

Miscellaneous HOL-Complex Examples

June 8, 2008

Contents

1	Binary arithmetic examples	2
1.1	Real Arithmetic	2
1.1.1	Addition	2
1.1.2	Negation	3
1.1.3	Multiplication	3
1.1.4	Inequalities	3
1.1.5	Powers	3
1.1.6	Tests	4
1.2	Complex Arithmetic	10
2	Square roots of primes are irrational	10
2.1	Preliminaries	10
2.2	Main theorem	11
2.3	Variations	12
3	Square roots of primes are irrational (script version)	13
3.1	Preliminaries	13
3.2	The set of rational numbers	14
3.3	Main theorem	14
4	The Nonstandard Primes as an Extension of the Prime Numbers	14
4.1	Another characterization of infinite set of natural numbers . .	16
4.2	An injective function cannot define an embedded natural number	17
4.3	Existence of Infinitely Many Primes: a Nonstandard Proof . .	19
5	Big O notation – continued	21
6	Arithmetic Series for Reals	22
7	Divergence of the Harmonic Series	22

8	Abstract	23
9	Formal Proof	23
10	Denumerability of the Rationals	29
11	Pretty integer literals for code generation	31
12	Quatifier elimination for $R(0,1,+, \text{floor}, i)$	33
13	Type of indices	167
13.1	Datatype of indices	167
13.2	Indices as datatype of ints	169
13.3	Basic arithmetic	169
13.4	ML interface	171
13.5	Specialized <i>op -</i> , <i>op div</i> and <i>op mod</i> operations	172
13.6	Code serialization	172
14	Implementation of natural numbers by target-language integers	173
14.1	Basic arithmetic	174
14.2	Case analysis	175
14.3	Preprocessors	175
14.4	Target language setup	175
15	Quatifier elimination for $R(0,1,+, i)$	179

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 $2 ^ 15 = (32768::real)$
by *simp*

lemma $-3 ^ 7 = (-2187::real)$
by *simp*

lemma $13 ^ 7 = (62748517::real)$
by *simp*

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

lemma $-5^{11} = (-48828125::real)$
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 $x \leq (x::real)$

by *arith*

lemma $((x::real) \leq y) = (x < y \vee x = y)$

by *arith*

lemma $((x::real) \leq y \wedge y \leq x) = (x = y)$

by *arith*

lemma $\neg(x < y \wedge y \leq (x::real))$

by *arith*

lemma $\neg(x \leq y \wedge y < (x::real))$

by *arith*

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

by *arith*

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

by *arith*

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

by *arith*

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

by *arith*

lemma $(x::real) = y \vee x < y \vee y < x$

by *arith*

lemma $(x::real) = 0 \vee 0 < x \vee 0 < -x$

by *arith*

lemma $(0::real) \leq x \vee 0 \leq -x$

by *arith*

lemma $((x::real) + y \leq x + z) = (y \leq z)$

by *arith*

lemma $((x::real) + z \leq y + z) = (x \leq y)$

by *arith*

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

by *arith*

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

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-distrib*)

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

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

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

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

theorem *rational*s-rep [*elim?*]:

assumes $x \in \mathbb{Q}$

obtains $m n$ **where** $n \neq 0$ **and** $|x| = \text{real } m / \text{real } n$ **and** $\text{gcd } (m, n) = 1$

proof –

from $\langle x \in \mathbb{Q} \rangle$ **obtain** $m n :: nat$ **where**

$n: n \neq 0$ **and** $x\text{-rat}: |x| = \text{real } m / \text{real } n$

unfolding *rational*s-def **by** *blast*

let $?gcd = \text{gcd } (m, n)$

from n **have** $\text{gcd}: ?gcd \neq 0$ **by** (*simp add: gcd-zero*)

let $?k = m \text{ div } ?gcd$

```

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: m2 = p * n2
  proof -
    from n and sqrt-rat have real m = |sqrt (real p)| * real n by simp
    then have real (m2) = (sqrt (real p))2 * real (n2)
      by (auto simp add: power2-eq-square)
    also have (sqrt (real p))2 = real p by simp
    also have ... * real (n2) = real (p * n2) by simp
    finally show ?thesis ..
  qed

```

```

qed
have p dvd m ∧ p dvd n
proof
  from eq have p dvd m2 ..
  with ⟨prime p⟩ show p dvd m by (rule prime-dvd-power-two)
  then obtain k where m = p * k ..
  with eq have p * n2 = p2 * k2 by (auto simp add: power2-eq-square mult-ac)
  with p have n2 = p * k2 by (simp add: power2-eq-square)
  then have p dvd n2 ..
  with ⟨prime p⟩ show p dvd n by (rule prime-dvd-power-two)
qed
then have p dvd gcd (m, n) ..
with gcd have p dvd 1 by simp
then have p ≤ 1 by (simp add: dvd-imp-le)
with p show False by simp
qed

corollary sqrt (real (2::nat)) ∉ 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 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 ..
  from n and sqrt-rat have real m = |sqrt (real p)| * real n by simp
  then have real (m2) = (sqrt (real p))2 * real (n2)
    by (auto simp add: power2-eq-square)
  also have (sqrt (real p))2 = real p by simp
  also have ... * real (n2) = real (p * n2) by simp
  finally have eq: m2 = p * n2 ..
  then have p dvd m2 ..
  with ⟨prime p⟩ have dvd-m: p dvd m by (rule prime-dvd-power-two)
  then obtain k where m = p * k ..
  with eq have p * n2 = p2 * k2 by (auto simp add: power2-eq-square mult-ac)
  with p have n2 = p * k2 by (simp add: power2-eq-square)
  then have p dvd n2 ..
  with ⟨prime p⟩ have p dvd n by (rule prime-dvd-power-two)
  with dvd-m have p dvd gcd (m, n) by (rule gcd-greatest)
  with gcd have p dvd 1 by simp

```

```

    then have  $p \leq 1$  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  $p \implies p \neq 0$ 
  by (force simp add: prime-def)

```

```

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

```

```

lemma reduction: prime  $p \implies$ 
   $0 < k \implies k * k = p * (j * j) \implies k < p * j \wedge 0 < j$ 
  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:  $(j::\text{nat}) * (p * j) = k * k \implies k * k = p * (j * j)$ 
  by (simp add: mult-ac)

```

```

lemma prime-not-square:
   $\text{prime } p \implies (\bigwedge k. 0 < k \implies m * m \neq p * (k * k))$ 
  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    (ℚ) where
ℚ = {x. ∃ m n. n ≠ 0 ∧ |x| = real (m::nat) / real (n::nat)}

```

3.3 Main theorem

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

theorem *prime-sqrt-irrational*:

```

prime p ⇒ x * x = real p ⇒ 0 ≤ x ⇒ x ∉ 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    (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:  $\forall M. \exists N. 0 < N \ \& \ (\forall m. 0 < m \ \& \ (m::nat) \leq M \longrightarrow m \text{ dvd } N)$ 
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:  $(\text{hypnat-of-nat } n \leq 0) = (n = 0)$ 
by (transfer, simp)
declare hypnat-of-nat-le-zero-iff [simp]

lemma hdvd-by-all:  $\forall M. \exists N. 0 < N \ \& \ (\forall m. 0 < m \ \& \ (m::\text{hypnat}) \leq M \longrightarrow m \text{ hdvd } N)$ 
by (transfer, rule dvd-by-all)

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

lemma hypnat-dvd-all-hypnat-of-nat:
   $\exists (N::\text{hypnat}). 0 < N \ \& \ (\forall n \in -\{0::nat\}. \text{hypnat-of-nat}(n) \text{ hdvd } N)$ 
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:
   $\text{starprime} = \{p. 1 < p \ \& \ (\forall m. m \text{ hdvd } p \longrightarrow m = 1 \mid m = p)\}$ 
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]:  $Suc\ 0 < n \longrightarrow (\exists k. \text{prime } k \ \& \ k \text{ dvd } n)$ 
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.  $1 < n \implies (\exists k \in \text{starprime}. k \text{ hdvd } n)$ 
by (transfer, simp add: prime-factor-exists)

```

```

lemma NatStar-hypnat-of-nat:  $\text{finite } A \implies \text{star}^* A = \text{hypnat-of-nat } A$ 
by (rule starset-finite)

```

4.1 Another characterization of infinite set of natural numbers

```

lemma finite-nat-set-bounded:  $\text{finite } N \implies \exists n. (\forall i \in N. i < (n::nat))$ 
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:  $\text{finite } N = (\exists n. (\forall i \in N. i < (n::nat)))$ 
by (blast intro: finite-nat-set-bounded bounded-nat-set-is-finite)

```

```

lemma not-finite-nat-set-iff:  $(\sim \text{finite } N) = (\forall n. \exists i \in N. n \leq (i::nat))$ 
by (auto simp add: finite-nat-set-bounded-iff not-less)

```

```

lemma bounded-nat-set-is-finite2:  $(\forall i \in N. i \leq (n::nat)) \implies \text{finite } N$ 
apply (rule finite-subset)
apply (rule-tac [2] finite-atMost, auto)
done

```

```

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

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

4.2 An injective function cannot define an embedded natural number

lemma *lemma-infinite-set-singleton*: $\forall m n. m \neq n \longrightarrow f n \neq f m$
 $\implies \{n. f n = N\} = \{\} \mid (\exists m. \{n. f n = N\} = \{m\})$
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*: $\text{inj } (f::\text{nat} \Rightarrow \text{nat})$
shows $\text{starfun } f \text{ whn} \notin \text{Nats}$
proof
from *inj-f* **have** *inj-f'*: $\text{inj } (\text{starfun } f)$
by (*transfer inj-on-def Ball-def UNIV-def*)
assume $\text{starfun } f \text{ whn} \in \text{Nats}$
then obtain *N* **where** $N: \text{starfun } f \text{ whn} = \text{hypnat-of-nat } N$
by (*auto simp add: Nats-def*)
hence $\exists n. \text{starfun } f n = \text{hypnat-of-nat } N$..
hence $\exists n. f n = N$ **by** *transfer*
then obtain *n* **where** $n: f n = N$..
hence $\text{starfun } f (\text{hypnat-of-nat } n) = \text{hypnat-of-nat } N$
by *transfer*
with *N* **have** $\text{starfun } f \text{ whn} = \text{starfun } f (\text{hypnat-of-nat } n)$
by *simp*
with *inj-f'* **have** $\text{whn} = \text{hypnat-of-nat } n$
by (*rule injD*)
thus *False*
by (*simp add: whn-neq-hypnat-of-nat*)
qed

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

```

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

```

```

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

```

```

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

```

```

lemma injf-max-order-preserving2:  $\forall x. \exists y \in E. x < y \implies \forall n \ m. m < n \longrightarrow \text{injf-max } m \ E < \text{injf-max } n \ E$ 
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  $x = \text{injf-max } m \ E$  in order-less-trans, auto)
done

```

```

lemma inj-injf-max:  $\forall x. \exists y \in E. x < y \implies \text{inj } (\%n. \text{injf-max } n \ E)$ 
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)) ≠ {}; ∀ x. ∃ y ∈ E. x < y |]
  ==> ∃ f. range f <= E & inj (f::nat => 'a) & (∀ 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:
  ~ finite A ==> hypnat-of-nat ' A < ( *s* A)
apply auto
apply (subgoal-tac A ≠ {})
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: (~ 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: ~ (∀ n ∈ - {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: ∃ n ∈ - {0}. ~ 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 ≠ 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: ¬ 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 ∉ starprime
by (transfer, simp)
declare hypnat-of-nat-zero-not-prime [simp]

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

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

lemma one-not-prime2: ¬ 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 ∉ starprime
by (transfer, simp)
declare hypnat-of-nat-one-not-prime [simp]

lemma hypnat-one-not-prime: 1 ∉ 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: fun-diff-def)
  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∈{..

```

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*:

$0 < m \implies (2::nat)^\wedge m - 2^\wedge(m - 1) = 2^\wedge(m - 1)$

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*:

$\forall m > 0. (\sum n \in \{(2::nat)^\wedge(m - 1) + 1 .. 2^\wedge m\}. 1/\text{real } n) \geq 1/2$

(**is** $\forall m > 0. (\sum n \in (?S\ m). 1/\text{real } n) \geq 1/2$)

proof

fix $m::nat$

obtain tm **where** $tm\text{def}: tm = (2::nat)^\wedge m$ **by** *simp*

{

assume $mgt0: 0 < m$

have $\bigwedge x. x \in (?S\ m) \implies 1/(\text{real } x) \geq 1/(\text{real } tm)$

proof –

fix $x::nat$

assume $xs: x \in (?S\ m)$

have $xgt0: x > 0$

proof –

from xs **have**

$x \geq 2^{(m-1)} + 1$ **by** *auto*
moreover with *mgt0* **have**
 $2^{(m-1)} + 1 \geq (1::nat)$ **by** *auto*
ultimately have
 $x \geq 1$ **by** (*rule xtrans*)
thus *?thesis* **by** *simp*
qed
moreover from *xs* **have** $x \leq 2^m$ **by** *auto*
ultimately have
 $\text{inverse } (real\ x) \geq \text{inverse } (real\ ((2::nat)^m))$ **by** *simp*
moreover
from *xgt0* **have** $real\ x \neq 0$ **by** *simp*
then have
 $\text{inverse } (real\ x) = 1 / (real\ x)$
by (*rule nonzero-inverse-eq-divide*)
moreover from *mgt0* **have** $real\ tm \neq 0$ **by** (*simp add: tmdef*)
then have
 $\text{inverse } (real\ tm) = 1 / (real\ tm)$
by (*rule nonzero-inverse-eq-divide*)
ultimately show
 $1/(real\ x) \geq 1/(real\ tm)$ **by** (*auto simp add: tmdef*)
qed
then have
 $(\sum_{n \in (?S\ m)}. 1 / real\ n) \geq (\sum_{n \in (?S\ m)}. 1/(real\ tm))$
by (*rule setsum-mono*)
moreover have
 $(\sum_{n \in (?S\ m)}. 1/(real\ tm)) = 1/2$
proof –
have
 $(\sum_{n \in (?S\ m)}. 1/(real\ tm)) =$
 $(1/(real\ tm)) * (\sum_{n \in (?S\ m)}. 1)$
by *simp*
also have
 $\dots = ((1/(real\ tm)) * real\ (card\ (?S\ m)))$
by (*simp add: real-of-card real-of-nat-def*)
also have
 $\dots = ((1/(real\ tm)) * real\ (tm - (2^{(m-1)})))$
by (*simp add: tmdef*)
also from *mgt0* **have**
 $\dots = ((1/(real\ tm)) * real\ ((2::nat)^{(m-1)}))$
by (*auto simp: tmdef dest: two-pow-sub*)
also have
 $\dots = (real\ (2::nat))^{(m-1)} / (real\ (2::nat))^m$
by (*simp add: tmdef realpow-real-of-nat [symmetric]*)
also from *mgt0* **have**
 $\dots = (real\ (2::nat))^{(m-1)} / (real\ (2::nat))^{((m-1) + 1)}$
by *auto*
also have $\dots = 1/2$ **by** *simp*
finally show *?thesis* .


```

qed
ultimately have
   $(\sum_{n \in (?S\ m)}. 1 / \text{real } n) \geq 1/2$ 
  by  $-(erule\ subst)$ 
}
thus  $0 < m \longrightarrow 1 / 2 \leq (\sum_{n \in (?S\ m)}. 1 / \text{real } n)$  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]:

```

 $0 < M \implies (\sum_{n \in \{1..(2::nat)^\wedge M\}}. 1 / \text{real } n) =$ 
 $(1 + (\sum_{m \in \{1..M\}}. \sum_{n \in \{(2::nat)^\wedge(m-1)+1..2^\wedge m\}}. 1 / \text{real } n))$ 
(is  $0 < M \implies ?LHS\ M = ?RHS\ M$ )

```

proof (induct M)

case 0 show ?case by simp

next

case (Suc M)

have ant: $0 < \text{Suc } M$ by fact

{

have suc: $?LHS\ (\text{Suc } M) = ?RHS\ (\text{Suc } M)$

proof cases — show that LHS = c and RHS = c, and thus LHS = RHS

assume mz: $M=0$

{

then have

$?LHS\ (\text{Suc } M) = ?LHS\ 1$ by simp

also have

$\dots = (\sum_{n \in \{(1::nat)..2\}}. 1 / \text{real } n)$ by simp

also have

$\dots = ((\sum_{n \in \{\text{Suc } 1..2\}}. 1 / \text{real } n) + 1 / (\text{real } (1::nat)))$

by (subst setsum-head)

(auto simp: atLeastSucAtMost-greaterThanAtMost)

also have

$\dots = ((\sum_{n \in \{2..2::nat\}}. 1 / \text{real } n) + 1 / (\text{real } (1::nat)))$

by (simp add: nat-number)

also have

$\dots = 1 / (\text{real } (2::nat)) + 1 / (\text{real } (1::nat))$ by simp

finally have

$?LHS\ (\text{Suc } M) = 1/2 + 1$ by simp

}

moreover

{

from mz have

$?RHS\ (\text{Suc } M) = ?RHS\ 1$ by simp

also have

$\dots = (\sum_{n \in \{((2::nat)^\wedge 0)+1..2^\wedge 1\}}. 1 / \text{real } n) + 1$

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  $\forall (M::nat). (\sum n \in \{1..(2::nat) \wedge M\}. 1 / \text{real } n) \geq 1 + (\text{real } M)/2$ 
  (is  $\forall M. ?P M \geq -$ )
proof (rule allI, cases)
  fix M::nat
  assume M=0
  then show ?P M  $\geq 1 + (\text{real } M)/2$  by simp
next
  fix M::nat
  assume M $\neq 0$ 
  then have M > 0 by simp
  then have
    (?P M) =
       $(1 + (\sum m \in \{1..M\}. \sum n \in \{(2::nat) \wedge (m - 1) + 1..2 \wedge m\}. 1/\text{real } n))$ 
    by (rule harmonic-aux2)
  also have
    ...  $\geq (1 + (\sum m \in \{1..M\}. 1/2))$ 
  proof -
    let ?f =  $(\lambda x. 1/2)$ 
    let ?g =  $(\lambda x. (\sum n \in \{(2::nat) \wedge (x - 1) + 1..2 \wedge x\}. 1/\text{real } n))$ 
    from harmonic-aux have  $\bigwedge x. x \in \{1..M\} \implies ?f x \leq ?g x$  by simp
    then have  $(\sum m \in \{1..M\}. ?g m) \geq (\sum m \in \{1..M\}. ?f m)$  by (rule setsum-mono)
    thus ?thesis by simp
  qed
  finally have  $(?P M) \geq (1 + (\sum m \in \{1..M\}. 1/2))$  .
  moreover
  {
    have
       $(\sum m \in \{1..M\}. (1::\text{real})/2) = 1/2 * (\sum m \in \{1..M\}. 1)$ 
      by auto
    also have
      ... =  $1/2 * (\text{real } (\text{card } \{1..M\}))$ 
      by (simp only: real-of-card[symmetric])
    also have
      ... =  $1/2 * (\text{real } M)$  by simp
    also have
      ... =  $(\text{real } M)/2$  by simp
  }

```

finally have $(\sum_{m \in \{1..M\}}. (1::\text{real})/2) = (\text{real } M)/2$.
 }
 ultimately show $(?P \ M) \geq (1 + (\text{real } M)/2)$ 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* ($\llbracket \text{summable } ?f; \forall m \geq ?n. 0 < ?f \ m \rrbracket \implies \text{setsum } ?f \ \{0..<?n\} < \text{suminf } ?f$) 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 $\neg \text{summable } (\lambda n. 1/\text{real } (\text{Suc } n))$
 (is $\neg \text{summable } ?f$)

proof — by contradiction

let $?s = \text{suminf } ?f$ — let $?s$ equal the sum of the harmonic series

assume $sf: \text{summable } ?f$

then obtain $n::\text{nat}$ where $n\text{def}: n = \text{nat } \lceil 2 * ?s \rceil$ by simp

then have $\text{ngt}: 1 + \text{real } n/2 > ?s$

proof —

have $\forall n. 0 \leq ?f \ n$ by simp

with sf have $?s \geq 0$

by — (rule *suminf-0-le*, *simp-all*)

then have $\text{cgt0}: \lceil 2 * ?s \rceil \geq 0$ by simp

from $n\text{def}$ have $n = \text{nat } \lceil (2 * ?s) \rceil$.

then have $\text{real } n = \text{real } (\text{nat } \lceil 2 * ?s \rceil)$ by simp

with cgt0 have $\text{real } n = \text{real } \lceil 2 * ?s \rceil$

by (auto dest: *real-nat-eq-real*)

then have $\text{real } n \geq 2 * (?s)$ by simp

then have $\text{real } n/2 \geq (?s)$ by simp

then show $1 + \text{real } n/2 > (?s)$ by simp

qed

obtain j where $j\text{def}: j = (2::\text{nat})^n$ by simp

have $\forall m \geq j. 0 < ?f \ m$ by simp

with sf have $(\sum_{i \in \{0..<j\}}. ?f \ i) < ?s$ by (rule *series-pos-less*)

then have $(\sum_{i \in \{1..<\text{Suc } j\}}. 1/(\text{real } i)) < ?s$

apply —

apply (subst(*asm*) *setsum-shift-bounds-Suc-ivl* [*symmetric*])

by simp

with $j\text{def}$ have

$(\sum_{i \in \{1..<\text{Suc } ((2::\text{nat})^n)\}}. 1 / (\text{real } i)) < ?s$ by simp

then have

$(\sum_{i \in \{1..(2::\text{nat})^n\}}. 1 / (\text{real } i)) < ?s$

by (simp only: *atLeastLessThanSuc-atLeastAtMost*)

moreover from *harmonic-aux3* have

$(\sum_{i \in \{1..(2::\text{nat})^n\}}. 1 / (\text{real } i)) \geq 1 + \text{real } n/2$ by simp

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

```

```

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

```

```

lemma nat2-to-int2-surj:  $\exists f::(\text{nat}*\text{nat})\Rightarrow(\text{int}*\text{int}). \text{surj } f$ 
proof -
  from nat-to-int-surj obtain  $g::\text{nat}\Rightarrow\text{int}$  where surj g ..
  hence  $\text{aux}: \forall y. \exists x. y = g\ x$  by (unfold surj-def)
  let  $?f = \lambda n. (g\ (\text{fst } n), g\ (\text{snd } n))$ 
  {
    fix  $y::(\text{int}*\text{int})$ 
    from aux have  $\exists x1\ x2. \text{fst } y = g\ x1 \wedge \text{snd } y = g\ x2$  by auto
    hence  $\exists x. \text{fst } y = g\ (\text{fst } x) \wedge \text{snd } y = g\ (\text{snd } x)$  by auto
    hence  $\exists x. (\text{fst } y, \text{snd } y) = (g\ (\text{fst } x), g\ (\text{snd } x))$  by blast
    hence  $\exists x. y = ?f\ x$  by auto
  }
  hence  $\forall y. \exists x. y = ?f\ x$  by auto
  hence surj ?f by (fold surj-def)
  thus ?thesis by auto
qed

```

```

lemma rat-denum:
   $\exists f::\text{nat}\Rightarrow\text{rat}. \text{surj } f$ 
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  $h::(\text{nat}*\text{nat})\Rightarrow(\text{int}*\text{int})$  where surj h ..
  ultimately have surj (h o (inv nat2-to-nat)) by (rule comp-surj)
  hence  $\exists f::\text{nat}\Rightarrow(\text{int}*\text{int}). \text{surj } f$  by auto
  then obtain  $g::\text{nat}\Rightarrow(\text{int}*\text{int})$  where surj g by auto
  hence  $g\text{def}: \forall y. \exists x. y = g\ x$  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  $\exists x. y = (split Fract o g) x$  ..
}
hence surj (split Fract o g)
  by (simp add: surj-def)
thus ?thesis by blast
qed

```

end

11 Pretty integer literals for code generation

```

theory Code-Integer
imports ATP-Linkup
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 (Numeral.add-code @{const-name number-int-inst.number-of-int}
    true true) [SML, OCaml, Haskell]
  ⟩⟩

```

```

code-const Int.Pls and Int.Min and Int.Bit0 and Int.Bit1
  (SML raise/ Fail/ Pls
    and raise/ Fail/ Min
    and !((-)/ raise/ Fail/ Bit0)
    and !((-)/ raise/ Fail/ Bit1))
  (OCaml failwith/ Pls
    and failwith/ Min
    and !((-)/ failwith/ Bit0)
    and !((-)/ failwith/ Bit1))
  (Haskell error/ Pls
    and error/ Min

```

```

and error/ Bit0
and error/ Bit1)

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

code-const Int.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-reserved SML IntInf

```


code-reserved *OCaml Big-int*

end

12 Quatifier elimination for $R(0,1,+, \text{floor}, i)$

theory *MIR*

imports *Real GCD Code-Integer*

uses (*mireif.ML*) (*mirtac.ML*)

begin

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

primrec *alluopairs*:: 'a list \Rightarrow ('a \times 'a) list **where**

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*:

$\llbracket x \in \text{set } xs ; y \in \text{set } xs \rrbracket \Longrightarrow (x,y) \in \text{set } (\text{alluopairs } xs) \vee (y,x) \in \text{set } (\text{alluopairs } xs)$

by (induct xs, auto)

lemma *alluopairs-ex*:

assumes *Pc*: $\forall x y. P x y = P y x$

shows $(\exists x \in \text{set } xs. \exists y \in \text{set } xs. P x y) = (\exists (x,y) \in \text{set } (\text{alluopairs } xs). P x y)$

proof

assume $\exists x \in \text{set } xs. \exists y \in \text{set } xs. P x y$

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** $\exists (x,y) \in \text{set } (\text{alluopairs } xs). P x y$

by auto

next

assume $\exists (x,y) \in \text{set } (\text{alluopairs } xs). P x y$

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** $\exists x \in \text{set } xs. \exists y \in \text{set } xs. P x y$ **by** blast

qed

consts *iupt* :: int \times int \Rightarrow int list

recdef *iupt measure* ($\lambda (i,j). \text{nat } (j-i+1)$)

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:  $0 < n \implies (x \# xs) ! n = xs ! (n - 1)$ 
using Nat.gr0-conv-Suc
by clarsimp

```

```

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

```

```

lemma myless:  $\forall (a::'a::\{pordered-ab-group-add\}) (b::'a). (a < b) = (0 < b - a)$ 
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  $\dots = (- (y - x) < 0)$  by simp
also have  $\dots = (0 < y - x)$  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:  $(\text{real } n \leq x \wedge x < \text{real } (n+1)) = (\text{floor } x = n)$ 
proof (auto)
assume lb:  $\text{real } n \leq x$ 
and ub:  $x < \text{real } n + 1$ 
have  $\text{real } (\text{floor } x) \leq x$  by simp
hence  $\text{real } (\text{floor } x) < \text{real } (n + 1)$  using ub by arith
hence  $\text{floor } x < n+1$  by simp
moreover from lb have  $n \leq \text{floor } x$  using floor-mono2 [where x=real n and y=x]
by simp ultimately show  $\text{floor } x = n$  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  $\forall x.\forall k. ((a::int) dvd (x + t)) = (a dvd (x+k*d + t))$  by simp
  then show ?thesis by simp
qed

```

definition

```

rdvd:: real  $\Rightarrow$  real  $\Rightarrow$  bool (infixl rdvd 50)
where
  rdvd-def: x rdvd y  $\longleftrightarrow$  ( $\exists k::int. y = x * real\ k$ )

```

lemma int-rdvd-real:

```

shows real (i::int) rdvd x = (i dvd (floor x)  $\wedge$  real (floor x) = x) (is ?l = ?r)
proof
  assume ?l
  hence th:  $\exists k. x = real\ (i*k)$  by (simp add: rdvd-def)
  hence th': real (floor x) = x by (auto simp del: real-of-int-mult)
  with th have  $\exists k. real\ (floor\ x) = real\ (i*k)$  by simp
  hence  $\exists k. floor\ x = i*k$  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  $\exists k. real\ (floor\ x) = real\ (i*k)$ 
  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

```

```

primrec 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

```

primrec *Inum* :: *real list* \Rightarrow *num* \Rightarrow *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* \equiv *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* \implies *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* \implies *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 $\text{isint } (\text{Sub } (C \ c) \ e) \ bs$ **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 $?a = \text{Inum } bs \ a$

let $?b = \text{Inum } bs \ b$

from *ai bi isint-iff* **have** $\text{real } (\text{floor } (?a + ?b)) = \text{real } (\text{floor } (\text{real } (\text{floor } ?a) + \text{real } (\text{floor } ?b)))$ **by** *simp*

also have $\dots = \text{real } (\text{floor } ?a) + \text{real } (\text{floor } ?b)$ **by** *simp*

also have $\dots = ?a + ?b$ **using** *ai bi isint-iff* **by** *simp*

finally show $\text{isint } (\text{Add } a \ b) \ bs$ **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*

primrec *Ifm* :: *real list* \Rightarrow *fm* \Rightarrow *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* \leq 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: ∧ bs. Ifm bs (prep p) = Ifm bs p
by (induct p rule: prep.induct, auto)

```

fun qfree :: fm ⇒ bool **where**

```

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

```

primrec numbound0 :: num ⇒ bool **where**

```

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) (auto simp add: nth-pos2)

```

lemma numbound0-gen:

```

assumes nb: numbound0 t and ti: isint t (x#bs)
shows ∀ 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 :: fm ⇒ bool **where**

```

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 ∧ bound0 q)

```


$\mid \text{bound0 } (\text{Or } p \ q) = (\text{bound0 } p \wedge \text{bound0 } q)$
 $\mid \text{bound0 } (\text{Imp } p \ q) = ((\text{bound0 } p) \wedge (\text{bound0 } q))$
 $\mid \text{bound0 } (\text{Iff } p \ q) = (\text{bound0 } p \wedge \text{bound0 } q)$
 $\mid \text{bound0 } (E \ p) = \text{False}$
 $\mid \text{bound0 } (A \ p) = \text{False}$

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*) (*auto simp add: nth-pos2*)

primrec *numsubst0*:: *num* \Rightarrow *num* \Rightarrow *num* **where**

numsubst0 t (C c) = (C c)

$\mid \text{numsubst0 } t \ (\text{Bound } n) = (\text{if } n=0 \text{ then } t \text{ else } \text{Bound } n)$

$\mid \text{numsubst0 } t \ (\text{CN } n \ i \ a) = (\text{if } n=0 \text{ then } \text{Add } (\text{Mul } i \ t) \ (\text{numsubst0 } t \ a) \text{ else } \text{CN } n \ i \ (\text{numsubst0 } t \ a))$

$\mid \text{numsubst0 } t \ (\text{CF } i \ a \ b) = \text{CF } i \ (\text{numsubst0 } t \ a) \ (\text{numsubst0 } t \ b)$

$\mid \text{numsubst0 } t \ (\text{Neg } a) = \text{Neg } (\text{numsubst0 } t \ a)$

$\mid \text{numsubst0 } t \ (\text{Add } a \ b) = \text{Add } (\text{numsubst0 } t \ a) \ (\text{numsubst0 } t \ b)$

$\mid \text{numsubst0 } t \ (\text{Sub } a \ b) = \text{Sub } (\text{numsubst0 } t \ a) \ (\text{numsubst0 } t \ b)$

$\mid \text{numsubst0 } t \ (\text{Mul } i \ a) = \text{Mul } i \ (\text{numsubst0 } t \ a)$

$\mid \text{numsubst0 } t \ (\text{Floor } a) = \text{Floor } (\text{numsubst0 } t \ a)$

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*:: *num* \Rightarrow *fm* \Rightarrow *fm* **where**

subst0 t T = T

$\mid \text{subst0 } t \ F = F$

$\mid \text{subst0 } t \ (\text{Lt } a) = \text{Lt } (\text{numsubst0 } t \ a)$

$\mid \text{subst0 } t \ (\text{Le } a) = \text{Le } (\text{numsubst0 } t \ a)$

$\mid \text{subst0 } t \ (\text{Gt } a) = \text{Gt } (\text{numsubst0 } t \ a)$

$\mid \text{subst0 } t \ (\text{Ge } a) = \text{Ge } (\text{numsubst0 } t \ a)$

$\mid \text{subst0 } t \ (\text{Eq } a) = \text{Eq } (\text{numsubst0 } t \ a)$

$\mid \text{subst0 } t \ (\text{NEq } a) = \text{NEq } (\text{numsubst0 } t \ a)$

$\mid \text{subst0 } t \ (\text{Dvd } i \ a) = \text{Dvd } i \ (\text{numsubst0 } t \ a)$

$\mid \text{subst0 } t \ (\text{NDvd } i \ a) = \text{NDvd } i \ (\text{numsubst0 } t \ a)$

$\mid \text{subst0 } t \ (\text{NOT } p) = \text{NOT } (\text{subst0 } t \ p)$

$\mid \text{subst0 } t \ (\text{And } p \ q) = \text{And } (\text{subst0 } t \ p) \ (\text{subst0 } t \ q)$

$\mid \text{subst0 } t \ (\text{Or } p \ q) = \text{Or } (\text{subst0 } t \ p) \ (\text{subst0 } t \ q)$

$\mid \text{subst0 } t \ (\text{Imp } p \ q) = \text{Imp } (\text{subst0 } t \ p) \ (\text{subst0 } t \ q)$

| $\text{subst0 } t \text{ (Iff } p \text{ } q) = \text{Iff } (\text{subst0 } t \text{ } p) (\text{subst0 } t \text{ } q)$

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

consts

decrnum:: *num* \Rightarrow *num*

decr :: *fm* \Rightarrow *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 $\text{Inum } (x\#bs) \text{ } t = \text{Inum } bs \text{ } (\text{decrnum } t)$

using *nb* **by** (*induct t* *rule: decrnum.induct, simp-all add: nth-pos2*)

lemma *decr*: **assumes** *nb*: *bound0 p*

shows $\text{Ifm } (x\#bs) \text{ } p = \text{Ifm } bs \text{ } (\text{decr } p)$

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 (*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*, *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*, *auto*)

lemma *bound0-qf*: *bound0* *p* \Longrightarrow *qfree* *p*

by (*induct* *p*, *simp-all*)

definition *djf*:: (*'a* \Rightarrow *fm*) \Rightarrow *'a* \Rightarrow *fm* \Rightarrow *fm* **where**

djf *f* *p* *q* = (*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*))

definition *evaldjf*:: (*'a* \Rightarrow *fm*) \Rightarrow *'a* *list* \Rightarrow *fm* **where**

evaldjf *f* *ps* = *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*: $\forall x \in \text{set } xs. \text{qfree } (f x)$
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*: $(\exists q \in \text{set } (\text{disjuncts } p). \text{Ifm } bs \ q) = \text{Ifm } bs \ p$

by(*induct p rule: disjuncts.induct, auto*)

lemma *conjuncts*: $(\forall q \in \text{set } (\text{conjuncts } p). \text{Ifm } bs \ q) = \text{Ifm } bs \ p$

by(*induct p rule: conjuncts.induct, auto*)

lemma *disjuncts-nb*: $\text{bound0 } p \implies \forall q \in \text{set } (\text{disjuncts } p). \text{bound0 } q$

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*: $\text{bound0 } p \implies \forall q \in \text{set } (\text{conjuncts } p). \text{bound0 } q$

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*: $\text{qfree } p \implies \forall q \in \text{set } (\text{disjuncts } p). \text{qfree } q$

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*: $\text{qfree } p \implies \forall q \in \text{set } (\text{conjuncts } p). \text{qfree } q$

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':  $\forall q \in set (disjuncts p). qfree (f q)$  by blast

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

lemma DJ-qe: assumes qe:  $\forall bs p. qfree p \longrightarrow qfree (qe p) \wedge (Ifm bs (qe p) =$ 
  Ifm bs (E p))
  shows  $\forall bs p. qfree p \longrightarrow qfree (DJ qe p) \wedge (Ifm bs ((DJ qe p)) = Ifm bs (E$ 
  p))
proof(clarify)
  fix p::fm and bs
  assume qf: qfree p
  from qe have qth:  $\forall p. qfree p \longrightarrow qfree (qe p)$  by blast
  from DJ-qf[OF qth] qf have qfth: qfree (DJ qe p) by auto
  have Ifm bs (DJ qe p) =  $(\exists q \in set (disjuncts p). Ifm bs (qe q))$ 
    by (simp add: DJ-def evaldjf-ex)
  also have ... =  $(\exists q \in set (disjuncts p). Ifm bs (E q))$  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)  $\wedge$  Ifm bs (DJ qe p) = Ifm bs (E p) using qfth by
    blast
qed

consts bnds:: num  $\Rightarrow$  nat list
  lex-ns:: nat list  $\times$  nat list  $\Rightarrow$  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 ( $\lambda$  (xs,ys). length xs + length ys)
  lex-ns ([], ms) = True
  lex-ns (ns, []) = False
  lex-ns (n#ns, m#ms) = (n<m  $\vee$  ((n = m)  $\wedge$  lex-ns (ns,ms)))
constdefs lex-bnd :: num  $\Rightarrow$  num  $\Rightarrow$  bool
  lex-bnd t s  $\equiv$  lex-ns (bnds t, bnds s)

consts
  numgcdh:: num  $\Rightarrow$  int  $\Rightarrow$  int
  reducecoeffh:: num  $\Rightarrow$  int  $\Rightarrow$  num
  dvdnumcoeff:: num  $\Rightarrow$  int  $\Rightarrow$  bool
consts maxcoeff:: num  $\Rightarrow$  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) = ( $\lambda$  g. igcd i g)
  numgcdh (CN n c t) = ( $\lambda$  g. igcd c (numgcdh t g))
  numgcdh (CF c s t) = ( $\lambda$  g. igcd c (numgcdh t g))
  numgcdh t = ( $\lambda$  g. 1)

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

recdef reducecoeffh measure size
  reducecoeffh (C i) = ( $\lambda$  g. C (i div g))
  reducecoeffh (CN n c t) = ( $\lambda$  g. CN n (c div g) (reducecoeffh t g))
  reducecoeffh (CF c s t) = ( $\lambda$  g. CF (c div g) s (reducecoeffh t g))
  reducecoeffh t = ( $\lambda$  g. t)

```

definition

$\text{reducecoeff} :: \text{num} \Rightarrow \text{num}$

where

$\text{reducecoeff-def: reducecoeff } t =$
 $(\text{let } g = \text{numgcd } t \text{ in}$
 $\text{if } g = 0 \text{ then } C \ 0 \text{ else if } g=1 \text{ then } t \text{ else reducecoeffh } t \ g)$

recdef dvdnumcoeff *measure size*

$\text{dvdnumcoeff } (C \ i) = (\lambda \ g. \ g \ \text{dvd} \ i)$
 $\text{dvdnumcoeff } (CN \ n \ c \ t) = (\lambda \ g. \ g \ \text{dvd} \ c \wedge (\text{dvdnumcoeff } t \ g))$
 $\text{dvdnumcoeff } (CF \ c \ s \ t) = (\lambda \ g. \ g \ \text{dvd} \ c \wedge (\text{dvdnumcoeff } t \ g))$
 $\text{dvdnumcoeff } t = (\lambda \ g. \ \text{False})$

lemma dvdnumcoeff-trans :

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

declare zdvd-trans $[\text{trans add}]$ **lemma** $\text{natabs0: } (\text{nat } (\text{abs } x) = 0) = (x = 0)$

by arith

lemma numgcd0 :

assumes $\text{g0: numgcd } t = 0$
shows $\text{Inum } bs \ t = 0$

proof–

have $\bigwedge x. \text{numgcdh } t \ x = 0 \implies \text{Inum } bs \ t = 0$
by $(\text{induct } t \text{ rule: numgcdh.induct, auto simp add: igcd-def gcd-zero natabs0 max-def maxcoeff-pos})$
thus $?thesis$ **using** $\text{g0[simplified numgcd-def]}$ **by** blast
qed

lemma $\text{numgcdh-pos: assumes } \text{gp: } g \geq 0 \text{ shows } \text{numgcdh } t \ g \geq 0$

using gp
by $(\text{induct } t \text{ rule: numgcdh.induct, auto simp add: igcd-def})$

lemma $\text{numgcd-pos: numgcd } t \geq 0$

by $(\text{simp add: numgcd-def numgcdh-pos maxcoeff-pos})$

lemma reducecoeffh :

assumes $\text{gt: dvdnumcoeff } t \ g \text{ and } \text{gp: } g > 0$
shows $\text{real } g * (\text{Inum } bs \ (\text{reducecoeffh } t \ g)) = \text{Inum } bs \ t$
using gt

proof $(\text{induct } t \text{ rule: reducecoeffh.induct})$

case $(1 \ i)$ **hence** $\text{gd: } g \ \text{dvd} \ i$ **by** simp
from gp **have** $\text{gnz: } g \neq 0$ **by** simp
from prems **show** $?case$ **by** $(\text{simp add: real-of-int-div}[OF \ \text{gnz gd}])$

```

next
  case (2 n c t) hence gd: g dvd c by simp
  from gp have gnz: g ≠ 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 ≠ 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 ⇒ c ≤ c' ⇒ 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*: $\text{real } (\text{numgcd } t) * (\text{Inum } bs (\text{reducecoeff } t)) = \text{Inum } bs t$

proof –

let $?g = \text{numgcd } t$
 have $?g \geq 0$ **by** (*simp add: numgcd-pos*)
 hence $?g = 0 \vee ?g = 1 \vee ?g > 1$ **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** $th1: \text{dvdnumcoeff } t ?g$ **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*: $\text{numbound0 } t \implies \text{numbound0 } (\text{reducecoeffh } t g)$
by (*induct t rule: reducecoeffh.induct, auto*)

lemma *reducecoeff-numbound0*: $\text{numbound0 } t \implies \text{numbound0 } (\text{reducecoeff } t)$
using *reducecoeffh-numbound0* **by** (*simp add: reducecoeff-def Let-def*)

consts

simpnum:: $\text{num} \Rightarrow \text{num}$
numadd:: $\text{num} \times \text{num} \Rightarrow \text{num}$
nummul:: $\text{num} \Rightarrow \text{int} \Rightarrow \text{num}$

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

lemma *numadd[simp]*: $\text{Inum } bs (\text{numadd } (t,s)) = \text{Inum } bs (\text{Add } t \ s)$
apply (*induct t s rule: numadd.induct, simp-all add: Let-def*)

```

apply (case-tac  $c1+c2 = 0$ , case-tac  $n1 \leq n2$ , 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]:  $\llbracket \text{numbound0 } t ; \text{numbound0 } s \rrbracket \implies \text{numbound0 } (\text{numadd } (t,s))$ 
by (induct  $t\ s$  rule: numadd.induct, auto simp add: Let-def)

```

```

recdef nummul measure size
  nummul ( $C\ j$ ) = ( $\lambda\ i.$   $C\ (i*j)$ )
  nummul ( $CN\ n\ c\ t$ ) = ( $\lambda\ i.$   $CN\ n\ (c*i)\ (\text{nummul } t\ i)$ )
  nummul ( $CF\ c\ t\ s$ ) = ( $\lambda\ i.$   $CF\ (c*i)\ t\ (\text{nummul } s\ i)$ )
  nummul ( $Mul\ c\ t$ ) = ( $\lambda\ i.$   $\text{nummul } t\ (i*c)$ )
  nummul  $t$  = ( $\lambda\ i.$   $Mul\ i\ t$ )

```

```

lemma nummul[simp]:  $\bigwedge\ i.$   $\text{Inum } bs\ (\text{nummul } t\ i) = \text{Inum } bs\ (Mul\ i\ t)$ 
by (induct  $t$  rule: nummul.induct, auto simp add: ring-simps)

```

```

lemma nummul-nb[simp]:  $\bigwedge\ i.$   $\text{numbound0 } t \implies \text{numbound0 } (\text{nummul } t\ i)$ 
by (induct  $t$  rule: nummul.induct, auto)

```

```

constdefs numneg ::  $\text{num} \Rightarrow \text{num}$ 
  numneg  $t \equiv \text{nummul } t\ (-1)$ 

```

```

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

```

```

lemma numneg[simp]:  $\text{Inum } bs\ (\text{numneg } t) = \text{Inum } bs\ (\text{Neg } t)$ 
using numneg-def nummul by simp

```

```

lemma numneg-nb[simp]:  $\text{numbound0 } t \implies \text{numbound0 } (\text{numneg } t)$ 
using numneg-def by simp

```

```

lemma numsub[simp]:  $\text{Inum } bs\ (\text{numsub } a\ b) = \text{Inum } bs\ (\text{Sub } a\ b)$ 
using numsub-def by simp

```

```

lemma numsub-nb[simp]:  $\llbracket \text{numbound0 } t ; \text{numbound0 } s \rrbracket \implies \text{numbound0 } (\text{numsub } t\ s)$ 
using numsub-def by simp

```

```

lemma isint-CF: assumes  $si:$   $\text{isint } s\ bs$  shows  $\text{isint } (CF\ c\ t\ s)\ bs$ 
proof–
  have  $cti:$   $\text{isint } (Mul\ c\ (\text{Floor } t))\ bs$  by (simp add: isint-Mul isint-Floor)

```

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

consts $split-int:: num \Rightarrow num \times num$

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 $tv ti : split-int\ t = (?tv, ?ti)$ **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 tv ti]* **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 $\dots = real\ (floor\ (?N\ ?tv) + (floor\ (?N\ ?ti)))$
 by (*simp, subst tii[simplified isint-iff, symmetric]*) *simp*
 also have $\dots = ?N\ (Add\ (Floor\ ?tv)\ ?ti)$ **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 tv ti]* **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 $\dots = real\ (floor\ (?N\ ?tv) + (floor\ (?N\ ?ti)))$
 by (*simp, subst tii[simplified isint-iff, symmetric]*) *simp*
 also have $\dots = ?N\ (Add\ (Floor\ ?tv)\ ?ti)$ **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]*: $numbound0\ t \implies numbound0\ (numfloor\ t)$
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]*:
 $numbound0\ t \implies numbound0\ (simpnum\ t)$

```

by (induct t rule: simpnum.induct, auto)

consts nozerocoeff :: num  $\Rightarrow$  bool
recdef nozerocoeff measure size
  nozerocoeff (C c) = True
  nozerocoeff (CN n c t) = (c  $\neq$  0  $\wedge$  nozerocoeff t)
  nozerocoeff (CF c s t) = (c  $\neq$  0  $\wedge$  nozerocoeff t)
  nozerocoeff (Mul c t) = (c  $\neq$  0  $\wedge$  nozerocoeff t)
  nozerocoeff t = True

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

lemma nummul-nz :  $\bigwedge i. i \neq 0 \implies$  nozerocoeff a  $\implies$  nozerocoeff (nummul a i)
by (induct a rule: nummul.induct, auto simp add: Let-def numadd-nz)

lemma numneg-nz : nozerocoeff a  $\implies$  nozerocoeff (numneg a)
by (simp add: numneg-def nummul-nz)

lemma numsub-nz : nozerocoeff a  $\implies$  nozerocoeff b  $\implies$  nozerocoeff (numsub a b)
by (simp add: numsub-def numneg-nz numadd-nz)

lemma split-int-nz : nozerocoeff t  $\implies$  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  $\implies$  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  $\implies$  maxcoeff t = 0  $\implies$  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 prem1 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 prem1 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 (reduceceffh 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 = reduceceffh ?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 reduceceffh[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 ∨ ?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' ≤ 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 ⇒ 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  $\implies$  qfree (not p)
by (induct p, auto)
lemma not-nb[simp]: bound0 p  $\implies$  bound0 (not p)
by (induct 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)

lemma conj-qf[simp]:  $\llbracket$ qfree p ; qfree q $\rrbracket \implies$  qfree (conj p q)
using conj-def by auto
lemma conj-nb[simp]:  $\llbracket$ bound0 p ; bound0 q $\rrbracket \implies$  bound0 (conj p q)
using conj-def by auto

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)
lemma disj-qf[simp]:  $\llbracket$ qfree p ; qfree q $\rrbracket \implies$  qfree (disj p q)
using disj-def by auto
lemma disj-nb[simp]:  $\llbracket$ bound0 p ; bound0 q $\rrbracket \implies$  bound0 (disj p q)
using disj-def by auto

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)
lemma imp-qf[simp]:  $\llbracket$ qfree p ; qfree q $\rrbracket \implies$  qfree (imp p q)
using imp-def by (cases p=F  $\vee$  q=T, simp-all add: imp-def)
lemma imp-nb[simp]:  $\llbracket$ bound0 p ; bound0 q $\rrbracket \implies$  bound0 (imp p q)
using imp-def by (cases p=F  $\vee$  q=T  $\vee$  p=q, 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 add:not)
(cases not p= q, auto simp add:not)
lemma iff-ql[simp]:  $\llbracket \text{qfree } p ; \text{qfree } q \rrbracket \implies \text{qfree } (\text{iff } p \text{ } q)$ 
  by (unfold iff-def, cases p=q, auto)
lemma iff-nb[simp]:  $\llbracket \text{bound0 } p ; \text{bound0 } q \rrbracket \implies \text{bound0 } (\text{iff } p \text{ } q)$ 
using iff-def by (unfold iff-def, cases p=q, auto)

consts check-int:: num  $\Rightarrow$  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  $\wedge$  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  $\implies$  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  $\lfloor t \rfloor = t \implies 1 \text{ rdvd } t$ 
  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  $\neq$  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  $\Rightarrow$  num  $\Rightarrow$  (int  $\times$  num)
  simpdvd d t  $\equiv$ 
    (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  $\neq$  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  $\neg$  ?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'  $\neq$  0 by simp
    hence g'p: ?g' > 0 using igcd-pos[where i=d and j=numgcd t] by arith
    hence ?g' = 1  $\vee$  ?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  $\Rightarrow$  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)

```

$\text{simpfm } (Lt \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v < 0) \text{ then } T$
 $\text{else } F$
 $\mid - \Rightarrow Lt \ (\text{reducecoeff } a')$
 $\text{simpfm } (Le \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v \leq 0) \text{ then } T$
 $\text{else } F \mid - \Rightarrow Le \ (\text{reducecoeff } a'))$
 $\text{simpfm } (Gt \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v > 0) \text{ then } T$
 $\text{else } F \mid - \Rightarrow Gt \ (\text{reducecoeff } a'))$
 $\text{simpfm } (Ge \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v \geq 0) \text{ then } T$
 $\text{else } F \mid - \Rightarrow Ge \ (\text{reducecoeff } a'))$
 $\text{simpfm } (Eq \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v = 0) \text{ then } T$
 $\text{else } F \mid - \Rightarrow Eq \ (\text{reducecoeff } a'))$
 $\text{simpfm } (NEq \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v \neq 0) \text{ then } T$
 $\text{else } F \mid - \Rightarrow NEq \ (\text{reducecoeff } a'))$
 $\text{simpfm } (Dvd \ i \ a) = (\text{if } i=0 \text{ then } \text{simpfm } (Eq \ a)$
 $\text{else if } (\text{abs } i = 1) \wedge \text{check-int } a \text{ then } T$
 $\text{else let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (i \text{ dvd } v) \text{ then } T \text{ else } F$
 $\mid - \Rightarrow (\text{let } (d,t) = \text{simpdvd } i \ a' \text{ in } Dvd \ d \ t))$
 $\text{simpfm } (NDvd \ i \ a) = (\text{if } i=0 \text{ then } \text{simpfm } (NEq \ a)$
 $\text{else if } (\text{abs } i = 1) \wedge \text{check-int } a \text{ then } F$
 $\text{else let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (\neg(i \text{ dvd } v)) \text{ then } T \text{ else } F$
 $\mid - \Rightarrow (\text{let } (d,t) = \text{simpdvd } i \ a' \text{ in } NDvd \ d \ t))$
 $\text{simpfm } p = p$

lemma $\text{simpfm}[\text{simp}]$: $\text{Ifm } bs \ (\text{simpfm } p) = \text{Ifm } bs \ p$

proof(*induct* p rule: simpfm.induct)

case (6 a) **let** $?sa = \text{simpnum } a$ **have** $sa: \text{Inum } bs \ ?sa = \text{Inum } bs \ a$ **by** simp
{fix v **assume** $?sa = C \ v$ **hence** $?case$ **using** sa **by** simp **}**

moreover **{assume** $H: \neg (\exists \ v. \ ?sa = C \ v)$

let $?g = \text{numgcd } ?sa$

let $?rsa = \text{reducecoeff } ?sa$

let $?r = \text{Inum } bs \ ?rsa$

have $sa\text{-nz}: \text{nozerocoeff } ?sa$ **by** (rule simpnum-nz)

{assume $gz: ?g=0$ **from** $\text{numgcd-nz}[OF \ sa\text{-nz} \ gz]$ H **have** False **by** auto **}**

with $\text{numgcd-pos}[\text{where } t=?sa]$ **have** $?g > 0$ **by** (cases $?g=0$, auto)

hence $gp: \text{real } ?g > 0$ **by** simp

have $\text{Inum } bs \ ?sa = \text{real } ?g * ?r$ **by** (simp add: reducecoeff)

with sa **have** $\text{Inum } bs \ a < 0 = (\text{real } ?g * ?r < \text{real } ?g * 0)$ **by** simp

also have $\dots = (?r < 0)$ **using** gp

by (simp only: $\text{mult-less-cancel-left}$) simp

finally have $?case$ **using** H **by** (cases $?sa$, simp-all add: Let-def) **}**

ultimately show $?case$ **by** blast

next

case (7 a) **let** $?sa = \text{simpnum } a$ **have** $sa: \text{Inum } bs \ ?sa = \text{Inum } bs \ a$ **by** simp
{fix v **assume** $?sa = C \ v$ **hence** $?case$ **using** sa **by** simp **}**

moreover **{assume** $H: \neg (\exists \ v. \ ?sa = C \ v)$

let $?g = \text{numgcd } ?sa$

let $?rsa = \text{reducecoeff } ?sa$

let $?r = \text{Inum } bs \ ?rsa$

have $sa\text{-nz}: \text{nozerocoeff } ?sa$ **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  $H:\neg (\exists v. ?sa = C v)$ 
      hence  $th: \text{simpfm } (Dvd\ i\ a) = Dvd\ (fst\ (\text{simpdvd}\ i\ ?sa))\ (snd\ (\text{simpdvd}\ i\ ?sa))$  using  $inz\ cond$  by (cases  $?sa$ , auto simp add: Let-def split-def)
      from  $\text{simpnum-nz}$  have  $nz:\text{nozerocoeff}\ ?sa$  by simp
      from  $\text{simpdvd}\ [OF\ nz\ inz]$  th have  $?case$  using  $sa$  by simp}
    ultimately have  $?case$  by blast}
  ultimately show  $?case$  by blast
next
  case (13  $i\ a$ ) let  $?sa = \text{simpnum}\ a$  have  $sa: \text{Inum}\ bs\ ?sa = \text{Inum}\ bs\ a$  by simp
  have  $i=0 \vee (abs\ i = 1 \wedge \text{check-int}\ a) \vee (i \neq 0 \wedge ((abs\ i \neq 1) \vee (\neg \text{check-int}\ a)))$  by auto
  {assume  $i=0$  hence  $?case$  using 13.hyps by (simp add:  $\text{rdvd-left-0-eq}$  Let-def)}
  moreover
  {assume  $ai1: abs\ i = 1$  and  $ai: \text{check-int}\ a$ 
    hence  $i=1 \vee i=-1$  by arith
    moreover {assume  $i1: i = 1$ 
      from  $\text{rdvd-left1-int}[OF\ \text{check-int}[OF\ ai, \text{simplified isint-iff}]]$ 
      have  $?case$  using  $i1\ ai$  by simp }
    moreover {assume  $i1: i = -1$ 
      from  $\text{rdvd-left1-int}[OF\ \text{check-int}[OF\ ai, \text{simplified isint-iff}]]$ 
       $\text{rdvd-abs1}[\text{where } d=-1 \text{ and } t=\text{Inum}\ bs\ a]$ 
      have  $?case$  using  $i1\ ai$  by simp }
    ultimately have  $?case$  by blast}
  moreover
  {assume  $inz: i \neq 0$  and  $cond: (abs\ i \neq 1) \vee (\neg \text{check-int}\ a)$ 
    {fix  $v$  assume  $?sa = C v$  hence  $?case$  using  $sa[\text{symmetric}]$   $inz\ cond$  by (cases  $abs\ i = 1$ , auto simp add:  $\text{int-rdvd-iff}$ ) }
    moreover {assume  $H:\neg (\exists v. ?sa = C v)$ 
      hence  $th: \text{simpfm } (NDvd\ i\ a) = NDvd\ (fst\ (\text{simpdvd}\ i\ ?sa))\ (snd\ (\text{simpdvd}\ i\ ?sa))$  using  $inz\ cond$  by (cases  $?sa$ , auto simp add: Let-def split-def)
      from  $\text{simpnum-nz}$  have  $nz:\text{nozerocoeff}\ ?sa$  by simp
      from  $\text{simpdvd}\ [OF\ nz\ inz]$  th have  $?case$  using  $sa$  by simp}
    ultimately have  $?case$  by blast}
  ultimately show  $?case$  by blast
qed (induct  $p$  rule:  $\text{simpfm.induct}$ ,  $\text{simp-all}$ )

```

lemma simpdvd-numbound0 : $\text{numbound0}\ t \implies \text{numbound0}\ (snd\ (\text{simpdvd}\ d\ t))$
 by (simp add: simpdvd-def Let-def split-def $\text{reducecoeffh-numbound0}$)

lemma $\text{simpfm-bound0}[\text{simp}]$: $\text{bound0}\ p \implies \text{bound0}\ (\text{simpfm}\ p)$

proof(induct p rule: simpfm.induct)

case (6 a) hence $nb: \text{numbound0}\ a$ by simp

hence $\text{numbound0}\ (\text{simpnum}\ a)$ by (simp only: $\text{simpnum-numbound0}[OF\ nb]$)

thus $?case$ by (cases $\text{simpnum}\ a$, auto simp add: Let-def $\text{reducecoeffh-numbound0}$)

next

case (7 a) hence $nb: \text{numbound0}\ a$ by simp

hence $\text{numbound0}\ (\text{simpnum}\ a)$ by (simp only: $\text{simpnum-numbound0}[OF\ nb]$)

```

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  $\implies$  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  $\Rightarrow$  fm
  list-conj ps  $\equiv$  foldr conj ps T
lemma list-conj: Ifm bs (list-conj ps) = ( $\forall p \in$  set ps. Ifm bs p)
  by (induct ps, auto simp add: list-conj-def)
lemma list-conj-qf:  $\forall p \in$  set ps. qfree p  $\implies$  qfree (list-conj ps)
  by (induct ps, auto simp add: list-conj-def)
lemma list-conj-nb:  $\forall p \in$  set ps. bound0 p  $\implies$  bound0 (list-conj ps)
  by (induct ps, auto simp add: list-conj-def)
constdefs CJNB:: (fm  $\Rightarrow$  fm)  $\Rightarrow$  fm  $\Rightarrow$  fm
  CJNB f p  $\equiv$  (let cjs = conjuncts p ; (yes,no) = partition bound0 cjs
    in conj (decr (list-conj yes)) (f (list-conj no)))

```

lemma CJNB-qe:


```

assumes  $qe: \forall bs\ p. qfree\ p \longrightarrow qfree\ (qe\ p) \wedge (Ifm\ bs\ (qe\ p) = Ifm\ bs\ (E\ p))$ 
shows  $\forall bs\ p. qfree\ p \longrightarrow qfree\ (CJNB\ qe\ p) \wedge (Ifm\ bs\ ((CJNB\ qe\ p)) = Ifm\ bs\ (E\ p))$ 
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  $\forall q \in set\ ?yes. bound0\ q$  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  $\forall q \in set\ ?no. qfree\ q$  by auto
  hence  $no-qf: qfree\ ?cno$  by (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  $Ifm\ bs\ p = (\forall q \in set\ ?cjs. Ifm\ bs\ q)$  by blast
    also have  $\dots = ((\forall q \in set\ ?yes. Ifm\ bs\ q) \wedge (\forall q \in set\ ?no. Ifm\ bs\ q))$ 
      using  $partition-set[OF\ part]$  by auto
    finally have  $Ifm\ bs\ p = ((Ifm\ bs\ ?cyes) \wedge (Ifm\ bs\ ?cno))$  using  $list-conj$  by
simp
    hence  $Ifm\ bs\ (E\ p) = (\exists x. (Ifm\ (x\ \#bs)\ ?cyes) \wedge (Ifm\ (x\ \#bs)\ ?cno))$  by simp
    also fix  $y$  have  $\dots = (\exists x. (Ifm\ (y\ \#bs)\ ?cyes) \wedge (Ifm\ (x\ \#bs)\ ?cno))$ 
      using  $bound0-I[OF\ yes-nb, where\ bs=bs\ and\ b'=y]$  by blast
    also have  $\dots = (Ifm\ bs\ (decr\ ?cyes) \wedge Ifm\ bs\ (E\ ?cno))$ 
      by (auto simp add: decr[OF\ yes-nb])
    also have  $\dots = (Ifm\ bs\ (conj\ (decr\ ?cyes)\ (qe\ ?cno)))$ 
      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  $qfree\ (CJNB\ qe\ p) \wedge Ifm\ bs\ (CJNB\ qe\ p) = Ifm\ bs\ (E\ p)$  by blast
  }
qed

```

```

consts  $qelim :: fm \Rightarrow (fm \Rightarrow fm) \Rightarrow fm$ 
recdef  $qelim\ measure\ fmsize$ 
   $qelim\ (E\ p) = (\lambda\ qe. DJ\ (CJNB\ qe)\ (qelim\ p\ qe))$ 
   $qelim\ (A\ p) = (\lambda\ qe. not\ (qe\ ((qelim\ (NOT\ p)\ qe))))$ 
   $qelim\ (NOT\ p) = (\lambda\ qe. not\ (qelim\ p\ qe))$ 
   $qelim\ (And\ p\ q) = (\lambda\ qe. conj\ (qelim\ p\ qe)\ (qelim\ q\ qe))$ 
   $qelim\ (Or\ p\ q) = (\lambda\ qe. disj\ (qelim\ p\ qe)\ (qelim\ q\ qe))$ 
   $qelim\ (Imp\ p\ q) = (\lambda\ qe. disj\ (qelim\ (NOT\ p)\ qe)\ (qelim\ q\ qe))$ 
   $qelim\ (Iff\ p\ q) = (\lambda\ qe. iff\ (qelim\ p\ qe)\ (qelim\ q\ qe))$ 

```

$qelim\ p = (\lambda\ y.\ simpfm\ p)$

lemma *qelim-ci*:

assumes *qe-inv*: $\forall\ bs\ p.\ qfree\ p \longrightarrow qfree\ (qe\ p) \wedge (Ifm\ bs\ (qe\ p) = Ifm\ bs\ (E\ p))$

shows $\bigwedge\ bs.\ qfree\ (qelim\ p\ qe) \wedge (Ifm\ bs\ (qelim\ p\ qe) = Ifm\ bs\ p)$

using *qe-inv DJ-qe[OF CJNB-qe[OF qe-inv]]*

by(*induct p rule: qelim.induct*)

(*auto simp del: simpfm.simps*)

The \mathbb{Z} Part

Linearity for fm where Bound 0 ranges over \mathbb{Z}

consts

zsplit0 :: *num* \Rightarrow *int* \times *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 $\bigwedge\ n\ a.\ zsplit0\ t = (n, a) \Longrightarrow (Inum\ ((real\ (x::int))\ \#bs)\ (CN\ 0\ n\ a) = Inum\ (real\ x\ \#bs)\ t) \wedge numbound0\ a$

(**is** $\bigwedge\ n\ a.\ ?S\ t = (n, a) \Longrightarrow (?I\ x\ (CN\ 0\ n\ a) = ?I\ x\ t) \wedge ?N\ a$)

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* \wedge *n*=*-?nt* **using** *prems*

by (*simp add: Let-def split-def*)

```

from abj prems have th2: ( $?I\ x\ (CN\ 0\ ?nt\ ?at) = ?I\ x\ t$ )  $\wedge\ ?N\ ?at$  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  $\wedge\ n$ =?ns + ?nt using prems
    by (simp add: Let-def split-def)
  from abjs[symmetric] have bluddy:  $\exists\ x\ y. (x,y) = zsplit0\ s$  by blast
  from prems have ( $\exists\ x\ y. (x,y) = zsplit0\ s$ )  $\longrightarrow$  ( $\forall\ xa\ xb. zsplit0\ t = (xa, xb)$ )
 $\longrightarrow$  Inum (real x # bs) (CN 0 xa xb) = Inum (real x # bs) t  $\wedge$  numbound0 xb)
by simp
  with bluddy abjt have th3: ( $?I\ x\ (CN\ 0\ ?nt\ ?at) = ?I\ x\ t$ )  $\wedge\ ?N\ ?at$  by blast
  from abjs prems have th2: ( $?I\ x\ (CN\ 0\ ?ns\ ?as) = ?I\ x\ s$ )  $\wedge\ ?N\ ?as$  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  $\wedge\ n$ =?ns - ?nt using prems
    by (simp add: Let-def split-def)
  from abjs[symmetric] have bluddy:  $\exists\ x\ y. (x,y) = zsplit0\ s$  by blast
  from prems have ( $\exists\ x\ y. (x,y) = zsplit0\ s$ )  $\longrightarrow$  ( $\forall\ xa\ xb. zsplit0\ t = (xa, xb)$ )
 $\longrightarrow$  Inum (real x # bs) (CN 0 xa xb) = Inum (real x # bs) t  $\wedge$  numbound0 xb)
by simp
  with bluddy abjt have th3: ( $?I\ x\ (CN\ 0\ ?nt\ ?at) = ?I\ x\ t$ )  $\wedge\ ?N\ ?at$  by blast
  from abjs prems have th2: ( $?I\ x\ (CN\ 0\ ?ns\ ?as) = ?I\ x\ s$ )  $\wedge\ ?N\ ?as$  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  $\wedge\ n$ =i*?nt
using prems
    by (simp add: Let-def split-def)
  from abj prems have th2: ( $?I\ x\ (CN\ 0\ ?nt\ ?at) = ?I\ x\ t$ )  $\wedge\ ?N\ ?at$  by blast
  hence  $?I\ x\ (Mul\ i\ t) = (real\ i) * ?I\ x\ (CN\ 0\ ?nt\ ?at)$  by simp
  also have  $\dots = ?I\ x\ (CN\ 0\ (i*?nt)\ (Mul\ i\ ?at))$  by (simp add: right-distrib)
  finally show ?case using th th2 by simp

```

```

next
  case (g t n a)
  let ?nt = fst (zsplitt0 t)
  let ?at = snd (zsplitt0 t)
  have abj: zsplitt0 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

```

iszfml :: fm ⇒ real list ⇒ bool
zfm :: fm ⇒ fm

```

recdef iszfml measure size

```

iszfml (And p q) = (λ bs. iszfml p bs ∧ iszfml q bs)
iszfml (Or p q) = (λ bs. iszfml p bs ∧ iszfml q bs)
iszfml (Eq (CN 0 c e)) = (λ bs. c>0 ∧ numbound0 e ∧ isint e bs)
iszfml (NEq (CN 0 c e)) = (λ bs. c>0 ∧ numbound0 e ∧ isint e bs)
iszfml (Lt (CN 0 c e)) = (λ bs. c>0 ∧ numbound0 e ∧ isint e bs)
iszfml (Le (CN 0 c e)) = (λ bs. c>0 ∧ numbound0 e ∧ isint e bs)
iszfml (Gt (CN 0 c e)) = (λ bs. c>0 ∧ numbound0 e ∧ isint e bs)
iszfml (Ge (CN 0 c e)) = (λ bs. c>0 ∧ numbound0 e ∧ isint e bs)
iszfml (Dvd i (CN 0 c e)) =
  (λ bs. c>0 ∧ i>0 ∧ numbound0 e ∧ isint e bs)
iszfml (NDvd i (CN 0 c e)) =
  (λ bs. c>0 ∧ i>0 ∧ numbound0 e ∧ isint e bs)
iszfml p = (λ bs. isatom p ∧ (bound0 p))

```

lemma zlin-qfree: iszfml p bs ⇒ qfree p
 by (induct p rule: iszfml.induct) auto

lemma iszfml-gen:

```

assumes lp: iszfml p (x#bs)
shows ∀ y. iszfml p (y#bs)

```

proof

```

fix y
show iszfml 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 \implies iszlfm q bs \implies iszlfm (conj p q) bs*

using *conj-def by (cases p,auto)*

lemma *disj-zl[simp]: iszlfm p bs \implies iszlfm q bs \implies iszlfm (disj p q) bs*

using *disj-def by (cases p,auto)*

lemma *not-zl[simp]: iszlfm p bs \implies 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) = zsplt0 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) = zsplt0 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) = zsplt0 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) = zsplt0 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) = zsplt0 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) = zsplt0 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) = zsplt0 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) = zsplt0 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 \vee (a = floor b \wedge real (floor b) < b))

proof(auto)

assume alb: real a < b **and** agb: \neg a < floor b

from agb **have** floor b \leq 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) \leq 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 \vee (real a - real (floor (-b)) = 0 \wedge real (floor (-b)) + b < 0))

proof–

have $(\text{real } a + b < 0) = (\text{real } a < -b)$ **by** *arith*
with *split-int-less-real*[**where** $a=a$ **and** $b=-b$] **show** *?thesis* **by** *arith*
qed

lemma *split-int-gt-real'*:

$(\text{real } (a::\text{int}) + b > 0) = (\text{real } a + \text{real } (\text{floor } b) > 0 \vee (\text{real } a + \text{real } (\text{floor } b) = 0 \wedge \text{real } (\text{floor } b) - b < 0))$

proof–

have *th*: $(\text{real } a + b > 0) = (\text{real } (-a) + (-b) < 0)$ **by** *arith*
show *?thesis* **using** *myless*[*rule-format*, **where** $b=\text{real } (\text{floor } b)$]
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*:

$(\text{real } (a::\text{int}) \leq b) = (a \leq \text{floor } b \vee (a = \text{floor } b \wedge \text{real } (\text{floor } b) < b))$

proof(*auto*)

assume *alb*: $\text{real } a \leq b$ **and** *agb*: $\neg a \leq \text{floor } b$
from *alb* **have** $\text{floor } (\text{real } a) \leq \text{floor } b$ **by** (*simp only*: *floor-mono2*)
hence $a \leq \text{floor } b$ **by** *simp* **with** *agb* **show** *False* **by** *simp*
next
assume *alb*: $a \leq \text{floor } b$
hence $\text{real } a \leq \text{real } (\text{floor } b)$ **by** (*simp only*: *floor-mono2*)
also have $\dots \leq b$ **by** *simp* **finally show** $\text{real } a \leq b$.
qed

lemma *split-int-le-real'*:

$(\text{real } (a::\text{int}) + b \leq 0) = (\text{real } a - \text{real } (\text{floor } (-b)) \leq 0 \vee (\text{real } a - \text{real } (\text{floor } (-b)) = 0 \wedge \text{real } (\text{floor } (-b)) + b < 0))$

proof–

have $(\text{real } a + b \leq 0) = (\text{real } a \leq -b)$ **by** *arith*
with *split-int-le-real*[**where** $a=a$ **and** $b=-b$] **show** *?thesis* **by** *arith*
qed

lemma *split-int-ge-real'*:

$(\text{real } (a::\text{int}) + b \geq 0) = (\text{real } a + \text{real } (\text{floor } b) \geq 0 \vee (\text{real } a + \text{real } (\text{floor } b) = 0 \wedge \text{real } (\text{floor } b) - b < 0))$

proof–

have *th*: $(\text{real } a + b \geq 0) = (\text{real } (-a) + (-b) \leq 0)$ **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*: $(\text{real } (a::\text{int}) = b) = (a = \text{floor } b \wedge b = \text{real } (\text{floor } b))$
(is ?l = ?r)
by *auto*

lemma *split-int-eq-real'*: $(\text{real } (a::\text{int}) + b = 0) = (a - \text{floor } (-b) = 0 \wedge \text{real } (\text{floor } (-b)) + b = 0)$

```

(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) ∧ iszlfm (zlfm p) (real
(i::int) #bs)
  (is (?I (?l p) = ?I p) ∧ ?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 ∨ (?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 (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) ≥ 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 ≠ 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) ≥ 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 ∨ (?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 (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≠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 $d \mid Dvd\ d\ c*x+t \in p\ d$ δ checks if a given l divides all the
 ds above

consts

```

plusinf:: fm ⇒ fm
minusinf:: fm ⇒ fm
δ :: fm ⇒ int
dδ :: 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 $p \implies$ 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  $\delta$  measure size
   $\delta$  (And p q) = ilcm ( $\delta$  p) ( $\delta$  q)
   $\delta$  (Or p q) = ilcm ( $\delta$  p) ( $\delta$  q)
   $\delta$  (Dvd i (CN 0 c e)) = i
   $\delta$  (NDvd i (CN 0 c e)) = i
   $\delta$  p = 1

recdef d $\delta$  measure size
  d $\delta$  (And p q) = ( $\lambda$  d. d $\delta$  p d  $\wedge$  d $\delta$  q d)
  d $\delta$  (Or p q) = ( $\lambda$  d. d $\delta$  p d  $\wedge$  d $\delta$  q d)
  d $\delta$  (Dvd i (CN 0 c e)) = ( $\lambda$  d. i dvd d)
  d $\delta$  (NDvd i (CN 0 c e)) = ( $\lambda$  d. i dvd d)
  d $\delta$  p = ( $\lambda$  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  $\delta$  : assumes lin:iszlfm p bs
  shows d $\delta$  p ( $\delta$  p)  $\wedge$   $\delta$  p > 0
using lin
proof (induct p rule: iszlfm.induct)
  case (1 p q)
  let ?d =  $\delta$  (And p q)
  from prems ilcm-pos have dp: ?d > 0 by simp
  have d1:  $\delta$  p dvd  $\delta$  (And p q) using prems by simp
  hence th: d $\delta$  p ?d
    using delta-mono prems by (auto simp del: dvd-ilcm-self1)
  have  $\delta$  q dvd  $\delta$  (And p q) using prems by simp
  hence th': d $\delta$  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 =  $\delta$  (And p q)
  from prems ilcm-pos have dp: ?d > 0 by simp
  have  $\delta$  p dvd  $\delta$  (And p q) using prems by simp hence th: d $\delta$  p ?d using
  delta-mono prems

```

```

    by (auto simp del: dvd-ilcm-self1)
    have  $\delta \mid q \mid \delta \mid (And \ p \ q)$  using prems by simp hence  $th': d \mid \delta \mid q \mid d$  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:  $\forall \ x < z1. ?I \ x \ ( ?M \ f) = ?I \ x \ f$  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  $\forall \ x < ?z. ?I \ x \ ( ?M \ (Or \ f \ g)) = ?I \ x \ (Or \ f \ g)$  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
      fix y
      have  $\forall \ x < (floor \ (- \ (Inum \ (y \ \# \ bs) \ e) \ / \ (real \ c))). ?I \ x \ ( ?M \ (Eq \ (CN \ 0 \ c \ e)))$ 
      = ?I x (Eq (CN 0 c e))
      proof (simp add: less-floor-eq , rule allI, rule impI)
        fix x
        assume A:  $real \ x + (1::real) \leq - \ (Inum \ (y \ \# \ bs) \ e \ / \ real \ c)$ 
        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  $real \ c * real \ x + Inum \ (y \ \# \ bs) \ e \neq 0$  using rcpos by simp
        thus  $real \ c * real \ x + Inum \ (real \ x \ \# \ bs) \ e \neq 0$ 
        using numbound0-I[OF nbe, where b=y and bs=bs and b'=real x] by simp
      qed
      thus ?case by blast
    end

```



```

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
  fix y
  have  $\forall x < (\text{floor } (- (\text{Inum } (y \# bs) e) / (\text{real } c))). ?I x (?M (NEq (CN 0 c e))) = ?I x (NEq (CN 0 c e))$ 
  proof (simp add: less-floor-eq, rule allI, rule impI)
    fix x
    assume A:  $\text{real } x + (1::\text{real}) \leq - (\text{Inum } (y \# bs) e / \text{real } c)$ 
    hence th1:  $\text{real } x < - (\text{Inum } (y \# bs) e / \text{real } c)$  by simp
    with rcpos have  $(\text{real } c) * (\text{real } x) < (\text{real } c) * (- (\text{Inum } (y \# bs) e / \text{real } c))$ 
      by (simp only: real-mult-less-mono2[OF rcpos th1])
    hence  $\text{real } c * \text{real } x + \text{Inum } (y \# bs) e \neq 0$  using rcpos by simp
    thus  $\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) e \neq 0$ 
      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
  fix y
  have  $\forall x < (\text{floor } (- (\text{Inum } (y \# bs) e) / (\text{real } c))). ?I x (?M (Lt (CN 0 c e))) = ?I x (Lt (CN 0 c e))$ 
  proof (simp add: less-floor-eq, rule allI, rule impI)
    fix x
    assume A:  $\text{real } x + (1::\text{real}) \leq - (\text{Inum } (y \# bs) e / \text{real } c)$ 
    hence th1:  $\text{real } x < - (\text{Inum } (y \# bs) e / \text{real } c)$  by simp
    with rcpos have  $(\text{real } c) * (\text{real } x) < (\text{real } c) * (- (\text{Inum } (y \# bs) e / \text{real } c))$ 
      by (simp only: real-mult-less-mono2[OF rcpos th1])
    thus  $\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) e < 0$ 
      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
  fix y
  have  $\forall x < (\text{floor } (- (\text{Inum } (y \# bs) e) / (\text{real } c))). ?I x (?M (Le (CN 0 c e))) = ?I x (Le (CN 0 c e))$ 
  proof (simp add: less-floor-eq, rule allI, rule impI)
    fix x
    assume A:  $\text{real } x + (1::\text{real}) \leq - (\text{Inum } (y \# bs) e / \text{real } c)$ 
    hence th1:  $\text{real } x < - (\text{Inum } (y \# bs) e / \text{real } c)$  by simp
    with rcpos have  $(\text{real } c) * (\text{real } x) < (\text{real } c) * (- (\text{Inum } (y \# bs) e / \text{real } c))$ 
      by (simp only: real-mult-less-mono2[OF rcpos th1])

```

```

      thus  $\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# \text{bs}) \text{ } e \leq 0$ 
      using numbound0-I[OF nbe, where  $b=y$  and  $bs=bs$  and  $b'=\text{real } x$ ] rcpos
by simp
qed
thus ?case by blast
next
case (7 c e)
from prems have  $c > 0$  by simp hence rcpos:  $\text{real } c > 0$  by simp
from prems have nbe: numbound0 e by simp
fix y
have  $\forall x < (\text{floor } (- (\text{Inum } (y \# \text{bs}) \text{ } e) / (\text{real } c))). ?I \text{ } x \text{ } (?M \text{ } (Gt \text{ } (CN \text{ } 0 \text{ } c \text{ } e)))$ 
= ?I x (Gt (CN 0 c e))
proof (simp add: less-floor-eq, rule allI, rule impI)
  fix x
  assume A:  $\text{real } x + (1::\text{real}) \leq - (\text{Inum } (y \# \text{bs}) \text{ } e / \text{real } c)$ 
  hence th1:  $\text{real } x < - (\text{Inum } (y \# \text{bs}) \text{ } e / \text{real } c)$  by simp
  with rcpos have  $(\text{real } c) * (\text{real } x) < (\text{real } c) * (- (\text{Inum } (y \# \text{bs}) \text{ } e / \text{real } c))$ 
  by (simp only: real-mult-less-mono2[OF rcpos th1])
  thus  $\neg (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# \text{bs}) \text{ } e > 0)$ 
  using numbound0-I[OF nbe, where  $b=y$  and  $bs=bs$  and  $b'=\text{real } x$ ] rcpos
by simp
qed
thus ?case by blast
next
case (8 c e)
from prems have  $c > 0$  by simp hence rcpos:  $\text{real } c > 0$  by simp
from prems have nbe: numbound0 e by simp
fix y
have  $\forall x < (\text{floor } (- (\text{Inum } (y \# \text{bs}) \text{ } e) / (\text{real } c))). ?I \text{ } x \text{ } (?M \text{ } (Ge \text{ } (CN \text{ } 0 \text{ } c \text{ } e)))$ 
= ?I x (Ge (CN 0 c e))
proof (simp add: less-floor-eq, rule allI, rule impI)
  fix x
  assume A:  $\text{real } x + (1::\text{real}) \leq - (\text{Inum } (y \# \text{bs}) \text{ } e / \text{real } c)$ 
  hence th1:  $\text{real } x < - (\text{Inum } (y \# \text{bs}) \text{ } e / \text{real } c)$  by simp
  with rcpos have  $(\text{real } c) * (\text{real } x) < (\text{real } c) * (- (\text{Inum } (y \# \text{bs}) \text{ } e / \text{real } c))$ 
  by (simp only: real-mult-less-mono2[OF rcpos th1])
  thus  $\neg \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# \text{bs}) \text{ } e \geq 0$ 
  using numbound0-I[OF nbe, where  $b=y$  and  $bs=bs$  and  $b'=\text{real } x$ ] rcpos
by simp
qed
thus ?case by blast
qed simp-all

lemma minusinf-repeats:
  assumes d:  $d \delta \text{ } p \text{ } d$  and linp:  $\text{iszlfm } p \text{ } (a \# \text{bs})$ 
  shows  $\text{Ifm } ((\text{real}(x - k*d)) \# \text{bs}) \text{ } (\text{minusinf } p) = \text{Ifm } (\text{real } x \# \text{bs}) \text{ } (\text{minusinf } p)$ 
using linp d
proof(induct p rule: iszlfm.induct)
  case (9 i c e) hence nbe: numbound0 e and id:  $i \text{ } dvd \text{ } d$  by simp+

```

hence $\exists k. d=i*k$ by (simp add: dvd-def)
 then obtain di where $di\text{-def}: d=i*di$ 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 $\exists (l::int). ?rt = ?ri * (real l)$ by (simp add: rdvd-def)
 hence $\exists (l::int). ?rc*?rx + ?I x e = ?ri*(real l) + ?rc*(?rk * (real i) * (real$
 $di))$
 by (simp add: ring-simps di-def)
 hence $\exists (l::int). ?rc*?rx + ?I x e = ?ri*(real (l + c*k*di))$
 by (simp add: ring-simps)
 hence $\exists (l::int). ?rc*?rx + ?I x e = ?ri * (real l)$ 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 $\exists (l::int). ?rc*?rx + ?e = ?ri * (real l)$ by (simp add: rdvd-def)
 hence $\exists (l::int). ?rc*?rx - real c * (real k * real d) + ?e = ?ri * (real l) -$
 $real c * (real k * real d)$ by simp
 hence $\exists (l::int). ?rc*?rx - real c * (real k * real d) + ?e = ?ri * (real l) -$
 $real c * (real k * real i * real di)$ by (simp add: di-def)
 hence $\exists (l::int). ?rc*?rx - real c * (real k * real d) + ?e = ?ri * (real (l -$
 $c*k*di))$ by (simp add: ring-simps)
 hence $\exists (l::int). ?rc*?rx - real c * (real k * real d) + ?e = ?ri * (real l)$
 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 $\exists k. d=i*k$ by (simp add: dvd-def)
 then obtain di where $di\text{-def}: d=i*di$ 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 $\exists (l::int). ?rt = ?ri * (real l)$ by (simp add: rdvd-def)
 hence $\exists (l::int). ?rc*?rx + ?I x e = ?ri*(real l) + ?rc*(?rk * (real i) * (real$
 $di))$
 by (simp add: ring-simps di-def)
 hence $\exists (l::int). ?rc*?rx + ?I x e = ?ri*(real (l + c*k*di))$
 by (simp add: ring-simps)
 hence $\exists (l::int). ?rc*?rx + ?I x e = ?ri * (real l)$ by blast

thus $\text{real } i \text{ rdvd } \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# \text{bs}) \text{ } e$ **using** *rdvd-def* **by** *simp*
next
assume
 $\text{real } i \text{ rdvd } \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# \text{bs}) \text{ } e$ **(is** $?ri \text{ rdvd } ?rc * ?rx + ?e$ **)**
hence $\exists (l::\text{int}). ?rc * ?rx + ?e = ?ri * (\text{real } l)$ **by** (*simp add: rdvd-def*)
hence $\exists (l::\text{int}). ?rc * ?rx - \text{real } c * (\text{real } k * \text{real } d) + ?e = ?ri * (\text{real } l) -$
 $\text{real } c * (\text{real } k * \text{real } d)$ **by** *simp*
hence $\exists (l::\text{int}). ?rc * ?rx - \text{real } c * (\text{real } k * \text{real } d) + ?e = ?ri * (\text{real } l) -$
 $\text{real } c * (\text{real } k * \text{real } i * \text{real } di)$ **by** (*simp add: di-def*)
hence $\exists (l::\text{int}). ?rc * ?rx - \text{real } c * (\text{real } k * \text{real } d) + ?e = ?ri * (\text{real } (l -$
 $c * k * di))$ **by** (*simp add: ring-simps*)
hence $\exists (l::\text{int}). ?rc * ?rx - \text{real } c * (\text{real } k * \text{real } d) + ?e = ?ri * (\text{real } l)$
by *blast*
thus $\text{real } i \text{ rdvd } \text{real } c * \text{real } x - \text{real } c * (\text{real } k * \text{real } d) + \text{Inum } (\text{real } x \#$
 $\text{bs}) \text{ } e$ **using** *rdvd-def* **by** *simp*
qed
qed (*auto simp add: nth-pos2 numbound0-I* [**where** $\text{bs}=\text{bs}$ **and** $b=\text{real}(x - k*d)$
and $b'=\text{real } x$] *simp del: real-of-int-mult real-of-int-diff*)

lemma *minusinf-ex*:

assumes $\text{lin}: \text{iszlfm } p (\text{real } (a::\text{int}) \# \text{bs})$
and $\text{exmi}: \exists (x::\text{int}). \text{Ifm } (\text{real } x \# \text{bs}) (\text{minusinf } p)$ **(is** $\exists x. ?P1 \text{ } x$ **)**
shows $\exists (x::\text{int}). \text{Ifm } (\text{real } x \# \text{bs}) p$ **(is** $\exists x. ?P \text{ } x$ **)**
proof–
let $?d = \delta \text{ } p$
from δ [*OF lin*] **have** $\text{dpos}: ?d > 0$ **by** *simp*
from δ [*OF lin*] **have** $\text{alld}: d \delta \text{ } p \text{ } ?d$ **by** *simp*
from *minusinf-repeats* [*OF alld lin*] **have** $\text{th1}: \forall x \text{ } k. ?P1 \text{ } x = ?P1 \text{ } (x - (k * ?d))$
by *simp*
from *minusinf-inf* [*OF lin*] **have** $\text{th2}: \exists z. \forall x. x < z \longrightarrow (?P \text{ } x = ?P1 \text{ } x)$ **by** *blast*
from *minusinf-finity* [*OF dpos th1 th2*] exmi **show** $?thesis$ **by** *blast*
qed

lemma *minusinf-bex*:

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

lemma *dvd1-eq1*: $x > 0 \implies (x::\text{int}) \text{ dvd } 1 = (x = 1)$ **by** *auto*

consts

$a\beta :: fm \Rightarrow int \Rightarrow fm$
 $d\beta :: fm \Rightarrow int \Rightarrow bool$
 $\zeta :: fm \Rightarrow int$
 $\beta :: fm \Rightarrow num\ list$
 $\alpha :: fm \Rightarrow num\ list$

recdef $a\beta$ *measure size*

$a\beta (And\ p\ q) = (\lambda\ k. And\ (a\beta\ p\ k)\ (a\beta\ q\ k))$
 $a\beta (Or\ p\ q) = (\lambda\ k. Or\ (a\beta\ p\ k)\ (a\beta\ q\ k))$
 $a\beta (Eq\ (CN\ 0\ c\ e)) = (\lambda\ k. Eq\ (CN\ 0\ 1\ (Mul\ (k\ div\ c)\ e)))$
 $a\beta (NEq\ (CN\ 0\ c\ e)) = (\lambda\ k. NEq\ (CN\ 0\ 1\ (Mul\ (k\ div\ c)\ e)))$
 $a\beta (Lt\ (CN\ 0\ c\ e)) = (\lambda\ k. Lt\ (CN\ 0\ 1\ (Mul\ (k\ div\ c)\ e)))$
 $a\beta (Le\ (CN\ 0\ c\ e)) = (\lambda\ k. Le\ (CN\ 0\ 1\ (Mul\ (k\ div\ c)\ e)))$
 $a\beta (Gt\ (CN\ 0\ c\ e)) = (\lambda\ k. Gt\ (CN\ 0\ 1\ (Mul\ (k\ div\ c)\ e)))$
 $a\beta (Ge\ (CN\ 0\ c\ e)) = (\lambda\ k. Ge\ (CN\ 0\ 1\ (Mul\ (k\ div\ c)\ e)))$
 $a\beta (Dvd\ i\ (CN\ 0\ c\ e)) = (\lambda\ k. Dvd\ ((k\ div\ c)*i)\ (CN\ 0\ 1\ (Mul\ (k\ div\ c)\ e)))$
 $a\beta (NDvd\ i\ (CN\ 0\ c\ e)) = (\lambda\ k. NDvd\ ((k\ div\ c)*i)\ (CN\ 0\ 1\ (Mul\ (k\ div\ c)\ e)))$
 $a\beta\ p = (\lambda\ k. p)$

recdef $d\beta$ *measure size*

$d\beta (And\ p\ q) = (\lambda\ k. (d\beta\ p\ k) \wedge (d\beta\ q\ k))$
 $d\beta (Or\ p\ q) = (\lambda\ k. (d\beta\ p\ k) \wedge (d\beta\ q\ k))$
 $d\beta (Eq\ (CN\ 0\ c\ e)) = (\lambda\ k. c\ dvd\ k)$
 $d\beta (NEq\ (CN\ 0\ c\ e)) = (\lambda\ k. c\ dvd\ k)$
 $d\beta (Lt\ (CN\ 0\ c\ e)) = (\lambda\ k. c\ dvd\ k)$
 $d\beta (Le\ (CN\ 0\ c\ e)) = (\lambda\ k. c\ dvd\ k)$
 $d\beta (Gt\ (CN\ 0\ c\ e)) = (\lambda\ k. c\ dvd\ k)$
 $d\beta (Ge\ (CN\ 0\ c\ e)) = (\lambda\ k. c\ dvd\ k)$
 $d\beta (Dvd\ i\ (CN\ 0\ c\ e)) = (\lambda\ k. c\ dvd\ k)$
 $d\beta (NDvd\ i\ (CN\ 0\ c\ e)) = (\lambda\ k. c\ dvd\ k)$
 $d\beta\ p = (\lambda\ k. True)$

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 ( $\alpha$  p) = (Inum (real i#bs)) 'set ( $\beta$  (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)  $\implies$  iszlfm (mirror p) (a#bs)
by (induct p rule: mirror.induct, auto simp add: isint-neg)

```

```

lemma mirror-d $\beta$ : iszlfm p (a#bs)  $\wedge$  d $\beta$  p 1
   $\implies$  iszlfm (mirror p) (a#bs)  $\wedge$  d $\beta$  (mirror p) 1
by (induct p rule: mirror.induct, auto simp add: isint-neg)

```

```

lemma mirror- $\delta$ : iszlfm p (a#bs)  $\implies$   $\delta$  (mirror p) =  $\delta$  p
by (induct p rule: mirror.induct, auto)

```

```

lemma mirror-ex:
  assumes lp: iszlfm p (real (i::int)#bs)
  shows  $(\exists (x::int). \text{Ifm } (\text{real } x \# bs) (\text{mirror } p)) = (\exists (x::int). \text{Ifm } (\text{real } x \# bs)$ 
p)
  (is  $(\exists x. ?I x ?mp) = (\exists x. ?I x p)$ )
proof(auto)
  fix x assume ?I x ?mp hence ?I (- x) p using mirror[OF lp] by blast
  thus  $\exists x. ?I x p$  by blast
next
  fix x assume ?I x p hence ?I (- x) ?mp
  using mirror[OF lp, where x=- x, symmetric] by auto
  thus  $\exists x. ?I x ?mp$  by blast
qed

```

```

lemma  $\beta$ -numbound0: assumes lp: iszlfm p bs
  shows  $\forall b \in \text{set } (\beta p). \text{numbound0 } b$ 
  using lp by (induct p rule:  $\beta$ .induct, auto)

```

```

lemma d $\beta$ -mono:
  assumes linp: iszlfm p (a #bs)
  and dr: d $\beta$  p l
  and d: l dvd l'
  shows d $\beta$  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 $\forall b \in \text{set } (\alpha p). \text{numbound0 } b \wedge \text{isint } b \text{ (a \# bs)}$
using lp
by(induct p rule: α .induct, auto simp add: isint-add isint-c)

lemma ζ :
assumes linp: iszlfm p (a # bs)
shows $\zeta p > 0 \wedge d\beta p (\zeta p)$
using linp
proof(induct p rule: iszlfm.induct)
 case (1 p q)
 from prems have dl1: $\zeta p \text{ dvd } \text{ilcm } (\zeta p) (\zeta q)$ **by** simp
 from prems have dl2: $\zeta q \text{ dvd } \text{ilcm } (\zeta p) (\zeta q)$ **by** simp
 from prems $d\beta$ -mono[where p = p and l = ζp and l' = $\text{ilcm } (\zeta p) (\zeta q)$]
 $d\beta$ -mono[where p = q and l = ζq and l' = $\text{ilcm } (\zeta p) (\zeta q)$]
 dl1 dl2 **show** ?case **by** (auto simp add: ilcm-pos)
next
 case (2 p q)
 from prems have dl1: $\zeta p \text{ dvd } \text{ilcm } (\zeta p) (\zeta q)$ **by** simp
 from prems have dl2: $\zeta q \text{ dvd } \text{ilcm } (\zeta p) (\zeta q)$ **by** simp
 from prems $d\beta$ -mono[where p = p and l = ζp and l' = $\text{ilcm } (\zeta p) (\zeta q)$]
 $d\beta$ -mono[where p = q and l = ζq and l' = $\text{ilcm } (\zeta p) (\zeta q)$]
 dl1 dl2 **show** ?case **by** (auto simp add: ilcm-pos)
qed (auto simp add: ilcm-pos)

lemma $a\beta$: **assumes** linp: iszlfm p (a # bs) and d: $d\beta p l$ and lp: $l > 0$
shows iszlfm ($a\beta p l$) (a # bs) $\wedge d\beta (a\beta p l) 1 \wedge (\text{Ifm } (\text{real } (l * x) \# \text{bs}) (a\beta p l) = \text{Ifm } ((\text{real } x) \# \text{bs}) p)$
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: $c \leq l$ **by** (simp add: zdvd-imp-le [OF d' lp])
 from cp have cnz: $c \neq 0$ **by** simp
 have c div $c \leq l \text{ div } c$
by (simp add: zdiv-mono1[OF clel cp])
 then have ldcpl: $0 < l \text{ div } c$
by (simp add: zdiv-self[OF cnz])
 have $c * (l \text{ div } c) = c * (l \text{ div } c) + l \text{ mod } c$ **using** d' zdvd-iff-zmod-eq-0[where
 $m=c$ and $n=l$] **by** simp
 hence cl: $c * (l \text{ div } c) = l$ **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) <

$(\text{real } (l \text{ div } c)) * 0)$ by (simp add: ring-simps)
 also have $\dots = (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ } e < 0)$
 using mult-less-0-iff [where $a = \text{real } (l \text{ div } c)$ and $b = \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ } e]$ ldcp by simp
 finally show ?case using numbound0-I[OF be, where $b = \text{real } (l * x)$ and $b' = \text{real } x$ 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 $d': c \text{ dvd } l$ 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 $c \text{ div } c \leq l \text{ div } c$
 by (simp add: zdiv-mono1[OF clel cp])
 then have ldcp: $0 < l \text{ div } c$
 by (simp add: zdiv-self[OF cnz])
 have $c * (l \text{ div } c) = c * (l \text{ div } c) + l \text{ mod } c$ using d' zdvd-iff-zmod-eq-0 [where $m = c$ and $n = l$] by simp
 hence cl: $c * (l \text{ div } c) = l$ using zmod-zdiv-equality [where $a = l$ and $b = c$, symmetric]
 by simp
 hence $(\text{real } l * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) \text{ } e \leq (0 :: \text{real})) =$
 $(\text{real } (c * (l \text{ div } c)) * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) \text{ } e \leq 0)$
 by simp
 also have $\dots = (\text{real } (l \text{ div } c) * (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ } e) \leq$
 $(\text{real } (l \text{ div } c)) * 0)$ by (simp add: ring-simps)
 also have $\dots = (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ } e \leq 0)$
 using mult-le-0-iff [where $a = \text{real } (l \text{ div } c)$ and $b = \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ } e]$ ldcp by simp
 finally show ?case using numbound0-I[OF be, where $b = \text{real } (l * x)$ and $b' = \text{real } x$ 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 $d': c \text{ dvd } l$ 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 $c \text{ div } c \leq l \text{ div } c$
 by (simp add: zdiv-mono1[OF clel cp])
 then have ldcp: $0 < l \text{ div } c$
 by (simp add: zdiv-self[OF cnz])
 have $c * (l \text{ div } c) = c * (l \text{ div } c) + l \text{ mod } c$ using d' zdvd-iff-zmod-eq-0 [where $m = c$ and $n = l$] by simp
 hence cl: $c * (l \text{ div } c) = l$ using zmod-zdiv-equality [where $a = l$ and $b = c$, symmetric]
 by simp
 hence $(\text{real } l * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) \text{ } e > (0 :: \text{real})) =$
 $(\text{real } (c * (l \text{ div } c)) * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) \text{ } e > 0)$
 by simp
 also have $\dots = (\text{real } (l \text{ div } c) * (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ } e) >$
 $(\text{real } (l \text{ div } c)) * 0)$ 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≤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 (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** $a = \text{real } (l \text{ div } c)$ **and** $b = \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ e}$] *ldcp* **by** *simp*
finally show ?*case* **using** *numbound0-I*[*OF* *be*, **where** $b = \text{real } (l * x)$ **and** $b' = \text{real } x$ **and** $bs = bs$] *be isint-Mul*[*OF* *ei*] **by** *simp*
next
case ($_4 \text{ c e}$) **hence** *cp*: $c > 0$ **and** *be*: *numbound0* *e* **and** *ei*: *isint* *e* ($a \# bs$) **and** d' : $c \text{ dvd } l$ **by** *simp*+
from *lp cp* **have** *clel*: $c \leq l$ **by** (*simp add*: *zdvd-imp-le* [*OF* $d' \text{ lp}$])
from *cp* **have** *cnz*: $c \neq 0$ **by** *simp*
have $c \text{ div } c \leq l \text{ div } c$
by (*simp add*: *zdiv-mono1*[*OF* *clel cp*])
then have *ldcp*: $0 < l \text{ div } c$
by (*simp add*: *zdiv-self*[*OF* *cnz*])
have $c * (l \text{ div } c) = c * (l \text{ div } c) + l \text{ mod } c$ **using** d' *zdvd-iff-zmod-eq-0* [**where** $m = c$ **and** $n = l$] **by** *simp*
hence *cl*: $c * (l \text{ div } c) = l$ **using** *zmod-zdiv-equality* [**where** $a = l$ **and** $b = c$, *symmetric*]
by *simp*
hence $(\text{real } l * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) \text{ e} \neq (0 :: \text{real})) =$
 $(\text{real } (c * (l \text{ div } c)) * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) \text{ e} \neq 0)$
by *simp*
also have $\dots = (\text{real } (l \text{ div } c) * (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ e}) \neq$
 $(\text{real } (l \text{ div } c)) * 0)$ **by** (*simp add*: *ring-simps*)
also have $\dots = (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ e} \neq 0)$
using *zero-le-mult-iff* [**where** $a = \text{real } (l \text{ div } c)$ **and** $b = \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ e}$] *ldcp* **by** *simp*
finally show ?*case* **using** *numbound0-I*[*OF* *be*, **where** $b = \text{real } (l * x)$ **and** $b' = \text{real } x$ **and** $bs = bs$] *be isint-Mul*[*OF* *ei*] **by** *simp*
next
case ($_9 \text{ j c e}$) **hence** *cp*: $c > 0$ **and** *be*: *numbound0* *e* **and** *ei*: *isint* *e* ($a \# bs$) **and** jp : $j > 0$ **and** d' : $c \text{ dvd } l$ **by** *simp*+
from *lp cp* **have** *clel*: $c \leq l$ **by** (*simp add*: *zdvd-imp-le* [*OF* $d' \text{ lp}$])
from *cp* **have** *cnz*: $c \neq 0$ **by** *simp*
have $c \text{ div } c \leq l \text{ div } c$
by (*simp add*: *zdiv-mono1*[*OF* *clel cp*])
then have *ldcp*: $0 < l \text{ div } c$
by (*simp add*: *zdiv-self*[*OF* *cnz*])
have $c * (l \text{ div } c) = c * (l \text{ div } c) + l \text{ mod } c$ **using** d' *zdvd-iff-zmod-eq-0* [**where** $m = c$ **and** $n = l$] **by** *simp*
hence *cl*: $c * (l \text{ div } c) = l$ **using** *zmod-zdiv-equality* [**where** $a = l$ **and** $b = c$, *symmetric*]
by *simp*
hence $(\exists (k :: \text{int}). \text{real } l * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) \text{ e} =$
 $(\text{real } (l \text{ div } c) * \text{real } j) * \text{real } k) = (\exists (k :: \text{int}). \text{real } (c * (l \text{ div } c)) * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) \text{ e} =$
 $(\text{real } (l \text{ div } c) * \text{real } j) * \text{real } k)$ **by** *simp*
also have $\dots = (\exists (k :: \text{int}). \text{real } (l \text{ div } c) * (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \#$
 $bs) \text{ e} - \text{real } j * \text{real } k) = \text{real } (l \text{ div } c) * 0)$ **by** (*simp add*: *ring-simps*)
also fix *k* **have** $\dots = (\exists (k :: \text{int}). \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) \text{ e} -$
 $\text{real } j * \text{real } k = 0)$

using zero-le-mult-iff [where $a = \text{real } (l \text{ div } c)$ and $b = \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) e - \text{real } j * \text{real } k$] ldcp **by** simp
also have ... = $(\exists (k::\text{int}). \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) e = \text{real } j * \text{real } k)$ **by** simp
finally show ?case **using** numbound0-I[OF be, where $b = \text{real } (l * x)$ and $b' = \text{real } x$ 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': $c \text{ dvd } l$ **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 $c \text{ div } c \leq l \text{ div } c$
by (simp add: zdiv-mono1[OF clel cp])
then have ldcp: $0 < l \text{ div } c$
by (simp add: zdiv-self[OF cnz])
have $c * (l \text{ div } c) = c * (l \text{ div } c) + l \text{ mod } c$ **using** d' zdvd-iff-zmod-eq-0 [where $m = c$ and $n = l$] **by** simp
hence cl: $c * (l \text{ div } c) = l$ **using** zmod-zdiv-equality [where $a = l$ and $b = c$, symmetric]
by simp
hence $(\exists (k::\text{int}). \text{real } l * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) e = (\text{real } (l \text{ div } c) * \text{real } j) * \text{real } k) = (\exists (k::\text{int}). \text{real } (c * (l \text{ div } c)) * \text{real } x + \text{real } (l \text{ div } c) * \text{Inum } (\text{real } x \# bs) e = (\text{real } (l \text{ div } c) * \text{real } j) * \text{real } k)$ **by** simp
also have ... = $(\exists (k::\text{int}). \text{real } (l \text{ div } c) * (\text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) e - \text{real } j * \text{real } k) = \text{real } (l \text{ div } c) * 0)$ **by** (simp add: ring-simps)
also fix k **have** ... = $(\exists (k::\text{int}). \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) e - \text{real } j * \text{real } k = 0)$
using zero-le-mult-iff [where $a = \text{real } (l \text{ div } c)$ and $b = \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) e - \text{real } j * \text{real } k$] ldcp **by** simp
also have ... = $(\exists (k::\text{int}). \text{real } c * \text{real } x + \text{Inum } (\text{real } x \# bs) e = \text{real } j * \text{real } k)$ **by** simp
finally show ?case **using** numbound0-I[OF be, where $b = \text{real } (l * x)$ and $b' = \text{real } x$ 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 $b = \text{real } (l * x)$ and $b' = \text{real } x$] isint-Mul del: real-of-int-mult)

lemma aβ-ex: **assumes** linp: iszlfm p ($a \# bs$) **and** d: $d \beta p l$ **and** lp: $l > 0$
shows $(\exists x. l \text{ dvd } x \wedge \text{Ifm } (\text{real } x \# bs) (a \beta p l)) = (\exists (x::\text{int}). \text{Ifm } (\text{real } x \# bs) p)$
(is $(\exists x. l \text{ dvd } x \wedge ?P x) = (\exists x. ?P' x)$ **)**
proof –
have $(\exists x. l \text{ dvd } x \wedge ?P x) = (\exists (x::\text{int}). ?P (l * x))$
using unity-coeff-ex [where $l = l$ and $P = ?P$, simplified] **by** simp
also have ... = $(\exists (x::\text{int}). ?P' x)$ **using** aβ [OF linp d lp] **by** simp
finally show ?thesis .
qed

```

lemma  $\beta$ :
  assumes  $lp: \text{iszlfn } p \ (a \# bs)$ 
  and  $u: d \beta \ p \ 1$ 
  and  $d: d \delta \ p \ d$ 
  and  $dp: d > 0$ 
  and  $nob: \neg(\exists(j::\text{int}) \in \{1 \ .. \ d\}. \exists \ b \in (\text{Inum } (a \# bs)) \ ' \ \text{set}(\beta \ p). \ \text{real } x = b +$ 
 $\text{real } j)$ 
  and  $p: \text{Ifm } (\text{real } x \# bs) \ p \ (\text{is } ?P \ x)$ 
  shows  $?P \ (x - d)$ 
using  $lp \ u \ d \ dp \ nob \ p$ 
proof(induct  $p$  rule:  $\text{iszlfn.induct}$ )
  case (5  $c \ e$ ) hence  $c1: c=1$  and  $bn:\text{numbound0 } e$  using  $dvd1\text{-eq1}[\text{where } x=c]$ 
  by  $\text{simp+}$ 
  with  $dp \ p \ c1 \ \text{numbound0-I}[OF \ bn, \text{where } b=\text{real } (x-d) \text{ and } b'=\text{real } x \text{ and } bs=bs]$   $\text{prems}$ 
  show  $?case$  by ( $\text{simp del: real-of-int-minus}$ )
next
  case (6  $c \ e$ ) hence  $c1: c=1$  and  $bn:\text{numbound0 } e$  using  $dvd1\text{-eq1}[\text{where } x=c]$ 
  by  $\text{simp+}$ 
  with  $dp \ p \ c1 \ \text{numbound0-I}[OF \ bn, \text{where } b=\text{real } (x-d) \text{ and } b'=\text{real } x \text{ and } bs=bs]$   $\text{prems}$ 
  show  $?case$  by ( $\text{simp del: real-of-int-minus}$ )
next
  case (7  $c \ e$ ) hence  $p: \text{Ifm } (\text{real } x \# bs) \ (Gt \ (CN \ 0 \ c \ e))$  and  $c1: c=1$  and
 $bn:\text{numbound0 } e$  and  $ie1:\text{isint } e \ (a \# bs)$  using  $dvd1\text{-eq1}[\text{where } x=c]$  by  $\text{simp+}$ 
  let  $?e = \text{Inum } (\text{real } x \# bs) \ e$ 
  from  $ie1$  have  $ie: \text{real } (\text{floor } ?e) = ?e$  using  $\text{isint-iff}[\text{where } n=e \text{ and } bs=a \# bs]$ 
  numbound0-I[OF  $bn, \text{where } b=a \text{ and } b'=\text{real } x \text{ and } bs=bs]$ 
  by ( $\text{simp add: isint-iff}$ )
  {assume  $\text{real } (x-d) + ?e > 0$  hence  $?case$  using  $c1$ 
  numbound0-I[OF  $bn, \text{where } b=\text{real } (x-d) \text{ and } b'=\text{real } x \text{ and } bs=bs]$ 
  by ( $\text{simp del: real-of-int-minus}$ )}
  moreover
  {assume  $H: \neg \text{real } (x-d) + ?e > 0$ 
  let  $?v = \text{Neg } e$ 
  have  $vb: ?v \in \text{set } (\beta \ (Gt \ (CN \ 0 \ c \ e)))$  by  $\text{simp}$ 
  from  $\text{prems}(11)[\text{simplified simp-thms Inum.simps } \beta.\text{simps set.simps } bex\text{-simps}]$ 
  numbound0-I[OF  $bn, \text{where } b=a \text{ and } b'=\text{real } x \text{ and } bs=bs]$ 
  have  $nob: \neg(\exists \ j \in \{1 \ .. \ d\}. \text{real } x = - ?e + \text{real } j)$  by  $\text{auto}$ 
  from  $H \ p$  have  $\text{real } x + ?e > 0 \wedge \text{real } x + ?e \leq \text{real } d$  by ( $\text{simp add: } c1$ )
  hence  $\text{real } (x + \text{floor } ?e) > \text{real } (0::\text{int}) \wedge \text{real } (x + \text{floor } ?e) \leq \text{real } d$ 
  using  $ie$  by  $\text{simp}$ 
  hence  $x + \text{floor } ?e \geq 1 \wedge x + \text{floor } ?e \leq d$  by  $\text{simp}$ 
  hence  $\exists \ (j::\text{int}) \in \{1 \ .. \ d\}. j = x + \text{floor } ?e$  by  $\text{simp}$ 
  hence  $\exists \ (j::\text{int}) \in \{1 \ .. \ d\}. \text{real } x = \text{real } (- \text{floor } ?e + j)$ 
  by ( $\text{simp only: real-of-int-inject}$ ) ( $\text{simp add: ring-simps}$ )
  hence  $\exists \ (j::\text{int}) \in \{1 \ .. \ d\}. \text{real } x = - ?e + \text{real } j$ 
  by ( $\text{simp add: ie[simplified isint-iff]}$ )}
```

```

    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 ... = ($\neg j \text{ dvd } x - d + \text{floor } ?e$)
 using dvd-period[OF id, where $x=x$ and $c=-1$ and $t=\text{floor } ?e$] by simp
 also have ... = ($\neg \text{real } j \text{ rdvd real } (x - d + \text{floor } ?e)$)
 using int-rdvd-real[where $i=j$ and $x=\text{real } (x-d + \text{floor } ?e)$, symmetric,
 simplified]
 ie by simp
 also have ... = ($\neg \text{real } j \text{ rdvd real } x - \text{real } d + ?e$)
 using ie by simp
 finally show ?case using numbound0-I[OF bn, where $b=\text{real } (x-d)$ and
 $b'=\text{real } x$ and $bs=bs$] c1 p by simp
 qed (auto simp add: numbound0-I[where $bs=bs$ and $b=\text{real } (x - d)$ and $b'=\text{real } x$]
 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 $\forall x. \neg(\exists (j::\text{int}) \in \{1 .. d\}. \exists b \in \text{set}(\beta p). \text{Ifm } ((\text{Inum } (a \# bs) b + \text{real } j) \# bs) p) \longrightarrow \text{Ifm } (\text{real } x \# bs) p \longrightarrow \text{Ifm } (\text{real } (x - d) \# bs) p$ (is $\forall x. ?b \longrightarrow ?P x \longrightarrow ?P (x - d)$)
 proof (clarify)
 fix x
 assume nb: ?b and px: ?P x
 hence nb2: $\neg(\exists (j::\text{int}) \in \{1 .. d\}. \exists b \in (\text{Inum } (a \# bs)) \text{ ' set}(\beta p). \text{real } x = b + \text{real } j)$
 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 $\forall b \in \text{set } (\beta p). \text{isint } b \text{ bs}$
 using lp by (induct p rule: iszlfm.induct) (auto simp add: isint-neg isint-sub)

lemma cpmi-eq: $0 < D \implies (\text{EX } z::\text{int}. \text{ALL } x. x < z \longrightarrow (P x = P1 x)) \implies \text{ALL } x. \sim(\text{EX } (j::\text{int}) : \{1..D\}. \text{EX } (b::\text{int}) : B. P(b+j)) \longrightarrow P(x) \longrightarrow P(x - D)$
 $\implies (\text{ALL } (x::\text{int}). \text{ALL } (k::\text{int}). ((P1 x) = (P1 (x - k * D))))$
 $\implies (\text{EX } (x::\text{int}). P(x)) = ((\text{EX } (j::\text{int}) : \{1..D\} . (P1(j))) \mid (\text{EX } (j::\text{int}) : \{1..D\}. \text{EX } (b::\text{int}) : B. P(b+j)))$
 apply (rule iffI)
 prefer 2
 apply (drule minusinfinity)
 apply assumption+
 apply (fastsimp)
 apply clarsimp
 apply (subgoal-tac !!k. $0 \leq k \implies !x. P x \longrightarrow P (x - k * D)$)
 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

```

$a\varrho :: fm \Rightarrow int \Rightarrow fm$
recdef ϱ *measure size*
 $\varrho (And\ p\ q) = (\varrho\ p\ @\ \varrho\ q)$
 $\varrho (Or\ p\ q) = (\varrho\ p\ @\ \varrho\ q)$
 $\varrho (Eq\ (CN\ 0\ c\ e)) = [(Sub\ (C - 1)\ e, c)]$
 $\varrho (NEq\ (CN\ 0\ c\ e)) = [(Neg\ e, c)]$
 $\varrho (Lt\ (CN\ 0\ c\ e)) = []$
 $\varrho (Le\ (CN\ 0\ c\ e)) = []$
 $\varrho (Gt\ (CN\ 0\ c\ e)) = [(Neg\ e, c)]$
 $\varrho (Ge\ (CN\ 0\ c\ e)) = [(Sub\ (C - 1)\ e, c)]$
 $\varrho\ p = []$

recdef $\sigma\varrho$ *measure size*
 $\sigma\varrho (And\ p\ q) = (\lambda\ (t, k). And\ (\sigma\varrho\ p\ (t, k))\ (\sigma\varrho\ q\ (t, k)))$
 $\sigma\varrho (Or\ p\ q) = (\lambda\ (t, k). Or\ (\sigma\varrho\ p\ (t, k))\ (\sigma\varrho\ q\ (t, k)))$
 $\sigma\varrho (Eq\ (CN\ 0\ c\ e)) = (\lambda\ (t, k). if\ k\ dvd\ c\ then\ (Eq\ (Add\ (Mul\ (c\ div\ k)\ t)\ e))$
 $\hspace{15em} else\ (Eq\ (Add\ (Mul\ c\ t)\ (Mul\ k\ e))))$
 $\sigma\varrho (NEq\ (CN\ 0\ c\ e)) = (\lambda\ (t, k). if\ k\ dvd\ c\ then\ (NEq\ (Add\ (Mul\ (c\ div\ k)\ t)$
 $e))$
 $\hspace{15em} else\ (NEq\ (Add\ (Mul\ c\ t)\ (Mul\ k\ e))))$
 $\sigma\varrho (Lt\ (CN\ 0\ c\ e)) = (\lambda\ (t, k). if\ k\ dvd\ c\ then\ (Lt\ (Add\ (Mul\ (c\ div\ k)\ t)\ e))$
 $\hspace{15em} else\ (Lt\ (Add\ (Mul\ c\ t)\ (Mul\ k\ e))))$
 $\sigma\varrho (Le\ (CN\ 0\ c\ e)) = (\lambda\ (t, k). if\ k\ dvd\ c\ then\ (Le\ (Add\ (Mul\ (c\ div\ k)\ t)\ e))$
 $\hspace{15em} else\ (Le\ (Add\ (Mul\ c\ t)\ (Mul\ k\ e))))$
 $\sigma\varrho (Gt\ (CN\ 0\ c\ e)) = (\lambda\ (t, k). if\ k\ dvd\ c\ then\ (Gt\ (Add\ (Mul\ (c\ div\ k)\ t)\ e))$
 $\hspace{15em} else\ (Gt\ (Add\ (Mul\ c\ t)\ (Mul\ k\ e))))$
 $\sigma\varrho (Ge\ (CN\ 0\ c\ e)) = (\lambda\ (t, k). if\ k\ dvd\ c\ then\ (Ge\ (Add\ (Mul\ (c\ div\ k)\ t)\ e))$
 $\hspace{15em} else\ (Ge\ (Add\ (Mul\ c\ t)\ (Mul\ k\ e))))$
 $\sigma\varrho (Dvd\ i\ (CN\ 0\ c\ e)) = (\lambda\ (t, k). if\ k\ dvd\ c\ then\ (Dvd\ i\ (Add\ (Mul\ (c\ div\ k)\ t)$
 $e))$
 $\hspace{15em} else\ (Dvd\ (i*k)\ (Add\ (Mul\ c\ t)\ (Mul\ k\ e))))$
 $\sigma\varrho (NDvd\ i\ (CN\ 0\ c\ e)) = (\lambda\ (t, k). if\ k\ dvd\ c\ then\ (NDvd\ i\ (Add\ (Mul\ (c\ div\ k)$
 $t)\ e))$
 $\hspace{15em} else\ (NDvd\ (i*k)\ (Add\ (Mul\ c\ t)\ (Mul\ k\ e))))$
 $\sigma\varrho\ p = (\lambda\ (t, k). p)$

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 $a\varrho$ *measure size*
 $a\varrho (And\ p\ q) = (\lambda\ k. And\ (a\varrho\ p\ k)\ (a\varrho\ q\ k))$

```

aQ (Or p q) = (λ k. Or (aQ p k) (aQ q k))
aQ (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))))
aQ (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))))
aQ (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))))
aQ (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))))
aQ (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))))
aQ (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))))
aQ (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))))
aQ (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))))
aQ p = (λ k. p)

```

```

constdefs σ :: fm ⇒ int ⇒ num ⇒ fm
σ p k t ≡ And (Dvd k t) (σQ p (t,k))

```

lemma σ_Q:

```

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) (σQ 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: σQ.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:  $k \neq 0$  by simp
    from tint have ti:  $\text{real}(\text{floor}(\text{?N}(\text{real } x) t)) = \text{?N}(\text{real } x) t$  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  $b'=\text{real } x$ ]
      numbound0-I[OF nb, where  $bs=bs$  and  $b=?tk$  and  $b'=\text{real } x$ ] by (simp add:
ti) }
    moreover
    {assume  $\neg k \text{ dvd } c$ 
      from kpos have knz:  $k \neq 0$  by simp
      from tint have ti:  $\text{real}(\text{floor}(\text{?N}(\text{real } x) t)) = \text{?N}(\text{real } x) t$  using isint-def
    by simp
      from prems have ?I  $(\text{real } x) (\text{?s}(\text{Lt}(\text{CN } 0 c e))) = ((\text{real } c * (\text{?N}(\text{real } x)
t / \text{real } k) + \text{?N}(\text{real } x) e) * \text{real } k < 0)$ 
        using real-of-int-div[OF knz kdt]
        numbound0-I[OF tnb, where  $bs=bs$  and  $b=b'$  and  $b'=\text{real } x$ ]
        numbound0-I[OF nb, where  $bs=bs$  and  $b=?tk$  and  $b'=\text{real } x$ ] by (simp
add: ti ring-simps)
      also have  $\dots = (\text{?I } ?tk (\text{Lt}(\text{CN } 0 c e)))$  using pos-less-divide-eq[OF kpos,
where  $a=\text{real } c * (\text{?N}(\text{real } x) t / \text{real } k) + \text{?N}(\text{real } x) e$  and  $b=0$ , symmetric]
      real-of-int-div[OF knz kdt] numbound0-I[OF tnb, where  $bs=bs$  and  $b=b'$  and
 $b'=\text{real } x$ ]
        numbound0-I[OF nb, where  $bs=bs$  and  $b=?tk$  and  $b'=\text{real } x$ ]
        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:  $k \text{ dvd } c$ 
    from kpos have knz:  $k \neq 0$  by simp
    from tint have ti:  $\text{real}(\text{floor}(\text{?N}(\text{real } x) t)) = \text{?N}(\text{real } x) t$  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  $b'=\text{real } x$ ]
      numbound0-I[OF nb, where  $bs=bs$  and  $b=?tk$  and  $b'=\text{real } x$ ] by (simp add:
ti) }
    moreover
    {assume  $\neg k \text{ dvd } c$ 
      from kpos have knz:  $k \neq 0$  by simp
      from tint have ti:  $\text{real}(\text{floor}(\text{?N}(\text{real } x) t)) = \text{?N}(\text{real } x) t$  using isint-def
    by simp
      from prems have ?I  $(\text{real } x) (\text{?s}(\text{Le}(\text{CN } 0 c e))) = ((\text{real } c * (\text{?N}(\text{real } x)
t / \text{real } k) + \text{?N}(\text{real } x) e) * \text{real } k \leq 0)$ 
        using real-of-int-div[OF knz kdt]
        numbound0-I[OF tnb, where  $bs=bs$  and  $b=b'$  and  $b'=\text{real } x$ ]
        numbound0-I[OF nb, where  $bs=bs$  and  $b=?tk$  and  $b'=\text{real } x$ ] 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  $\neg k \text{ dvd } c$ 
    from kpos have knz:  $k \neq 0$  by simp
    from tint have ti:  $\text{real}(\text{floor}(\text{?N}(\text{real } x) \ t)) = \text{?N}(\text{real } x) \ t$  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  $\geq 0$ )
    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:  $k \text{ dvd } c$ 
    from kpos have knz:  $k \neq 0$  by simp
    from tint have ti:  $\text{real}(\text{floor}(\text{?N}(\text{real } x) \ t)) = \text{?N}(\text{real } x) \ t$  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  $\neg k \text{ dvd } c$ 
    from kpos have knz:  $k \neq 0$  by simp hence knz':  $\text{real } k \neq 0$  by simp
    from tint have ti:  $\text{real}(\text{floor}(\text{?N}(\text{real } x) \ t)) = \text{?N}(\text{real } x) \ t$  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≠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 (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 ag:

```

  assumes lp: iszlfm p (real (x::int)#bs) and kp: real k > 0
  shows Ifm (real (x*k)#bs) (ag 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: ag.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:  $\neg k \text{ dvd } c$  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:  $k \neq 0$  by simp hence knz': real k  $\neq 0$  by simp
  {assume kdc:  $k \text{ dvd } c$  from prems have ?case using real-of-int-div[OF knz
kdc] by simp }
  moreover
  {assume nkdc:  $\neg k \text{ dvd } c$  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:  $k \neq 0$  by simp hence knz': real k  $\neq 0$  by simp
  {assume kdc:  $k \text{ dvd } c$  from prems have ?case using real-of-int-div[OF knz
kdc] by simp }
  moreover
  {assume nkdc:  $\neg k \text{ dvd } c$  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:  $k \neq 0$  by simp hence knz': real k  $\neq 0$  by simp
  {assume kdc:  $k \text{ dvd } c$  from prems have ?case using real-of-int-div[OF knz
kdc] by simp }
  moreover
  {assume nkdc:  $\neg k \text{ dvd } c$  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:  $k \neq 0$  by simp hence knz': real k  $\neq 0$  by simp
  {assume kdc:  $k \text{ dvd } c$  from prems have ?case using real-of-int-div[OF knz
kdc] by simp }
  moreover
  {assume nkdc:  $\neg k \text{ dvd } c$  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:  $k \neq 0$  by simp hence knz':  $\text{real } k \neq 0$  by simp
  {assume kdc:  $k \text{ dvd } c$  from prems have ?case using real-of-int-div[OF knz
kdc] by simp }
  moreover
  {assume nkdc:  $\neg k \text{ dvd } c$  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:  $k \neq 0$  by simp hence knz':  $\text{real } k \neq 0$  by simp
  {assume kdc:  $k \text{ dvd } c$  from prems have ?case using real-of-int-div[OF knz
kdc] by simp }
  moreover
  {assume  $\neg k \text{ dvd } c$ 
  hence Ifm (real (x*k)#bs) (aQ (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:  $k \neq 0$  by simp hence knz':  $\text{real } k \neq 0$  by simp
  {assume kdc:  $k \text{ dvd } c$  from prems have ?case using real-of-int-div[OF knz
kdc] by simp }
  moreover
  {assume  $\neg k \text{ dvd } c$ 
  hence Ifm (real (x*k)#bs) (aQ (NDvd i (CN 0 c e)) k) =
    ( $\neg$  (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 aQ-ex:
  assumes lp: iszlfm p (real (x::int)#bs) and kp:  $k > 0$ 
  shows ( $\exists$  (x::int). real k rdvd real x  $\wedge$  Ifm (real x # bs) (aQ p k)) =

```

$(\exists (x::int). \text{Ifm } (real\ x\ \#bs)\ p) \text{ (is } (\exists x. ?D\ x \wedge ?P'\ x) = (\exists x. ?P\ x))$
proof –
 have $(\exists x. ?D\ x \wedge ?P'\ x) = (\exists x. k\ dvd\ x \wedge ?P'\ x)$ **using** *int-rdvd-iff* **by** *simp*
 also have $\dots = (\exists x. ?P'\ (x*k))$ **using** *unity-coeff-ex* [where $P=?P'$ and $l=k$,
simplified]
 by (*simp add: ring-simps*)
 also have $\dots = (\exists x. ?P\ x)$ **using** *aQ iszlfm-gen* [*OF lp*] *kp* **by** *auto*
 finally **show** *?thesis* .
qed

lemma σ_Q' : **assumes** *lp: iszlfm p (real (x::int)#bs)* **and** *kp: k > 0* **and** *nb: numbound0 t*
shows $\text{Ifm } (real\ x\ \#bs)\ (\sigma_Q\ p\ (t,k)) = \text{Ifm } ((\text{Inum } (real\ x\ \#bs)\ t)\ \#bs)\ (a_Q\ p\ k)$
using *lp*
by (*induct p rule: sigma_Q.induct, simp-all add:*
numbound0-I [*OF nb*, **where** $bs=bs$ **and** $b=\text{Inum } (real\ x\ \#bs)\ t$ **and** $b'=\text{real } x$]
numbound0-I [**where** $bs=bs$ **and** $b=\text{Inum } (real\ x\ \#bs)\ t$ **and** $b'=\text{real } x$]
bound0-I [**where** $bs=bs$ **and** $b=\text{Inum } (real\ x\ \#bs)\ t$ **and** $b'=\text{real } x$] *nth-pos2 cong:*
imp-cong)

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

lemma $Q\text{-l}$:
assumes *lp: iszlfm p (real (i::int)#bs)*
shows $\forall (b,k) \in \text{set } (Q\ p). k > 0 \wedge \text{numbound0 } b \wedge \text{isint } b \text{ (real } i\ \#bs)$
using *lp* **by** (*induct p rule: Q.induct, auto simp add: isint-sub isint-neg*)

lemma $\alpha_Q\text{-l}$:
assumes *lp: iszlfm p (real (i::int)#bs)*
shows $\forall (b,k) \in \text{set } (\alpha_Q\ p). k > 0 \wedge \text{numbound0 } b \wedge \text{isint } b \text{ (real } i\ \#bs)$
using *lp isint-add* [*OF isint-c*] [**where** $j=-1$], **where** $bs=\text{real } i\ \#bs$
by (*induct p rule: alpha_Q.induct, auto*)

lemma *zminusinf-Q*:
assumes *lp: iszlfm p (real (i::int)#bs)*
and *nmi: $\neg (\text{Ifm } (real\ i\ \#bs)\ (\text{minusinf } p))$ (is $\neg (\text{Ifm } (real\ i\ \#bs)\ (?M\ p))$)*
and *ex: $\text{Ifm } (real\ i\ \#bs)\ p$ (is $?I\ i\ p$)*
shows $\exists (e,c) \in \text{set } (Q\ p). \text{real } (c*i) > \text{Inum } (real\ i\ \#bs)\ e$ (is $\exists (e,c) \in ?R\ p.$
 $\text{real } (c*i) > ?N\ i\ e$)
using *lp nmi ex*
by (*induct p rule: minusinf.induct, auto*)

lemma $\sigma\text{-And}$: $\text{Ifm } bs\ (\sigma\ (\text{And } p\ q)\ k\ t) = \text{Ifm } bs\ (\text{And } (\sigma\ p\ k\ t)\ (\sigma\ q\ k\ t))$
using $\sigma\text{-def}$ **by** *auto*
lemma $\sigma\text{-Or}$: $\text{Ifm } bs\ (\sigma\ (\text{Or } p\ q)\ k\ t) = \text{Ifm } bs\ (\text{Or } (\sigma\ p\ k\ t)\ (\sigma\ q\ k\ t))$

using σ -def by auto

```

lemma  $\varrho$ : assumes  $lp$ : iszlfm  $p$  (real ( $i::int$ ) #bs)
  and  $pi$ : Ifm (real  $i$ #bs)  $p$ 
  and  $d$ :  $d\delta$   $p$   $d$ 
  and  $dp$ :  $d > 0$ 
  and  $nob$ :  $\forall (e,c) \in \text{set } (\varrho p). \forall j \in \{1 \dots c*d\}. \text{real } (c*i) \neq \text{Inum } (\text{real } i\#bs) e$ 
+ real  $j$ 
  (is  $\forall (e,c) \in \text{set } (\varrho p). \forall j \in \{1 \dots c*d\}. - \neq ?N i e + -$ )
  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 ( $c*i$ ) =  $-1 - ?N i e + \text{real } (1::int)$  and  $nob$ :  $\forall j \in \{1 \dots c*d\}. \text{real } (c*i) \neq -1 - ?N i e + \text{real } j$ 
    by simp+
    from mult-strict-left-mono[OF  $dp$   $cp$ ] have  $one: 1 \in \{1 \dots c*d\}$  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  $nob$ :  $\forall j \in \{1 \dots c*d\}. \text{real } (c*i) \neq - ?N i e + \text{real } j$ 
      by simp+
    {assume real ( $c*i$ )  $\neq - ?N i e + \text{real } (c*d)$ 
      with numbound0-I[OF  $nb$ , where  $bs=bs$  and  $b=\text{real } i - \text{real } d$  and  $b'=\text{real } i$ ]
      have ?case by (simp add: ring-simps)}
    moreover
    {assume  $pi$ : real ( $c*i$ ) =  $- ?N i e + \text{real } (c*d)$ 
      from mult-strict-left-mono[OF  $dp$   $cp$ ] have  $d$ :  $(c*d) \in \{1 \dots c*d\}$  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  $b=\text{real } i - \text{real } d$  and
 $b'=\text{real } i$ ]
        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  $b=\text{real } i - \text{real } d$  and
 $b'=\text{real } i$ ]
        ring-simps)

```

```

next
case (7 c e) hence cp: c > 0 and nb: numbound0 e and ei: isint e (real i # bs)
  and nob:  $\forall j \in \{1 \dots c*d\}. \text{real } (c*i) \neq - ?N i e + \text{real } j$ 
  and pi:  $\text{real } (c*i) + ?N i e > 0$  and cp':  $\text{real } c > 0$ 
  by simp+
let ?fe = floor (?N i e)
from pi cp have th:  $(\text{real } i + ?N i e / \text{real } c) * \text{real } c > 0$  by (simp add: ring-simps)
from pi ei[simplified isint-iff] have  $\text{real } (c*i + ?fe) > \text{real } (0::\text{int})$  by simp
hence pi':  $c*i + ?fe > 0$  by (simp only: real-of-int-less-iff[symmetric])
have  $\text{real } (c*i) + ?N i e > \text{real } (c*d) \vee \text{real } (c*i) + ?N i e \leq \text{real } (c*d)$  by
auto
moreover
{assume  $\text{real } (c*i) + ?N i e > \text{real } (c*d)$  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:  $\text{real } (c*i) + ?N i e \leq \text{real } (c*d)$ 
  with ei[simplified isint-iff] have  $\text{real } (c*i + ?fe) \leq \text{real } (c*d)$  by simp
  hence pid:  $c*i + ?fe \leq c*d$  by (simp only: real-of-int-le-iff)
  with pi' have  $\exists j1 \in \{1 \dots c*d\}. c*i + ?fe = j1$  by auto
  hence  $\exists j1 \in \{1 \dots c*d\}. \text{real } (c*i) = - ?N i e + \text{real } j1$ 
  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:  $\forall j \in \{1 \dots c*d\}. \text{real } (c*i) \neq - 1 - ?N i e + \text{real } j$ 
  and pi:  $\text{real } (c*i) + ?N i e \geq 0$  and cp':  $\text{real } c > 0$ 
  by simp+
let ?fe = floor (?N i e)
from pi cp have th:  $(\text{real } i + ?N i e / \text{real } c) * \text{real } c \geq 0$  by (simp add: ring-simps)
from pi ei[simplified isint-iff] have  $\text{real } (c*i + ?fe) \geq \text{real } (0::\text{int})$  by simp
hence pi':  $c*i + 1 + ?fe \geq 1$  by (simp only: real-of-int-le-iff[symmetric])
have  $\text{real } (c*i) + ?N i e \geq \text{real } (c*d) \vee \text{real } (c*i) + ?N i e < \text{real } (c*d)$  by
auto
moreover
{assume  $\text{real } (c*i) + ?N i e \geq \text{real } (c*d)$  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:  $\text{real } (c*i) + ?N i e < \text{real } (c*d)$ 
  with ei[simplified isint-iff] have  $\text{real } (c*i + ?fe) < \text{real } (c*d)$  by simp
  hence pid:  $c*i + 1 + ?fe \leq c*d$  by (simp only: real-of-int-le-iff)
  with pi' have  $\exists j1 \in \{1 \dots c*d\}. c*i + 1 + ?fe = j1$  by auto
  hence  $\exists j1 \in \{1 \dots c*d\}. \text{real } (c*i) + 1 = - ?N i e + \text{real } j1$ 
  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  $\exists j1 \in \{1 \dots c*d\}. \text{real } (c*i) = (- ?N i e + \text{real } j1) - 1$ 

```

by (simp only: ring-simps diff-def[symmetric])
 hence $\exists j1 \in \{1 \dots c*d\}. \text{real } (c*i) = -1 - ?N \ i \ e + \text{real } j1$
 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: $\text{real } j \text{ rdvd } \text{real } (c*i) + ?N \ i \ e$ (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: $\text{real } (\text{floor } ?e) = ?e$ 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: $j \text{ dvd } d$ by simp
 from ie[symmetric] have ?p i = $(\text{real } j \text{ rdvd } \text{real } (c*i + \text{floor } ?e))$ by simp
 also have $\dots = (j \text{ dvd } c*i + \text{floor } ?e)$
 using int-rdvd-iff [where i=j and t=c*i + floor ?e] by simp
 also have $\dots = (j \text{ dvd } c*i - c*d + \text{floor } ?e)$
 using dvd-period[OF id, where x=c*i and c=-c and t=floor ?e] by simp
 also have $\dots = (\text{real } j \text{ rdvd } \text{real } (c*i - c*d + \text{floor } ?e))$
 using int-rdvd-iff[where i=j and t=(c*i - c*d + floor ?e),symmetric,
 simplified]
 ie by simp
 also have $\dots = (\text{real } j \text{ rdvd } \text{real } (c*(i - d)) + ?e)$
 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: $\neg (\text{real } j \text{ rdvd } \text{real } (c*i) + ?N \ i \ e)$ (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: $\text{real } (\text{floor } ?e) = ?e$ 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: $j \text{ dvd } d$ by simp
 from ie[symmetric] have ?p i = $(\neg (\text{real } j \text{ rdvd } \text{real } (c*i + \text{floor } ?e)))$ by simp
 also have $\dots = \text{Not } (j \text{ dvd } c*i + \text{floor } ?e)$
 using int-rdvd-iff [where i=j and t=c*i + floor ?e] by simp
 also have $\dots = \text{Not } (j \text{ dvd } c*i - c*d + \text{floor } ?e)$
 using dvd-period[OF id, where x=c*i and c=-c and t=floor ?e] by simp
 also have $\dots = \text{Not } (\text{real } j \text{ rdvd } \text{real } (c*i - c*d + \text{floor } ?e))$
 using int-rdvd-iff[where i=j and t=(c*i - c*d + floor ?e),symmetric,
 simplified]
 ie by simp
 also have $\dots = \text{Not } (\text{real } j \text{ rdvd } \text{real } (c*(i - d)) + ?e)$
 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  $\sigma$ -nb: assumes lp: iszlfm p (a#bs) and nb: numbound0 t
shows bound0 ( $\sigma$  p k t)
using  $\sigma$ -nb[OF lp nb] nb by (simp add:  $\sigma$ -def)

lemma  $\varrho'$ : assumes lp: iszlfm p (a #bs)
and d:  $d\delta$  p d
and dp:  $d > 0$ 
shows  $\forall x. \neg(\exists (e,c) \in \text{set}(\varrho p). \exists (j::\text{int}) \in \{1 \dots c*d\}. \text{Ifm } (a \#bs) (\sigma p c$ 
(Add e (C j))))  $\longrightarrow \text{Ifm } (\text{real } x \#bs) p \longrightarrow \text{Ifm } (\text{real } (x - d) \#bs) p$  (is  $\forall x. ?b x$ 
 $\longrightarrow ?P x \longrightarrow ?P (x - d)$ )
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:  $\forall (e, c) \in \text{set}(\varrho p). \forall j \in \{1 \dots c * d\}. \text{real } (c * x) \neq \text{Inum } (\text{real } x \# bs)$ 
 $e + \text{real } j$ 
  proof(clarify)
    fix e c j assume ecR:  $(e,c) \in \text{set}(\varrho p)$  and jD:  $j \in \{1 \dots c*d\}$ 
    and cx:  $\text{real } (c*x) = \text{Inum } (\text{real } x \#bs) e + \text{real } j$ 
    let ?e =  $\text{Inum } (\text{real } x \#bs) e$ 
    let ?fe =  $\text{floor } ?e$ 
    from  $\varrho$ -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  $\text{real } (c*x) = \text{real } (?fe + j)$  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  $\text{real } c \text{ rdvd } \text{real } (?fe + j)$  by (simp only: int-rdvd-iff)
    hence rcdej:  $\text{real } c \text{ rdvd } ?e + \text{real } j$  by (simp add: ei[simplified isint-iff])
    from cx have  $(c*x) \text{ div } c = (?fe + j) \text{ div } c$  by simp
    with cp have  $x = (?fe + j) \text{ div } c$  by simp
    with px have th: ?P ((?fe + j) div c) by auto
    from cp have cp':  $\text{real } c > 0$  by simp
    from cdej have cdej':  $c \text{ dvd } \text{floor } (\text{Inum } (\text{real } x \#bs) (\text{Add } e (C j)))$  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  $\sigma$  $\varrho$ [where b'=real x, OF lp' cp' nb' ei' cdej',symmetric]
    have  $\text{Ifm } (\text{real } x \#bs) (\sigma \varrho p (\text{Add } e (C j), c))$  by simp

```

with $rcdej$ **have** $th: \text{Ifm } (real\ x \# bs) (\sigma\ p\ c\ (Add\ e\ (C\ j)))$ **by** ($simp\ add: \sigma\text{-def}$)
from $th\ bound0\text{-}I[OF\ \sigma\text{-nb}[OF\ lp\ nb', \text{where } k=c], \text{where } bs=bs \text{ and } b=real\ x \text{ and } b'=a]$
have $\text{Ifm } (a \# bs) (\sigma\ p\ c\ (Add\ e\ (C\ j)))$ **by** $blast$
with $exR\ jD\ nob1$ **show** $False$ **by** $blast$
qed
from $\varrho[OF\ lp'\ px\ d\ dp\ nob]$ **show** $?P\ (x - d)$.
qed

lemma $rl\text{-}thm$:

assumes $lp: iszlfm\ p\ (real\ (i::int) \# bs)$
shows $(\exists\ (x::int). \text{Ifm } (real\ x \# bs)\ p) = ((\exists\ j \in \{1 .. \delta\ p\}. \text{Ifm } (real\ j \# bs)\ (minusinf\ p)) \vee (\exists\ (e, c) \in set\ (\varrho\ p). \exists\ j \in \{1 .. c * (\delta\ p)\}. \text{Ifm } (a \# bs) (\sigma\ p\ c\ (Add\ e\ (C\ j)))))$
(is $(\exists\ (x::int). ?P\ x) = ((\exists\ j \in \{1 .. \delta\ p\}. ?MP\ j) \vee (\exists\ (e, c) \in ?R. \exists\ j \in .. ?SP\ c\ e\ j))$
is $?lhs = (?MD \vee ?RD)$ **is** $?lhs = ?rhs$
proof–
let $?d = \delta\ p$
from $\delta[OF\ lp]$ **have** $d: d \delta\ p\ ?d$ **and** $dp: ?d > 0$ **by** $auto$
{ assume $H: ?MD$ **hence** $th: \exists\ (x::int). ?MP\ x$ **by** $blast$
from $H\ minusinf\text{-}ex[OF\ lp\ th]$ **have** $?thesis$ **by** $blast$
moreover
{ fix $e\ c\ j$ **assume** $exR: (e, c) \in ?R$ **and** $jD: j \in \{1 .. c * ?d\}$ **and** $spx: ?SP\ c\ e\ j$
from $exR\ \varrho\text{-}l[OF\ lp]$ **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** ($simp\ add: isint\text{-}iff$)
with $isint\text{-}add[OF\ numbound0\text{-}gen[OF\ nb\ ei, rule\text{-}format, \text{where } y=real\ i]]$
have $ej: isint\ (Add\ e\ (C\ j))\ (real\ i \# bs)$ **by** $simp$
from nb **have** $nb': numbound0\ (Add\ e\ (C\ j))$ **by** $simp$
from $spx\ bound0\text{-}I[OF\ \sigma\text{-nb}[OF\ lp\ nb', \text{where } k=c], \text{where } bs=bs \text{ and } b=a$
and $b'=real\ i]$
have $spx': \text{Ifm } (real\ i \# bs) (\sigma\ p\ c\ (Add\ e\ (C\ j)))$ **by** $blast$
from spx' **have** $rcdej: real\ c\ rdvd\ (Inum\ (real\ i \# bs)\ (Add\ e\ (C\ j)))$
and $sr: \text{Ifm } (real\ i \# bs) (\sigma\ \varrho\ p\ (Add\ e\ (C\ j), c))$ **by** ($simp\ add: \sigma\text{-def}$) +
from $rcdej\ ej$ [$simplified\ isint\text{-}iff$]
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** ($simp\ only: int\text{-}rdvd\text{-}iff$)
from cp **have** $cp': real\ c > 0$ **by** $simp$
from $\sigma\varrho[OF\ lp\ cp'\ nb'\ ej\ cdej]$ spx' **have** $?P\ (\lfloor Inum\ (real\ i \# bs)\ (Add\ e\ (C\ j)) \rfloor\ div\ c)$
by ($simp\ add: \sigma\text{-def}$)
hence $?lhs$ **by** $blast$
with $exR\ jD\ spx$ **have** $?thesis$ **by** $blast$
moreover


```

{ fix x assume px: ?P x and nob:  $\neg$  ?RD
  from iszlfm-gen [OF lp, rule-format, where y=a] have lp': iszlfm p (a#bs) .
  from  $\varrho'$ [OF lp' d dp, rule-format, OF nob] have th:  $\forall x. ?P x \longrightarrow ?P (x -$ 
?d) by blast
  from minusinf-inf[OF lp] obtain z where z:  $\forall x < z. ?MP x = ?P x$  by blast
  have zp:  $\text{abs } (x - z) + 1 \geq 0$  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:  $\exists x. ?MP x$  by
auto
  with minusinf-bex[OF lp] px nob have ?thesis by blast}
ultimately show ?thesis by blast
qed

```

```

lemma mirror- $\alpha\varrho$ : assumes lp: iszlfm p (a#bs)
  shows  $(\lambda (t,k). (\text{Inum } (a\#bs) t, k)) \text{ ' set } (\alpha\varrho p) = (\lambda (t,k). (\text{Inum } (a\#bs) t, k))$ 
  ' set ( $\varrho$  (mirror p))
using lp
by (induct p rule: mirror.induct, simp-all add: split-def image-Un )

```

The \mathbb{R} part

Linearity for fm where Bound 0 ranges over \mathbb{R}

consts

isrlfm :: fm \Rightarrow bool

recdef *isrlfm* measure size

```

isrlfm (And p q) = (isrlfm p  $\wedge$  isrlfm q)
isrlfm (Or p q) = (isrlfm p  $\wedge$  isrlfm q)
isrlfm (Eq (CN 0 c e)) = (c > 0  $\wedge$  numbound0 e)
isrlfm (NEq (CN 0 c e)) = (c > 0  $\wedge$  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-size

```

rsplit0 (Bound 0) = [(T, 1, C 0)]
rsplit0 (Add a b) = (let acs = rsplit0 a ; bcs = rsplit0 b
  in map  $(\lambda ((p,n,t),(q,m,s)). (\text{And } p \ q, \ n+m, \ \text{Add } t \ s)) [(a,b).$ 
 $a \leftarrow acs, b \leftarrow bcs]$ )

```

```

rsplit0 (Sub a b) = rsplit0 (Add a (Neg b))
rsplit0 (Neg a) = map (λ (p,n,s). (p,-n,Neg s)) (rsplit0 a)
rsplit0 (Floor a) = foldl (op @) [] (map
  (λ (p,n,s). if n=0 then [(p,0,Floor s)]
    else (map (λ 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 (λ (p,n,s). (p,n+c,s)) (rsplit0 a)
rsplit0 (CN m c a) = map (λ (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 (λ (p,n,s). (p,c*n,Mul c s)) (rsplit0 a)
rsplit0 t = [(T,0,t)]

```

```

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

```

```

lemma rsplit0-cs:
shows  $\forall (p,n,s) \in \text{set } (rsplit0 t).$ 
  (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:  $\forall (i::int). i = 0 \vee i < 0 \vee i > 0$  by arith
have U1: (UNION {(p,n,s). (p,n,s)  $\in$  ?SS a  $\wedge$  n=0} (λ (p,n,s). set (?ff
(p,n,s)))) =
  (UNION {(p,n,s). (p,n,s)  $\in$  ?SS a  $\wedge$  n=0} (λ (p,n,s). set [(p,0,Floor s)])) by
auto
have U2':  $\forall (p,n,s) \in \{(p,n,s). (p,n,s) \in ?SS a \wedge n>0\}.$ 
  ?ff (p,n,s) = map (?f(p,n,s)) (iupt(0,n)) by auto
hence U2: (UNION {(p,n,s). (p,n,s)  $\in$  ?SS a  $\wedge$  n>0} (λ (p,n,s). set (?ff
(p,n,s)))) =
  (UNION {(p,n,s). (p,n,s)  $\in$  ?SS a  $\wedge$  n>0} (λ (p,n,s).
    set (map (?f(p,n,s)) (iupt(0,n)))))
proof—
fix M :: ('a $\times$ 'b $\times$ 'c) set and f :: ('a $\times$ 'b $\times$ 'c)  $\Rightarrow$  'd list and g
assume  $\forall (a,b,c) \in M. f (a,b,c) = g a b c$ 
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 $U3'$: $\forall (p,n,s) \in \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n < 0\}. ?ff\ (p,n,s) = map\ (?f(p,n,s))\ (iupt(n,0))$
by *auto*
hence $U3$: $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n < 0\}\ (\lambda\ (p,n,s). set\ (?ff\ (p,n,s)))) =$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n < 0\}\ (\lambda(p,n,s). set\ (map\ (?f(p,n,s))\ (iupt(n,0)))))$
proof–
fix $M :: ('a \times 'b \times 'c)\ set$ **and** $f :: ('a \times 'b \times 'c) \Rightarrow 'd\ list$ **and** g
assume $\forall (a,b,c) \in M. f\ (a,b,c) = g\ a\ b\ c$
thus $(UNION\ M\ (\lambda\ (a,b,c). set\ (f\ (a,b,c)))) = (UNION\ M\ (\lambda\ (a,b,c). set\ (g\ a\ b\ c)))$
by *(auto simp add: split-def)*
qed
have $?SS\ (Floor\ a) = UNION\ (?SS\ a)\ (\lambda x. set\ (?ff\ x))$
by *(auto simp add: foldl-conv-concat)*
also have $\dots = UNION\ (?SS\ a)\ (\lambda\ (p,n,s). set\ (?ff\ (p,n,s)))$ **by** *auto*
also have $\dots =$
 $((UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n = 0\}\ (\lambda\ (p,n,s). set\ (?ff\ (p,n,s))))\ Un$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n > 0\}\ (\lambda\ (p,n,s). set\ (?ff\ (p,n,s))))\ Un$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n < 0\}\ (\lambda\ (p,n,s). set\ (?ff\ (p,n,s))))$
using *int-cases[rule-format]* **by** *blast*
also have $\dots =$
 $((UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n = 0\}\ (\lambda\ (p,n,s). set\ [(p,0,Floor\ s)]))\ Un$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n > 0\}\ (\lambda(p,n,s). set\ (map\ (?f(p,n,s))\ (iupt(0,n)))))$
 Un
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n < 0\}\ (\lambda\ (p,n,s). set\ (map\ (?f(p,n,s))\ (iupt(n,0)))))$ **by** *(simp only: U1 U2 U3)*
also have $\dots =$
 $((UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n = 0\}\ (\lambda\ (p,n,s). \{(p,0,Floor\ s)\}))\ Un$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n > 0\}\ (\lambda\ (p,n,s). \{f(p,n,s)\}\ ' \{0\ .. n\}))$
 Un
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n < 0\}\ (\lambda\ (p,n,s). \{f(p,n,s)\}\ ' \{n\ .. 0\})))$
by *(simp only: set-map iupt-set set.simps)*
also have $\dots =$
 $((UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n = 0\}\ (\lambda\ (p,n,s). \{(p,0,Floor\ s)\}))\ Un$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n > 0\}\ (\lambda\ (p,n,s). \{f(p,n,s)\ j\ |\ j. j \in \{0\ .. n\}\}))\ Un$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n < 0\}\ (\lambda\ (p,n,s). \{f(p,n,s)\ j\ |\ j. j \in \{n\ .. 0\}\}))$ **by** *blast*
finally
have FS : $?SS\ (Floor\ a) =$
 $((UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n = 0\}\ (\lambda\ (p,n,s). \{(p,0,Floor\ s)\}))\ Un$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n > 0\}\ (\lambda\ (p,n,s). \{f(p,n,s)\ j\ |\ j. j \in \{0\ .. n\}\}))\ Un$
 $(UNION\ \{(p,n,s). (p,n,s) \in ?SS\ a \wedge n < 0\}\ (\lambda\ (p,n,s). \{f(p,n,s)\ j\ |\ j. j \in \{n\ .. 0\}\}))$ **by** *blast*

```

show ?case
proof(simp only: FS, clarsimp simp del: Ifm.simps Inum.simps, -)
  fix p n s
  let ?ths = (?I p  $\longrightarrow$  (?N (Floor a) = ?N (CN 0 n s)))  $\wedge$  numbound0 s  $\wedge$ 
  isrlfm p
  assume ( $\exists$  ba. (p, 0, ba)  $\in$  set (rsplit0 a)  $\wedge$  n = 0  $\wedge$  s = Floor ba)  $\vee$ 
  ( $\exists$  ab ac ba.
    (ab, ac, ba)  $\in$  set (rsplit0 a)  $\wedge$ 
    0 < ac  $\wedge$ 
    ( $\exists$  j. p = fp ab ac ba j  $\wedge$ 
      n = 0  $\wedge$  s = Add (Floor ba) (C j)  $\wedge$  0  $\leq$  j  $\wedge$  j  $\leq$  ac))  $\vee$ 
  ( $\exists$  ab ac ba.
    (ab, ac, ba)  $\in$  set (rsplit0 a)  $\wedge$ 
    ac < 0  $\wedge$ 
    ( $\exists$  j. p = fp ab ac ba j  $\wedge$ 
      n = 0  $\wedge$  s = Add (Floor ba) (C j)  $\wedge$  ac  $\leq$  j  $\wedge$  j  $\leq$  0))
  moreover
  {fix s'
    assume (p, 0, s')  $\in$  ?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')  $\in$  ?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  $\leq$  j and jn: j  $\leq$  n'
    from prems pns have H: (Ifm ((x::real) # (bs::real list)) p'  $\longrightarrow$ 
      Inum (x # bs) a = Inum (x # bs) (CN 0 n' s'))  $\wedge$ 
      numbound0 s'  $\wedge$  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)  $\longrightarrow$ 
      (((?N ?nxs  $\geq$  real ?l)  $\wedge$  (?N ?nxs < real (?l + 1)))  $\wedge$  (?N a = ?N ?nxs
  ))
    by (simp add: fp-def np ring-simps numsub numadd numfloor)
    also have ...  $\longrightarrow$  ((floor (?N ?nxs) = ?l)  $\wedge$  (?N a = ?N ?nxs))
    using floor-int-eq[where x=?N ?nxs and n=?l] by simp
    moreover
    have ...  $\longrightarrow$  (?N (Floor a) = ?N ((Add (Floor s') (C j)))) by simp
    ultimately have ?I (?p (p', n', s') j)  $\longrightarrow$  (?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 $pa: (pa, na, sa) \in ?SS\ a \wedge ?I\ pa$ by *blast*
 from *prems* have $\exists (pb, nb, sb) \in ?SS\ b. ?I\ pb$ by *auto*
 then obtain $pb\ nb\ sb$ where $pb: (pb, nb, sb) \in ?SS\ b \wedge ?I\ pb$ by *blast*
 from $pa\ pb$ have $th: ((pa, na, sa), (pb, nb, sb)) \in set[(x, y). x \leftarrow rsplit0\ a, y \leftarrow rsplit0\ b]$
 by (*auto*)
 let $?f = (\lambda (p, n, t), (q, m, s)). (And\ p\ q,\ n+m,\ Add\ t\ s)$
 from $imageI[OF\ th, \text{ where } f = ?f]$ have $?f\ ((pa, na, sa), (pb, nb, sb)) \in ?SS\ (Add\ a\ b)$
 by (*simp add: Let-def*)
 hence $(And\ pa\ pb,\ na + nb,\ Add\ sa\ sb) \in ?SS\ (Add\ a\ b)$ by *simp*
 moreover from $pa\ pb$ have $?I\ (And\ pa\ pb)$ by *simp*
 ultimately show *?case* by *blast*
 next
 case (5 a)
 let $?p = \lambda (p, n, s)\ j. fp\ p\ n\ s\ j$
 let $?f = (\lambda (p, n, s)\ j. (?p\ (p, n, s)\ j,\ (0::int), (Add\ (Floor\ s)\ (C\ j))))$
 let $?J = \lambda n. \text{ if } n > 0 \text{ then } iupt\ (0, n) \text{ else } iupt\ (n, 0)$
 let $?ff = (\lambda (p, n, s). \text{ if } n = 0 \text{ then } [(p, 0, Floor\ s)] \text{ else } map\ (?f\ (p, n, s))\ (?J\ n))$
 have *int-cases*: $\forall (i::int). i = 0 \vee i < 0 \vee i > 0$ by *arith*
 have $U1: (UNION\ \{(p, n, s). (p, n, s) \in ?SS\ a \wedge n = 0\}\ (\lambda (p, n, s). set\ (?ff\ (p, n, s)))) = (UNION\ \{(p, n, s). (p, n, s) \in ?SS\ a \wedge n = 0\}\ (\lambda (p, n, s). set\ [(p, 0, Floor\ s)]))$ by *auto*
 have $U2': \forall (p, n, s) \in \{(p, n, s). (p, n, s) \in ?SS\ a \wedge n > 0\}. ?ff\ (p, n, s) = map\ (?f\ (p, n, s))\ (iupt(0, n))$
 by *auto*
 hence $U2: (UNION\ \{(p, n, s). (p, n, s) \in ?SS\ a \wedge n > 0\}\ (\lambda (p, n, s). set\ (?ff\ (p, n, s)))) = (UNION\ \{(p, n, s). (p, n, s) \in ?SS\ a \wedge n > 0\}\ (\lambda (p, n, s). set\ (map\ (?f\ (p, n, s))\ (iupt(0, n)))))$
 proof—
 fix $M :: ('a \times 'b \times 'c)\ set$ and $f :: ('a \times 'b \times 'c) \Rightarrow 'd\ list$ and g
 assume $\forall (a, b, c) \in M. f\ (a, b, c) = g\ a\ b\ c$
 thus $(UNION\ M\ (\lambda (a, b, c). set\ (f\ (a, b, c)))) = (UNION\ M\ (\lambda (a, b, c). set\ (g\ a\ b\ c)))$
 by (*auto simp add: split-def*)
 qed
 have $U3': \forall (p, n, s) \in \{(p, n, s). (p, n, s) \in ?SS\ a \wedge n < 0\}. ?ff\ (p, n, s) = map\ (?f\ (p, n, s))\ (iupt(n, 0))$
 by *auto*
 hence $U3: (UNION\ \{(p, n, s). (p, n, s) \in ?SS\ a \wedge n < 0\}\ (\lambda (p, n, s). set\ (?ff\ (p, n, s)))) = (UNION\ \{(p, n, s). (p, n, s) \in ?SS\ a \wedge n < 0\}\ (\lambda (p, n, s). set\ (map\ (?f\ (p, n, s))\ (iupt(n, 0)))))$
 proof—
 fix $M :: ('a \times 'b \times 'c)\ set$ and $f :: ('a \times 'b \times 'c) \Rightarrow 'd\ list$ and g
 assume $\forall (a, b, c) \in M. f\ (a, b, c) = g\ a\ b\ c$
 thus $(UNION\ M\ (\lambda (a, b, c). set\ (f\ (a, b, c)))) = (UNION\ M\ (\lambda (a, b, c). set\ (g\ a\ b\ c)))$
 by (*auto simp add: split-def*)
 qed

```

have ?SS (Floor a) = UNION (?SS a) (λx. 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 $?N \text{ (Floor } s) \leq \text{real } n * x + ?N s$.
moreover
{from *mult-strict-left-mono*[*OF* $x1$] *np*
have $\text{real } n * x + ?N s < \text{real } n + ?N s$ **by** *simp*
also from *real-of-int-floor-add-one-gt*[**where** $r=?N s$]
have $\dots < \text{real } n + ?N \text{ (Floor } s) + 1$ **by** *simp*
finally have $\text{real } n * x + ?N s < ?N \text{ (Floor } s) + \text{real } (n+1)$ **by** *simp*
ultimately have $?N \text{ (Floor } s) \leq \text{real } n * x + ?N s \wedge \text{real } n * x + ?N s < ?N \text{ (Floor } s) + \text{real } (n+1)$ **by** *simp*
hence th: $0 \leq \text{real } n * x + ?N s - ?N \text{ (Floor } s) \wedge \text{real } n * x + ?N s - ?N \text{ (Floor } s) < \text{real } (n+1)$ **by** *simp*
from *real-in-int-intervals* **th have** $\exists j \in \{0 .. n\}. \text{real } j \leq \text{real } n * x + ?N s - ?N \text{ (Floor } s) \wedge \text{real } n * x + ?N s - ?N \text{ (Floor } s) < \text{real } (j+1)$ **by** *simp*

hence $\exists j \in \{0 .. n\}. 0 \leq \text{real } n * x + ?N s - ?N \text{ (Floor } s) - \text{real } j \wedge \text{real } n * x + ?N s - ?N \text{ (Floor } s) - \text{real } (j+1) < 0$
by(*simp only: myl*[*rule-format*, **where** $b=\text{real } n * x + \text{Inum } (x \# bs) s - \text{Inum } (x \# bs) \text{ (Floor } s)$] *less-iff-diff-less-0*[**where** $a=\text{real } n * x + ?N s - ?N \text{ (Floor } s)$])
hence $\exists j \in \{0 .. n\}. ?I \text{ (?p } (p,n,s) j)$
using *pns* **by** (*simp add: fp-def np ring-simps numsub numadd*)
then obtain j **where** $j\text{-def: } j \in \{0 .. n\} \wedge ?I \text{ (?p } (p,n,s) j)$ **by** *blast*
hence $\exists x \in \{?p \text{ (p,n,s) } j \mid j. 0 \leq j \wedge j \leq n\}. ?I x$ **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** $?N \text{ (Floor } s) + 1 > ?N s$ **by** *simp*
moreover from *mult-left-mono-neg*[*OF* xp] nn **have** $?N s \geq \text{real } n * x + ?N s$
by *simp*
ultimately have $?N \text{ (Floor } s) + 1 > \text{real } n * x + ?N s$ **by** *arith*
moreover
{from *mult-strict-left-mono-neg*[*OF* $x1$, **where** $c=\text{real } n$] nn
have $\text{real } n * x + ?N s \geq \text{real } n + ?N s$ **by** *simp*
moreover from *real-of-int-floor-le*[**where** $r=?N s$] **have** $\text{real } n + ?N s \geq \text{real } n + ?N \text{ (Floor } s)$ **by** *simp*
ultimately have $\text{real } n * x + ?N s \geq ?N \text{ (Floor } s) + \text{real } n$
by (*simp only: ring-simps*)
ultimately have $?N \text{ (Floor } s) + \text{real } n \leq \text{real } n * x + ?N s \wedge \text{real } n * x + ?N s < ?N \text{ (Floor } s) + \text{real } (1::\text{int})$ **by** *simp*
hence th: $\text{real } n \leq \text{real } n * x + ?N s - ?N \text{ (Floor } s) \wedge \text{real } n * x + ?N s - ?N \text{ (Floor } s) < \text{real } (1::\text{int})$ **by** *simp*
have $th1: \forall (a::\text{real}). (- a > 0) = (a < 0)$ **by** *auto*
have $th2: \forall (a::\text{real}). (0 \geq - a) = (a \geq 0)$ **by** *auto*
from *real-in-int-intervals* **th have** $\exists j \in \{n .. 0\}. \text{real } j \leq \text{real } n * x + ?N s$

– $?N \text{ (Floor } s) \wedge \text{real } n * x + ?N \text{ } s - ?N \text{ (Floor } s) < \text{real } (j+1)$ **by** *simp*

hence $\exists j \in \{n \dots 0\}. 0 \leq \text{real } n * x + ?N \text{ } s - ?N \text{ (Floor } s) - \text{real } j \wedge \text{real } n * x + ?N \text{ } s - ?N \text{ (Floor } s) - \text{real } (j+1) < 0$

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 $\exists j \in \{n \dots 0\}. 0 \geq -(\text{real } n * x + ?N \text{ } s - ?N \text{ (Floor } s) - \text{real } j) \wedge -(\text{real } n * x + ?N \text{ } s - ?N \text{ (Floor } s) - \text{real } (j+1)) > 0$ **by** (*simp only: th1[rule-format] th2[rule-format]*)

hence $\exists j \in \{n \dots 0\}. ?I \text{ } (?p \text{ } (p, n, s) \text{ } j)$

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*: $j \in \{n \dots 0\} \wedge ?I \text{ } (?p \text{ } (p, n, s) \text{ } j)$ **by** *blast*

hence $\exists x \in \{?p \text{ } (p, n, s) \text{ } j \mid j. n \leq j \wedge j \leq 0\}. ?I \text{ } x$ **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* :: $(\text{int} \Rightarrow \text{num} \Rightarrow \text{fm}) \Rightarrow \text{num} \Rightarrow \text{fm}$

rsplit *f a* $\equiv \text{foldr } \text{disj } (\text{map } (\lambda (\varphi, n, s). \text{conj } \varphi \text{ } (f \text{ } n \text{ } s))) \text{ } (\text{rsplit0 } a)) \text{ } F$

lemma *foldr-disj-map*: *Ifm* *bs* $(\text{foldr } \text{disj } (\text{map } f \text{ } xs) \text{ } F) = (\exists x \in \text{set } xs. \text{Ifm } bs \text{ } (f \text{ } x))$

by(*induct xs, simp-all*)

lemma *foldr-conj-map*: *Ifm* *bs* $(\text{foldr } \text{conj } (\text{map } f \text{ } xs) \text{ } T) = (\forall x \in \text{set } xs. \text{Ifm } bs \text{ } (f \text{ } x))$

by(*induct xs, simp-all*)

lemma *foldr-disj-map-rlfm*:

assumes *lf*: $\forall n \text{ } s. \text{numbound0 } s \longrightarrow \text{isrlfm } (f \text{ } n \text{ } s)$

and $\varphi: \forall (\varphi, n, s) \in \text{set } xs. \text{numbound0 } s \wedge \text{isrlfm } \varphi$

shows $\text{isrlfm } (\text{foldr } \text{disj } (\text{map } (\lambda (\varphi, n, s). \text{conj } \varphi \text{ } (f \text{ } n \text{ } s))) \text{ } xs) \text{ } F$

using *lf* φ **by** (*induct xs, auto*)

lemma *rsplit-ex*: *Ifm* *bs* $(\text{rsplit } f \text{ } a) = (\exists (\varphi, n, s) \in \text{set } (\text{rsplit0 } a). \text{Ifm } bs \text{ } (\text{conj } \varphi \text{ } (f \text{ } n \text{ } s)))$

using *foldr-disj-map[where xs=rsplit0 a] rsplit-def* **by** (*simp add: split-def*)

lemma *rsplit-l*: **assumes** *lf*: $\forall n \text{ } s. \text{numbound0 } s \longrightarrow \text{isrlfm } (f \text{ } n \text{ } s)$

shows $\text{isrlfm } (\text{rsplit } f \text{ } a)$

proof–

from *rsplit0-cs*[**where** $t=a$] **have** $th: \forall (\varphi, n, s) \in \text{set } (rsplit0\ a). \text{numbound0 } s$
 $\wedge \text{isrlfm } \varphi$ **by** *blast*
from *foldr-disj-map-rlfm*[*OF lf th*] *rsplit-def* **show** *?thesis* **by** *simp*
qed

lemma *rsplit*:

assumes $xp: x \geq 0$ **and** $x1: x < 1$
and $f: \forall a\ n\ s. \text{Inum } (x\#bs)\ a = \text{Inum } (x\#bs)\ (CN\ 0\ n\ s) \wedge \text{numbound0 } s \longrightarrow$
 $(\text{Ifm } (x\#bs)\ (f\ n\ s) = \text{Ifm } (x\#bs)\ (g\ a))$
shows $\text{Ifm } (x\#bs)\ (rsplit\ f\ a) = \text{Ifm } (x\#bs)\ (g\ a)$
proof(*auto*)
let $?I = \lambda x\ p. \text{Ifm } (x\#bs)\ p$
let $?N = \lambda x\ t. \text{Inum } (x\#bs)\ t$
assume $?I\ x\ (rsplit\ f\ a)$
hence $\exists (\varphi, n, s) \in \text{set } (rsplit0\ a). ?I\ x\ (\text{And } \varphi\ (f\ n\ s))$ **using** *rsplit-ex* **by** *simp*
then obtain $\varphi\ n\ s$ **where** $\text{fnsS}:(\varphi, n, s) \in \text{set } (rsplit0\ a)$ **and** $?I\ x\ (\text{And } \varphi\ (f\ n\ s))$ **by** *blast*
hence $\varphi: ?I\ x\ \varphi$ **and** $\text{fns}: ?I\ x\ (f\ n\ s)$ **by** *auto*
from *rsplit0-cs*[**where** $t=a$ **and** $bs=bs$ **and** $x=x$, *rule-format*, *OF fnsS*] φ
have $th: (?N\ x\ a = ?N\ x\ (CN\ 0\ n\ s)) \wedge \text{numbound0 } s$ **by** *auto*
from $f[\text{rule-format},\ OF\ th]\ \text{fns}$ **show** $?I\ x\ (g\ a)$ **by** *simp*
next
let $?I = \lambda x\ p. \text{Ifm } (x\#bs)\ p$
let $?N = \lambda x\ t. \text{Inum } (x\#bs)\ t$
assume $ga: ?I\ x\ (g\ a)$
from *rsplit0-complete*[*OF xp x1*, **where** $bs=bs$ **and** $t=a$]
obtain $\varphi\ n\ s$ **where** $\text{fnsS}:(\varphi, n, s) \in \text{set } (rsplit0\ a)$ **and** $fx: ?I\ x\ \varphi$ **by** *blast*
from *rsplit0-cs*[**where** $t=a$ **and** $x=x$ **and** $bs=bs$] $\text{fnsS}\ fx$
have $\text{ans}: ?N\ x\ a = ?N\ x\ (CN\ 0\ n\ s)$ **and** $\text{nb}: \text{numbound0 } s$ **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 $lt :: \text{int} \Rightarrow \text{num} \Rightarrow \text{fm}$ **where**

lt-def: $lt\ c\ t = (\text{if } c = 0 \text{ then } (Lt\ t) \text{ else if } c > 0 \text{ then } (Lt\ (CN\ 0\ c\ t))$
 $\text{else } (Gt\ (CN\ 0\ (-c)\ (Neg\ t))))$

definition $le :: \text{int} \Rightarrow \text{num} \Rightarrow \text{fm}$ **where**

le-def: $le\ c\ t = (\text{if } c = 0 \text{ then } (Le\ t) \text{ else if } c > 0 \text{ then } (Le\ (CN\ 0\ c\ t))$
 $\text{else } (Ge\ (CN\ 0\ (-c)\ (Neg\ t))))$

definition $gt :: \text{int} \Rightarrow \text{num} \Rightarrow \text{fm}$ **where**

gt-def: $gt\ c\ t = (\text{if } c = 0 \text{ then } (Gt\ t) \text{ else if } c > 0 \text{ then } (Gt\ (CN\ 0\ c\ t))$
 $\text{else } (Lt\ (CN\ 0\ (-c)\ (Neg\ t))))$

definition $ge :: \text{int} \Rightarrow \text{num} \Rightarrow \text{fm}$ **where**

ge-def: $ge\ c\ t = (\text{if } c = 0 \text{ then } (Ge\ t) \text{ else if } c > 0 \text{ then } (Ge\ (CN\ 0\ c\ t))$
 $\text{else } (Le\ (CN\ 0\ (-c)\ (Neg\ t))))$

definition $eq :: int \Rightarrow num \Rightarrow fm$ **where**

$eq\text{-}def: eq\ c\ t = (if\ c = 0\ then\ (Eq\ t)\ else\ if\ c > 0\ then\ (Eq\ (CN\ 0\ c\ t))$
 $\quad\quad\quad else\ (Eq\ (CN\ 0\ (-c)\ (Neg\ t))))$

definition $neg :: int \Rightarrow num \Rightarrow fm$ **where**

$neg\text{-}def: neg\ c\ t = (if\ c = 0\ then\ (NEq\ t)\ else\ if\ c > 0\ then\ (NEq\ (CN\ 0\ c\ t))$
 $\quad\quad\quad else\ (NEq\ (CN\ 0\ (-c)\ (Neg\ t))))$

lemma $lt\text{-}mono: \forall\ a\ n\ s. Inum\ (x\#bs)\ a = Inum\ (x\#bs)\ (CN\ 0\ n\ s) \wedge numbound0\ s \longrightarrow Ifm\ (x\#bs)\ (lt\ n\ s) = Ifm\ (x\#bs)\ (Lt\ a)$
 $(is\ \forall\ a\ n\ s. ?N\ a = ?N\ (CN\ 0\ n\ s) \wedge \longrightarrow ?I\ (lt\ n\ s) = ?I\ (Lt\ a))$

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: $lt\text{-}def$))

(cases $n > 0$, simp-all add: $lt\text{-}def$ ring-simps myless[rule-format, **where** $b=0$])

qed

lemma $lt\text{-}l: isrlfm\ (rsplit\ lt\ a)$

by (rule $rsplit\text{-}l$ [**where** $f=lt$ **and** $a=a$], auto simp add: $lt\text{-}def$,
 $case\text{-}tac\ s$, simp-all, case-tac nat, simp-all)

lemma $le\text{-}mono: \forall\ a\ n\ s. Inum\ (x\#bs)\ a = Inum\ (x\#bs)\ (CN\ 0\ n\ s) \wedge numbound0\ s \longrightarrow Ifm\ (x\#bs)\ (le\ n\ s) = Ifm\ (x\#bs)\ (Le\ a)$ **(is** $\forall\ a\ n\ s. ?N\ a = ?N\ (CN\ 0\ n\ s) \wedge \longrightarrow ?I\ (le\ n\ s) = ?I\ (Le\ a)$)

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: $le\text{-}def$))

(cases $n > 0$, simp-all add: $le\text{-}def$ ring-simps myl[rule-format, **where** $b=0$])

qed

lemma $le\text{-}l: isrlfm\ (rsplit\ le\ a)$

by (rule $rsplit\text{-}l$ [**where** $f=le$ **and** $a=a$], auto simp add: $le\text{-}def$)
 $(case\text{-}tac\ s$, simp-all, case-tac nat, simp-all)

lemma $gt\text{-}mono: \forall\ a\ n\ s. Inum\ (x\#bs)\ a = Inum\ (x\#bs)\ (CN\ 0\ n\ s) \wedge numbound0\ s \longrightarrow Ifm\ (x\#bs)\ (gt\ n\ s) = Ifm\ (x\#bs)\ (Gt\ a)$ **(is** $\forall\ a\ n\ s. ?N\ a = ?N\ (CN\ 0\ n\ s) \wedge \longrightarrow ?I\ (gt\ n\ s) = ?I\ (Gt\ a)$)

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: $gt\text{-}def$))

(cases $n > 0$, simp-all add: $gt\text{-}def$ ring-simps myless[rule-format, **where** $b=0$])

qed

lemma $gt\text{-}l: isrlfm\ (rsplit\ gt\ a)$

by (rule $rsplit\text{-}l$ [**where** $f=gt$ **and** $a=a$], auto simp add: $gt\text{-}def$)
 $(case\text{-}tac\ s$, simp-all, case-tac nat, simp-all)

lemma *ge-mono*: $\forall a n s. \text{Inum } (x\#bs) a = \text{Inum } (x\#bs) (CN\ 0\ n\ s) \wedge \text{num-bound0 } s \longrightarrow \text{Ifm } (x\#bs) (ge\ n\ s) = \text{Ifm } (x\#bs) (Ge\ a) \text{ (is } \forall a n s. ?N\ a = ?N (CN\ 0\ n\ s) \wedge - \longrightarrow ?I (ge\ n\ s) = ?I (Ge\ a))$

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*: $\forall a n s. \text{Inum } (x\#bs) a = \text{Inum } (x\#bs) (CN\ 0\ n\ s) \wedge \text{num-bound0 } s \longrightarrow \text{Ifm } (x\#bs) (eq\ n\ s) = \text{Ifm } (x\#bs) (Eq\ a) \text{ (is } \forall a n s. ?N\ a = ?N (CN\ 0\ n\ s) \wedge - \longrightarrow ?I (eq\ n\ s) = ?I (Eq\ a))$

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-tac nat*, *simp-all*)

lemma *neq-mono*: $\forall a n s. \text{Inum } (x\#bs) a = \text{Inum } (x\#bs) (CN\ 0\ n\ s) \wedge \text{num-bound0 } s \longrightarrow \text{Ifm } (x\#bs) (neq\ n\ s) = \text{Ifm } (x\#bs) (NEq\ a) \text{ (is } \forall a n s. ?N\ a = ?N (CN\ 0\ n\ s) \wedge - \longrightarrow ?I (neq\ n\ s) = ?I (NEq\ a))$

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-tac nat*, *simp-all*)

lemma *small-le*:

assumes *u0*: $0 \leq u$ **and** *u1*: $u < 1$

shows $(-u \leq \text{real } (n::\text{int})) = (0 \leq n)$

using *u0 u1* **by** *auto*

lemma *small-lt*:

assumes *u0*: $0 \leq u$ **and** *u1*: $u < 1$

shows $(\text{real } (n::\text{int}) < \text{real } (m::\text{int}) - u) = (n < m)$

using *u0 u1* **by** *auto*

lemma *rdvd01-cs*:

assumes *up*: $u \geq 0$ **and** *u1*: $u < 1$ **and** *np*: $\text{real } n > 0$
shows $(\text{real } (i::\text{int}) \text{ rdvd } \text{real } (n::\text{int}) * u - s) = (\exists j \in \{0 \dots n - 1\}. \text{real } n * u = s - \text{real } (\text{floor } s) + \text{real } j \wedge \text{real } i \text{ rdvd } \text{real } (j - \text{floor } s))$ (**is** *?lhs* = *?rhs*)
proof –
let *?ss* = $s - \text{real } (\text{floor } s)$
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** $ss0::?ss \geq 0$ **and** $ss1::?ss < 1$
by (*auto simp add: myl* [*rule-format*, **where** $b=s$, *symmetric*] *myless* [*rule-format*, **where** $a=?ss$])
from *np* **have** $n0::\text{real } n \geq 0$ **by** *simp*
from *mult-left-mono* [*OF up n0*] *mult-strict-left-mono* [*OF u1 np*]
have $nu0::\text{real } n * u - s \geq -s$ **and** $nun::\text{real } n * u - s < \text{real } n - s$ **by** *auto*
from *int-rdvd-real* [**where** $i=i$ **and** $x=\text{real } (n::\text{int}) * u - s$]
have $\text{real } i \text{ rdvd } \text{real } n * u - s =$
 $(i \text{ dvd } \text{floor } (\text{real } n * u - s) \wedge (\text{real } (\text{floor } (\text{real } n * u - s)) = \text{real } n * u - s))$
(is $- = (?DE)$ **is** $- = (?D \wedge ?E)$ **) by** *simp*
also have $\dots = (?DE \wedge \text{real } (\text{floor } (\text{real } n * u - s) + \text{floor } s) \geq -?ss$
 $\wedge \text{real } (\text{floor } (\text{real } n * u - s) + \text{floor } s) < \text{real } n - ?ss)$ (**is** $- = (?DE \wedge \text{real } ?a \geq$
 $- \wedge \text{real } ?a < -)$)
using *nu0 nun* **by** *auto*
also have $\dots = (?DE \wedge ?a \geq 0 \wedge ?a < n)$ **by** (*simp only: small-le* [*OF ss0 ss1*]
small-lt [*OF ss0 ss1*])
also have $\dots = (?DE \wedge (\exists j \in \{0 \dots (n - 1)\}. ?a = j))$ **by** *simp*
also have $\dots = (?DE \wedge (\exists j \in \{0 \dots (n - 1)\}. \text{real } (\lfloor \text{real } n * u - s \rfloor) = \text{real } j$
 $- \text{real } \lfloor s \rfloor))$
by (*simp only: ring-simps real-of-int-diff* [*symmetric*] *real-of-int-inject del: real-of-int-diff*)
also have $\dots = ((\exists j \in \{0 \dots (n - 1)\}. \text{real } n * u - s = \text{real } j - \text{real } \lfloor s \rfloor \wedge \text{real } i \text{ rdvd } \text{real } n * u - s))$ **using** *int-rdvd-iff* [**where** $i=i$ **and** $t=\lfloor \text{real } n * u - s \rfloor$]
by (*auto cong: conj-cong*)
also have $\dots = ?rhs$ **by** (*simp cong: conj-cong*) (*simp add: ring-simps*)
finally show *?thesis* .
qed

definition

DVDJ:: $\text{int} \Rightarrow \text{int} \Rightarrow \text{num} \Rightarrow \text{fm}$

where

DVDJ-def: $\text{DVDJ } i \ n \ s = (\text{foldr } \text{disj } (\text{map } (\lambda j. \text{conj } (\text{Eq } (\text{CN } 0 \ n \ (\text{Add } s \ (\text{Sub } (\text{Floor } (\text{Neg } s)) \ (C \ j)))))) \ (\text{Dvd } i \ (\text{Sub } (C \ j) \ (\text{Floor } (\text{Neg } s)))))) \ (\text{iupt}(0, n - 1))) \ F)$

definition

NDVDJ:: $\text{int} \Rightarrow \text{int} \Rightarrow \text{num} \Rightarrow \text{fm}$

where

NDVDJ-def: $\text{NDVDJ } i \ n \ s = (\text{foldr } \text{conj } (\text{map } (\lambda j. \text{disj } (\text{NEq } (\text{CN } 0 \ n \ (\text{Add } s \ (\text{Sub } (\text{Floor } (\text{Neg } s)) \ (C \ j)))))) \ (\text{NDvd } i \ (\text{Sub } (C \ j) \ (\text{Floor } (\text{Neg } s)))))) \ (\text{iupt}(0, n - 1))) \ T)$

lemma DVDJ-DVD:

assumes $xp:x \geq 0$ **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 $?f = \lambda j. conj\ (Eq\ (CN\ 0\ n\ (Add\ s\ (Sub\ (Floor\ (Neg\ s))\ (C\ j))))\ (Dvd\ i\ (Sub\ (C\ j)\ (Floor\ (Neg\ s))))$
let $?s = Inum\ (x\#bs)\ s$
from $foldr-disj-map[where\ xs=iupt(0,n-1)\ and\ bs=x\#bs\ and\ f=?f]$
have $Ifm\ (x\#bs)\ (DVDJ\ i\ n\ s) = (\exists\ j \in \{0 \dots (n-1)\}. Ifm\ (x\#bs)\ (?f\ j))$
by $(simp\ add: iupt-set\ np\ DVDJ-def\ del: iupt.simps)$
also have $\dots = (\exists\ j \in \{0 \dots (n-1)\}. real\ n * x = (-\ ?s) - real\ (floor\ (-\ ?s)) + real\ j \wedge real\ i\ rdvd\ real\ (j - floor\ (-\ ?s)))$ **by** $(simp\ add: ring-simps\ diff-def[symmetric])$
also from $rdvd01-cs[OF\ xp\ x1\ np, where\ i=i\ and\ s=-?s]$
have $\dots = (real\ i\ rdvd\ real\ n * x - (-?s))$ **by** $simp$
finally show $?thesis$ **by** $simp$
qed

lemma NDVDJ-NDVD:

assumes $xp:x \geq 0$ **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 $?f = \lambda j. disj\ (NEq\ (CN\ 0\ n\ (Add\ s\ (Sub\ (Floor\ (Neg\ s))\ (C\ j))))\ (NDvd\ i\ (Sub\ (C\ j)\ (Floor\ (Neg\ s))))$
let $?s = Inum\ (x\#bs)\ s$
from $foldr-conj-map[where\ xs=iupt(0,n-1)\ and\ bs=x\#bs\ and\ f=?f]$
have $Ifm\ (x\#bs)\ (NDVDJ\ i\ n\ s) = (\forall\ j \in \{0 \dots (n-1)\}. Ifm\ (x\#bs)\ (?f\ j))$
by $(simp\ add: iupt-set\ np\ NDVDJ-def\ del: iupt.simps)$
also have $\dots = (\neg\ (\exists\ j \in \{0 \dots (n-1)\}. real\ n * x = (-\ ?s) - real\ (floor\ (-\ ?s)) + real\ j \wedge real\ i\ rdvd\ real\ (j - floor\ (-\ ?s))))$ **by** $(simp\ add: ring-simps\ diff-def[symmetric])$
also from $rdvd01-cs[OF\ xp\ x1\ np, where\ i=i\ and\ s=-?s]$
have $\dots = (\neg\ (real\ i\ rdvd\ real\ n * x - (-?s)))$ **by** $simp$
finally show $?thesis$ **by** $simp$
qed

lemma foldr-disj-map-rlfm2:

assumes $lf: \forall\ n. isrlfm\ (f\ n)$
shows $isrlfm\ (foldr\ disj\ (map\ f\ xs)\ F)$
using lf **by** $(induct\ xs, auto)$
lemma foldr-And-map-rlfm2:
assumes $lf: \forall\ n. isrlfm\ (f\ n)$
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 $?f = \lambda j. conj\ (Eq\ (CN\ 0\ n\ (Add\ s\ (Sub\ (Floor\ (Neg\ s))\ (C\ j))))$

(*Dvd i (Sub (C j) (Floor (Neg s))))*)

have *th*: $\forall j. \text{isrlfm } (?f j)$ **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*= $\lambda j. \text{disj } (NEq (CN 0 n (Add s (Sub (Floor (Neg s)) (C j))))$
 $(NDvd i (Sub (C j) (Floor (Neg s))))$
have *th*: $\forall j. \text{isrlfm } (?f j)$ **using** *nb np* **by** *auto*
from *NDVDJ-def foldr-And-map-rlfm2[OF th]* **show** *?thesis* **by** *auto*
qed

definition *DVD* :: $\text{int} \Rightarrow \text{int} \Rightarrow \text{num} \Rightarrow \text{fm}$ **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* :: $\text{int} \Rightarrow \text{int} \Rightarrow \text{num} \Rightarrow \text{fm}$ **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*: $0 \leq x$ **and** *x1*: $x < 1$
shows $\forall a n s. \text{Inum } (x \# bs) a = \text{Inum } (x \# bs) (CN 0 n s) \wedge \text{numbound0 } s \longrightarrow$
 $\text{Ifm } (x \# bs) (DVD i n s) = \text{Ifm } (x \# bs) (Dvd i a)$
(is $\forall a n s. ?N a = ?N (CN 0 n s) \wedge - \longrightarrow ?I (DVD i n s) = ?I (Dvd i a)$)
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 $i=0 \vee (i \neq 0 \wedge n=0) \vee (i \neq 0 \wedge n < 0) \vee (i \neq 0 \wedge n > 0)$ **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*: $i \neq 0$ **and** $n=0$ **hence** *?th* **by** (*simp add: H DVD-def*)
}
moreover {**assume** *inz*: $i \neq 0$ **and** $n < 0$ **hence** *?th*
by (*simp add: DVD-def H DVDJ-DVD[OF xp x1] rdvd-abs1*
 $\text{rdvd-minus[where } d=i \text{ and } t=\text{real } n * x + \text{Inum } (x \# bs) s]$) }
moreover {**assume** *inz*: $i \neq 0$ **and** $n > 0$ **hence** *?th* **by** (*simp add: DVD-def H*
 $\text{DVDJ-DVD[OF xp x1] rdvd-abs1}$)}
ultimately show *?th* **by** *blast*
qed

lemma *NDVD-mono*: **assumes** $xp: 0 \leq x$ **and** $x1: x < 1$
shows $\forall a n s. \text{Inum } (x \# bs) a = \text{Inum } (x \# bs) (CN\ 0\ n\ s) \wedge \text{numbound0 } s \longrightarrow$
 $\text{Ifm } (x \# bs) (NDVD\ i\ n\ s) = \text{Ifm } (x \# bs) (NDvd\ i\ a)$
 $(\text{is } \forall a n s. ?N\ a = ?N\ (CN\ 0\ n\ s) \wedge - \longrightarrow ?I\ (NDVD\ i\ n\ s) = ?I\ (NDvd\ i\ a))$
proof(*clarify*)
fix $a\ n\ s$
assume $H: ?N\ a = ?N\ (CN\ 0\ n\ s)$ **and** $nb: \text{numbound0 } s$
let $?th = ?I\ (NDVD\ i\ n\ s) = ?I\ (NDvd\ i\ a)$
have $i=0 \vee (i \neq 0 \wedge n=0) \vee (i \neq 0 \wedge n < 0) \vee (i \neq 0 \wedge n > 0)$ **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** $inz: i \neq 0$ **and** $n=0$ **hence** $?th$ **by** (*simp add: H NDVD-def*)}

lemma *DVD-l*: $\text{isrlfm } (rsplit\ (DVD\ i)\ a)$
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*: $\text{isrlfm } (rsplit\ (NDVD\ i)\ a)$
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 $\text{rlfm} :: \text{fm} \Rightarrow \text{fm}$
recdef rlfm *measure fmsize*
 $\text{rlfm } (And\ p\ q) = \text{conj } (\text{rlfm } p) (\text{rlfm } q)$
 $\text{rlfm } (Or\ p\ q) = \text{disj } (\text{rlfm } p) (\text{rlfm } q)$
 $\text{rlfm } (Imp\ p\ q) = \text{disj } (\text{rlfm } (NOT\ p)) (\text{rlfm } q)$
 $\text{rlfm } (Iff\ p\ q) = \text{disj } (\text{conj } (\text{rlfm } p) (\text{rlfm } q)) (\text{conj } (\text{rlfm } (NOT\ p)) (\text{rlfm } (NOT\ q)))$
 $\text{rlfm } (Lt\ a) = \text{rsplit } lt\ a$
 $\text{rlfm } (Le\ a) = \text{rsplit } le\ a$
 $\text{rlfm } (Gt\ a) = \text{rsplit } gt\ a$
 $\text{rlfm } (Ge\ a) = \text{rsplit } ge\ a$
 $\text{rlfm } (Eq\ a) = \text{rsplit } eq\ a$
 $\text{rlfm } (NEq\ a) = \text{rsplit } neq\ a$
 $\text{rlfm } (Dvd\ i\ a) = \text{rsplit } (\lambda t. DVD\ i\ t)\ a$
 $\text{rlfm } (NDvd\ i\ a) = \text{rsplit } (\lambda t. NDVD\ i\ t)\ a$
 $\text{rlfm } (NOT\ (And\ p\ q)) = \text{disj } (\text{rlfm } (NOT\ p)) (\text{rlfm } (NOT\ q))$
 $\text{rlfm } (NOT\ (Or\ p\ q)) = \text{conj } (\text{rlfm } (NOT\ p)) (\text{rlfm } (NOT\ q))$


```

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 :  $\llbracket \text{isatom } p ; \text{bound0 } p \rrbracket \implies \text{isrlfm } p$ 
  by (induct p rule: isrlfm.induct, auto)
lemma igcd-le1: assumes ip:  $0 < i$  shows  $\text{igcd } i \ j \leq i$ 
proof-
  from igcd-dvd1 have th:  $\text{igcd } i \ j \ \text{dvd } i$  by blast
  from zdvd-imp-le[OF th ip] show ?thesis .
qed

```

```

lemma simpfm-rl:  $\text{isrlfm } p \implies \text{isrlfm } (\text{simpfm } p)$ 
proof (induct p)
  case (Lt a)
  hence bound0 (Lt a)  $\vee (\exists \ c \ e. \ a = \text{CN } 0 \ c \ e \wedge c > 0 \wedge \text{numbound0 } e)$ 
    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))  $\neq 1$  and cnz:numgcd (CN 0 c
(simpnum e))  $\neq 0$ 
      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))  $\leq c$ 
        by (simp add: numgcd-def igcd-le1)
      from prems have th':  $c \neq 0$  by auto
      from prems have cp:  $c \geq 0$  by simp
      from zdiv-mono2[OF cp th1 th, simplified zdiv-self[OF th']]
      have  $0 < c \ \text{div} \ \text{numgcd } (\text{CN } 0 \ c \ (\text{simpnum } e))$  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)  $\vee$  ( $\exists c e. a = CN\ 0\ c\ e \wedge c > 0 \wedge numbound0\ e$ )
  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))  $\neq$  1 and cnz:numgcd (CN 0 c
(simpnum e))  $\neq$  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))  $\leq$  c
      by (simp add: numgcd-def igcd-le1)
    from prems have th': c $\neq$ 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 (Gt a)
hence bound0 (Gt a)  $\vee$  ( $\exists c e. a = CN\ 0\ c\ e \wedge c > 0 \wedge numbound0\ e$ )
  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))  $\neq$  1 and cnz:numgcd (CN 0 c
(simpnum e))  $\neq$  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))  $\leq$  c
      by (simp add: numgcd-def igcd-le1)
    from prems have th': c $\neq$ 0 by auto
    from prems have cp: c  $\geq$  0 by simp
  }
  with prems have ?case
    by (cases simpnum a, auto simp add: Let-def)
  ultimately show ?case by blast
}

```

```

    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)  $\vee$  ( $\exists$  c e. a = CN 0 c e  $\wedge$  c > 0  $\wedge$  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))  $\neq$  1 and cnz:numgcd (CN 0 c
(simpnum e))  $\neq$  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))  $\leq$  c
      by (simp add: numgcd-def igcd-le1)
    from prems have th': c $\neq$ 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 (Eq a)
hence bound0 (Eq a)  $\vee$  ( $\exists$  c e. a = CN 0 c e  $\wedge$  c > 0  $\wedge$  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))  $\neq$  1 and cnz:numgcd (CN 0 c
(simpnum e))  $\neq$  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))  $\leq$  c
  }
}

```

```

      by (simp add: numgcd-def igcd-le1)
    from prems have th':  $c \neq 0$  by auto
    from prems have cp:  $c \geq 0$  by simp
    from zdiv-mono2[OF cp th1 th, simplified zdiv-self[OF th']]
      have  $0 < c \text{ div numgcd } (CN\ 0\ c\ (simpnum\ e))$  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)  $\vee (\exists\ c\ e. a = CN\ 0\ c\ e \wedge c > 0 \wedge numbound0\ e)$ 
  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))  $\neq 1$  and cnz:numgcd (CN 0 c
(simpnum e))  $\neq 0$ 
    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))  $\leq c$ 
      by (simp add: numgcd-def igcd-le1)
    from prems have th':  $c \neq 0$  by auto
    from prems have cp:  $c \geq 0$  by simp
    from zdiv-mono2[OF cp th1 th, simplified zdiv-self[OF th']]
      have  $0 < c \text{ div numgcd } (CN\ 0\ c\ (simpnum\ e))$  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 ≤ x and x1: x < 1*
shows $(\text{Ifm } (x\#bs) (\text{rlfm } p) = \text{Ifm } (x\# bs) p) \wedge \text{isrlfm } (\text{rlfm } p)$
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 $\exists z. \forall x < z. \text{Ifm } (x\#bs) (\text{minusinf } p) = \text{Ifm } (x\#bs) p$ (**is** $\exists z. \forall x. ?P \ z \ x \ p$)
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*
fix *a*
let *?e = Inum (a#bs) e*
let *?z = (- ?e) / real c*
{fix *x*
assume *xz: x < ?z*
hence $(\text{real } c * x < - ?e)$
by (*simp only: pos-less-divide-eq[OF cp, where a=x and b=- ?e] mult-ac*)
hence $\text{real } c * x + ?e < 0$ **by** *arith*
hence $\text{real } c * x + ?e \neq 0$ **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 $\forall x < ?z. ?P \ ?z \ x \ (\text{Eq } (\text{CN } 0 \ c \ e))$ **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
fix a
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
fix a
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
fix a
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
  fix a
  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  $\forall x < ?z. ?P ?z x (Gt (CN 0 c e))$  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
  fix a
  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  $\forall x < ?z. ?P ?z x (Ge (CN 0 c e))$  by simp
  thus ?case by blast
qed simp-all

lemma rplusinf-inf:
  assumes lp: isrlfm p
  shows  $\exists z. \forall x > z. \text{Ifm } (x\#bs) (\text{plusinf } p) = \text{Ifm } (x\#bs) p$  (is  $\exists z. \forall x. ?P$ 
 $z x p$ )
  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
    fix a
    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
fix a
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
fix a
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
fix a
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  $\forall x > ?z. ?P ?z x (Le (CN 0 c e))$  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
  fix a
  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  $\forall x > ?z. ?P ?z x (Gt (CN 0 c e))$  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
    fix a
    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  $\exists x. \text{Ifm } (x\#bs) \text{ } p$ 
proof –
  from bound0-I [OF rminusinf-bound0[OF lp], where b=a and bs=bs] ex
  have th:  $\forall x. \text{Ifm } (x\#bs) \text{ } (minusinf p)$  by auto
  from rminusinf-inf[OF lp, where bs=bs]
  obtain z where z-def:  $\forall x < z. \text{Ifm } (x \# bs) \text{ } (minusinf p) = \text{Ifm } (x \# bs) \text{ } p$  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  $\exists x. \text{Ifm } (x\#bs) \text{ } p$ 
proof –
  from bound0-I [OF rplusinf-bound0[OF lp], where b=a and bs=bs] ex
  have th:  $\forall x. \text{Ifm } (x\#bs) \text{ } (plusinf p)$  by auto
  from rplusinf-inf[OF lp, where bs=bs]
  obtain z where z-def:  $\forall x > z. \text{Ifm } (x \# bs) \text{ } (plusinf p) = \text{Ifm } (x \# bs) \text{ } p$  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
   $\Upsilon :: fm \Rightarrow (num \times int) \text{ list}$ 
   $v :: fm \Rightarrow (num \times int) \Rightarrow fm$ 
recdef  $\Upsilon$  measure size
   $\Upsilon (And \ p \ q) = (\Upsilon \ p \ @ \ \Upsilon \ q)$ 
   $\Upsilon (Or \ p \ q) = (\Upsilon \ p \ @ \ \Upsilon \ q)$ 
   $\Upsilon (Eq \ (CN \ 0 \ c \ e)) = [(Neg \ e, c)]$ 
   $\Upsilon (NEq \ (CN \ 0 \ c \ e)) = [(Neg \ e, c)]$ 
   $\Upsilon (Lt \ (CN \ 0 \ c \ e)) = [(Neg \ e, c)]$ 
   $\Upsilon (Le \ (CN \ 0 \ c \ e)) = [(Neg \ e, c)]$ 
   $\Upsilon (Gt \ (CN \ 0 \ c \ e)) = [(Neg \ e, c)]$ 
   $\Upsilon (Ge \ (CN \ 0 \ c \ e)) = [(Neg \ e, c)]$ 

```

$\Upsilon p = []$

recdef v *measure size*

v ($And\ p\ q$) = $(\lambda\ (t,n).\ And\ (v\ p\ (t,n))\ (v\ q\ (t,n)))$
 v ($Or\ p\ q$) = $(\lambda\ (t,n).\ Or\ (v\ p\ (t,n))\ (v\ q\ (t,n)))$
 v ($Eq\ (CN\ 0\ c\ e)$) = $(\lambda\ (t,n).\ Eq\ (Add\ (Mul\ c\ t)\ (Mul\ n\ e)))$
 v ($NEq\ (CN\ 0\ c\ e)$) = $(\lambda\ (t,n).\ NEq\ (Add\ (Mul\ c\ t)\ (Mul\ n\ e)))$
 v ($Lt\ (CN\ 0\ c\ e)$) = $(\lambda\ (t,n).\ Lt\ (Add\ (Mul\ c\ t)\ (Mul\ n\ e)))$
 v ($Le\ (CN\ 0\ c\ e)$) = $(\lambda\ (t,n).\ Le\ (Add\ (Mul\ c\ t)\ (Mul\ n\ e)))$
 v ($Gt\ (CN\ 0\ c\ e)$) = $(\lambda\ (t,n).\ Gt\ (Add\ (Mul\ c\ t)\ (Mul\ n\ e)))$
 v ($Ge\ (CN\ 0\ c\ e)$) = $(\lambda\ (t,n).\ Ge\ (Add\ (Mul\ c\ t)\ (Mul\ n\ e)))$
 $v\ p = (\lambda\ (t,n). p)$

lemma v - I : **assumes** lp : $isrlfm\ p$

and np : $real\ n > 0$ **and** nbt : $numbound0\ t$

shows $(Ifm\ (x\#bs)\ (v\ p\ (t,n))) = Ifm\ (((Inum\ (x\#bs)\ t)/(real\ n))\#bs)\ p) \wedge$
 $bound0\ (v\ p\ (t,n))$ **is** $(?I\ x\ (v\ p\ (t,n))) = ?I\ ?u\ p \wedge ?B\ p$ **is** $(- = ?I\ (?t/?n)\ p)$
 $\wedge -$ **is** $(- = ?I\ (?N\ x\ t\ /-) p) \wedge -$

using lp

proof($induct\ p\ rule: v.induct$)

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** $\dots = (?n*(real\ c * (?t/?n)) + ?n*(?N\ x\ e) < 0)$

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** $\dots = (real\ c * ?t + ?n * (?N\ x\ e) < 0)$

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 $?I\ ?u\ (Le\ (CN\ 0\ c\ e)) = (real\ c * (?t/?n) + (?N\ x\ e) \leq 0)$

using $numbound0-I[OF\ nb, where\ bs=bs\ and\ b=?u\ and\ b'=x]$ **by** $simp$

also **have** $\dots = (?n*(real\ c * (?t/?n)) + ?n*(?N\ x\ e) \leq 0)$

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** $\dots = (real\ c * ?t + ?n * (?N\ x\ e) \leq 0)$

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** $\dots = (?n*(real\ c * (?t/?n)) + ?n*(?N\ x\ e) > 0)$

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** $\dots = (real\ c * ?t + ?n * (?N\ x\ e) > 0)$

```

    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  $\forall (t,k) \in \text{set } (\Upsilon p). \text{numbound0 } t \wedge k > 0$ 
using lp
by(induct p rule:  $\Upsilon$ .induct) auto

```

lemma rminusinf- Υ :

```

  assumes lp: isrlfm p
  and nmi:  $\neg (\text{Ifm } (a\#bs) (\text{minusinf } p))$  (is  $\neg (\text{Ifm } (a\#bs) (?M p))$ )
  and ex:  $\text{Ifm } (x\#bs) p$  (is  $?I x p$ )

```

shows $\exists (s,m) \in \text{set } (\Upsilon p). x \geq \text{Inum } (a\#bs) s / \text{real } m$ **(is** $\exists (s,m) \in ?U p.$
 $x \geq ?N a s / \text{real } m)$
proof –
have $\exists (s,m) \in \text{set } (\Upsilon p). \text{real } m * x \geq \text{Inum } (a\#bs) s$ **(is** $\exists (s,m) \in ?U p.$
 $\text{real } m * x \geq ?N a s)$
using $lp \text{ nmi } ex$
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** $smU: (s,m) \in \text{set } (\Upsilon p)$ **and** $mx: \text{real } m * x \geq ?N a$
 s **by** *blast*
from $\Upsilon\text{-l}[OF lp]$ smU **have** $mp: \text{real } m > 0$ **by** *auto*
from *pos-divide-le-eq* $[OF mp, \text{where } a=x \text{ and } b=?N a s, \text{symmetric}]$ mx **have**
 $x \geq ?N a s / \text{real } m$
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 $\exists (s,m) \in \text{set } (\Upsilon p). x \leq \text{Inum } (a\#bs) s / \text{real } m$ **(is** $\exists (s,m) \in ?U p.$
 $x \leq ?N a s / \text{real } m)$
proof –
have $\exists (s,m) \in \text{set } (\Upsilon p). \text{real } m * x \leq \text{Inum } (a\#bs) s$ **(is** $\exists (s,m) \in ?U p.$
 $\text{real } m * x \leq ?N a s)$
using $lp \text{ nmi } ex$
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** $smU: (s,m) \in \text{set } (\Upsilon p)$ **and** $mx: \text{real } m * x \leq ?N a$
 s **by** *blast*
from $\Upsilon\text{-l}[OF lp]$ smU **have** $mp: \text{real } m > 0$ **by** *auto*
from *pos-le-divide-eq* $[OF mp, \text{where } a=x \text{ and } b=?N a s, \text{symmetric}]$ mx **have**
 $x \leq ?N a s / \text{real } m$
by (*auto simp add: mult-commute*)
thus *?thesis* **using** smU **by** *auto*
qed

lemma *lin-dense*:

assumes $lp: isrlfm \ p$
and $noS: \forall t. l < t \wedge t < u \longrightarrow t \notin (\lambda (t,n). \text{Inum } (x\#bs) t / \text{real } n) \text{ ' set } (\Upsilon$
 $p)$
(is $\forall t. - \wedge - \longrightarrow t \notin (\lambda (t,n). ?N x t / \text{real } n) \text{ ' } (?U p)$)
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 $lp \text{ px } noS$
proof (*induct p rule: isrlfm.induct*)
case $(5 \ c \ e)$ **hence** $cp: \text{real } c > 0$ **and** $nb: \text{numbound0 } e$ **by** *simp+*

from prems have $x * \text{real } c + ?N \ x \ e < 0$ by (simp add: ring-simps)
 hence pxc: $x < (- ?N \ x \ e) / \text{real } c$
 by (simp only: pos-less-divide-eq[OF cp, where a=x and b=-?N x e])
 from prems have noSc: $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N \ x \ e) / \text{real } c$ by
 auto
 with ly yu have yne: $y \neq - ?N \ x \ e / \text{real } c$ by auto
 hence $y < (- ?N \ x \ e) / \text{real } c \vee y > (- ?N \ x \ e) / \text{real } c$ by auto
 moreover {assume y: $y < (- ?N \ x \ e) / \text{real } c$
 hence $y * \text{real } c < - ?N \ x \ e$
 by (simp add: pos-less-divide-eq[OF cp, where a=y and b=-?N x e,
 symmetric])
 hence $\text{real } c * y + ?N \ x \ e < 0$ 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: $y > (- ?N \ x \ e) / \text{real } c$
 with yu have eu: $u > (- ?N \ x \ e) / \text{real } c$ by auto
 with noSc ly yu have $(- ?N \ x \ e) / \text{real } c \leq l$ by (cases $(- ?N \ x \ e) / \text{real } c$
 > 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: $\text{real } c > 0$ and nb: numbound0 e by simp +
 from prems have $x * \text{real } c + ?N \ x \ e \leq 0$ by (simp add: ring-simps)
 hence pxc: $x \leq (- ?N \ x \ e) / \text{real } c$
 by (simp only: pos-le-divide-eq[OF cp, where a=x and b=-?N x e])
 from prems have noSc: $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N \ x \ e) / \text{real } c$ by
 auto
 with ly yu have yne: $y \neq - ?N \ x \ e / \text{real } c$ by auto
 hence $y < (- ?N \ x \ e) / \text{real } c \vee y > (- ?N \ x \ e) / \text{real } c$ by auto
 moreover {assume y: $y < (- ?N \ x \ e) / \text{real } c$
 hence $y * \text{real } c < - ?N \ x \ e$
 by (simp add: pos-less-divide-eq[OF cp, where a=y and b=-?N x e,
 symmetric])
 hence $\text{real } c * y + ?N \ x \ e < 0$ 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: $y > (- ?N \ x \ e) / \text{real } c$
 with yu have eu: $u > (- ?N \ x \ e) / \text{real } c$ by auto
 with noSc ly yu have $(- ?N \ x \ e) / \text{real } c \leq l$ by (cases $(- ?N \ x \ e) / \text{real } c$
 > 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: $\text{real } c > 0$ and nb: numbound0 e by simp +
 from prems have $x * \text{real } c + ?N \ x \ e > 0$ by (simp add: ring-simps)
 hence pxc: $x > (- ?N \ x \ e) / \text{real } c$
 by (simp only: pos-divide-less-eq[OF cp, where a=x and b=-?N x e])

from prems have noSc: $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$ by
 auto
 with ly yu have yne: $y \neq - ?N x e / \text{real } c$ by auto
 hence $y < (- ?N x e) / \text{real } c \vee y > (- ?N x e) / \text{real } c$ by auto
 moreover {assume y: $y > (- ?N x e) / \text{real } c$
 hence $y * \text{real } c > - ?N x e$
 by (simp add: pos-divide-less-eq[OF cp, where a=y and b=-?N x e,
 symmetric])
 hence $\text{real } c * y + ?N x e > 0$ 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: $y < (- ?N x e) / \text{real } c$
 with ly have eu: $l < (- ?N x e) / \text{real } c$ by auto
 with noSc ly yu have $(- ?N x e) / \text{real } c \geq u$ by (cases $(- ?N x e) / \text{real } c$
 > 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: $\text{real } c > 0$ and nb: numbound0 e by simp+
 from prems have $x * \text{real } c + ?N x e \geq 0$ by (simp add: ring-simps)
 hence pxc: $x \geq (- ?N x e) / \text{real } c$
 by (simp only: pos-divide-le-eq[OF cp, where a=x and b=-?N x e])
 from prems have noSc: $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$ by
 auto
 with ly yu have yne: $y \neq - ?N x e / \text{real } c$ by auto
 hence $y < (- ?N x e) / \text{real } c \vee y > (- ?N x e) / \text{real } c$ by auto
 moreover {assume y: $y > (- ?N x e) / \text{real } c$
 hence $y * \text{real } c > - ?N x e$
 by (simp add: pos-divide-less-eq[OF cp, where a=y and b=-?N x e,
 symmetric])
 hence $\text{real } c * y + ?N x e > 0$ 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: $y < (- ?N x e) / \text{real } c$
 with ly have eu: $l < (- ?N x e) / \text{real } c$ by auto
 with noSc ly yu have $(- ?N x e) / \text{real } c \geq u$ by (cases $(- ?N x e) / \text{real } c$
 > 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: $\text{real } c > 0$ and nb: numbound0 e by simp+
 from cp have cnz: $\text{real } c \neq 0$ by simp
 from prems have $x * \text{real } c + ?N x e = 0$ by (simp add: ring-simps)
 hence pxc: $x = (- ?N x e) / \text{real } c$
 by (simp only: nonzero-eq-divide-eq[OF cnz, where a=x and b=-?N x e])
 from prems have noSc: $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$ by
 auto

```

with lx xu have yne:  $x \neq - ?N x e / \text{real } c$  by auto
with pxc show ?case by simp
next
case (4 c e) hence cp:  $\text{real } c > 0$  and nb:  $\text{numbound0 } e$  by simp+
from cp have cnz:  $\text{real } c \neq 0$  by simp
from prems have noSc:  $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$  by
auto
with ly yu have yne:  $y \neq - ?N x e / \text{real } c$  by auto
hence y*  $\text{real } c \neq - ?N x e$ 
by (simp only: nonzero-eq-divide-eq[OF cnz, where a=y and b=-?N x e])
simp
hence y*  $\text{real } c + ?N x e \neq 0$  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:  $P (x::\text{real})$ 
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 =  $\text{Max } ?Mx$ 
let ?b =  $\text{Min } ?xM$ 
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-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 $\exists (l,n) \in set\ (\Upsilon\ p). \exists (s,m) \in set\ (\Upsilon\ p). ?I\ ((Inum\ (x\#bs)\ l / real\ n + Inum\ (x\#bs)\ s / real\ m) / 2)\ p$
 proof –
 let $?N = \lambda x\ t. Inum\ (x\#bs)\ t$
 let $?U = set\ (\Upsilon\ p)$
 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 $\exists (l,n) \in set\ (\Upsilon\ p). \exists (s,m) \in set\ (\Upsilon\ p). ?I\ ((?N\ a\ l / real\ n + ?N\ a\ s / real\ m) / 2)\ p$
 proof –
 let $?M = (\lambda (t,c). ?N\ a\ t / real\ c)\ ' ?U$
 have $fM: finite\ ?M$ by *auto*
 from $rminusinf-\Upsilon[OF\ lp\ nmi\ pa]\ rplusinf-\Upsilon[OF\ lp\ npi\ pa]$
 have $\exists (l,n) \in set\ (\Upsilon\ p). \exists (s,m) \in set\ (\Upsilon\ p). a \leq ?N\ x\ l / real\ n \wedge a \geq ?N\ x\ s / real\ m$ by *blast*
 then obtain $t\ n\ s\ m$ where

$tnU: (t,n) \in ?U$ and $smU: (s,m) \in ?U$
 and $xs1: a \leq ?N \ x \ s \ / \ real \ m$ and $tx1: a \geq ?N \ x \ t \ / \ real \ n$ by *blast*
 from $\Upsilon\text{-}l[OF \ lp] \ tnU \ smU \ numbound0\text{-}I[\text{where } bs=bs \text{ and } b=x \text{ and } b'=a]$
 $xs1 \ tx1$ have $xs: a \leq ?N \ a \ s \ / \ real \ m$ and $tx: a \geq ?N \ a \ t \ / \ real \ n$ by *auto*
 from tnU have $Mne: ?M \neq \{\}$ by *auto*
 hence $Une: ?U \neq \{\}$ by *simp*
 let $?l = Min \ ?M$
 let $?u = Max \ ?M$
 have $linM: ?l \in ?M$ using $fM \ Mne$ by *simp*
 have $uinM: ?u \in ?M$ using $fM \ Mne$ by *simp*
 have $tnM: ?N \ a \ t \ / \ real \ n \in ?M$ using tnU by *auto*
 have $smM: ?N \ a \ s \ / \ real \ m \in ?M$ using smU by *auto*
 have $lM: \forall \ t \in ?M. ?l \leq t$ using $Mne \ fM$ by *auto*
 have $Mu: \forall \ t \in ?M. t \leq ?u$ using $Mne \ fM$ by *auto*
 have $?l \leq ?N \ a \ t \ / \ real \ n$ using $tnM \ Mne$ by *simp* hence $lx: ?l \leq a$ using
 tx by *simp*
 have $?N \ a \ s \ / \ real \ m \leq ?u$ using $smM \ Mne$ by *simp* hence $xu: a \leq ?u$ using
 xs by *simp*
 from *finite-set-intervals2*[where $P=\lambda \ x. ?I \ x \ p, OF \ pa \ lx \ xu \ linM \ uinM \ fM \ lM \ Mu$]
 have $(\exists \ s \in ?M. ?I \ s \ p) \vee$
 $(\exists \ t1 \in ?M. \exists \ t2 \in ?M. (\forall \ y. t1 < y \wedge y < t2 \longrightarrow y \notin ?M) \wedge t1 < a \wedge a$
 $< t2 \wedge ?I \ a \ p) .$
 moreover { fix u assume $um: u \in ?M$ and $pu: ?I \ u \ p$
 hence $\exists \ (tu, nu) \in ?U. u = ?N \ a \ tu \ / \ real \ nu$ by *auto*
 then obtain $tu \ nu$ where $tuU: (tu, nu) \in ?U$ 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 $\exists \ t1 \in ?M. \exists \ t2 \in ?M. (\forall \ y. t1 < y \wedge y < t2 \longrightarrow y \notin ?M) \wedge t1$
 $< a \wedge a < t2 \wedge ?I \ a \ p$
 then obtain $t1$ and $t2$ where $t1M: t1 \in ?M$ and $t2M: t2 \in ?M$
 and $noM: \forall \ y. t1 < y \wedge y < t2 \longrightarrow y \notin ?M$ and $t1x: t1 < a$ and $xt2: a$
 $< t2$ and $px: ?I \ a \ p$
 by *blast*
 from $t1M$ have $\exists \ (t1u, t1n) \in ?U. t1 = ?N \ a \ t1u \ / \ real \ t1n$ by *auto*
 then obtain $t1u \ t1n$ where $t1uU: (t1u, t1n) \in ?U$ and $t1u: t1 = ?N \ a \ t1u$
 $/ \ real \ t1n$ by *blast*
 from $t2M$ have $\exists \ (t2u, t2n) \in ?U. t2 = ?N \ a \ t2u \ / \ real \ t2n$ by *auto*
 then obtain $t2u \ t2n$ where $t2uU: (t2u, t2n) \in ?U$ 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 $lnU: (l,n) \in ?U$ and $smU: (s,m) \in ?U$
and $pu: ?I ((?N\ a\ l\ /\ real\ n\ +\ ?N\ a\ s\ /\ real\ m)\ /\ 2)\ p$ by *blast*
from $lnU\ smU\ \Upsilon\text{-}l[OF\ lp]$ have $nbl: numbound0\ l$ and $nbs: numbound0\ s$ by
auto
from $numbound0\text{-}I[OF\ nbl, \text{ where } bs=bs \text{ and } b=a \text{ and } b'=x]$
 $numbound0\text{-}I[OF\ nbs, \text{ where } bs=bs \text{ and } b=a \text{ and } b'=x]$ *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 $(\exists\ x. Ifm\ (x\#bs)\ p) = ((Ifm\ (x\#bs)\ (minusinf\ p)) \vee (Ifm\ (x\#bs)\ (plusinf\ p))) \vee (\exists\ (t,n) \in set\ (\Upsilon\ p). \exists\ (s,m) \in set\ (\Upsilon\ p). Ifm\ (((Inum\ (x\#bs)\ t) /\ real\ n + (Inum\ (x\#bs)\ s) /\ real\ m) /\ 2)\#bs)\ p))$
(is $(\exists\ x. ?I\ x\ p) = (?M \vee ?P \vee ?F)$ is $?E = ?D$)
proof
assume $px: \exists\ x. ?I\ x\ p$
have $?M \vee ?P \vee (\neg\ ?M \wedge \neg\ ?P)$ by *blast*
moreover {assume $?M \vee ?P$ hence $?D$ by *blast*}
moreover {assume $nmi: \neg\ ?M$ and $npi: \neg\ ?P$
from $rinf\text{-}\Upsilon[OF\ lp\ nmi\ npi]$ have $?F$ using px by *blast* hence $?D$ by *blast*}
ultimately show $?D$ by *blast*

next

assume $?D$
moreover {assume $m: ?M$ from $rminusinf\text{-}ex[OF\ lp\ m]$ have $?E$.}
moreover {assume $p: ?P$ from $rplusinf\text{-}ex[OF\ lp\ p]$ have $?E$.}
moreover {assume $f: ?F$ hence $?E$ by *blast*}
ultimately show $?E$ by *blast*

qed

lemma fr-equiv:

assumes $lp: isrlfm\ p$
shows $(\exists\ x. Ifm\ (x\#bs)\ p) = ((Ifm\ (x\#bs)\ (minusinf\ p)) \vee (Ifm\ (x\#bs)\ (plusinf\ p))) \vee (\exists\ (t,k) \in set\ (\Upsilon\ p). \exists\ (s,l) \in set\ (\Upsilon\ p). Ifm\ (x\#bs)\ (v\ p\ (Add\ (Mul\ l\ t)\ (Mul\ k\ s)\ ,\ 2*k*l))))$
(is $(\exists\ x. ?I\ x\ p) = (?M \vee ?P \vee ?F)$ is $?E = ?D$)
proof
assume $px: \exists\ x. ?I\ x\ p$
have $?M \vee ?P \vee (\neg\ ?M \wedge \neg\ ?P)$ by *blast*
moreover {assume $?M \vee ?P$ hence $?D$ by *blast*}
moreover {assume $nmi: \neg\ ?M$ and $npi: \neg\ ?P$
let $?f = \lambda\ (t,n). Inum\ (x\#bs)\ t\ /\ real\ n$
let $?N = \lambda\ t. Inum\ (x\#bs)\ t$

```

{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 v-I[OF lp mnp st-nb, where x=x and bs=bs]
  have ?I x (v 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 (v 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 v-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 $(\exists (x::real). P x) = (\exists (i::int) (u::real). 0 \leq u \wedge u < 1 \wedge P (real\ i + u))$
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 ≤ x using real-of-int-floor-le by simp hence 0 ≤ ?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* ⇒ *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 $(Ifm\ bs\ (E\ p)) = (\exists\ (i::int). Ifm\ (real\ i\#bs)\ (E\ (And\ (And\ (Ge\ (CN\ 0\ 1\ (C\ 0))))\ (Lt\ (CN\ 0\ 1\ (C\ (-\ 1))))))\ (exsplit\ p)))$ (**is** *?lhs* = *?rhs*)
proof—
have *?rhs* = $(\exists\ (i::int). \exists\ x. 0 \leq x \wedge x < 1 \wedge Ifm\ (x\#(real\ i)\#bs)\ (exsplit\ p))$
by (*simp* *add*: *myless*[*rule-format*, **where** *b=1*] *myless*[*rule-format*, **where** *b=0*] *add-ac* *diff-def*)
also have $\dots = (\exists\ (i::int). \exists\ x. 0 \leq x \wedge x < 1 \wedge Ifm\ ((real\ i + x)\#bs)\ p)$

by (simp only: exsplit[OF qf] add-ac)
 also have ... = ($\exists x. \text{Ifm } (x\#bs) p$)
 by (simp only: real-ex-int-real01[where $P=\lambda x. \text{Ifm } (x\#bs) p$])
 finally show ?thesis by simp
 qed

constdefs ferrack01:: $fm \Rightarrow fm$
 ferrack01 $p \equiv$ (let $p' = \text{rlfm}(\text{And } (\text{And } (\text{Ge}(\text{CN } 0 \ 1 \ (C \ 0)))) \ (\text{Lt } (\text{CN } 0 \ 1 \ (C \ (-1)))))) \ p$;

$$U = \text{remdups}(\text{map } \text{simp-num-pair}$$

$$(\text{map } (\lambda ((t,n),(s,m)). (\text{Add } (\text{Mul } m \ t) (\text{Mul } n \ s), 2*n*m))$$

$$(\text{alluopairs } (\Upsilon \ p'))))$$

 in $\text{decr } (\text{evaldjf } (v \ p') \ U))$

lemma fr-eq-01:

assumes $qf: qfree \ p$
 shows ($\exists x. \text{Ifm } (x\#bs) (\text{And } (\text{And } (\text{Ge}(\text{CN } 0 \ 1 \ (C \ 0)))) \ (\text{Lt } (\text{CN } 0 \ 1 \ (C \ (-1)))))) \ p$) = ($\exists (t,n) \in \text{set } (\Upsilon \ (\text{rlfm } (\text{And } (\text{And } (\text{Ge}(\text{CN } 0 \ 1 \ (C \ 0)))) \ (\text{Lt } (\text{CN } 0 \ 1 \ (C \ (-1)))))) \ p$)). $\exists (s,m) \in \text{set } (\Upsilon \ (\text{rlfm } (\text{And } (\text{And } (\text{Ge}(\text{CN } 0 \ 1 \ (C \ 0)))) \ (\text{Lt } (\text{CN } 0 \ 1 \ (C \ (-1)))))) \ p$)). $\text{Ifm } (x\#bs) (v \ (\text{rlfm } (\text{And } (\text{And } (\text{Ge}(\text{CN } 0 \ 1 \ (C \ 0)))) \ (\text{Lt } (\text{CN } 0 \ 1 \ (C \ (-1)))))) \ p) (\text{Add } (\text{Mul } m \ t) (\text{Mul } n \ s), 2*n*m))$)
 (is ($\exists x. ?I \ x \ ?q$) = ?F)
proof—
 let $?rq = \text{rlfm } ?q$
 let $?M = ?I \ x \ (\text{minusinf } ?rq)$
 let $?P = ?I \ x \ (\text{plusinf } ?rq)$
 have $MF: ?M = \text{False}$
 apply (simp add: Let-def reducecoeff-def numgcd-def igcd-def rsplit-def ge-def lt-def conj-def disj-def)
 by (cases $\text{rlfm } p = \text{And } (\text{Ge } (\text{CN } 0 \ 1 \ (C \ 0))) \ (\text{Lt } (\text{CN } 0 \ 1 \ (C \ -1)))$, simp-all)
 have $PF: ?P = \text{False}$ apply (simp add: Let-def reducecoeff-def numgcd-def igcd-def rsplit-def ge-def lt-def conj-def disj-def)
 by (cases $\text{rlfm } p = \text{And } (\text{Ge } (\text{CN } 0 \ 1 \ (C \ 0))) \ (\text{Lt } (\text{CN } 0 \ 1 \ (C \ -1)))$, simp-all)
 have ($\exists x. ?I \ x \ ?q$) =
 (($?I \ x \ (\text{minusinf } ?rq)$) \vee ($?I \ x \ (\text{plusinf } ?rq)$) \vee ($\exists (t,n) \in \text{set } (\Upsilon \ ?rq). \exists (s,m) \in \text{set } (\Upsilon \ ?rq). ?I \ x \ (v \ ?rq \ (\text{Add } (\text{Mul } m \ t) (\text{Mul } n \ s), 2*n*m))$))
 (is ($\exists x. ?I \ x \ ?q$) = ($?M \vee ?P \vee ?F$) is $?E = ?D$)

proof

assume $\exists x. ?I \ x \ ?q$
 then obtain x where $qx: ?I \ x \ ?q$ by blast
 hence $xp: 0 \leq x$ 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 $\text{simpfm}[\text{where } p=?rq \text{ and } bs=x\#bs]$ by auto
 from qf have $qfq: \text{isrlfm } ?rq$
 by (auto simp add: rsplit-def lt-def ge-def rlfm-I[OF qf xp x1])

```

  with lqx fr-equiv[OF qfq] show ?M  $\vee$  ?P  $\vee$  ?F by blast
next
  assume D: ?D
  let ?U = set (Y ?rq )
  from MF PF D have ?F by auto
  then obtain t n s m where aU:(t,n)  $\in$  ?U and bU:(s,m)  $\in$  ?U and rqx: ?I
x (v ?rq (Add (Mul m t) (Mul n s), 2*n*m)) by blast
  from qf have lrq:isrlfm ?rqusing rlfm-l[OF qf]
  by (auto simp add: rsplit-def lt-def ge-def)
  from aU bU Y-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 v-I[OF lrq mnp stnb, where bs=bs and x=x], symmetric]
rqx
  have  $\exists$  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:  $\forall (t,n) \in \text{set } U. \text{numbound0 } t \wedge n > 0$ 
  shows (( $\lambda (t,n). \text{Inum } (x \# bs) t / \text{real } n$ ) ' (set (map ( $\lambda ((t,n),(s,m)). (\text{Add } (\text{Mul } m t) (\text{Mul } n s), 2*n*m)) (\text{alluopairs } U)))) = ((\lambda ((t,n),(s,m)). (\text{Inum } (x \# bs) t / \text{real } n + \text{Inum } (x \# bs) s / \text{real } m) / 2)$  ' (set U  $\times$  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 =  $\lambda t. \text{Inum } (x \# bs) t$ 
  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)
     $\in$  ( $\lambda((t, n), s, m).$ 
      (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) ∈ 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) (v p (Add (Mul m t) (Mul n s), 2*n*m))) = (∃ (t,n) ∈ U'. Ifm (x#bs) (v p (t,n)))
    (is ?lhs = ?rhs)
proof
  assume ?lhs

```


then obtain $t\ n\ s\ m$ **where** $tnU: (t,n) \in U$ **and** $smU: (s,m) \in U$ **and**
Pst: *Ifm* ($x \# bs$) ($v\ p\ (Add\ (Mul\ m\ t)\ (Mul\ n\ s), 2*n*m)$) **by** *blast*
let $?N = \lambda\ t.\ Inum\ (x \# bs)\ t$
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** $?g\ ((t,n),(s,m)) \in ?f\ 'U'$ **by** *blast*
hence $\exists\ (t',n') \in U'.\ ?g\ ((t,n),(s,m)) = ?f\ (t',n')$
by *auto* (*rule-tac x=(a,b) in bestI, auto*)
then obtain $t'\ n'$ **where** $tnU': (t',n') \in U'$ **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 $v-I[OF\ lp\ mnp\ stnb,\ where\ bs=bs\ and\ x=x]\ Pst$
have $Pst2: Ifm\ (Inum\ (x \# bs)\ (Add\ (Mul\ m\ t)\ (Mul\ n\ s)) / real\ (2 * n * m)$
 $\# bs) p$ **by** *simp*
from $conjunct1[OF\ v-I[OF\ lp\ np'\ tnb',\ where\ bs=bs\ and\ x=x],\ symmetric]$
 $th[simplified\ split-def\ fst-conv\ snd-conv,\ symmetric]\ Pst2[simplified\ st[symmetric]]$
have $Ifm\ (x \# bs)\ (v\ p\ (t',n'))$ **by** (*simp only: st*)
then show $?rhs$ **using** tnU' **by** *auto*
next
assume $?rhs$
then obtain $t'\ n'$ **where** $tnU': (t',n') \in U'$ **and** $Pt': Ifm\ (x \# bs)\ (v\ p\ (t',n'))$
by *blast*
from $tnU'\ UU'$ **have** $?f\ (t',n') \in ?g\ '(U \times U)$ **by** *blast*
hence $\exists\ ((t,n),(s,m)) \in (U \times U). ?f\ (t',n') = ?g\ ((t,n),(s,m))$
by *auto* (*rule-tac x=(a,b) in bestI, auto*)
then obtain $t\ n\ s\ m$ **where** $tnU: (t,n) \in U$ **and** $smU: (s,m) \in U$ **and**
 $th: ?f\ (t',n') = ?g\ ((t,n),(s,m))$ **by** *blast*
let $?N = \lambda\ t.\ Inum\ (x \# bs)\ t$
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 $v-I[OF\ lp\ np'\ tnb',\ where\ bs=bs\ and\ x=x,\ simplified\ th[simplified\ split-def\ fst-conv\ snd-conv]\ st]\ Pt'$
have $Pst2: Ifm\ (Inum\ (x \# bs)\ (Add\ (Mul\ m\ t)\ (Mul\ n\ s)) / real\ (2 * n * m)$
 $\# bs) p$ **by** *simp*
with $v-I[OF\ lp\ mnp\ stnb,\ where\ x=x\ and\ bs=bs]\ tnU\ smU$ **show** $?lhs$ **by** *blast*

qed

lemma ferrack01:

assumes qf : $qfree\ p$
shows $((\exists x. Ifm\ (x\#bs)\ (And\ (And\ (Ge\ (CN\ 0\ 1\ (C\ 0))))\ (Lt\ (CN\ 0\ 1\ (C\ (-\ 1))))))\ p)) = (Ifm\ bs\ (ferrack01\ p)) \wedge qfree\ (ferrack01\ p)\ (is\ (?lhs = ?rhs) \wedge -)$
proof–
let $?I = \lambda x\ p. Ifm\ (x\#bs)\ p$
fix x
let $?N = \lambda t. Inum\ (x\#bs)\ t$
let $?q = rlfm\ (And\ (And\ (Ge\ (CN\ 0\ 1\ (C\ 0))))\ (Lt\ (CN\ 0\ 1\ (C\ (-\ 1))))))\ p$
let $?U = \Upsilon\ ?q$
let $?Up = alluopairs\ ?U$
let $?g = \lambda ((t,n),(s,m)). (Add\ (Mul\ m\ t)\ (Mul\ n\ s),\ 2*n*m)$
let $?S = map\ ?g\ ?Up$
let $?SS = map\ simp-num-pair\ ?S$
let $?Y = remdups\ ?SS$
let $?f = (\lambda (t,n). ?N\ t\ /\ real\ n)$
let $?h = \lambda ((t,n),(s,m)). (?N\ t / real\ n + ?N\ s / real\ m) / 2$
let $?F = \lambda p. \exists a \in set\ (\Upsilon\ p). \exists b \in set\ (\Upsilon\ p). ?I\ x\ (v\ p\ (?g(a,b)))$
let $?ep = evaldjf\ (v\ ?q)\ ?Y$
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 : $set\ ?Up \leq (set\ ?U \times set\ ?U)$
by $simp$
from $\Upsilon-l[OF\ lq]$ have $U-l$: $\forall (t,n) \in set\ ?U. numbound0\ t \wedge n > 0$
from $U-l\ UpU$
have Up –: $\forall ((t,n),(s,m)) \in set\ ?Up. numbound0\ t \wedge n > 0 \wedge numbound0\ s \wedge m > 0$ **by** $auto$
hence Snb : $\forall (t,n) \in set\ ?S. numbound0\ t \wedge n > 0$
by ($auto\ simp\ add$: $mult-pos-pos$)
have $Y-l$: $\forall (t,n) \in set\ ?Y. numbound0\ t \wedge n > 0$
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

```

  ∀ 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  $\Upsilon$ -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 (v ?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 v-I[OF lq np tnb]
    have bound0 (v ?q (t,n)) by simp}
  thus ?thesis by blast
qed
hence ep-nb: bound0 ?ep using evaldjf-bound0[where xs=?Y and f=v ?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 (v ?q (t,n))) using  $\Upsilon$ -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=v ?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

consts unit:: fm ⇒ fm × num list × int
  unit p ≡ (let p' = zlfm p ; l = ζ p' ; q = And (Dvd l (CN 0 1 (C 0))) (aβ p'
l); d = δ q;
    B = remdups (map simpnum (β q)) ; a = remdups (map simpnum (α
q))

```

in if length $B \leq \text{length } a$ then (q, B, d) else $(\text{mirror } q, a, d)$

lemma *unit*: **assumes** qf : $qfree\ p$

shows $\bigwedge q\ B\ d. \text{unit } p = (q, B, d) \implies ((\exists (x::int). \text{Ifm } (real\ x\ \#bs)\ p) = (\exists (x::int). \text{Ifm } (real\ x\ \#bs)\ q)) \wedge (\text{Inum } (real\ i\ \#bs))\ ' \text{set } B = (\text{Inum } (real\ i\ \#bs))\ ' \text{set } (\beta\ q) \wedge d\beta\ q\ 1 \wedge d\delta\ q\ d \wedge d > 0 \wedge \text{iszlfm } q\ (real\ (i::int)\ \#bs) \wedge (\forall b \in \text{set } B. \text{numbound0 } b)$

proof–

fix $q\ B\ d$

assume qBd : $\text{unit } p = (q, B, d)$

let $?thes = ((\exists (x::int). \text{Ifm } (real\ x\ \#bs)\ p) = (\exists (x::int). \text{Ifm } (real\ x\ \#bs)\ q))$

\wedge

$\text{Inum } (real\ i\ \#bs)\ ' \text{set } B = \text{Inum } (real\ i\ \#bs)\ ' \text{set } (\beta\ q) \wedge$

$d\beta\ q\ 1 \wedge d\delta\ q\ d \wedge 0 < d \wedge \text{iszlfm } q\ (real\ i\ \#bs) \wedge (\forall b \in \text{set } B. \text{numbound0 } b)$

$b)$

let $?I = \lambda (x::int). p. \text{Ifm } (real\ x\ \#bs)\ p$

let $?p' = \text{zlfm } p$

let $?l = \zeta\ ?p'$

let $?q = \text{And } (Dvd\ ?l\ (CN\ 0\ 1\ (C\ 0)))\ (a\beta\ ?p'\ ?l)$

let $?d = \delta\ ?q$

let $?B = \text{set } (\beta\ ?q)$

let $?B' = \text{remdups } (\text{map } \text{simpnum } (\beta\ ?q))$

let $?A = \text{set } (\alpha\ ?q)$

let $?A' = \text{remdups } (\text{map } \text{simpnum } (\alpha\ ?q))$

from $\text{conjunct1}[OF\ \text{zlfm-I}[OF\ qf, \text{where } bs=bs]]$

have pp' : $\forall i. ?I\ i\ ?p' = ?I\ i\ p$ **by** *auto*

from $\text{iszlfm-gen}[OF\ \text{conjunct2}[OF\ \text{zlfm-I}[OF\ qf, \text{where } bs=bs\ \text{and } i=i]]]$

have lp' : $\forall (i::int). \text{iszlfm } ?p'\ (real\ i\ \#bs)$ **by** *simp*

hence lp'' : $\text{iszlfm } ?p'\ (real\ (i::int)\ \#bs)$ **by** *simp*

from $lp'\ \zeta[\text{where } p=?p'\ \text{and } bs=bs]$ **have** lp : $?l > 0$ **and** dl : $d\beta\ ?p'\ ?l$ **by** *auto*

from $a\beta\text{-ex}[\text{where } p=?p'\ \text{and } l=?l\ \text{and } bs=bs, OF\ lp''\ dl\ lp]$ pp'

have $pq\text{-ex}$: $(\exists (x::int). ?I\ x\ p) = (\exists x. ?I\ x\ ?q)$ **by** $(\text{simp add: int-rdvd-iff})$

from $lp''\ lp\ a\beta[OF\ lp''\ dl\ lp]$ **have** lq : $\text{iszlfm } ?q\ (real\ i\ \#bs)$ **and** uq : $d\beta\ ?q\ 1$

by $(\text{auto simp add: isint-def})$

from $\delta[OF\ lq]$ **have** dp : $?d > 0$ **and** dd : $d\delta\ ?q\ ?d$ **by** *blast+*

let $?N = \lambda t. \text{Inum } (real\ (i::int)\ \#bs)\ t$

have $?N\ ' \text{set } ?B' = ((?N\ o\ \text{simpnum})\ ' ?B)$ **by** $(\text{simp add: image-compose})$

also have $\dots = ?N\ ' ?B$ **using** $\text{simpnum-ci}[\text{where } bs=\text{real } i\ \#bs]$ **by** *auto*

finally have BB' : $?N\ ' \text{set } ?B' = ?N\ ' ?B$.

have $?N\ ' \text{set } ?A' = ((?N\ o\ \text{simpnum})\ ' ?A)$ **by** $(\text{simp add: image-compose})$

also have $\dots = ?N\ ' ?A$ **using** $\text{simpnum-ci}[\text{where } bs=\text{real } i\ \#bs]$ **by** *auto*

finally have AA' : $?N\ ' \text{set } ?A' = ?N\ ' ?A$.

from $\beta\text{-numbound0}[OF\ lq]$ **have** $B\text{-nb}$: $\forall b \in \text{set } ?B'. \text{numbound0 } b$

by $(\text{simp add: simpnum-numbound0})$

from $\alpha\text{-l}[OF\ lq]$ **have** $A\text{-nb}$: $\forall b \in \text{set } ?A'. \text{numbound0 } b$

by $(\text{simp add: simpnum-numbound0})$

{assume $\text{length } ?B' \leq \text{length } ?A'$

hence $q:q=?q$ **and** $B = ?B'$ **and** $d:d = ?d$

using qBd **by** $(\text{auto simp add: Let-def unit-def})$

```

with  $BB'$   $B$ -nb have  $b: ?N \text{ ' } (set\ B) = ?N \text{ ' } set\ (\beta\ q)$ 
and  $bn: \forall b \in set\ B. numbound0\ b$  by simp+
with  $pq\text{-}ex\ dp\ uq\ dd\ lq\ q\ d$  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 unit-def)
with  $AA'$   $mirror\alpha\beta[OF\ lq]$   $A$ -nb have  $b: ?N \text{ ' } (set\ B) = ?N \text{ ' } set\ (\beta\ q)$ 
and  $bn: \forall b \in set\ B. numbound0\ b$  by simp+
from  $mirror\text{-}ex[OF\ lq]\ pq\text{-}ex\ q$ 
have  $pqm\text{-}eq: (\exists (x::int). ?I\ x\ p) = (\exists (x::int). ?I\ x\ q)$  by simp
from  $lq\ uq\ q\ mirror\text{-}d\beta$  [where  $p=?q$  and  $bs=bs$  and  $a=real\ i$ ]
have  $lq': iszlfm\ q\ (real\ i\#bs)$  and  $uq: d\beta\ q\ 1$  by auto
from  $\delta[OF\ lq']\ mirror\text{-}\delta[OF\ lq]\ q\ d$  have  $dq:d\delta\ q\ d$  by auto
from  $pqm\text{-}eq\ b\ bn\ uq\ lq'\ dp\ dq\ q\ dp\ d$  have  $?thes$  by simp
}
ultimately show  $?thes$  by blast
qed

```

constdefs *cooper* :: $fm \Rightarrow fm$

```

cooper  $p \equiv$ 
(let  $(q,B,d) = unit\ p; js = iupt\ (1,d);$ 
 $mq = simpfm\ (minusinf\ q);$ 
 $md = evaldjf\ (\lambda j. simpfm\ (subst0\ (C\ j)\ mq))\ js$ 
in if  $md = T$  then  $T$  else
(let  $qd = evaldjf\ (\lambda t. simpfm\ (subst0\ t\ q))$ 
 $(remdups\ (map\ (\lambda (b,j). simpnum\ (Add\ b\ (C\ j)))$ 
 $[(b,j). b \leftarrow B, j \leftarrow js]))$ 
in  $decr\ (disj\ md\ qd)))$ 
lemma cooper: assumes  $qf: qfree\ p$ 
shows  $((\exists (x::int). Ifm\ (real\ x\#bs)\ p) = (Ifm\ bs\ (cooper\ p))) \wedge qfree\ (cooper\ p)$ 

(is  $(?lhs = ?rhs) \wedge -$ )
proof–

```

```

let  $?I = \lambda (x::int)\ p. Ifm\ (real\ x\#bs)\ p$ 
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  $?md = evaldjf\ (\lambda j. simpfm\ (subst0\ (C\ j)\ ?smq))\ ?js$ 
fix  $i$ 
let  $?N = \lambda t. Inum\ (real\ (i::int)\#bs)\ t$ 
let  $?bjs = [(b,j). b \leftarrow ?B, j \leftarrow ?js]$ 
let  $?sbjs = map\ (\lambda (b,j). simpnum\ (Add\ b\ (C\ j)))\ ?bjs$ 
let  $?qd = evaldjf\ (\lambda t. simpfm\ (subst0\ t\ ?q))\ (remdups\ ?sbjs)$ 

```

have $qbf:unit\ p = (?q, ?B, ?d)$ **by** *simp*
from $unit[OF\ qf\ qbf]$ **have** $pq\text{-}ex: (\exists (x::int). ?I\ x\ p) = (\exists (x::int). ?I\ x\ ?q)$ **and**

$B: ?N \text{ ' set } ?B = ?N \text{ ' set } (\beta\ ?q)$ **and**
 $uq: d\beta\ ?q\ 1$ **and** $dd: d\delta\ ?q\ ?d$ **and** $dp: ?d > 0$ **and**
 $lq: iszlfm\ ?q\ (real\ i\#bs)$ **and**
 $Bn: \forall\ b \in set\ ?B. numbound0\ b$ **by** *auto*
from $zlin\text{-}qfree[OF\ lq]$ **have** $qfq: qfree\ ?q$.
from $simpfm\text{-}qf[OF\ minusinf\text{-}qfree[OF\ qfq]]$ **have** $qfmq: qfree\ ?smq$.
have $jsnb: \forall\ j \in set\ ?js. numbound0\ (C\ j)$ **by** *simp*
hence $\forall\ j \in set\ ?js. bound0\ (subst0\ (C\ j)\ ?smq)$
by (*auto simp only: subst0-bound0[OF\ qfmq]*)
hence $th: \forall\ j \in set\ ?js. bound0\ (simpfm\ (subst0\ (C\ j)\ ?smq))$
by (*auto simp add: simpfm-bound0*)
from $evaldjf\text{-}bound0[OF\ th]$ **have** $mdb: bound0\ ?md$ **by** *simp*
from $Bn\ jsnb$ **have** $\forall\ (b,j) \in set\ ?bjs. numbound0\ (Add\ b\ (C\ j))$
by *simp*
hence $\forall\ (b,j) \in set\ ?bjs. numbound0\ (simpnum\ (Add\ b\ (C\ j)))$
using *simpnum-numbound0* **by** *blast*
hence $\forall\ t \in set\ ?sbjs. numbound0\ t$ **by** *simp*
hence $\forall\ t \in set\ (remdups\ ?sbjs). bound0\ (subst0\ t\ ?q)$
using *subst0-bound0[OF\ qfq]* **by** *auto*
hence $th': \forall\ t \in set\ (remdups\ ?sbjs). bound0\ (simpfm\ (subst0\ t\ ?q))$
using *simpfm-bound0* **by** *blast*
from $evaldjf\text{-}bound0\ [OF\ th']$ **have** $qdb: bound0\ ?qd$ **by** *simp*
from $mdb\ qdb$
have $mdqdb: bound0\ (disj\ ?md\ ?qd)$ **by** (*simp only: disj-def, cases\ ?md=T \vee\ ?qd=T, simp-all*)
from $trans\ [OF\ pq\text{-}ex\ cp\text{-}thm'[OF\ lq\ uq\ dd\ dp]]\ B$
have $?lhs = (\exists\ j \in \{1.. ?d\}. ?I\ j\ ?mq \vee (\exists\ b \in ?N \text{ ' set } ?B. Ifm\ ((b+ real\ j)\#bs)\ ?q))$ **by** *auto*
also have $\dots = ((\exists\ j \in set\ ?js. ?I\ j\ ?smq) \vee (\exists\ (b,j) \in (?N \text{ ' set } ?B \times set\ ?js). Ifm\ ((b+ real\ j)\#bs)\ ?q))$ **apply** (*simp only: iupt-set simpfm*) **by** *auto*
also have $\dots = ((\exists\ j \in set\ ?js. ?I\ j\ ?smq) \vee (\exists\ t \in (\lambda\ (b,j). ?N\ (Add\ b\ (C\ j))) \text{ ' set } ?bjs. Ifm\ (t\ \#bs)\ ?q))$ **by** *simp*
also have $\dots = ((\exists\ j \in set\ ?js. ?I\ j\ ?smq) \vee (\exists\ t \in (\lambda\ (b,j). ?N\ (simpnum\ (Add\ b\ (C\ j)))) \text{ ' set } ?bjs. Ifm\ (t\ \#bs)\ ?q))$ **by** (*simp only: simpnum-ci*)
also have $\dots = ((\exists\ j \in set\ ?js. ?I\ j\ ?smq) \vee (\exists\ t \in set\ ?sbjs. Ifm\ (?N\ t\ \#bs)\ ?q))$
by (*auto simp add: split-def*)
also have $\dots = ((\exists\ j \in set\ ?js. (\lambda\ j. ?I\ i\ (simpfm\ (subst0\ (C\ j)\ ?smq)))\ j) \vee (\exists\ t \in set\ (remdups\ ?sbjs). (\lambda\ t. ?I\ i\ (simpfm\ (subst0\ t\ ?q)))\ t))$ **by** (*simp only: simpfm subst0-I[OF\ qfq] simpfm Inum.simps subst0-I[OF\ qfmq] set-remdups*)
also have $\dots = ((?I\ i\ (evaldjf\ (\lambda\ j. simpfm\ (subst0\ (C\ j)\ ?smq))\ ?js)) \vee (?I\ i\ (evaldjf\ (\lambda\ t. simpfm\ (subst0\ t\ ?q))\ (remdups\ ?sbjs))))$ **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 $\dots = (\exists\ q \in set\ (disjuncts\ p). \exists\ (x::int). Ifm\ (real\ x\ \#bs)\ q)$
 using $cooper\ disjuncts\ qf[OF\ qf]$ by blast
 also have $\dots = (\exists\ (x::int). Ifm\ (real\ x\ \#bs)\ p)$ by (induct p rule: disjuncts.induct, auto)
 finally show $?thesis$ using $thqf$ by blast
 qed

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

lemma ϱ -cong: assumes $lp: iszlfm\ p\ (a\ \#bs)$
 and $RR: (\lambda(b,k). (Inum\ (a\ \#bs)\ b,k))\ 'R = (\lambda(b,k). (Inum\ (a\ \#bs)\ b,k))\ 'set\ (\varrho\ p)$
 shows $(\exists\ (e,c) \in R. \exists\ j \in \{1..c*(\delta\ p)\}. Ifm\ (a\ \#bs)\ (\sigma\ p\ c\ (Add\ e\ (C\ j)))) =$
 $(\exists\ (e,c) \in set\ (\varrho\ p). \exists\ j \in \{1..c*(\delta\ p)\}. Ifm\ (a\ \#bs)\ (\sigma\ p\ c\ (Add\ e\ (C\ j))))$
 (is $?lhs = ?rhs$)
 proof
 let $?d = \delta\ p$

assume $?lhs$ **then obtain** $e\ c\ j$ **where** $ecR: (e,c) \in R$ **and** $jD: j \in \{1 \dots c * ?d\}$
and $px: \text{Ifm } (a\#bs) (\sigma\ p\ c\ (\text{Add } e\ (C\ j)))$ **(is** $?sp\ c\ e\ j$) **by** *blast*
from ecR **have** $(\text{Inum } (a\#bs)\ e, c) \in (\lambda(b,k). (\text{Inum } (a\#bs)\ b, k))$ ‘ R **by** *auto*
hence $(\text{Inum } (a\#bs)\ e, c) \in (\lambda(b,k). (\text{Inum } (a\#bs)\ b, k))$ ‘ *set* $(\varrho\ p)$ **using** RR
by *simp*
hence $\exists (e', c') \in \text{set } (\varrho\ p). \text{Inum } (a\#bs)\ e = \text{Inum } (a\#bs)\ e' \wedge c = c'$ **by** *auto*
then obtain $e'\ c'$ **where** $ecRo: (e', c') \in \text{set } (\varrho\ p)$ **and** $ee': \text{Inum } (a\#bs)\ e =$
 $\text{Inum } (a\#bs)\ e'$
and $cc': c = c'$ **by** *blast*
from ee' **have** $tt': \text{Inum } (a\#bs)\ (\text{Add } e\ (C\ j)) = \text{Inum } (a\#bs)\ (\text{Add } e'\ (C\ j))$
by *simp*

from $\sigma\text{-cong}[OF\ lp\ tt', \text{ where } c=c]$ px **have** $px': ?sp\ c\ e'\ j$ **by** *simp*
from $ecRo\ jD\ px'\ cc'$ **show** $?rhs$ **apply** *auto*
by $(\text{rule-tac } x=(e', c') \text{ in } \text{bexI}, \text{simp-all})$
 $(\text{rule-tac } x=j \text{ in } \text{bexI}, \text{simp-all add: } cc'[\text{symmetric}])$

next

let $?d = \delta\ p$
assume $?rhs$ **then obtain** $e\ c\ j$ **where** $ecR: (e,c) \in \text{set } (\varrho\ p)$ **and** $jD: j \in \{1 \dots c * ?d\}$
and $px: \text{Ifm } (a\#bs) (\sigma\ p\ c\ (\text{Add } e\ (C\ j)))$ **(is** $?sp\ c\ e\ j$) **by** *blast*
from ecR **have** $(\text{Inum } (a\#bs)\ e, c) \in (\lambda(b,k). (\text{Inum } (a\#bs)\ b, k))$ ‘ *set* $(\varrho\ p)$
by *auto*
hence $(\text{Inum } (a\#bs)\ e, c) \in (\lambda(b,k). (\text{Inum } (a\#bs)\ b, k))$ ‘ R **using** RR **by** *simp*
hence $\exists (e', c') \in R. \text{Inum } (a\#bs)\ e = \text{Inum } (a\#bs)\ e' \wedge c = c'$ **by** *auto*
then obtain $e'\ c'$ **where** $ecRo: (e', c') \in R$ **and** $ee': \text{Inum } (a\#bs)\ e = \text{Inum } (a\#bs)\ e'$
and $cc': c = c'$ **by** *blast*
from ee' **have** $tt': \text{Inum } (a\#bs)\ (\text{Add } e\ (C\ j)) = \text{Inum } (a\#bs)\ (\text{Add } e'\ (C\ j))$
by *simp*
from $\sigma\text{-cong}[OF\ lp\ tt', \text{ where } c=c]$ px **have** $px': ?sp\ c\ e'\ j$ **by** *simp*
from $ecRo\ jD\ px'\ cc'$ **show** $?lhs$ **apply** *auto*
by $(\text{rule-tac } x=(e', c') \text{ in } \text{bexI}, \text{simp-all})$
 $(\text{rule-tac } x=j \text{ in } \text{bexI}, \text{simp-all add: } cc'[\text{symmetric}])$

qed

lemma $rl\text{-thm}'$:

assumes $lp: \text{iszlfm } p\ (\text{real } (i::\text{int})\#bs)$
and $R: (\lambda(b,k). (\text{Inum } (a\#bs)\ b, k))$ ‘ $R = (\lambda(b,k). (\text{Inum } (a\#bs)\ b, k))$ ‘ *set* $(\varrho\ p)$
shows $(\exists (x::\text{int}). \text{Ifm } (\text{real } x\#bs)\ p) = ((\exists j \in \{1 \dots \delta\ p\}. \text{Ifm } (\text{real } j\#bs)\ (\text{minusinf } p)) \vee (\exists (e,c) \in R. \exists j \in \{1 \dots c * (\delta\ p)\}. \text{Ifm } (a\#bs)\ (\sigma\ p\ c\ (\text{Add } e\ (C\ j))))))$
using $rl\text{-thm}[OF\ lp]\ \varrho\text{-cong}[OF\ \text{iszlfm-gen}[OF\ lp, \text{rule-format}, \text{ where } y=a]\ R]$
by *simp*

constdefs $\text{chooset}:: \text{fm} \Rightarrow \text{fm} \times ((\text{num} \times \text{int})\ \text{list}) \times \text{int}$

$\text{chooset } p \equiv (\text{let } q = \text{zlfm } p ; d = \delta\ q ;$

$B = \text{remdups } (\text{map } (\lambda (t,k). (\text{simpnum } t, k))\ (\varrho\ q)) ;$

$a = \text{remdups } (\text{map } (\lambda (t,k). (\text{simpnum } t,k)) (\alpha q))$
 $\text{in if length } B \leq \text{length } a \text{ then } (q,B,d) \text{ else } (\text{mirror } q, a,d)$

lemma chooset: **assumes** qf : $qfree\ p$

shows $\bigwedge q\ B\ d. \text{chooset } p = (q,B,d) \implies ((\exists (x::int). \text{Ifm } (\text{real } x\#bs) \ p) =$
 $(\exists (x::int). \text{Ifm } (\text{real } x\#bs) \ q)) \wedge ((\lambda(t,k). (\text{Inum } (\text{real } i\#bs) \ t,k)) \text{ ' set } B =$
 $(\lambda(t,k). (\text{Inum } (\text{real } i\#bs) \ t,k)) \text{ ' set } (q\ q)) \wedge (\delta\ q = d) \wedge d > 0 \wedge \text{iszfmlm } q (\text{real } (i::int)\#bs) \wedge (\forall (e,c) \in \text{set } B. \text{numbound0 } e \wedge c > 0)$

proof—

fix $q\ B\ d$

assume qBd : $\text{chooset } p = (q,B,d)$

let $?thes = ((\exists (x::int). \text{Ifm } (\text{real } x\#bs) \ p) = (\exists (x::int). \text{Ifm } (\text{real } x\#bs) \ q))$
 $\wedge ((\lambda(t,k). (\text{Inum } (\text{real } i\#bs) \ t,k)) \text{ ' set } B = (\lambda(t,k). (\text{Inum } (\text{real } i\#bs) \ t,k)) \text{ ' set } (q\ q)) \wedge (\delta\ q = d) \wedge d > 0 \wedge \text{iszfmlm } q (\text{real } (i::int)\#bs) \wedge (\forall (e,c) \in \text{set } B. \text{numbound0 } e \wedge c > 0)$

let $?I = \lambda (x::int). \text{Ifm } (\text{real } x\#bs) \ p$

let $?q = \text{zlfm } p$

let $?d = \delta\ ?q$

let $?B = \text{set } (q\ ?q)$

let $?f = \lambda (t,k). (\text{simpnum } t,k)$

let $?B' = \text{remdups } (\text{map } ?f (q\ ?q))$

let $?A = \text{set } (\alpha q\ ?q)$

let $?A' = \text{remdups } (\text{map } ?f (\alpha q\ ?q))$

from $\text{conjunct1}[OF\ \text{zlfm-I}[OF\ qf, \text{where } bs=bs]]$

have pp' : $\forall i. ?I\ i\ ?q = ?I\ i\ p$ **by** *auto*

hence $pq\text{-ex}$: $(\exists (x::int). ?I\ x\ p) = (\exists x. ?I\ x\ ?q)$ **by** *simp*

from $\text{iszfmlm-gen}[OF\ \text{conjunct2}[OF\ \text{zlfm-I}[OF\ qf, \text{where } bs=bs \text{ and } i=i]], \text{rule-format},$

where $y=\text{real } i]$

have lq : $\text{iszfmlm } ?q (\text{real } (i::int)\#bs) .$

from $\delta[OF\ lq]$ **have** dp : $?d > 0$ **by** *blast*

let $?N = \lambda (t,c). (\text{Inum } (\text{real } (i::int)\#bs) \ t,c)$

have $?N \text{ ' set } ?B' = ((?N\ o\ ?f) \text{ ' } ?B)$ **by** *(simp add: split-def image-compose)*

also have $\dots = ?N \text{ ' } ?B$

by *(simp add: split-def image-compose simpnum-ci[where bs=real i #bs] image-def)*

finally have BB' : $?N \text{ ' set } ?B' = ?N \text{ ' } ?B .$

have $?N \text{ ' set } ?A' = ((?N\ o\ ?f) \text{ ' } ?A)$ **by** *(simp add: split-def image-compose)*

also have $\dots = ?N \text{ ' } ?A$ **using** $\text{simpnum-ci[where bs=real i #bs]}$

by *(simp add: split-def image-compose simpnum-ci[where bs=real i #bs] image-def)*

finally have AA' : $?N \text{ ' set } ?A' = ?N \text{ ' } ?A .$

from $q\text{-l}[OF\ lq]$ **have** $B\text{-nb}$: $\forall (e,c) \in \text{set } ?B'. \text{numbound0 } e \wedge c > 0$

by *(simp add: simpnum-numbound0 split-def)*

from $\alpha q\text{-l}[OF\ lq]$ **have** $A\text{-nb}$: $\forall (e,c) \in \text{set } ?A'. \text{numbound0 } e \wedge c > 0$

by *(simp add: simpnum-numbound0 split-def)*

{assume $\text{length } ?B' \leq \text{length } ?A'$

hence $q:q=?q$ **and** $B = ?B'$ **and** $d:d = ?d$

using qBd **by** *(auto simp add: Let-def chooset-def)*

with $BB' B\text{-nb}$ **have** b : $?N \text{ ' (set } B) = ?N \text{ ' set } (q\ q)$

and bn : $\forall (e,c) \in \text{set } B. \text{numbound0 } e \wedge c > 0$ **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- $\alpha$ [OF lq] A-nb have b:?N ' (set B) = ?N ' set (q q)
  and bn:  $\forall (e,c) \in \text{set } B. \text{numbound0 } e \wedge c > 0$  by auto
  from mirror-ex[OF lq] pq-ex q
  have pqm-eq:  $(\exists (x::\text{int}). ?I x p) = (\exists (x::\text{int}). ?I x q)$  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- $\delta$ [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  $\equiv$  ( $\lambda (e,c). \text{evaldjf } (\lambda j. \text{simpfm } (\sigma p c (\text{Add } e (C j)))) (iupt (1,c*d))$ )
lemma stage:
  shows Ifm bs (stage p d (e,c)) =  $(\exists j \in \{1 .. c*d\}. \text{Ifm bs } (\sigma p c (\text{Add } e (C j))))$ 
  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 =  $\lambda j. \text{simpfm } (\sigma p c (\text{Add } e (C j)))$ 
  have th:  $\forall j \in \text{set } (iupt(1,c*d)). \text{bound0 } (?f j)$ 
  proof
    fix j
    from nb have nb':numbound0 (Add e (C j)) by simp
    from simpfm-bound0[OF  $\sigma$ -nb[OF lp nb', where k=c]]
    show bound0 (simpfm ( $\sigma p c (\text{Add } e (C j))$ )) .
  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 ( $\lambda j. \text{simpfm } (\text{subst0 } (C j) mq)$ ) (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  $((\exists (x::\text{int}). \text{Ifm } (\text{real } x\#bs) p) = (\text{Ifm bs } (\text{redlove } p))) \wedge \text{qfree } (\text{redlove } p)$ 

```

```

(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
fix i
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 (q ?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 mirefr :: fm ⇒ fm
mirefr ≡ (DJ cooper) o ferrack01 o simplfm o exsplit

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

lemma mirefr: ∀ bs p. qfree p ⟶ qfree (mirefr p) ∧ Ifm bs (mirefr p) = Ifm bs
(E p)
proof (clarsimp simp del: Ifm.simps)
  fix bs p
  assume qf: qfree p
  show qfree (mirefr p) ∧ (Ifm bs (mirefr 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))))) (simplfm
(exsplit p)))
    have ?rhs = (∃ (i::int). ∃ x. Ifm (x#real i#bs) ?es)
      using spliter[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 Mir.test1 ()
ML Mir.test2 ()
ML Mir.test3 ()
ML Mir.test4 ()
ML Mir.test5 ()
```

```
use mireif.ML
oracle mircfr-oracle (term) = ReflectedMir.mircfr-oracle
oracle mirlfr-oracle (term) = ReflectedMir.mirlfr-oracle
use mirtac.ML
setup MirTac.setup
```

```
lemma ALL (x::real). ( $\lfloor x \rfloor = \lceil x \rceil = (x = \text{real } \lfloor x \rfloor)$ )
apply mir
done
```

```
lemma ALL (x::real).  $\text{real } (2::\text{int})*x - (\text{real } (1::\text{int})) < \text{real } \lfloor x \rfloor + \text{real } \lceil x \rceil \wedge$ 
 $\text{real } \lfloor x \rfloor + \text{real } \lceil x \rceil \leq \text{real } (2::\text{int})*x + (\text{real } (1::\text{int}))$ 
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).  $\exists y \leq x. (\lfloor x \rfloor = \lceil y \rceil)$ 
apply mir
done
```

end

13 Type of indices

```
theory Code-Index
imports ATP-Linkup
begin
```

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

13.1 Datatype of indices

```
typedef index = UNIV :: nat set
morphisms nat-of-index index-of-nat by rule

lemma index-of-nat-nat-of-index [simp]:
  index-of-nat (nat-of-index k) = k
  by (rule nat-of-index-inverse)

lemma nat-of-index-index-of-nat [simp]:
  nat-of-index (index-of-nat n) = n
  by (rule index-of-nat-inverse)
  (unfold index-def, rule UNIV-I)

lemma index:
  ( $\bigwedge n::index. PROP P n$ )  $\equiv$  ( $\bigwedge n::nat. PROP P (index-of-nat n)$ )
proof
  fix n :: nat
  assume  $\bigwedge n::index. PROP P n$ 
  then show  $PROP P (index-of-nat n)$  .
next
  fix n :: index
  assume  $\bigwedge n::nat. PROP P (index-of-nat n)$ 
  then have  $PROP P (index-of-nat (nat-of-index n))$  .
  then show  $PROP P n$  by simp
qed

lemma index-case:
  assumes  $\bigwedge n. k = index-of-nat n \implies P$ 
  shows  $P$ 
  by (rule assms [of nat-of-index k]) simp

lemma index-induct-raw:
  assumes  $\bigwedge n. P (index-of-nat n)$ 
  shows  $P k$ 
proof -
  from assms have  $P (index-of-nat (nat-of-index k))$  .
  then show ?thesis by simp
```

qed

lemma *nat-of-index-inject* [simp]:
 $\text{nat-of-index } k = \text{nat-of-index } l \longleftrightarrow k = l$
 by (rule *nat-of-index-inject*)

lemma *index-of-nat-inject* [simp]:
 $\text{index-of-nat } n = \text{index-of-nat } m \longleftrightarrow n = m$
 by (auto intro!: *index-of-nat-inject simp add: index-def*)

instantiation *index* :: zero
begin

definition [simp, code func del]:
 $0 = \text{index-of-nat } 0$

instance ..

end

definition [simp]:
 $\text{Suc-index } k = \text{index-of-nat } (\text{Suc } (\text{nat-of-index } k))$

lemma *index-induct*: $P\ 0 \implies (\bigwedge k. P\ k \implies P\ (\text{Suc-index } k)) \implies P\ k$
proof –

assume $P\ 0$ **then have** *init*: $P\ (\text{index-of-nat } 0)$ **by** *simp*
 assume $\bigwedge k. P\ k \implies P\ (\text{Suc-index } k)$
 then have $\bigwedge n. P\ (\text{index-of-nat } n) \implies P\ (\text{Suc-index } (\text{index-of-nat } (n)))$.
 then have *step*: $\bigwedge n. P\ (\text{index-of-nat } n) \implies P\ (\text{index-of-nat } (\text{Suc } n))$ **by** *simp*
 from *init step* **have** $P\ (\text{index-of-nat } (\text{nat-of-index } k))$
 by (*induct nat-of-index k simp-all*)
 then show $P\ k$ **by** *simp*

qed

lemma *Suc-not-Zero-index*: $\text{Suc-index } k \neq 0$
 by *simp*

lemma *Zero-not-Suc-index*: $0 \neq \text{Suc-index } k$
 by *simp*

lemma *Suc-Suc-index-eq*: $\text{Suc-index } k = \text{Suc-index } l \longleftrightarrow k = l$
 by *simp*

rep-datatype *index*
 distinct *Suc-not-Zero-index Zero-not-Suc-index*
 inject *Suc-Suc-index-eq*
 induction *index-induct*

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


```

declare index-case [case-names nat, cases type: index]
declare index-induct [case-names nat, induct type: index]

lemma [code func]:
  index-size = nat-of-index
proof (rule ext)
  fix k
  have index-size k = nat-size (nat-of-index k)
    by (induct k rule: index.induct) (simp-all del: zero-index-def Suc-index-def, simp-all)
  also have nat-size (nat-of-index k) = nat-of-index k by (induct nat-of-index k) simp-all
  finally show index-size k = nat-of-index k .
qed

lemma [code func]:
  size = nat-of-index
proof (rule ext)
  fix k
  show size k = nat-of-index k
    by (induct k) (simp-all del: zero-index-def Suc-index-def, simp-all)
qed

lemma [code func]:
  k = l  $\longleftrightarrow$  nat-of-index k = nat-of-index l
  by (cases k, cases l) simp

```

13.2 Indices as datatype of ints

```

instantiation index :: number
begin

```

```

definition
  number-of = index-of-nat o nat

```

```

instance ..

```

```

end

```

```

lemma nat-of-index-number [simp]:
  nat-of-index (number-of k) = number-of k
  by (simp add: number-of-index-def nat-number-of-def number-of-is-id)

```

```

code-datatype number-of :: int  $\Rightarrow$  index

```

13.3 Basic arithmetic

```

instantiation index :: {minus, ordered-semidom, Divides.div, linorder}
begin

```

lemma *zero-index-code* [*code inline, code func*]:
 $(0::index) = \text{Numeral0}$
by (*simp add: number-of-index-def Pls-def*)
lemma [*code post*]: $\text{Numeral0} = (0::index)$
using *zero-index-code ..*

definition [*simp, code func del*]:
 $(1::index) = \text{index-of-nat } 1$

lemma *one-index-code* [*code inline, code func*]:
 $(1::index) = \text{Numeral1}$
by (*simp add: number-of-index-def Pls-def Bit1-def*)
lemma [*code post*]: $\text{Numeral1} = (1::index)$
using *one-index-code ..*

definition [*simp, code func del*]:
 $n + m = \text{index-of-nat } (\text{nat-of-index } n + \text{nat-of-index } m)$

lemma *plus-index-code* [*code func*]:
 $\text{index-of-nat } n + \text{index-of-nat } m = \text{index-of-nat } (n + m)$
by *simp*

definition [*simp, code func del*]:
 $n - m = \text{index-of-nat } (\text{nat-of-index } n - \text{nat-of-index } m)$

definition [*simp, code func del*]:
 $n * m = \text{index-of-nat } (\text{nat-of-index } n * \text{nat-of-index } m)$

lemma *times-index-code* [*code func*]:
 $\text{index-of-nat } n * \text{index-of-nat } m = \text{index-of-nat } (n * m)$
by *simp*

definition [*simp, code func del*]:
 $n \text{ div } m = \text{index-of-nat } (\text{nat-of-index } n \text{ div } \text{nat-of-index } m)$

definition [*simp, code func del*]:
 $n \text{ mod } m = \text{index-of-nat } (\text{nat-of-index } n \text{ mod } \text{nat-of-index } m)$

lemma *div-index-code* [*code func*]:
 $\text{index-of-nat } n \text{ div } \text{index-of-nat } m = \text{index-of-nat } (n \text{ div } m)$
by *simp*

lemma *mod-index-code* [*code func*]:
 $\text{index-of-nat } n \text{ mod } \text{index-of-nat } m = \text{index-of-nat } (n \text{ mod } m)$
by *simp*

definition [*simp, code func del*]:
 $n \leq m \longleftrightarrow \text{nat-of-index } n \leq \text{nat-of-index } m$

```

definition [simp, code func del]:
   $n < m \iff \text{nat-of-index } n < \text{nat-of-index } m$ 

lemma less-eq-index-code [code func]:
   $\text{index-of-nat } n \leq \text{index-of-nat } m \iff n \leq m$ 
  by simp

lemma less-index-code [code func]:
   $\text{index-of-nat } n < \text{index-of-nat } m \iff n < m$ 
  by simp

instance by default (auto simp add: left-distrib index)

end

lemma Suc-index-minus-one:  $\text{Suc-index } n - 1 = n$  by simp

lemma index-of-nat-code [code]:
   $\text{index-of-nat} = \text{of-nat}$ 
proof
  fix  $n :: \text{nat}$ 
  have  $\text{of-nat } n = \text{index-of-nat } n$ 
  by (induct  $n$ ) simp-all
  then show  $\text{index-of-nat } n = \text{of-nat } n$ 
  by (rule sym)
qed

lemma index-not-eq-zero:  $i \neq \text{index-of-nat } 0 \iff i \geq 1$ 
  by (cases  $i$ ) auto

definition
   $\text{nat-of-index-aux} :: \text{index} \Rightarrow \text{nat} \Rightarrow \text{nat}$ 
where
   $\text{nat-of-index-aux } i \ n = \text{nat-of-index } i + n$ 

lemma nat-of-index-aux-code [code]:
   $\text{nat-of-index-aux } i \ n = (\text{if } i = 0 \text{ then } n \text{ else } \text{nat-of-index-aux } (i - 1) (\text{Suc } n))$ 
  by (auto simp add: nat-of-index-aux-def index-not-eq-zero)

lemma nat-of-index-code [code]:
   $\text{nat-of-index } i = \text{nat-of-index-aux } i \ 0$ 
  by (simp add: nat-of-index-aux-def)

```

13.4 ML interface

```

ML <<
structure Index =
struct

```

```

fun mk k = HLogic.mk-number @{typ index} k;

end;
>>

```

13.5 Specialized $op -$, $op div$ and $op mod$ operations

definition

```
minus-index-aux :: index  $\Rightarrow$  index  $\Rightarrow$  index
```

where

```
[code func del]: minus-index-aux = op -
```

lemma [code func]: $op - = minus-index-aux$
using minus-index-aux-def ..

definition

```
div-mod-index :: index  $\Rightarrow$  index  $\Rightarrow$  index  $\times$  index
```

where

```
[code func del]: div-mod-index n m = (n div m, n mod m)
```

lemma [code func]:

```
div-mod-index n m = (if m = 0 then (0, n) else (n div m, n mod m))
unfolding div-mod-index-def by auto
```

lemma [code func]:

```
n div m = fst (div-mod-index n m)
unfolding div-mod-index-def by simp
```

lemma [code func]:

```
n mod m = snd (div-mod-index n m)
unfolding div-mod-index-def by simp
```

13.6 Code serialization

Implementation of indices by bounded integers

code-type index

```
(SML int)
```

```
(OCaml int)
```

```
(Haskell Int)
```

code-instance index :: eq

```
(Haskell -)
```

setup <<

```
fold (Numeral.add-code @{const-name number-index-inst.number-of-index}
false false) [SML, OCaml, Haskell]
```

>>

```

code-reserved SML Int int
code-reserved OCaml Pervasives int

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

code-const minus-index-aux :: index ⇒ index ⇒ index
  (SML Int.max / (- / - / -, / 0 : int))
  (OCaml Pervasives.max / (- / - / -) / (0 : int) ))
  (Haskell max / (- / - / -) / (0 :: Int))

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

code-const div-mod-index
  (SML (fn n => fn m => / (n div m, n mod m)))
  (OCaml (fun n -> fun m -> / (n ' / m, n mod m)))
  (Haskell divMod)

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

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

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

end

```

14 Implementation of natural numbers by target-language integers

```

theory Efficient-Nat
imports Code-Integer Code-Index
begin

```

When generating code for functions on natural numbers, the canonical rep-

resentation 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 target-language integers. To do this, just include this theory.

14.1 Basic arithmetic

Most standard arithmetic functions on natural numbers are implemented using their counterparts on the integers:

code-datatype *number-nat-inst.number-of-nat*

lemma *zero-nat-code* [*code*, *code unfold*]:

$0 = (\text{Numeral } 0 :: \text{nat})$

by *simp*

lemmas [*code post*] = *zero-nat-code* [*symmetric*]

lemma *one-nat-code* [*code*, *code unfold*]:

$1 = (\text{Numeral } 1 :: \text{nat})$

by *simp*

lemmas [*code post*] = *one-nat-code* [*symmetric*]

lemma *Suc-code* [*code*]:

$Suc\ n = n + 1$

by *simp*

lemma *plus-nat-code* [*code*]:

$n + m = \text{nat } (\text{of-nat } n + \text{of-nat } m)$

by *simp*

lemma *minus-nat-code* [*code*]:

$n - m = \text{nat } (\text{of-nat } n - \text{of-nat } m)$

by *simp*

lemma *times-nat-code* [*code*]:

$n * m = \text{nat } (\text{of-nat } n * \text{of-nat } m)$

unfolding *of-nat-mult* [*symmetric*] **by** *simp*

Specialized *op div* and *op mod* operations.

definition

divmod-aux :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \times \text{nat}$

where

[*code func del*]: *divmod-aux* = *divmod*

lemma [*code func*]:

$\text{divmod } n\ m = (\text{if } m = 0 \text{ then } (0, n) \text{ else } \text{divmod-aux } n\ m)$

unfolding *divmod-aux-def* *divmod-div-mod* **by** *simp*

lemma *divmod-aux-code* [*code*]:

$\text{divmod-aux } n \ m = (\text{nat } (\text{of-nat } n \ \text{div } \text{of-nat } m), \text{ nat } (\text{of-nat } n \ \text{mod } \text{of-nat } m))$
unfolding $\text{divmod-aux-def divmod-div-mod zdiv-int [symmetric] zmod-int [symmetric]}$
by *simp*

lemma *eq-nat-code* [*code*]:
 $n = m \iff (\text{of-nat } n :: \text{int}) = \text{of-nat } m$
by *simp*

lemma *less-eq-nat-code* [*code*]:
 $n \leq m \iff (\text{of-nat } n :: \text{int}) \leq \text{of-nat } m$
by *simp*

lemma *less-nat-code* [*code*]:
 $n < m \iff (\text{of-nat } n :: \text{int}) < \text{of-nat } m$
by *simp*

14.2 Case analysis

Case analysis on natural numbers is rephrased using a conditional expression:

lemma [*code func, code unfold*]:
 $\text{nat-case} = (\lambda f \ g \ n. \text{if } n = 0 \text{ then } f \text{ else } g \ (n - 1))$
by (*auto simp add: expand-fun-eq dest!: gr0-implies-Suc*)

14.3 Preprocessors

In contrast to $\text{Suc } n$, 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:

lemma *Suc-if-eq*: $(\bigwedge n. f \ (\text{Suc } n) = h \ n) \implies f \ 0 = g \implies$
 $f \ n = (\text{if } n = 0 \text{ then } g \text{ else } h \ (n - 1))$
by (*case-tac n simp-all*)

lemma *Suc-clause*: $(\bigwedge n. P \ n \ (\text{Suc } n)) \implies n \neq 0 \implies P \ (n - 1) \ n$
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 Target language setup

For ML, we map *nat* to target language integers, where we assert that values are always non-negative.

```

code-type nat
  (SML int)
  (OCaml Big'-int.big'-int)

```

```

types-code
  nat (int)
attach (term-of) ⟨⟨
  val term-of-nat = HLogic.mk-number HLogic.natT;
  ⟩⟩
attach (test) ⟨⟨
  fun gen-nat i =
    let val n = random-range 0 i
    in (n, fn () => term-of-nat n) end;
  ⟩⟩

```

For Haskell we define our own *nat* type. The reason is that we have to distinguish type class instances for *nat* and *int*.

```

code-include Haskell Nat ⟨⟨
  newtype Nat = Nat Integer deriving (Show, Eq);

```

```

  instance Num Nat where {
    fromInteger k = Nat (if k >= 0 then k else 0);
    Nat n + Nat m = Nat (n + m);
    Nat n - Nat m = fromInteger (n - m);
    Nat n * Nat m = Nat (n * m);
    abs n = n;
    signum - = 1;
    negate n = error negate Nat;
  };

```

```

  instance Ord Nat where {
    Nat n <= Nat m = n <= m;
    Nat n < Nat m = n < m;
  };

```

```

  instance Real Nat where {
    toRational (Nat n) = toRational n;
  };

```

```

  instance Enum Nat where {
    toEnum k = fromInteger (toEnum k);
    fromEnum (Nat n) = fromEnum n;
  };

```

```

  instance Integral Nat where {
    toInteger (Nat n) = n;
    divMod n m = quotRem n m;
    quotRem (Nat n) (Nat m) = (Nat k, Nat l) where (k, l) = quotRem n m;
  };

```


»

code-reserved *Haskell Nat*

code-type *nat*
(*Haskell Nat*)

code-instance *nat :: eq*
(*Haskell -*)

Natural numerals.

lemma [*code inline, symmetric, code post*]:
 nat (number-of i) = number-nat-inst.number-of-nat i
 — this interacts as desired with *number-of ?v = nat (number-of ?v)*
 by (*simp add: number-nat-inst.number-of-nat*)

setup «
 fold (Numeral.add-code @{const-name number-nat-inst.number-of-nat}
 true false) [SML, OCaml, Haskell]
»

Since natural numbers are implemented using integers in ML, the coercion function *of-nat* of type *nat* \Rightarrow *int* is simply implemented by the identity function. 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.

definition

int :: nat \Rightarrow *int*

where

[*code func del*]: *int = of-nat*

lemma *int-code'* [*code func*]:
 int (number-of l) = (if neg (number-of l :: int) then 0 else number-of l)
 unfolding *int-nat-number-of [folded int-def]* ..

lemma *nat-code'* [*code func*]:
 nat (number-of l) = (if neg (number-of l :: int) then 0 else number-of l)
 by *auto*

lemma *of-nat-int* [*code unfold*]:
 of-nat = int **by** (*simp add: int-def*)
declare *of-nat-int* [*symmetric, code post*]

code-const *int*
(*SML -*)
(*OCaml -*)

consts-code
int ((-))

```

    nat (<module>nat)
attach <<
  fun nat i = if i < 0 then 0 else i;
  >>

code-const nat
  (SML IntInf.max / (/0,/ -))
  (OCaml Big'-int.max'-big'-int / Big'-int.zero'-big'-int)

```

For Haskell, things are slightly different again.

```

code-const int and nat
  (Haskell toInteger and fromInteger)

```

Conversion from and to indices.

```

code-const index-of-nat
  (SML IntInf.toInt)
  (OCaml Big'-int.int'-of'-big'-int)
  (Haskell toEnum)

```

```

code-const nat-of-index
  (SML IntInf.fromInt)
  (OCaml Big'-int.big'-int'-of'-int)
  (Haskell fromEnum)

```

Using target language arithmetic operations whenever appropriate

```

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

```

```

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

```

```

code-const divmod-aux
  (SML IntInf.divMod / ((-), / (-)))
  (OCaml Big'-int.quomod'-big'-int)
  (Haskell divMod)

```

```

code-const op = :: nat ⇒ nat ⇒ bool
  (SML !((- : IntInf.int) = -))
  (OCaml Big'-int.eq'-big'-int)
  (Haskell infixl 4 ==)

```

```

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

```

```

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

consts-code
  0 (0)
  Suc ((- +/ 1))
  op + :: nat ⇒ nat ⇒ nat ((- +/ -))
  op * :: nat ⇒ nat ⇒ nat ((- */ -))
  op ≤ :: nat ⇒ nat ⇒ bool ((- <=/ -))
  op < :: nat ⇒ nat ⇒ bool ((- </ -))

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 int

end

```

15 Quatifier elimination for R(0,1,+,i)

```

theory ReflectedFerrack
  imports GCD Real Efficient-Nat
  uses (linreif.ML) (linrtac.ML)
begin

consts alluopairs:: 'a list ⇒ ('a × 'a) list
```

primrec

```

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

lemma nth-pos2: 0 < n ⇒ (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 ⇒ 'a list

recdef remdps measure size
  remdps [] = []
  remdps (x#xs) = (x#(remdps (List.filter (λ y. y ≠ 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  $\Rightarrow$  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  $\Rightarrow$  num  $\Rightarrow$  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  $\Rightarrow$  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*: $\llbracket \text{Inum } bs \ s = s' ; \text{Inum } bs \ t = t' \rrbracket \Longrightarrow \text{Ifm } bs \ (\text{Le } (\text{Sub } s \ t))$
= ($s' \leq t'$)

apply *simp*

done

lemma *IfmLtSub*: $\llbracket \text{Inum } bs \ s = s' ; \text{Inum } bs \ t = t' \rrbracket \Longrightarrow \text{Ifm } bs \ (\text{Lt } (\text{Sub } s \ t))$
= ($s' < t'$)

apply *simp*

done

lemma *IfmEqSub*: $\llbracket \text{Inum } bs \ s = s' ; \text{Inum } bs \ t = t' \rrbracket \Longrightarrow \text{Ifm } bs \ (\text{Eq } (\text{Sub } s \ t))$
= ($s' = t'$)

apply *simp*

done

lemma *IfmNOT*: (*Ifm* *bs* *p* = *P*) \Longrightarrow (*Ifm* *bs* (*NOT* *p*) = (\neg *P*))

apply *simp*

done

lemma *IfmAnd*: $\llbracket \text{Ifm } bs \ p = P ; \text{Ifm } bs \ q = Q \rrbracket \Longrightarrow (\text{Ifm } bs \ (\text{And } p \ q) = (P \wedge Q))$

apply *simp*

done

lemma *IfmOr*: $\llbracket \text{Ifm } bs \ p = P ; \text{Ifm } bs \ q = Q \rrbracket \Longrightarrow (\text{Ifm } bs \ (\text{Or } p \ q) = (P \vee Q))$

apply *simp*

done

lemma *IfmImp*: $\llbracket \text{Ifm } bs \ p = P ; \text{Ifm } bs \ q = Q \rrbracket \Longrightarrow (\text{Ifm } bs \ (\text{Imp } p \ q) = (P \longrightarrow Q))$

apply *simp*

done

lemma *IfmIff*: $\llbracket \text{Ifm } bs \ p = P ; \text{Ifm } bs \ q = Q \rrbracket \Longrightarrow (\text{Ifm } bs \ (\text{Iff } p \ q) = (P = Q))$

apply *simp*

done

```

lemma IfmE:  $(!! x. \text{Ifm } (x\#bs) p = P x) \implies (\text{Ifm } bs (E p) = (\exists x. P x))$ 
apply simp
done
lemma IfmA:  $(!! x. \text{Ifm } (x\#bs) p = P x) \implies (\text{Ifm } bs (A p) = (\forall x. P x))$ 
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]:  $\text{Ifm } bs (\text{not } p) = \text{Ifm } bs (\text{NOT } p)$ 
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]:  $\text{Ifm } bs (\text{conj } p \ q) = \text{Ifm } bs (\text{And } p \ q)$ 
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]:  $\text{Ifm } bs (\text{disj } p \ q) = \text{Ifm } bs (\text{Or } p \ q)$ 
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]:  $\text{Ifm } bs (\text{imp } p \ q) = \text{Ifm } bs (\text{Imp } p \ q)$ 
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]:  $\text{Ifm } bs (\text{iff } p \ q) = \text{Ifm } bs (\text{Iff } p \ q)$ 
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 ≠ T ⇒ P ≠ F ⇒ Q ≠ T ⇒ Q ≠ F ⇒ P ≠ Q ⇒ 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 ≠ T ⇒ P ≠ F ⇒ Q ≠ T ⇒ Q ≠ F ⇒ P ≠ Q ⇒ 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 ≠ T ⇒ P ≠ F ⇒ P ≠ Q ⇒ Q ≠ T ⇒ Q ≠ F ⇒ imp P Q = Imp P
Q

```

by (simp-all add: imp-def)

lemma *trivNOT*: $p \neq \text{NOT } p \text{ NOT } p \neq p$

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 ≠ NOT T ⇒ iff T p = p
p ≠ NOT T ⇒ iff p T = p
p ≠ q ⇒ p ≠ NOT q ⇒ q ≠ NOT p ⇒ p ≠ F ⇒ q ≠ F ⇒ p ≠ T ⇒ q ≠
T ⇒ iff p q = Iff p q

```

using *trivNOT*

by (simp-all add: iff-def, cases p, auto)

consts *qfree*:: $fm \Rightarrow bool$

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  $\Rightarrow$  bool
  bound0:: fm  $\Rightarrow$  bool
primrec
  numbound0 (C c) = True
  numbound0 (Bound n) = (n>0)
  numbound0 (CN n c a) = (n $\neq$ 0  $\wedge$  numbound0 a)
  numbound0 (Neg a) = numbound0 a
  numbound0 (Add a b) = (numbound0 a  $\wedge$  numbound0 b)
  numbound0 (Sub a b) = (numbound0 a  $\wedge$  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  $\wedge$  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)

lemma not-qf[simp]: qfree p  $\Longrightarrow$  qfree (not p)
by (cases p, auto)
lemma not-bn[simp]: bound0 p  $\Longrightarrow$  bound0 (not p)
by (cases p, auto)

```

```

lemma conj-qf[simp]:  $\llbracket qfree\ p\ ;\ qfree\ q \rrbracket \implies qfree\ (conj\ p\ q)$ 
using conj-def by auto
lemma conj-nb[simp]:  $\llbracket bound0\ p\ ;\ bound0\ q \rrbracket \implies bound0\ (conj\ p\ q)$ 
using conj-def by auto

lemma disj-qf[simp]:  $\llbracket qfree\ p\ ;\ qfree\ q \rrbracket \implies qfree\ (disj\ p\ q)$ 
using disj-def by auto
lemma disj-nb[simp]:  $\llbracket bound0\ p\ ;\ bound0\ q \rrbracket \implies bound0\ (disj\ p\ q)$ 
using disj-def by auto

lemma imp-qf[simp]:  $\llbracket qfree\ p\ ;\ qfree\ q \rrbracket \implies qfree\ (imp\ p\ q)$ 
using imp-def by (cases p=F  $\vee$  q=T, simp-all add: imp-def)
lemma imp-nb[simp]:  $\llbracket bound0\ p\ ;\ bound0\ q \rrbracket \implies bound0\ (imp\ p\ q)$ 
using imp-def by (cases p=F  $\vee$  q=T  $\vee$  p=q, simp-all add: imp-def)

lemma iff-qf[simp]:  $\llbracket qfree\ p\ ;\ qfree\ q \rrbracket \implies qfree\ (iff\ p\ q)$ 
by (unfold iff-def, cases p=q, auto)
lemma iff-nb[simp]:  $\llbracket bound0\ p\ ;\ bound0\ q \rrbracket \implies bound0\ (iff\ p\ q)$ 
using iff-def by (unfold iff-def, cases p=q, auto)

consts
  decrnum:: num  $\Rightarrow$  num
  decr :: fm  $\Rightarrow$  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  $\implies$  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  $\implies$  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  $\implies$  q  $\neq$  F  $\implies$  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) = ( $\exists$  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*: $\forall x \in \text{set } xs. \text{qfree } (f \ x)$
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*
recdef *disjuncts* *measure* *size*
disjuncts (*Or* *p* *q*) = (*disjuncts* *p*) @ (*disjuncts* *q*)
disjuncts *F* = []
disjuncts *p* = [*p*]

lemma *disjuncts*: $(\exists q \in \text{set } (\text{disjuncts } p). \text{Ifm } bs \ q) = \text{Ifm } bs \ p$
by(*induct* *p* *rule*: *disjuncts.induct*, *auto*)

lemma *disjuncts-nb*: *bound0* *p* $\implies \forall q \in \text{set } (\text{disjuncts } p). \text{bound0 } q$
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* $\implies \forall q \in \text{set } (\text{disjuncts } p). \text{qfree } q$
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* \Rightarrow *fm*) \Rightarrow *fm* \Rightarrow *fm*
DJ *f* *p* $\equiv \text{evaldjf } f \ (\text{disjuncts } p)$

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

lemma *DJ-qf*: **assumes**

```

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

  from evaldjf-qf[OF th'] th show  $\text{qfree } (DJ \ f \ p)$  by simp
qed

lemma DJ-qe: assumes qe:  $\forall bs \ p. \text{qfree } p \longrightarrow \text{qfree } (qe \ p) \wedge (\text{Ifm } bs \ (qe \ p) = \text{Ifm } bs \ (E \ p))$ 
  shows  $\forall bs \ p. \text{qfree } p \longrightarrow \text{qfree } (DJ \ qe \ p) \wedge (\text{Ifm } bs \ ((DJ \ qe \ p)) = \text{Ifm } bs \ (E \ p))$ 
proof (clarify)
  fix p::fm and bs
  assume qf:  $\text{qfree } p$ 
  from qe have qth:  $\forall p. \text{qfree } p \longrightarrow \text{qfree } (qe \ p)$  by blast
  from DJ-qf[OF qth] qf have qfth:  $\text{qfree } (DJ \ qe \ p)$  by auto
  have  $\text{Ifm } bs \ (DJ \ qe \ p) = (\exists q \in \text{set } (\text{disjuncts } p). \text{Ifm } bs \ (qe \ q))$ 
    by (simp add: DJ-def evaldjf-ex)
  also have  $\dots = (\exists q \in \text{set } (\text{disjuncts } p). \text{Ifm } bs \ (E \ q))$  using qe disjuncts-qf[OF qf] by auto
  also have  $\dots = \text{Ifm } bs \ (E \ p)$  by (induct p rule: disjuncts.induct, auto)
  finally show  $\text{qfree } (DJ \ qe \ p) \wedge \text{Ifm } bs \ (DJ \ qe \ p) = \text{Ifm } bs \ (E \ p)$  using qfth by blast
qed

consts
  numgcd ::  $\text{num} \Rightarrow \text{int}$ 
  numgcdh ::  $\text{num} \Rightarrow \text{int} \Rightarrow \text{int}$ 
  reducecoeffh ::  $\text{num} \Rightarrow \text{int} \Rightarrow \text{num}$ 
  reducecoeff ::  $\text{num} \Rightarrow \text{num}$ 
  dvdnumcoeff ::  $\text{num} \Rightarrow \text{int} \Rightarrow \text{bool}$ 
consts maxcoeff ::  $\text{num} \Rightarrow \text{int}$ 
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:  $\text{maxcoeff } t \geq 0$ 
  by (induct t rule: maxcoeff.induct, auto)

recdef numgcdh measure size
  numgcdh (C i) = ( $\lambda g. \text{igcd } i \ g$ )
  numgcdh (CN n c t) = ( $\lambda g. \text{igcd } c \ (\text{numgcdh } t \ g)$ )
  numgcdh t = ( $\lambda g. 1$ )
defs numgcd-def [code func]:  $\text{numgcd } t \equiv \text{numgcdh } t \ (\text{maxcoeff } t)$ 

```

```

recdef reduceceffh measure size
  reduceceffh (C i) = (λ g. C (i div g))
  reduceceffh (CN n c t) = (λ g. CN n (c div g) (reduceceffh t g))
  reduceceffh t = (λ g. t)

defs reduceceff-def: reduceceff t ≡
  (let g = numgcd t in
   if g = 0 then C 0 else if g=1 then t else reduceceffh t g)

recdef dvdnumcoeff measure size
  dvdnumcoeff (C i) = (λ g. g dvd i)
  dvdnumcoeff (CN n c t) = (λ g. g dvd c ∧ (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 ≥ 0 shows numgcdh t g ≥ 0
  using gp
  by (induct t rule: numgcdh.induct, auto simp add: igcd-def)

lemma numgcd-pos: numgcd t ≥ 0
  by (simp add: numgcd-def numgcdh-pos maxcoeff-pos)

lemma reduceceffh:
  assumes gt: dvdnumcoeff t g and gp: g > 0
  shows real g *(Inum bs (reduceceffh t g)) = Inum bs t
  using gt
proof(induct t rule: reduceceffh.induct)
  case (1 i) hence gd: g dvd i by simp
  from gp have gnz: g ≠ 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 ≠ 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 t = (λ x. True)

lemma ismaxcoeff-mono: ismaxcoeff t c ⇒ c ≤ c' ⇒ 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)
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 (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 ⇒ 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
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]:  $\llbracket \text{numbound0 } t ; \text{numbound0 } s \rrbracket \implies \text{numbound0 } (\text{numadd } (t,s))$ 
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]:  $\bigwedge i. \text{Inum bs } (\text{nummul } t \ i) = \text{Inum bs } (\text{Mul } i \ t)$ 
by (induct t rule: nummul.induct, auto simp add: ring-simps)

lemma nummul-nb[simp]:  $\bigwedge i. \text{numbound0 } t \implies \text{numbound0 } (\text{nummul } t \ i)$ 
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 by simp

lemma numneg-nb[simp]: numbound0 t  $\implies$  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]:  $\llbracket \text{numbound0 } t ; \text{numbound0 } s \rrbracket \implies \text{numbound0 } (\text{numsub } t \ s)$ 
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]*:
 $numbound0\ t \implies numbound0\ (simpnum\ t)$
by (*induct t rule: simpnum.induct, auto*)

consts *nozerocoeff*:: $num \Rightarrow bool$
recdef *nozerocoeff* *measure size*
 $nozerocoeff\ (C\ c) = True$
 $nozerocoeff\ (CN\ n\ c\ t) = (c \neq 0 \wedge nozerocoeff\ t)$
 $nozerocoeff\ t = True$

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

lemma *nummul-nz* : $\bigwedge i. i \neq 0 \implies nozerocoeff\ a \implies nozerocoeff\ (nummul\ a\ i)$
by (*induct a rule: nummul.induct, auto simp add: Let-def numadd-nz*)

lemma *numneg-nz* : $nozerocoeff\ a \implies nozerocoeff\ (numneg\ a)$
by (*simp add: numneg-def nummul-nz*)

lemma *numsub-nz*: $nozerocoeff\ a \implies nozerocoeff\ b \implies nozerocoeff\ (numsub\ a\ b)$
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*: $nozerocoeff\ t \implies maxcoeff\ t = 0 \implies t = C\ 0$
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 (reduceceffh 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
 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 = reduceceffh ?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 reduceceffh[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)

```

$\text{simpfm } (Lt \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v < 0) \text{ then } T \text{ else } F$
 $\mid - \Rightarrow Lt \ a')$
 $\text{simpfm } (Le \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v \leq 0) \text{ then } T \text{ else } F \mid - \Rightarrow Le \ a')$
 $\text{simpfm } (Gt \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v > 0) \text{ then } T \text{ else } F \mid - \Rightarrow Gt \ a')$
 $\text{simpfm } (Ge \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v \geq 0) \text{ then } T \text{ else } F \mid - \Rightarrow Ge \ a')$
 $\text{simpfm } (Eq \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v = 0) \text{ then } T \text{ else } F \mid - \Rightarrow Eq \ a')$
 $\text{simpfm } (NEq \ a) = (\text{let } a' = \text{simpnum } a \text{ in case } a' \text{ of } C \ v \Rightarrow \text{if } (v \neq 0) \text{ then } T \text{ else } F \mid - \Rightarrow NEq \ a')$
 $\text{simpfm } p = p$
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*
 $\{\text{fix } v \text{ assume } ?sa = C \ v \text{ hence } ?case \text{ using } sa \text{ by } simp \}$
moreover $\{\text{assume } \neg (\exists \ v. \ ?sa = C \ v) \text{ hence } ?case \text{ using } sa$
 $\text{by } (cases \ ?sa, \ simp\text{-all} \ \text{add: } Let\text{-def})\}$
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*
 $\{\text{fix } v \text{ assume } ?sa = C \ v \text{ hence } ?case \text{ using } sa \text{ by } simp \}$
moreover $\{\text{assume } \neg (\exists \ v. \ ?sa = C \ v) \text{ hence } ?case \text{ using } sa$
 $\text{by } (cases \ ?sa, \ simp\text{-all} \ \text{add: } Let\text{-def})\}$
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*
 $\{\text{fix } v \text{ assume } ?sa = C \ v \text{ hence } ?case \text{ using } sa \text{ by } simp \}$
moreover $\{\text{assume } \neg (\exists \ v. \ ?sa = C \ v) \text{ hence } ?case \text{ using } sa$
 $\text{by } (cases \ ?sa, \ simp\text{-all} \ \text{add: } Let\text{-def})\}$
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*
 $\{\text{fix } v \text{ assume } ?sa = C \ v \text{ hence } ?case \text{ using } sa \text{ by } simp \}$
moreover $\{\text{assume } \neg (\exists \ v. \ ?sa = C \ v) \text{ hence } ?case \text{ using } sa$
 $\text{by } (cases \ ?sa, \ simp\text{-all} \ \text{add: } Let\text{-def})\}$
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*
 $\{\text{fix } v \text{ assume } ?sa = C \ v \text{ hence } ?case \text{ using } sa \text{ by } simp \}$
moreover $\{\text{assume } \neg (\exists \ v. \ ?sa = C \ v) \text{ hence } ?case \text{ using } sa$
 $\text{by } (cases \ ?sa, \ simp\text{-all} \ \text{add: } Let\text{-def})\}$

```

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  $\implies$  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  $\implies$  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  $\Rightarrow$  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:  $\bigwedge bs. \text{Ifm } bs \text{ (prep } p) = \text{Ifm } bs \text{ } p$ 
by (induct p rule: prep.induct, auto)

```

```

consts qelim :: fm  $\Rightarrow$  (fm  $\Rightarrow$  fm)  $\Rightarrow$  fm
recdef qelim measure fmsize
    qelim (E p) = ( $\lambda$  qe. DJ qe (qelim p qe))
    qelim (A p) = ( $\lambda$  qe. not (qe ((qelim (NOT p) qe))))
    qelim (NOT p) = ( $\lambda$  qe. not (qelim p qe))
    qelim (And p q) = ( $\lambda$  qe. conj (qelim p qe) (qelim q qe))
    qelim (Or p q) = ( $\lambda$  qe. disj (qelim p qe) (qelim q qe))
    qelim (Imp p q) = ( $\lambda$  qe. imp (qelim p qe) (qelim q qe))
    qelim (Iff p q) = ( $\lambda$  qe. iff (qelim p qe) (qelim q qe))
    qelim p = ( $\lambda$  y. simpfm p)

```

```

lemma qelim-ci:
  assumes qe-inv:  $\forall bs \text{ } p. \text{qfree } p \longrightarrow \text{qfree } (qe \text{ } p) \wedge (\text{Ifm } bs \text{ } (qe \text{ } p) = \text{Ifm } bs \text{ } (E \text{ } p))$ 
  shows  $\bigwedge bs. \text{qfree } (qelim \text{ } p \text{ } qe) \wedge (\text{Ifm } bs \text{ } (qelim \text{ } p \text{ } qe) = \text{Ifm } bs \text{ } p)$ 
  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  $\Rightarrow$  fm
  minusinf :: fm  $\Rightarrow$  fm

```

recdef *minusinf measure size*

$\text{minusinf } (\text{And } p \ q) = \text{conj } (\text{minusinf } p) (\text{minusinf } q)$
 $\text{minusinf } (\text{Or } p \ q) = \text{disj } (\text{minusinf } p) (\text{minusinf } q)$
 $\text{minusinf } (\text{Eq } (\text{CN } 0 \ c \ e)) = F$
 $\text{minusinf } (\text{NEq } (\text{CN } 0 \ c \ e)) = T$
 $\text{minusinf } (\text{Lt } (\text{CN } 0 \ c \ e)) = T$
 $\text{minusinf } (\text{Le } (\text{CN } 0 \ c \ e)) = T$
 $\text{minusinf } (\text{Gt } (\text{CN } 0 \ c \ e)) = F$
 $\text{minusinf } (\text{Ge } (\text{CN } 0 \ c \ e)) = F$
 $\text{minusinf } p = p$

recdef *plusinf measure size*

$\text{plusinf } (\text{And } p \ q) = \text{conj } (\text{plusinf } p) (\text{plusinf } q)$
 $\text{plusinf } (\text{Or } p \ q) = \text{disj } (\text{plusinf } p) (\text{plusinf } q)$
 $\text{plusinf } (\text{Eq } (\text{CN } 0 \ c \ e)) = F$
 $\text{plusinf } (\text{NEq } (\text{CN } 0 \ c \ e)) = T$
 $\text{plusinf } (\text{Lt } (\text{CN } 0 \ c \ e)) = F$
 $\text{plusinf } (\text{Le } (\text{CN } 0 \ c \ e)) = F$
 $\text{plusinf } (\text{Gt } (\text{CN } 0 \ c \ e)) = T$
 $\text{plusinf } (\text{Ge } (\text{CN } 0 \ c \ e)) = T$
 $\text{plusinf } p = p$

consts

$\text{isrlfm} :: \text{fm} \Rightarrow \text{bool}$

recdef *isrlfm measure size*

$\text{isrlfm } (\text{And } p \ q) = (\text{isrlfm } p \wedge \text{isrlfm } q)$
 $\text{isrlfm } (\text{Or } p \ q) = (\text{isrlfm } p \wedge \text{isrlfm } q)$
 $\text{isrlfm } (\text{Eq } (\text{CN } 0 \ c \ e)) = (c > 0 \wedge \text{numbound0 } e)$
 $\text{isrlfm } (\text{NEq } (\text{CN } 0 \ c \ e)) = (c > 0 \wedge \text{numbound0 } e)$
 $\text{isrlfm } (\text{Lt } (\text{CN } 0 \ c \ e)) = (c > 0 \wedge \text{numbound0 } e)$
 $\text{isrlfm } (\text{Le } (\text{CN } 0 \ c \ e)) = (c > 0 \wedge \text{numbound0 } e)$
 $\text{isrlfm } (\text{Gt } (\text{CN } 0 \ c \ e)) = (c > 0 \wedge \text{numbound0 } e)$
 $\text{isrlfm } (\text{Ge } (\text{CN } 0 \ c \ e)) = (c > 0 \wedge \text{numbound0 } e)$
 $\text{isrlfm } p = (\text{isatom } p \wedge (\text{bound0 } p))$

consts $\text{rsplit0} :: \text{num} \Rightarrow \text{int} \times \text{num}$

recdef *rsplit0 measure num-size*

$\text{rsplit0 } (\text{Bound } 0) = (1, C \ 0)$
 $\text{rsplit0 } (\text{Add } a \ b) = (\text{let } (ca, ta) = \text{rsplit0 } a ; (cb, tb) = \text{rsplit0 } b$
 $\quad \text{in } (ca + cb, \text{Add } ta \ tb))$
 $\text{rsplit0 } (\text{Sub } a \ b) = \text{rsplit0 } (\text{Add } a \ (\text{Neg } b))$
 $\text{rsplit0 } (\text{Neg } a) = (\text{let } (c, t) = \text{rsplit0 } a \text{ in } (-c, \text{Neg } t))$
 $\text{rsplit0 } (\text{Mul } c \ a) = (\text{let } (ca, ta) = \text{rsplit0 } a \text{ in } (c * ca, \text{Mul } c \ ta))$
 $\text{rsplit0 } (\text{CN } 0 \ c \ a) = (\text{let } (ca, ta) = \text{rsplit0 } a \text{ in } (c + ca, ta))$
 $\text{rsplit0 } (\text{CN } n \ c \ a) = (\text{let } (ca, ta) = \text{rsplit0 } a \text{ in } (ca, \text{CN } n \ c \ ta))$
 $\text{rsplit0 } t = (0, t)$

lemma *rsplit0:*

shows $\text{Inum } bs \ ((\text{split } (\text{CN } 0)) (\text{rsplit0 } t)) = \text{Inum } bs \ t \wedge \text{numbound0 } (\text{snd}$


```

(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

$neq\ c\ t = (if\ c = 0\ then\ (NEq\ t)\ else\ if\ c > 0\ then\ (NEq\ (CN\ 0\ c\ t))$
 $\quad\quad\quad else\ (NEq\ (CN\ 0\ (-c)\ (Neg\ t))))$

lemma *lt*: $numnoabs\ t \implies Ifm\ bs\ (split\ lt\ (rsplit0\ t)) = Ifm\ bs\ (Lt\ t) \wedge isrlfm\ (split\ lt\ (rsplit0\ t))$

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*: $numnoabs\ t \implies Ifm\ bs\ (split\ le\ (rsplit0\ t)) = Ifm\ bs\ (Le\ t) \wedge isrlfm\ (split\ le\ (rsplit0\ t))$

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*: $numnoabs\ t \implies Ifm\ bs\ (split\ gt\ (rsplit0\ t)) = Ifm\ bs\ (Gt\ t) \wedge isrlfm\ (split\ gt\ (rsplit0\ t))$

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*: $numnoabs\ t \implies Ifm\ bs\ (split\ ge\ (rsplit0\ t)) = Ifm\ bs\ (Ge\ t) \wedge isrlfm\ (split\ ge\ (rsplit0\ t))$

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*: $numnoabs\ t \implies Ifm\ bs\ (split\ eq\ (rsplit0\ t)) = Ifm\ bs\ (Eq\ t) \wedge isrlfm\ (split\ eq\ (rsplit0\ t))$

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 *neg*: $numnoabs\ t \implies Ifm\ bs\ (split\ neg\ (rsplit0\ t)) = Ifm\ bs\ (NEq\ t) \wedge isrlfm\ (split\ neg\ (rsplit0\ t))$

using *rsplit0*[**where** $bs = bs$ **and** $t=t$]

by (*auto simp add: neg-def split-def*) (*cases snd(rsplit0 t), auto, case-tac nat, auto*)

lemma *conj-lin*: $isrlfm\ p \implies isrlfm\ q \implies isrlfm\ (conj\ p\ q)$

by (*auto simp add: conj-def*)

lemma *disj-lin*: $isrlfm\ p \implies isrlfm\ q \implies isrlfm\ (disj\ p\ q)$

by (*auto simp add: disj-def*)

consts *rlfm* :: $fm \Rightarrow fm$

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
  fix a
  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  $numbound0-I[OF \ nb, \text{where } b=x \text{ and } bs=bs \text{ and } b'=a]$  by simp }
    hence  $\forall \ x < ?z. \ ?P \ ?z \ x \ (Eq \ (CN \ 0 \ c \ e))$  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
  fix  $a$ 
  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 $[OF \ cp, \text{where } a=x \text{ and } b=- \ ?e]$  mult-ac)
    hence  $real \ c * x + ?e < 0$  by arith
    hence  $real \ c * x + ?e \neq 0$  by simp
    with  $xz$  have  $?P \ ?z \ x \ (NEq \ (CN \ 0 \ c \ e))$ 
      using  $numbound0-I[OF \ nb, \text{where } b=x \text{ and } bs=bs \text{ and } b'=a]$  by simp }
    hence  $\forall \ x < ?z. \ ?P \ ?z \ x \ (NEq \ (CN \ 0 \ c \ e))$  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
  fix  $a$ 
  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 $[OF \ cp, \text{where } a=x \text{ and } b=- \ ?e]$  mult-ac)
    hence  $real \ c * x + ?e < 0$  by arith
    with  $xz$  have  $?P \ ?z \ x \ (Lt \ (CN \ 0 \ c \ e))$ 
      using  $numbound0-I[OF \ nb, \text{where } b=x \text{ and } bs=bs \text{ and } b'=a]$  by simp }
    hence  $\forall \ x < ?z. \ ?P \ ?z \ x \ (Lt \ (CN \ 0 \ c \ e))$  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
  fix  $a$ 
  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 $[OF \ cp, \text{where } a=x \text{ and } b=- \ ?e]$  mult-ac)
    hence  $real \ c * x + ?e < 0$  by arith

```

```

    with  $xz$  have  $?P \ ?z \ x \ (Le \ (CN \ 0 \ c \ e))$ 
      using  $numbound0-I[OF \ nb, \text{where } b=x \text{ and } bs=bs \text{ and } b'=a]$  by  $simp$  }
    hence  $\forall \ x < ?z. \ ?P \ ?z \ x \ (Le \ (CN \ 0 \ c \ e))$  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  $simp$ 
    fix  $a$ 
    let  $?e = Inum \ (a \# bs) \ e$ 
    let  $?z = (- \ ?e) / real \ c$ 
    {fix  $x$ 
      assume  $xz: x < ?z$ 
      hence  $(real \ c * x < - \ ?e)$ 
        by  $(simp \ only: pos-less-divide-eq[OF \ cp, \text{where } a=x \text{ and } b=- \ ?e] \ mult-ac)$ 
      hence  $real \ c * x + ?e < 0$  by  $arith$ 
      with  $xz$  have  $?P \ ?z \ x \ (Gt \ (CN \ 0 \ c \ e))$ 
        using  $numbound0-I[OF \ nb, \text{where } b=x \text{ and } bs=bs \text{ and } b'=a]$  by  $simp$  }
      hence  $\forall \ x < ?z. \ ?P \ ?z \ x \ (Gt \ (CN \ 0 \ c \ e))$  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  $simp$ 
    fix  $a$ 
    let  $?e = Inum \ (a \# bs) \ e$ 
    let  $?z = (- \ ?e) / real \ c$ 
    {fix  $x$ 
      assume  $xz: x < ?z$ 
      hence  $(real \ c * x < - \ ?e)$ 
        by  $(simp \ only: pos-less-divide-eq[OF \ cp, \text{where } a=x \text{ and } b=- \ ?e] \ mult-ac)$ 
      hence  $real \ c * x + ?e < 0$  by  $arith$ 
      with  $xz$  have  $?P \ ?z \ x \ (Ge \ (CN \ 0 \ c \ e))$ 
        using  $numbound0-I[OF \ nb, \text{where } b=x \text{ and } bs=bs \text{ and } b'=a]$  by  $simp$  }
      hence  $\forall \ x < ?z. \ ?P \ ?z \ x \ (Ge \ (CN \ 0 \ c \ e))$  by  $simp$ 
      thus  $?case$  by  $blast$ 
    }
qed  $simp-all$ 

lemma  $rplusinf-inf$ :
  assumes  $lp: isrlfm \ p$ 
  shows  $\exists \ z. \ \forall \ x > z. \ Ifm \ (x \# bs) \ (plusinf \ p) = Ifm \ (x \# bs) \ p$  (is  $\exists \ z. \ \forall \ x. \ ?P \ z \ x \ p$ )
  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
fix a
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
fix a
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
fix a
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
fix a
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  $\forall x > ?z. ?P ?z x (Le (CN 0 c e))$  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
fix a
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  $\forall x > ?z. ?P ?z x (Gt (CN 0 c e))$  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
fix a
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  $\exists x. \text{Ifm } (x\#bs) \text{ } p$ 
proof –
  from bound0-I [OF rminusinf-bound0[OF lp], where b=a and bs=bs] ex
  have th:  $\forall x. \text{Ifm } (x\#bs) \text{ } (minusinf p)$  by auto
  from rminusinf-inf[OF lp, where bs=bs]
  obtain z where z-def:  $\forall x < z. \text{Ifm } (x \# bs) \text{ } (minusinf p) = \text{Ifm } (x \# bs) \text{ } p$  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  $\exists x. \text{Ifm } (x\#bs) \text{ } p$ 
proof –
  from bound0-I [OF rplusinf-bound0[OF lp], where b=a and bs=bs] ex
  have th:  $\forall x. \text{Ifm } (x\#bs) \text{ } (plusinf p)$  by auto
  from rplusinf-inf[OF lp, where bs=bs]
  obtain z where z-def:  $\forall x > z. \text{Ifm } (x \# bs) \text{ } (plusinf p) = \text{Ifm } (x \# bs) \text{ } p$  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  $\Rightarrow$  (num  $\times$  int) list
  usubst:: fm  $\Rightarrow$  (num  $\times$  int)  $\Rightarrow$  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*: $\neg (Ifm (a\#bs) (minusinf p))$ (**is** $\neg (Ifm (a\#bs) (?M p))$)
and *ex*: *Ifm (x#bs) p* (**is** $?I x p$)
shows $\exists (s,m) \in set (uset p). x \geq Inum (a\#bs) s / real m$ (**is** $\exists (s,m) \in ?U p. x \geq ?N a s / real m$)
proof–
have $\exists (s,m) \in set (uset p). real m * x \geq Inum (a\#bs) s$ (**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*: $(s,m) \in set (uset p)$ **and** *mx*: $real m * x \geq ?N a s$ **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** $x \geq ?N a s / real m$
by (*auto simp add: mult-commute*)
thus *?thesis* **using** *smU* **by** *auto*
qed

lemma *rplusinf-uset*:

assumes *lp*: *isrlfm p*
and *nmi*: $\neg (Ifm (a\#bs) (plusinf p))$ (**is** $\neg (Ifm (a\#bs) (?M p))$)
and *ex*: *Ifm (x#bs) p* (**is** $?I x p$)
shows $\exists (s,m) \in set (uset p). x \leq Inum (a\#bs) s / real m$ (**is** $\exists (s,m) \in ?U p. x \leq ?N a s / real m$)
proof–
have $\exists (s,m) \in set (uset p). real m * x \leq Inum (a\#bs) s$ (**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*: $(s,m) \in set (uset p)$ **and** *mx*: $real m * x \leq ?N a s$ **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** $x \leq ?N a s / real m$
by (*auto simp add: mult-commute*)
thus *?thesis* **using** *smU* **by** *auto*
qed

lemma *lin-dense*:

assumes *lp*: *isrlfm p*
and *noS*: $\forall t. l < t \wedge t < u \longrightarrow t \notin (\lambda (t,n). Inum (x\#bs) t / real n) ' set (uset p)$
(is $\forall t. - \wedge - \longrightarrow t \notin (\lambda (t,n). ?N x t / real n) ' (?U p)$ **)**
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:  $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$  by
auto
  with ly yu have yne:  $y \neq - ?N x e / \text{real } c$  by auto
  hence  $y < (- ?N x e) / \text{real } c \vee y > (- ?N x e) / \text{real } c$  by auto
  moreover {assume y:  $y < (- ?N x e) / \text{real } c$ 
    hence  $y * \text{real } c < - ?N x e$ 
    by (simp add: pos-less-divide-eq[OF cp, where a=y and b=-?N x e,
symmetric])
    hence  $\text{real } c * y + ?N x e < 0$  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:  $y > (- ?N x e) / \text{real } c$ 
    with yu have eu:  $u > (- ?N x e) / \text{real } c$  by auto
    with noSc ly yu have  $(- ?N x e) / \text{real } c \leq l$  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:  $x \leq (- ?N x e) / \text{real } c$ 
  by (simp only: pos-le-divide-eq[OF cp, where a=x and b=-?N x e])
  from prems have noSc:  $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$  by
auto
  with ly yu have yne:  $y \neq - ?N x e / \text{real } c$  by auto
  hence  $y < (- ?N x e) / \text{real } c \vee y > (- ?N x e) / \text{real } c$  by auto
  moreover {assume y:  $y < (- ?N x e) / \text{real } c$ 
    hence  $y * \text{real } c < - ?N x e$ 
    by (simp add: pos-less-divide-eq[OF cp, where a=y and b=-?N x e,
symmetric])
    hence  $\text{real } c * y + ?N x e < 0$  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:  $y > (- ?N x e) / \text{real } c$ 
    with yu have eu:  $u > (- ?N x e) / \text{real } c$  by auto
    with noSc ly yu have  $(- ?N x e) / \text{real } c \leq l$  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:  $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$  by
auto
with ly yu have yne:  $y \neq - ?N x e / \text{real } c$  by auto
hence  $y < (- ?N x e) / \text{real } c \vee y > (- ?N x e) / \text{real } c$  by auto
moreover {assume y:  $y > (- ?N x e) / \text{real } c$ 
  hence  $y * \text{real } c > - ?N x e$ 
  by (simp add: pos-divide-less-eq[OF cp, where a=y and b=-?N x e,
symmetric])
  hence  $\text{real } c * y + ?N x e > 0$  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:  $y < (- ?N x e) / \text{real } c$ 
  with ly have eu:  $l < (- ?N x e) / \text{real } c$  by auto
  with noSc ly yu have  $(- ?N x e) / \text{real } c \geq u$  by (cases  $(- ?N x e) / \text{real } c$ 
> 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:  $x \geq (- ?N x e) / \text{real } c$ 
  by (simp only: pos-divide-le-eq[OF cp, where a=x and b=-?N x e])
from prems have noSc:  $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$  by
auto
with ly yu have yne:  $y \neq - ?N x e / \text{real } c$  by auto
hence  $y < (- ?N x e) / \text{real } c \vee y > (- ?N x e) / \text{real } c$  by auto
moreover {assume y:  $y > (- ?N x e) / \text{real } c$ 
  hence  $y * \text{real } c > - ?N x e$ 
  by (simp add: pos-divide-less-eq[OF cp, where a=y and b=-?N x e,
symmetric])
  hence  $\text{real } c * y + ?N x e > 0$  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:  $y < (- ?N x e) / \text{real } c$ 
  with ly have eu:  $l < (- ?N x e) / \text{real } c$  by auto
  with noSc ly yu have  $(- ?N x e) / \text{real } c \geq u$  by (cases  $(- ?N x e) / \text{real } c$ 
> 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:  $x = (- ?N x e) / \text{real } c$ 
    by (simp only: nonzero-eq-divide-eq[OF cnz, where a=x and b=-?N x e])
    from prems have noSc: $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$  by
auto
    with lx xu have yne:  $x \neq - ?N x e / \text{real } c$  by auto
    with pxc show ?case by simp
next
case (4 c e) hence cp:  $\text{real } c > 0$  and nb: numbound0 e by simp+
    from cp have cnz:  $\text{real } c \neq 0$  by simp
    from prems have noSc: $\forall t. l < t \wedge t < u \longrightarrow t \neq (- ?N x e) / \text{real } c$  by
auto
    with ly yu have yne:  $y \neq - ?N x e / \text{real } c$  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:  $P (x::\text{real})$ 
    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 $xMne fMx$ 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 $?N = \lambda x \ t. Inum \ (x\#bs) \ t$
 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 $?M = (\lambda (t,c). ?N \ a \ t \ / \ real \ c) \ ' \ ?U$
 have $fM: finite \ ?M$ by *auto*

from $rminusinf\text{-}uset[OF\ lp\ nmi\ pa]$ $rplusinf\text{-}uset[OF\ lp\ npi\ pa]$
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**
 $tnU: (t,n) \in ?U$ **and** $smU: (s,m) \in ?U$
and $xs1: a \leq ?N\ x\ s / real\ m$ **and** $tx1: a \geq ?N\ x\ t / real\ n$ **by** *blast*
from $uset\text{-}l[OF\ lp]$ $tnU\ smU\ numbound0\text{-}I$ **where** $bs=bs$ **and** $b=x$ **and** $b'=a]$
 $xs1\ tx1$ **have** $xs: a \leq ?N\ a\ s / real\ m$ **and** $tx: a \geq ?N\ a\ t / real\ n$ **by** *auto*
from tnU **have** $Mne: ?M \neq \{\}$ **by** *auto*
hence $Une: ?U \neq \{\}$ **by** *simp*
let $?l = Min\ ?M$
let $?u = Max\ ?M$
have $linM: ?l \in ?M$ **using** $fM\ Mne$ **by** *simp*
have $uinM: ?u \in ?M$ **using** $fM\ Mne$ **by** *simp*
have $tnM: ?N\ a\ t / real\ n \in ?M$ **using** tnU **by** *auto*
have $smM: ?N\ a\ s / real\ m \in ?M$ **using** smU **by** *auto*
have $lM: \forall t \in ?M. ?l \leq t$ **using** $Mne\ fM$ **by** *auto*
have $Mu: \forall t \in ?M. t \leq ?u$ **using** $Mne\ fM$ **by** *auto*
have $?l \leq ?N\ a\ t / real\ n$ **using** $tnM\ Mne$ **by** *simp* **hence** $lx: ?l \leq a$ **using**
 tx **by** *simp*
have $?N\ a\ s / real\ m \leq ?u$ **using** $smM\ Mne$ **by** *simp* **hence** $xu: a \leq ?u$ **using**
 xs **by** *simp*
from $finite\text{-}set\text{-}intervals2$ **where** $P=\lambda\ x. ?I\ x\ p, OF\ pa\ lx\ xu\ linM\ uinM\ fM\ lM$
 $Mu]$
have $(\exists\ s \in ?M. ?I\ s\ p) \vee$
 $(\exists\ t1 \in ?M. \exists\ t2 \in ?M. (\forall\ y. t1 < y \wedge y < t2 \longrightarrow y \notin ?M) \wedge t1 < a \wedge a$
 $< t2 \wedge ?I\ a\ p) .$
moreover $\{ \text{fix } u \text{ assume } um: u \in ?M \text{ and } pu: ?I\ u\ p$
hence $\exists (tu,nu) \in ?U. u = ?N\ a\ tu / real\ nu$ **by** *auto*
then obtain $tu\ nu$ **where** $tuU: (tu,nu) \in ?U$ **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 $\exists\ t1 \in ?M. \exists\ t2 \in ?M. (\forall\ y. t1 < y \wedge y < t2 \longrightarrow y \notin ?M) \wedge t1$
 $< a \wedge a < t2 \wedge ?I\ a\ p$
then obtain $t1$ **and** $t2$ **where** $t1M: t1 \in ?M$ **and** $t2M: t2 \in ?M$
and $noM: \forall\ y. t1 < y \wedge y < t2 \longrightarrow y \notin ?M$ **and** $t1x: t1 < a$ **and** $xt2: a$
 $< t2$ **and** $px: ?I\ a\ p$
by *blast*
from $t1M$ **have** $\exists (t1u,t1n) \in ?U. t1 = ?N\ a\ t1u / real\ t1n$ **by** *auto*
then obtain $t1u\ t1n$ **where** $t1uU: (t1u,t1n) \in ?U$ **and** $t1u: t1 = ?N\ a\ t1u$
 $/ real\ t1n$ **by** *blast*
from $t2M$ **have** $\exists (t2u,t2n) \in ?U. t2 = ?N\ a\ t2u / real\ t2n$ **by** *auto*
then obtain $t2u\ t2n$ **where** $t2uU: (t2u,t2n) \in ?U$ **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** $lnU: (l,n) \in ?U$ **and** $smU: (s,m) \in ?U$
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 $numbound0-I[OF\ nbl, \text{ where } bs=bs \text{ and } b=a \text{ and } b'=x]$
 $numbound0-I[OF\ nbs, \text{ where } bs=bs \text{ and } b=a \text{ and } b'=x]\ pu$
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 $(\exists\ x. Ifm\ (x\#bs)\ p) = ((Ifm\ (x\#bs)\ (minusinf\ p)) \vee (Ifm\ (x\#bs)\ (plusinf\ p))) \vee (\exists\ (t,n) \in set\ (uset\ p). \exists\ (s,m) \in set\ (uset\ p). Ifm\ (((Inum\ (x\#bs)\ t) / real\ n + (Inum\ (x\#bs)\ s) / real\ m) / 2) \# bs)\ p))$
(is $(\exists\ x. ?I\ x\ p) = (?M \vee ?P \vee ?F)$ **is** $?E = ?D)$
proof
assume $px: \exists\ x. ?I\ x\ p$
have $?M \vee ?P \vee (\neg\ ?M \wedge \neg\ ?P)$ **by** *blast*
moreover {**assume** $?M \vee ?P$ **hence** $?D$ **by** *blast*}
moreover {**assume** $nmi: \neg\ ?M$ **and** $npi: \neg\ ?P$
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-eqsubst*:

assumes $lp: isrlfm\ p$
shows $(\exists\ x. Ifm\ (x\#bs)\ p) = ((Ifm\ (x\#bs)\ (minusinf\ p)) \vee (Ifm\ (x\#bs)\ (plusinf\ p))) \vee (\exists\ (t,k) \in set\ (uset\ p). \exists\ (s,l) \in set\ (uset\ p). Ifm\ (x\#bs)\ (usubst\ p\ (Add\ (Mul\ l\ t)\ (Mul\ k\ s)\ ,\ 2*k*l))))$
(is $(\exists\ x. ?I\ x\ p) = (?M \vee ?P \vee ?F)$ **is** $?E = ?D)$
proof
assume $px: \exists\ x. ?I\ x\ p$
have $?M \vee ?P \vee (\neg\ ?M \wedge \neg\ ?P)$ **by** *blast*

```

moreover {assume ? $M \vee ?P$  hence ? $D$  by blast}
moreover {assume  $nmi: \neg ?M$  and  $npi: \neg ?P$ 
  let ? $f = \lambda (t,n). Inum (x\#bs) t / real\ n$ 
  let ? $N = \lambda t. Inum (x\#bs) t$ 
  {fix  $t\ n\ s\ m$  assume  $(t,n) \in set (uset\ p)$  and  $(s,m) \in set (uset\ p)$ 
    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  $usubst-I[OF\ lp\ mnp\ st-nb, \text{where } x=x \text{ and } bs=bs]$ 
    have ? $I\ x (usubst\ p\ (?st, 2*n*m)) = ?I\ ((?N\ t / real\ n + ?N\ s / real\ m) / 2)$ 
  }
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  $(t,k) \in set (uset\ p)$  and  $(s,l) \in set (uset\ p)$ 
  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  $usubst-I[OF\ lp\ mnp\ st-nb, \text{where } bs=bs]$   $px$  have ? $E$  by auto}
ultimately show ? $E$  by blast
qed

```

```

constdefs  $ferrack:: fm \Rightarrow fm$ 
   $ferrack\ p \equiv (let\ p' = rlfm\ (simpfm\ p); mp = minusinf\ p'; pp = plusinf\ p'$ 
    in if  $(mp = T \vee pp = T)$  then  $T$  else
      (let  $U = remdps(map\ simp-num-pair$ 
        ( $map\ (\lambda ((t,n),(s,m)). (Add\ (Mul\ m\ t)\ (Mul\ n\ s), 2*n*m))$ 
        ( $alluopairs\ (uset\ p')$ )))
        in  $decr\ (disj\ mp\ (disj\ pp\ (evaldjf\ (simpfm\ o\ (usubst\ p'))\ U))))$ 

```

lemma *uset-cong-aux*:

```

assumes  $Ul: \forall (t,n) \in set\ U. numbound0\ t \wedge n > 0$ 
shows  $((\lambda (t,n). Inum\ (x\#bs)\ t / real\ n) ' (set\ (map\ (\lambda ((t,n),(s,m)). (Add\ (Mul\ m\ t)\ (Mul\ n\ s), 2*n*m))\ (alluopairs\ U)))) = ((\lambda ((t,n),(s,m)). (Inum\ (x\#bs)\ t$ 

```

```

/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 ... = (($\lambda(t, n)$). *Inum* ($x \# bs$) $t / \text{real } n$) ($\lambda((t, n), s, m)$). (*Add* (*Mul* m t) (*Mul* n s), $2 * n * m$)) ((t', n'), (s', m'))) **by** (*simp add: st'*)
finally show (*Inum* ($x \# bs$) $t / \text{real } n$ + *Inum* ($x \# bs$) $s / \text{real } m$) / 2
 $\in (\lambda(t, n)$. *Inum* ($x \# bs$) $t / \text{real } n$) '
 $(\lambda((t, n), s, m)$. (*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' : (($\lambda(t, n)$). *Inum* ($x \# bs$) $t / \text{real } n$) ' U') = (($\lambda((t, n), (s, m))$). (*Inum* ($x \# bs$) $t / \text{real } n$ + *Inum* ($x \# bs$) $s / \text{real } m$) / 2) ' $(U \times U)$) (**is** $?f$ ' $U' = ?g$ ' $(U \times U)$)
and U : $\forall (t, n) \in U$. *numbound0* $t \wedge n > 0$
and U' : $\forall (t, n) \in U'$. *numbound0* $t \wedge n > 0$
shows ($\exists (t, n) \in U$. $\exists (s, m) \in U$. *Ifm* ($x \# bs$) (*usubst* p (*Add* (*Mul* m t) (*Mul* n s), $2 * n * m$))) = ($\exists (t, n) \in U'$. *Ifm* ($x \# bs$) (*usubst* p (t, n)))
(is $?lhs = ?rhs$)

proof

assume $?lhs$
then obtain $t n s m$ **where** tnU : $(t, n) \in U$ **and** smU : $(s, m) \in U$ **and**
 Pst : *Ifm* ($x \# bs$) (*usubst* p (*Add* (*Mul* m t) (*Mul* n s), $2 * n * m$)) **by** *blast*
let $?N = \lambda t$. *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 $?st = \text{Add} (\text{Mul } m \ t) (\text{Mul } n \ s)$
from *mult-pos-pos[OF np mp]* **have** mpn : *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 : ($?N \ t / \text{real } n + ?N \ s / \text{real } m$) / 2 = $?N \ ?st / \text{real } (2 * n * m)$
using $mp \ np$ **by** (*simp add: ring-simps add-divide-distrib*)
from tnU smU UU' **have** $?g ((t, n), (s, m)) \in ?f$ ' U' **by** *blast*
hence $\exists (t', n') \in U'$. $?g ((t, n), (s, m)) = ?f (t', n')$
by *auto* (*rule-tac x=(a,b) in beXI, auto*)
then obtain $t' n'$ **where** tnU' : $(t', n') \in U'$ **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 mpn stnb, where bs=bs and x=x]* Pst
have $Pst2$: *Ifm* ($x \# bs$) (*Add* (*Mul* m t) (*Mul* n s)) / *real* ($2 * n * m$)
 $\# bs$) p **by** *simp*
from *conject1[OF usubst-I[OF lp np' tnb', where bs=bs and x=x], symmetric]*
 $th[simplified \text{split-def fst-conv snd-conv, symmetric}]$ $Pst2[simplified \text{st[symmetric]}]$
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' : $(t', n') \in U'$ **and** Pt' : *Ifm* ($x \# bs$) (*usubst* p (t', n'))

by blast
 from $tnU' UU'$ have $?f (t',n') \in ?g \cdot (U \times U)$ by blast
 hence $\exists ((t,n),(s,m)) \in (U \times U). ?f (t',n') = ?g ((t,n),(s,m))$
 by auto (rule-tac $x=(a,b)$ in $bezI$, auto)
 then obtain $t n s m$ where $tnU: (t,n) \in U$ and $smU: (s,m) \in U$ and
 $th: ?f (t',n') = ?g((t,n),(s,m))$ by blast
 let $?N = \lambda t. Inum (x \# bs) t$
 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 $qfree (ferrack p) \wedge ((Ifm bs (ferrack p)) = (\exists x. Ifm (x \# bs) p))$
 (is - $\wedge (?rhs = ?lhs)$)
 proof-
 let $?I = \lambda x p. Ifm (x \# bs) p$
 fix x
 let $?N = \lambda t. Inum (x \# bs) t$
 let $?q = rlfm (simpfm p)$
 let $?U = uset ?q$
 let $?Up = alluopairs ?U$
 let $?g = \lambda ((t,n),(s,m)). (Add (Mul m t) (Mul n s), 2*n*m)$
 let $?S = map ?g ?Up$
 let $?SS = map simp-num-pair ?S$
 let $?Y = remdps ?SS$
 let $?f = (\lambda (t,n). ?N t / real n)$
 let $?h = \lambda ((t,n),(s,m)). (?N t / real n + ?N s / real m) / 2$
 let $?F = \lambda p. \exists a \in set (uset p). \exists b \in set (uset p). ?I x (usubst p (?g(a,b)))$
 let $?ep = evaldif (simpfm o (usubst ?q)) ?Y$
 from $rlfm-I[OF simpfm-qf[OF qf]]$ have $lq: isrlfm ?q$ by blast
 from $alluopairs-set1[where xs=?U]$ have $UpU: set ?Up \leq (set ?U \times set ?U)$
 by simp
 from $uset-l[OF lq]$ have $U-l: \forall (t,n) \in set ?U. numbound0 t \wedge n > 0$.
 from $U-l UpU$
 have $\forall ((t,n),(s,m)) \in set ?Up. numbound0 t \wedge n > 0 \wedge numbound0 s \wedge m >$

```

0 by auto
hence Snb:  $\forall (t,n) \in \text{set } ?S. \text{numbound0 } t \wedge n > 0$ 
  by (auto simp add: mult-pos-pos)
have Y-l:  $\forall (t,n) \in \text{set } ?Y. \text{numbound0 } t \wedge n > 0$ 
proof-
  { fix t n assume tnY:  $(t,n) \in \text{set } ?Y$ 
    hence  $(t,n) \in \text{set } ?SS$  by simp
    hence  $\exists (t',n') \in \text{set } ?S. \text{simp-num-pair } (t',n') = (t,n)$ 
    by (auto simp add: split-def) (rule-tac  $x=((aa,ba),(ab,bb))$ ) in bexI, simp-all
    then obtain t' n' where tn'S:  $(t',n') \in \text{set } ?S$  and tns:  $\text{simp-num-pair}$ 
       $(t',n') = (t,n)$  by blast
    from tn'S Snb have tnb:  $\text{numbound0 } t'$  and np:  $n' > 0$  by auto
    from  $\text{simp-num-pair-l}[OF \text{ tnb np tns}]$ 
    have  $\text{numbound0 } t \wedge n > 0$  . }
  thus ?thesis by blast
qed

have YU:  $(?f ' \text{set } ?Y) = (?h ' (\text{set } ?U \times \text{set } ?U))$ 
proof-
  from  $\text{simp-num-pair-ci}[\text{where } bs=x\#bs]$  have
     $\forall x. (?f \circ \text{simp-num-pair}) x = ?f x$  by auto
  hence th:  $?f \circ \text{simp-num-pair} = ?f$  using ext by blast
  have  $(?f ' \text{set } ?Y) = ((?f \circ \text{simp-num-pair}) ' \text{set } ?S)$  by (simp add: image-compose)
  also have ... =  $(?f ' \text{set } ?S)$  by (simp add: th)
  also have ... =  $((?f \circ ?g) ' \text{set } ?Up)$ 
    by (simp only: set-map o-def image-compose[symmetric])
  also have ... =  $(?h ' (\text{set } ?U \times \text{set } ?U))$ 
    using  $\text{uset-cong-aux}[OF U-l, \text{ where } x=x \text{ and } bs=bs, \text{ simplified set-map}$ 
       $\text{image-compose[symmetric]}]$  by blast
  finally show ?thesis .
qed
have  $\forall (t,n) \in \text{set } ?Y. \text{bound0 } (\text{simpfm } (\text{usubst } ?q (t,n)))$ 
proof-
  { fix t n assume tnY:  $(t,n) \in \text{set } ?Y$ 
    with Y-l have tnb:  $\text{numbound0 } t$  and np:  $\text{real } n > 0$  by auto
    from  $\text{usubst-I}[OF lq np tnb]$ 
    have  $\text{bound0 } (\text{usubst } ?q (t,n))$  by simp hence  $\text{bound0 } (\text{simpfm } (\text{usubst } ?q$ 
       $(t,n)))$ 
    using  $\text{simpfm-bound0}$  by simp
    thus ?thesis by blast
  }
qed
hence ep-nb:  $\text{bound0 } ?ep$  using  $\text{evaldjf-bound0}[\text{where } xs=?Y \text{ and } f=\text{simpfm}$ 
   $\circ (\text{usubst } ?q)]$  by auto
let ?mp =  $\text{minusinf } ?q$ 
let ?pp =  $\text{plusinf } ?q$ 
let ?M =  $?I x ?mp$ 
let ?P =  $?I x ?pp$ 
let ?res =  $\text{disj } ?mp (\text{disj } ?pp ?ep)$ 
from  $\text{rminusinf-bound0}[OF lq] \text{ rplusinf-bound0}[OF lq] \text{ ep-nb}$ 

```

```

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-eqsubst[OF lq, where bs=bs and x=x] have lhfr: ?lhs = (?M  $\vee$ 
?P  $\vee$  ?F ?q)
  by (simp only: split-def fst-conv snd-conv)
also have ... = (?M  $\vee$  ?P  $\vee$  ( $\exists$  (t,n)  $\in$  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  $\vee$  ?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  $\Rightarrow$  fm
  linrqe  $\equiv$  ( $\lambda$  p. qelim (prep p) ferrack)

theorem linrqe: (Ifm bs (linrqe p) = Ifm bs p)  $\wedge$  qfree (linrqe p)
using ferrack qelim-ci prep
unfolding linrqe-def by auto

definition
  ferrack-test :: unit  $\Rightarrow$  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  $\ll$  Ferrack.ferrack-test ()  $\gg$ 

use linreif.ML
oracle linr-oracle (term) = ReflectedFerrack.linrqe-oracle
use linrtac.ML
setup LinrTac.setup

end

```