∈ {*n*
.. *0*}}))) **by** *blast*
finally
have *FS*: *?SS* (*Floor a*) =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). {(*p*,*0*,*Floor s*)})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). {*?f*(*p*,*n*,*s*) *j*| *j*. *j*∈ {*0*
.. *n*}})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). {*?f*(*p*,*n*,*s*) *j*| *j*. *j*∈ {*n*
.. *0*}}))) **by** *blast*
show *?case*
proof(*simp only*: *FS*, *clarsimp simp del*: *Ifm.simps Inum.simps*, −)
fix *p n s*
let *?ths* = (*?I p* ⟶ (*?N* (*Floor a*) = *?N* (*CN 0 n s*))) ∧ *numbound0 s* ∧
isrlfm p
assume (∃ *ba*. (*p*, *0*, *ba*) ∈ *set* (*rsplit0 a*) ∧ *n* = *0* ∧ *s* = *Floor ba*) ∨
(∃ *ab ac ba*.
(*ab*, *ac*, *ba*) ∈ *set* (*rsplit0 a*) ∧
0 < *ac* ∧
(∃ *j*. *p* = *fp ab ac ba j* ∧
n = *0* ∧ *s* = *Add* (*Floor ba*) (*C j*) ∧ *0* ≤ *j* ∧ *j* ≤ *ac*)) ∨
(∃ *ab ac ba*.
(*ab*, *ac*, *ba*) ∈ *set* (*rsplit0 a*) ∧
ac < *0* ∧
(∃ *j*. *p* = *fp ab ac ba j* ∧
n = *0* ∧ *s* = *Add* (*Floor ba*) (*C j*) ∧ *ac* ≤ *j* ∧ *j* ≤ *0*))
moreover
{**fix** *s*′
assume (*p*, *0*, *s*′) ∈ *?SS a* **and** *n* = *0* **and** *s* = *Floor s*′
hence *?ths* **using** *prems* **by** *auto*}
moreover
{ **fix** *p*′ *n*′ *s*′ *j*
assume *pns*: (*p*′, *n*′, *s*′) ∈ *?SS a*
and *np*: *0* < *n*′
and *p-def*: *p* = *?p* (*p*′,*n*′,*s*′) *j*
and *n0*: *n* = *0*
and *s-def*: *s* = (*Add* (*Floor s*′) (*C j*))
and *jp*: *0* ≤ *j* **and** *jn*: *j* ≤ *n*′

from *prems pns* **have** *H*:(*Ifm* ((*x*::*real*) # (*bs*::*real list*)) *p'* ⟶
Inum (*x* # *bs*) *a* = *Inum* (*x* # *bs*) (*CN 0 n' s'*)) ∧
numbound0 s' ∧ *isrlfm p'* **by** *blast*
hence *nb*: *numbound0 s'* **by** *simp*
from *H* **have** *nf*: *isrlfm* (*?p* (*p'*,*n'*,*s'*) *j*) **using** *fp-def np* **by** (*simp add*: *numsub-nb*)
let *?nxs* = *CN 0 n' s'*
let *?l* = *floor* (*?N s'*) + *j*
from *H*
have *?I* (*?p* (*p'*,*n'*,*s'*) *j*) ⟶
(((*?N ?nxs* ≥ *real ?l*) ∧ (*?N ?nxs* < *real* (*?l* + *1*))) ∧ (*?N a* = *?N ?nxs*))
by (*simp add*: *fp-def np ring-simps numsub numadd numfloor*)
also have ... ⟶ ((*floor* (*?N ?nxs*) = *?l*) ∧ (*?N a* = *?N ?nxs*))
using *floor-int-eq*[**where** *x*=*?N ?nxs* **and** *n*=*?l*] **by** *simp*
moreover
have ... ⟶ (*?N* (*Floor a*) = *?N* ((*Add* (*Floor s'*) (*C j*)))) **by** *simp*
ultimately have *?I* (*?p* (*p'*,*n'*,*s'*) *j*) ⟶ (*?N* (*Floor a*) = *?N* ((*Add* (*Floor s'*) (*C j*))))
by *blast*
with *s-def n0 p-def nb nf* **have** *?ths* **by** *auto*}
moreover
{**fix** *p' n' s' j*
assume *pns*: (*p'*, *n'*, *s'*) ∈ *?SS a*
and *np*: *n'* < *0*
and *p-def*: *p* = *?p* (*p'*,*n'*,*s'*) *j*
and *n0*: *n* = *0*
and *s-def*: *s* = (*Add* (*Floor s'*) (*C j*))
and *jp*: *n'* ≤ *j* **and** *jn*: *j* ≤ *0*
from *prems pns* **have** *H*:(*Ifm* ((*x*::*real*) # (*bs*::*real list*)) *p'* ⟶
Inum (*x* # *bs*) *a* = *Inum* (*x* # *bs*) (*CN 0 n' s'*)) ∧
numbound0 s' ∧ *isrlfm p'* **by** *blast*
hence *nb*: *numbound0 s'* **by** *simp*
from *H* **have** *nf*: *isrlfm* (*?p* (*p'*,*n'*,*s'*) *j*) **using** *fp-def np* **by** (*simp add*: *numneg-nb*)
let *?nxs* = *CN 0 n' s'*
let *?l* = *floor* (*?N s'*) + *j*
from *H*
have *?I* (*?p* (*p'*,*n'*,*s'*) *j*) ⟶
(((*?N ?nxs* ≥ *real ?l*) ∧ (*?N ?nxs* < *real* (*?l* + *1*))) ∧ (*?N a* = *?N ?nxs*))
by (*simp add*: *np fp-def ring-simps numneg numfloor numadd numsub*)
also have ... ⟶ ((*floor* (*?N ?nxs*) = *?l*) ∧ (*?N a* = *?N ?nxs*))
using *floor-int-eq*[**where** *x*=*?N ?nxs* **and** *n*=*?l*] **by** *simp*
moreover
have ... ⟶ (*?N* (*Floor a*) = *?N* ((*Add* (*Floor s'*) (*C j*)))) **by** *simp*
ultimately have *?I* (*?p* (*p'*,*n'*,*s'*) *j*) ⟶ (*?N* (*Floor a*) = *?N* ((*Add* (*Floor s'*) (*C j*))))
by *blast*

with *s-def n0 p-def nb nf* **have** *?ths* **by** *auto*}
ultimately show *?ths* **by** *auto*
qed
next
case (*3 a b*) **thus** *?case* **by** *auto*
qed (*auto simp add*: *Let-def split-def ring-simps conj-rl*)

lemma *real-in-int-intervals*:
assumes *xb*: *real m* ≤ *x* ∧ *x* < *real* ((*n*::*int*) + *1*)
shows ∃ *j*∈ {*m*.. *n*}. *real j* ≤ *x* ∧ *x* < *real* (*j*+*1*) (**is** ∃ *j*∈ *?N*. *?P j*)
by (*rule bexI*[**where** *P*=*?P* **and** *x*=*floor x* **and** *A*=*?N*])
(*auto simp add*: *floor-less-eq*[**where** *x*=*x* **and** *a*=*n*+*1*, *simplified*] *xb*[*simplified*] *floor-mono2*[**where** *x*=*real m* **and** *y*=*x*, *OF conjunct1*[*OF xb*], *simplified floor-real-of-int*[**where** *n*=*m*]])

lemma *rsplit0-complete*:
assumes *xp*:*0* ≤ *x* **and** *x1*:*x* < *1*
shows ∃ (*p*,*n*,*s*) ∈ *set* (*rsplit0 t*). *Ifm* (*x*#*bs*) *p* (**is** ∃ (*p*,*n*,*s*) ∈ *?SS t*. *?I p*)
proof(*induct t rule*: *rsplit0.induct*)
case (*2 a b*)
from *prems* **have** ∃ (*pa*,*na*,*sa*) ∈ *?SS a*. *?I pa* **by** *auto*
then obtain *pa na sa* **where** *pa*: (*pa*,*na*,*sa*)∈ *?SS a* ∧ *?I pa* **by** *blast*
from *prems* **have** ∃ (*pb*,*nb*,*sb*) ∈ *?SS b*. *?I pb* **by** *auto*
then obtain *pb nb sb* **where** *pb*: (*pb*,*nb*,*sb*)∈ *?SS b* ∧ *?I pb* **by** *blast*
from *pa pb* **have** *th*: ((*pa*,*na*,*sa*),(*pb*,*nb*,*sb*)) ∈ *set*[(*x*,*y*). *x*←*rsplit0 a*, *y*←*rsplit0 b*]
by (*auto*)
let *?f*=(λ ((*p*,*n*,*t*),(*q*,*m*,*s*)). (*And p q*, *n*+*m*, *Add t s*))
from *imageI*[*OF th*, **where** *f*=*?f*] **have** *?f* ((*pa*,*na*,*sa*),(*pb*,*nb*,*sb*)) ∈ *?SS* (*Add a b*)
by (*simp add*: *Let-def*)
hence (*And pa pb*, *na* +*nb*, *Add sa sb*) ∈ *?SS* (*Add a b*) **by** *simp*
moreover from *pa pb* **have** *?I* (*And pa pb*) **by** *simp*
ultimately show *?case* **by** *blast*
next
case (*5 a*)
let *?p* = λ (*p*,*n*,*s*) *j*. *fp p n s j*
let *?f* = (λ (*p*,*n*,*s*) *j*. (*?p* (*p*,*n*,*s*) *j*, (*0*::*int*),(*Add* (*Floor s*) (*C j*))))
let *?J* = λ *n*. *if n*>*0 then iupt* (*0*,*n*) *else iupt* (*n*,*0*)
let *?ff*= (λ (*p*,*n*,*s*). *if n*= *0 then* [(*p*,*0*,*Floor s*)] *else map* (*?f* (*p*,*n*,*s*)) (*?J n*))
have *int-cases*: ∀ (*i*::*int*). *i*= *0* ∨ *i* < *0* ∨ *i* > *0* **by** *arith*
have *U1*: (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) = (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). *set* [(*p*,*0*,*Floor s*)])) **by** *auto*
have *U2'*: ∀ (*p*,*n*,*s*) ∈ {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*}. *?ff* (*p*,*n*,*s*) = *map* (*?f*(*p*,*n*,*s*)) (*iupt*(*0*,*n*))
by *auto*
hence *U2*: (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) = (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). *set* (*map*

(*?f*(*p*,*n*,*s*)) (*iupt*(*0*,*n*)))))
proof−
fix *M* :: ('*a*×'*b*×'*c*) *set* **and** *f* :: ('*a*×'*b*×'*c*) ⇒ '*d list* **and** *g*
assume ∀ (*a*,*b*,*c*) ∈ *M*. *f* (*a*,*b*,*c*) = *g a b c*
thus (*UNION M* (λ (*a*,*b*,*c*). *set* (*f* (*a*,*b*,*c*)))) = (*UNION M* (λ (*a*,*b*,*c*). *set* (*g a b c*)))
by (*auto simp add*: *split-def*)
qed
have *U3'*: ∀ (*p*,*n*,*s*) ∈ {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*}. *?ff* (*p*,*n*,*s*) = *map* (*?f*(*p*,*n*,*s*)) (*iupt*(*n*,*0*))
by *auto*
hence *U3*: (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) = (*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). *set* (*map* (*?f*(*p*,*n*,*s*)) (*iupt*(*n*,*0*)))))
proof−
fix *M* :: ('*a*×'*b*×'*c*) *set* **and** *f* :: ('*a*×'*b*×'*c*) ⇒ '*d list* **and** *g*
assume ∀ (*a*,*b*,*c*) ∈ *M*. *f* (*a*,*b*,*c*) = *g a b c*
thus (*UNION M* (λ (*a*,*b*,*c*). *set* (*f* (*a*,*b*,*c*)))) = (*UNION M* (λ (*a*,*b*,*c*). *set* (*g a b c*)))
by (*auto simp add*: *split-def*)
qed

have *?SS* (*Floor a*) = *UNION* (*?SS a*) (λ*x*. *set* (*?ff x*)) **by** (*auto simp add*: *foldl-conv-concat*)
also have . . . = *UNION* (*?SS a*) (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*))) **by** *auto*
also have . . . =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). *set* (*?ff* (*p*,*n*,*s*)))))
using *int-cases*[*rule-format*] **by** *blast*
also have . . . =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). *set* [(*p*,*0*,*Floor s*)])) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). *set* (*map* (*?f*(*p*,*n*,*s*)) (*iupt*(*0*,*n*))))) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). *set* (*map* (*?f*(*p*,*n*,*s*)) (*iupt*(*n*,*0*)))))) **by** (*simp only*: *U1 U2 U3*)
also have . . . =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). {(*p*,*0*,*Floor s*)})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). (*?f*(*p*,*n*,*s*)) ' {*0* .. *n*})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). (*?f*(*p*,*n*,*s*)) ' {*n* .. *0*})))
by (*simp only*: *set-map iupt-set set.simps*)
also have . . . =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). {(*p*,*0*,*Floor s*)})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). {*?f*(*p*,*n*,*s*) *j*| *j*. *j*∈ {*0* .. *n*}})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). {*?f*(*p*,*n*,*s*) *j*| *j*. *j*∈ {*n* .. *0*}}))) **by** *blast*

finally
have *FS*: *?SS* (*Floor a*) =
((*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*=*0*} (λ (*p*,*n*,*s*). {(*p*,*0*,*Floor s*)})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*>*0*} (λ (*p*,*n*,*s*). {*?f*(*p*,*n*,*s*) *j*| *j*. *j*∈ {*0* .. *n*}})) *Un*
(*UNION* {(*p*,*n*,*s*). (*p*,*n*,*s*) ∈ *?SS a* ∧ *n*<*0*} (λ (*p*,*n*,*s*). {*?f*(*p*,*n*,*s*) *j*| *j*. *j*∈ {*n* .. *0*}}))) **by** *blast*
from *prems* **have** ∃ (*p*,*n*,*s*) ∈ *?SS a*. *?I p* **by** *auto*
then obtain *p n s* **where** *pns*: (*p*,*n*,*s*) ∈ *?SS a* ∧ *?I p* **by** *blast*
let *?N* = λ *t*. *Inum* (*x*#*bs*) *t*
from *rsplit0-cs*[*rule-format*] *pns* **have** *ans*:(*?N a* = *?N* (*CN 0 n s*)) ∧ *numbound0 s* ∧ *isrlfm p*
by *auto*

have *n*=*0* ∨ *n* >*0* ∨ *n* <*0* **by** *arith*
moreover {**assume** *n*=*0* **hence** *?case* **using** *pns* **by** (*simp only*: *FS*) *auto* }
moreover
{
assume *np*: *n* > *0*
from *real-of-int-floor-le*[**where** *r*=*?N s*] **have** *?N* (*Floor s*) ≤ *?N s* **by** *simp*
also from *mult-left-mono*[*OF xp*] *np* **have** *?N s* ≤ *real n* ∗ *x* + *?N s* **by** *simp*
finally have *?N* (*Floor s*) ≤ *real n* ∗ *x* + *?N s* **.**
moreover
{**from** *mult-strict-left-mono*[*OF x1*] *np*
have *real n* ∗*x* + *?N s* < *real n* + *?N s* **by** *simp*
also from *real-of-int-floor-add-one-gt*[**where** *r*=*?N s*]
have . . . < *real n* + *?N* (*Floor s*) + *1* **by** *simp*
finally have *real n* ∗*x* + *?N s* < *?N* (*Floor s*) + *real* (*n*+*1*) **by** *simp*}
ultimately have *?N* (*Floor s*) ≤ *real n* ∗*x* + *?N s*∧ *real n* ∗*x* + *?N s* < *?N* (*Floor s*) + *real* (*n*+*1*) **by** *simp*
hence *th*: *0* ≤ *real n* ∗*x* + *?N s* − *?N* (*Floor s*) ∧ *real n* ∗*x* + *?N s* − *?N* (*Floor s*) < *real* (*n*+*1*) **by** *simp*
from *real-in-int-intervals th* **have** ∃ *j*∈ {*0* .. *n*}. *real j* ≤ *real n* ∗*x* + *?N s* − *?N* (*Floor s*)∧ *real n* ∗*x* + *?N s* − *?N* (*Floor s*) < *real* (*j*+*1*) **by** *simp*

hence ∃ *j*∈ {*0* .. *n*}. *0* ≤ *real n* ∗*x* + *?N s* − *?N* (*Floor s*) − *real j* ∧ *real n* ∗*x* + *?N s* − *?N* (*Floor s*) − *real* (*j*+*1*) < *0*
by(*simp only*: *myl*[*rule-format*, **where** *b*=*real n* ∗ *x* + *Inum* (*x* # *bs*) *s* − *Inum* (*x* # *bs*) (*Floor s*)] *less-iff-diff-less-0*[**where** *a*=*real n* ∗*x* + *?N s* − *?N* (*Floor s*)])
hence ∃ *j*∈ {*0*.. *n*}. *?I* (*?p* (*p*,*n*,*s*) *j*)
using *pns* **by** (*simp add*: *fp-def np ring-simps numsub numadd*)
then obtain *j* **where** *j-def*: *j*∈ {*0* .. *n*} ∧ *?I* (*?p* (*p*,*n*,*s*) *j*) **by** *blast*
hence ∃*x* ∈ {*?p* (*p*,*n*,*s*) *j* |*j*. *0*≤ *j* ∧ *j* ≤ *n* }. *?I x* **by** *auto*
hence *?case* **using** *pns*
by (*simp only*: *FS*,*simp add*: *bex-Un*)
(*rule disjI2*, *rule disjI1*,*rule exI* [**where** *x*=*p*],
rule exI [**where** *x*=*n*],*rule exI* [**where** *x*=*s*],*simp-all add*: *np*)
}

moreover
{ **assume** *nn*: $n < 0$ **hence** *np*: $-n > 0$ **by** *simp*
from *real-of-int-floor-le*[**where** $r=?N\ s$] **have** $?N\ (Floor\ s) + 1 > ?N\ s$ **by** *simp*
moreover from *mult-left-mono-neg*[*OF xp*] *nn* **have** $?N\ s \geq real\ n * x + ?N\ s$ **by** *simp*
ultimately have $?N\ (Floor\ s) + 1 > real\ n * x + ?N\ s$ **by** *arith*
moreover
{**from** *mult-strict-left-mono-neg*[*OF x1*, **where** $c=real\ n$] *nn*
have $real\ n *x + ?N\ s \geq real\ n + ?N\ s$ **by** *simp*
moreover from *real-of-int-floor-le*[**where** $r=?N\ s$] **have** $real\ n + ?N\ s \geq real\ n + ?N\ (Floor\ s)$ **by** *simp*
ultimately have $real\ n *x + ?N\ s \geq ?N\ (Floor\ s) + real\ n$
by (*simp only*: *ring-simps*)}
ultimately have $?N\ (Floor\ s) + real\ n \leq real\ n *x + ?N\ s \wedge real\ n *x + ?N\ s < ?N\ (Floor\ s) + real\ (1::int)$ **by** *simp*
hence *th*: $real\ n \leq real\ n *x + ?N\ s - ?N\ (Floor\ s) \wedge real\ n *x + ?N\ s - ?N\ (Floor\ s) < real\ (1::int)$ **by** *simp*
have *th1*: $\forall\ (a::real).\ (-\ a > 0) = (a < 0)$ **by** *auto*
have *th2*: $\forall\ (a::real).\ (0 \geq -\ a) = (a \geq 0)$ **by** *auto*
from *real-in-int-intervals th* **have** $\exists\ j\in \{n\ ..\ 0\}.\ real\ j \leq real\ n *x + ?N\ s - ?N\ (Floor\ s) \wedge real\ n *x + ?N\ s - ?N\ (Floor\ s) < real\ (j+1)$ **by** *simp*

hence $\exists\ j\in \{n\ ..\ 0\}.\ 0 \leq real\ n *x + ?N\ s - ?N\ (Floor\ s) - real\ j \wedge real\ n *x + ?N\ s - ?N\ (Floor\ s) - real\ (j+1) < 0$
by(*simp only*: *myl*[*rule-format*, **where** $b=real\ n * x + Inum\ (x\ \#\ bs)\ s - Inum\ (x\ \#\ bs)\ (Floor\ s)$] *less-iff-diff-less-0*[**where** $a=real\ n *x + ?N\ s - ?N\ (Floor\ s)$])
hence $\exists\ j\in \{n\ ..\ 0\}.\ 0 \geq -\ (real\ n *x + ?N\ s - ?N\ (Floor\ s) - real\ j) \wedge -\ (real\ n *x + ?N\ s - ?N\ (Floor\ s) - real\ (j+1)) > 0$ **by** (*simp only*: *th1*[*rule-format*] *th2*[*rule-format*])
hence $\exists\ j\in \{n..\ 0\}.\ ?I\ (?p\ (p,n,s)\ j)$
using *pns* **by** (*simp add*: *fp-def nn diff-def add-ac mult-ac numfloor numadd numneg*
del: *diff-less-0-iff-less diff-le-0-iff-le*)
then obtain *j* **where** *j-def*: $j\in \{n\ ..\ 0\} \wedge ?I\ (?p\ (p,n,s)\ j)$ **by** *blast*
hence $\exists x \in \{?p\ (p,n,s)\ j\ |j.\ n\leq j \wedge j \leq 0\ \}.\ ?I\ x$ **by** *auto*
hence *?case* **using** *pns*
by (*simp only*: *FS*,*simp add*: *bex-Un*)
(*rule disjI2*, *rule disjI2*,*rule exI* [**where** $x=p$],
rule exI [**where** $x=n$],*rule exI* [**where** $x=s$],*simp-all add*: *nn*)
}
ultimately show *?case* **by** *blast*
qed (*auto simp add*: *Let-def split-def*)

constdefs *rsplit* :: $(int \Rightarrow num \Rightarrow fm) \Rightarrow num \Rightarrow fm$
$rsplit\ f\ a \equiv foldr\ disj\ (map\ (\lambda\ (\varphi,\ n,\ s).\ conj\ \varphi\ (f\ n\ s))\ (rsplit0\ a))\ F$

lemma *foldr-disj-map*: *Ifm bs* (*foldr disj* (*map f xs*) *F*) = (∃ *x* ∈ *set xs*. *Ifm bs* (*f x*))
by(*induct xs*, *simp-all*)

lemma *foldr-conj-map*: *Ifm bs* (*foldr conj* (*map f xs*) *T*) = (∀ *x* ∈ *set xs*. *Ifm bs* (*f x*))
by(*induct xs*, *simp-all*)

lemma *foldr-disj-map-rlfm*:
  assumes *lf*: ∀ *n s*. *numbound0 s* ⟶ *isrlfm* (*f n s*)
  and *φ*: ∀ (*φ*,*n*,*s*) ∈ *set xs*. *numbound0 s* ∧ *isrlfm φ*
  shows *isrlfm* (*foldr disj* (*map* (λ (*φ*, *n*, *s*). *conj φ* (*f n s*)) *xs*) *F*)
using *lf φ* **by** (*induct xs*, *auto*)

lemma *rsplit-ex*: *Ifm bs* (*rsplit f a*) = (∃ (*φ*,*n*,*s*) ∈ *set* (*rsplit0 a*). *Ifm bs* (*conj φ* (*f n s*)))
using *foldr-disj-map*[**where** *xs*=*rsplit0 a*] *rsplit-def* **by** (*simp add*: *split-def*)

lemma *rsplit-l*: **assumes** *lf*: ∀ *n s*. *numbound0 s* ⟶ *isrlfm* (*f n s*)
  shows *isrlfm* (*rsplit f a*)
proof−
  from *rsplit0-cs*[**where** *t*=*a*] **have** *th*: ∀ (*φ*,*n*,*s*) ∈ *set* (*rsplit0 a*). *numbound0 s* ∧ *isrlfm φ* **by** *blast*
  from *foldr-disj-map-rlfm*[*OF lf th*] *rsplit-def* **show** *?thesis* **by** *simp*
qed

lemma *rsplit*:
  assumes *xp*: *x* ≥ *0* **and** *x1*: *x* < *1*
  and *f*: ∀ *a n s*. *Inum* (*x*#*bs*) *a* = *Inum* (*x*#*bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ (*Ifm* (*x*#*bs*) (*f n s*) = *Ifm* (*x*#*bs*) (*g a*))
  shows *Ifm* (*x*#*bs*) (*rsplit f a*) = *Ifm* (*x*#*bs*) (*g a*)
proof(*auto*)
  let *?I* = λ*x p*. *Ifm* (*x*#*bs*) *p*
  let *?N* = λ *x t*. *Inum* (*x*#*bs*) *t*
  assume *?I x* (*rsplit f a*)
  hence ∃ (*φ*,*n*,*s*) ∈ *set* (*rsplit0 a*). *?I x* (*And φ* (*f n s*)) **using** *rsplit-ex* **by** *simp*
  then obtain *φ n s* **where** *fnsS*:(*φ*,*n*,*s*) ∈ *set* (*rsplit0 a*) **and** *?I x* (*And φ* (*f n s*)) **by** *blast*
  hence *φ*: *?I x φ* **and** *fns*: *?I x* (*f n s*) **by** *auto*
  from *rsplit0-cs*[**where** *t*=*a* **and** *bs*=*bs* **and** *x*=*x*, *rule-format*, *OF fnsS*] *φ*
  have *th*: (*?N x a* = *?N x* (*CN 0 n s*)) ∧ *numbound0 s* **by** *auto*
  from *f*[*rule-format*, *OF th*] *fns* **show** *?I x* (*g a*) **by** *simp*
next
  let *?I* = λ*x p*. *Ifm* (*x*#*bs*) *p*
  let *?N* = λ *x t*. *Inum* (*x*#*bs*) *t*
  assume *ga*: *?I x* (*g a*)
  from *rsplit0-complete*[*OF xp x1*, **where** *bs*=*bs* **and** *t*=*a*]
  obtain *φ n s* **where** *fnsS*:(*φ*,*n*,*s*) ∈ *set* (*rsplit0 a*) **and** *fx*: *?I x φ* **by** *blast*

from *rsplit0-cs*[**where** $t=a$ **and** $x=x$ **and** $bs=bs$] *fnsS fx*
have *ans*: *?N x a* = *?N x* (*CN 0 n s*) **and** *nb*: *numbound0 s* **by** *auto*
with *ga f* **have** *?I x* (*f n s*) **by** *auto*
with *rsplit-ex fnsS fx* **show** *?I x* (*rsplit f a*) **by** *auto*
qed

definition *lt* :: *int* ⇒ *num* ⇒ *fm* **where**
lt-def: *lt c t* = (*if c* = *0 then* (*Lt t*) *else if c* > *0 then* (*Lt* (*CN 0 c t*))
else (*Gt* (*CN 0* (−*c*) (*Neg t*))))

definition *le* :: *int* ⇒ *num* ⇒ *fm* **where**
le-def: *le c t* = (*if c* = *0 then* (*Le t*) *else if c* > *0 then* (*Le* (*CN 0 c t*))
else (*Ge* (*CN 0* (−*c*) (*Neg t*))))

definition *gt* :: *int* ⇒ *num* ⇒ *fm* **where**
gt-def: *gt c t* = (*if c* = *0 then* (*Gt t*) *else if c* > *0 then* (*Gt* (*CN 0 c t*))
else (*Lt* (*CN 0* (−*c*) (*Neg t*))))

definition *ge* :: *int* ⇒ *num* ⇒ *fm* **where**
ge-def: *ge c t* = (*if c* = *0 then* (*Ge t*) *else if c* > *0 then* (*Ge* (*CN 0 c t*))
else (*Le* (*CN 0* (−*c*) (*Neg t*))))

definition *eq* :: *int* ⇒ *num* ⇒ *fm* **where**
eq-def: *eq c t* = (*if c* = *0 then* (*Eq t*) *else if c* > *0 then* (*Eq* (*CN 0 c t*))
else (*Eq* (*CN 0* (−*c*) (*Neg t*))))

definition *neq* :: *int* ⇒ *num* ⇒ *fm* **where**
neq-def: *neq c t* = (*if c* = *0 then* (*NEq t*) *else if c* > *0 then* (*NEq* (*CN 0 c t*))
else (*NEq* (*CN 0* (−*c*) (*Neg t*))))

lemma *lt-mono*: ∀ *a n s*. *Inum* (*x#bs*) *a* = *Inum* (*x#bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x#bs*) (*lt n s*) = *Ifm* (*x#bs*) (*Lt a*)
(**is** ∀ *a n s* . *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*lt n s*) = *?I* (*Lt a*))
proof(*clarify*)
fix *a n s*
assume *H*: *?N a* = *?N* (*CN 0 n s*)
show *?I* (*lt n s*) = *?I* (*Lt a*) **using** *H* **by** (*cases n*=*0*, (*simp add*: *lt-def*))
(*cases n* > *0*, *simp-all add*: *lt-def ring-simps myless*[*rule-format*, **where** *b*=*0*])
qed

lemma *lt-l*: *isrlfm* (*rsplit lt a*)
by (*rule rsplit-l*[**where** *f*=*lt* **and** *a*=*a*], *auto simp add*: *lt-def*,
case-tac s, *simp-all*, *case-tac nat*, *simp-all*)

lemma *le-mono*: ∀ *a n s*. *Inum* (*x#bs*) *a* = *Inum* (*x#bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x#bs*) (*le n s*) = *Ifm* (*x#bs*) (*Le a*) (**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*le n s*) = *?I* (*Le a*))
proof(*clarify*)
fix *a n s*

assume *H*: *?N a* = *?N* (*CN 0 n s*)
show *?I* (*le n s*) = *?I* (*Le a*) **using** *H* **by** (*cases n=0*, (*simp add*: *le-def*))
(*cases n* > *0*, *simp-all add*: *le-def ring-simps myl*[*rule-format*, **where** *b=0*])
qed

lemma *le-l*: *isrlfm* (*rsplit le a*)
by (*rule rsplit-l*[**where** *f=le* **and** *a=a*], *auto simp add*: *le-def*)
(*case-tac s*, *simp-all*, *case-tac nat*,*simp-all*)

lemma *gt-mono*: ∀ *a n s*. *Inum* (*x#bs*) *a* = *Inum* (*x#bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x#bs*) (*gt n s*) = *Ifm* (*x#bs*) (*Gt a*) (**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*gt n s*) = *?I* (*Gt a*))
proof(*clarify*)
fix *a n s*
assume *H*: *?N a* = *?N* (*CN 0 n s*)
show *?I* (*gt n s*) = *?I* (*Gt a*) **using** *H* **by** (*cases n=0*, (*simp add*: *gt-def*))
(*cases n* > *0*, *simp-all add*: *gt-def ring-simps myless*[*rule-format*, **where** *b=0*])
qed
lemma *gt-l*: *isrlfm* (*rsplit gt a*)
by (*rule rsplit-l*[**where** *f=gt* **and** *a=a*], *auto simp add*: *gt-def*)
(*case-tac s*, *simp-all*, *case-tac nat*, *simp-all*)

lemma *ge-mono*: ∀ *a n s*. *Inum* (*x#bs*) *a* = *Inum* (*x#bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x#bs*) (*ge n s*) = *Ifm* (*x#bs*) (*Ge a*) (**is** ∀ *a n s* . *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*ge n s*) = *?I* (*Ge a*))
proof(*clarify*)
fix *a n s*
assume *H*: *?N a* = *?N* (*CN 0 n s*)
show *?I* (*ge n s*) = *?I* (*Ge a*) **using** *H* **by** (*cases n=0*, (*simp add*: *ge-def*))
(*cases n* > *0*, *simp-all add*: *ge-def ring-simps myl*[*rule-format*, **where** *b=0*])
qed
lemma *ge-l*: *isrlfm* (*rsplit ge a*)
by (*rule rsplit-l*[**where** *f=ge* **and** *a=a*], *auto simp add*: *ge-def*)
(*case-tac s*, *simp-all*, *case-tac nat*, *simp-all*)

lemma *eq-mono*: ∀ *a n s*. *Inum* (*x#bs*) *a* = *Inum* (*x#bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x#bs*) (*eq n s*) = *Ifm* (*x#bs*) (*Eq a*) (**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*eq n s*) = *?I* (*Eq a*))
proof(*clarify*)
fix *a n s*
assume *H*: *?N a* = *?N* (*CN 0 n s*)
show *?I* (*eq n s*) = *?I* (*Eq a*) **using** *H* **by** (*auto simp add*: *eq-def ring-simps*)
qed
lemma *eq-l*: *isrlfm* (*rsplit eq a*)
by (*rule rsplit-l*[**where** *f=eq* **and** *a=a*], *auto simp add*: *eq-def*)
(*case-tac s*, *simp-all*, *case-tacnat*, *simp-all*)

lemma *neq-mono*: ∀ *a n s*. *Inum* (*x#bs*) *a* = *Inum* (*x#bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x#bs*) (*neq n s*) = *Ifm* (*x#bs*) (*NEq a*) (**is** ∀ *a n s*. *?N a* =

?N (*CN 0 n s*) ∧ - ⟶ *?I* (*neq n s*) = *?I* (*NEq a*))
proof(*clarify*)
 fix *a n s bs*
 assume *H*: *?N a* = *?N* (*CN 0 n s*)
 show *?I* (*neq n s*) = *?I* (*NEq a*) **using** *H* **by** (*auto simp add*: *neq-def ring-simps*)
qed

lemma *neq-l*: *isrlfm* (*rsplit neq a*)
 by (*rule rsplit-l*[**where** *f*=*neq* **and** *a*=*a*], *auto simp add*: *neq-def*)
(*case-tac s*, *simp-all*, *case-tacnat*, *simp-all*)

lemma *small-le*:
 assumes *u0*:*0* ≤ *u* **and** *u1*: *u* < *1*
 shows (−*u* ≤ *real* (*n*::*int*)) = (*0* ≤ *n*)
using *u0 u1* **by** *auto*

lemma *small-lt*:
 assumes *u0*:*0* ≤ *u* **and** *u1*: *u* < *1*
 shows (*real* (*n*::*int*) < *real* (*m*::*int*) − *u*) = (*n* < *m*)
using *u0 u1* **by** *auto*

lemma *rdvd01-cs*:
 assumes *up*: *u* ≥ *0* **and** *u1*: *u*<*1* **and** *np*: *real n* > *0*
 shows (*real* (*i*::*int*) *rdvd real* (*n*::*int*) ∗ *u* − *s*) = (∃ *j*∈ {*0* .. *n* − *1*}. *real n* ∗ *u* = *s* − *real* (*floor s*) + *real j* ∧ *real i rdvd real* (*j* − *floor s*)) (**is** *?lhs* = *?rhs*)
proof−
 let *?ss* = *s* − *real* (*floor s*)
 from *real-of-int-floor-add-one-gt*[**where** *r*=*s*, *simplified myless*[*rule-format*,**where** *a*=*s*]]
  real-of-int-floor-le[**where** *r*=*s*] **have** *ss0*:*?ss* ≥ *0* **and** *ss1*:*?ss* < *1*
 by (*auto simp add*: *myl*[*rule-format*, **where** *b*=*s*, *symmetric*] *myless*[*rule-format*, **where** *a*=*?ss*])
 from *np* **have** *n0*: *real n* ≥ *0* **by** *simp*
 from *mult-left-mono*[*OF up n0*] *mult-strict-left-mono*[*OF u1 np*]
 have *nu0*:*real n* ∗ *u* − *s* ≥ −*s* **and** *nun*:*real n* ∗ *u* −*s* < *real n* − *s* **by** *auto*
 from *int-rdvd-real*[**where** *i*=*i* **and** *x*=*real* (*n*::*int*) ∗ *u* − *s*]
 have *real i rdvd real n* ∗ *u* − *s* =
  (*i dvd floor* (*real n* ∗ *u* −*s*) ∧ (*real* (*floor* (*real n* ∗ *u* − *s*)) = *real n* ∗ *u* − *s*))
  (**is** - = (*?DE*) **is** - = (*?D* ∧ *?E*)) **by** *simp*
 also have . . . = (*?DE* ∧ *real*(*floor* (*real n* ∗ *u* − *s*) + *floor s*)≥ −*?ss*
  ∧ *real*(*floor* (*real n* ∗ *u* − *s*) + *floor s*)< *real n* − *?ss*) (**is** -=(*?DE* ∧*real ?a* ≥ - ∧ *real ?a* < -))
  using *nu0 nun* **by** *auto*
 also have . . . = (*?DE* ∧ *?a* ≥ *0* ∧ *?a* < *n*) **by**(*simp only*: *small-le*[*OF ss0 ss1*] *small-lt*[*OF ss0 ss1*])
 also have . . . = (*?DE* ∧ (∃ *j*∈ {*0* .. (*n* − *1*)}. *?a* = *j*)) **by** *simp*
 also have . . . = (*?DE* ∧ (∃ *j*∈ {*0* .. (*n* − *1*)}. *real* (⌊*real n* ∗ *u* − *s*⌋) = *real j* − *real* ⌊*s*⌋))
 by (*simp only*: *ring-simps real-of-int-diff* [*symmetric*] *real-of-int-inject del*: *real-of-int-diff*)

also have ... = ((∃ j∈ {*0* .. (*n* − *1*)}. *real n* ∗ *u* − *s* = *real j* − *real* ⌊*s*⌋ ∧ *real i rdvd real n* ∗ *u* − *s*)) **using** *int-rdvd-iff* [**where** *i*=*i* **and** *t*=⌊*real n* ∗ *u* − *s*⌋]
by (*auto cong*: *conj-cong*)
also have ... = *?rhs* **by**(*simp cong*: *conj-cong*) (*simp add*: *ring-simps*)
finally show *?thesis* **.**
qed

definition
DVDJ:: *int* ⇒ *int* ⇒ *num* ⇒ *fm*
where
DVDJ-def: *DVDJ i n s* = (*foldr disj* (*map* (λ *j*. *conj* (*Eq* (*CN 0 n* (*Add s* (*Sub* (*Floor* (*Neg s*)) (*C j*))))) (*Dvd i* (*Sub* (*C j*) (*Floor* (*Neg s*))))) (*iupt*(*0*,*n* − *1*))) *F*)

definition
NDVDJ:: *int* ⇒ *int* ⇒ *num* ⇒ *fm*
where
NDVDJ-def: *NDVDJ i n s* = (*foldr conj* (*map* (λ *j*. *disj* (*NEq* (*CN 0 n* (*Add s* (*Sub* (*Floor* (*Neg s*)) (*C j*))))) (*NDvd i* (*Sub* (*C j*) (*Floor* (*Neg s*))))) (*iupt*(*0*,*n* − *1*))) *T*)

lemma *DVDJ-DVD*:
assumes *xp*:*x*≥ *0* **and** *x1*: *x* < *1* **and** *np*:*real n* > *0*
shows *Ifm* (*x*#*bs*) (*DVDJ i n s*) = *Ifm* (*x*#*bs*) (*Dvd i* (*CN 0 n s*))
proof−
let *?f* = λ *j*. *conj* (*Eq*(*CN 0 n* (*Add s* (*Sub*(*Floor* (*Neg s*)) (*C j*))))) (*Dvd i* (*Sub* (*C j*) (*Floor* (*Neg s*))))
let *?s*= *Inum* (*x*#*bs*) *s*
from *foldr-disj-map*[**where** *xs*=*iupt*(*0*,*n* − *1*) **and** *bs*=*x*#*bs* **and** *f*=*?f*]
have *Ifm* (*x*#*bs*) (*DVDJ i n s*) = (∃ *j*∈ {*0* .. (*n* − *1*)}. *Ifm* (*x*#*bs*) (*?f j*))
by (*simp add*: *iupt-set np DVDJ-def del*: *iupt.simps*)
also have ... = (∃ *j*∈ {*0* .. (*n* − *1*)}. *real n* ∗ *x* = (− *?s*) − *real* (*floor* (− *?s*)) + *real j* ∧ *real i rdvd real* (*j* − *floor* (− *?s*))) **by** (*simp add*: *ring-simps diff-def* [*symmetric*])
also from *rdvd01-cs*[*OF xp x1 np*, **where** *i*=*i* **and** *s*=−*?s*]
have ... = (*real i rdvd real n* ∗ *x* − (−*?s*)) **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *NDVDJ-NDVD*:
assumes *xp*:*x*≥ *0* **and** *x1*: *x* < *1* **and** *np*:*real n* > *0*
shows *Ifm* (*x*#*bs*) (*NDVDJ i n s*) = *Ifm* (*x*#*bs*) (*NDvd i* (*CN 0 n s*))
proof−
let *?f* = λ *j*. *disj*(*NEq*(*CN 0 n* (*Add s* (*Sub* (*Floor* (*Neg s*)) (*C j*))))) (*NDvd i* (*Sub* (*C j*) (*Floor*(*Neg s*))))
let *?s*= *Inum* (*x*#*bs*) *s*
from *foldr-conj-map*[**where** *xs*=*iupt*(*0*,*n* − *1*) **and** *bs*=*x*#*bs* **and** *f*=*?f*]
have *Ifm* (*x*#*bs*) (*NDVDJ i n s*) = (∀ *j*∈ {*0* .. (*n* − *1*)}. *Ifm* (*x*#*bs*) (*?f j*))
by (*simp add*: *iupt-set np NDVDJ-def del*: *iupt.simps*)

also have ... = (¬ (∃ j∈ {*0* .. (*n* − *1*)}. *real n* ∗ *x* = (− *?s*) − *real* (*floor* (− *?s*)) + *real j* ∧ *real i rdvd real* (*j* − *floor* (− *?s*)))) **by** (*simp add*: *ring-simps diff-def*[*symmetric*])
also from *rdvd01-cs*[*OF xp x1 np*, **where** *i*=*i* **and** *s*=−*?s*]
have ... = (¬ (*real i rdvd real n* ∗ *x* − (−*?s*))) **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *foldr-disj-map-rlfm2*:
assumes *lf*: ∀ *n* . *isrlfm* (*f n*)
shows *isrlfm* (*foldr disj* (*map f xs*) *F*)
using *lf* **by** (*induct xs*, *auto*)
lemma *foldr-And-map-rlfm2*:
assumes *lf*: ∀ *n* . *isrlfm* (*f n*)
shows *isrlfm* (*foldr conj* (*map f xs*) *T*)
using *lf* **by** (*induct xs*, *auto*)

lemma *DVDJ-l*: **assumes** *ip*: *i* >*0* **and** *np*: *n*>*0* **and** *nb*: *numbound0 s*
shows *isrlfm* (*DVDJ i n s*)
proof−
let *?f*=λ*j*. *conj* (*Eq* (*CN 0 n* (*Add s* (*Sub* (*Floor* (*Neg s*)) (*C j*)))))
(*Dvd i* (*Sub* (*C j*) (*Floor* (*Neg s*))))
have *th*: ∀ *j*. *isrlfm* (*?f j*) **using** *nb np* **by** *auto*
from *DVDJ-def foldr-disj-map-rlfm2*[*OF th*] **show** *?thesis* **by** *simp*
qed

lemma *NDVDJ-l*: **assumes** *ip*: *i* >*0* **and** *np*: *n*>*0* **and** *nb*: *numbound0 s*
shows *isrlfm* (*NDVDJ i n s*)
proof−
let *?f*=λ*j*. *disj* (*NEq* (*CN 0 n* (*Add s* (*Sub* (*Floor* (*Neg s*)) (*C j*)))))
(*NDvd i* (*Sub* (*C j*) (*Floor* (*Neg s*))))
have *th*: ∀ *j*. *isrlfm* (*?f j*) **using** *nb np* **by** *auto*
from *NDVDJ-def foldr-And-map-rlfm2*[*OF th*] **show** *?thesis* **by** *auto*
qed

definition *DVD* :: *int* ⇒ *int* ⇒ *num* ⇒ *fm* **where**
DVD-def: *DVD i c t* =
(*if i*=*0 then eq c t else*
if c = *0 then* (*Dvd i t*) *else if c* >*0 then DVDJ* (*abs i*) *c t else DVDJ* (*abs i*) (−*c*) (*Neg t*))

definition *NDVD* :: *int* ⇒ *int* ⇒ *num* ⇒ *fm* **where**
NDVD i c t =
(*if i*=*0 then neq c t else*
if c = *0 then* (*NDvd i t*) *else if c* >*0 then NDVDJ* (*abs i*) *c t else NDVDJ* (*abs i*) (−*c*) (*Neg t*))

lemma *DVD-mono*:
assumes *xp*: *0*≤ *x* **and** *x1*: *x* < *1*

shows ∀ *a n s*. *Inum* (*x*#*bs*) *a* = *Inum* (*x*#*bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x*#*bs*) (*DVD i n s*) = *Ifm* (*x*#*bs*) (*Dvd i a*)
(**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*DVD i n s*) = *?I* (*Dvd i a*))
proof(*clarify*)
fix *a n s*
assume *H*: *?N a* = *?N* (*CN 0 n s*) **and** *nb*: *numbound0 s*
let *?th* = *?I* (*DVD i n s*) = *?I* (*Dvd i a*)
have *i=0* ∨ (*i≠0* ∧ *n=0*) ∨ (*i≠0* ∧ *n* < *0*) ∨ (*i≠0* ∧ *n* > *0*) **by** *arith*
moreover {**assume** *iz*: *i=0* **hence** *?th* **using** *eq-mono*[*rule-format*, *OF conjI*[*OF H nb*]]
by (*simp add*: *DVD-def rdvd-left-0-eq*)}
moreover {**assume** *inz*: *i≠0* **and** *n=0* **hence** *?th* **by** (*simp add*: *H DVD-def*)
}
moreover {**assume** *inz*: *i≠0* **and** *n<0* **hence** *?th*
by (*simp add*: *DVD-def H DVDJ-DVD*[*OF xp x1*] *rdvd-abs1*
rdvd-minus[**where** *d=i* **and** *t=real n* ∗ *x* + *Inum* (*x* # *bs*) *s*]) }
moreover {**assume** *inz*: *i≠0* **and** *n>0* **hence** *?th* **by** (*simp add*:*DVD-def H DVDJ-DVD*[*OF xp x1*] *rdvd-abs1*)}
ultimately show *?th* **by** *blast*
qed

lemma *NDVD-mono*: **assumes** *xp*: *0≤ x* **and** *x1*: *x* < *1*
shows ∀ *a n s*. *Inum* (*x*#*bs*) *a* = *Inum* (*x*#*bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x*#*bs*) (*NDVD i n s*) = *Ifm* (*x*#*bs*) (*NDvd i a*)
(**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*NDVD i n s*) = *?I* (*NDvd i a*))
proof(*clarify*)
fix *a n s*
assume *H*: *?N a* = *?N* (*CN 0 n s*) **and** *nb*: *numbound0 s*
let *?th* = *?I* (*NDVD i n s*) = *?I* (*NDvd i a*)
have *i=0* ∨ (*i≠0* ∧ *n=0*) ∨ (*i≠0* ∧ *n* < *0*) ∨ (*i≠0* ∧ *n* > *0*) **by** *arith*
moreover {**assume** *iz*: *i=0* **hence** *?th* **using** *neq-mono*[*rule-format*, *OF conjI*[*OF H nb*]]
by (*simp add*: *NDVD-def rdvd-left-0-eq*)}
moreover {**assume** *inz*: *i≠0* **and** *n=0* **hence** *?th* **by** (*simp add*: *H NDVD-def*)
}
moreover {**assume** *inz*: *i≠0* **and** *n<0* **hence** *?th*
by (*simp add*: *NDVD-def H NDVDJ-NDVD*[*OF xp x1*] *rdvd-abs1*
rdvd-minus[**where** *d=i* **and** *t=real n* ∗ *x* + *Inum* (*x* # *bs*) *s*]) }
moreover {**assume** *inz*: *i≠0* **and** *n>0* **hence** *?th*
by (*simp add*:*NDVD-def H NDVDJ-NDVD*[*OF xp x1*] *rdvd-abs1*)}
ultimately show *?th* **by** *blast*
qed

lemma *DVD-l*: *isrlfm* (*rsplit* (*DVD i*) *a*)
by (*rule rsplit-l*[**where** *f=DVD i* **and** *a=a*], *auto simp add*: *DVD-def eq-def DVDJ-l*)
(*case-tac s*, *simp-all*, *case-tac nat*, *simp-all*)

lemma *NDVD-l*: *isrlfm* (*rsplit* (*NDVD i*) *a*)

by (*rule rsplit-l*[**where** *f*=*NDVD i* **and** *a*=*a*], *auto simp add*: *NDVD-def neq-def NDVDJ-l*)
(*case-tac s*, *simp-all*, *case-tac nat*, *simp-all*)

consts *rlfm* :: *fm* ⇒ *fm*
recdef *rlfm measure fmsize*
rlfm (*And p q*) = *conj* (*rlfm p*) (*rlfm q*)
rlfm (*Or p q*) = *disj* (*rlfm p*) (*rlfm q*)
rlfm (*Imp p q*) = *disj* (*rlfm* (*NOT p*)) (*rlfm q*)
rlfm (*Iff p q*) = *disj* (*conj*(*rlfm p*) (*rlfm q*)) (*conj*(*rlfm* (*NOT p*)) (*rlfm* (*NOT q*)))
rlfm (*Lt a*) = *rsplit lt a*
rlfm (*Le a*) = *rsplit le a*
rlfm (*Gt a*) = *rsplit gt a*
rlfm (*Ge a*) = *rsplit ge a*
rlfm (*Eq a*) = *rsplit eq a*
rlfm (*NEq a*) = *rsplit neq a*
rlfm (*Dvd i a*) = *rsplit* (λ *t*. *DVD i t*) *a*
rlfm (*NDvd i a*) = *rsplit* (λ *t*. *NDVD i t*) *a*
rlfm (*NOT* (*And p q*)) = *disj* (*rlfm* (*NOT p*)) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Or p q*)) = *conj* (*rlfm* (*NOT p*)) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Imp p q*)) = *conj* (*rlfm p*) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Iff p q*)) = *disj* (*conj*(*rlfm p*) (*rlfm*(*NOT q*))) (*conj*(*rlfm*(*NOT p*)) (*rlfm q*))
rlfm (*NOT* (*NOT p*)) = *rlfm p*
rlfm (*NOT T*) = *F*
rlfm (*NOT F*) = *T*
rlfm (*NOT* (*Lt a*)) = *simpfm* (*rlfm* (*Ge a*))
rlfm (*NOT* (*Le a*)) = *simpfm* (*rlfm* (*Gt a*))
rlfm (*NOT* (*Gt a*)) = *simpfm* (*rlfm* (*Le a*))
rlfm (*NOT* (*Ge a*)) = *simpfm* (*rlfm* (*Lt a*))
rlfm (*NOT* (*Eq a*)) = *simpfm* (*rlfm* (*NEq a*))
rlfm (*NOT* (*NEq a*)) = *simpfm* (*rlfm* (*Eq a*))
rlfm (*NOT* (*Dvd i a*)) = *simpfm* (*rlfm* (*NDvd i a*))
rlfm (*NOT* (*NDvd i a*)) = *simpfm* (*rlfm* (*Dvd i a*))
rlfm p = *p* (**hints** *simp add*: *fmsize-pos*)

lemma *bound0at-l* : ⟦*isatom p* ; *bound0 p*⟧ ⟹ *isrlfm p*
by (*induct p rule*: *isrlfm.induct*, *auto*)
lemma *igcd-le1*: **assumes** *ip*: $0 < i$ **shows** *igcd i j* ≤ *i*
proof−
from *igcd-dvd1* **have** *th*: *igcd i j dvd i* **by** *blast*
from *zdvd-imp-le*[*OF th ip*] **show** *?thesis* .
qed

lemma *simpfm-rl*: *isrlfm p* ⟹ *isrlfm* (*simpfm p*)
proof (*induct p*)
case (*Lt a*)

hence *bound0* (*Lt a*) ∨ (∃ *c e*. *a* = *CN 0 c e* ∧ *c* > *0* ∧ *numbound0 e*)
  by (*cases a*,*simp-all*, *case-tac nat*, *simp-all*)
moreover
{**assume** *bound0* (*Lt a*) **hence** *bn*:*bound0* (*simpfm* (*Lt a*))
    using *simpfm-bound0* **by** *blast*
  have *isatom* (*simpfm* (*Lt a*)) **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
  with *bn bound0at-l* **have** *?case* **by** *blast*}
moreover
{**fix** *c e* **assume** *a* = *CN 0 c e* **and** *c*>*0* **and** *numbound0 e*
  {
    assume *cn1*:*numgcd* (*CN 0 c* (*simpnum e*)) ≠ *1* **and** *cnz*:*numgcd* (*CN 0 c* (*simpnum e*)) ≠ *0*
    with *numgcd-pos*[**where** *t*=*CN 0 c* (*simpnum e*)]
    have *th1*:*numgcd* (*CN 0 c* (*simpnum e*)) > *0* **by** *simp*
    from *prems* **have** *th*:*numgcd* (*CN 0 c* (*simpnum e*)) ≤ *c*
      by (*simp add*: *numgcd-def igcd-le1*)
    from *prems* **have** *th'*: *c*≠*0* **by** *auto*
    from *prems* **have** *cp*: *c* ≥ *0* **by** *simp*
    from *zdiv-mono2*[*OF cp th1 th*, *simplified zdiv-self*[*OF th'*]]
      have *0* < *c div numgcd* (*CN 0 c* (*simpnum e*)) **by** *simp*
  }
  with *prems* **have** *?case*
    by (*simp add*: *Let-def reducecoeff-def reducecoeffh-numbound0*)}
ultimately show *?case* **by** *blast*
next
case (*Le a*)
hence *bound0* (*Le a*) ∨ (∃ *c e*. *a* = *CN 0 c e* ∧ *c* > *0* ∧ *numbound0 e*)
  by (*cases a*,*simp-all*, *case-tac nat*, *simp-all*)
moreover
{**assume** *bound0* (*Le a*) **hence** *bn*:*bound0* (*simpfm* (*Le a*))
    using *simpfm-bound0* **by** *blast*
  have *isatom* (*simpfm* (*Le a*)) **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
  with *bn bound0at-l* **have** *?case* **by** *blast*}
moreover
{**fix** *c e* **assume** *a* = *CN 0 c e* **and** *c*>*0* **and** *numbound0 e*
  {
    assume *cn1*:*numgcd* (*CN 0 c* (*simpnum e*)) ≠ *1* **and** *cnz*:*numgcd* (*CN 0 c* (*simpnum e*)) ≠ *0*
    with *numgcd-pos*[**where** *t*=*CN 0 c* (*simpnum e*)]
    have *th1*:*numgcd* (*CN 0 c* (*simpnum e*)) > *0* **by** *simp*
    from *prems* **have** *th*:*numgcd* (*CN 0 c* (*simpnum e*)) ≤ *c*
      by (*simp add*: *numgcd-def igcd-le1*)
    from *prems* **have** *th'*: *c*≠*0* **by** *auto*
    from *prems* **have** *cp*: *c* ≥ *0* **by** *simp*
    from *zdiv-mono2*[*OF cp th1 th*, *simplified zdiv-self*[*OF th'*]]
      have *0* < *c div numgcd* (*CN 0 c* (*simpnum e*)) **by** *simp*
  }
  with *prems* **have** *?case*
  by (*simp add*: *Let-def reducecoeff-def simpnum-numbound0 reducecoeffh-numbound0*)}

```
  ultimately show ?case by blast
next
  case (Gt a)
  hence bound0 (Gt a) ∨ (∃ c e. a = CN 0 c e ∧ c > 0 ∧ numbound0 e)
    by (cases a,simp-all, case-tac nat, simp-all)
  moreover
  {assume bound0 (Gt a) hence bn:bound0 (simpfm (Gt a))
      using simpfm-bound0 by blast
    have isatom (simpfm (Gt a)) by (cases simpnum a, auto simp add: Let-def)
    with bn bound0at-l have ?case by blast}
  moreover
  {fix c e assume a = CN 0 c e and c>0 and numbound0 e
    {
      assume cn1:numgcd (CN 0 c (simpnum e)) ≠ 1 and cnz:numgcd (CN 0 c
(simpnum e)) ≠ 0
      with numgcd-pos[where t=CN 0 c (simpnum e)]
      have th1:numgcd (CN 0 c (simpnum e)) > 0 by simp
      from prems have th:numgcd (CN 0 c (simpnum e)) ≤ c
        by (simp add: numgcd-def igcd-le1)
      from prems have th': c≠0 by auto
      from prems have cp: c ≥ 0 by simp
      from zdiv-mono2[OF cp th1 th, simplified zdiv-self[OF th']]
        have 0 < c div numgcd (CN 0 c (simpnum e)) by simp
    }
    with prems have ?case
    by (simp add: Let-def reducecoeff-def simpnum-numbound0 reducecoeffh-numbound0)}
  ultimately show ?case by blast
next
  case (Ge a)
  hence bound0 (Ge a) ∨ (∃ c e. a = CN 0 c e ∧ c > 0 ∧ numbound0 e)
    by (cases a,simp-all, case-tac nat, simp-all)
  moreover
  {assume bound0 (Ge a) hence bn:bound0 (simpfm (Ge a))
      using simpfm-bound0 by blast
    have isatom (simpfm (Ge a)) by (cases simpnum a, auto simp add: Let-def)
    with bn bound0at-l have ?case by blast}
  moreover
  {fix c e assume a = CN 0 c e and c>0 and numbound0 e
    {
      assume cn1:numgcd (CN 0 c (simpnum e)) ≠ 1 and cnz:numgcd (CN 0 c
(simpnum e)) ≠ 0
      with numgcd-pos[where t=CN 0 c (simpnum e)]
      have th1:numgcd (CN 0 c (simpnum e)) > 0 by simp
      from prems have th:numgcd (CN 0 c (simpnum e)) ≤ c
        by (simp add: numgcd-def igcd-le1)
      from prems have th': c≠0 by auto
      from prems have cp: c ≥ 0 by simp
      from zdiv-mono2[OF cp th1 th, simplified zdiv-self[OF th']]
        have 0 < c div numgcd (CN 0 c (simpnum e)) by simp
```

}
with *prems* **have** *?case*
by (*simp add*: *Let-def reducecoeff-def simpnum-numbound0 reducecoeffh-numbound0*)}
ultimately show *?case* **by** *blast*
next
case (*Eq a*)
hence *bound0* (*Eq a*) ∨ (∃ *c e*. *a* = *CN 0 c e* ∧ *c* > *0* ∧ *numbound0 e*)
by (*cases a*,*simp-all*, *case-tac nat*, *simp-all*)
moreover
{**assume** *bound0* (*Eq a*) **hence** *bn*:*bound0* (*simpfm* (*Eq a*))
using *simpfm-bound0* **by** *blast*
have *isatom* (*simpfm* (*Eq a*)) **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
with *bn bound0at-l* **have** *?case* **by** *blast*}
moreover
{**fix** *c e* **assume** *a* = *CN 0 c e* **and** *c*>*0* **and** *numbound0 e*
{
assume *cn1*:*numgcd* (*CN 0 c* (*simpnum e*)) ≠ *1* **and** *cnz*:*numgcd* (*CN 0 c* (*simpnum e*)) ≠ *0*
with *numgcd-pos*[**where** *t*=*CN 0 c* (*simpnum e*)]
have *th1*:*numgcd* (*CN 0 c* (*simpnum e*)) > *0* **by** *simp*
from *prems* **have** *th*:*numgcd* (*CN 0 c* (*simpnum e*)) ≤ *c*
by (*simp add*: *numgcd-def igcd-le1*)
from *prems* **have** *th'*: *c*≠*0* **by** *auto*
from *prems* **have** *cp*: *c* ≥ *0* **by** *simp*
from *zdiv-mono2*[*OF cp th1 th*, *simplified zdiv-self*[*OF th'*]]
have *0* < *c div numgcd* (*CN 0 c* (*simpnum e*)) **by** *simp*
}
with *prems* **have** *?case*
by (*simp add*: *Let-def reducecoeff-def simpnum-numbound0 reducecoeffh-numbound0*)}
ultimately show *?case* **by** *blast*
next
case (*NEq a*)
hence *bound0* (*NEq a*) ∨ (∃ *c e*. *a* = *CN 0 c e* ∧ *c* > *0* ∧ *numbound0 e*)
by (*cases a*,*simp-all*, *case-tac nat*, *simp-all*)
moreover
{**assume** *bound0* (*NEq a*) **hence** *bn*:*bound0* (*simpfm* (*NEq a*))
using *simpfm-bound0* **by** *blast*
have *isatom* (*simpfm* (*NEq a*)) **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
with *bn bound0at-l* **have** *?case* **by** *blast*}
moreover
{**fix** *c e* **assume** *a* = *CN 0 c e* **and** *c*>*0* **and** *numbound0 e*
{
assume *cn1*:*numgcd* (*CN 0 c* (*simpnum e*)) ≠ *1* **and** *cnz*:*numgcd* (*CN 0 c* (*simpnum e*)) ≠ *0*
with *numgcd-pos*[**where** *t*=*CN 0 c* (*simpnum e*)]
have *th1*:*numgcd* (*CN 0 c* (*simpnum e*)) > *0* **by** *simp*
from *prems* **have** *th*:*numgcd* (*CN 0 c* (*simpnum e*)) ≤ *c*
by (*simp add*: *numgcd-def igcd-le1*)
from *prems* **have** *th'*: *c*≠*0* **by** *auto*

from *prems* **have** *cp*: $c \geq 0$ **by** *simp*
from *zdiv-mono2*[*OF cp th1 th*, *simplified zdiv-self*[*OF th'*]]
have $0 < c$ *div numgcd* (*CN 0 c* (*simpnum e*)) **by** *simp*
}
with *prems* **have** *?case*
by (*simp add*: *Let-def reducecoeff-def simpnum-numbound0 reducecoeffh-numbound0*)}
ultimately show *?case* **by** *blast*
next
case (*Dvd i a*) **hence** *bound0* (*Dvd i a*) **by** *auto* **hence** *bn*:*bound0* (*simpfm* (*Dvd i a*))
using *simpfm-bound0* **by** *blast*
have *isatom* (*simpfm* (*Dvd i a*)) **by** (*cases simpnum a*, *auto simp add*: *Let-def split-def*)
with *bn bound0at-l* **show** *?case* **by** *blast*
next
case (*NDvd i a*) **hence** *bound0* (*NDvd i a*) **by** *auto* **hence** *bn*:*bound0* (*simpfm* (*NDvd i a*))
using *simpfm-bound0* **by** *blast*
have *isatom* (*simpfm* (*NDvd i a*)) **by** (*cases simpnum a*, *auto simp add*: *Let-def split-def*)
with *bn bound0at-l* **show** *?case* **by** *blast*
qed(*auto simp add*: *conj-def imp-def disj-def iff-def Let-def simpfm-bound0 numadd-nb numneg-nb*)

lemma *rlfm-I*:
assumes *qfp*: *qfree p*
and *xp*: $0 \leq x$ **and** *x1*: $x < 1$
shows (*Ifm* (*x#bs*) (*rlfm p*) = *Ifm* (*x# bs*) *p*) ∧ *isrlfm* (*rlfm p*)
using *qfp*
by (*induct p rule*: *rlfm.induct*)
(*auto simp add*: *rsplit*[*OF xp x1 lt-mono*] *lt-l rsplit*[*OF xp x1 le-mono*] *le-l rsplit*[*OF xp x1 gt-mono*] *gt-l*
rsplit[*OF xp x1 ge-mono*] *ge-l rsplit*[*OF xp x1 eq-mono*] *eq-l rsplit*[*OF xp x1 neq-mono*] *neq-l*
rsplit[*OF xp x1 DVD-mono*[*OF xp x1*]] *DVD-l rsplit*[*OF xp x1 NDVD-mono*[*OF xp x1*]] *NDVD-l simpfm-rl*)
lemma *rlfm-l*:
assumes *qfp*: *qfree p*
shows *isrlfm* (*rlfm p*)
using *qfp lt-l gt-l ge-l le-l eq-l neq-l DVD-l NDVD-l*
by (*induct p rule*: *rlfm.induct*,*auto simp add*: *simpfm-rl*)

lemma *rminusinf-inf*:
assumes *lp*: *isrlfm p*
shows ∃ *z*. ∀ $x < z$. *Ifm* (*x#bs*) (*minusinf p*) = *Ifm* (*x#bs*) *p* (**is** ∃ *z*. ∀ *x*. *?P z x p*)
using *lp*
proof (*induct p rule*: *minusinf.induct*)

```
  case (1 p q) thus ?case by (auto,rule-tac x= min z za in exI) auto
next
  case (2 p q) thus ?case by (auto,rule-tac x= min z za in exI) auto
next
  case (3 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    hence real c ∗ x + ?e ≠ 0 by simp
    with xz have ?P ?z x (Eq (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp  }
  hence ∀ x < ?z. ?P ?z x (Eq (CN 0 c e)) by simp
  thus ?case by blast
next
  case (4 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    hence real c ∗ x + ?e ≠ 0 by simp
    with xz have ?P ?z x (NEq (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x < ?z. ?P ?z x (NEq (CN 0 c e)) by simp
  thus ?case by blast
next
  case (5 c e)
    from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    with xz have ?P ?z x (Lt (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a]  by simp }
  hence ∀ x < ?z. ?P ?z x (Lt (CN 0 c e)) by simp
```

```
  thus ?case by blast
next
  case (6 c e)
    from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    with xz have ?P ?z x (Le (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x < ?z. ?P ?z x (Le (CN 0 c e)) by simp
  thus ?case by blast
next
  case (7 c e)
    from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    with xz have ?P ?z x (Gt (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x < ?z. ?P ?z x (Gt (CN 0 c e)) by simp
  thus ?case by blast
next
  case (8 c e)
    from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    with xz have ?P ?z x (Ge (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x < ?z. ?P ?z x (Ge (CN 0 c e)) by simp
  thus ?case by blast
qed simp-all

lemma rplusinf-inf:
```

assumes *lp*: *isrlfm p*
shows ∃ *z*. ∀ *x* > *z*. *Ifm* (*x*#*bs*) (*plusinf p*) = *Ifm* (*x*#*bs*) *p* (**is** ∃ *z*. ∀ *x*. *?P z x p*)
using *lp*
proof (*induct p rule*: *isrlfm.induct*)
case (*1 p q*) **thus** *?case* **by** (*auto*,*rule-tac x*= *max z za* **in** *exI*) *auto*
next
case (*2 p q*) **thus** *?case* **by** (*auto*,*rule-tac x*= *max z za* **in** *exI*) *auto*
next
case (*3 c e*)
from *prems* **have** *nb*: *numbound0 e* **by** *simp*
from *prems* **have** *cp*: *real c* > *0* **by** *simp*
let *?e*=*Inum* (*a*#*bs*) *e*
let *?z* = (− *?e*) / *real c*
{**fix** *x*
assume *xz*: *x* > *?z*
with *mult-strict-right-mono* [*OF xz cp*] *cp*
have (*real c* ∗ *x* > − *?e*) **by** (*simp add*: *mult-ac*)
hence *real c* ∗ *x* + *?e* > *0* **by** *arith*
hence *real c* ∗ *x* + *?e* ≠ *0* **by** *simp*
with *xz* **have** *?P ?z x* (*Eq* (*CN 0 c e*))
using *numbound0-I*[*OF nb*, **where** *b*=*x* **and** *bs*=*bs* **and** *b′*=*a*] **by** *simp* }
hence ∀ *x* > *?z*. *?P ?z x* (*Eq* (*CN 0 c e*)) **by** *simp*
thus *?case* **by** *blast*
next
case (*4 c e*)
from *prems* **have** *nb*: *numbound0 e* **by** *simp*
from *prems* **have** *cp*: *real c* > *0* **by** *simp*
let *?e*=*Inum* (*a*#*bs*) *e*
let *?z* = (− *?e*) / *real c*
{**fix** *x*
assume *xz*: *x* > *?z*
with *mult-strict-right-mono* [*OF xz cp*] *cp*
have (*real c* ∗ *x* > − *?e*) **by** (*simp add*: *mult-ac*)
hence *real c* ∗ *x* + *?e* > *0* **by** *arith*
hence *real c* ∗ *x* + *?e* ≠ *0* **by** *simp*
with *xz* **have** *?P ?z x* (*NEq* (*CN 0 c e*))
using *numbound0-I*[*OF nb*, **where** *b*=*x* **and** *bs*=*bs* **and** *b′*=*a*] **by** *simp* }
hence ∀ *x* > *?z*. *?P ?z x* (*NEq* (*CN 0 c e*)) **by** *simp*
thus *?case* **by** *blast*
next
case (*5 c e*)
from *prems* **have** *nb*: *numbound0 e* **by** *simp*
from *prems* **have** *cp*: *real c* > *0* **by** *simp*
let *?e*=*Inum* (*a*#*bs*) *e*
let *?z* = (− *?e*) / *real c*
{**fix** *x*
assume *xz*: *x* > *?z*
with *mult-strict-right-mono* [*OF xz cp*] *cp*

```
    have (real c * x > - ?e) by (simp add: mult-ac)
    hence real c * x + ?e > 0 by arith
    with xz have ?P ?z x (Lt (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b'=a] by simp }
  hence ∀ x > ?z. ?P ?z x (Lt (CN 0 c e)) by simp
  thus ?case by blast
next
  case (6 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (- ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c * x > - ?e) by (simp add: mult-ac)
    hence real c * x + ?e > 0 by arith
    with xz have ?P ?z x (Le (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b'=a] by simp }
  hence ∀ x > ?z. ?P ?z x (Le (CN 0 c e)) by simp
  thus ?case by blast
next
  case (7 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (- ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c * x > - ?e) by (simp add: mult-ac)
    hence real c * x + ?e > 0 by arith
    with xz have ?P ?z x (Gt (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b'=a] by simp }
  hence ∀ x > ?z. ?P ?z x (Gt (CN 0 c e)) by simp
  thus ?case by blast
next
  case (8 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (- ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c * x > - ?e) by (simp add: mult-ac)
    hence real c * x + ?e > 0 by arith
    with xz have ?P ?z x (Ge (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b'=a]   by simp }
```

hence $\forall\ x > ?z.\ ?P\ ?z\ x\ (Ge\ (CN\ 0\ c\ e))$ **by** *simp*
thus *?case* **by** *blast*
qed *simp-all*

lemma *rminusinf-bound0*:
assumes *lp*: *isrlfm p*
shows *bound0* (*minusinf p*)
using *lp*
by (*induct p rule*: *minusinf.induct*) *simp-all*

lemma *rplusinf-bound0*:
assumes *lp*: *isrlfm p*
shows *bound0* (*plusinf p*)
using *lp*
by (*induct p rule*: *plusinf.induct*) *simp-all*

lemma *rminusinf-ex*:
assumes *lp*: *isrlfm p*
and *ex*: *Ifm* (*a#bs*) (*minusinf p*)
shows $\exists\ x.$ *Ifm* (*x#bs*) *p*
proof−
from *bound0-I* [*OF rminusinf-bound0*[*OF lp*], **where** *b=a* **and** *bs =bs*] *ex*
have *th*: $\forall\ x.$ *Ifm* (*x#bs*) (*minusinf p*) **by** *auto*
from *rminusinf-inf*[*OF lp*, **where** *bs=bs*]
obtain *z* **where** *z-def*: $\forall x<z.$ *Ifm* (*x # bs*) (*minusinf p*) = *Ifm* (*x # bs*) *p* **by** *blast*
from *th* **have** *Ifm* ((*z* − *1*)*#bs*) (*minusinf p*) **by** *simp*
moreover have $z - 1 < z$ **by** *simp*
ultimately show *?thesis* **using** *z-def* **by** *auto*
qed

lemma *rplusinf-ex*:
assumes *lp*: *isrlfm p*
and *ex*: *Ifm* (*a#bs*) (*plusinf p*)
shows $\exists\ x.$ *Ifm* (*x#bs*) *p*
proof−
from *bound0-I* [*OF rplusinf-bound0*[*OF lp*], **where** *b=a* **and** *bs =bs*] *ex*
have *th*: $\forall\ x.$ *Ifm* (*x#bs*) (*plusinf p*) **by** *auto*
from *rplusinf-inf*[*OF lp*, **where** *bs=bs*]
obtain *z* **where** *z-def*: $\forall x>z.$ *Ifm* (*x # bs*) (*plusinf p*) = *Ifm* (*x # bs*) *p* **by** *blast*
from *th* **have** *Ifm* ((*z* + *1*)*#bs*) (*plusinf p*) **by** *simp*
moreover have $z + 1 > z$ **by** *simp*
ultimately show *?thesis* **using** *z-def* **by** *auto*
qed

consts
Υ:: *fm* ⇒ (*num* × *int*) *list*
υ :: *fm* ⇒ (*num* × *int*) ⇒ *fm*

recdef Υ *measure size*
  Υ (*And p q*) = (Υ *p* @ Υ *q*)
  Υ (*Or p q*) = (Υ *p* @ Υ *q*)
  Υ (*Eq* (*CN 0 c e*)) = [(*Neg e*,*c*)]
  Υ (*NEq* (*CN 0 c e*)) = [(*Neg e*,*c*)]
  Υ (*Lt* (*CN 0 c e*)) = [(*Neg e*,*c*)]
  Υ (*Le* (*CN 0 c e*)) = [(*Neg e*,*c*)]
  Υ (*Gt* (*CN 0 c e*)) = [(*Neg e*,*c*)]
  Υ (*Ge* (*CN 0 c e*)) = [(*Neg e*,*c*)]
  Υ *p* = []

recdef υ *measure size*
  υ (*And p q*) = (λ (*t*,*n*). *And* (υ *p* (*t*,*n*)) (υ *q* (*t*,*n*)))
  υ (*Or p q*) = (λ (*t*,*n*). *Or* (υ *p* (*t*,*n*)) (υ *q* (*t*,*n*)))
  υ (*Eq* (*CN 0 c e*)) = (λ (*t*,*n*). *Eq* (*Add* (*Mul c t*) (*Mul n e*)))
  υ (*NEq* (*CN 0 c e*)) = (λ (*t*,*n*). *NEq* (*Add* (*Mul c t*) (*Mul n e*)))
  υ (*Lt* (*CN 0 c e*)) = (λ (*t*,*n*). *Lt* (*Add* (*Mul c t*) (*Mul n e*)))
  υ (*Le* (*CN 0 c e*)) = (λ (*t*,*n*). *Le* (*Add* (*Mul c t*) (*Mul n e*)))
  υ (*Gt* (*CN 0 c e*)) = (λ (*t*,*n*). *Gt* (*Add* (*Mul c t*) (*Mul n e*)))
  υ (*Ge* (*CN 0 c e*)) = (λ (*t*,*n*). *Ge* (*Add* (*Mul c t*) (*Mul n e*)))
  υ *p* = (λ (*t*,*n*). *p*)

lemma υ-*I*: **assumes** *lp*: *isrlfm p*
  and *np*: *real n* > *0* **and** *nbt*: *numbound0 t*
  shows (*Ifm* (*x*#*bs*) (υ *p* (*t*,*n*)) = *Ifm* (((*Inum* (*x*#*bs*) *t*)/(*real n*))#*bs*) *p*) $\wedge$ *bound0* (υ *p* (*t*,*n*)) (**is** (*?I x* (υ *p* (*t*,*n*)) = *?I ?u p*) $\wedge$ *?B p* **is** (- = *?I* (*?t*/*?n*) *p*) $\wedge$ - **is** (- = *?I* (*?N x t* /-) *p*) $\wedge$ -)
  using *lp*
proof(*induct p rule*: υ.*induct*)
  case (*5 c e*) **from** *prems* **have** *cp*: *c* >*0* **and** *nb*: *numbound0 e* **by** *simp*+
  have *?I ?u* (*Lt* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) < *0*)
    using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
  also have . . . = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) < *0*)
    by (*simp only*: *pos-less-divide-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)

      and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
  also have . . . = (*real c* ∗*?t* + *?n*∗ (*?N x e*) < *0*)
    using *np* **by** *simp*
  finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
  case (*6 c e*) **from** *prems* **have** *cp*: *c* >*0* **and** *nb*: *numbound0 e* **by** *simp*+
  have *?I ?u* (*Le* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) ≤ *0*)
    using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
  also have . . . = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) ≤ *0*)
    by (*simp only*: *pos-le-divide-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)
      and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
  also have . . . = (*real c* ∗*?t* + *?n*∗ (*?N x e*) ≤ *0*)
    using *np* **by** *simp*
  finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)

next
  case (*7 c e*) **from** *prems* **have** *cp*: *c >0* **and** *nb*: *numbound0 e* **by** *simp+*
  have *?I ?u* (*Gt* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) > *0*)
    using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
  also have ... = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) > *0*)
    by (*simp only*: *pos-divide-less-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)

      and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
  also have ... = (*real c* ∗*?t* + *?n*∗ (*?N x e*) > *0*)
    using *np* **by** *simp*
  finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
  case (*8 c e*) **from** *prems* **have** *cp*: *c >0* **and** *nb*: *numbound0 e* **by** *simp+*
  have *?I ?u* (*Ge* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) ≥ *0*)
    using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
  also have ... = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) ≥ *0*)
    by (*simp only*: *pos-divide-le-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)
      and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
  also have ... = (*real c* ∗*?t* + *?n*∗ (*?N x e*) ≥ *0*)
    using *np* **by** *simp*
  finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
  case (*3 c e*) **from** *prems* **have** *cp*: *c >0* **and** *nb*: *numbound0 e* **by** *simp+*
  from *np* **have** *np*: *real n* ≠ *0* **by** *simp*
  have *?I ?u* (*Eq* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) = *0*)
    using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
  also have ... = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) = *0*)
    by (*simp only*: *nonzero-eq-divide-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)
      and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
  also have ... = (*real c* ∗*?t* + *?n*∗ (*?N x e*) = *0*)
    using *np* **by** *simp*
  finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
  case (*4 c e*) **from** *prems* **have** *cp*: *c >0* **and** *nb*: *numbound0 e* **by** *simp+*
  from *np* **have** *np*: *real n* ≠ *0* **by** *simp*
  have *?I ?u* (*NEq* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) ≠ *0*)
    using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
  also have ... = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) ≠ *0*)
    by (*simp only*: *nonzero-eq-divide-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)
      and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
  also have ... = (*real c* ∗*?t* + *?n*∗ (*?N x e*) ≠ *0*)
    using *np* **by** *simp*
  finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
qed(*simp-all add*: *nbt numbound0-I*[**where** *bs* =*bs* **and** *b*=(*Inum* (*x#bs*) *t*)/ *real n* **and** *b′*=*x*] *nth-pos2*)

lemma Υ-*l*:

assumes *lp*: *isrlfm p*
shows $\forall$ (*t*,*k*) $\in$ *set* (Υ *p*). *numbound0 t* $\wedge$ *k* >*0*
using *lp*
by(*induct p rule*: Υ.*induct*) *auto*

lemma *rminusinf*-Υ:
assumes *lp*: *isrlfm p*
and *nmi*: $\neg$ (*Ifm* (*a*#*bs*) (*minusinf p*)) (**is** $\neg$ (*Ifm* (*a*#*bs*) (*?M p*)))
and *ex*: *Ifm* (*x*#*bs*) *p* (**is** *?I x p*)
shows $\exists$ (*s*,*m*) $\in$ *set* (Υ *p*). *x* $\geq$ *Inum* (*a*#*bs*) *s* / *real m* (**is** $\exists$ (*s*,*m*) $\in$ *?U p*. *x* $\geq$ *?N a s* / *real m*)
proof−
have $\exists$ (*s*,*m*) $\in$ *set* (Υ *p*). *real m* $*$ *x* $\geq$ *Inum* (*a*#*bs*) *s* (**is** $\exists$ (*s*,*m*) $\in$ *?U p*. *real m* $*$*x* $\geq$ *?N a s*)
using *lp nmi ex*
by (*induct p rule*: *minusinf*.*induct*, *auto simp add*:*numbound0-I*[**where** *bs*=*bs* **and** *b*=*a* **and** *b'*=*x*] *nth-pos2*)
then obtain *s m* **where** *smU*: (*s*,*m*) $\in$ *set* (Υ *p*) **and** *mx*: *real m* $*$ *x* $\geq$ *?N a s* **by** *blast*
from Υ-*l*[*OF lp*] *smU* **have** *mp*: *real m* > *0* **by** *auto*
from *pos-divide-le-eq*[*OF mp*, **where** *a*=*x* **and** *b*=*?N a s*, *symmetric*] *mx* **have** *x* $\geq$ *?N a s* / *real m*
by (*auto simp add*: *mult-commute*)
thus *?thesis* **using** *smU* **by** *auto*
qed

lemma *rplusinf*-Υ:
assumes *lp*: *isrlfm p*
and *nmi*: $\neg$ (*Ifm* (*a*#*bs*) (*plusinf p*)) (**is** $\neg$ (*Ifm* (*a*#*bs*) (*?M p*)))
and *ex*: *Ifm* (*x*#*bs*) *p* (**is** *?I x p*)
shows $\exists$ (*s*,*m*) $\in$ *set* (Υ *p*). *x* $\leq$ *Inum* (*a*#*bs*) *s* / *real m* (**is** $\exists$ (*s*,*m*) $\in$ *?U p*. *x* $\leq$ *?N a s* / *real m*)
proof−
have $\exists$ (*s*,*m*) $\in$ *set* (Υ *p*). *real m* $*$ *x* $\leq$ *Inum* (*a*#*bs*) *s* (**is** $\exists$ (*s*,*m*) $\in$ *?U p*. *real m* $*$*x* $\leq$ *?N a s*)
using *lp nmi ex*
by (*induct p rule*: *minusinf*.*induct*, *auto simp add*:*numbound0-I*[**where** *bs*=*bs* **and** *b*=*a* **and** *b'*=*x*] *nth-pos2*)
then obtain *s m* **where** *smU*: (*s*,*m*) $\in$ *set* (Υ *p*) **and** *mx*: *real m* $*$ *x* $\leq$ *?N a s* **by** *blast*
from Υ-*l*[*OF lp*] *smU* **have** *mp*: *real m* > *0* **by** *auto*
from *pos-le-divide-eq*[*OF mp*, **where** *a*=*x* **and** *b*=*?N a s*, *symmetric*] *mx* **have** *x* $\leq$ *?N a s* / *real m*
by (*auto simp add*: *mult-commute*)
thus *?thesis* **using** *smU* **by** *auto*
qed

lemma *lin-dense*:
assumes *lp*: *isrlfm p*

and *noS*: $\forall$ *t*. *l* < *t* $\wedge$ *t*< *u* $\longrightarrow$ *t* $\notin$ (λ (*t*,*n*). *Inum* (*x*#*bs*) *t* / *real n*) ‘ *set* (Υ *p*)
(**is** $\forall$ *t*. - $\wedge$ - $\longrightarrow$ *t* $\notin$ (λ (*t*,*n*). *?N x t* / *real n*) ‘ (*?U p*))
and *lx*: *l* < *x* **and** *xu*:*x* < *u* **and** *px*: *Ifm* (*x*#*bs*) *p*
and *ly*: *l* < *y* **and** *yu*: *y* < *u*
shows *Ifm* (*y*#*bs*) *p*
using *lp px noS*
proof (*induct p rule*: *isrlfm.induct*)
case (*5 c e*) **hence** *cp*: *real c* > *0* **and** *nb*: *numbound0 e* **by** *simp*+
from *prems* **have** *x* ∗ *real c* + *?N x e* < *0* **by** (*simp add*: *ring-simps*)
hence *pxc*: *x* < (− *?N x e*) / *real c*
by (*simp only*: *pos-less-divide-eq*[*OF cp*, **where** *a*=*x* **and** *b*=−*?N x e*])
from *prems* **have** *noSc*:$\forall$ *t*. *l* < *t* $\wedge$ *t* < *u* $\longrightarrow$ *t* $\neq$ (− *?N x e*) / *real c* **by** *auto*
with *ly yu* **have** *yne*: *y* $\neq$ − *?N x e* / *real c* **by** *auto*
hence *y* < (− *?N x e*) / *real c* $\vee$ *y* > (−*?N x e*) / *real c* **by** *auto*
moreover {**assume** *y*: *y* < (−*?N x e*)/ *real c*
hence *y* ∗ *real c* < − *?N x e*
by (*simp add*: *pos-less-divide-eq*[*OF cp*, **where** *a*=*y* **and** *b*=−*?N x e*, *symmetric*])
hence *real c* ∗ *y* + *?N x e* < *0* **by** (*simp add*: *ring-simps*)
hence *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*y*] **by** *simp*}
moreover {**assume** *y*: *y* > (− *?N x e*) / *real c*
with *yu* **have** *eu*: *u* > (− *?N x e*) / *real c* **by** *auto*
with *noSc ly yu* **have** (− *?N x e*) / *real c* ≤ *l* **by** (*cases* (− *?N x e*) / *real c* > *l*, *auto*)
with *lx pxc* **have** *False* **by** *auto*
hence *?case* **by** *simp* }
ultimately show *?case* **by** *blast*
next
case (*6 c e*) **hence** *cp*: *real c* > *0* **and** *nb*: *numbound0 e* **by** *simp* +
from *prems* **have** *x* ∗ *real c* + *?N x e* ≤ *0* **by** (*simp add*: *ring-simps*)
hence *pxc*: *x* ≤ (− *?N x e*) / *real c*
by (*simp only*: *pos-le-divide-eq*[*OF cp*, **where** *a*=*x* **and** *b*=−*?N x e*])
from *prems* **have** *noSc*:$\forall$ *t*. *l* < *t* $\wedge$ *t* < *u* $\longrightarrow$ *t* $\neq$ (− *?N x e*) / *real c* **by** *auto*
with *ly yu* **have** *yne*: *y* $\neq$ − *?N x e* / *real c* **by** *auto*
hence *y* < (− *?N x e*) / *real c* $\vee$ *y* > (−*?N x e*) / *real c* **by** *auto*
moreover {**assume** *y*: *y* < (−*?N x e*)/ *real c*
hence *y* ∗ *real c* < − *?N x e*
by (*simp add*: *pos-less-divide-eq*[*OF cp*, **where** *a*=*y* **and** *b*=−*?N x e*, *symmetric*])
hence *real c* ∗ *y* + *?N x e* < *0* **by** (*simp add*: *ring-simps*)
hence *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*y*] **by** *simp*}
moreover {**assume** *y*: *y* > (− *?N x e*) / *real c*
with *yu* **have** *eu*: *u* > (− *?N x e*) / *real c* **by** *auto*
with *noSc ly yu* **have** (− *?N x e*) / *real c* ≤ *l* **by** (*cases* (− *?N x e*) / *real c*

> *l*, *auto*)
with *lx pxc* **have** *False* **by** *auto*
hence *?case* **by** *simp* **}**
ultimately show *?case* **by** *blast*
next
case (*7 c e*) **hence** *cp*: *real c* > *0* **and** *nb*: *numbound0 e* **by** *simp*+
from *prems* **have** $x * $ *real c* + *?N x e* > *0* **by** (*simp add*: *ring-simps*)
hence *pxc*: *x* > (− *?N x e*) / *real c*
by (*simp only*: *pos-divide-less-eq*[*OF cp*, **where** *a*=*x* **and** *b*=− *?N x e*])
from *prems* **have** *noSc*:∀ *t*. *l* < *t* ∧ *t* < *u* ⟶ *t* ≠ (− *?N x e*) / *real c* **by** *auto*
with *ly yu* **have** *yne*: *y* ≠ − *?N x e* / *real c* **by** *auto*
hence *y* < (− *?N x e*) / *real c* ∨ *y* > (−*?N x e*) / *real c* **by** *auto*
moreover **{assume** *y*: *y* > (−*?N x e*)/ *real c*
hence *y* ∗ *real c* > − *?N x e*
by (*simp add*: *pos-divide-less-eq*[*OF cp*, **where** *a*=*y* **and** *b*=−*?N x e*, *symmetric*])
hence *real c* ∗ *y* + *?N x e* > *0* **by** (*simp add*: *ring-simps*)
hence *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*y*] **by** *simp***}**
moreover **{assume** *y*: *y* < (− *?N x e*) / *real c*
with *ly* **have** *eu*: *l* < (− *?N x e*) / *real c* **by** *auto*
with *noSc ly yu* **have** (− *?N x e*) / *real c* ≥ *u* **by** (*cases* (− *?N x e*) / *real c* > *l*, *auto*)
with *xu pxc* **have** *False* **by** *auto*
hence *?case* **by** *simp* **}**
ultimately show *?case* **by** *blast*
next
case (*8 c e*) **hence** *cp*: *real c* > *0* **and** *nb*: *numbound0 e* **by** *simp*+
from *prems* **have** *x* ∗ *real c* + *?N x e* ≥ *0* **by** (*simp add*: *ring-simps*)
hence *pxc*: *x* ≥ (− *?N x e*) / *real c*
by (*simp only*: *pos-divide-le-eq*[*OF cp*, **where** *a*=*x* **and** *b*=−*?N x e*])
from *prems* **have** *noSc*:∀ *t*. *l* < *t* ∧ *t* < *u* ⟶ *t* ≠ (− *?N x e*) / *real c* **by** *auto*
with *ly yu* **have** *yne*: *y* ≠ − *?N x e* / *real c* **by** *auto*
hence *y* < (− *?N x e*) / *real c* ∨ *y* > (−*?N x e*) / *real c* **by** *auto*
moreover **{assume** *y*: *y* > (−*?N x e*)/ *real c*
hence *y* ∗ *real c* > − *?N x e*
by (*simp add*: *pos-divide-less-eq*[*OF cp*, **where** *a*=*y* **and** *b*=−*?N x e*, *symmetric*])
hence *real c* ∗ *y* + *?N x e* > *0* **by** (*simp add*: *ring-simps*)
hence *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*y*] **by** *simp***}**
moreover **{assume** *y*: *y* < (− *?N x e*) / *real c*
with *ly* **have** *eu*: *l* < (− *?N x e*) / *real c* **by** *auto*
with *noSc ly yu* **have** (− *?N x e*) / *real c* ≥ *u* **by** (*cases* (− *?N x e*) / *real c* > *l*, *auto*)
with *xu pxc* **have** *False* **by** *auto*
hence *?case* **by** *simp* **}**

ultimately show *?case* **by** *blast*
next
case (*3 c e*) **hence** *cp*: *real c* > *0* **and** *nb*: *numbound0 e* **by** *simp*+
from *cp* **have** *cnz*: *real c* ≠ *0* **by** *simp*
from *prems* **have** *x* ∗ *real c* + *?N x e* = *0* **by** (*simp add*: *ring-simps*)
hence *pxc*: *x* = (− *?N x e*) / *real c*
by (*simp only*: *nonzero-eq-divide-eq*[*OF cnz*, **where** *a*=*x* **and** *b*=−*?N x e*])
from *prems* **have** *noSc*:∀ *t*. *l* < *t* ∧ *t* < *u* ⟶ *t* ≠ (− *?N x e*) / *real c* **by** *auto*
with *lx xu* **have** *yne*: *x* ≠ − *?N x e* / *real c* **by** *auto*
with *pxc* **show** *?case* **by** *simp*
next
case (*4 c e*) **hence** *cp*: *real c* > *0* **and** *nb*: *numbound0 e* **by** *simp*+
from *cp* **have** *cnz*: *real c* ≠ *0* **by** *simp*
from *prems* **have** *noSc*:∀ *t*. *l* < *t* ∧ *t* < *u* ⟶ *t* ≠ (− *?N x e*) / *real c* **by** *auto*
with *ly yu* **have** *yne*: *y* ≠ − *?N x e* / *real c* **by** *auto*
hence *y*∗ *real c* ≠ −*?N x e*
by (*simp only*: *nonzero-eq-divide-eq*[*OF cnz*, **where** *a*=*y* **and** *b*=−*?N x e*]) *simp*
hence *y*∗ *real c* + *?N x e* ≠ *0* **by** (*simp add*: *ring-simps*)
thus *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*y*]
by (*simp add*: *ring-simps*)
qed (*auto simp add*: *nth-pos2 numbound0-I*[**where** *bs*=*bs* **and** *b*=*y* **and** *b'*=*x*])

lemma *finite-set-intervals*:
assumes *px*: *P* (*x*::*real*)
and *lx*: *l* ≤ *x* **and** *xu*: *x* ≤ *u*
and *linS*: *l*∈ *S* **and** *uinS*: *u* ∈ *S*
and *fS*:*finite S* **and** *lS*: ∀ *x*∈ *S*. *l* ≤ *x* **and** *Su*: ∀ *x*∈ *S*. *x* ≤ *u*
shows ∃ *a* ∈ *S*. ∃ *b* ∈ *S*. (∀ *y*. *a* < *y* ∧ *y* < *b* ⟶ *y* ∉ *S*) ∧ *a* ≤ *x* ∧ *x* ≤ *b* ∧ *P x*
proof−
let *?Mx* = {*y*. *y*∈ *S* ∧ *y* ≤ *x*}
let *?xM* = {*y*. *y*∈ *S* ∧ *x* ≤ *y*}
let *?a* = *Max ?Mx*
let *?b* = *Min ?xM*
have *MxS*: *?Mx* ⊆ *S* **by** *blast*
hence *fMx*: *finite ?Mx* **using** *fS finite-subset* **by** *auto*
from *lx linS* **have** *linMx*: *l* ∈ *?Mx* **by** *blast*
hence *Mxne*: *?Mx* ≠ {} **by** *blast*
have *xMS*: *?xM* ⊆ *S* **by** *blast*
hence *fxM*: *finite ?xM* **using** *fS finite-subset* **by** *auto*
from *xu uinS* **have** *linxM*: *u* ∈ *?xM* **by** *blast*
hence *xMne*: *?xM* ≠ {} **by** *blast*
have *ax*:*?a* ≤ *x* **using** *Mxne fMx* **by** *auto*
have *xb*:*x* ≤ *?b* **using** *xMne fxM* **by** *auto*
have *?a* ∈ *?Mx* **using** *Max-in*[*OF fMx Mxne*] **by** *simp* **hence** *ainS*: *?a* ∈ *S* **using** *MxS* **by** *blast*

have *?b* $\in$ *?xM* **using** *Min-in*[*OF fxM xMne*] **by** *simp* **hence** *binS*: *?b* $\in$ *S* **using** *xMS* **by** *blast*
have *noy*:$\forall$ *y*. *?a* $<$ *y* $\wedge$ *y* $<$ *?b* $\longrightarrow$ *y* $\notin$ *S*
proof(*clarsimp*)
fix *y*
assume *ay*: *?a* $<$ *y* **and** *yb*: *y* $<$ *?b* **and** *yS*: *y* $\in$ *S*
from *yS* **have** *y*$\in$ *?Mx* $\vee$ *y*$\in$ *?xM* **by** *auto*
moreover {**assume** *y* $\in$ *?Mx* **hence** *y* $\leq$ *?a* **using** *Mxne fMx* **by** *auto* **with** *ay* **have** *False* **by** *simp*}
moreover {**assume** *y* $\in$ *?xM* **hence** *y* $\geq$ *?b* **using** *xMne fxM* **by** *auto* **with** *yb* **have** *False* **by** *simp*}
ultimately show *False* **by** *blast*
qed
from *ainS binS noy ax xb px* **show** *?thesis* **by** *blast*
qed

lemma *finite-set-intervals2*:
assumes *px*: *P* (*x*::*real*)
and *lx*: *l* $\leq$ *x* **and** *xu*: *x* $\leq$ *u*
and *linS*: *l*$\in$ *S* **and** *uinS*: *u* $\in$ *S*
and *fS*:*finite S* **and** *lS*: $\forall$ *x*$\in$ *S*. *l* $\leq$ *x* **and** *Su*: $\forall$ *x*$\in$ *S*. *x* $\leq$ *u*
shows ($\exists$ *s*$\in$ *S*. *P s*) $\vee$ ($\exists$ *a* $\in$ *S*. $\exists$ *b* $\in$ *S*. ($\forall$ *y*. *a* $<$ *y* $\wedge$ *y* $<$ *b* $\longrightarrow$ *y* $\notin$ *S*) $\wedge$ *a* $<$ *x* $\wedge$ *x* $<$ *b* $\wedge$ *P x*)
proof−
from *finite-set-intervals*[**where** *P*=*P*, *OF px lx xu linS uinS fS lS Su*]
obtain *a* **and** *b* **where**
as: *a*$\in$ *S* **and** *bs*: *b*$\in$ *S* **and** *noS*:$\forall$*y*. *a* $<$ *y* $\wedge$ *y* $<$ *b* $\longrightarrow$ *y* $\notin$ *S* **and** *axb*: *a* $\leq$ *x* $\wedge$ *x* $\leq$ *b* $\wedge$ *P x* **by** *auto*
from *axb* **have** *x*= *a* $\vee$ *x*= *b* $\vee$ (*a* $<$ *x* $\wedge$ *x* $<$ *b*) **by** *auto*
thus *?thesis* **using** *px as bs noS* **by** *blast*
qed

lemma *rinf*-Υ:
assumes *lp*: *isrlfm p*
and *nmi*: $\neg$ (*Ifm* (*x*#*bs*) (*minusinf p*)) (**is** $\neg$ (*Ifm* (*x*#*bs*) (*?M p*)))
and *npi*: $\neg$ (*Ifm* (*x*#*bs*) (*plusinf p*)) (**is** $\neg$ (*Ifm* (*x*#*bs*) (*?P p*)))
and *ex*: $\exists$ *x*. *Ifm* (*x*#*bs*) *p* (**is** $\exists$ *x*. *?I x p*)
shows $\exists$ (*l*,*n*) $\in$ *set* (Υ *p*). $\exists$ (*s*,*m*) $\in$ *set* (Υ *p*). *?I* ((*Inum* (*x*#*bs*) *l* / *real n* + *Inum* (*x*#*bs*) *s* / *real m*) / *2*) *p*
proof−
let *?N* = λ *x t*. *Inum* (*x*#*bs*) *t*
let *?U* = *set* (Υ *p*)
from *ex* **obtain** *a* **where** *pa*: *?I a p* **by** *blast*
from *bound0-I*[*OF rminusinf-bound0*[*OF lp*], **where** *bs*=*bs* **and** *b*=*x* **and** *b*′=*a*] *nmi*
have *nmi*′: $\neg$ (*?I a* (*?M p*)) **by** *simp*
from *bound0-I*[*OF rplusinf-bound0*[*OF lp*], **where** *bs*=*bs* **and** *b*=*x* **and** *b*′=*a*] *npi*
have *npi*′: $\neg$ (*?I a* (*?P p*)) **by** *simp*

have ∃ (*l*,*n*) ∈ *set* (Υ *p*). ∃ (*s*,*m*) ∈ *set* (Υ *p*). *?I* ((*?N a l*/*real n* + *?N a s* /*real m*) / *2*) *p*
proof−
let *?M* = (λ (*t*,*c*). *?N a t* / *real c*) ‘ *?U*
have *fM*: *finite ?M* **by** *auto*
from *rminusinf*-Υ[*OF lp nmi pa*] *rplusinf*-Υ[*OF lp npi pa*]
have ∃ (*l*,*n*) ∈ *set* (Υ *p*). ∃ (*s*,*m*) ∈ *set* (Υ *p*). *a* ≤ *?N x l* / *real n* ∧ *a* ≥ *?N x s* / *real m* **by** *blast*
then obtain *t n s m* **where**
tnU: (*t*,*n*) ∈ *?U* **and** *smU*: (*s*,*m*) ∈ *?U*
and *xs1*: *a* ≤ *?N x s* / *real m* **and** *tx1*: *a* ≥ *?N x t* / *real n* **by** *blast*
from Υ-*l*[*OF lp*] *tnU smU numbound0-I*[**where** *bs*=*bs* **and** *b*=*x* **and** *b*′=*a*] *xs1 tx1* **have** *xs*: *a* ≤ *?N a s* / *real m* **and** *tx*: *a* ≥ *?N a t* / *real n* **by** *auto*
from *tnU* **have** *Mne*: *?M* ≠ {} **by** *auto*
hence *Une*: *?U* ≠ {} **by** *simp*
let *?l* = *Min ?M*
let *?u* = *Max ?M*
have *linM*: *?l* ∈ *?M* **using** *fM Mne* **by** *simp*
have *uinM*: *?u* ∈ *?M* **using** *fM Mne* **by** *simp*
have *tnM*: *?N a t* / *real n* ∈ *?M* **using** *tnU* **by** *auto*
have *smM*: *?N a s* / *real m* ∈ *?M* **using** *smU* **by** *auto*
have *lM*: ∀ *t*∈ *?M*. *?l* ≤ *t* **using** *Mne fM* **by** *auto*
have *Mu*: ∀ *t*∈ *?M*. *t* ≤ *?u* **using** *Mne fM* **by** *auto*
have *?l* ≤ *?N a t* / *real n* **using** *tnM Mne* **by** *simp* **hence** *lx*: *?l* ≤ *a* **using** *tx* **by** *simp*
have *?N a s* / *real m* ≤ *?u* **using** *smM Mne* **by** *simp* **hence** *xu*: *a* ≤ *?u* **using** *xs* **by** *simp*
from *finite-set-intervals2*[**where** *P*=λ *x*. *?I x p*,*OF pa lx xu linM uinM fM lM Mu*]
have (∃ *s*∈ *?M*. *?I s p*) ∨
(∃ *t1*∈ *?M*. ∃ *t2* ∈ *?M*. (∀ *y*. *t1* < *y* ∧ *y* < *t2* ⟶ *y* ∉ *?M*) ∧ *t1* < *a* ∧ *a* < *t2* ∧ *?I a p*) **.**
moreover **{ fix** *u* **assume** *um*: *u*∈ *?M* **and** *pu*: *?I u p*
hence ∃ (*tu*,*nu*) ∈ *?U*. *u* = *?N a tu* / *real nu* **by** *auto*
then obtain *tu nu* **where** *tuU*: (*tu*,*nu*) ∈ *?U* **and** *tuu*:*u*= *?N a tu* / *real nu* **by** *blast*
have (*u* + *u*) / *2* = *u* **by** *auto* **with** *pu tuu*
have *?I* (((*?N a tu* / *real nu*) + (*?N a tu* / *real nu*)) / *2*) *p* **by** *simp*
with *tuU* **have** *?thesis* **by** *blast***}**
moreover{
assume ∃ *t1*∈ *?M*. ∃ *t2* ∈ *?M*. (∀ *y*. *t1* < *y* ∧ *y* < *t2* ⟶ *y* ∉ *?M*) ∧ *t1* < *a* ∧ *a* < *t2* ∧ *?I a p*
then obtain *t1* **and** *t2* **where** *t1M*: *t1* ∈ *?M* **and** *t2M*: *t2*∈ *?M*
and *noM*: ∀ *y*. *t1* < *y* ∧ *y* < *t2* ⟶ *y* ∉ *?M* **and** *t1x*: *t1* < *a* **and** *xt2*: *a* < *t2* **and** *px*: *?I a p*
by *blast*
from *t1M* **have** ∃ (*t1u*,*t1n*) ∈ *?U*. *t1* = *?N a t1u* / *real t1n* **by** *auto*
then obtain *t1u t1n* **where** *t1uU*: (*t1u*,*t1n*) ∈ *?U* **and** *t1u*: *t1* = *?N a t1u* / *real t1n* **by** *blast*

from t2M **have** ∃ (t2u,t2n) ∈ ?U. t2 = ?N a t2u / real t2n **by** auto
then obtain t2u t2n **where** t2uU: (t2u,t2n) ∈ ?U **and** t2u: t2 = ?N a t2u / real t2n **by** blast
from t1x xt2 **have** t1t2: t1 < t2 **by** simp
let ?u = (t1 + t2) / 2
from less-half-sum[OF t1t2] gt-half-sum[OF t1t2] **have** t1lu: t1 < ?u **and** ut2: ?u < t2 **by** auto
from lin-dense[OF lp noM t1x xt2 px t1lu ut2] **have** ?I ?u p **.**
with t1uU t2uU t1u t2u **have** ?thesis **by** blast**}**
ultimately show ?thesis **by** blast
qed
then obtain l n s m **where** lnU: (l,n) ∈ ?U **and** smU:(s,m) ∈ ?U
and pu: ?I ((?N a l / real n + ?N a s / real m) / 2) p **by** blast
from lnU smU Υ-l[OF lp] **have** nbl: numbound0 l **and** nbs: numbound0 s **by** auto
from numbound0-I[OF nbl, **where** bs=bs **and** b=a **and** b′=x]
numbound0-I[OF nbs, **where** bs=bs **and** b=a **and** b′=x] pu
have ?I ((?N x l / real n + ?N x s / real m) / 2) p **by** simp
with lnU smU
show ?thesis **by** auto
qed

theorem fr-eq:
assumes lp: isrlfm p
shows (∃ x. Ifm (x#bs) p) = ((Ifm (x#bs) (minusinf p)) ∨ (Ifm (x#bs) (plusinf p)) ∨ (∃ (t,n) ∈ set (Υ p). ∃ (s,m) ∈ set (Υ p). Ifm ((((Inum (x#bs) t)/ real n + (Inum (x#bs) s) / real m) /2)#bs) p))
(**is** (∃ x. ?I x p) = (?M ∨ ?P ∨ ?F) **is** ?E = ?D)
proof
assume px: ∃ x. ?I x p
have ?M ∨ ?P ∨ (¬ ?M ∧ ¬ ?P) **by** blast
moreover **{assume** ?M ∨ ?P **hence** ?D **by** blast**}**
moreover **{assume** nmi: ¬ ?M **and** npi: ¬ ?P
from rinf-Υ[OF lp nmi npi] **have** ?F **using** px **by** blast **hence** ?D **by** blast**}**
ultimately show ?D **by** blast
next
assume ?D
moreover **{assume** m:?M **from** rminusinf-ex[OF lp m] **have** ?E **.}**
moreover **{assume** p: ?P **from** rplusinf-ex[OF lp p] **have** ?E **. }**
moreover **{assume** f:?F **hence** ?E **by** blast**}**
ultimately show ?E **by** blast
qed

lemma fr-eqυ:
assumes lp: isrlfm p
shows (∃ x. Ifm (x#bs) p) = ((Ifm (x#bs) (minusinf p)) ∨ (Ifm (x#bs) (plusinf p)) ∨ (∃ (t,k) ∈ set (Υ p). ∃ (s,l) ∈ set (Υ p). Ifm (x#bs) (υ p (Add(Mul l t)

(*Mul k s*) , *2*∗*k*∗*l*))))
  (**is** (∃ *x*. *?I x p*) = (*?M* ∨ *?P* ∨ *?F*) **is** *?E* = *?D*)
proof
  assume *px*: ∃ *x*. *?I x p*
  have *?M* ∨ *?P* ∨ (¬ *?M* ∧ ¬ *?P*) **by** *blast*
  moreover {**assume** *?M* ∨ *?P* **hence** *?D* **by** *blast*}
  moreover {**assume** *nmi*: ¬ *?M* **and** *npi*: ¬ *?P*
    let *?f* =λ (*t*,*n*). *Inum* (*x*#*bs*) *t* / *real n*
    let *?N* = λ *t*. *Inum* (*x*#*bs*) *t*
    {**fix** *t n s m* **assume** (*t*,*n*)∈ *set* (Υ *p*) **and** (*s*,*m*) ∈ *set* (Υ *p*)
        with Υ-*l*[*OF lp*] **have** *tnb*: *numbound0 t* **and** *np*:*real n* > *0* **and** *snb*: *numbound0 s* **and** *mp*:*real m* > *0*
        by *auto*
      let *?st* = *Add* (*Mul m t*) (*Mul n s*)
      from *mult-pos-pos*[*OF np mp*] **have** *mnp*: *real* (*2*∗*n*∗*m*) > *0*
        by (*simp add*: *mult-commute*)
      from *tnb snb* **have** *st-nb*: *numbound0 ?st* **by** *simp*
      have *st*: (*?N t* / *real n* + *?N s* / *real m*)/*2* = *?N ?st* / *real* (*2*∗*n*∗*m*)
        using *mnp mp np* **by** (*simp add*: *ring-simps add-divide-distrib*)
      from υ-*I*[*OF lp mnp st-nb*, **where** *x*=*x* **and** *bs*=*bs*]
      have *?I x* (υ *p* (*?st*,*2*∗*n*∗*m*)) = *?I* ((*?N t* / *real n* + *?N s* / *real m*) /*2*) *p* **by** (*simp only*: *st*[*symmetric*])}
    with *rinf*-Υ[*OF lp nmi npi px*] **have** *?F* **by** *blast* **hence** *?D* **by** *blast*}
  ultimately show *?D* **by** *blast*
next
  assume *?D*
  moreover {**assume** *m*:*?M* **from** *rminusinf-ex*[*OF lp m*] **have** *?E* **.**}
  moreover {**assume** *p*: *?P* **from** *rplusinf-ex*[*OF lp p*] **have** *?E* **.** }
  moreover {**fix** *t k s l* **assume** (*t*,*k*) ∈ *set* (Υ *p*) **and** (*s*,*l*) ∈ *set* (Υ *p*)
    and *px*:*?I x* (υ *p* (*Add* (*Mul l t*) (*Mul k s*), *2*∗*k*∗*l*))
   with Υ-*l*[*OF lp*] **have** *tnb*: *numbound0 t* **and** *np*:*real k* > *0* **and** *snb*: *numbound0 s* **and** *mp*:*real l* > *0* **by** *auto*
    let *?st* = *Add* (*Mul l t*) (*Mul k s*)
    from *mult-pos-pos*[*OF np mp*] **have** *mnp*: *real* (*2*∗*k*∗*l*) > *0*
      by (*simp add*: *mult-commute*)
    from *tnb snb* **have** *st-nb*: *numbound0 ?st* **by** *simp*
    from υ-*I*[*OF lp mnp st-nb*, **where** *bs*=*bs*] *px* **have** *?E* **by** *auto*}
  ultimately show *?E* **by** *blast*
qed

The overall Part

lemma *real-ex-int-real01*:
  shows (∃ (*x*::*real*). *P x*) = (∃ (*i*::*int*) (*u*::*real*). *0*≤ *u* ∧ *u*< *1* ∧ *P* (*real i* + *u*))
proof(*auto*)
  fix *x*
  assume *Px*: *P x*
  let *?i* = *floor x*
  let *?u* = *x* − *real ?i*
  have *x* = *real ?i* + *?u* **by** *simp*

hence *P* (*real ?i* + *?u*) **using** *Px* **by** *simp*
moreover have *real ?i* $\leq$ *x* **using** *real-of-int-floor-le* **by** *simp* **hence** *0* $\leq$ *?u* **by** *arith*
moreover have *?u* < *1* **using** *real-of-int-floor-add-one-gt*[**where** *r*=*x*] **by** *arith*

ultimately show ($\exists$ (*i*::*int*) (*u*::*real*). *0*$\leq$ *u* $\wedge$ *u*< *1* $\wedge$ *P* (*real i* + *u*)) **by** *blast*
qed

consts *exsplitnum* :: *num* $\Rightarrow$ *num*
exsplit :: *fm* $\Rightarrow$ *fm*
recdef *exsplitnum measure size*
exsplitnum (*C c*) = (*C c*)
exsplitnum (*Bound 0*) = *Add* (*Bound 0*) (*Bound 1*)
exsplitnum (*Bound n*) = *Bound* (*n*+*1*)
exsplitnum (*Neg a*) = *Neg* (*exsplitnum a*)
exsplitnum (*Add a b*) = *Add* (*exsplitnum a*) (*exsplitnum b*)
exsplitnum (*Sub a b*) = *Sub* (*exsplitnum a*) (*exsplitnum b*)
exsplitnum (*Mul c a*) = *Mul c* (*exsplitnum a*)
exsplitnum (*Floor a*) = *Floor* (*exsplitnum a*)
exsplitnum (*CN 0 c a*) = *CN 0 c* (*Add* (*Mul c* (*Bound 1*)) (*exsplitnum a*))
exsplitnum (*CN n c a*) = *CN* (*n*+*1*) *c* (*exsplitnum a*)
exsplitnum (*CF c s t*) = *CF c* (*exsplitnum s*) (*exsplitnum t*)

recdef *exsplit measure size*
exsplit (*Lt a*) = *Lt* (*exsplitnum a*)
exsplit (*Le a*) = *Le* (*exsplitnum a*)
exsplit (*Gt a*) = *Gt* (*exsplitnum a*)
exsplit (*Ge a*) = *Ge* (*exsplitnum a*)
exsplit (*Eq a*) = *Eq* (*exsplitnum a*)
exsplit (*NEq a*) = *NEq* (*exsplitnum a*)
exsplit (*Dvd i a*) = *Dvd i* (*exsplitnum a*)
exsplit (*NDvd i a*) = *NDvd i* (*exsplitnum a*)
exsplit (*And p q*) = *And* (*exsplit p*) (*exsplit q*)
exsplit (*Or p q*) = *Or* (*exsplit p*) (*exsplit q*)
exsplit (*Imp p q*) = *Imp* (*exsplit p*) (*exsplit q*)
exsplit (*Iff p q*) = *Iff* (*exsplit p*) (*exsplit q*)
exsplit (*NOT p*) = *NOT* (*exsplit p*)
exsplit p = *p*

lemma *exsplitnum*:
Inum (*x*#*y*#*bs*) (*exsplitnum t*) = *Inum* ((*x*+*y*) #*bs*) *t*
by(*induct t rule*: *exsplitnum.induct*) (*simp-all add*: *ring-simps*)

lemma *exsplit*:
assumes *qfp*: *qfree p*
shows *Ifm* (*x*#*y*#*bs*) (*exsplit p*) = *Ifm* ((*x*+*y*)#*bs*) *p*
using *qfp exsplitnum*[**where** *x*=*x* **and** *y*=*y* **and** *bs*=*bs*]
by(*induct p rule*: *exsplit.induct*) *simp-all*

lemma *splitex*:
assumes *qf*: *qfree p*
shows (*Ifm bs* (*E p*)) = (∃ (*i*::*int*). *Ifm* (*real i*#*bs*) (*E* (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) (*exsplit p*)))) (**is** *?lhs* = *?rhs*)
proof−
have *?rhs* = (∃ (*i*::*int*). ∃ *x*. *0*≤ *x* ∧ *x* < *1* ∧ *Ifm* (*x*#(*real i*)#*bs*) (*exsplit p*))
by (*simp add*: *myless*[*rule-format*, **where** *b*=*1*] *myless*[*rule-format*, **where** *b*=*0*] *add-ac diff-def*)
also have ... = (∃ (*i*::*int*). ∃ *x*. *0*≤ *x* ∧ *x* < *1* ∧ *Ifm* ((*real i* + *x*) #*bs*) *p*)
by (*simp only*: *exsplit*[*OF qf*] *add-ac*)
also have ... = (∃ *x*. *Ifm* (*x*#*bs*) *p*)
by (*simp only*: *real-ex-int-real01*[**where** *P*=λ *x*. *Ifm* (*x*#*bs*) *p*])
finally show *?thesis* **by** *simp*
qed

constdefs *ferrack01*:: *fm* ⇒ *fm*
ferrack01 p ≡ (*let p′* = *rlfm*(*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (−*1*))))) *p*);
U = *remdups*(*map simp-num-pair*
(*map* (λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , *2*∗*n*∗*m*))
(*alluopairs* (Υ *p′*))))
in decr (*evaldjf* (υ *p′*) *U*))

lemma *fr-eq-01*:
assumes *qf*: *qfree p*
shows (∃ *x*. *Ifm* (*x*#*bs*) (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (−*1*))))) *p*)) = (∃ (*t*,*n*) ∈ *set* (Υ (*rlfm* (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*))). ∃ (*s*,*m*) ∈ *set* (Υ (*rlfm* (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*))). *Ifm* (*x*#*bs*) (υ (*rlfm* (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*)) (*Add* (*Mul m t*) (*Mul n s*), *2*∗*n*∗*m*)))
(**is** (∃ *x*. *?I x ?q*) = *?F*)
proof−
let *?rq* = *rlfm ?q*
let *?M* = *?I x* (*minusinf ?rq*)
let *?P* = *?I x* (*plusinf ?rq*)
have *MF*: *?M* = *False*
apply (*simp add*: *Let-def reducecoeff-def numgcd-def igcd-def rsplit-def ge-def lt-def conj-def disj-def*)
by (*cases rlfm p* = *And* (*Ge* (*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* −*1*))), *simp-all*)
have *PF*: *?P* = *False* **apply** (*simp add*: *Let-def reducecoeff-def numgcd-def igcd-def rsplit-def ge-def lt-def conj-def disj-def*)
by (*cases rlfm p* = *And* (*Ge* (*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* −*1*))), *simp-all*)
have (∃ *x*. *?I x ?q*) =
((*?I x* (*minusinf ?rq*)) ∨ (*?I x* (*plusinf ?rq*)) ∨ (∃ (*t*,*n*) ∈ *set* (Υ *?rq*). ∃ (*s*,*m*) ∈ *set* (Υ *?rq*). *?I x* (υ *?rq* (*Add* (*Mul m t*) (*Mul n s*), *2*∗*n*∗*m*))))
(**is** (∃ *x*. *?I x ?q*) = (*?M* ∨ *?P* ∨ *?F*) **is** *?E* = *?D*)
proof

assume ∃ *x*. *?I x ?q*
then obtain *x* **where** *qx*: *?I x ?q* **by** *blast*
hence *xp*: *0≤ x* **and** *x1*: *x< 1* **and** *px*: *?I x p*
by (*auto simp add*: *rsplit-def lt-def ge-def rlfm-I*[*OF qf*])
from *qx* **have** *?I x ?rq*
by (*simp add*: *rsplit-def lt-def ge-def rlfm-I*[*OF qf xp x1*])
hence *lqx*: *?I x ?rq* **using** *simpfm*[**where** *p=?rq* **and** *bs=x#bs*] **by** *auto*
from *qf* **have** *qfq*:*isrlfm ?rq*
by (*auto simp add*: *rsplit-def lt-def ge-def rlfm-I*[*OF qf xp x1*])
with *lqx fr-eqυ*[*OF qfq*] **show** *?M* ∨ *?P* ∨ *?F* **by** *blast*
next
assume *D*: *?D*
let *?U = set* (Υ *?rq*)
from *MF PF D* **have** *?F* **by** *auto*
then obtain *t n s m* **where** *aU*:(*t*,*n*) ∈ *?U* **and** *bU*:(*s*,*m*)∈ *?U* **and** *rqx*: *?I x* (*υ ?rq* (*Add* (*Mul m t*) (*Mul n s*), *2∗n∗m*)) **by** *blast*
from *qf* **have** *lrq*:*isrlfm ?rq***using** *rlfm-l*[*OF qf*]
by (*auto simp add*: *rsplit-def lt-def ge-def*)
from *aU bU* Υ*-l*[*OF lrq*] **have** *tnb*: *numbound0 t* **and** *np*:*real n > 0* **and** *snb*: *numbound0 s* **and** *mp*:*real m > 0* **by** (*auto simp add*: *split-def*)
let *?st = Add* (*Mul m t*) (*Mul n s*)
from *tnb snb* **have** *stnb*: *numbound0 ?st* **by** *simp*
from *mult-pos-pos*[*OF np mp*] **have** *mnp*: *real* (*2∗n∗m*) *> 0*
by (*simp add*: *mult-commute*)
from *conjunct1*[*OF υ-I*[*OF lrq mnp stnb*, **where** *bs=bs* **and** *x=x*], *symmetric*] *rqx*
have ∃ *x*. *?I x ?rq* **by** *auto*
thus *?E*
using *rlfm-I*[*OF qf*] **by** (*auto simp add*: *rsplit-def lt-def ge-def*)
qed
with *MF PF* **show** *?thesis* **by** *blast*
qed

lemma Υ*-cong-aux*:
assumes *Ul*: ∀ (*t*,*n*) ∈ *set U*. *numbound0 t* ∧ *n >0*
shows ((λ (*t*,*n*). *Inum* (*x#bs*) *t* /*real n*) ' (*set* (*map* (λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , *2∗n∗m*)) (*alluopairs U*)))) = ((λ ((*t*,*n*),(*s*,*m*)). (*Inum* (*x#bs*) *t* /*real n* + *Inum* (*x#bs*) *s* /*real m*)/*2*) ' (*set U* × *set U*))
(**is** *?lhs = ?rhs*)
proof(*auto*)
fix *t n s m*
assume ((*t*,*n*),(*s*,*m*)) ∈ *set* (*alluopairs U*)
hence *th*: ((*t*,*n*),(*s*,*m*)) ∈ (*set U* × *set U*)
using *alluopairs-set1*[**where** *xs=U*] **by** *blast*
let *?N* = λ *t*. *Inum* (*x#bs*) *t*
let *?st= Add* (*Mul m t*) (*Mul n s*)
from *Ul th* **have** *mnz*: *m* ≠ *0* **by** *auto*
from *Ul th* **have** *nnz*: *n* ≠ *0* **by** *auto*
have *st*: (*?N t* / *real n* + *?N s* / *real m*)/*2* = *?N ?st* / *real* (*2∗n∗m*)

using *mnz nnz* **by** (*simp add*: *ring-simps add-divide-distrib*)

thus (*real m* ∗ *Inum* (*x* # *bs*) *t* + *real n* ∗ *Inum* (*x* # *bs*) *s*) /
(*2* ∗ *real n* ∗ *real m*)
∈ (λ((*t*, *n*), *s*, *m*).
(*Inum* (*x* # *bs*) *t* / *real n* + *Inum* (*x* # *bs*) *s* / *real m*) / *2*) ‘
(*set U* × *set U*)**using** *mnz nnz th*
apply (*auto simp add*: *th add-divide-distrib ring-simps split-def image-def*)
by (*rule-tac x*=(*s*,*m*) **in** *bexI*,*simp-all*)
(*rule-tac x*=(*t*,*n*) **in** *bexI*,*simp-all*)
next
fix *t n s m*
assume *tnU*: (*t*,*n*) ∈ *set U* **and** *smU*:(*s*,*m*) ∈ *set U*
let *?N* = λ *t*. *Inum* (*x*#*bs*) *t*
let *?st*= *Add* (*Mul m t*) (*Mul n s*)
from *Ul smU* **have** *mnz*: *m* ≠ *0* **by** *auto*
from *Ul tnU* **have** *nnz*: *n* ≠ *0* **by** *auto*
have *st*: (*?N t* / *real n* + *?N s* / *real m*)/*2* = *?N ?st* / *real* (*2*∗*n*∗*m*)
using *mnz nnz* **by** (*simp add*: *ring-simps add-divide-distrib*)
let *?P* = λ (*t′*,*n′*) (*s′*,*m′*). (*Inum* (*x* # *bs*) *t* / *real n* + *Inum* (*x* # *bs*) *s* / *real m*)/*2* = (*Inum* (*x* # *bs*) *t′* / *real n′* + *Inum* (*x* # *bs*) *s′* / *real m′*)/*2*
have *Pc*:∀ *a b*. *?P a b* = *?P b a*
by *auto*
from *Ul alluopairs-set1* **have** *Up*:∀ ((*t*,*n*),(*s*,*m*)) ∈ *set* (*alluopairs U*). *n* ≠ *0* ∧ *m* ≠ *0* **by** *blast*
from *alluopairs-ex*[*OF Pc*, **where** *xs*=*U*] *tnU smU*
have *th′*:∃ ((*t′*,*n′*),(*s′*,*m′*)) ∈ *set* (*alluopairs U*). *?P* (*t′*,*n′*) (*s′*,*m′*)
by *blast*
then obtain *t′ n′ s′ m′* **where** *ts′-U*: ((*t′*,*n′*),(*s′*,*m′*)) ∈ *set* (*alluopairs U*)
and *Pts′*: *?P* (*t′*,*n′*) (*s′*,*m′*) **by** *blast*
from *ts′-U Up* **have** *mnz′*: *m′* ≠ *0* **and** *nnz′*: *n′*≠ *0* **by** *auto*
let *?st′* = *Add* (*Mul m′ t′*) (*Mul n′ s′*)
have *st′*: (*?N t′* / *real n′* + *?N s′* / *real m′*)/*2* = *?N ?st′* / *real* (*2*∗*n′*∗*m′*)
using *mnz′ nnz′* **by** (*simp add*: *ring-simps add-divide-distrib*)
from *Pts′* **have**
(*Inum* (*x* # *bs*) *t* / *real n* + *Inum* (*x* # *bs*) *s* / *real m*)/*2* = (*Inum* (*x* # *bs*) *t′* / *real n′* + *Inum* (*x* # *bs*) *s′* / *real m′*)/*2* **by** *simp*
also have . . . = ((λ(*t*, *n*). *Inum* (*x* # *bs*) *t* / *real n*) ((λ((*t*, *n*), *s*, *m*). (*Add* (*Mul m t*) (*Mul n s*), *2* ∗ *n* ∗ *m*)) ((*t′*,*n′*),(*s′*,*m′*)))) **by** (*simp add*: *st′*)
finally show (*Inum* (*x* # *bs*) *t* / *real n* + *Inum* (*x* # *bs*) *s* / *real m*) / *2*
∈ (λ(*t*, *n*). *Inum* (*x* # *bs*) *t* / *real n*) ‘
(λ((*t*, *n*), *s*, *m*). (*Add* (*Mul m t*) (*Mul n s*), *2* ∗ *n* ∗ *m*)) ‘
set (*alluopairs U*)
using *ts′-U* **by** *blast*
qed

lemma Υ-*cong*:
assumes *lp*: *isrlfm p*
and *UU′*: ((λ (*t*,*n*). *Inum* (*x*#*bs*) *t* /*real n*) ‘ *U′*) = ((λ ((*t*,*n*),(*s*,*m*)). (*Inum*

(x#bs) t /real n + Inum (x#bs) s /real m)/2) ‘ (U × U)) (**is** ?f ‘ U′ = ?g ‘ (U×U))
and U: ∀ (t,n) ∈ U. numbound0 t ∧ n > 0
and U′: ∀ (t,n) ∈ U′. numbound0 t ∧ n > 0
shows (∃ (t,n) ∈ U. ∃ (s,m) ∈ U. Ifm (x#bs) (υ p (Add (Mul m t) (Mul n s),2*n*m))) = (∃ (t,n) ∈ U′. Ifm (x#bs) (υ p (t,n)))
(**is** ?lhs = ?rhs)
proof
assume ?lhs
then obtain t n s m **where** tnU: (t,n) ∈ U **and** smU:(s,m) ∈ U **and**
Pst: Ifm (x#bs) (υ p (Add (Mul m t) (Mul n s),2*n*m)) **by** blast
let ?N = λ t. Inum (x#bs) t
from tnU smU U **have** tnb: numbound0 t **and** np: n > 0
and snb: numbound0 s **and** mp:m > 0 **by** auto
let ?st= Add (Mul m t) (Mul n s)
from mult-pos-pos[OF np mp] **have** mnp: real (2*n*m) > 0
by (simp add: mult-commute real-of-int-mult[symmetric] del: real-of-int-mult)
from tnb snb **have** stnb: numbound0 ?st **by** simp
have st: (?N t / real n + ?N s / real m)/2 = ?N ?st / real (2*n*m)
using mp np **by** (simp add: ring-simps add-divide-distrib)
from tnU smU UU′ **have** ?g ((t,n),(s,m)) ∈ ?f ‘ U′ **by** blast
hence ∃ (t′,n′) ∈ U′. ?g ((t,n),(s,m)) = ?f (t′,n′)
by auto (rule-tac x=(a,b) **in** bexI, auto)
then obtain t′ n′ **where** tnU′: (t′,n′) ∈ U′ **and** th: ?g ((t,n),(s,m)) = ?f (t′,n′) **by** blast
from U′ tnU′ **have** tnb′: numbound0 t′ **and** np′: real n′ > 0 **by** auto
from υ-I[OF lp mnp stnb, **where** bs=bs **and** x=x] Pst
have Pst2: Ifm (Inum (x # bs) (Add (Mul m t) (Mul n s)) / real (2 * n * m) # bs) p **by** simp
from conjunct1[OF υ-I[OF lp np′ tnb′, **where** bs=bs **and** x=x], symmetric] th[simplified split-def fst-conv snd-conv,symmetric] Pst2[simplified st[symmetric]]
have Ifm (x # bs) (υ p (t′, n′)) **by** (simp only: st)
then show ?rhs **using** tnU′ **by** auto
next
assume ?rhs
then obtain t′ n′ **where** tnU′: (t′,n′) ∈ U′ **and** Pt′: Ifm (x # bs) (υ p (t′, n′))
by blast
from tnU′ UU′ **have** ?f (t′,n′) ∈ ?g ‘ (U×U) **by** blast
hence ∃ ((t,n),(s,m)) ∈ (U×U). ?f (t′,n′) = ?g ((t,n),(s,m))
by auto (rule-tac x=(a,b) **in** bexI, auto)
then obtain t n s m **where** tnU: (t,n) ∈ U **and** smU:(s,m) ∈ U **and**
th: ?f (t′,n′) = ?g((t,n),(s,m)) **by** blast
let ?N = λ t. Inum (x#bs) t
from tnU smU U **have** tnb: numbound0 t **and** np: n > 0
and snb: numbound0 s **and** mp:m > 0 **by** auto
let ?st= Add (Mul m t) (Mul n s)
from mult-pos-pos[OF np mp] **have** mnp: real (2*n*m) > 0
by (simp add: mult-commute real-of-int-mult[symmetric] del: real-of-int-mult)

from *tnb snb* **have** *stnb*: *numbound0 ?st* **by** *simp*
have *st*: $(?N\ t\ /\ real\ n + ?N\ s\ /\ real\ m)/2 = ?N\ ?st\ /\ real\ (2*n*m)$
using *mp np* **by** (*simp add*: *ring-simps add-divide-distrib*)
from U' tnU' **have** tnb': *numbound0* t' **and** np': *real* $n' > 0$ **by** *auto*
from υ-I[*OF lp* np' tnb', **where** *bs=bs* **and** *x=x*,*simplified th*[*simplified split-def fst-conv snd-conv*] *st*] Pt'
have *Pst2*: *Ifm* (*Inum* (*x* # *bs*) (*Add* (*Mul m t*) (*Mul n s*)) / *real* (2 ∗ *n* ∗ *m*) # *bs*) *p* **by** *simp*
with υ-I[*OF lp mnp stnb*, **where** *x=x* **and** *bs=bs*] *tnU smU* **show** *?lhs* **by** *blast*
qed

lemma *ferrack01*:
assumes *qf*: *qfree p*
shows ((∃ *x*. *Ifm* (*x#bs*) (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*)) = (*Ifm bs* (*ferrack01 p*))) ∧ *qfree* (*ferrack01 p*) (**is** (*?lhs* = *?rhs*) ∧ -)
proof−
let *?I* = λ *x p*. *Ifm* (*x#bs*) *p*
let *?N* = λ *t*. *Inum* (*x#bs*) *t*
let *?q* = *rlfm* (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*)
let *?U* = ϒ *?q*
let *?Up* = *alluopairs ?U*
let *?g* = λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , *2*∗*n*∗*m*)
let *?S* = *map ?g ?Up*
let *?SS* = *map simp-num-pair ?S*
let *?Y* = *remdups ?SS*
let *?f*= (λ (*t*,*n*). *?N t* / *real n*)
let *?h* = λ ((*t*,*n*),(*s*,*m*)). (*?N t*/*real n* + *?N s*/ *real m*) /*2*
let *?F* = λ *p*. ∃ *a* ∈ *set* (ϒ *p*). ∃ *b* ∈ *set* (ϒ *p*). *?I x* (υ *p* (*?g*(*a*,*b*)))
let *?ep* = *evaldjf* (υ *?q*) *?Y*
from *rlfm-l*[*OF qf*] **have** *lq*: *isrlfm ?q*
by (*simp add*: *rsplit-def lt-def ge-def conj-def disj-def Let-def reducecoeff-def numgcd-def igcd-def*)
from *alluopairs-set1*[**where** *xs=?U*] **have** *UpU*: *set ?Up* ≤ (*set ?U* × *set ?U*) **by** *simp*
from ϒ-*l*[*OF lq*] **have** *U-l*: ∀ (*t*,*n*) ∈ *set ?U*. *numbound0 t* ∧ *n* > *0* **.**
from *U-l UpU*
have *Up-*: ∀ ((*t*,*n*),(*s*,*m*)) ∈ *set ?Up*. *numbound0 t* ∧ *n*> *0* ∧ *numbound0 s* ∧ *m* > *0* **by** *auto*
hence *Snb*: ∀ (*t*,*n*) ∈ *set ?S*. *numbound0 t* ∧ *n* > *0*
by (*auto simp add*: *mult-pos-pos*)
have *Y-l*: ∀ (*t*,*n*) ∈ *set ?Y*. *numbound0 t* ∧ *n* > *0*
proof−
{ **fix** *t n* **assume** *tnY*: (*t*,*n*) ∈ *set ?Y*
hence (*t*,*n*) ∈ *set ?SS* **by** *simp*
hence ∃ (t',n') ∈ *set ?S*. *simp-num-pair* (t',n') = (*t*,*n*)
by (*auto simp add*: *split-def*) (*rule-tac x*=((*aa*,*ba*),(*ab*,*bb*)) **in** *bexI*, *simp-all*)
then obtain t' n' **where** $tn'S$: (t',n') ∈ *set ?S* **and** *tns*: *simp-num-pair* (t',n') = (*t*,*n*) **by** *blast*
from $tn'S$ *Snb* **have** *tnb*: *numbound0* t' **and** *np*: $n' > 0$ **by** *auto*

```
      from simp-num-pair-l[OF tnb np tns]
      have numbound0 t ∧ n > 0 . }
    thus ?thesis by blast
  qed

  have YU: (?f ‘ set ?Y) = (?h ‘ (set ?U × set ?U))
  proof−
     from simp-num-pair-ci[where bs=x#bs] have
    ∀x. (?f o simp-num-pair) x = ?f x by auto
     hence th: ?f o simp-num-pair = ?f using ext by blast
   have (?f ‘ set ?Y) = ((?f o simp-num-pair) ‘ set ?S) by (simp add: image-compose)
    also have ... = (?f ‘ set ?S) by (simp add: th)
    also have ... = ((?f o ?g) ‘ set ?Up)
      by (simp only: set-map o-def image-compose[symmetric])
    also have ... = (?h ‘ (set ?U × set ?U))
         using ϒ-cong-aux[OF U-l, where x=x and bs=bs, simplified set-map
image-compose[symmetric]] by blast
    finally show ?thesis .
  qed
  have ∀ (t,n) ∈ set ?Y. bound0 (υ ?q (t,n))
  proof−
    { fix t n assume tnY: (t,n) ∈ set ?Y
      with Y-l have tnb: numbound0 t and np: real n > 0 by auto
      from υ-I[OF lq np tnb]
    have bound0 (υ ?q (t,n)) by simp}
    thus ?thesis by blast
  qed
  hence ep-nb: bound0 ?ep using evaldjf-bound0[where xs=?Y and f=υ ?q]
    by auto

  from fr-eq-01[OF qf, where bs=bs and x=x] have ?lhs = ?F ?q
    by (simp only: split-def fst-conv snd-conv)
  also have ... = (∃ (t,n) ∈ set ?Y. ?I x (υ ?q (t,n))) using ϒ-cong[OF lq YU
U-l Y-l]
    by (simp only: split-def fst-conv snd-conv)
  also have ... = (Ifm (x#bs) ?ep)
    using evaldjf-ex[where ps=?Y and bs = x#bs and f=υ ?q,symmetric]
    by (simp only: split-def pair-collapse)
  also have ... = (Ifm bs (decr ?ep)) using decr[OF ep-nb] by blast
  finally have lr: ?lhs = ?rhs by (simp only: ferrack01-def Let-def)
 from decr-qf[OF ep-nb] have qfree (ferrack01 p) by (simp only: Let-def ferrack01-def)
  with lr show ?thesis by blast
qed

lemma cp-thm′:
  assumes lp: iszlfm p (real (i::int)#bs)
  and up: dβ p 1 and dd: dδ p d and dp: d > 0
   shows (∃ (x::int). Ifm (real x#bs) p) = ((∃ j∈ {1 .. d}. Ifm (real j#bs)
(minusinf p)) ∨ (∃ j∈ {1.. d}. ∃ b∈ (Inum (real i#bs)) ‘ set (β p). Ifm ((b+real
```

$j)\#bs)\ p))$
using *cp-thm*[*OF lp up dd dp*] **by** *auto*

constdefs *unit*:: $fm \Rightarrow fm \times num\ list \times int$
unit p $\equiv$ (*let* $p' = zlfm\ p$; $l = \zeta\ p'$; $q = And\ (Dvd\ l\ (CN\ 0\ 1\ (C\ 0)))\ (a\beta\ p'\ l)$; $d = \delta\ q$;
$B = remdups\ (map\ simpnum\ (\beta\ q))$; $a = remdups\ (map\ simpnum\ (\alpha\ q))$
in if length B $\leq$ *length a then* (q,B,d) *else* $(mirror\ q,\ a,d))$

lemma *unit*: **assumes** *qf*: *qfree p*
shows $\bigwedge q\ B\ d.\ unit\ p = (q,B,d) \Longrightarrow ((\exists\ (x::int).\ Ifm\ (real\ x\#bs)\ p) = (\exists\ (x::int).\ Ifm\ (real\ x\#bs)\ q)) \wedge (Inum\ (real\ i\#bs))$ ` $set\ B = (Inum\ (real\ i\#bs))$ ` $set\ (\beta\ q) \wedge d\beta\ q\ 1 \wedge d\delta\ q\ d \wedge d > 0 \wedge iszlfm\ q\ (real\ (i::int)\#bs) \wedge (\forall\ b \in set\ B.\ numbound0\ b)$
proof−
fix *q B d*
assume *qBd*: *unit p* = (q,B,d)
let *?thes* = $((\exists\ (x::int).\ Ifm\ (real\ x\#bs)\ p) = (\exists\ (x::int).\ Ifm\ (real\ x\#bs)\ q)) \wedge$
$Inum\ (real\ i\#bs)$ ` $set\ B = Inum\ (real\ i\#bs)$ ` $set\ (\beta\ q) \wedge$
$d\beta\ q\ 1 \wedge d\delta\ q\ d \wedge 0 < d \wedge iszlfm\ q\ (real\ i\ \#\ bs) \wedge (\forall\ b \in set\ B.\ numbound0\ b)$
let $?I = \lambda\ (x::int)\ p.\ Ifm\ (real\ x\#bs)\ p$
let $?p' = zlfm\ p$
let $?l = \zeta\ ?p'$
let $?q = And\ (Dvd\ ?l\ (CN\ 0\ 1\ (C\ 0)))\ (a\beta\ ?p'\ ?l)$
let $?d = \delta\ ?q$
let $?B = set\ (\beta\ ?q)$
let $?B' = remdups\ (map\ simpnum\ (\beta\ ?q))$
let $?A = set\ (\alpha\ ?q)$
let $?A' = remdups\ (map\ simpnum\ (\alpha\ ?q))$
from *conjunct1*[*OF zlfm-I*[*OF qf*, **where** *bs=bs*]]
have pp': $\forall\ i.\ ?I\ i\ ?p' = ?I\ i\ p$ **by** *auto*
from *iszlfm-gen*[*OF conjunct2*[*OF zlfm-I*[*OF qf*, **where** *bs=bs* **and** *i=i*]]]
have lp': $\forall\ (i::int).\ iszlfm\ ?p'\ (real\ i\#bs)$ **by** *simp*
hence lp'': $iszlfm\ ?p'\ (real\ (i::int)\#bs)$ **by** *simp*
from $lp'\ \zeta$[**where** $p=?p'$ **and** $bs=bs$] **have** *lp*: $?l > 0$ **and** *dl*: $d\beta\ ?p'\ ?l$ **by** *auto*
from $a\beta$-*ex*[**where** $p=?p'$ **and** $l=?l$ **and** $bs=bs$, *OF* lp'' *dl lp*] pp'
have *pq-ex*:$(\exists\ (x::int).\ ?I\ x\ p) = (\exists\ x.\ ?I\ x\ ?q)$ **by** (*simp add*: *int-rdvd-iff*)
from lp'' *lp* $a\beta$[*OF* lp'' *dl lp*] **have** *lq*:$iszlfm\ ?q\ (real\ i\#bs)$ **and** *uq*: $d\beta\ ?q\ 1$
by (*auto simp add*: *isint-def*)
from δ[*OF lq*] **have** *dp*:$?d > 0$ **and** *dd*: $d\delta\ ?q\ ?d$ **by** *blast+*
let $?N = \lambda\ t.\ Inum\ (real\ (i::int)\#bs)\ t$
have $?N$ ` $set\ ?B' = ((?N\ o\ simpnum)$ ` $?B)$ **by** (*simp add*:*image-compose*)
also have $\ldots = ?N$ ` $?B$ **using** *simpnum-ci*[**where** *bs=real i #bs*] **by** *auto*
finally have BB': $?N$ ` $set\ ?B' = ?N$ ` $?B$ **.**
have $?N$ ` $set\ ?A' = ((?N\ o\ simpnum)$ ` $?A)$ **by** (*simp add*:*image-compose*)
also have $\ldots = ?N$ ` $?A$ **using** *simpnum-ci*[**where** *bs=real i #bs*] **by** *auto*

finally have AA': *?N ‘ set ?A'* = *?N ‘ ?A* **.**
from β-*numbound0*[*OF lq*] **have** *B-nb*:∀ *b*∈ *set ?B'*. *numbound0 b*
  by (*simp add*: *simpnum-numbound0*)
from α-*l*[*OF lq*] **have** *A-nb*: ∀ *b*∈ *set ?A'*. *numbound0 b*
  by (*simp add*: *simpnum-numbound0*)
  {**assume** *length ?B'* ≤ *length ?A'*
  hence *q*:*q*=*?q* **and** *B* = *?B'* **and** *d*:*d* = *?d*
    using *qBd* **by** (*auto simp add*: *Let-def unit-def*)
  with *BB' B-nb* **have** *b*: *?N ‘* (*set B*) = *?N ‘ set* (β *q*)
    and *bn*: ∀ *b*∈ *set B*. *numbound0 b* **by** *simp*+
with *pq-ex dp uq dd lq q d* **have** *?thes* **by** *simp*}
moreover
{**assume** ¬ (*length ?B'* ≤ *length ?A'*)
  hence *q*:*q*=*mirror ?q* **and** *B* = *?A'* **and** *d*:*d* = *?d*
    using *qBd* **by** (*auto simp add*: *Let-def unit-def*)
  with *AA' mirror*$\alpha\beta$[*OF lq*] *A-nb* **have** *b*:*?N ‘* (*set B*) = *?N ‘ set* (β *q*)
    and *bn*: ∀ *b*∈ *set B*. *numbound0 b* **by** *simp*+
  from *mirror-ex*[*OF lq*] *pq-ex q*
  have *pqm-eq*:(∃ (*x*::*int*). *?I x p*) = (∃ (*x*::*int*). *?I x q*) **by** *simp*
  from *lq uq q mirror-d*β [**where** *p*=*?q* **and** *bs*=*bs* **and** *a*=*real i*]
  have *lq'*: *iszlfm q* (*real i*#*bs*) **and** *uq*: *d*β *q 1* **by** *auto*
  from δ[*OF lq'*] *mirror-*δ[*OF lq*] *q d* **have** *dq*:*d*δ *q d* **by** *auto*
  from *pqm-eq b bn uq lq' dp dq q dp d* **have** *?thes* **by** *simp*
}
ultimately show *?thes* **by** *blast*
qed

constdefs *cooper* :: *fm* ⇒ *fm*
  cooper p ≡
  (*let* (*q*,*B*,*d*) = *unit p*; *js* = *iupt* (*1*,*d*);
      mq = *simpfm* (*minusinf q*);
      md = *evaldjf* (λ *j*. *simpfm* (*subst0* (*C j*) *mq*)) *js*
   in if md = *T then T else*
    (*let qd* = *evaldjf* (λ *t*. *simpfm* (*subst0 t q*))
                (*remdups* (*map* (λ (*b*,*j*). *simpnum* (*Add b* (*C j*)))
                    [(*b*,*j*). *b*←*B*,*j*←*js*]))
     in decr (*disj md qd*)))
lemma *cooper*: **assumes** *qf*: *qfree p*
  shows ((∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) = (*Ifm bs* (*cooper p*))) ∧ *qfree* (*cooper p*)

  (**is** (*?lhs* = *?rhs*) ∧ -)
proof−

  let *?I* = λ (*x*::*int*) *p*. *Ifm* (*real x*#*bs*) *p*
  let *?q* = *fst* (*unit p*)
  let *?B* = *fst* (*snd*(*unit p*))
  let *?d* = *snd* (*snd* (*unit p*))
  let *?js* = *iupt* (*1*,*?d*)

let *?mq* = *minusinf ?q*
let *?smq* = *simpfm ?mq*
let *?md* = *evaldjf* (λ *j*. *simpfm* (*subst0* (*C j*) *?smq*)) *?js*
let *?N* = λ *t*. *Inum* (*real* (*i*::*int*)#*bs*) *t*
let *?bjs* = [(*b*,*j*). *b*←*?B*,*j*←*?js*]
let *?sbjs* = *map* (λ (*b*,*j*). *simpnum* (*Add b* (*C j*))) *?bjs*
let *?qd* = *evaldjf* (λ *t*. *simpfm* (*subst0 t ?q*)) (*remdups ?sbjs*)
have *qbf*:*unit p* = (*?q*,*?B*,*?d*) **by** *simp*
from *unit*[*OF qf qbf*] **have** *pq-ex*: (∃(*x*::*int*). *?I x p*) = (∃ (*x*::*int*). *?I x ?q*) **and**

B:*?N* ' *set ?B* = *?N* ' *set* (β *?q*) **and**
uq:*d*β *?q 1* **and** *dd*: *d*δ *?q ?d* **and** *dp*: *?d* > *0* **and**
lq: *iszlfm ?q* (*real i*#*bs*) **and**
Bn: ∀ *b*∈ *set ?B*. *numbound0 b* **by** *auto*
from *zlin-qfree*[*OF lq*] **have** *qfq*: *qfree ?q* **.**
from *simpfm-qf*[*OF minusinf-qfree*[*OF qfq*]] **have** *qfmq*: *qfree ?smq***.**
have *jsnb*: ∀ *j* ∈ *set ?js*. *numbound0* (*C j*) **by** *simp*
hence ∀ *j*∈ *set ?js*. *bound0* (*subst0* (*C j*) *?smq*)
by (*auto simp only*: *subst0-bound0*[*OF qfmq*])
hence *th*: ∀ *j*∈ *set ?js*. *bound0* (*simpfm* (*subst0* (*C j*) *?smq*))
by (*auto simp add*: *simpfm-bound0*)
from *evaldjf-bound0*[*OF th*] **have** *mdb*: *bound0 ?md* **by** *simp*
from *Bn jsnb* **have** ∀ (*b*,*j*) ∈ *set ?bjs*. *numbound0* (*Add b* (*C j*))
by *simp*
hence ∀ (*b*,*j*) ∈ *set ?bjs*. *numbound0* (*simpnum* (*Add b* (*C j*)))
using *simpnum-numbound0* **by** *blast*
hence ∀ *t* ∈ *set ?sbjs*. *numbound0 t* **by** *simp*
hence ∀ *t* ∈ *set* (*remdups ?sbjs*). *bound0* (*subst0 t ?q*)
using *subst0-bound0*[*OF qfq*] **by** *auto*
hence *th'*: ∀ *t* ∈ *set* (*remdups ?sbjs*). *bound0* (*simpfm* (*subst0 t ?q*))
using *simpfm-bound0* **by** *blast*
from *evaldjf-bound0* [*OF th'*] **have** *qdb*: *bound0 ?qd* **by** *simp*
from *mdb qdb*
have *mdqdb*: *bound0* (*disj ?md ?qd*) **by** (*simp only*: *disj-def*, *cases ?md*=*T* ∨ *?qd*=*T*, *simp-all*)
from *trans* [*OF pq-ex cp-thm'*[*OF lq uq dd dp*]] *B*
have *?lhs* = (∃ *j*∈ {*1*.. *?d*}. *?I j ?mq* ∨ (∃ *b*∈ *?N* ' *set ?B*. *Ifm* ((*b*+ *real j*)#*bs*) *?q*)) **by** *auto*
also have ... = ((∃ *j*∈ *set ?js*. *?I j ?smq*) ∨ (∃ (*b*,*j*) ∈ (*?N* ' *set ?B* × *set ?js*). *Ifm* ((*b*+ *real j*)#*bs*) *?q*)) **apply** (*simp only*: *iupt-set simpfm*) **by** *auto*
also have ...= ((∃ *j*∈ *set ?js*. *?I j ?smq*) ∨ (∃ *t* ∈ (λ (*b*,*j*). *?N* (*Add b* (*C j*))) ' *set ?bjs*. *Ifm* (*t* #*bs*) *?q*)) **by** *simp*
also have ...= ((∃ *j*∈ *set ?js*. *?I j ?smq*) ∨ (∃ *t* ∈ (λ (*b*,*j*). *?N* (*simpnum* (*Add b* (*C j*)))) ' *set ?bjs*. *Ifm* (*t* #*bs*) *?q*)) **by** (*simp only*: *simpnum-ci*)
also have ...= ((∃ *j*∈ *set ?js*. *?I j ?smq*) ∨ (∃ *t* ∈ *set ?sbjs*. *Ifm* (*?N t* #*bs*) *?q*))
by (*auto simp add*: *split-def*)
also have ... = ((∃ *j*∈ *set ?js*. (λ *j*. *?I i* (*simpfm* (*subst0* (*C j*) *?smq*))) *j*) ∨ (∃ *t* ∈ *set* (*remdups ?sbjs*). (λ *t*. *?I i* (*simpfm* (*subst0 t ?q*))) *t*)) **by** (*simp only*:

simpfm subst0-I[*OF qfq*] *simpfm Inum.simps subst0-I*[*OF qfmq*] *set-remdups*)
also have ... = ((*?I i* (*evaldjf* (λ *j*. *simpfm* (*subst0* (*C j*) *?smq*)) *?js*)) $\vee$ (*?I i* (*evaldjf* (λ *t*. *simpfm* (*subst0 t ?q*)) (*remdups ?sbjs*)))) **by** (*simp only*: *evaldjf-ex*)
finally have *mdqd*: *?lhs* = (*?I i* (*disj ?md ?qd*)) **by** (*simp add*: *disj*)
hence *mdqd2*: *?lhs* = (*Ifm bs* (*decr* (*disj ?md ?qd*))) **using** *decr* [*OF mdqdb*] **by** *simp*
{**assume** *mdT*: *?md* = *T*
hence *cT*:*cooper p* = *T*
by (*simp only*: *cooper-def unit-def split-def Let-def if-True*) *simp*
from *mdT mdqd* **have** *lhs*:*?lhs* **by** (*auto simp add*: *disj*)
from *mdT* **have** *?rhs* **by** (*simp add*: *cooper-def unit-def split-def*)
with *lhs cT* **have** *?thesis* **by** *simp* }
moreover
{**assume** *mdT*: *?md* $\neq$ *T* **hence** *cooper p* = *decr* (*disj ?md ?qd*)
by (*simp only*: *cooper-def unit-def split-def Let-def if-False*)
with *mdqd2 decr-qf*[*OF mdqdb*] **have** *?thesis* **by** *simp* }
ultimately show *?thesis* **by** *blast*
qed

lemma *DJcooper*:
assumes *qf*: *qfree p*
shows (($\exists$ (*x*::*int*). *Ifm* (*real x#bs*) *p*) = (*Ifm bs* (*DJ cooper p*))) $\wedge$ *qfree* (*DJ cooper p*)
proof−
from *cooper* **have** *cqf*: $\forall$ *p*. *qfree p* $\longrightarrow$ *qfree* (*cooper p*) **by** *blast*
from *DJ-qf*[*OF cqf*] *qf* **have** *thqf*:*qfree* (*DJ cooper p*) **by** *blast*
have *Ifm bs* (*DJ cooper p*) = ($\exists$ *q*$\in$ *set* (*disjuncts p*). *Ifm bs* (*cooper q*))
by (*simp add*: *DJ-def evaldjf-ex*)
also have ... = ($\exists$ *q* $\in$ *set*(*disjuncts p*). $\exists$ (*x*::*int*). *Ifm* (*real x#bs*) *q*)
using *cooper disjuncts-qf*[*OF qf*] **by** *blast*
also have ... = ($\exists$ (*x*::*int*). *Ifm* (*real x#bs*) *p*) **by** (*induct p rule*: *disjuncts.induct*, *auto*)
finally show *?thesis* **using** *thqf* **by** *blast*
qed

lemma *σϱ-cong*: **assumes** *lp*: *iszlfm p* (*a#bs*) **and** *tt'*: *Inum* (*a#bs*) *t* = *Inum* (*a#bs*) *t'*
shows *Ifm* (*a#bs*) (*σϱ p* (*t*,*c*)) = *Ifm* (*a#bs*) (*σϱ p* (*t'*,*c*))
using *lp*
by (*induct p rule*: *iszlfm.induct*, *auto simp add*: *tt'*)

lemma *σ-cong*: **assumes** *lp*: *iszlfm p* (*a#bs*) **and** *tt'*: *Inum* (*a#bs*) *t* = *Inum* (*a#bs*) *t'*
shows *Ifm* (*a#bs*) (*σ p c t*) = *Ifm* (*a#bs*) (*σ p c t'*)
by (*simp add*: *σ-def tt' σϱ-cong*[*OF lp tt'*])

lemma *ϱ-cong*: **assumes** *lp*: *iszlfm p* (*a#bs*)

and *RR*: (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *R* = (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *set* (ϱ *p*)
shows (∃ (*e*,*c*) ∈ *R*. ∃ *j*∈ {*1*.. *c*∗(δ *p*)}. *Ifm* (*a*#*bs*) (σ *p c* (*Add e* (*C j*)))) = (∃ (*e*,*c*) ∈ *set* (ϱ *p*). ∃ *j*∈ {*1*.. *c*∗(δ *p*)}. *Ifm* (*a*#*bs*) (σ *p c* (*Add e* (*C j*))))
(**is** *?lhs* = *?rhs*)
proof
let *?d* = δ *p*
assume *?lhs* **then obtain** *e c j* **where** *ecR*: (*e*,*c*) ∈ *R* **and** *jD*:*j* ∈ {*1* .. *c*∗*?d*}
and *px*: *Ifm* (*a*#*bs*) (σ *p c* (*Add e* (*C j*))) (**is** *?sp c e j*) **by** *blast*
from *ecR* **have** (*Inum* (*a*#*bs*) *e*,*c*) ∈ (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *R* **by** *auto*
hence (*Inum* (*a*#*bs*) *e*,*c*) ∈ (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *set* (ϱ *p*) **using** *RR* **by** *simp*
hence ∃ (*e*′,*c*′) ∈ *set* (ϱ *p*). *Inum* (*a*#*bs*) *e* = *Inum* (*a*#*bs*) *e*′ ∧ *c* = *c*′ **by** *auto*
then obtain *e*′ *c*′ **where** *ecRo*:(*e*′,*c*′) ∈ *set* (ϱ *p*) **and** *ee*′:*Inum* (*a*#*bs*) *e* = *Inum* (*a*#*bs*) *e*′
and *cc*′:*c* = *c*′ **by** *blast*
from *ee*′ **have** *tt*′: *Inum* (*a*#*bs*) (*Add e* (*C j*)) = *Inum* (*a*#*bs*) (*Add e*′ (*C j*)) **by** *simp*

from σ-*cong*[*OF lp tt*′, **where** *c*=*c*] *px* **have** *px*′:*?sp c e*′ *j* **by** *simp*
from *ecRo jD px*′ *cc*′ **show** *?rhs* **apply** *auto*
by (*rule-tac x*=(*e*′, *c*′) **in** *bexI*,*simp-all*)
(*rule-tac x*=*j* **in** *bexI*, *simp-all add*: *cc*′[*symmetric*])
next
let *?d* = δ *p*
assume *?rhs* **then obtain** *e c j* **where** *ecR*: (*e*,*c*) ∈ *set* (ϱ *p*) **and** *jD*:*j* ∈ {*1* .. *c*∗*?d*}
and *px*: *Ifm* (*a*#*bs*) (σ *p c* (*Add e* (*C j*))) (**is** *?sp c e j*) **by** *blast*
from *ecR* **have** (*Inum* (*a*#*bs*) *e*,*c*) ∈ (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *set* (ϱ *p*) **by** *auto*
hence (*Inum* (*a*#*bs*) *e*,*c*) ∈ (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *R* **using** *RR* **by** *simp*
hence ∃ (*e*′,*c*′) ∈ *R*. *Inum* (*a*#*bs*) *e* = *Inum* (*a*#*bs*) *e*′ ∧ *c* = *c*′ **by** *auto*
then obtain *e*′ *c*′ **where** *ecRo*:(*e*′,*c*′) ∈ *R* **and** *ee*′:*Inum* (*a*#*bs*) *e* = *Inum* (*a*#*bs*) *e*′
and *cc*′:*c* = *c*′ **by** *blast*
from *ee*′ **have** *tt*′: *Inum* (*a*#*bs*) (*Add e* (*C j*)) = *Inum* (*a*#*bs*) (*Add e*′ (*C j*)) **by** *simp*
from σ-*cong*[*OF lp tt*′, **where** *c*=*c*] *px* **have** *px*′:*?sp c e*′ *j* **by** *simp*
from *ecRo jD px*′ *cc*′ **show** *?lhs* **apply** *auto*
by (*rule-tac x*=(*e*′, *c*′) **in** *bexI*,*simp-all*)
(*rule-tac x*=*j* **in** *bexI*, *simp-all add*: *cc*′[*symmetric*])
qed

lemma *rl-thm*′:
assumes *lp*: *iszlfm p* (*real* (*i*::*int*)#*bs*)
and *R*: (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *R* = (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *set* (ϱ *p*)
shows (∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) = ((∃ *j*∈ {*1* .. δ *p*}. *Ifm* (*real j*#*bs*) (*minusinf p*)) ∨ (∃ (*e*,*c*) ∈ *R*. ∃ *j*∈ {*1*.. *c*∗(δ *p*)}. *Ifm* (*a*#*bs*) (σ *p c* (*Add e* (*C*

```
j)))))
  using rl-thm[OF lp] ϱ-cong[OF iszlfm-gen[OF lp, rule-format, where y=a] R]
by simp

constdefs chooset:: fm ⇒ fm × ((num×int) list) × int
  chooset p ≡ (let q = zlfm p ; d = δ q;
           B = remdups (map (λ (t,k). (simpnum t,k)) (ϱ q)) ;
           a = remdups (map (λ (t,k). (simpnum t,k)) (αϱ q))
           in if length B ≤ length a then (q,B,d) else (mirror q, a,d))

lemma chooset: assumes qf: qfree p
  shows ⋀ q B d. chooset p = (q,B,d) ⟹ ((∃ (x::int). Ifm (real x#bs) p) =
(∃ (x::int). Ifm (real x#bs) q)) ∧ ((λ(t,k). (Inum (real i#bs) t,k)) ` set B =
(λ(t,k). (Inum (real i#bs) t,k)) ` set (ϱ q)) ∧ (δ q = d) ∧ d >0 ∧ iszlfm q (real
(i::int)#bs) ∧ (∀ (e,c)∈ set B. numbound0 e ∧ c>0)
proof−
  fix q B d
  assume qBd: chooset p = (q,B,d)
  let ?thes = ((∃ (x::int). Ifm (real x#bs) p) = (∃ (x::int). Ifm (real x#bs) q))
∧ ((λ(t,k). (Inum (real i#bs) t,k)) ` set B = (λ(t,k). (Inum (real i#bs) t,k)) `
set (ϱ q)) ∧ (δ q = d) ∧ d >0 ∧ iszlfm q (real (i::int)#bs) ∧ (∀ (e,c)∈ set B.
numbound0 e ∧ c>0)
  let ?I = λ (x::int) p. Ifm (real x#bs) p
  let ?q = zlfm p
  let ?d = δ ?q
  let ?B = set (ϱ ?q)
  let ?f = λ (t,k). (simpnum t,k)
  let ?B'= remdups (map ?f (ϱ ?q))
  let ?A = set (αϱ ?q)
  let ?A'= remdups (map ?f (αϱ ?q))
  from conjunct1[OF zlfm-I[OF qf, where bs=bs]]
  have pp': ∀ i. ?I i ?q = ?I i p by auto
  hence pq-ex:(∃ (x::int). ?I x p) = (∃ x. ?I x ?q) by simp
 from iszlfm-gen[OF conjunct2[OF zlfm-I[OF qf, where bs=bs and i=i]], rule-format,
where y=real i]
  have lq: iszlfm ?q (real (i::int)#bs) .
  from δ[OF lq] have dp:?d >0 by blast
  let ?N = λ (t,c). (Inum (real (i::int)#bs) t,c)
  have ?N ` set ?B' = ((?N o ?f) ` ?B) by (simp add: split-def image-compose)
  also have . . . = ?N ` ?B
   by(simp add: split-def image-compose simpnum-ci[where bs=real i #bs] image-def)
  finally have BB': ?N ` set ?B' = ?N ` ?B .
  have ?N ` set ?A' = ((?N o ?f) ` ?A) by (simp add: split-def image-compose)
  also have . . . = ?N ` ?A using simpnum-ci[where bs=real i #bs]
   by(simp add: split-def image-compose simpnum-ci[where bs=real i #bs] image-def)

  finally have AA': ?N ` set ?A' = ?N ` ?A .
  from ϱ-l[OF lq] have B-nb:∀ (e,c)∈ set ?B'. numbound0 e ∧ c > 0
    by (simp add: simpnum-numbound0 split-def)
```

from $\alpha\varrho$-*l*[*OF lq*] **have** *A-nb*: $\forall$ (*e*,*c*)$\in$ *set ?A'*. *numbound0 e* $\wedge$ *c* > *0*
  by (*simp add*: *simpnum-numbound0 split-def*)
  {**assume** *length ?B'* $\leq$ *length ?A'*
  hence *q*:*q*=*?q* **and** *B* = *?B'* **and** *d*:*d* = *?d*
    using *qBd* **by** (*auto simp add*: *Let-def chooset-def*)
  with *BB' B-nb* **have** *b*: *?N* ‘ (*set B*) = *?N* ‘ *set* (ϱ *q*)
    and *bn*: $\forall$(*e*,*c*)$\in$ *set B*. *numbound0 e* $\wedge$ *c* > *0* **by** *auto*
with *pq-ex dp lq q d* **have** *?thes* **by** *simp*}
moreover
{**assume** $\neg$ (*length ?B'* $\leq$ *length ?A'*)
  hence *q*:*q*=*mirror ?q* **and** *B* = *?A'* **and** *d*:*d* = *?d*
    using *qBd* **by** (*auto simp add*: *Let-def chooset-def*)
  with *AA' mirror-*$\alpha\varrho$[*OF lq*] *A-nb* **have** *b*:*?N* ‘ (*set B*) = *?N* ‘ *set* (ϱ *q*)
    and *bn*: $\forall$(*e*,*c*)$\in$ *set B*. *numbound0 e* $\wedge$ *c* > *0* **by** *auto*
  from *mirror-ex*[*OF lq*] *pq-ex q*
  have *pqm-eq*:($\exists$ (*x*::*int*). *?I x p*) = ($\exists$ (*x*::*int*). *?I x q*) **by** *simp*
  from *lq q mirror-l* [**where** *p*=*?q* **and** *bs*=*bs* **and** *a*=*real i*]
  have *lq'*: *iszlfm q* (*real i*#*bs*) **by** *auto*
  from *mirror-*δ[*OF lq*] *pqm-eq b bn lq' dp q dp d* **have** *?thes* **by** *simp*
}
ultimately show *?thes* **by** *blast*
qed

constdefs *stage*:: *fm* $\Rightarrow$ *int* $\Rightarrow$ (*num* $\times$ *int*) $\Rightarrow$ *fm*
  stage p d $\equiv$ (λ (*e*,*c*). *evaldjf* (λ *j*. *simpfm* (σ *p c* (*Add e* (*C j*)))) (*iupt* (*1*,*c*∗*d*)))
lemma *stage*:
  shows *Ifm bs* (*stage p d* (*e*,*c*)) = ($\exists$ *j*$\in${*1* .. *c*∗*d*}. *Ifm bs* (σ *p c* (*Add e* (*C j*))))
  by (*unfold stage-def split-def* ,*simp only*: *evaldjf-ex iupt-set simpfm*) *simp*

lemma *stage-nb*: **assumes** *lp*: *iszlfm p* (*a*#*bs*) **and** *cp*: *c* >*0* **and** *nb*:*numbound0 e*
  shows *bound0* (*stage p d* (*e*,*c*))
proof−
  let *?f* = λ *j*. *simpfm* (σ *p c* (*Add e* (*C j*)))
  have *th*: $\forall$ *j*$\in$ *set* (*iupt*(*1*,*c*∗*d*)). *bound0* (*?f j*)
  proof
    fix *j*
    from *nb* **have** *nb'*:*numbound0* (*Add e* (*C j*)) **by** *simp*
    from *simpfm-bound0*[*OF* σ-*nb*[*OF lp nb'*, **where** *k*=*c*]]
    show *bound0* (*simpfm* (σ *p c* (*Add e* (*C j*)))) .
  qed
  from *evaldjf-bound0*[*OF th*] **show** *?thesis* **by** (*unfold stage-def split-def*) *simp*
qed

constdefs *redlove*:: *fm* $\Rightarrow$ *fm*
  redlove p $\equiv$
  (*let* (*q*,*B*,*d*) = *chooset p*;
    mq = *simpfm* (*minusinf q*);
    md = *evaldjf* (λ *j*. *simpfm* (*subst0* (*C j*) *mq*)) (*iupt* (*1*,*d*))

in if md = T then T else
(*let qd = evaldjf (stage q d) B*
in decr (disj md qd)))

lemma *redlove*: **assumes** *qf*: *qfree p*
shows ((∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) = (*Ifm bs* (*redlove p*))) ∧ *qfree* (*redlove p*)
(**is** (*?lhs* = *?rhs*) ∧ -)
proof−

let *?I* = λ (*x*::*int*) *p*. *Ifm* (*real x*#*bs*) *p*
let *?q* = *fst* (*chooset p*)
let *?B* = *fst* (*snd*(*chooset p*))
let *?d* = *snd* (*snd* (*chooset p*))
let *?js* = *iupt* (*1*,*?d*)
let *?mq* = *minusinf ?q*
let *?smq* = *simpfm ?mq*
let *?md* = *evaldjf* (λ *j*. *simpfm* (*subst0* (*C j*) *?smq*)) *?js*
let *?N* = λ (*t*,*k*). (*Inum* (*real* (*i*::*int*)#*bs*) *t*,*k*)
let *?qd* = *evaldjf* (*stage ?q ?d*) *?B*
have *qbf*:*chooset p* = (*?q*,*?B*,*?d*) **by** *simp*
from *chooset*[*OF qf qbf*] **have** *pq-ex*: (∃(*x*::*int*). *?I x p*) = (∃ (*x*::*int*). *?I x ?q*) **and**
B:*?N* ‘ *set ?B* = *?N* ‘ *set* (ϱ *?q*) **and** *dd*: δ *?q* = *?d* **and** *dp*: *?d* > *0* **and**
lq: *iszlfm ?q* (*real i*#*bs*) **and**
Bn: ∀ (*e*,*c*)∈ *set ?B*. *numbound0 e* ∧ *c* > *0* **by** *auto*
from *zlin-qfree*[*OF lq*] **have** *qfq*: *qfree ?q* **.**
from *simpfm-qf*[*OF minusinf-qfree*[*OF qfq*]] **have** *qfmq*: *qfree ?smq***.**
have *jsnb*: ∀ *j* ∈ *set ?js*. *numbound0* (*C j*) **by** *simp*
hence ∀ *j*∈ *set ?js*. *bound0* (*subst0* (*C j*) *?smq*)
by (*auto simp only*: *subst0-bound0*[*OF qfmq*])
hence *th*: ∀ *j*∈ *set ?js*. *bound0* (*simpfm* (*subst0* (*C j*) *?smq*))
by (*auto simp add*: *simpfm-bound0*)
from *evaldjf-bound0*[*OF th*] **have** *mdb*: *bound0 ?md* **by** *simp*
from *Bn stage-nb*[*OF lq*] **have** *th*:∀ *x* ∈ *set ?B*. *bound0* (*stage ?q ?d x*) **by** *auto*
from *evaldjf-bound0*[*OF th*] **have** *qdb*: *bound0 ?qd* **.**
from *mdb qdb*
have *mdqdb*: *bound0* (*disj ?md ?qd*) **by** (*simp only*: *disj-def*, *cases ?md*=*T* ∨ *?qd*=*T*, *simp-all*)
from *trans* [*OF pq-ex rl-thm'*[*OF lq B*]] *dd*
have *?lhs* = ((∃ *j*∈ {*1*.. *?d*}. *?I j ?mq*) ∨ (∃ (*e*,*c*)∈ *set ?B*. ∃ *j*∈ {*1* .. *c*∗*?d*}. *Ifm* (*real i*#*bs*) (σ *?q c* (*Add e* (*C j*))))) **by** *auto*
also have ... = ((∃ *j*∈ {*1*.. *?d*}. *?I j ?smq*) ∨ (∃ (*e*,*c*)∈ *set ?B*. *?I i* (*stage ?q ?d* (*e*,*c*))))
by (*simp add*: *simpfm stage split-def*)
also have ... = ((∃ *j*∈ {*1* .. *?d*}. *?I i* (*subst0* (*C j*) *?smq*)) ∨ *?I i ?qd*)
by (*simp add*: *evaldjf-ex subst0-I*[*OF qfmq*])
finally have *mdqd*:*?lhs* = (*?I i ?md* ∨ *?I i ?qd*) **by** (*simp only*: *evaldjf-ex iupt-set simpfm*)

also have ... = (*?I i* (*disj ?md ?qd*)) **by** (*simp add*: *disj*)
also have ... = (*Ifm bs* (*decr* (*disj ?md ?qd*))) **by** (*simp only*: *decr* [*OF mdqdb*])

finally have *mdqd2*: *?lhs* = (*Ifm bs* (*decr* (*disj ?md ?qd*))) **.**
{assume *mdT*: *?md* = *T*
hence *cT*:*redlove p* = *T* **by** (*simp add*: *redlove-def Let-def chooset-def split-def*)
from *mdT* **have** *lhs*:*?lhs* **using** *mdqd* **by** *simp*
from *mdT* **have** *?rhs* **by** (*simp add*: *redlove-def chooset-def split-def*)
with *lhs cT* **have** *?thesis* **by** *simp* **}**
moreover
{assume *mdT*: *?md* ≠ *T* **hence** *redlove p* = *decr* (*disj ?md ?qd*)
by (*simp add*: *redlove-def chooset-def split-def Let-def*)
with *mdqd2 decr-qf*[*OF mdqdb*] **have** *?thesis* **by** *simp* **}**
ultimately show *?thesis* **by** *blast*
qed

lemma *DJredlove*:
assumes *qf*: *qfree p*
shows ((∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) = (*Ifm bs* (*DJ redlove p*))) ∧ *qfree* (*DJ redlove p*)
proof−
from *redlove* **have** *cqf*: ∀ *p*. *qfree p* ⟶ *qfree* (*redlove p*) **by** *blast*
from *DJ-qf*[*OF cqf*] *qf* **have** *thqf*:*qfree* (*DJ redlove p*) **by** *blast*
have *Ifm bs* (*DJ redlove p*) = (∃ *q*∈ *set* (*disjuncts p*). *Ifm bs* (*redlove q*))
by (*simp add*: *DJ-def evaldjf-ex*)
also have ... = (∃ *q* ∈ *set*(*disjuncts p*). ∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *q*)
using *redlove disjuncts-qf*[*OF qf*] **by** *blast*
also have ... = (∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) **by** (*induct p rule*: *disjuncts.induct*, *auto*)
finally show *?thesis* **using** *thqf* **by** *blast*
qed

lemma *exsplit-qf*: **assumes** *qf*: *qfree p*
shows *qfree* (*exsplit p*)
using *qf* **by** (*induct p rule*: *exsplit.induct*, *auto*)

constdefs *mircfr* :: *fm* ⇒ *fm*
mircfr ≡ (*DJ cooper*) *o ferrack01 o simpfm o exsplit*

constdefs *mirlfr* :: *fm* ⇒ *fm*
mirlfr ≡ (*DJ redlove*) *o ferrack01 o simpfm o exsplit*

lemma *mircfr*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*mircfr p*) ∧ *Ifm bs* (*mircfr p*) = *Ifm bs* (*E p*)
proof(*clarsimp simp del*: *Ifm.simps*)
fix *bs p*
assume *qf*: *qfree p*

show *qfree* (*mircfr p*)∧(*Ifm bs* (*mircfr p*) = *Ifm bs* (*E p*)) (**is** - ∧ (*?lhs* = *?rhs*))
proof−
let *?es* = (*And* (*And* (*Ge* (*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) (*simpfm* (*exsplit p*)))
have *?rhs* = (∃ (*i*::*int*). ∃ *x*. *Ifm* (*x*#*real i*#*bs*) *?es*)
using *splitex*[*OF qf*] **by** *simp*
with *ferrack01*[*OF simpfm-qf*[*OF exsplit-qf*[*OF qf*]]] **have** *th1*: *?rhs* = (∃ (*i*::*int*). *Ifm* (*real i*#*bs*) (*ferrack01* (*simpfm* (*exsplit p*)))) **and** *qf'*:*qfree* (*ferrack01* (*simpfm* (*exsplit p*))) **by** *simp+*
with *DJcooper*[*OF qf'*] **show** *?thesis* **by** (*simp add*: *mircfr-def*)
qed
qed

lemma *mirlfr*: ∀ *bs p*. *qfree p* ⟶ *qfree*(*mirlfr p*) ∧ *Ifm bs* (*mirlfr p*) = *Ifm bs* (*E p*)
proof(*clarsimp simp del*: *Ifm.simps*)
fix *bs p*
assume *qf*: *qfree p*
show *qfree* (*mirlfr p*)∧(*Ifm bs* (*mirlfr p*) = *Ifm bs* (*E p*)) (**is** - ∧ (*?lhs* = *?rhs*))
proof−
let *?es* = (*And* (*And* (*Ge* (*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) (*simpfm* (*exsplit p*)))
have *?rhs* = (∃ (*i*::*int*). ∃ *x*. *Ifm* (*x*#*real i*#*bs*) *?es*)
using *splitex*[*OF qf*] **by** *simp*
with *ferrack01*[*OF simpfm-qf*[*OF exsplit-qf*[*OF qf*]]] **have** *th1*: *?rhs* = (∃ (*i*::*int*). *Ifm* (*real i*#*bs*) (*ferrack01* (*simpfm* (*exsplit p*)))) **and** *qf'*:*qfree* (*ferrack01* (*simpfm* (*exsplit p*))) **by** *simp+*
with *DJredlove*[*OF qf'*] **show** *?thesis* **by** (*simp add*: *mirlfr-def*)
qed
qed

constdefs *mircfrqe*:: *fm* ⇒ *fm*
mircfrqe ≡ (λ *p*. *qelim* (*prep p*) *mircfr*)

constdefs *mirlfrqe*:: *fm* ⇒ *fm*
mirlfrqe ≡ (λ *p*. *qelim* (*prep p*) *mirlfr*)

theorem *mircfrqe*: (*Ifm bs* (*mircfrqe p*) = *Ifm bs p*) ∧ *qfree* (*mircfrqe p*)
using *qelim-ci*[*OF mircfr*] *prep* **by** (*auto simp add*: *mircfrqe-def*)

theorem *mirlfrqe*: (*Ifm bs* (*mirlfrqe p*) = *Ifm bs p*) ∧ *qfree* (*mirlfrqe p*)
using *qelim-ci*[*OF mirlfr*] *prep* **by** (*auto simp add*: *mirlfrqe-def*)

declare *zdvd-iff-zmod-eq-0* [*code*]
declare *max-def* [*code unfold*]

definition
test1 (*u*::*unit*) = *mircfrqe* (*A* (*And* (*Le* (*Sub* (*Floor* (*Bound 0*)) (*Bound 0*))) (*Le* (*Add* (*Bound 0*) (*Floor* (*Neg* (*Bound 0*)))))))

definition
test2 (*u*::*unit*) = *mircfrqe* (*A* (*Iff* (*Eq* (*Add* (*Floor* (*Bound 0*)) (*Floor* (*Neg* (*Bound 0*))))) (*Eq* (*Sub* (*Floor* (*Bound 0*)) (*Bound 0*)))))

definition
test3 (*u*::*unit*) = *mirlfrqe* (*A* (*And* (*Le* (*Sub* (*Floor* (*Bound 0*)) (*Bound 0*))) (*Le* (*Add* (*Bound 0*) (*Floor* (*Neg* (*Bound 0*)))))))

definition
test4 (*u*::*unit*) = *mirlfrqe* (*A* (*Iff* (*Eq* (*Add* (*Floor* (*Bound 0*)) (*Floor* (*Neg* (*Bound 0*))))) (*Eq* (*Sub* (*Floor* (*Bound 0*)) (*Bound 0*)))))

definition
test5 (*u*::*unit*) = *mircfrqe* (*A*(*E*(*And* (*Ge*(*Sub* (*Bound 1*) (*Bound 0*))) (*Eq* (*Add* (*Floor* (*Bound 1*)) (*Floor* (*Neg*(*Bound 0*))))))))

export-code *mircfrqe mirlfrqe test1 test2 test3 test4 test5*
in *SML* **module-name** *Mir*

ML *set Toplevel.timing*
ML *Mir.test1* ()
ML *Mir.test2* ()
ML *Mir.test3* ()
ML *Mir.test4* ()
ML *Mir.test5* ()
ML *reset Toplevel.timing*

use *mireif.ML*
oracle *mircfr-oracle* (*term*) = *ReflectedMir.mircfr-oracle*
oracle *mirlfr-oracle* (*term*) = *ReflectedMir.mirlfr-oracle*
use *mirtac.ML*
setup *MirTac.setup*

ML *set Toplevel.timing*

lemma *ALL* (*x*::*real*). $(\lfloor x \rfloor = \lceil x \rceil = (x = real\ \lfloor x \rfloor))$
apply *mir*
done

lemma *ALL* (*x*::*real*). $real\ (2{::}int){*}x - (real\ (1{::}int)) < real\ \lfloor x \rfloor + real\ \lceil x \rceil \wedge real\ \lfloor x \rfloor + real\ \lceil x \rceil \ \leq real\ (2{::}int){*}x + (real\ (1{::}int))$
apply *mir*
done

lemma *ALL* (*x*::*real*). $2{*}\lfloor x \rfloor \leq \lfloor 2{*}x \rfloor \wedge \lfloor 2{*}x \rfloor \leq 2{*}\lfloor x{+}1 \rfloor$
apply *mir*

done

lemma *ALL* $(x::real)$. $\exists\, y \le x.\ (\lfloor x \rfloor = \lceil y \rceil)$
apply *mir*
done

ML *reset Toplevel.timing*

end

14 Implementation of natural numbers by integers

theory *Efficient-Nat*
imports *Main Code-Integer*
begin

When generating code for functions on natural numbers, the canonical representation using *0* and *Suc* is unsuitable for computations involving large numbers. The efficiency of the generated code can be improved drastically by implementing natural numbers by integers. To do this, just include this theory.

14.1 Logical rewrites

An int-to-nat conversion restricted to non-negative ints (in contrast to *nat*). Note that this restriction has no logical relevance and is just a kind of proof hint – nothing prevents you from writing nonsense like *nat-of-int* $(-4::'a)$

definition
  nat-of-int :: *int* $\Rightarrow$ *nat* **where**
  $k \ge 0 \Longrightarrow$ *nat-of-int* k = *nat* k

definition
  int-of-nat :: *nat* $\Rightarrow$ *int* **where**
  int-of-nat n = *of-nat* n

lemma *int-of-nat-Suc* [*simp*]:
  int-of-nat (*Suc* n) = *1* + *int-of-nat* n
  unfolding *int-of-nat-def* **by** *simp*

lemma *int-of-nat-add*:
  int-of-nat $(m + n)$ = *int-of-nat* m + *int-of-nat* n
  unfolding *int-of-nat-def* **by** (*rule of-nat-add*)

lemma *int-of-nat-mult*:
  int-of-nat $(m * n)$ = *int-of-nat* m $*$ *int-of-nat* n

unfolding *int-of-nat-def* **by** (*rule of-nat-mult*)

lemma *nat-of-int-of-number-of*:
fixes *k*
assumes $k \geq 0$
shows *number-of k* = *nat-of-int* (*number-of k*)
unfolding *nat-of-int-def* [*OF assms*] *nat-number-of-def number-of-is-id* **..**

lemma *nat-of-int-of-number-of-aux*:
fixes *k*
assumes *Numeral.Pls* $\leq$ *k* $\equiv$ *True*
shows $k \geq 0$
using *assms* **unfolding** *Pls-def* **by** *simp*

lemma *nat-of-int-int*:
nat-of-int (*int-of-nat n*) = *n*
using *nat-of-int-def int-of-nat-def* **by** *simp*

lemma *eq-nat-of-int*: *int-of-nat n* = *x* $\Longrightarrow$ *n* = *nat-of-int x*
by (*erule subst*, *simp only*: *nat-of-int-int*)

code-datatype *nat-of-int*

Case analysis on natural numbers is rephrased using a conditional expression:

lemma [*code unfold*, *code inline del*]:
nat-case $\equiv$ (λ*f g n. if n* = *0 then f else g* (*n* − *1*))
proof −
have *rewrite*: $\bigwedge$*f g n. nat-case f g n* = (*if n* = *0 then f else g* (*n* − *1*))
proof −
fix *f g n*
show *nat-case f g n* = (*if n* = *0 then f else g* (*n* − *1*))
by (*cases n*) *simp-all*
qed
show *nat-case* $\equiv$ (λ*f g n. if n* = *0 then f else g* (*n* − *1*))
by (*rule eq-reflection ext rewrite*)+
qed

lemma [*code inline*]:
nat-case = (λ*f g n. if n* = *0 then f else g* (*nat-of-int* (*int-of-nat n* − *1*)))
proof (*rule ext*)+
fix *f g n*
show *nat-case f g n* = (*if n* = *0 then f else g* (*nat-of-int* (*int-of-nat n* − *1*)))
by (*cases n*) (*simp-all add*: *nat-of-int-int*)
qed

Most standard arithmetic functions on natural numbers are implemented using their counterparts on the integers:

lemma [*code func*]: *0* = *nat-of-int 0*

by (*simp add*: *nat-of-int-def*)

lemma [*code func*, *code inline*]: *1* = *nat-of-int 1*
by (*simp add*: *nat-of-int-def*)

lemma [*code func*]: *Suc n* = *nat-of-int* (*int-of-nat n* + *1*)
by (*simp add*: *eq-nat-of-int*)

lemma [*code*]: *m* + *n* = *nat* (*int-of-nat m* + *int-of-nat n*)
by (*simp add*: *int-of-nat-def nat-eq-iff2*)

lemma [*code func*, *code inline*]: *m* + *n* = *nat-of-int* (*int-of-nat m* + *int-of-nat n*)
by (*simp add*: *eq-nat-of-int int-of-nat-add*)

lemma [*code*, *code inline*]: *m* − *n* = *nat* (*int-of-nat m* − *int-of-nat n*)
by (*simp add*: *int-of-nat-def nat-eq-iff2 of-nat-diff*)

lemma [*code*]: *m* ∗ *n* = *nat* (*int-of-nat m* ∗ *int-of-nat n*)
unfolding *int-of-nat-def*
by (*simp add*: *of-nat-mult* [*symmetric*] *del*: *of-nat-mult*)

lemma [*code func*, *code inline*]: *m* ∗ *n* = *nat-of-int* (*int-of-nat m* ∗ *int-of-nat n*)
by (*simp add*: *eq-nat-of-int int-of-nat-mult*)

lemma [*code*]: *m div n* = *nat* (*int-of-nat m div int-of-nat n*)
unfolding *int-of-nat-def zdiv-int* [*symmetric*] **by** *simp*

lemma *div-nat-code* [*code func*]:
m div k = *nat-of-int* (*fst* (*divAlg* (*int-of-nat m*, *int-of-nat k*)))
unfolding *div-def* [*symmetric*] *int-of-nat-def zdiv-int* [*symmetric*]
unfolding *int-of-nat-def* [*symmetric*] *nat-of-int-int* **..**

lemma [*code*]: *m mod n* = *nat* (*int-of-nat m mod int-of-nat n*)
unfolding *int-of-nat-def zmod-int* [*symmetric*] **by** *simp*

lemma *mod-nat-code* [*code func*]:
m mod k = *nat-of-int* (*snd* (*divAlg* (*int-of-nat m*, *int-of-nat k*)))
unfolding *mod-def* [*symmetric*] *int-of-nat-def zmod-int* [*symmetric*]
unfolding *int-of-nat-def* [*symmetric*] *nat-of-int-int* **..**

lemma [*code*, *code inline*]: (*m* < *n*) ⟷ (*int-of-nat m* < *int-of-nat n*)
unfolding *int-of-nat-def* **by** *simp*

lemma [*code func*, *code inline*]: (*m* ≤ *n*) ⟷ (*int-of-nat m* ≤ *int-of-nat n*)
unfolding *int-of-nat-def* **by** *simp*

lemma [*code func*, *code inline*]: *m* = *n* ⟷ *int-of-nat m* = *int-of-nat n*
unfolding *int-of-nat-def* **by** *simp*

lemma [*code func*]: *nat k* = (*if k* < *0 then 0 else nat-of-int k*)
  by (*cases k* < *0*) (*simp*, *simp add*: *nat-of-int-def*)

lemma [*code func*]:
  int-aux n i = (*if int-of-nat n* = *0 then i else int-aux* (*nat-of-int* (*int-of-nat n* − *1*)) (*i* + *1*))
proof −
  have *0* < *n* ⟹ *int-of-nat n* = *1* + *int-of-nat* (*nat-of-int* (*int-of-nat n* − *1*))
  proof −
    assume *prem*: *n* > *0*
    then have *int-of-nat n* − *1* ≥ *0* **unfolding** *int-of-nat-def* **by** *auto*
    then have *nat-of-int* (*int-of-nat n* − *1*) = *nat* (*int-of-nat n* − *1*) **by** (*simp add*: *nat-of-int-def*)
    with *prem* **show** *int-of-nat n* = *1* + *int-of-nat* (*nat-of-int* (*int-of-nat n* − *1*)) **unfolding** *int-of-nat-def* **by** *simp*
  qed
  then show *?thesis* **unfolding** *int-aux-def int-of-nat-def* **by** *auto*
qed

lemma *index-of-nat-code* [*code func*, *code inline*]:
  index-of-nat n = *index-of-int* (*int-of-nat n*)
  unfolding *index-of-nat-def int-of-nat-def* **by** *simp*

lemma *nat-of-index-code* [*code func*, *code inline*]:
  nat-of-index k = *nat* (*int-of-index k*)
  unfolding *nat-of-index-def* **by** *simp*

14.2 Code generator setup for basic functions

nat is no longer a datatype but embedded into the integers.

code-type *nat*
  (*SML int*)
  (*OCaml Big′-int.big′-int*)
  (*Haskell Integer*)

types-code
  nat (*int*)
attach (*term-of*) ⟪
val term-of-nat = *HOLogic.mk-number HOLogic.natT*;
⟫
attach (*test*) ⟪
fun gen-nat i = *random-range 0 i*;
⟫

consts-code
  0 :: *nat* (*0*)
  Suc ((- + *1*))

Since natural numbers are implemented using integers, the coercion function *int* of type $nat \Rightarrow int$ is simply implemented by the identity function, likewise *nat-of-int* of type $int \Rightarrow nat$. For the *nat* function for converting an integer to a natural number, we give a specific implementation using an ML function that returns its input value, provided that it is non-negative, and otherwise returns *0*.

```
consts-code
  int-of-nat ((-))
  nat (⟨module⟩nat)
attach ⟪
fun nat i = if i < 0 then 0 else i;
⟫

code-const int-of-nat
  (SML -)
  (OCaml -)
  (Haskell -)

code-const nat-of-int
  (SML -)
  (OCaml -)
  (Haskell -)
```

14.3 Preprocessors

Natural numerals should be expressed using *nat-of-int*.

```
lemmas [code inline del] = nat-number-of-def

ML ⟪
fun nat-of-int-of-number-of thy cts =
  let
    val simplify-less = Simplifier.rewrite
    (HOL-basic-ss addsimps (@{thms less-numeral-code} @ @{thms less-eq-numeral-code}));
    fun mk-rew (t, ty) =
      if ty = HOLogic.natT andalso 0 <= HOLogic.dest-numeral t then
        Thm.capply @{cterm (op ≤) Numeral.Pls} (Thm.cterm-of thy t)
        |> simplify-less
        |> (fn thm => @{thm nat-of-int-of-number-of-aux} OF [thm])
        |> (fn thm => @{thm nat-of-int-of-number-of} OF [thm])
        |> (fn thm => @{thm eq-reflection} OF [thm])
        |> SOME
      else NONE
  in
    fold (HOLogic.add-numerals o Thm.term-of) cts []
    |> map-filter mk-rew
  end;
⟫

setup ⟪
```

Code.add-inline-proc (*nat-of-int-of-number-of*, *nat-of-int-of-number-of*)
⟫

In contrast to *Suc n*, the term $n + 1$ is no longer a constructor term. Therefore, all occurrences of this term in a position where a pattern is expected (i.e. on the left-hand side of a recursion equation or in the arguments of an inductive relation in an introduction rule) must be eliminated. This can be accomplished by applying the following transformation rules:

theorem *Suc-if-eq*: $(\bigwedge n.\ f\ (Suc\ n) = h\ n) \Longrightarrow f\ 0 = g \Longrightarrow$
$f\ n = (if\ n = 0\ then\ g\ else\ h\ (n - 1))$
by (*case-tac n*) *simp-all*

theorem *Suc-clause*: $(\bigwedge n.\ P\ n\ (Suc\ n)) \Longrightarrow n \neq 0 \Longrightarrow P\ (n - 1)\ n$
by (*case-tac n*) *simp-all*

The rules above are built into a preprocessor that is plugged into the code generator. Since the preprocessor for introduction rules does not know anything about modes, some of the modes that worked for the canonical representation of natural numbers may no longer work.

14.4 Module names

code-modulename *SML*
Nat Integer
Divides Integer
Efficient-Nat Integer

code-modulename *OCaml*
Nat Integer
Divides Integer
Efficient-Nat Integer

code-modulename *Haskell*
Nat Integer
Divides Integer
Efficient-Nat Integer

hide *const nat-of-int int-of-nat*

end

15 Quatifier elimination for R(0,1,+,¡)

theory *ReflectedFerrack*
imports *GCD Real Efficient-Nat*
uses (*linreif.ML*) (*linrtac.ML*)
begin

consts *alluopairs*:: $'a$ *list* $\Rightarrow$ $('a \times 'a)$ *list*
primrec
  alluopairs [] = []
  alluopairs $(x\#xs)$ = (*map* (*Pair* x) $(x\#xs)$)@(*alluopairs* xs)

lemma *alluopairs-set1*: *set* (*alluopairs* xs) $\leq \{(x,y).\ x\in$ *set* $xs \wedge y\in$ *set* $xs\}$
by (*induct* xs, *auto*)

lemma *alluopairs-set*:
  $[\![x\in$ *set* xs ; $y \in$ *set* $xs]\!] \Longrightarrow (x,y) \in$ *set* (*alluopairs* xs) $\vee$ $(y,x) \in$ *set* (*alluopairs* xs)
by (*induct* xs, *auto*)

lemma *alluopairs-ex*:
  assumes *Pc*: $\forall\ x\ y.\ P\ x\ y = P\ y\ x$
  shows $(\exists\ x \in$ *set* $xs.\ \exists\ y \in$ *set* $xs.\ P\ x\ y) = (\exists\ (x,y) \in$ *set* (*alluopairs* xs). $P\ x\ y)$
proof
  assume $\exists x\in$*set* $xs.\ \exists y\in$*set* $xs.\ P\ x\ y$
  then obtain $x\ y$ **where** x: $x \in$ *set* xs **and** y:$y \in$ *set* xs **and** P: $P\ x\ y$ **by** *blast*
  from *alluopairs-set*[*OF* $x\ y$] P *Pc* **show**$\exists(x,\ y)\in$*set* (*alluopairs* xs). $P\ x\ y$
    by *auto*
next
  assume $\exists(x,\ y)\in$*set* (*alluopairs* xs). $P\ x\ y$
  then obtain x **and** y **where** *xy*:$(x,y) \in$ *set* (*alluopairs* xs) **and** P: $P\ x\ y$ **by** *blast+*
  from *xy* **have** $x \in$ *set* $xs \wedge y\in$ *set* xs **using** *alluopairs-set1* **by** *blast*
  with P **show** $\exists x\in$*set* $xs.\ \exists y\in$*set* $xs.\ P\ x\ y$ **by** *blast*
qed

lemma *nth-pos2*: $0 < n \Longrightarrow (x\#xs)\ !\ n = xs\ !\ (n - 1)$
using *Nat.gr0-conv-Suc*
by *clarsimp*

lemma *filter-length*: *length* (*List.filter* $P\ xs$) < *Suc* (*length* xs)
  apply (*induct* xs, *auto*) **done**

consts *remdps*:: $'a$ *list* $\Rightarrow$ $'a$ *list*

recdef *remdps measure size*
  remdps [] = []
  remdps $(x\#xs)$ = $(x\#($*remdps* (*List.filter* $(\lambda\ y.\ y \neq x)\ xs)))$
(**hints** *simp add*: *filter-length*[*rule-format*])

lemma *remdps-set*[*simp*]: *set* (*remdps xs*) = *set xs*
  by (*induct xs rule*: *remdps.induct*, *auto*)

datatype *num* = *C int* | *Bound nat* | *CN nat int num* | *Neg num* | *Add num num*|
Sub num num
  | *Mul int num*

consts *num-size* :: *num* ⇒ *nat*
primrec
  num-size (*C c*) = *1*
  num-size (*Bound n*) = *1*
  num-size (*Neg a*) = *1* + *num-size a*
  num-size (*Add a b*) = *1* + *num-size a* + *num-size b*
  num-size (*Sub a b*) = *3* + *num-size a* + *num-size b*
  num-size (*Mul c a*) = *1* + *num-size a*
  num-size (*CN n c a*) = *3* + *num-size a*

consts *Inum* :: *real list* ⇒ *num* ⇒ *real*
primrec
  Inum bs (*C c*) = (*real c*)
  Inum bs (*Bound n*) = *bs*!*n*
  Inum bs (*CN n c a*) = (*real c*) * (*bs*!*n*) + (*Inum bs a*)
  Inum bs (*Neg a*) = −(*Inum bs a*)
  Inum bs (*Add a b*) = *Inum bs a* + *Inum bs b*
  Inum bs (*Sub a b*) = *Inum bs a* − *Inum bs b*
  Inum bs (*Mul c a*) = (*real c*) * *Inum bs a*

datatype *fm* =
  T| *F*| *Lt num*| *Le num*| *Gt num*| *Ge num*| *Eq num*| *NEq num*|
  NOT fm| *And fm fm*| *Or fm fm*| *Imp fm fm*| *Iff fm fm*| *E fm*| *A fm*

consts *fmsize* :: *fm* ⇒ *nat*
recdef *fmsize measure size*
  fmsize (*NOT p*) = *1* + *fmsize p*
  fmsize (*And p q*) = *1* + *fmsize p* + *fmsize q*
  fmsize (*Or p q*) = *1* + *fmsize p* + *fmsize q*
  fmsize (*Imp p q*) = *3* + *fmsize p* + *fmsize q*
  fmsize (*Iff p q*) = *3* + *2**(*fmsize p* + *fmsize q*)

fmsize $(E\ p) = 1 + $ *fmsize p*
fmsize $(A\ p) = 4+$ *fmsize p*
fmsize p $= 1$

lemma *fmsize-pos*: *fmsize p* > 0
by (*induct p rule*: *fmsize.induct*) *simp-all*

consts *Ifm* ::*real list* $\Rightarrow$ *fm* $\Rightarrow$ *bool*
primrec
Ifm bs T = *True*
Ifm bs F = *False*
Ifm bs (*Lt a*) = (*Inum bs a* $<$ *0*)
Ifm bs (*Gt a*) = (*Inum bs a* $>$ *0*)
Ifm bs (*Le a*) = (*Inum bs a* $\leq$ *0*)
Ifm bs (*Ge a*) = (*Inum bs a* $\geq$ *0*)
Ifm bs (*Eq a*) = (*Inum bs a* = *0*)
Ifm bs (*NEq a*) = (*Inum bs a* $\neq$ *0*)
Ifm bs (*NOT p*) = ($\neg$ (*Ifm bs p*))
Ifm bs (*And p q*) = (*Ifm bs p* $\wedge$ *Ifm bs q*)
Ifm bs (*Or p q*) = (*Ifm bs p* $\vee$ *Ifm bs q*)
Ifm bs (*Imp p q*) = ((*Ifm bs p*) $\longrightarrow$ (*Ifm bs q*))
Ifm bs (*Iff p q*) = (*Ifm bs p* = *Ifm bs q*)
Ifm bs (*E p*) = ($\exists$ *x*. *Ifm* (*x#bs*) *p*)
Ifm bs (*A p*) = ($\forall$ *x*. *Ifm* (*x#bs*) *p*)

lemma *IfmLeSub*: $[\![$ *Inum bs s* = s' ; *Inum bs t* = t' $]\!] \Longrightarrow$ *Ifm bs* (*Le* (*Sub s t*)) = ($s' \leq t'$)
apply *simp*
done

lemma *IfmLtSub*: $[\![$ *Inum bs s* = s' ; *Inum bs t* = t' $]\!] \Longrightarrow$ *Ifm bs* (*Lt* (*Sub s t*)) = ($s' < t'$)
apply *simp*
done
lemma *IfmEqSub*: $[\![$ *Inum bs s* = s' ; *Inum bs t* = t' $]\!] \Longrightarrow$ *Ifm bs* (*Eq* (*Sub s t*)) = ($s' = t'$)
apply *simp*
done
lemma *IfmNOT*: (*Ifm bs p* = *P*) $\Longrightarrow$ (*Ifm bs* (*NOT p*) = ($\neg$*P*))
apply *simp*
done
lemma *IfmAnd*: $[\![$ *Ifm bs p* = *P* ; *Ifm bs q* = *Q*$]\!] \Longrightarrow$ (*Ifm bs* (*And p q*) = (*P* $\wedge$ *Q*))
apply *simp*
done
lemma *IfmOr*: $[\![$ *Ifm bs p* = *P* ; *Ifm bs q* = *Q*$]\!] \Longrightarrow$ (*Ifm bs* (*Or p q*) = (*P* $\vee$ *Q*))
apply *simp*
done

lemma *IfmImp*: $[\![$ *Ifm bs p* = *P* ; *Ifm bs q* = *Q*$]\!] \Longrightarrow$ (*Ifm bs* (*Imp p q*) = (*P* $\longrightarrow$ *Q*))
apply *simp*
done
lemma *IfmIff*: $[\![$ *Ifm bs p* = *P* ; *Ifm bs q* = *Q*$]\!] \Longrightarrow$ (*Ifm bs* (*Iff p q*) = (*P* = *Q*))
apply *simp*
done

lemma *IfmE*: (!! *x*. *Ifm* (*x#bs*) *p* = *P x*) $\Longrightarrow$ (*Ifm bs* (*E p*) = ($\exists$ *x*. *P x*))
apply *simp*
done
lemma *IfmA*: (!! *x*. *Ifm* (*x#bs*) *p* = *P x*) $\Longrightarrow$ (*Ifm bs* (*A p*) = ($\forall$ *x*. *P x*))
apply *simp*
done

consts *not*:: *fm* $\Rightarrow$ *fm*
recdef *not measure size*
  not (*NOT p*) = *p*
  not T = *F*
  not F = *T*
  not p = *NOT p*
lemma *not*[*simp*]: *Ifm bs* (*not p*) = *Ifm bs* (*NOT p*)
by (*cases p*) *auto*

constdefs *conj* :: *fm* $\Rightarrow$ *fm* $\Rightarrow$ *fm*
  conj p q $\equiv$ (*if* (*p* = *F* $\vee$ *q*=*F*) *then F else if p*=*T then q else if q*=*T then p else*
   if p = *q then p else And p q*)
lemma *conj*[*simp*]: *Ifm bs* (*conj p q*) = *Ifm bs* (*And p q*)
by (*cases p*=*F* $\vee$ *q*=*F*,*simp-all add*: *conj-def*) (*cases p*,*simp-all*)

constdefs *disj* :: *fm* $\Rightarrow$ *fm* $\Rightarrow$ *fm*
  disj p q $\equiv$ (*if* (*p* = *T* $\vee$ *q*=*T*) *then T else if p*=*F then q else if q*=*F then p*
     else if p=*q then p else Or p q*)

lemma *disj*[*simp*]: *Ifm bs* (*disj p q*) = *Ifm bs* (*Or p q*)
by (*cases p*=*T* $\vee$ *q*=*T*,*simp-all add*: *disj-def*) (*cases p*,*simp-all*)

constdefs *imp* :: *fm* $\Rightarrow$ *fm* $\Rightarrow$ *fm*
  imp p q $\equiv$ (*if* (*p* = *F* $\vee$ *q*=*T* $\vee$ *p*=*q*) *then T else if p*=*T then q else if q*=*F then not p*
    else Imp p q)
lemma *imp*[*simp*]: *Ifm bs* (*imp p q*) = *Ifm bs* (*Imp p q*)
by (*cases p*=*F* $\vee$ *q*=*T*,*simp-all add*: *imp-def*)

constdefs *iff* :: *fm* $\Rightarrow$ *fm* $\Rightarrow$ *fm*
  iff p q $\equiv$ (*if* (*p* = *q*) *then T else if* (*p* = *NOT q* $\vee$ *NOT p* = *q*) *then F else*
     if p=*F then not q else if q*=*F then not p else if p*=*T then q else if q*=*T then p else*
  Iff p q)

lemma *iff* [*simp*]: *Ifm bs* (*iff p q*) = *Ifm bs* (*Iff p q*)
  by (*unfold iff-def*,*cases p*=*q*, *simp*,*cases p*=*NOT q*, *simp*) (*cases NOT p*= *q*, *auto*)

lemma *conj-simps*:
  $conj\ F\ Q = F$
  $conj\ P\ F = F$
  $conj\ T\ Q = Q$
  $conj\ P\ T = P$
  $conj\ P\ P = P$
  $P \neq T \Longrightarrow P \neq F \Longrightarrow Q \neq T \Longrightarrow Q \neq F \Longrightarrow P \neq Q \Longrightarrow conj\ P\ Q = And\ P\ Q$
  by (*simp-all add*: *conj-def*)

lemma *disj-simps*:
  $disj\ T\ Q = T$
  $disj\ P\ T = T$
  $disj\ F\ Q = Q$
  $disj\ P\ F = P$
  $disj\ P\ P = P$
  $P \neq T \Longrightarrow P \neq F \Longrightarrow Q \neq T \Longrightarrow Q \neq F \Longrightarrow P \neq Q \Longrightarrow disj\ P\ Q = Or\ P\ Q$
  by (*simp-all add*: *disj-def*)
lemma *imp-simps*:
  $imp\ F\ Q = T$
  $imp\ P\ T = T$
  $imp\ T\ Q = Q$
  $imp\ P\ F = not\ P$
  $imp\ P\ P = T$
  $P \neq T \Longrightarrow P \neq F \Longrightarrow P \neq Q \Longrightarrow Q \neq T \Longrightarrow Q \neq F \Longrightarrow imp\ P\ Q = Imp\ P\ Q$
  by (*simp-all add*: *imp-def*)
lemma *trivNOT*: $p \neq NOT\ p$ $NOT\ p \neq p$
apply (*induct p*, *auto*)
done

lemma *iff-simps*:
  $iff\ p\ p = T$
  $iff\ p\ (NOT\ p) = F$
  $iff\ (NOT\ p)\ p = F$
  $iff\ p\ F = not\ p$
  $iff\ F\ p = not\ p$
  $p \neq NOT\ T \Longrightarrow iff\ T\ p = p$
  $p\neq NOT\ T \Longrightarrow iff\ p\ T = p$
  $p\neq q \Longrightarrow p\neq NOT\ q \Longrightarrow q\neq NOT\ p \Longrightarrow p\neq F \Longrightarrow q\neq F \Longrightarrow p \neq T \Longrightarrow q \neq T \Longrightarrow iff\ p\ q = Iff\ p\ q$
  using *trivNOT*
  by (*simp-all add*: *iff-def*, *cases p*, *auto*)

consts *qfree*:: *fm* ⇒ *bool*

```
recdef qfree measure size
  qfree (E p) = False
  qfree (A p) = False
  qfree (NOT p) = qfree p
  qfree (And p q) = (qfree p ∧ qfree q)
  qfree (Or  p q) = (qfree p ∧ qfree q)
  qfree (Imp p q) = (qfree p ∧ qfree q)
  qfree (Iff p q) = (qfree p ∧ qfree q)
  qfree p = True


consts
  numbound0:: num ⇒ bool
  bound0:: fm ⇒ bool
primrec
  numbound0 (C c) = True
  numbound0 (Bound n) = (n>0)
  numbound0 (CN n c a) = (n≠0 ∧ numbound0 a)
  numbound0 (Neg a) = numbound0 a
  numbound0 (Add a b) = (numbound0 a ∧ numbound0 b)
  numbound0 (Sub a b) = (numbound0 a ∧ numbound0 b)
  numbound0 (Mul i a) = numbound0 a
lemma numbound0-I:
  assumes nb: numbound0 a
  shows Inum (b#bs) a = Inum (b'#bs) a
using nb
by (induct a rule: numbound0.induct,auto simp add: nth-pos2)

primrec
  bound0 T = True
  bound0 F = True
  bound0 (Lt a) = numbound0 a
  bound0 (Le a) = numbound0 a
  bound0 (Gt a) = numbound0 a
  bound0 (Ge a) = numbound0 a
  bound0 (Eq a) = numbound0 a
  bound0 (NEq a) = numbound0 a
  bound0 (NOT p) = bound0 p
  bound0 (And p q) = (bound0 p ∧ bound0 q)
  bound0 (Or p q) = (bound0 p ∧ bound0 q)
  bound0 (Imp p q) = ((bound0 p) ∧ (bound0 q))
  bound0 (Iff p q) = (bound0 p ∧ bound0 q)
  bound0 (E p) = False
  bound0 (A p) = False

lemma bound0-I:
  assumes bp: bound0 p
  shows Ifm (b#bs) p = Ifm (b'#bs) p
using bp numbound0-I[where b=b and bs=bs and b'=b']
```

by (*induct p rule*: *bound0.induct*) (*auto simp add*: *nth-pos2*)

lemma *not-qf*[*simp*]: *qfree p* ⟹ *qfree* (*not p*)
by (*cases p*, *auto*)
lemma *not-bn*[*simp*]: *bound0 p* ⟹ *bound0* (*not p*)
by (*cases p*, *auto*)

lemma *conj-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*conj p q*)
using *conj-def* **by** *auto*
lemma *conj-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*conj p q*)
using *conj-def* **by** *auto*

lemma *disj-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*disj p q*)
using *disj-def* **by** *auto*
lemma *disj-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*disj p q*)
using *disj-def* **by** *auto*

lemma *imp-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*imp p q*)
using *imp-def* **by** (*cases p=F* ∨ *q=T*,*simp-all add*: *imp-def*)
lemma *imp-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*imp p q*)
using *imp-def* **by** (*cases p=F* ∨ *q=T* ∨ *p=q*,*simp-all add*: *imp-def*)

lemma *iff-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*iff p q*)
  by (*unfold iff-def*,*cases p=q*, *auto*)
lemma *iff-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*iff p q*)
using *iff-def* **by** (*unfold iff-def*,*cases p=q*, *auto*)

consts
  decrnum:: *num* ⇒ *num*
  decr :: *fm* ⇒ *fm*

recdef *decrnum measure size*
  decrnum (*Bound n*) = *Bound* (*n* − *1*)
  decrnum (*Neg a*) = *Neg* (*decrnum a*)
  decrnum (*Add a b*) = *Add* (*decrnum a*) (*decrnum b*)
  decrnum (*Sub a b*) = *Sub* (*decrnum a*) (*decrnum b*)
  decrnum (*Mul c a*) = *Mul c* (*decrnum a*)
  decrnum (*CN n c a*) = *CN* (*n* − *1*) *c* (*decrnum a*)
  decrnum a = *a*

recdef *decr measure size*
  decr (*Lt a*) = *Lt* (*decrnum a*)
  decr (*Le a*) = *Le* (*decrnum a*)
  decr (*Gt a*) = *Gt* (*decrnum a*)
  decr (*Ge a*) = *Ge* (*decrnum a*)
  decr (*Eq a*) = *Eq* (*decrnum a*)
  decr (*NEq a*) = *NEq* (*decrnum a*)
  decr (*NOT p*) = *NOT* (*decr p*)

decr (*And p q*) = *conj* (*decr p*) (*decr q*)
decr (*Or p q*) = *disj* (*decr p*) (*decr q*)
decr (*Imp p q*) = *imp* (*decr p*) (*decr q*)
decr (*Iff p q*) = *iff* (*decr p*) (*decr q*)
decr p = *p*

lemma *decrnum*: **assumes** *nb*: *numbound0 t*
shows *Inum* (*x#bs*) *t* = *Inum bs* (*decrnum t*)
using *nb* **by** (*induct t rule*: *decrnum.induct*, *simp-all add*: *nth-pos2*)

lemma *decr*: **assumes** *nb*: *bound0 p*
shows *Ifm* (*x#bs*) *p* = *Ifm bs* (*decr p*)
using *nb*
by (*induct p rule*: *decr.induct*, *simp-all add*: *nth-pos2 decrnum*)

lemma *decr-qf*: *bound0 p* $\Longrightarrow$ *qfree* (*decr p*)
by (*induct p*, *simp-all*)

consts
isatom :: *fm* $\Rightarrow$ *bool*
recdef *isatom measure size*
isatom T = *True*
isatom F = *True*
isatom (*Lt a*) = *True*
isatom (*Le a*) = *True*
isatom (*Gt a*) = *True*
isatom (*Ge a*) = *True*
isatom (*Eq a*) = *True*
isatom (*NEq a*) = *True*
isatom p = *False*

lemma *bound0-qf*: *bound0 p* $\Longrightarrow$ *qfree p*
by (*induct p*, *simp-all*)

constdefs *djf*:: (*'a* $\Rightarrow$ *fm*) $\Rightarrow$ *'a* $\Rightarrow$ *fm* $\Rightarrow$ *fm*
djf f p q $\equiv$ (*if q=T then T else if q=F then f p else*
(*let fp* = *f p in case fp of T* $\Rightarrow$ *T* | *F* $\Rightarrow$ *q* | - $\Rightarrow$ *Or* (*f p*) *q*))
constdefs *evaldjf*:: (*'a* $\Rightarrow$ *fm*) $\Rightarrow$ *'a list* $\Rightarrow$ *fm*
evaldjf f ps $\equiv$ *foldr* (*djf f*) *ps F*

lemma *djf-Or*: *Ifm bs* (*djf f p q*) = *Ifm bs* (*Or* (*f p*) *q*)
by (*cases q=T*, *simp add*: *djf-def*,*cases q=F*,*simp add*: *djf-def*)
(*cases f p*, *simp-all add*: *Let-def djf-def*)

lemma *djf-simps*:
djf f p T = *T*
djf f p F = *f p*
q$\neq$*T* $\Longrightarrow$ *q*$\neq$*F* $\Longrightarrow$ *djf f p q* = (*let fp* = *f p in case fp of T* $\Rightarrow$ *T* | *F* $\Rightarrow$ *q* | - $\Rightarrow$

Or (*f p*) *q*)
  by (*simp-all add*: *djf-def*)

lemma *evaldjf-ex*: *Ifm bs* (*evaldjf f ps*) = (∃ *p* ∈ *set ps*. *Ifm bs* (*f p*))
  by(*induct ps*, *simp-all add*: *evaldjf-def djf-Or*)

lemma *evaldjf-bound0*:
  assumes *nb*: ∀ *x*∈ *set xs*. *bound0* (*f x*)
  shows *bound0* (*evaldjf f xs*)
  using *nb* **by** (*induct xs*, *auto simp add*: *evaldjf-def djf-def Let-def*) (*case-tac f a*, *auto*)

lemma *evaldjf-qf*:
  assumes *nb*: ∀ *x*∈ *set xs*. *qfree* (*f x*)
  shows *qfree* (*evaldjf f xs*)
  using *nb* **by** (*induct xs*, *auto simp add*: *evaldjf-def djf-def Let-def*) (*case-tac f a*, *auto*)

consts *disjuncts* :: *fm* ⇒ *fm list*
recdef *disjuncts measure size*
  disjuncts (*Or p q*) = (*disjuncts p*) @ (*disjuncts q*)
  disjuncts F = []
  disjuncts p = [*p*]

lemma *disjuncts*: (∃ *q*∈ *set* (*disjuncts p*). *Ifm bs q*) = *Ifm bs p*
by(*induct p rule*: *disjuncts.induct*, *auto*)

lemma *disjuncts-nb*: *bound0 p* ⟹ ∀ *q*∈ *set* (*disjuncts p*). *bound0 q*
proof−
  assume *nb*: *bound0 p*
  hence *list-all bound0* (*disjuncts p*) **by** (*induct p rule*:*disjuncts.induct*,*auto*)
  thus *?thesis* **by** (*simp only*: *list-all-iff*)
qed

lemma *disjuncts-qf*: *qfree p* ⟹ ∀ *q*∈ *set* (*disjuncts p*). *qfree q*
proof−
  assume *qf*: *qfree p*
  hence *list-all qfree* (*disjuncts p*)
    by (*induct p rule*: *disjuncts.induct*, *auto*)
  thus *?thesis* **by** (*simp only*: *list-all-iff*)
qed

constdefs *DJ* :: (*fm* ⇒ *fm*) ⇒ *fm* ⇒ *fm*
  DJ f p ≡ *evaldjf f* (*disjuncts p*)

lemma *DJ*: **assumes** *fdj*: ∀ *p q*. *Ifm bs* (*f* (*Or p q*)) = *Ifm bs* (*Or* (*f p*) (*f q*))
  and *fF*: *f F* = *F*
  shows *Ifm bs* (*DJ f p*) = *Ifm bs* (*f p*)
proof−

have *Ifm bs* (*DJ f p*) = (∃ *q* ∈ *set* (*disjuncts p*). *Ifm bs* (*f q*))
by (*simp add*: *DJ-def evaldjf-ex*)
also have ... = *Ifm bs* (*f p*) **using** *fdj fF* **by** (*induct p rule*: *disjuncts.induct*, *auto*)
finally show *?thesis* **.**
qed

lemma *DJ-qf*: **assumes**
fqf: ∀ *p*. *qfree p* ⟶ *qfree* (*f p*)
shows ∀ *p*. *qfree p* ⟶ *qfree* (*DJ f p*)
proof(*clarify*)
fix *p* **assume** *qf*: *qfree p*
have *th*: *DJ f p* = *evaldjf f* (*disjuncts p*) **by** (*simp add*: *DJ-def*)
from *disjuncts-qf*[*OF qf*] **have** ∀ *q*∈ *set* (*disjuncts p*). *qfree q* **.**
with *fqf* **have** *th'*:∀ *q*∈ *set* (*disjuncts p*). *qfree* (*f q*) **by** *blast*

from *evaldjf-qf*[*OF th'*] *th* **show** *qfree* (*DJ f p*) **by** *simp*
qed

lemma *DJ-qe*: **assumes** *qe*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ∀ *bs p*. *qfree p* ⟶ *qfree* (*DJ qe p*) ∧ (*Ifm bs* ((*DJ qe p*)) = *Ifm bs* (*E p*))
proof(*clarify*)
fix *p*::*fm* **and** *bs*
assume *qf*: *qfree p*
from *qe* **have** *qth*: ∀ *p*. *qfree p* ⟶ *qfree* (*qe p*) **by** *blast*
from *DJ-qf*[*OF qth*] *qf* **have** *qfth*:*qfree* (*DJ qe p*) **by** *auto*
have *Ifm bs* (*DJ qe p*) = (∃ *q*∈ *set* (*disjuncts p*). *Ifm bs* (*qe q*))
by (*simp add*: *DJ-def evaldjf-ex*)
also have ... = (∃ *q* ∈ *set*(*disjuncts p*). *Ifm bs* (*E q*)) **using** *qe disjuncts-qf*[*OF qf*] **by** *auto*
also have ... = *Ifm bs* (*E p*) **by** (*induct p rule*: *disjuncts.induct*, *auto*)
finally show *qfree* (*DJ qe p*) ∧ *Ifm bs* (*DJ qe p*) = *Ifm bs* (*E p*) **using** *qfth* **by** *blast*
qed

consts
numgcd :: *num* ⇒ *int*
numgcdh:: *num* ⇒ *int* ⇒ *int*
reducecoeffh:: *num* ⇒ *int* ⇒ *num*
reducecoeff :: *num* ⇒ *num*
dvdnumcoeff:: *num* ⇒ *int* ⇒ *bool*
consts *maxcoeff*:: *num* ⇒ *int*
recdef *maxcoeff measure size*
maxcoeff (*C i*) = *abs i*
maxcoeff (*CN n c t*) = *max* (*abs c*) (*maxcoeff t*)
maxcoeff t = *1*

lemma *maxcoeff-pos*: *maxcoeff t* $\geq$ *0*
  by (*induct t rule*: *maxcoeff.induct*, *auto*)

recdef *numgcdh measure size*
  numgcdh (*C i*) = (λ*g*. *igcd i g*)
  numgcdh (*CN n c t*) = (λ*g*. *igcd c* (*numgcdh t g*))
  numgcdh t = (λ*g*. *1*)
defs *numgcd-def* [*code func*]: *numgcd t* $\equiv$ *numgcdh t* (*maxcoeff t*)

recdef *reducecoeffh measure size*
  reducecoeffh (*C i*) = (λ *g*. *C* (*i div g*))
  reducecoeffh (*CN n c t*) = (λ *g*. *CN n* (*c div g*) (*reducecoeffh t g*))
  reducecoeffh t = (λ*g*. *t*)

defs *reducecoeff-def*: *reducecoeff t* $\equiv$
  (*let g* = *numgcd t in*
  if g = *0 then C 0 else if g=1 then t else reducecoeffh t g*)

recdef *dvdnumcoeff measure size*
  dvdnumcoeff (*C i*) = (λ *g*. *g dvd i*)
  dvdnumcoeff (*CN n c t*) = (λ *g*. *g dvd c* $\wedge$ (*dvdnumcoeff t g*))
  dvdnumcoeff t = (λ*g*. *False*)

lemma *dvdnumcoeff-trans*:
  assumes *gdg*: *g dvd g'* **and** *dgt'*:*dvdnumcoeff t g'*
  shows *dvdnumcoeff t g*
  using *dgt' gdg*
  by (*induct t rule*: *dvdnumcoeff.induct*, *simp-all add*: *gdg zdvd-trans*[*OF gdg*])

declare *zdvd-trans* [*trans add*]

lemma *natabs0*: (*nat* (*abs x*) = *0*) = (*x* = *0*)
by *arith*

lemma *numgcd0*:
  assumes *g0*: *numgcd t* = *0*
  shows *Inum bs t* = *0*
  using *g0*[*simplified numgcd-def*]
  by (*induct t rule*: *numgcdh.induct*, *auto simp add*: *igcd-def gcd-zero natabs0 max-def maxcoeff-pos*)

lemma *numgcdh-pos*: **assumes** *gp*: *g* $\geq$ *0* **shows** *numgcdh t g* $\geq$ *0*
  using *gp*
  by (*induct t rule*: *numgcdh.induct*, *auto simp add*: *igcd-def*)

lemma *numgcd-pos*: *numgcd t* $\geq$*0*
  by (*simp add*: *numgcd-def numgcdh-pos maxcoeff-pos*)

lemma *reducecoeffh*:

assumes *gt*: *dvdnumcoeff t g* **and** *gp*: $g > 0$
shows *real* $g *(Inum\ bs\ (reducecoeffh\ t\ g)) = Inum\ bs\ t$
using *gt*
proof(*induct t rule*: *reducecoeffh.induct*)
case (*1 i*) **hence** *gd*: *g dvd i* **by** *simp*
from *gp* **have** *gnz*: $g \neq 0$ **by** *simp*
from *prems* **show** *?case* **by** (*simp add*: *real-of-int-div*[*OF gnz gd*])
next
case (*2 n c t*) **hence** *gd*: *g dvd c* **by** *simp*
from *gp* **have** *gnz*: $g \neq 0$ **by** *simp*
from *prems* **show** *?case* **by** (*simp add*: *real-of-int-div*[*OF gnz gd*] *ring-simps*)
qed (*auto simp add*: *numgcd-def gp*)
consts *ismaxcoeff*:: $num \Rightarrow int \Rightarrow bool$
recdef *ismaxcoeff measure size*
ismaxcoeff $(C\ i) = (\lambda\ x.\ abs\ i \leq x)$
ismaxcoeff $(CN\ n\ c\ t) = (\lambda x.\ abs\ c \leq x \wedge (ismaxcoeff\ t\ x))$
ismaxcoeff $t = (\lambda x.\ True)$

lemma *ismaxcoeff-mono*: *ismaxcoeff* $t\ c \Longrightarrow c \leq c' \Longrightarrow$ *ismaxcoeff* $t\ c'$
by (*induct t rule*: *ismaxcoeff.induct*, *auto*)

lemma *maxcoeff-ismaxcoeff*: *ismaxcoeff t* (*maxcoeff t*)
proof (*induct t rule*: *maxcoeff.induct*)
case (*2 n c t*)
hence *H*:*ismaxcoeff t* (*maxcoeff t*) **.**
have *thh*: *maxcoeff* $t \leq max\ (abs\ c)$ (*maxcoeff t*) **by** (*simp add*: *le-maxI2*)
from *ismaxcoeff-mono*[*OF H thh*] **show** *?case* **by** (*simp add*: *le-maxI1*)
qed *simp-all*

lemma *igcd-gt1*: $igcd\ i\ j > 1 \Longrightarrow ((abs\ i > 1 \wedge abs\ j > 1) \vee (abs\ i = 0 \wedge abs\ j > 1) \vee (abs\ i > 1 \wedge abs\ j = 0))$
apply (*cases* $abs\ i = 0$, *simp-all add*: *igcd-def*)
apply (*cases* $abs\ j = 0$, *simp-all*)
apply (*cases* $abs\ i = 1$, *simp-all*)
apply (*cases* $abs\ j = 1$, *simp-all*)
apply *auto*
done
lemma *numgcdh0*:$numgcdh\ t\ m = 0 \Longrightarrow m = 0$
by (*induct t rule*: *numgcdh.induct*, *auto simp add*:*igcd0*)

lemma *dvdnumcoeff-aux*:
assumes *ismaxcoeff t m* **and** *mp*:$m \geq 0$ **and** $numgcdh\ t\ m > 1$
shows *dvdnumcoeff t* (*numgcdh t m*)
using *prems*
proof(*induct t rule*: *numgcdh.induct*)
case (*2 n c t*)
let $?g = numgcdh\ t\ m$
from *prems* **have** *th*:$igcd\ c\ ?g > 1$ **by** *simp*
from *igcd-gt1*[*OF th*] *numgcdh-pos*[*OF mp*, **where** *t*=*t*]

have (*abs c* > *1* ∧ *?g* > *1*) ∨ (*abs c* = *0* ∧ *?g* > *1*) ∨ (*abs c* > *1* ∧ *?g* = *0*)
by *simp*
moreover {**assume** *abs c* > *1* **and** *gp*: *?g* > *1* **with** *prems*
have *th*: *dvdnumcoeff t ?g* **by** *simp*
have *th'*: *igcd c ?g dvd ?g* **by** (*simp add*:*igcd-dvd2*)
from *dvdnumcoeff-trans*[*OF th' th*] **have** *?case* **by** (*simp add*: *igcd-dvd1*)}
moreover {**assume** *abs c* = *0* ∧ *?g* > *1*
with *prems* **have** *th*: *dvdnumcoeff t ?g* **by** *simp*
have *th'*: *igcd c ?g dvd ?g* **by** (*simp add*:*igcd-dvd2*)
from *dvdnumcoeff-trans*[*OF th' th*] **have** *?case* **by** (*simp add*: *igcd-dvd1*)
hence *?case* **by** *simp* }
moreover {**assume** *abs c* > *1* **and** *g0*:*?g* = *0*
from *numgcdh0*[*OF g0*] **have** *m*=*0*. **with** *prems* **have** *?case* **by** *simp* }
ultimately show *?case* **by** *blast*
qed(*auto simp add*: *igcd-dvd1*)

lemma *dvdnumcoeff-aux2*:
assumes *numgcd t* > *1* **shows** *dvdnumcoeff t* (*numgcd t*) ∧ *numgcd t* > *0*
using *prems*
proof (*simp add*: *numgcd-def*)
let *?mc* = *maxcoeff t*
let *?g* = *numgcdh t ?mc*
have *th1*: *ismaxcoeff t ?mc* **by** (*rule maxcoeff-ismaxcoeff*)
have *th2*: *?mc* ≥ *0* **by** (*rule maxcoeff-pos*)
assume *H*: *numgcdh t ?mc* > *1*
from *dvdnumcoeff-aux*[*OF th1 th2 H*] **show** *dvdnumcoeff t ?g* .
qed

lemma *reducecoeff*: *real* (*numgcd t*) ∗ (*Inum bs* (*reducecoeff t*)) = *Inum bs t*
proof−
let *?g* = *numgcd t*
have *?g* ≥ *0* **by** (*simp add*: *numgcd-pos*)
hence *?g* = *0* ∨ *?g* = *1* ∨ *?g* > *1* **by** *auto*
moreover {**assume** *?g* = *0* **hence** *?thesis* **by** (*simp add*: *numgcd0*)}
moreover {**assume** *?g* = *1* **hence** *?thesis* **by** (*simp add*: *reducecoeff-def*)}
moreover { **assume** *g1*:*?g* > *1*
from *dvdnumcoeff-aux2*[*OF g1*] **have** *th1*:*dvdnumcoeff t ?g* **and** *g0*: *?g* > *0* **by** *blast*+
from *reducecoeffh*[*OF th1 g0*, **where** *bs*=*bs*] *g1* **have** *?thesis*
by (*simp add*: *reducecoeff-def Let-def*)}
ultimately show *?thesis* **by** *blast*
qed

lemma *reducecoeffh-numbound0*: *numbound0 t* ⟹ *numbound0* (*reducecoeffh t g*)
by (*induct t rule*: *reducecoeffh.induct*, *auto*)

lemma *reducecoeff-numbound0*: *numbound0 t* ⟹ *numbound0* (*reducecoeff t*)
using *reducecoeffh-numbound0* **by** (*simp add*: *reducecoeff-def Let-def*)

consts
simpnum:: *num* ⇒ *num*
numadd:: *num* × *num* ⇒ *num*
nummul:: *num* ⇒ *int* ⇒ *num*
recdef *numadd measure* (λ (*t*,*s*). *size t* + *size s*)
numadd (*CN n1 c1 r1*,*CN n2 c2 r2*) =
(*if n1*=*n2 then*
(*let c* = *c1* + *c2*
in (*if c*=*0 then numadd*(*r1*,*r2*) *else CN n1 c* (*numadd* (*r1*,*r2*))))
else if n1 ≤ *n2 then* (*CN n1 c1* (*numadd* (*r1*,*CN n2 c2 r2*)))
else (*CN n2 c2* (*numadd* (*CN n1 c1 r1*,*r2*))))
numadd (*CN n1 c1 r1*,*t*) = *CN n1 c1* (*numadd* (*r1*, *t*))
numadd (*t*,*CN n2 c2 r2*) = *CN n2 c2* (*numadd* (*t*,*r2*))
numadd (*C b1*, *C b2*) = *C* (*b1*+*b2*)
numadd (*a*,*b*) = *Add a b*

lemma *numadd*[*simp*]: *Inum bs* (*numadd* (*t*,*s*)) = *Inum bs* (*Add t s*)
apply (*induct t s rule*: *numadd.induct*, *simp-all add*: *Let-def*)
apply (*case-tac c1*+*c2* = *0*,*case-tac n1* ≤ *n2*, *simp-all*)
apply (*case-tac n1* = *n2*, *simp-all add*: *ring-simps*)
by (*simp only*: *left-distrib*[*symmetric*],*simp*)

lemma *numadd-nb*[*simp*]: ⟦ *numbound0 t* ; *numbound0 s*⟧ ⟹ *numbound0* (*numadd* (*t*,*s*))
by (*induct t s rule*: *numadd.induct*, *auto simp add*: *Let-def*)

recdef *nummul measure size*
nummul (*C j*) = (λ *i*. *C* (*i*∗*j*))
nummul (*CN n c a*) = (λ *i*. *CN n* (*i*∗*c*) (*nummul a i*))
nummul t = (λ *i*. *Mul i t*)

lemma *nummul*[*simp*]: ⋀ *i*. *Inum bs* (*nummul t i*) = *Inum bs* (*Mul i t*)
by (*induct t rule*: *nummul.induct*, *auto simp add*: *ring-simps*)

lemma *nummul-nb*[*simp*]: ⋀ *i*. *numbound0 t* ⟹ *numbound0* (*nummul t i*)
by (*induct t rule*: *nummul.induct*, *auto*)

constdefs *numneg* :: *num* ⇒ *num*
numneg t ≡ *nummul t* (− *1*)

constdefs *numsub* :: *num* ⇒ *num* ⇒ *num*
numsub s t ≡ (*if s* = *t then C 0 else numadd* (*s*,*numneg t*))

lemma *numneg*[*simp*]: *Inum bs* (*numneg t*) = *Inum bs* (*Neg t*)
using *numneg-def* **by** *simp*

lemma *numneg-nb*[*simp*]: *numbound0 t* ⟹ *numbound0* (*numneg t*)
using *numneg-def* **by** *simp*

lemma *numsub*[*simp*]: *Inum bs* (*numsub a b*) = *Inum bs* (*Sub a b*)
using *numsub-def* **by** *simp*

lemma *numsub-nb*[*simp*]: ⟦ *numbound0 t* ; *numbound0 s*⟧ ⟹ *numbound0* (*numsub t s*)
using *numsub-def* **by** *simp*

recdef *simpnum measure size*
  simpnum (*C j*) = *C j*
  simpnum (*Bound n*) = *CN n 1* (*C 0*)
  simpnum (*Neg t*) = *numneg* (*simpnum t*)
  simpnum (*Add t s*) = *numadd* (*simpnum t*,*simpnum s*)
  simpnum (*Sub t s*) = *numsub* (*simpnum t*) (*simpnum s*)
  simpnum (*Mul i t*) = (*if i* = *0 then* (*C 0*) *else nummul* (*simpnum t*) *i*)
  simpnum (*CN n c t*) = (*if c* = *0 then simpnum t else numadd* (*CN n c* (*C 0*),*simpnum t*))

lemma *simpnum-ci*[*simp*]: *Inum bs* (*simpnum t*) = *Inum bs t*
by (*induct t rule*: *simpnum.induct*, *auto simp add*: *numneg numadd numsub nummul*)

lemma *simpnum-numbound0*[*simp*]:
  numbound0 t ⟹ *numbound0* (*simpnum t*)
by (*induct t rule*: *simpnum.induct*, *auto*)

consts *nozerocoeff*:: *num* ⇒ *bool*
recdef *nozerocoeff measure size*
  nozerocoeff (*C c*) = *True*
  nozerocoeff (*CN n c t*) = (*c*≠*0* ∧ *nozerocoeff t*)
  nozerocoeff t = *True*

lemma *numadd-nz* : *nozerocoeff a* ⟹ *nozerocoeff b* ⟹ *nozerocoeff* (*numadd* (*a*,*b*))
by (*induct a b rule*: *numadd.induct*,*auto simp add*: *Let-def*)

lemma *nummul-nz* : ⋀ *i*. *i*≠*0* ⟹ *nozerocoeff a* ⟹ *nozerocoeff* (*nummul a i*)
by (*induct a rule*: *nummul.induct*,*auto simp add*: *Let-def numadd-nz*)

lemma *numneg-nz* : *nozerocoeff a* ⟹ *nozerocoeff* (*numneg a*)
by (*simp add*: *numneg-def nummul-nz*)

lemma *numsub-nz*: *nozerocoeff a* ⟹ *nozerocoeff b* ⟹ *nozerocoeff* (*numsub a b*)
by (*simp add*: *numsub-def numneg-nz numadd-nz*)

lemma *simpnum-nz*: *nozerocoeff* (*simpnum t*)
by(*induct t rule*: *simpnum.induct*, *auto simp add*: *numadd-nz numneg-nz numsub-nz nummul-nz*)

lemma *maxcoeff-nz*: *nozerocoeff t* ⟹ *maxcoeff t* = *0* ⟹ *t* = *C 0*

proof (*induct t rule*: *maxcoeff.induct*)
  case (*2 n c t*)
  hence *cnz*: $c \neq 0$ **and** *mx*: *max* (*abs c*) (*maxcoeff t*) = *0* **by** *simp+*
  have *max* (*abs c*) (*maxcoeff t*) $\geq$ *abs c* **by** (*simp add*: *le-maxI1*)
  with *cnz* **have** *max* (*abs c*) (*maxcoeff t*) > *0* **by** *arith*
  with *prems* **show** *?case* **by** *simp*
qed *auto*

lemma *numgcd-nz*: **assumes** *nz*: *nozerocoeff t* **and** *g0*: *numgcd t* = *0* **shows** *t* = *C 0*
proof−
  from *g0* **have** *th*:*numgcdh t* (*maxcoeff t*) = *0* **by** (*simp add*: *numgcd-def*)
  from *numgcdh0*[*OF th*] **have** *th*:*maxcoeff t* = *0* **.**
  from *maxcoeff-nz*[*OF nz th*] **show** *?thesis* **.**
qed

constdefs *simp-num-pair*:: (*num* × *int*) ⇒ *num* × *int*
  simp-num-pair ≡ (λ (*t*,*n*). (*if n* = *0 then* (*C 0*, *0*) *else*
   (*let t′* = *simpnum t* ; *g* = *numgcd t′ in*
     if g > *1 then* (*let g′* = *igcd n g in*
       if g′ = *1 then* (*t′*,*n*)
       else (*reducecoeffh t′ g′*, *n div g′*))
     else (*t′*,*n*))))

lemma *simp-num-pair-ci*:
  shows ((λ (*t*,*n*). *Inum bs t* / *real n*) (*simp-num-pair* (*t*,*n*))) = ((λ (*t*,*n*). *Inum bs t* / *real n*) (*t*,*n*))
  (**is** *?lhs* = *?rhs*)
proof−
  let *?t′* = *simpnum t*
  let *?g* = *numgcd ?t′*
  let *?g′* = *igcd n ?g*
  {**assume** *nz*: *n* = *0* **hence** *?thesis* **by** (*simp add*: *Let-def simp-num-pair-def*)}
  moreover
  { **assume** *nnz*: $n \neq 0$
    {**assume** ¬ *?g* > *1* **hence** *?thesis* **by** (*simp add*: *Let-def simp-num-pair-def simpnum-ci*)}
    moreover
    {**assume** *g1*:*?g*>*1* **hence** *g0*: *?g* > *0* **by** *simp*
      from *igcd0 g1 nnz* **have** *gp0*: *?g′* ≠ *0* **by** *simp*
      hence *g′p*: *?g′* > *0* **using** *igcd-pos*[**where** *i*=*n* **and** *j*=*numgcd ?t′*] **by** *arith*
      hence *?g′*= *1* ∨ *?g′* > *1* **by** *arith*
    moreover {**assume** *?g′*=*1* **hence** *?thesis* **by** (*simp add*: *Let-def simp-num-pair-def simpnum-ci*)}
      moreover {**assume** *g′1*:*?g′*>*1*
        from *dvdnumcoeff-aux2*[*OF g1*] **have** *th1*:*dvdnumcoeff ?t′ ?g* **..**
        let *?tt* = *reducecoeffh ?t′ ?g′*
        let *?t* = *Inum bs ?tt*
        have *gpdg*: *?g′ dvd ?g* **by** (*simp add*: *igcd-dvd2*)

have *gpdd*: *?g′ dvd n* **by** (*simp add*: *igcd-dvd1*)
have *gpdgp*: *?g′ dvd ?g′* **by** *simp*
from *reducecoeffh*[*OF dvdnumcoeff-trans*[*OF gpdg th1*] *g′p*]
have *th2*:*real ?g′ ∗ ?t = Inum bs ?t′* **by** *simp*
from *prems* **have** *?lhs = ?t / real* (*n div ?g′*) **by** (*simp add*: *simp-num-pair-def Let-def*)
also have ... = (*real ?g′ ∗ ?t*) / (*real ?g′ ∗* (*real* (*n div ?g′*))) **by** *simp*
also have ... = (*Inum bs ?t′ / real n*)
using *real-of-int-div*[*OF gp0 gpdd*] *th2 gp0* **by** *simp*
finally have *?lhs = Inum bs t / real n* **by** (*simp add*: *simpnum-ci*)
then have *?thesis* **using** *prems* **by** (*simp add*: *simp-num-pair-def*)}
ultimately have *?thesis* **by** *blast*}
ultimately have *?thesis* **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

lemma *simp-num-pair-l*: **assumes** *tnb*: *numbound0 t* **and** *np*: *n >0* **and** *tn*: *simp-num-pair* (*t*,*n*) = (*t′*,*n′*)
shows *numbound0 t′ ∧ n′ >0*
proof−
let *?t′ = simpnum t*
let *?g = numgcd ?t′*
let *?g′ = igcd n ?g*
{**assume** *nz*: *n = 0* **hence** *?thesis* **using** *prems* **by** (*simp add*: *Let-def simp-num-pair-def*)}
moreover
{ **assume** *nnz*: *n ≠ 0*
{**assume** ¬ *?g > 1* **hence** *?thesis* **using** *prems* **by** (*auto simp add*: *Let-def simp-num-pair-def simpnum-numbound0*)}
moreover
{**assume** *g1*:*?g>1* **hence** *g0*: *?g > 0* **by** *simp*
from *igcd0 g1 nnz* **have** *gp0*: *?g′ ≠ 0* **by** *simp*
hence *g′p*: *?g′ > 0* **using** *igcd-pos*[**where** *i=n* **and** *j=numgcd ?t′*] **by** *arith*
hence *?g′= 1 ∨ ?g′ > 1* **by** *arith*
moreover {**assume** *?g′=1* **hence** *?thesis* **using** *prems*
by (*auto simp add*: *Let-def simp-num-pair-def simpnum-numbound0*)}
moreover {**assume** *g′1*:*?g′>1*
have *gpdg*: *?g′ dvd ?g* **by** (*simp add*: *igcd-dvd2*)
have *gpdd*: *?g′ dvd n* **by** (*simp add*: *igcd-dvd1*)
have *gpdgp*: *?g′ dvd ?g′* **by** *simp*
from *zdvd-imp-le*[*OF gpdd np*] **have** *g′n*: *?g′ ≤ n* **.**
from *zdiv-mono1*[*OF g′n g′p*, *simplified zdiv-self*[*OF gp0*]]
have *n div ?g′ >0* **by** *simp*
hence *?thesis* **using** *prems*
by(*auto simp add*: *simp-num-pair-def Let-def reducecoeffh-numbound0 simpnum-numbound0*)}
ultimately have *?thesis* **by** *blast*}
ultimately have *?thesis* **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

consts *simpfm* :: *fm* ⇒ *fm*
recdef *simpfm measure fmsize*
simpfm (*And p q*) = *conj* (*simpfm p*) (*simpfm q*)
simpfm (*Or p q*) = *disj* (*simpfm p*) (*simpfm q*)
simpfm (*Imp p q*) = *imp* (*simpfm p*) (*simpfm q*)
simpfm (*Iff p q*) = *iff* (*simpfm p*) (*simpfm q*)
simpfm (*NOT p*) = *not* (*simpfm p*)
simpfm (*Lt a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* (*v* < *0*) *then T else F*
| - ⇒ *Lt* a')
simpfm (*Le a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* (*v* ≤ *0*) *then T else F* | - ⇒ *Le* a')
simpfm (*Gt a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* (*v* > *0*) *then T else F* | - ⇒ *Gt* a')
simpfm (*Ge a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* (*v* ≥ *0*) *then T else F* | - ⇒ *Ge* a')
simpfm (*Eq a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* (*v* = *0*) *then T else F* | - ⇒ *Eq* a')
simpfm (*NEq a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* (*v* ≠ *0*) *then T else F* | - ⇒ *NEq* a')
simpfm p = *p*
lemma *simpfm*: *Ifm bs* (*simpfm p*) = *Ifm bs p*
proof(*induct p rule*: *simpfm.induct*)
case (*6 a*) **let** *?sa* = *simpnum a* **from** *simpnum-ci* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
{**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
moreover {**assume** ¬ (∃ *v*. *?sa* = *C v*) **hence** *?case* **using** *sa*
by (*cases ?sa*, *simp-all add*: *Let-def*)}
ultimately show *?case* **by** *blast*
next
case (*7 a*) **let** *?sa* = *simpnum a*
from *simpnum-ci* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
{**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
moreover {**assume** ¬ (∃ *v*. *?sa* = *C v*) **hence** *?case* **using** *sa*
by (*cases ?sa*, *simp-all add*: *Let-def*)}
ultimately show *?case* **by** *blast*
next
case (*8 a*) **let** *?sa* = *simpnum a*
from *simpnum-ci* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
{**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
moreover {**assume** ¬ (∃ *v*. *?sa* = *C v*) **hence** *?case* **using** *sa*
by (*cases ?sa*, *simp-all add*: *Let-def*)}
ultimately show *?case* **by** *blast*
next
case (*9 a*) **let** *?sa* = *simpnum a*
from *simpnum-ci* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
{**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
moreover {**assume** ¬ (∃ *v*. *?sa* = *C v*) **hence** *?case* **using** *sa*

    by (*cases ?sa*, *simp-all add*: *Let-def*)}
  ultimately show *?case* **by** *blast*
next
  case (*10 a*) **let** *?sa* = *simpnum a*
  from *simpnum-ci* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
  {**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
  moreover {**assume** ¬ (∃ *v*. *?sa* = *C v*) **hence** *?case* **using** *sa*
    by (*cases ?sa*, *simp-all add*: *Let-def*)}
  ultimately show *?case* **by** *blast*
next
  case (*11 a*) **let** *?sa* = *simpnum a*
  from *simpnum-ci* **have** *sa*: *Inum bs ?sa* = *Inum bs a* **by** *simp*
  {**fix** *v* **assume** *?sa* = *C v* **hence** *?case* **using** *sa* **by** *simp* }
  moreover {**assume** ¬ (∃ *v*. *?sa* = *C v*) **hence** *?case* **using** *sa*
    by (*cases ?sa*, *simp-all add*: *Let-def*)}
  ultimately show *?case* **by** *blast*
qed (*induct p rule*: *simpfm.induct*, *simp-all add*: *conj disj imp iff not*)

lemma *simpfm-bound0*: *bound0 p* ⟹ *bound0* (*simpfm p*)
proof(*induct p rule*: *simpfm.induct*)
  case (*6 a*) **hence** *nb*: *numbound0 a* **by** *simp*
  hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
  thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
next
  case (*7 a*) **hence** *nb*: *numbound0 a* **by** *simp*
  hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
  thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
next
  case (*8 a*) **hence** *nb*: *numbound0 a* **by** *simp*
  hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
  thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
next
  case (*9 a*) **hence** *nb*: *numbound0 a* **by** *simp*
  hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
  thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
next
  case (*10 a*) **hence** *nb*: *numbound0 a* **by** *simp*
  hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
  thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
next
  case (*11 a*) **hence** *nb*: *numbound0 a* **by** *simp*
  hence *numbound0* (*simpnum a*) **by** (*simp only*: *simpnum-numbound0*[*OF nb*])
  thus *?case* **by** (*cases simpnum a*, *auto simp add*: *Let-def*)
qed(*auto simp add*: *disj-def imp-def iff-def conj-def not-bn*)

lemma *simpfm-qf*: *qfree p* ⟹ *qfree* (*simpfm p*)
by (*induct p rule*: *simpfm.induct*, *auto simp add*: *disj-qf imp-qf iff-qf conj-qf not-qf*
Let-def)

(*case-tac simpnum a*,*auto*)+

consts *prep* :: *fm* ⇒ *fm*
recdef *prep measure fmsize*
prep (*E T*) = *T*
prep (*E F*) = *F*
prep (*E* (*Or p q*)) = *disj* (*prep* (*E p*)) (*prep* (*E q*))
prep (*E* (*Imp p q*)) = *disj* (*prep* (*E* (*NOT p*))) (*prep* (*E q*))
prep (*E* (*Iff p q*)) = *disj* (*prep* (*E* (*And p q*))) (*prep* (*E* (*And* (*NOT p*) (*NOT q*))))
prep (*E* (*NOT* (*And p q*))) = *disj* (*prep* (*E* (*NOT p*))) (*prep* (*E*(*NOT q*)))
prep (*E* (*NOT* (*Imp p q*))) = *prep* (*E* (*And p* (*NOT q*)))
prep (*E* (*NOT* (*Iff p q*))) = *disj* (*prep* (*E* (*And p* (*NOT q*)))) (*prep* (*E*(*And* (*NOT p*) *q*)))
prep (*E p*) = *E* (*prep p*)
prep (*A* (*And p q*)) = *conj* (*prep* (*A p*)) (*prep* (*A q*))
prep (*A p*) = *prep* (*NOT* (*E* (*NOT p*)))
prep (*NOT* (*NOT p*)) = *prep p*
prep (*NOT* (*And p q*)) = *disj* (*prep* (*NOT p*)) (*prep* (*NOT q*))
prep (*NOT* (*A p*)) = *prep* (*E* (*NOT p*))
prep (*NOT* (*Or p q*)) = *conj* (*prep* (*NOT p*)) (*prep* (*NOT q*))
prep (*NOT* (*Imp p q*)) = *conj* (*prep p*) (*prep* (*NOT q*))
prep (*NOT* (*Iff p q*)) = *disj* (*prep* (*And p* (*NOT q*))) (*prep* (*And* (*NOT p*) *q*))
prep (*NOT p*) = *not* (*prep p*)
prep (*Or p q*) = *disj* (*prep p*) (*prep q*)
prep (*And p q*) = *conj* (*prep p*) (*prep q*)
prep (*Imp p q*) = *prep* (*Or* (*NOT p*) *q*)
prep (*Iff p q*) = *disj* (*prep* (*And p q*)) (*prep* (*And* (*NOT p*) (*NOT q*)))
prep p = *p*
(**hints** *simp add*: *fmsize-pos*)
lemma *prep*: ⋀ *bs*. *Ifm bs* (*prep p*) = *Ifm bs p*
by (*induct p rule*: *prep.induct*, *auto*)

consts *qelim* :: *fm* ⇒ (*fm* ⇒ *fm*) ⇒ *fm*
recdef *qelim measure fmsize*
qelim (*E p*) = (λ *qe*. *DJ qe* (*qelim p qe*))
qelim (*A p*) = (λ *qe*. *not* (*qe* ((*qelim* (*NOT p*) *qe*))))
qelim (*NOT p*) = (λ *qe*. *not* (*qelim p qe*))
qelim (*And p q*) = (λ *qe*. *conj* (*qelim p qe*) (*qelim q qe*))
qelim (*Or p q*) = (λ *qe*. *disj* (*qelim p qe*) (*qelim q qe*))
qelim (*Imp p q*) = (λ *qe*. *imp* (*qelim p qe*) (*qelim q qe*))
qelim (*Iff p q*) = (λ *qe*. *iff* (*qelim p qe*) (*qelim q qe*))
qelim p = (λ *y*. *simpfm p*)

lemma *qelim-ci*:
assumes *qe-inv*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ⋀ *bs*. *qfree* (*qelim p qe*) ∧ (*Ifm bs* (*qelim p qe*) = *Ifm bs p*)

using *qe-inv DJ-qe*[*OF qe-inv*]
by(*induct p rule*: *qelim.induct*)
(*auto simp add*: *not disj conj iff imp not-qf disj-qf conj-qf imp-qf iff-qf*
simpfm simpfm-qf simp del: *simpfm.simps*)

consts
plusinf:: *fm* ⇒ *fm*
minusinf:: *fm* ⇒ *fm*
recdef *minusinf measure size*
minusinf (*And p q*) = *conj* (*minusinf p*) (*minusinf q*)
minusinf (*Or p q*) = *disj* (*minusinf p*) (*minusinf q*)
minusinf (*Eq* (*CN 0 c e*)) = *F*
minusinf (*NEq* (*CN 0 c e*)) = *T*
minusinf (*Lt* (*CN 0 c e*)) = *T*
minusinf (*Le* (*CN 0 c e*)) = *T*
minusinf (*Gt* (*CN 0 c e*)) = *F*
minusinf (*Ge* (*CN 0 c e*)) = *F*
minusinf p = *p*

recdef *plusinf measure size*
plusinf (*And p q*) = *conj* (*plusinf p*) (*plusinf q*)
plusinf (*Or p q*) = *disj* (*plusinf p*) (*plusinf q*)
plusinf (*Eq* (*CN 0 c e*)) = *F*
plusinf (*NEq* (*CN 0 c e*)) = *T*
plusinf (*Lt* (*CN 0 c e*)) = *F*
plusinf (*Le* (*CN 0 c e*)) = *F*
plusinf (*Gt* (*CN 0 c e*)) = *T*
plusinf (*Ge* (*CN 0 c e*)) = *T*
plusinf p = *p*

consts
isrlfm :: *fm* ⇒ *bool*
recdef *isrlfm measure size*
isrlfm (*And p q*) = (*isrlfm p* ∧ *isrlfm q*)
isrlfm (*Or p q*) = (*isrlfm p* ∧ *isrlfm q*)
isrlfm (*Eq* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*NEq* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*Lt* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*Le* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*Gt* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*Ge* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm p = (*isatom p* ∧ (*bound0 p*))

consts *rsplit0* :: *num* ⇒ *int* × *num*
recdef *rsplit0 measure num-size*
rsplit0 (*Bound 0*) = (*1*,*C 0*)
rsplit0 (*Add a b*) = (*let* (*ca*,*ta*) = *rsplit0 a* ; (*cb*,*tb*) = *rsplit0 b*
in (*ca*+*cb*, *Add ta tb*))

rsplit0 (*Sub a b*) = *rsplit0* (*Add a* (*Neg b*))
rsplit0 (*Neg a*) = (*let* (*c*,*t*) = *rsplit0 a in* (−*c*,*Neg t*))
rsplit0 (*Mul c a*) = (*let* (*ca*,*ta*) = *rsplit0 a in* (*c*∗*ca*,*Mul c ta*))
rsplit0 (*CN 0 c a*) = (*let* (*ca*,*ta*) = *rsplit0 a in* (*c*+*ca*,*ta*))
rsplit0 (*CN n c a*) = (*let* (*ca*,*ta*) = *rsplit0 a in* (*ca*,*CN n c ta*))
rsplit0 t = (*0*,*t*)
lemma *rsplit0*:
shows *Inum bs* ((*split* (*CN 0*)) (*rsplit0 t*)) = *Inum bs t* ∧ *numbound0* (*snd* (*rsplit0 t*))
proof (*induct t rule*: *rsplit0.induct*)
case (*2 a b*)
let *?sa* = *rsplit0 a* **let** *?sb* = *rsplit0 b*
let *?ca* = *fst ?sa* **let** *?cb* = *fst ?sb*
let *?ta* = *snd ?sa* **let** *?tb* = *snd ?sb*
from *prems* **have** *nb*: *numbound0* (*snd*(*rsplit0* (*Add a b*)))
by(*cases rsplit0 a*,*auto simp add*: *Let-def split-def*)
have *Inum bs* ((*split* (*CN 0*)) (*rsplit0* (*Add a b*))) =
Inum bs ((*split* (*CN 0*)) *?sa*)+*Inum bs* ((*split* (*CN 0*)) *?sb*)
by (*simp add*: *Let-def split-def ring-simps*)
also have ... = *Inum bs a* + *Inum bs b* **using** *prems* **by** (*cases rsplit0 a*, *simp-all*)
finally show *?case* **using** *nb* **by** *simp*
qed(*auto simp add*: *Let-def split-def ring-simps* , *simp add*: *right-distrib*[*symmetric*])

definition
lt :: *int* ⇒ *num* ⇒ *fm*
where
lt c t = (*if c* = *0 then* (*Lt t*) *else if c* > *0 then* (*Lt* (*CN 0 c t*))
else (*Gt* (*CN 0* (−*c*) (*Neg t*))))

definition
le :: *int* ⇒ *num* ⇒ *fm*
where
le c t = (*if c* = *0 then* (*Le t*) *else if c* > *0 then* (*Le* (*CN 0 c t*))
else (*Ge* (*CN 0* (−*c*) (*Neg t*))))

definition
gt :: *int* ⇒ *num* ⇒ *fm*
where
gt c t = (*if c* = *0 then* (*Gt t*) *else if c* > *0 then* (*Gt* (*CN 0 c t*))
else (*Lt* (*CN 0* (−*c*) (*Neg t*))))

definition
ge :: *int* ⇒ *num* ⇒ *fm*
where
ge c t = (*if c* = *0 then* (*Ge t*) *else if c* > *0 then* (*Ge* (*CN 0 c t*))
else (*Le* (*CN 0* (−*c*) (*Neg t*))))

definition
  eq :: *int* ⇒ *num* ⇒ *fm*
where
  eq c t = (*if c* = *0 then* (*Eq t*) *else if c* > *0 then* (*Eq* (*CN 0 c t*))
    else (*Eq* (*CN 0* (−*c*) (*Neg t*))))

definition
  neq :: *int* ⇒ *num* ⇒ *fm*
where
  neq c t = (*if c* = *0 then* (*NEq t*) *else if c* > *0 then* (*NEq* (*CN 0 c t*))
    else (*NEq* (*CN 0* (−*c*) (*Neg t*))))

lemma *lt*: *numnoabs t* ⟹ *Ifm bs* (*split lt* (*rsplit0 t*)) = *Ifm bs* (*Lt t*) ∧ *isrlfm* (*split lt* (*rsplit0 t*))
using *rsplit0*[**where** *bs* = *bs* **and** *t*=*t*]
by (*auto simp add*: *lt-def split-def*,*cases snd*(*rsplit0 t*),*auto*,*case-tac nat*,*auto*)

lemma *le*: *numnoabs t* ⟹ *Ifm bs* (*split le* (*rsplit0 t*)) = *Ifm bs* (*Le t*) ∧ *isrlfm* (*split le* (*rsplit0 t*))
using *rsplit0*[**where** *bs* = *bs* **and** *t*=*t*]
by (*auto simp add*: *le-def split-def*) (*cases snd*(*rsplit0 t*),*auto*,*case-tac nat*,*auto*)

lemma *gt*: *numnoabs t* ⟹ *Ifm bs* (*split gt* (*rsplit0 t*)) = *Ifm bs* (*Gt t*) ∧ *isrlfm* (*split gt* (*rsplit0 t*))
using *rsplit0*[**where** *bs* = *bs* **and** *t*=*t*]
by (*auto simp add*: *gt-def split-def*) (*cases snd*(*rsplit0 t*),*auto*,*case-tac nat*,*auto*)

lemma *ge*: *numnoabs t* ⟹ *Ifm bs* (*split ge* (*rsplit0 t*)) = *Ifm bs* (*Ge t*) ∧ *isrlfm* (*split ge* (*rsplit0 t*))
using *rsplit0*[**where** *bs* = *bs* **and** *t*=*t*]
by (*auto simp add*: *ge-def split-def*) (*cases snd*(*rsplit0 t*),*auto*,*case-tac nat*,*auto*)

lemma *eq*: *numnoabs t* ⟹ *Ifm bs* (*split eq* (*rsplit0 t*)) = *Ifm bs* (*Eq t*) ∧ *isrlfm* (*split eq* (*rsplit0 t*))
using *rsplit0*[**where** *bs* = *bs* **and** *t*=*t*]
by (*auto simp add*: *eq-def split-def*) (*cases snd*(*rsplit0 t*),*auto*,*case-tac nat*,*auto*)

lemma *neq*: *numnoabs t* ⟹ *Ifm bs* (*split neq* (*rsplit0 t*)) = *Ifm bs* (*NEq t*) ∧ *isrlfm* (*split neq* (*rsplit0 t*))
using *rsplit0*[**where** *bs* = *bs* **and** *t*=*t*]
by (*auto simp add*: *neq-def split-def*) (*cases snd*(*rsplit0 t*),*auto*,*case-tac nat*,*auto*)

lemma *conj-lin*: *isrlfm p* ⟹ *isrlfm q* ⟹ *isrlfm* (*conj p q*)
by (*auto simp add*: *conj-def*)
lemma *disj-lin*: *isrlfm p* ⟹ *isrlfm q* ⟹ *isrlfm* (*disj p q*)
by (*auto simp add*: *disj-def*)

consts *rlfm* :: *fm* ⇒ *fm*
recdef *rlfm measure fmsize*

rlfm (*And p q*) = *conj* (*rlfm p*) (*rlfm q*)
rlfm (*Or p q*) = *disj* (*rlfm p*) (*rlfm q*)
rlfm (*Imp p q*) = *disj* (*rlfm* (*NOT p*)) (*rlfm q*)
rlfm (*Iff p q*) = *disj* (*conj* (*rlfm p*) (*rlfm q*)) (*conj* (*rlfm* (*NOT p*)) (*rlfm* (*NOT q*)))
rlfm (*Lt a*) = *split lt* (*rsplit0 a*)
rlfm (*Le a*) = *split le* (*rsplit0 a*)
rlfm (*Gt a*) = *split gt* (*rsplit0 a*)
rlfm (*Ge a*) = *split ge* (*rsplit0 a*)
rlfm (*Eq a*) = *split eq* (*rsplit0 a*)
rlfm (*NEq a*) = *split neq* (*rsplit0 a*)
rlfm (*NOT* (*And p q*)) = *disj* (*rlfm* (*NOT p*)) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Or p q*)) = *conj* (*rlfm* (*NOT p*)) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Imp p q*)) = *conj* (*rlfm p*) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Iff p q*)) = *disj* (*conj*(*rlfm p*) (*rlfm*(*NOT q*))) (*conj*(*rlfm*(*NOT p*)) (*rlfm q*))
rlfm (*NOT* (*NOT p*)) = *rlfm p*
rlfm (*NOT T*) = *F*
rlfm (*NOT F*) = *T*
rlfm (*NOT* (*Lt a*)) = *rlfm* (*Ge a*)
rlfm (*NOT* (*Le a*)) = *rlfm* (*Gt a*)
rlfm (*NOT* (*Gt a*)) = *rlfm* (*Le a*)
rlfm (*NOT* (*Ge a*)) = *rlfm* (*Lt a*)
rlfm (*NOT* (*Eq a*)) = *rlfm* (*NEq a*)
rlfm (*NOT* (*NEq a*)) = *rlfm* (*Eq a*)
rlfm p = *p* (**hints** *simp add*: *fmsize-pos*)

lemma *rlfm-I*:
assumes *qfp*: *qfree p*
shows (*Ifm bs* (*rlfm p*) = *Ifm bs p*) ∧ *isrlfm* (*rlfm p*)
using *qfp*
by (*induct p rule*: *rlfm.induct*, *auto simp add*: *lt le gt ge eq neq conj disj conj-lin disj-lin*)

lemma *rminusinf-inf*:
assumes *lp*: *isrlfm p*
shows ∃ *z*. ∀ *x* < *z*. *Ifm* (*x#bs*) (*minusinf p*) = *Ifm* (*x#bs*) *p* (**is** ∃ *z*. ∀ *x*. *?P z x p*)
using *lp*
proof (*induct p rule*: *minusinf.induct*)
case (*1 p q*) **thus** *?case* **by** (*auto*,*rule-tac x*= *min z za* **in** *exI*) *auto*
next
case (*2 p q*) **thus** *?case* **by** (*auto*,*rule-tac x*= *min z za* **in** *exI*) *auto*
next
case (*3 c e*)
from *prems* **have** *nb*: *numbound0 e* **by** *simp*
from *prems* **have** *cp*: *real c* > *0* **by** *simp*
let *?e*=*Inum* (*a#bs*) *e*

```
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    hence real c ∗ x + ?e ≠ 0 by simp
    with xz have ?P ?z x (Eq (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x < ?z. ?P ?z x (Eq (CN 0 c e)) by simp
  thus ?case by blast
next
  case (4 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    hence real c ∗ x + ?e ≠ 0 by simp
    with xz have ?P ?z x (NEq (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x < ?z. ?P ?z x (NEq (CN 0 c e)) by simp
  thus ?case by blast
next
  case (5 c e)
    from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x < ?z
    hence (real c ∗ x < − ?e)
      by (simp only: pos-less-divide-eq[OF cp, where a=x and b=− ?e] mult-ac)
    hence real c ∗ x + ?e < 0 by arith
    with xz have ?P ?z x (Lt (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a]  by simp }
  hence ∀ x < ?z. ?P ?z x (Lt (CN 0 c e)) by simp
  thus ?case by blast
next
  case (6 c e)
    from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
```

assume *xz*: *x* < *?z*
hence (*real c* ∗ *x* < − *?e*)
by (*simp only*: *pos-less-divide-eq*[*OF cp*, **where** *a*=*x* **and** *b*=− *?e*] *mult-ac*)
hence *real c* ∗ *x* + *?e* < *0* **by** *arith*
with *xz* **have** *?P ?z x* (*Le* (*CN 0 c e*))
using *numbound0-I*[*OF nb*, **where** *b*=*x* **and** *bs*=*bs* **and** *b'*=*a*] **by** *simp* **}**
hence ∀ *x* < *?z*. *?P ?z x* (*Le* (*CN 0 c e*)) **by** *simp*
thus *?case* **by** *blast*
next
case (*7 c e*)
from *prems* **have** *nb*: *numbound0 e* **by** *simp*
from *prems* **have** *cp*: *real c* > *0* **by** *simp*
let *?e*=*Inum* (*a*#*bs*) *e*
let *?z* = (− *?e*) / *real c*
{fix *x*
assume *xz*: *x* < *?z*
hence (*real c* ∗ *x* < − *?e*)
by (*simp only*: *pos-less-divide-eq*[*OF cp*, **where** *a*=*x* **and** *b*=− *?e*] *mult-ac*)
hence *real c* ∗ *x* + *?e* < *0* **by** *arith*
with *xz* **have** *?P ?z x* (*Gt* (*CN 0 c e*))
using *numbound0-I*[*OF nb*, **where** *b*=*x* **and** *bs*=*bs* **and** *b'*=*a*] **by** *simp* **}**
hence ∀ *x* < *?z*. *?P ?z x* (*Gt* (*CN 0 c e*)) **by** *simp*
thus *?case* **by** *blast*
next
case (*8 c e*)
from *prems* **have** *nb*: *numbound0 e* **by** *simp*
from *prems* **have** *cp*: *real c* > *0* **by** *simp*
let *?e*=*Inum* (*a*#*bs*) *e*
let *?z* = (− *?e*) / *real c*
{fix *x*
assume *xz*: *x* < *?z*
hence (*real c* ∗ *x* < − *?e*)
by (*simp only*: *pos-less-divide-eq*[*OF cp*, **where** *a*=*x* **and** *b*=− *?e*] *mult-ac*)
hence *real c* ∗ *x* + *?e* < *0* **by** *arith*
with *xz* **have** *?P ?z x* (*Ge* (*CN 0 c e*))
using *numbound0-I*[*OF nb*, **where** *b*=*x* **and** *bs*=*bs* **and** *b'*=*a*] **by** *simp* **}**
hence ∀ *x* < *?z*. *?P ?z x* (*Ge* (*CN 0 c e*)) **by** *simp*
thus *?case* **by** *blast*
qed *simp-all*

lemma *rplusinf-inf*:
assumes *lp*: *isrlfm p*
shows ∃ *z*. ∀ *x* > *z*. *Ifm* (*x*#*bs*) (*plusinf p*) = *Ifm* (*x*#*bs*) *p* (**is** ∃ *z*. ∀ *x*. *?P z x p*)
using *lp*
proof (*induct p rule*: *isrlfm.induct*)
case (*1 p q*) **thus** *?case* **by** (*auto*,*rule-tac x*= *max z za* **in** *exI*) *auto*
next
case (*2 p q*) **thus** *?case* **by** (*auto*,*rule-tac x*= *max z za* **in** *exI*) *auto*

```
next
  case (3 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c ∗ x > − ?e) by (simp add: mult-ac)
    hence real c ∗ x + ?e > 0 by arith
    hence real c ∗ x + ?e ≠ 0 by simp
    with xz have ?P ?z x (Eq (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x > ?z. ?P ?z x (Eq (CN 0 c e)) by simp
  thus ?case by blast
next
  case (4 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c ∗ x > − ?e) by (simp add: mult-ac)
    hence real c ∗ x + ?e > 0 by arith
    hence real c ∗ x + ?e ≠ 0 by simp
    with xz have ?P ?z x (NEq (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x > ?z. ?P ?z x (NEq (CN 0 c e)) by simp
  thus ?case by blast
next
  case (5 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (− ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c ∗ x > − ?e) by (simp add: mult-ac)
    hence real c ∗ x + ?e > 0 by arith
    with xz have ?P ?z x (Lt (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b′=a] by simp }
  hence ∀ x > ?z. ?P ?z x (Lt (CN 0 c e)) by simp
  thus ?case by blast
next
  case (6 c e)
```

```
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (- ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c * x > - ?e) by (simp add: mult-ac)
    hence real c * x + ?e > 0 by arith
    with xz have ?P ?z x (Le (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b'=a] by simp }
  hence ∀ x > ?z. ?P ?z x (Le (CN 0 c e)) by simp
  thus ?case by blast
next
  case (7 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (- ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c * x > - ?e) by (simp add: mult-ac)
    hence real c * x + ?e > 0 by arith
    with xz have ?P ?z x (Gt (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b'=a] by simp }
  hence ∀ x > ?z. ?P ?z x (Gt (CN 0 c e)) by simp
  thus ?case by blast
next
  case (8 c e)
  from prems have nb: numbound0 e by simp
  from prems have cp: real c > 0 by simp
  let ?e=Inum (a#bs) e
  let ?z = (- ?e) / real c
  {fix x
    assume xz: x > ?z
    with mult-strict-right-mono [OF xz cp] cp
    have (real c * x > - ?e) by (simp add: mult-ac)
    hence real c * x + ?e > 0 by arith
    with xz have ?P ?z x (Ge (CN 0 c e))
      using numbound0-I[OF nb, where b=x and bs=bs and b'=a]  by simp }
  hence ∀ x > ?z. ?P ?z x (Ge (CN 0 c e)) by simp
  thus ?case by blast
qed simp-all

lemma rminusinf-bound0:
  assumes lp: isrlfm p
  shows bound0 (minusinf p)
  using lp
```

```
  by (induct p rule: minusinf.induct) simp-all

lemma rplusinf-bound0:
  assumes lp: isrlfm p
  shows bound0 (plusinf p)
  using lp
  by (induct p rule: plusinf.induct) simp-all

lemma rminusinf-ex:
  assumes lp: isrlfm p
  and ex: Ifm (a#bs) (minusinf p)
  shows ∃ x. Ifm (x#bs) p
proof−
  from bound0-I [OF rminusinf-bound0[OF lp], where b=a and bs =bs] ex
  have th: ∀ x. Ifm (x#bs) (minusinf p) by auto
  from rminusinf-inf[OF lp, where bs=bs]
  obtain z where z-def: ∀x<z. Ifm (x # bs) (minusinf p) = Ifm (x # bs) p by
blast
  from th have Ifm ((z − 1)#bs) (minusinf p) by simp
  moreover have z − 1 < z by simp
  ultimately show ?thesis using z-def by auto
qed

lemma rplusinf-ex:
  assumes lp: isrlfm p
  and ex: Ifm (a#bs) (plusinf p)
  shows ∃ x. Ifm (x#bs) p
proof−
  from bound0-I [OF rplusinf-bound0[OF lp], where b=a and bs =bs] ex
  have th: ∀ x. Ifm (x#bs) (plusinf p) by auto
  from rplusinf-inf[OF lp, where bs=bs]
  obtain z where z-def: ∀x>z. Ifm (x # bs) (plusinf p) = Ifm (x # bs) p by
blast
  from th have Ifm ((z + 1)#bs) (plusinf p) by simp
  moreover have z + 1 > z by simp
  ultimately show ?thesis using z-def by auto
qed

consts
  uset:: fm ⇒ (num × int) list
  usubst :: fm ⇒ (num × int) ⇒ fm
recdef uset measure size
  uset (And p q) = (uset p @ uset q)
  uset (Or p q) = (uset p @ uset q)
  uset (Eq  (CN 0 c e)) = [(Neg e,c)]
  uset (NEq (CN 0 c e)) = [(Neg e,c)]
  uset (Lt  (CN 0 c e)) = [(Neg e,c)]
  uset (Le  (CN 0 c e)) = [(Neg e,c)]
  uset (Gt  (CN 0 c e)) = [(Neg e,c)]
```

uset (*Ge* (*CN 0 c e*)) = [(*Neg e*,*c*)]
uset p = []
recdef *usubst measure size*
usubst (*And p q*) = (λ (*t*,*n*). *And* (*usubst p* (*t*,*n*)) (*usubst q* (*t*,*n*)))
usubst (*Or p q*) = (λ (*t*,*n*). *Or* (*usubst p* (*t*,*n*)) (*usubst q* (*t*,*n*)))
usubst (*Eq* (*CN 0 c e*)) = (λ (*t*,*n*). *Eq* (*Add* (*Mul c t*) (*Mul n e*)))
usubst (*NEq* (*CN 0 c e*)) = (λ (*t*,*n*). *NEq* (*Add* (*Mul c t*) (*Mul n e*)))
usubst (*Lt* (*CN 0 c e*)) = (λ (*t*,*n*). *Lt* (*Add* (*Mul c t*) (*Mul n e*)))
usubst (*Le* (*CN 0 c e*)) = (λ (*t*,*n*). *Le* (*Add* (*Mul c t*) (*Mul n e*)))
usubst (*Gt* (*CN 0 c e*)) = (λ (*t*,*n*). *Gt* (*Add* (*Mul c t*) (*Mul n e*)))
usubst (*Ge* (*CN 0 c e*)) = (λ (*t*,*n*). *Ge* (*Add* (*Mul c t*) (*Mul n e*)))
usubst p = (λ (*t*,*n*). *p*)

lemma *usubst-I*: **assumes** *lp*: *isrlfm p*
and *np*: *real n* > *0* **and** *nbt*: *numbound0 t*
shows (*Ifm* (*x#bs*) (*usubst p* (*t*,*n*)) = *Ifm* (((*Inum* (*x#bs*) *t*)/(*real n*))#*bs*) *p*) ∧ *bound0* (*usubst p* (*t*,*n*)) (**is** (*?I x* (*usubst p* (*t*,*n*)) = *?I ?u p*) ∧ *?B p* **is** (- = *?I* (*?t*/*?n*) *p*) ∧ - **is** (- = *?I* (*?N x t* /-) *p*) ∧ -)
using *lp*
proof(*induct p rule*: *usubst.induct*)
case (*5 c e*) **from** *prems* **have** *cp*: *c* >*0* **and** *nb*: *numbound0 e* **by** *simp*+
have *?I ?u* (*Lt* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) < *0*)
using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
also have … = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) < *0*)
by (*simp only*: *pos-less-divide-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)

and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
also have … = (*real c* ∗*?t* + *?n*∗ (*?N x e*) < *0*)
using *np* **by** *simp*
finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
case (*6 c e*) **from** *prems* **have** *cp*: *c* >*0* **and** *nb*: *numbound0 e* **by** *simp*+
have *?I ?u* (*Le* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) ≤ *0*)
using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
also have … = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) ≤ *0*)
by (*simp only*: *pos-le-divide-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)
and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
also have … = (*real c* ∗*?t* + *?n*∗ (*?N x e*) ≤ *0*)
using *np* **by** *simp*
finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
case (*7 c e*) **from** *prems* **have** *cp*: *c* >*0* **and** *nb*: *numbound0 e* **by** *simp*+
have *?I ?u* (*Gt* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) > *0*)
using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b′*=*x*] **by** *simp*
also have … = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) > *0*)
by (*simp only*: *pos-divide-less-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)

and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
also have … = (*real c* ∗*?t* + *?n*∗ (*?N x e*) > *0*)

using *np* **by** *simp*
finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
case (*8 c e*) **from** *prems* **have** *cp*: *c >0* **and** *nb*: *numbound0 e* **by** *simp+*
have *?I ?u* (*Ge* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) ≥ *0*)
using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b'*=*x*] **by** *simp*
also have ... = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) ≥ *0*)
by (*simp only*: *pos-divide-le-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x e*)
and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
also have ... = (*real c* ∗*?t* + *?n*∗ (*?N x e*) ≥ *0*)
using *np* **by** *simp*
finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
case (*3 c e*) **from** *prems* **have** *cp*: *c >0* **and** *nb*: *numbound0 e* **by** *simp+*
from *np* **have** *np*: *real n* ≠ *0* **by** *simp*
have *?I ?u* (*Eq* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) = *0*)
using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b'*=*x*] **by** *simp*
also have ... = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) = *0*)
by (*simp only*: *nonzero-eq-divide-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x*
e)
and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
also have ... = (*real c* ∗*?t* + *?n*∗ (*?N x e*) = *0*)
using *np* **by** *simp*
finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
next
case (*4 c e*) **from** *prems* **have** *cp*: *c >0* **and** *nb*: *numbound0 e* **by** *simp+*
from *np* **have** *np*: *real n* ≠ *0* **by** *simp*
have *?I ?u* (*NEq* (*CN 0 c e*)) = (*real c* ∗(*?t*/*?n*) + (*?N x e*) ≠ *0*)
using *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*?u* **and** *b'*=*x*] **by** *simp*
also have ... = (*?n*∗(*real c* ∗(*?t*/*?n*)) + *?n*∗(*?N x e*) ≠ *0*)
by (*simp only*: *nonzero-eq-divide-eq*[*OF np*, **where** *a*=*real c* ∗(*?t*/*?n*) + (*?N x*
e)
and *b*=*0*, *simplified divide-zero-left*]) (*simp only*: *ring-simps*)
also have ... = (*real c* ∗*?t* + *?n*∗ (*?N x e*) ≠ *0*)
using *np* **by** *simp*
finally show *?case* **using** *nbt nb* **by** (*simp add*: *ring-simps*)
qed(*simp-all add*: *nbt numbound0-I*[**where** *bs* =*bs* **and** *b*=(*Inum* (*x*#*bs*) *t*)/ *real*
n **and** *b'*=*x*] *nth-pos2*)

lemma *uset-l*:
assumes *lp*: *isrlfm p*
shows ∀ (*t*,*k*) ∈ *set* (*uset p*). *numbound0 t* ∧ *k >0*
using *lp*
by(*induct p rule*: *uset.induct*,*auto*)

lemma *rminusinf-uset*:
assumes *lp*: *isrlfm p*
and *nmi*: ¬ (*Ifm* (*a*#*bs*) (*minusinf p*)) (**is** ¬ (*Ifm* (*a*#*bs*) (*?M p*)))
and *ex*: *Ifm* (*x*#*bs*) *p* (**is** *?I x p*)

shows ∃ (*s*,*m*) ∈ *set* (*uset p*). *x* ≥ *Inum* (*a*#*bs*) *s* / *real m* (**is** ∃ (*s*,*m*) ∈ *?U p*. *x* ≥ *?N a s* / *real m*)
proof−
have ∃ (*s*,*m*) ∈ *set* (*uset p*). *real m* ∗ *x* ≥ *Inum* (*a*#*bs*) *s* (**is** ∃ (*s*,*m*) ∈ *?U p*. *real m* ∗*x* ≥ *?N a s*)
using *lp nmi ex*
by (*induct p rule*: *minusinf.induct*, *auto simp add*:*numbound0-I*[**where** *bs*=*bs* **and** *b*=*a* **and** *b'*=*x*] *nth-pos2*)
then obtain *s m* **where** *smU*: (*s*,*m*) ∈ *set* (*uset p*) **and** *mx*: *real m* ∗ *x* ≥ *?N a s* **by** *blast*
from *uset-l*[*OF lp*] *smU* **have** *mp*: *real m* > *0* **by** *auto*
from *pos-divide-le-eq*[*OF mp*, **where** *a*=*x* **and** *b*=*?N a s*, *symmetric*] *mx* **have** *x* ≥ *?N a s* / *real m*
by (*auto simp add*: *mult-commute*)
thus *?thesis* **using** *smU* **by** *auto*
qed

lemma *rplusinf-uset*:
assumes *lp*: *isrlfm p*
and *nmi*: ¬ (*Ifm* (*a*#*bs*) (*plusinf p*)) (**is** ¬ (*Ifm* (*a*#*bs*) (*?M p*)))
and *ex*: *Ifm* (*x*#*bs*) *p* (**is** *?I x p*)
shows ∃ (*s*,*m*) ∈ *set* (*uset p*). *x* ≤ *Inum* (*a*#*bs*) *s* / *real m* (**is** ∃ (*s*,*m*) ∈ *?U p*. *x* ≤ *?N a s* / *real m*)
proof−
have ∃ (*s*,*m*) ∈ *set* (*uset p*). *real m* ∗ *x* ≤ *Inum* (*a*#*bs*) *s* (**is** ∃ (*s*,*m*) ∈ *?U p*. *real m* ∗*x* ≤ *?N a s*)
using *lp nmi ex*
by (*induct p rule*: *minusinf.induct*, *auto simp add*:*numbound0-I*[**where** *bs*=*bs* **and** *b*=*a* **and** *b'*=*x*] *nth-pos2*)
then obtain *s m* **where** *smU*: (*s*,*m*) ∈ *set* (*uset p*) **and** *mx*: *real m* ∗ *x* ≤ *?N a s* **by** *blast*
from *uset-l*[*OF lp*] *smU* **have** *mp*: *real m* > *0* **by** *auto*
from *pos-le-divide-eq*[*OF mp*, **where** *a*=*x* **and** *b*=*?N a s*, *symmetric*] *mx* **have** *x* ≤ *?N a s* / *real m*
by (*auto simp add*: *mult-commute*)
thus *?thesis* **using** *smU* **by** *auto*
qed

lemma *lin-dense*:
assumes *lp*: *isrlfm p*
and *noS*: ∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ (λ (*t*,*n*). *Inum* (*x*#*bs*) *t* / *real n*) ‘ *set* (*uset p*)
(**is** ∀ *t*. - ∧ - ⟶ *t* ∉ (λ (*t*,*n*). *?N x t* / *real n*) ‘ (*?U p*))
and *lx*: *l* < *x* **and** *xu*:*x* < *u* **and** *px*: *Ifm* (*x*#*bs*) *p*
and *ly*: *l* < *y* **and** *yu*: *y* < *u*
shows *Ifm* (*y*#*bs*) *p*
using *lp px noS*
proof (*induct p rule*: *isrlfm.induct*)
case (*5 c e*) **hence** *cp*: *real c* > *0* **and** *nb*: *numbound0 e* **by** *simp*+

from *prems* **have** $x * real\ c + ?N\ x\ e < 0$ **by** (*simp add*: *ring-simps*)
hence *pxc*: $x < (- ?N\ x\ e) / real\ c$
by (*simp only*: *pos-less-divide-eq*[*OF cp*, **where** $a=x$ **and** $b=-?N\ x\ e$])
from *prems* **have** *noSc*:$\forall\ t.\ l < t \wedge t < u \longrightarrow t \neq (- ?N\ x\ e) / real\ c$ **by** *auto*
with *ly yu* **have** *yne*: $y \neq - ?N\ x\ e / real\ c$ **by** *auto*
hence $y < (- ?N\ x\ e) / real\ c \vee y > (-?N\ x\ e) / real\ c$ **by** *auto*
moreover {**assume** *y*: $y < (-?N\ x\ e)/ real\ c$
hence $y * real\ c < - ?N\ x\ e$
by (*simp add*: *pos-less-divide-eq*[*OF cp*, **where** $a=y$ **and** $b=-?N\ x\ e$, *symmetric*])
hence $real\ c * y + ?N\ x\ e < 0$ **by** (*simp add*: *ring-simps*)
hence *?case* **using** *numbound0-I*[*OF nb*, **where** $bs=bs$ **and** $b=x$ **and** $b'=y$] **by** *simp*}
moreover {**assume** *y*: $y > (- ?N\ x\ e) / real\ c$
with *yu* **have** *eu*: $u > (- ?N\ x\ e) / real\ c$ **by** *auto*
with *noSc ly yu* **have** $(- ?N\ x\ e) / real\ c \leq l$ **by** (*cases* $(- ?N\ x\ e) / real\ c > l$, *auto*)
with *lx pxc* **have** *False* **by** *auto*
hence *?case* **by** *simp* }
ultimately show *?case* **by** *blast*
next
case (*6 c e*) **hence** *cp*: $real\ c > 0$ **and** *nb*: *numbound0 e* **by** *simp* +
from *prems* **have** $x * real\ c + ?N\ x\ e \leq 0$ **by** (*simp add*: *ring-simps*)
hence *pxc*: $x \leq (- ?N\ x\ e) / real\ c$
by (*simp only*: *pos-le-divide-eq*[*OF cp*, **where** $a=x$ **and** $b=-?N\ x\ e$])
from *prems* **have** *noSc*:$\forall\ t.\ l < t \wedge t < u \longrightarrow t \neq (- ?N\ x\ e) / real\ c$ **by** *auto*
with *ly yu* **have** *yne*: $y \neq - ?N\ x\ e / real\ c$ **by** *auto*
hence $y < (- ?N\ x\ e) / real\ c \vee y > (-?N\ x\ e) / real\ c$ **by** *auto*
moreover {**assume** *y*: $y < (-?N\ x\ e)/ real\ c$
hence $y * real\ c < - ?N\ x\ e$
by (*simp add*: *pos-less-divide-eq*[*OF cp*, **where** $a=y$ **and** $b=-?N\ x\ e$, *symmetric*])
hence $real\ c * y + ?N\ x\ e < 0$ **by** (*simp add*: *ring-simps*)
hence *?case* **using** *numbound0-I*[*OF nb*, **where** $bs=bs$ **and** $b=x$ **and** $b'=y$] **by** *simp*}
moreover {**assume** *y*: $y > (- ?N\ x\ e) / real\ c$
with *yu* **have** *eu*: $u > (- ?N\ x\ e) / real\ c$ **by** *auto*
with *noSc ly yu* **have** $(- ?N\ x\ e) / real\ c \leq l$ **by** (*cases* $(- ?N\ x\ e) / real\ c > l$, *auto*)
with *lx pxc* **have** *False* **by** *auto*
hence *?case* **by** *simp* }
ultimately show *?case* **by** *blast*
next
case (*7 c e*) **hence** *cp*: $real\ c > 0$ **and** *nb*: *numbound0 e* **by** *simp*+
from *prems* **have** $x * real\ c + ?N\ x\ e > 0$ **by** (*simp add*: *ring-simps*)
hence *pxc*: $x > (- ?N\ x\ e) / real\ c$
by (*simp only*: *pos-divide-less-eq*[*OF cp*, **where** $a=x$ **and** $b=-?N\ x\ e$])

from *prems* **have** *noSc*:$\forall$ *t*. $l < t \wedge t < u \longrightarrow t \neq (- \ ?N\ x\ e)$ / *real c* **by** *auto*
with *ly yu* **have** *yne*: $y \neq - \ ?N\ x\ e$ / *real c* **by** *auto*
hence $y < (- \ ?N\ x\ e)$ / *real c* $\vee$ $y > (-?N\ x\ e)$ / *real c* **by** *auto*
moreover {**assume** *y*: $y > (-?N\ x\ e)$/ *real c*
hence $y *$ *real c* $> - \ ?N\ x\ e$
by (*simp add*: *pos-divide-less-eq*[*OF cp*, **where** *a*=*y* **and** *b*=$-?N\ x\ e$, *symmetric*])
hence *real c* $* \ y + \ ?N\ x\ e > 0$ **by** (*simp add*: *ring-simps*)
hence *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*y*] **by** *simp*}
moreover {**assume** *y*: $y < (- \ ?N\ x\ e)$ / *real c*
with *ly* **have** *eu*: $l < (- \ ?N\ x\ e)$ / *real c* **by** *auto*
with *noSc ly yu* **have** $(- \ ?N\ x\ e)$ / *real c* $\geq u$ **by** (*cases* $(- \ ?N\ x\ e)$ / *real c* $> l$, *auto*)
with *xu pxc* **have** *False* **by** *auto*
hence *?case* **by** *simp* }
ultimately show *?case* **by** *blast*
next
case (*8 c e*) **hence** *cp*: *real c* > 0 **and** *nb*: *numbound0 e* **by** *simp*+
from *prems* **have** $x *$ *real c* $+ \ ?N\ x\ e \geq 0$ **by** (*simp add*: *ring-simps*)
hence *pxc*: $x \geq (- \ ?N\ x\ e)$ / *real c*
by (*simp only*: *pos-divide-le-eq*[*OF cp*, **where** *a*=*x* **and** *b*=$-?N\ x\ e$])
from *prems* **have** *noSc*:$\forall$ *t*. $l < t \wedge t < u \longrightarrow t \neq (- \ ?N\ x\ e)$ / *real c* **by** *auto*
with *ly yu* **have** *yne*: $y \neq - \ ?N\ x\ e$ / *real c* **by** *auto*
hence $y < (- \ ?N\ x\ e)$ / *real c* $\vee$ $y > (-?N\ x\ e)$ / *real c* **by** *auto*
moreover {**assume** *y*: $y > (-?N\ x\ e)$/ *real c*
hence $y *$ *real c* $> - \ ?N\ x\ e$
by (*simp add*: *pos-divide-less-eq*[*OF cp*, **where** *a*=*y* **and** *b*=$-?N\ x\ e$, *symmetric*])
hence *real c* $* \ y + \ ?N\ x\ e > 0$ **by** (*simp add*: *ring-simps*)
hence *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*y*] **by** *simp*}
moreover {**assume** *y*: $y < (- \ ?N\ x\ e)$ / *real c*
with *ly* **have** *eu*: $l < (- \ ?N\ x\ e)$ / *real c* **by** *auto*
with *noSc ly yu* **have** $(- \ ?N\ x\ e)$ / *real c* $\geq u$ **by** (*cases* $(- \ ?N\ x\ e)$ / *real c* $> l$, *auto*)
with *xu pxc* **have** *False* **by** *auto*
hence *?case* **by** *simp* }
ultimately show *?case* **by** *blast*
next
case (*3 c e*) **hence** *cp*: *real c* > 0 **and** *nb*: *numbound0 e* **by** *simp*+
from *cp* **have** *cnz*: *real c* $\neq 0$ **by** *simp*
from *prems* **have** $x *$ *real c* $+ \ ?N\ x\ e = 0$ **by** (*simp add*: *ring-simps*)
hence *pxc*: $x = (- \ ?N\ x\ e)$ / *real c*
by (*simp only*: *nonzero-eq-divide-eq*[*OF cnz*, **where** *a*=*x* **and** *b*=$-?N\ x\ e$])
from *prems* **have** *noSc*:$\forall$ *t*. $l < t \wedge t < u \longrightarrow t \neq (- \ ?N\ x\ e)$ / *real c* **by** *auto*

with *lx xu* **have** *yne*: $x \neq -$ *?N x e* / *real c* **by** *auto*
with *pxc* **show** *?case* **by** *simp*
next
case (*4 c e*) **hence** *cp*: *real c* > *0* **and** *nb*: *numbound0 e* **by** *simp*+
from *cp* **have** *cnz*: *real c* $\neq$ *0* **by** *simp*
from *prems* **have** *noSc*:$\forall$ *t*. $l < t \wedge t < u \longrightarrow t \neq$ (− *?N x e*) / *real c* **by** *auto*
with *ly yu* **have** *yne*: $y \neq -$ *?N x e* / *real c* **by** *auto*
hence *y*∗ *real c* $\neq$ −*?N x e*
by (*simp only*: *nonzero-eq-divide-eq*[*OF cnz*, **where** *a*=*y* **and** *b*=−*?N x e*]) *simp*
hence *y*∗ *real c* + *?N x e* $\neq$ *0* **by** (*simp add*: *ring-simps*)
thus *?case* **using** *numbound0-I*[*OF nb*, **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*y*]
by (*simp add*: *ring-simps*)
qed (*auto simp add*: *nth-pos2 numbound0-I*[**where** *bs*=*bs* **and** *b*=*y* **and** *b'*=*x*])

lemma *finite-set-intervals*:
assumes *px*: *P* (*x*::*real*)
and *lx*: $l \leq x$ **and** *xu*: $x \leq u$
and *linS*: $l \in S$ **and** *uinS*: $u \in S$
and *fS*:*finite S* **and** *lS*: $\forall$ $x \in S$. $l \leq x$ **and** *Su*: $\forall$ $x \in S$. $x \leq u$
shows $\exists$ $a \in S$. $\exists$ $b \in S$. ($\forall$ *y*. $a < y \wedge y < b \longrightarrow y \notin S$) $\wedge$ $a \leq x \wedge x \leq b \wedge$ *P x*
proof−
let *?Mx* = {*y*. $y \in S \wedge y \leq x$}
let *?xM* = {*y*. $y \in S \wedge x \leq y$}
let *?a* = *Max ?Mx*
let *?b* = *Min ?xM*
have *MxS*: *?Mx* $\subseteq$ *S* **by** *blast*
hence *fMx*: *finite ?Mx* **using** *fS finite-subset* **by** *auto*
from *lx linS* **have** *linMx*: $l \in$ *?Mx* **by** *blast*
hence *Mxne*: *?Mx* $\neq$ {} **by** *blast*
have *xMS*: *?xM* $\subseteq$ *S* **by** *blast*
hence *fxM*: *finite ?xM* **using** *fS finite-subset* **by** *auto*
from *xu uinS* **have** *linxM*: $u \in$ *?xM* **by** *blast*
hence *xMne*: *?xM* $\neq$ {} **by** *blast*
have *ax*:*?a* $\leq x$ **using** *Mxne fMx* **by** *auto*
have *xb*:$x \leq$ *?b* **using** *xMne fxM* **by** *auto*
have *?a* $\in$ *?Mx* **using** *Max-in*[*OF fMx Mxne*] **by** *simp* **hence** *ainS*: *?a* $\in$ *S* **using** *MxS* **by** *blast*
have *?b* $\in$ *?xM* **using** *Min-in*[*OF fxM xMne*] **by** *simp* **hence** *binS*: *?b* $\in$ *S* **using** *xMS* **by** *blast*
have *noy*:$\forall$ *y*. *?a* < $y \wedge y$ < *?b* $\longrightarrow y \notin S$
proof(*clarsimp*)
fix *y*
assume *ay*: *?a* < *y* **and** *yb*: *y* < *?b* **and** *yS*: $y \in S$
from *yS* **have** $y \in$ *?Mx* $\vee$ $y \in$ *?xM* **by** *auto*
moreover {**assume** $y \in$ *?Mx* **hence** $y \leq$ *?a* **using** *Mxne fMx* **by** *auto* **with** *ay* **have** *False* **by** *simp*}

moreover {**assume** $y \in$ *?xM* **hence** $y \geq$ *?b* **using** *xMne fxM* **by** *auto* **with** *yb* **have** *False* **by** *simp*}
ultimately show *False* **by** *blast*
qed
from *ainS binS noy ax xb px* **show** *?thesis* **by** *blast*
qed

lemma *finite-set-intervals2*:
assumes *px*: $P\ (x{::}real)$
and *lx*: $l \leq x$ **and** *xu*: $x \leq u$
and *linS*: $l \in S$ **and** *uinS*: $u \in S$
and *fS*:*finite S* **and** *lS*: $\forall\ x\in S.\ l \leq x$ **and** *Su*: $\forall\ x\in S.\ x \leq u$
shows $(\exists\ s\in S.\ P\ s) \vee (\exists\ a \in S.\ \exists\ b \in S.\ (\forall\ y.\ a < y \wedge y < b \longrightarrow y \notin S) \wedge a < x \wedge x < b \wedge P\ x)$
proof−
from *finite-set-intervals*[**where** $P{=}P$, *OF px lx xu linS uinS fS lS Su*]
obtain a **and** b **where**
as: $a\in S$ **and** *bs*: $b\in S$ **and** *noS*:$\forall y.\ a < y \wedge y < b \longrightarrow y \notin S$ **and** *axb*: $a \leq x \wedge x \leq b \wedge P\ x$ **by** *auto*
from *axb* **have** $x{=}\ a \vee x{=}\ b \vee (a < x \wedge x < b)$ **by** *auto*
thus *?thesis* **using** *px as bs noS* **by** *blast*
qed

lemma *rinf-uset*:
assumes *lp*: *isrlfm p*
and *nmi*: $\neg$ (*Ifm* (*x#bs*) (*minusinf p*)) (**is** $\neg$ (*Ifm* (*x#bs*) (*?M p*)))
and *npi*: $\neg$ (*Ifm* (*x#bs*) (*plusinf p*)) (**is** $\neg$ (*Ifm* (*x#bs*) (*?P p*)))
and *ex*: $\exists\ x.$ *Ifm* (*x#bs*) *p* (**is** $\exists\ x.$ *?I x p*)
shows $\exists\ (l,n) \in$ *set* (*uset p*). $\exists\ (s,m) \in$ *set* (*uset p*). *?I* ((*Inum* (*x#bs*) *l* / *real n* + *Inum* (*x#bs*) *s* / *real m*) / *2*) *p*
proof−
let *?N* = λ *x t. Inum* (*x#bs*) *t*
let *?U* = *set* (*uset p*)
from *ex* **obtain** *a* **where** *pa*: *?I a p* **by** *blast*
from *bound0-I*[*OF rminusinf-bound0*[*OF lp*], **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*a*] *nmi*
have *nmi'*: $\neg$ (*?I a* (*?M p*)) **by** *simp*
from *bound0-I*[*OF rplusinf-bound0*[*OF lp*], **where** *bs*=*bs* **and** *b*=*x* **and** *b'*=*a*] *npi*
have *npi'*: $\neg$ (*?I a* (*?P p*)) **by** *simp*
have $\exists\ (l,n) \in$ *set* (*uset p*). $\exists\ (s,m) \in$ *set* (*uset p*). *?I* ((*?N a l/real n* + *?N a s /real m*) / *2*) *p*
proof−
let *?M* = (λ (*t,c*). *?N a t* / *real c*) ‘ *?U*
have *fM*: *finite ?M* **by** *auto*
from *rminusinf-uset*[*OF lp nmi pa*] *rplusinf-uset*[*OF lp npi pa*]
have $\exists\ (l,n) \in$ *set* (*uset p*). $\exists\ (s,m) \in$ *set* (*uset p*). $a \leq$ *?N x l* / *real n* $\wedge\ a \geq$ *?N x s* / *real m* **by** *blast*
then obtain *t n s m* **where**

tnU: (*t*,*n*) ∈ *?U* **and** *smU*: (*s*,*m*) ∈ *?U*
and *xs1*: *a* ≤ *?N x s* / *real m* **and** *tx1*: *a* ≥ *?N x t* / *real n* **by** *blast*
from *uset-l*[*OF lp*] *tnU smU numbound0-I*[**where** *bs*=*bs* **and** *b*=*x* **and** *b′*=*a*] *xs1 tx1* **have** *xs*: *a* ≤ *?N a s* / *real m* **and** *tx*: *a* ≥ *?N a t* / *real n* **by** *auto*
from *tnU* **have** *Mne*: *?M* ≠ {} **by** *auto*
hence *Une*: *?U* ≠ {} **by** *simp*
let *?l* = *Min ?M*
let *?u* = *Max ?M*
have *linM*: *?l* ∈ *?M* **using** *fM Mne* **by** *simp*
have *uinM*: *?u* ∈ *?M* **using** *fM Mne* **by** *simp*
have *tnM*: *?N a t* / *real n* ∈ *?M* **using** *tnU* **by** *auto*
have *smM*: *?N a s* / *real m* ∈ *?M* **using** *smU* **by** *auto*
have *lM*: ∀ *t*∈ *?M*. *?l* ≤ *t* **using** *Mne fM* **by** *auto*
have *Mu*: ∀ *t*∈ *?M*. *t* ≤ *?u* **using** *Mne fM* **by** *auto*
have *?l* ≤ *?N a t* / *real n* **using** *tnM Mne* **by** *simp* **hence** *lx*: *?l* ≤ *a* **using** *tx* **by** *simp*
have *?N a s* / *real m* ≤ *?u* **using** *smM Mne* **by** *simp* **hence** *xu*: *a* ≤ *?u* **using** *xs* **by** *simp*
from *finite-set-intervals2*[**where** *P*=λ *x*. *?I x p*,*OF pa lx xu linM uinM fM lM Mu*]
have (∃ *s*∈ *?M*. *?I s p*) ∨
(∃ *t1*∈ *?M*. ∃ *t2* ∈ *?M*. (∀ *y*. *t1* < *y* ∧ *y* < *t2* ⟶ *y* ∉ *?M*) ∧ *t1* < *a* ∧ *a* < *t2* ∧ *?I a p*) **.**
moreover **{ fix** *u* **assume** *um*: *u*∈ *?M* **and** *pu*: *?I u p*
hence ∃ (*tu*,*nu*) ∈ *?U*. *u* = *?N a tu* / *real nu* **by** *auto*
then obtain *tu nu* **where** *tuU*: (*tu*,*nu*) ∈ *?U* **and** *tuu*:*u*= *?N a tu* / *real nu* **by** *blast*
have (*u* + *u*) / *2* = *u* **by** *auto* **with** *pu tuu*
have *?I* (((*?N a tu* / *real nu*) + (*?N a tu* / *real nu*)) / *2*) *p* **by** *simp*
with *tuU* **have** *?thesis* **by** *blast***}**
moreover{
assume ∃ *t1*∈ *?M*. ∃ *t2* ∈ *?M*. (∀ *y*. *t1* < *y* ∧ *y* < *t2* ⟶ *y* ∉ *?M*) ∧ *t1* < *a* ∧ *a* < *t2* ∧ *?I a p*
then obtain *t1* **and** *t2* **where** *t1M*: *t1* ∈ *?M* **and** *t2M*: *t2*∈ *?M*
and *noM*: ∀ *y*. *t1* < *y* ∧ *y* < *t2* ⟶ *y* ∉ *?M* **and** *t1x*: *t1* < *a* **and** *xt2*: *a* < *t2* **and** *px*: *?I a p*
by *blast*
from *t1M* **have** ∃ (*t1u*,*t1n*) ∈ *?U*. *t1* = *?N a t1u* / *real t1n* **by** *auto*
then obtain *t1u t1n* **where** *t1uU*: (*t1u*,*t1n*) ∈ *?U* **and** *t1u*: *t1* = *?N a t1u* / *real t1n* **by** *blast*
from *t2M* **have** ∃ (*t2u*,*t2n*) ∈ *?U*. *t2* = *?N a t2u* / *real t2n* **by** *auto*
then obtain *t2u t2n* **where** *t2uU*: (*t2u*,*t2n*) ∈ *?U* **and** *t2u*: *t2* = *?N a t2u* / *real t2n* **by** *blast*
from *t1x xt2* **have** *t1t2*: *t1* < *t2* **by** *simp*
let *?u* = (*t1* + *t2*) / *2*
from *less-half-sum*[*OF t1t2*] *gt-half-sum*[*OF t1t2*] **have** *t1lu*: *t1* < *?u* **and** *ut2*: *?u* < *t2* **by** *auto*
from *lin-dense*[*OF lp noM t1x xt2 px t1lu ut2*] **have** *?I ?u p* **.**
with *t1uU t2uU t1u t2u* **have** *?thesis* **by** *blast***}**

ultimately show *?thesis* **by** *blast*
qed
then obtain *l n s m* **where** *lnU*: (*l*,*n*) ∈ *?U* **and** *smU*:(*s*,*m*) ∈ *?U*
and *pu*: *?I* ((*?N a l* / *real n* + *?N a s* / *real m*) / *2*) *p* **by** *blast*
from *lnU smU uset-l*[*OF lp*] **have** *nbl*: *numbound0 l* **and** *nbs*: *numbound0 s* **by** *auto*
from *numbound0-I*[*OF nbl*, **where** *bs*=*bs* **and** *b*=*a* **and** *b'*=*x*]
numbound0-I[*OF nbs*, **where** *bs*=*bs* **and** *b*=*a* **and** *b'*=*x*] *pu*
have *?I* ((*?N x l* / *real n* + *?N x s* / *real m*) / *2*) *p* **by** *simp*
with *lnU smU*
show *?thesis* **by** *auto*
qed

theorem *fr-eq*:
assumes *lp*: *isrlfm p*
shows (∃ *x*. *Ifm* (*x*#*bs*) *p*) = ((*Ifm* (*x*#*bs*) (*minusinf p*)) ∨ (*Ifm* (*x*#*bs*) (*plusinf p*)) ∨ (∃ (*t*,*n*) ∈ *set* (*uset p*). ∃ (*s*,*m*) ∈ *set* (*uset p*). *Ifm* ((((*Inum* (*x*#*bs*) *t*)/ *real n* + (*Inum* (*x*#*bs*) *s*) / *real m*) /*2*)#*bs*) *p*))
(**is** (∃ *x*. *?I x p*) = (*?M* ∨ *?P* ∨ *?F*) **is** *?E* = *?D*)
proof
assume *px*: ∃ *x*. *?I x p*
have *?M* ∨ *?P* ∨ (¬ *?M* ∧ ¬ *?P*) **by** *blast*
moreover {**assume** *?M* ∨ *?P* **hence** *?D* **by** *blast*}
moreover {**assume** *nmi*: ¬ *?M* **and** *npi*: ¬ *?P*
from *rinf-uset*[*OF lp nmi npi*] **have** *?F* **using** *px* **by** *blast* **hence** *?D* **by** *blast*}
ultimately show *?D* **by** *blast*
next
assume *?D*
moreover {**assume** *m*:*?M* **from** *rminusinf-ex*[*OF lp m*] **have** *?E* **.**}
moreover {**assume** *p*: *?P* **from** *rplusinf-ex*[*OF lp p*] **have** *?E* **.** }
moreover {**assume** *f*:*?F* **hence** *?E* **by** *blast*}
ultimately show *?E* **by** *blast*
qed

lemma *fr-equsubst*:
assumes *lp*: *isrlfm p*
shows (∃ *x*. *Ifm* (*x*#*bs*) *p*) = ((*Ifm* (*x*#*bs*) (*minusinf p*)) ∨ (*Ifm* (*x*#*bs*) (*plusinf p*)) ∨ (∃ (*t*,*k*) ∈ *set* (*uset p*). ∃ (*s*,*l*) ∈ *set* (*uset p*). *Ifm* (*x*#*bs*) (*usubst p* (*Add*(*Mul l t*) (*Mul k s*) , *2*∗*k*∗*l*))))
(**is** (∃ *x*. *?I x p*) = (*?M* ∨ *?P* ∨ *?F*) **is** *?E* = *?D*)
proof
assume *px*: ∃ *x*. *?I x p*
have *?M* ∨ *?P* ∨ (¬ *?M* ∧ ¬ *?P*) **by** *blast*
moreover {**assume** *?M* ∨ *?P* **hence** *?D* **by** *blast*}
moreover {**assume** *nmi*: ¬ *?M* **and** *npi*: ¬ *?P*
let *?f* =λ (*t*,*n*). *Inum* (*x*#*bs*) *t* / *real n*
let *?N* = λ *t*. *Inum* (*x*#*bs*) *t*

{**fix** *t n s m* **assume** (*t*,*n*)∈ *set* (*uset p*) **and** (*s*,*m*) ∈ *set* (*uset p*)
with *uset-l*[*OF lp*] **have** *tnb*: *numbound0 t* **and** *np*:*real n* > *0* **and** *snb*: *numbound0 s* **and** *mp*:*real m* > *0*
by *auto*
let *?st* = *Add* (*Mul m t*) (*Mul n s*)
from *mult-pos-pos*[*OF np mp*] **have** *mnp*: *real* (*2*∗*n*∗*m*) > *0*
by (*simp add*: *mult-commute*)
from *tnb snb* **have** *st-nb*: *numbound0 ?st* **by** *simp*
have *st*: (*?N t* / *real n* + *?N s* / *real m*)/*2* = *?N ?st* / *real* (*2*∗*n*∗*m*)
using *mnp mp np* **by** (*simp add*: *ring-simps add-divide-distrib*)
from *usubst-I*[*OF lp mnp st-nb*, **where** *x*=*x* **and** *bs*=*bs*]
have *?I x* (*usubst p* (*?st*,*2*∗*n*∗*m*)) = *?I* ((*?N t* / *real n* + *?N s* / *real m*) /*2*) *p* **by** (*simp only*: *st*[*symmetric*])}
with *rinf-uset*[*OF lp nmi npi px*] **have** *?F* **by** *blast* **hence** *?D* **by** *blast*}
ultimately show *?D* **by** *blast*
next
assume *?D*
moreover {**assume** *m*:*?M* **from** *rminusinf-ex*[*OF lp m*] **have** *?E* **.**}
moreover {**assume** *p*: *?P* **from** *rplusinf-ex*[*OF lp p*] **have** *?E* **.** }
moreover {**fix** *t k s l* **assume** (*t*,*k*) ∈ *set* (*uset p*) **and** (*s*,*l*) ∈ *set* (*uset p*)
and *px*:*?I x* (*usubst p* (*Add* (*Mul l t*) (*Mul k s*), *2*∗*k*∗*l*))
with *uset-l*[*OF lp*] **have** *tnb*: *numbound0 t* **and** *np*:*real k* > *0* **and** *snb*: *numbound0 s* **and** *mp*:*real l* > *0* **by** *auto*
let *?st* = *Add* (*Mul l t*) (*Mul k s*)
from *mult-pos-pos*[*OF np mp*] **have** *mnp*: *real* (*2*∗*k*∗*l*) > *0*
by (*simp add*: *mult-commute*)
from *tnb snb* **have** *st-nb*: *numbound0 ?st* **by** *simp*
from *usubst-I*[*OF lp mnp st-nb*, **where** *bs*=*bs*] *px* **have** *?E* **by** *auto*}
ultimately show *?E* **by** *blast*
qed

constdefs *ferrack*:: *fm* ⇒ *fm*
ferrack p ≡ (*let p′* = *rlfm* (*simpfm p*); *mp* = *minusinf p′*; *pp* = *plusinf p′*
in if (*mp* = *T* ∨ *pp* = *T*) *then T else*
(*let U* = *remdps*(*map simp-num-pair*
(*map* (λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , *2*∗*n*∗*m*))
(*alluopairs* (*uset p′*))))
in decr (*disj mp* (*disj pp* (*evaldjf* (*simpfm o* (*usubst p′*)) *U*)))))

lemma *uset-cong-aux*:
assumes *Ul*: ∀ (*t*,*n*) ∈ *set U*. *numbound0 t* ∧ *n* >*0*
shows ((λ (*t*,*n*). *Inum* (*x*#*bs*) *t* /*real n*) ‘ (*set* (*map* (λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , *2*∗*n*∗*m*)) (*alluopairs U*)))) = ((λ ((*t*,*n*),(*s*,*m*)). (*Inum* (*x*#*bs*) *t* /*real n* + *Inum* (*x*#*bs*) *s* /*real m*)/*2*) ‘ (*set U* × *set U*))
(**is** *?lhs* = *?rhs*)
proof(*auto*)
fix *t n s m*

assume $((t,n),(s,m)) \in$ *set* (*alluopairs U*)
hence *th*: $((t,n),(s,m)) \in$ (*set U* $\times$ *set U*)
using *alluopairs-set1*[**where** *xs*=*U*] **by** *blast*
let *?N* = λ *t*. *Inum* (*x*#*bs*) *t*
let *?st*= *Add* (*Mul m t*) (*Mul n s*)
from *Ul th* **have** *mnz*: $m \neq 0$ **by** *auto*
from *Ul th* **have** *nnz*: $n \neq 0$ **by** *auto*
have *st*: (*?N t* / *real n* + *?N s* / *real m*)/2 = *?N ?st* / *real* (2*n*m)
using *mnz nnz* **by** (*simp add*: *ring-simps add-divide-distrib*)

thus (*real m* * *Inum* (*x* # *bs*) *t* + *real n* * *Inum* (*x* # *bs*) *s*) /
(2 * *real n* * *real m*)
$\in$ (λ((*t*, *n*), *s*, *m*).
(*Inum* (*x* # *bs*) *t* / *real n* + *Inum* (*x* # *bs*) *s* / *real m*) / 2) ‘
(*set U* $\times$ *set U*)**using** *mnz nnz th*
apply (*auto simp add*: *th add-divide-distrib ring-simps split-def image-def*)
by (*rule-tac x*=(*s*,*m*) **in** *bexI*,*simp-all*)
(*rule-tac x*=(*t*,*n*) **in** *bexI*,*simp-all*)
next
fix *t n s m*
assume *tnU*: $(t,n) \in$ *set U* **and** *smU*:$(s,m) \in$ *set U*
let *?N* = λ *t*. *Inum* (*x*#*bs*) *t*
let *?st*= *Add* (*Mul m t*) (*Mul n s*)
from *Ul smU* **have** *mnz*: $m \neq 0$ **by** *auto*
from *Ul tnU* **have** *nnz*: $n \neq 0$ **by** *auto*
have *st*: (*?N t* / *real n* + *?N s* / *real m*)/2 = *?N ?st* / *real* (2*n*m)
using *mnz nnz* **by** (*simp add*: *ring-simps add-divide-distrib*)
let *?P* = λ (*t′*,*n′*) (*s′*,*m′*). (*Inum* (*x* # *bs*) *t* / *real n* + *Inum* (*x* # *bs*) *s* / *real m*)/2 = (*Inum* (*x* # *bs*) *t′* / *real n′* + *Inum* (*x* # *bs*) *s′* / *real m′*)/2
have *Pc*:$\forall$ *a b*. *?P a b* = *?P b a*
by *auto*
from *Ul alluopairs-set1* **have** *Up*:$\forall$ $((t,n),(s,m)) \in$ *set* (*alluopairs U*). $n \neq 0 \wedge m \neq 0$ **by** *blast*
from *alluopairs-ex*[*OF Pc*, **where** *xs*=*U*] *tnU smU*
have *th′*:$\exists$ ((*t′*,*n′*),(*s′*,*m′*)) $\in$ *set* (*alluopairs U*). *?P* (*t′*,*n′*) (*s′*,*m′*)
by *blast*
then obtain *t′ n′ s′ m′* **where** *ts′-U*: ((*t′*,*n′*),(*s′*,*m′*)) $\in$ *set* (*alluopairs U*)
and *Pts′*: *?P* (*t′*,*n′*) (*s′*,*m′*) **by** *blast*
from *ts′-U Up* **have** *mnz′*: $m' \neq 0$ **and** *nnz′*: $n' \neq 0$ **by** *auto*
let *?st′* = *Add* (*Mul m′ t′*) (*Mul n′ s′*)
have *st′*: (*?N t′* / *real n′* + *?N s′* / *real m′*)/2 = *?N ?st′* / *real* (2*n′*m′)
using *mnz′ nnz′* **by** (*simp add*: *ring-simps add-divide-distrib*)
from *Pts′* **have**
(*Inum* (*x* # *bs*) *t* / *real n* + *Inum* (*x* # *bs*) *s* / *real m*)/2 = (*Inum* (*x* # *bs*) *t′* / *real n′* + *Inum* (*x* # *bs*) *s′* / *real m′*)/2 **by** *simp*
also have ... = ((λ(*t*, *n*). *Inum* (*x* # *bs*) *t* / *real n*) ((λ((*t*, *n*), *s*, *m*). (*Add* (*Mul m t*) (*Mul n s*), 2 * *n* * *m*)) ((*t′*,*n′*),(*s′*,*m′*)))) **by** (*simp add*: *st′*)
finally show (*Inum* (*x* # *bs*) *t* / *real n* + *Inum* (*x* # *bs*) *s* / *real m*) / 2
$\in$ (λ(*t*, *n*). *Inum* (*x* # *bs*) *t* / *real n*) ‘

(λ((t, n), s, m). (Add (Mul m t) (Mul n s), 2 ∗ n ∗ m)) ‘
set (alluopairs U)
using ts′-U by blast
qed

lemma *uset-cong*:
assumes *lp*: *isrlfm p*
and *UU′*: ((λ (t,n). Inum (x#bs) t /real n) ‘ U′) = ((λ ((t,n),(s,m)). (Inum (x#bs) t /real n + Inum (x#bs) s /real m)/2) ‘ (U × U)) (**is** ?f ‘ U′ = ?g ‘ (U×U))
and *U*: ∀ (t,n) ∈ U. numbound0 t ∧ n > 0
and *U′*: ∀ (t,n) ∈ U′. numbound0 t ∧ n > 0
shows (∃ (t,n) ∈ U. ∃ (s,m) ∈ U. Ifm (x#bs) (usubst p (Add (Mul m t) (Mul n s),2∗n∗m))) = (∃ (t,n) ∈ U′. Ifm (x#bs) (usubst p (t,n)))
(**is** ?lhs = ?rhs)
proof
assume *?lhs*
then obtain *t n s m* **where** *tnU*: (t,n) ∈ U **and** *smU*:(s,m) ∈ U **and**
Pst: Ifm (x#bs) (usubst p (Add (Mul m t) (Mul n s),2∗n∗m)) **by** *blast*
let ?N = λ t. Inum (x#bs) t
from *tnU smU U* **have** *tnb*: *numbound0 t* **and** *np*: n > 0
and *snb*: *numbound0 s* **and** *mp*:m > 0 **by** *auto*
let ?st= Add (Mul m t) (Mul n s)
from *mult-pos-pos*[*OF np mp*] **have** *mnp*: real (2∗n∗m) > 0
by (*simp add*: *mult-commute real-of-int-mult*[*symmetric*] *del*: *real-of-int-mult*)
from *tnb snb* **have** *stnb*: *numbound0 ?st* **by** *simp*
have *st*: (?N t / real n + ?N s / real m)/2 = ?N ?st / real (2∗n∗m)
using *mp np* **by** (*simp add*: *ring-simps add-divide-distrib*)
from *tnU smU UU′* **have** ?g ((t,n),(s,m)) ∈ ?f ‘ U′ **by** *blast*
hence ∃ (t′,n′) ∈ U′. ?g ((t,n),(s,m)) = ?f (t′,n′)
by *auto* (*rule-tac x*=(a,b) **in** *bexI*, *auto*)
then obtain *t′ n′* **where** *tnU′*: (t′,n′) ∈ U′ **and** *th*: ?g ((t,n),(s,m)) = ?f (t′,n′) **by** *blast*
from *U′ tnU′* **have** *tnb′*: *numbound0 t′* **and** *np′*: real n′ > 0 **by** *auto*
from *usubst-I*[*OF lp mnp stnb*, **where** *bs*=*bs* **and** *x*=*x*] *Pst*
have *Pst2*: Ifm (Inum (x # bs) (Add (Mul m t) (Mul n s)) / real (2 ∗ n ∗ m) # bs) p **by** *simp*
from *conjunct1*[*OF usubst-I*[*OF lp np′ tnb′*, **where** *bs*=*bs* **and** *x*=*x*], *symmetric*] *th*[*simplified split-def fst-conv snd-conv*,*symmetric*] *Pst2*[*simplified st*[*symmetric*]]
have Ifm (x # bs) (usubst p (t′, n′)) **by** (*simp only*: *st*)
then show *?rhs* **using** *tnU′* **by** *auto*
next
assume *?rhs*
then obtain *t′ n′* **where** *tnU′*: (t′,n′) ∈ U′ **and** *Pt′*: Ifm (x # bs) (usubst p (t′, n′))
by *blast*
from *tnU′ UU′* **have** ?f (t′,n′) ∈ ?g ‘ (U×U) **by** *blast*
hence ∃ ((t,n),(s,m)) ∈ (U×U). ?f (t′,n′) = ?g ((t,n),(s,m))
by *auto* (*rule-tac x*=(a,b) **in** *bexI*, *auto*)

then obtain *t n s m* **where** *tnU*: (*t*,*n*) ∈ *U* **and** *smU*:(*s*,*m*) ∈ *U* **and**
th: *?f* (*t′*,*n′*) = *?g*((*t*,*n*),(*s*,*m*)) **by** *blast*
let *?N* = λ *t*. *Inum* (*x#bs*) *t*
from *tnU smU U* **have** *tnb*: *numbound0 t* **and** *np*: *n* > *0*
and *snb*: *numbound0 s* **and** *mp*:*m* > *0* **by** *auto*
let *?st*= *Add* (*Mul m t*) (*Mul n s*)
from *mult-pos-pos*[*OF np mp*] **have** *mnp*: *real* (*2*∗*n*∗*m*) > *0*
by (*simp add*: *mult-commute real-of-int-mult*[*symmetric*] *del*: *real-of-int-mult*)
from *tnb snb* **have** *stnb*: *numbound0 ?st* **by** *simp*
have *st*: (*?N t* / *real n* + *?N s* / *real m*)/*2* = *?N ?st* / *real* (*2*∗*n*∗*m*)
using *mp np* **by** (*simp add*: *ring-simps add-divide-distrib*)
from *U′ tnU′* **have** *tnb′*: *numbound0 t′* **and** *np′*: *real n′* > *0* **by** *auto*
from *usubst-I*[*OF lp np′ tnb′*, **where** *bs*=*bs* **and** *x*=*x*,*simplified th*[*simplified split-def fst-conv snd-conv*] *st*] *Pt′*
have *Pst2*: *Ifm* (*Inum* (*x* # *bs*) (*Add* (*Mul m t*) (*Mul n s*)) / *real* (*2* ∗ *n* ∗ *m*) # *bs*) *p* **by** *simp*
with *usubst-I*[*OF lp mnp stnb*, **where** *x*=*x* **and** *bs*=*bs*] *tnU smU* **show** *?lhs* **by** *blast*
qed

lemma *ferrack*:
assumes *qf*: *qfree p*
shows *qfree* (*ferrack p*) ∧ ((*Ifm bs* (*ferrack p*)) = (∃ *x*. *Ifm* (*x#bs*) *p*))
(**is** - ∧ (*?rhs* = *?lhs*))
proof−
let *?I* = λ *x p*. *Ifm* (*x#bs*) *p*
let *?N* = λ *t*. *Inum* (*x#bs*) *t*
let *?q* = *rlfm* (*simpfm p*)
let *?U* = *uset ?q*
let *?Up* = *alluopairs ?U*
let *?g* = λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , *2*∗*n*∗*m*)
let *?S* = *map ?g ?Up*
let *?SS* = *map simp-num-pair ?S*
let *?Y* = *remdps ?SS*
let *?f*= (λ (*t*,*n*). *?N t* / *real n*)
let *?h* = λ ((*t*,*n*),(*s*,*m*)). (*?N t*/*real n* + *?N s*/ *real m*) /*2*
let *?F* = λ *p*. ∃ *a* ∈ *set* (*uset p*). ∃ *b* ∈ *set* (*uset p*). *?I x* (*usubst p* (*?g*(*a*,*b*)))
let *?ep* = *evaldjf* (*simpfm o* (*usubst ?q*)) *?Y*
from *rlfm-I*[*OF simpfm-qf*[*OF qf*]] **have** *lq*: *isrlfm ?q* **by** *blast*
from *alluopairs-set1*[**where** *xs*=*?U*] **have** *UpU*: *set ?Up* ≤ (*set ?U* × *set ?U*) **by** *simp*
from *uset-l*[*OF lq*] **have** *U-l*: ∀ (*t*,*n*) ∈ *set ?U*. *numbound0 t* ∧ *n* > *0* **.**
from *U-l UpU*
have ∀ ((*t*,*n*),(*s*,*m*)) ∈ *set ?Up*. *numbound0 t* ∧ *n*> *0* ∧ *numbound0 s* ∧ *m* > *0* **by** *auto*
hence *Snb*: ∀ (*t*,*n*) ∈ *set ?S*. *numbound0 t* ∧ *n* > *0*
by (*auto simp add*: *mult-pos-pos*)
have *Y-l*: ∀ (*t*,*n*) ∈ *set ?Y*. *numbound0 t* ∧ *n* > *0*
proof−

{ **fix** *t n* **assume** *tnY*: $(t,n) \in$ *set ?Y*
hence $(t,n) \in$ *set ?SS* **by** *simp*
hence $\exists$ $(t',n') \in$ *set ?S*. *simp-num-pair* $(t',n') = (t,n)$
by (*auto simp add*: *split-def*) (*rule-tac x*=((*aa*,*ba*),(*ab*,*bb*)) **in** *bexI*, *simp-all*)
then obtain t' n' **where** *tn'S*: $(t',n') \in$ *set ?S* **and** *tns*: *simp-num-pair* $(t',n') = (t,n)$ **by** *blast*
from *tn'S Snb* **have** *tnb*: *numbound0* t' **and** *np*: $n' > 0$ **by** *auto*
from *simp-num-pair-l*[*OF tnb np tns*]
have *numbound0* $t \wedge n > 0$ **.** }
thus *?thesis* **by** *blast*
qed

have *YU*: (*?f* ' *set ?Y*) = (*?h* ' (*set ?U* $\times$ *set ?U*))
proof−
from *simp-num-pair-ci*[**where** *bs*=*x*#*bs*] **have**
$\forall x$. (*?f o simp-num-pair*) x = *?f x* **by** *auto*
hence *th*: *?f o simp-num-pair* = *?f* **using** *ext* **by** *blast*
have (*?f* ' *set ?Y*) = ((*?f o simp-num-pair*) ' *set ?S*) **by** (*simp add*: *image-compose*)
also have ... = (*?f* ' *set ?S*) **by** (*simp add*: *th*)
also have ... = ((*?f o ?g*) ' *set ?Up*)
by (*simp only*: *set-map o-def image-compose*[*symmetric*])
also have ... = (*?h* ' (*set ?U* $\times$ *set ?U*))
using *uset-cong-aux*[*OF U-l*, **where** *x*=*x* **and** *bs*=*bs*, *simplified set-map image-compose*[*symmetric*]] **by** *blast*
finally show *?thesis* **.**
qed
have $\forall$ $(t,n) \in$ *set ?Y*. *bound0* (*simpfm* (*usubst ?q* (t,n)))
proof−
{ **fix** *t n* **assume** *tnY*: $(t,n) \in$ *set ?Y*
with *Y-l* **have** *tnb*: *numbound0 t* **and** *np*: *real* $n > 0$ **by** *auto*
from *usubst-I*[*OF lq np tnb*]
have *bound0* (*usubst ?q* (t,n)) **by** *simp* **hence** *bound0* (*simpfm* (*usubst ?q* (t,n)))
using *simpfm-bound0* **by** *simp*}
thus *?thesis* **by** *blast*
qed
hence *ep-nb*: *bound0 ?ep* **using** *evaldjf-bound0*[**where** *xs*=*?Y* **and** *f*=*simpfm o* (*usubst ?q*)] **by** *auto*
let *?mp* = *minusinf ?q*
let *?pp* = *plusinf ?q*
let *?M* = *?I x ?mp*
let *?P* = *?I x ?pp*
let *?res* = *disj ?mp* (*disj ?pp ?ep*)
from *rminusinf-bound0*[*OF lq*] *rplusinf-bound0*[*OF lq*] *ep-nb*
have *nbth*: *bound0 ?res* **by** *auto*

from *conjunct1*[*OF rlfm-I*[*OF simpfm-qf*[*OF qf*]]] *simpfm*

have *th*: *?lhs* = ($\exists$ x. *?I x ?q*) **by** *auto*

from *th fr-equsubst*[*OF lq*, **where** *bs*=*bs* **and** *x*=*x*] **have** *lhfr*: *?lhs* = (*?M* ∨ *?P* ∨ *?F ?q*)
by (*simp only*: *split-def fst-conv snd-conv*)
also have ... = (*?M* ∨ *?P* ∨ (∃ (*t*,*n*) ∈ *set ?Y*. *?I x* (*simpfm* (*usubst ?q* (*t*,*n*)))))

using *uset-cong*[*OF lq YU U-l Y-l*] **by** (*simp only*: *split-def fst-conv snd-conv simpfm*)
also have ... = (*Ifm* (*x*#*bs*) *?res*)
using *evaldjf-ex*[**where** *ps*=*?Y* **and** *bs* = *x*#*bs* **and** *f*=*simpfm o* (*usubst ?q*),*symmetric*]
by (*simp add*: *split-def pair-collapse*)
finally have *lheq*: *?lhs* = (*Ifm bs* (*decr ?res*)) **using** *decr*[*OF nbth*] **by** *blast*
hence *lr*: *?lhs* = *?rhs* **apply** (*unfold ferrack-def Let-def*)
by (*cases ?mp* = *T* ∨ *?pp* = *T*, *auto*) (*simp add*: *disj-def*)+
from *decr-qf*[*OF nbth*] **have** *qfree* (*ferrack p*) **by** (*auto simp add*: *Let-def ferrack-def*)
with *lr* **show** *?thesis* **by** *blast*
qed

constdefs *linrqe*:: *fm* ⇒ *fm*
linrqe ≡ (λ *p*. *qelim* (*prep p*) *ferrack*)

theorem *linrqe*: (*Ifm bs* (*linrqe p*) = *Ifm bs p*) ∧ *qfree* (*linrqe p*)
using *ferrack qelim-ci prep*
unfolding *linrqe-def* **by** *auto*

definition
ferrack-test :: *unit* ⇒ *fm*
where
ferrack-test u = *linrqe* (*A* (*A* (*Imp* (*Lt* (*Sub* (*Bound 1*) (*Bound 0*)))
(*E* (*Eq* (*Sub* (*Add* (*Bound 0*) (*Bound 2*)) (*Bound 1*)))))))

export-code *linrqe ferrack-test* **in** *SML* **module-name** *Ferrack*

ML ⟪ *Ferrack.ferrack-test* () ⟫

use *linreif.ML*
oracle *linr-oracle* (*term*) = *ReflectedFerrack.linrqe-oracle*
use *linrtac.ML*
setup *LinrTac.setup*

end