Miscellaneous HOL-Complex Examples

June 8, 2008

Contents

1 Binary arithmetic examples

theory *BinEx*
imports *Complex-Main*
begin

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

1.1 Real Arithmetic

1.1.1 Addition

lemma (*1359*::*real*) + *−2468* = *−1109*
⟨*proof*⟩

lemma (*93746*::*real*) + *−46375* = *47371*
⟨*proof*⟩

1.1.2 Negation

lemma $-\ (65745::real) = -65745$
⟨*proof*⟩

lemma $-\ (-54321::real) = 54321$
⟨*proof*⟩

1.1.3 Multiplication

lemma $(-84::real) * 51 = -4284$
⟨*proof*⟩

lemma $(255::real) * 255 = 65025$
⟨*proof*⟩

lemma $(1359::real) * -2468 = -3354012$
⟨*proof*⟩

1.1.4 Inequalities

lemma $(89::real) * 10 \neq 889$
⟨*proof*⟩

lemma $(13::real) < 18 - 4$
⟨*proof*⟩

lemma $(-345::real) < -242 + -100$
⟨*proof*⟩

lemma $(13557456::real) < 18678654$
⟨*proof*⟩

lemma $(999999::real) \leq (1000001 + 1) - 2$
⟨*proof*⟩

lemma $(1234567::real) \leq 1234567$
⟨*proof*⟩

1.1.5 Powers

lemma $2\ \hat{}\ 15 = (32768::real)$
⟨*proof*⟩

lemma $-3\ \hat{}\ 7 = (-2187::real)$
⟨*proof*⟩

lemma $13\ \hat{}\ 7 = (62748517::real)$
⟨*proof*⟩

lemma $3 \hat{} 15 = (14348907::real)$
⟨*proof*⟩

lemma $-5 \hat{} 11 = (-48828125::real)$
⟨*proof*⟩

1.1.6 Tests

lemma $(x + y = x) = (y = (0::real))$
⟨*proof*⟩

lemma $(x + y = y) = (x = (0::real))$
⟨*proof*⟩

lemma $(x + y = (0::real)) = (x = -y)$
⟨*proof*⟩

lemma $(x + y = (0::real)) = (y = -x)$
⟨*proof*⟩

lemma $((x + y) < (x + z)) = (y < (z::real))$
⟨*proof*⟩

lemma $((x + z) < (y + z)) = (x < (y::real))$
⟨*proof*⟩

lemma $(\neg\ x < y) = (y \leq (x::real))$
⟨*proof*⟩

lemma $\neg\ (x < y \wedge y < (x::real))$
⟨*proof*⟩

lemma $(x::real) < y ==> \neg\ y < x$
⟨*proof*⟩

lemma $((x::real) \neq y) = (x < y \vee y < x)$
⟨*proof*⟩

lemma $(\neg\ x \leq y) = (y < (x::real))$
⟨*proof*⟩

lemma $x \leq y \vee y \leq (x::real)$
⟨*proof*⟩

lemma $x \leq y \vee y < (x::real)$
⟨*proof*⟩

lemma $x < y \vee y \leq (x::real)$
⟨*proof*⟩

lemma $x \leq (x::real)$
⟨*proof*⟩

lemma $((x::real) \leq y) = (x < y \vee x = y)$
⟨*proof*⟩

lemma $((x::real) \leq y \wedge y \leq x) = (x = y)$
⟨*proof*⟩

lemma $\neg(x < y \wedge y \leq (x::real))$
⟨*proof*⟩

lemma $\neg(x \leq y \wedge y < (x::real))$
⟨*proof*⟩

lemma $(-x < (0::real)) = (0 < x)$
⟨*proof*⟩

lemma $((0::real) < -x) = (x < 0)$
⟨*proof*⟩

lemma $(-x \leq (0::real)) = (0 \leq x)$
⟨*proof*⟩

lemma $((0::real) \leq -x) = (x \leq 0)$
⟨*proof*⟩

lemma $(x::real) = y \vee x < y \vee y < x$
⟨*proof*⟩

lemma $(x::real) = 0 \vee 0 < x \vee 0 < -x$
⟨*proof*⟩

lemma $(0::real) \leq x \vee 0 \leq -x$
⟨*proof*⟩

lemma $((x::real) + y \leq x + z) = (y \leq z)$
⟨*proof*⟩

lemma $((x::real) + z \leq y + z) = (x \leq y)$
⟨*proof*⟩

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

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

lemma $(0::real) \leq x \wedge 0 \leq y ==> 0 \leq x + y$
⟨*proof*⟩

lemma $(0::real) < x \wedge 0 < y ==> 0 < x + y$
⟨*proof*⟩

lemma $(-x < y) = (0 < x + (y::real))$
⟨*proof*⟩

lemma $(x < -y) = (x + y < (0::real))$
⟨*proof*⟩

lemma $(y < x + -z) = (y + z < (x::real))$
⟨*proof*⟩

lemma $(x + -y < z) = (x < z + (y::real))$
⟨*proof*⟩

lemma $x \leq y ==> x < y + (1::real)$
⟨*proof*⟩

lemma $(x - y) + y = (x::real)$
⟨*proof*⟩

lemma $y + (x - y) = (x::real)$
⟨*proof*⟩

lemma $x - x = (0::real)$
⟨*proof*⟩

lemma $(x - y = 0) = (x = (y::real))$
⟨*proof*⟩

lemma $((0::real) \leq x + x) = (0 \leq x)$
⟨*proof*⟩

lemma $(-x \leq x) = ((0::real) \leq x)$
⟨*proof*⟩

lemma $(x \leq -x) = (x \leq (0::real))$
⟨*proof*⟩

lemma $(-x = (0::real)) = (x = 0)$
⟨*proof*⟩

lemma $-(x - y) = y - (x::real)$
⟨*proof*⟩

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

⟨*proof*⟩

lemma $((0::real) \leq x - y) = (y \leq x)$
⟨*proof*⟩

lemma $(x + y) - x = (y::real)$
⟨*proof*⟩

lemma $(-x = y) = (x = (-y::real))$
⟨*proof*⟩

lemma $x < (y::real) ==> \neg(x = y)$
⟨*proof*⟩

lemma $(x \leq x + y) = ((0::real) \leq y)$
⟨*proof*⟩

lemma $(y \leq x + y) = ((0::real) \leq x)$
⟨*proof*⟩

lemma $(x < x + y) = ((0::real) < y)$
⟨*proof*⟩

lemma $(y < x + y) = ((0::real) < x)$
⟨*proof*⟩

lemma $(x - y) - x = (-y::real)$
⟨*proof*⟩

lemma $(x + y < z) = (x < z - (y::real))$
⟨*proof*⟩

lemma $(x - y < z) = (x < z + (y::real))$
⟨*proof*⟩

lemma $(x < y - z) = (x + z < (y::real))$
⟨*proof*⟩

lemma $(x \leq y - z) = (x + z \leq (y::real))$
⟨*proof*⟩

lemma $(x - y \leq z) = (x \leq z + (y::real))$
⟨*proof*⟩

lemma $(-x < -y) = (y < (x::real))$
⟨*proof*⟩

lemma $(-x \leq -y) = (y \leq (x::real))$
⟨*proof*⟩

lemma $(a + b) - (c + d) = (a - c) + (b - (d::real))$
⟨*proof*⟩

lemma $(0::real) - x = -x$
⟨*proof*⟩

lemma $x - (0::real) = x$
⟨*proof*⟩

lemma $w \leq x \wedge y < z \Longrightarrow w + y < x + (z::real)$
⟨*proof*⟩

lemma $w < x \wedge y \leq z \Longrightarrow w + y < x + (z::real)$
⟨*proof*⟩

lemma $(0::real) \leq x \wedge 0 < y \Longrightarrow 0 < x + (y::real)$
⟨*proof*⟩

lemma $(0::real) < x \wedge 0 \leq y \Longrightarrow 0 < x + y$
⟨*proof*⟩

lemma $-x - y = -(x + (y::real))$
⟨*proof*⟩

lemma $x - (-y) = x + (y::real)$
⟨*proof*⟩

lemma $-x - -y = y - (x::real)$
⟨*proof*⟩

lemma $(a - b) + (b - c) = a - (c::real)$
⟨*proof*⟩

lemma $(x = y - z) = (x + z = (y::real))$
⟨*proof*⟩

lemma $(x - y = z) = (x = z + (y::real))$
⟨*proof*⟩

lemma $x - (x - y) = (y::real)$
⟨*proof*⟩

lemma $x - (x + y) = -(y::real)$
⟨*proof*⟩

lemma $x = y \Longrightarrow x \leq (y::real)$
⟨*proof*⟩

lemma $(0::real) < x \Longrightarrow \neg(x = 0)$
⟨*proof*⟩

lemma $(x + y) * (x - y) = (x * x) - (y * y)$
⟨*proof*⟩

lemma $(-x = -y) = (x = (y::real))$
⟨*proof*⟩

lemma $(-x < -y) = (y < (x::real))$
⟨*proof*⟩

lemma $!!a::real.\ a \leq b \Longrightarrow c \leq d \Longrightarrow x + y < z \Longrightarrow a + c \leq b + d$
⟨*proof*⟩

lemma $!!a::real.\ a < b \Longrightarrow c < d \Longrightarrow a - d \leq b + (-c)$
⟨*proof*⟩

lemma $!!a::real.\ a \leq b \Longrightarrow b + b \leq c \Longrightarrow a + a \leq c$
⟨*proof*⟩

lemma $!!a::real.\ a + b \leq i + j \Longrightarrow a \leq b \Longrightarrow i \leq j \Longrightarrow a + a \leq j + j$
⟨*proof*⟩

lemma $!!a::real.\ a + b < i + j \Longrightarrow a < b \Longrightarrow i < j \Longrightarrow a + a < j + j$
⟨*proof*⟩

lemma $!!a::real.\ a + b + c \leq i + j + k \wedge a \leq b \wedge b \leq c \wedge i \leq j \wedge j \leq k \longrightarrow a + a + a \leq k + k + k$
⟨*proof*⟩

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l \Longrightarrow a \leq b \Longrightarrow b \leq c \Longrightarrow c \leq d \Longrightarrow i \leq j \Longrightarrow j \leq k \Longrightarrow k \leq l \Longrightarrow a \leq l$
⟨*proof*⟩

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l \Longrightarrow a \leq b \Longrightarrow b \leq c \Longrightarrow c \leq d \Longrightarrow i \leq j \Longrightarrow j \leq k \Longrightarrow k \leq l \Longrightarrow a + a + a + a \leq l + l + l + l$
⟨*proof*⟩

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l \Longrightarrow a \leq b \Longrightarrow b \leq c \Longrightarrow c \leq d \Longrightarrow i \leq j \Longrightarrow j \leq k \Longrightarrow k \leq l \Longrightarrow a + a + a + a + a \leq l + l + l + l + i$
⟨*proof*⟩

lemma $!!a::real.\ a + b + c + d \leq i + j + k + l \Longrightarrow a \leq b \Longrightarrow b \leq c \Longrightarrow c \leq d \Longrightarrow i \leq j \Longrightarrow j \leq k \Longrightarrow k \leq l \Longrightarrow a + a + a + a + a + a \leq l + l + l + l + i + l$
⟨*proof*⟩

1.2 Complex Arithmetic

lemma $(1359 + 93746*ii) - (2468 + 46375*ii) = -1109 + 47371*ii$
⟨*proof*⟩

lemma $-(65745 + -47371*ii) = -65745 + 47371*ii$
⟨*proof*⟩

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

lemma $(1 + ii) * (1 - ii) = 2$
⟨*proof*⟩

lemma $(1 + 2*ii) * (1 + 3*ii) = -5 + 5*ii$
⟨*proof*⟩

lemma $(-84 + 255*ii) + (51 * 255*ii) = -84 + 13260 * ii$
⟨*proof*⟩

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

No powers (not supported yet)

end

2 Square roots of primes are irrational

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

2.1 Preliminaries

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

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

theorem *rationals-rep* [*elim?*]:
  assumes $x \in \mathbb{Q}$
  obtains $m\ n$ **where** $n \neq 0$ **and** $|x| = real\ m\ /\ real\ n$ **and** $gcd\ (m, n) = 1$
⟨*proof*⟩

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*) $\notin \mathbb{Q}$
⟨*proof*⟩

corollary *sqrt* (*real* (*2*::*nat*)) $\notin \mathbb{Q}$
⟨*proof*⟩

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*) $\notin \mathbb{Q}$
⟨*proof*⟩

end

3 Square roots of primes are irrational (script version)

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

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

3.1 Preliminaries

lemma *prime-nonzero*: *prime p* $\Longrightarrow$ *p* $\neq$ *0*
⟨*proof*⟩

lemma *prime-dvd-other-side*:
n $*$ *n* = *p* $*$ (*k* $*$ *k*) $\Longrightarrow$ *prime p* $\Longrightarrow$ *p dvd n*
⟨*proof*⟩

lemma *reduction*: *prime p* $\Longrightarrow$
0 $<$ *k* $\Longrightarrow$ *k* $*$ *k* = *p* $*$ (*j* $*$ *j*) $\Longrightarrow$ *k* $<$ *p* $*$ *j* $\wedge$ *0* $<$ *j*
⟨*proof*⟩

lemma *rearrange*: (*j*::*nat*) $*$ (*p* $*$ *j*) = *k* $*$ *k* $\Longrightarrow$ *k* $*$ *k* = *p* $*$ (*j* $*$ *j*)
⟨*proof*⟩

lemma *prime-not-square*:

prime p $\Longrightarrow$ $(\bigwedge k.\ 0 < k \Longrightarrow m * m \neq p * (k * k))$
⟨*proof*⟩

3.2 The set of rational numbers

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

3.3 Main theorem

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

theorem *prime-sqrt-irrational*:
prime p $\Longrightarrow$ $x * x = real\ p \Longrightarrow 0 \leq x \Longrightarrow x \notin \mathbb{Q}$
⟨*proof*⟩

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*: $\forall M.\ \exists N.\ 0 < N\ \&\ (\forall m.\ 0 < m\ \&\ (m::nat) <= M\ --> m\ dvd\ N)$
⟨*proof*⟩

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

lemma *hypnat-of-nat-le-zero-iff*: $(hypnat\text{-}of\text{-}nat\ n <= 0) = (n = 0)$
⟨*proof*⟩
declare *hypnat-of-nat-le-zero-iff* [*simp*]

lemma *hdvd-by-all*: $\forall M.\ \exists N.\ 0 < N\ \&\ (\forall m.\ 0 < m\ \&\ (m::hypnat) <= M\ --> m\ hdvd\ N)$
⟨*proof*⟩

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

lemma *hypnat-dvd-all-hypnat-of-nat*:
$\exists (N::hypnat).\ 0 < N\ \&\ (\forall n \in -\{0::nat\}.\ hypnat\text{-}of\text{-}nat(n)\ hdvd\ N)$
⟨*proof*⟩

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

lemma *starprime*:
$starprime = \{p.\ 1 < p\ \&\ (\forall m.\ m\ hdvd\ p\ --> m = 1 \mid m = p)\}$
⟨*proof*⟩

lemma *prime-two*: $prime\ 2$
⟨*proof*⟩
declare *prime-two* [*simp*]

lemma *prime-factor-exists* [*rule-format*]: $Suc\ 0 < n\ --> (\exists k.\ prime\ k\ \&\ k\ dvd\ n)$
⟨*proof*⟩

lemma *hyperprime-factor-exists* [*rule-format*]:
$!!n.\ 1 < n ==> (\exists k \in starprime.\ k\ hdvd\ n)$
⟨*proof*⟩

lemma *NatStar-hypnat-of-nat*: $finite\ A ==> *s*\ A = hypnat\text{-}of\text{-}nat\ `\ A$
⟨*proof*⟩

4.1 Another characterization of infinite set of natural numbers

lemma *finite-nat-set-bounded*: *finite* N ==> $\exists\, n.\ (\forall\, i \in N.\ i<(n::nat))$
⟨*proof*⟩

lemma *finite-nat-set-bounded-iff*: *finite* $N = (\exists\, n.\ (\forall\, i \in N.\ i<(n::nat)))$
⟨*proof*⟩

lemma *not-finite-nat-set-iff*: $(\sim$ *finite* $N) = (\forall\, n.\ \exists\, i \in N.\ n <= (i::nat))$
⟨*proof*⟩

lemma *bounded-nat-set-is-finite2*: $(\forall\, i \in N.\ i<=(n::nat))$ ==> *finite* N
⟨*proof*⟩

lemma *finite-nat-set-bounded2*: *finite* N ==> $\exists\, n.\ (\forall\, i \in N.\ i<=(n::nat))$
⟨*proof*⟩

lemma *finite-nat-set-bounded-iff2*: *finite* $N = (\exists\, n.\ (\forall\, i \in N.\ i<=(n::nat)))$
⟨*proof*⟩

lemma *not-finite-nat-set-iff2*: $(\sim$ *finite* $N) = (\forall\, n.\ \exists\, i \in N.\ n < (i::nat))$
⟨*proof*⟩

4.2 An injective function cannot define an embedded natural number

lemma *lemma-infinite-set-singleton*: $\forall\, m\ n.\ m \neq n$ --> $f\ n \neq f\ m$
==> $\{n.\ f\ n = N\} = \{\}\ |\ (\exists\, m.\ \{n.\ f\ n = N\} = \{m\})$
⟨*proof*⟩

lemma *inj-fun-not-hypnat-in-SHNat*:
assumes *inj-f*: *inj* $(f::nat=>nat)$
shows *starfun f whn* $\notin$ *Nats*
⟨*proof*⟩

lemma *range-subset-mem-starsetNat*:
range f <= *A* ==> *starfun f whn* $\in *s*$ *A*
⟨*proof*⟩

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

⟨*proof*⟩

lemma *choicefun-mem-set*: $E \neq \{\} ==> choicefun\ E \in E$
⟨*proof*⟩
declare *choicefun-mem-set* [*simp*]

lemma *injf-max-mem-set*: $[|\ E \neq \{\};\ \forall x.\ \exists y \in E.\ x < y\ |] ==> injf\text{-}max\ n\ E \in E$
⟨*proof*⟩

lemma *injf-max-order-preserving*: $\forall x.\ \exists y \in E.\ x < y ==> injf\text{-}max\ n\ E < injf\text{-}max\ (Suc\ n)\ E$
⟨*proof*⟩

lemma *injf-max-order-preserving2*: $\forall x.\ \exists y \in E.\ x < y$
$==> \forall n\ m.\ m < n --> injf\text{-}max\ m\ E < injf\text{-}max\ n\ E$
⟨*proof*⟩

lemma *inj-injf-max*: $\forall x.\ \exists y \in E.\ x < y ==> inj\ (\%n.\ injf\text{-}max\ n\ E)$
⟨*proof*⟩

lemma *infinite-set-has-order-preserving-inj*:
$[|\ (E::('a::\{order\}\ set)) \neq \{\};\ \forall x.\ \exists y \in E.\ x < y\ |]$
$==> \exists f.\ range\ f <= E\ \&\ inj\ (f::nat => 'a)\ \&\ (\forall m.\ f\ m < f(Suc\ m))$
⟨*proof*⟩

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

lemma *hypnat-infinite-has-nonstandard*:
$\sim finite\ A ==> hypnat\text{-}of\text{-}nat\ `\ A < (\ *s*\ A)$
⟨*proof*⟩

lemma *starsetNat-eq-hypnat-of-nat-image-finite*: $*s*\ A = hypnat\text{-}of\text{-}nat\ `\ A ==> finite\ A$
⟨*proof*⟩

lemma *finite-starsetNat-iff*: $(\ *s*\ A = hypnat\text{-}of\text{-}nat\ `\ A) = (finite\ A)$
⟨*proof*⟩

lemma *hypnat-infinite-has-nonstandard-iff*: $(\sim finite\ A) = (hypnat\text{-}of\text{-}nat\ `\ A < *s*\ A)$
⟨*proof*⟩

4.3 Existence of Infinitely Many Primes: a Nonstandard Proof

lemma *lemma-not-dvd-hypnat-one*: $\sim (\forall n \in -\{0\}.\ hypnat\text{-}of\text{-}nat\ n\ hdvd\ 1)$
⟨*proof*⟩
declare *lemma-not-dvd-hypnat-one* [*simp*]

lemma *lemma-not-dvd-hypnat-one2*: $\exists n \in - \{0\}. \sim$ *hypnat-of-nat n hdvd 1*
⟨*proof*⟩
declare *lemma-not-dvd-hypnat-one2* [*simp*]

lemma *hypnat-gt-zero-gt-one*:
$!!N.$ [| $0 < (N::hypnat)$; $N \neq 1$ |] ==> $1 < N$
⟨*proof*⟩

lemma *hypnat-add-one-gt-one*:
$!!N.$ $0 < N$ ==> $1 < (N::hypnat) + 1$
⟨*proof*⟩

lemma *zero-not-prime*: ¬ *prime 0*
⟨*proof*⟩
declare *zero-not-prime* [*simp*]

lemma *hypnat-of-nat-zero-not-prime*: *hypnat-of-nat 0* ∉ *starprime*
⟨*proof*⟩
declare *hypnat-of-nat-zero-not-prime* [*simp*]

lemma *hypnat-zero-not-prime*:
0 ∉ *starprime*
⟨*proof*⟩
declare *hypnat-zero-not-prime* [*simp*]

lemma *one-not-prime*: ¬ *prime 1*
⟨*proof*⟩
declare *one-not-prime* [*simp*]

lemma *one-not-prime2*: ¬ *prime*(*Suc 0*)
⟨*proof*⟩
declare *one-not-prime2* [*simp*]

lemma *hypnat-of-nat-one-not-prime*: *hypnat-of-nat 1* ∉ *starprime*
⟨*proof*⟩
declare *hypnat-of-nat-one-not-prime* [*simp*]

lemma *hypnat-one-not-prime*: *1* ∉ *starprime*
⟨*proof*⟩
declare *hypnat-one-not-prime* [*simp*]

lemma *hdvd-diff*: $!!k\ m\ n.$ [| *k hdvd m*; *k hdvd n* |] ==> *k hdvd* $(m - n)$
⟨*proof*⟩

lemma *dvd-one-eq-one*: *x dvd* $(1::nat)$ ==> $x = 1$
⟨*proof*⟩

lemma *hdvd-one-eq-one*: $!!x.$ *x hdvd 1* ==> $x = 1$

⟨*proof*⟩

theorem *not-finite-prime*: ~ *finite* {*p*. *prime p*}
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *bigo-LIMSEQ2*: *f* =*o* *g* +*o* *O*(*h*) ==> *h* ----> *0* ==> *f* ----> *a*
==> *g* ----> (*a*::*real*)
⟨*proof*⟩

end

6 Arithmetic Series for Reals

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

lemma *arith-series-real*:
(*2*::*real*) * (∑ *i*∈{..<*n*}. *a* + *of-nat i* * *d*) =
of-nat n * (*a* + (*a* + *of-nat*(*n* − *1*)*d*))
⟨*proof*⟩

end

7 Divergence of the Harmonic Series

theory *HarmonicSeries*
imports *Complex-Main*
begin

8 Abstract

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

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

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

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

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

9 Formal Proof

lemma *two-pow-sub*:
0 < m ⟹ (2::nat) ^ m − 2 ^ (m − 1) = 2 ^ (m − 1)
⟨*proof*⟩

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

lemma *harmonic-aux*:
∀ m>0. (∑ n∈{(2::nat) ^ (m − 1)+1..2 ^ m}. 1/real n) ≥ 1/2
(**is** *∀ m>0. (∑ n∈(?S m). 1/real n) ≥ 1/2*)
⟨*proof*⟩

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

lemma *harmonic-aux2* [*rule-format*]:
0<M ⟹ (∑ n∈{1..(2::nat) ^ M}. 1/real n) =
(1 + (∑ m∈{1..M}. ∑ n∈{(2::nat) ^ (m − 1)+1..2 ^ m}. 1/real n))
(**is** *0<M ⟹ ?LHS M = ?RHS M*)
⟨*proof*⟩

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 $\forall (M::nat).\ (\sum n \in \{1..(2::nat)\hat{}M\}.\ 1\ /\ real\ n) \geq 1 + (real\ M)/2$
(**is** $\forall M.\ ?P\ M \geq$ -)
⟨*proof*⟩

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

theorem *DivergenceOfHarmonicSeries*:
shows $\neg summable\ (\lambda n.\ 1/real\ (Suc\ n))$
(**is** $\neg summable\ ?f$)
⟨*proof*⟩

end

10 Denumerability of the Rationals

theory *DenumRat*
imports
Complex-Main NatPair
begin

lemma *nat-to-int-surj*: $\exists f::nat \Rightarrow int.\ surj\ f$
⟨*proof*⟩

lemma *nat2-to-int2-surj*: $\exists f::(nat*nat) \Rightarrow (int*int).\ surj\ f$
⟨*proof*⟩

lemma *rat-denum*:
$\exists f::nat \Rightarrow rat.\ surj\ f$
⟨*proof*⟩

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 −)

⟨ML⟩

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,+,floor,¡)

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

lemma *alluopairs-set1*: *set* (*alluopairs xs*) ≤ {(*x*,*y*). *x*∈ *set xs* ∧ *y*∈ *set xs*}
⟨*proof*⟩

lemma *alluopairs-set*:
⟦*x*∈ *set xs* ; *y* ∈ *set xs*⟧ ⟹ (*x*,*y*) ∈ *set* (*alluopairs xs*) ∨ (*y*,*x*) ∈ *set* (*alluopairs xs*)
⟨*proof*⟩

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*⟩

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

lemma *iupt-set*: *set* (*iupt*(*i*,*j*)) = {*i* .. *j*}
⟨*proof*⟩

lemma *nth-pos2*: *0* < *n* ⟹ (*x*#*xs*) ! *n* = *xs* ! (*n* − *1*)
⟨*proof*⟩

lemma *myl*: ∀ (*a*::'*a*::{*pordered-ab-group-add*}) (*b*::'*a*). (*a* ≤ *b*) = (*0* ≤ *b* − *a*)
⟨*proof*⟩

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

⟨*proof*⟩

lemma *myeq*: ∀ (*a*::'*a*::{*pordered-ab-group-add*}) (*b*::'*a*). (*a* = *b*) = (*0* = *b* − *a*)
⟨*proof*⟩

lemma *floor-int-eq*: (*real n*≤ *x* ∧ *x* < *real* (*n*+*1*)) = (*floor x* = *n*)
⟨*proof*⟩

lemma *dvd-period*:
assumes *advdd*: (*a*::*int*) *dvd d*
shows (*a dvd* (*x* + *t*)) = (*a dvd* ((*x*+ *c*∗*d*) + *t*))
⟨*proof*⟩

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

lemma *int-rdvd-real*:
shows *real* (*i*::*int*) *rdvd x* = (*i dvd* (*floor x*) ∧ *real* (*floor x*) = *x*) (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *int-rdvd-iff*: (*real* (*i*::*int*) *rdvd real t*) = (*i dvd t*)
⟨*proof*⟩

lemma *rdvd-abs1*:

$(abs\ (real\ d)\ rdvd\ t) = (real\ (d::int)\ rdvd\ t)$
⟨*proof*⟩

lemma *rdvd-minus*: $(real\ (d::int)\ rdvd\ t) = (real\ d\ rdvd\ -t)$
⟨*proof*⟩

lemma *rdvd-left-0-eq*: $(0\ rdvd\ t) = (t=0)$
⟨*proof*⟩

lemma *rdvd-mult*:
assumes *knz*: $k\neq 0$
shows $(real\ (n::int) * real\ (k::int)\ rdvd\ x * real\ k) = (real\ n\ rdvd\ x)$
⟨*proof*⟩

lemma *rdvd-trans*: **assumes** *mn*:$m\ rdvd\ n$ **and** *nk*:$n\ rdvd\ k$
shows $m\ rdvd\ k$
⟨*proof*⟩

datatype $num = C\ int \mid Bound\ nat \mid CN\ nat\ int\ num \mid Neg\ num \mid Add\ num\ num \mid$
$Sub\ num\ num$
$\mid Mul\ int\ num \mid Floor\ num \mid CF\ int\ num\ num$

primrec *num-size* :: $num \Rightarrow nat$ **where**
$num\text{-}size\ (C\ c) = 1$
$\mid num\text{-}size\ (Bound\ n) = 1$
$\mid num\text{-}size\ (Neg\ a) = 1 + num\text{-}size\ a$
$\mid num\text{-}size\ (Add\ a\ b) = 1 + num\text{-}size\ a + num\text{-}size\ b$
$\mid num\text{-}size\ (Sub\ a\ b) = 3 + num\text{-}size\ a + num\text{-}size\ b$
$\mid num\text{-}size\ (CN\ n\ c\ a) = 4 + num\text{-}size\ a$
$\mid num\text{-}size\ (CF\ c\ a\ b) = 4 + num\text{-}size\ a + num\text{-}size\ b$
$\mid num\text{-}size\ (Mul\ c\ a) = 1 + num\text{-}size\ a$
$\mid num\text{-}size\ (Floor\ a) = 1 + num\text{-}size\ a$

primrec *Inum* :: $real\ list \Rightarrow num \Rightarrow real$ **where**
$Inum\ bs\ (C\ c) = (real\ c)$
$\mid Inum\ bs\ (Bound\ n) = bs!n$
$\mid Inum\ bs\ (CN\ n\ c\ a) = (real\ c) * (bs!n) + (Inum\ bs\ a)$
$\mid Inum\ bs\ (Neg\ a) = -(Inum\ bs\ a)$
$\mid Inum\ bs\ (Add\ a\ b) = Inum\ bs\ a + Inum\ bs\ b$
$\mid Inum\ bs\ (Sub\ a\ b) = Inum\ bs\ a - Inum\ bs\ b$
$\mid Inum\ bs\ (Mul\ c\ a) = (real\ c) * Inum\ bs\ a$
$\mid Inum\ bs\ (Floor\ a) = real\ (floor\ (Inum\ bs\ a))$
$\mid Inum\ bs\ (CF\ c\ a\ b) = real\ c * real\ (floor\ (Inum\ bs\ a)) + Inum\ bs\ b$

definition *isint t bs* ≡ *real* (*floor* (*Inum bs t*)) = *Inum bs t*

lemma *isint-iff*: *isint n bs* = (*real* (*floor* (*Inum bs n*)) = *Inum bs n*)
⟨*proof*⟩

lemma *isint-Floor*: *isint* (*Floor n*) *bs*
⟨*proof*⟩

lemma *isint-Mul*: *isint e bs* ⟹ *isint* (*Mul c e*) *bs*
⟨*proof*⟩

lemma *isint-neg*: *isint e bs* ⟹ *isint* (*Neg e*) *bs*
⟨*proof*⟩

lemma *isint-sub*:
assumes *ie*: *isint e bs* **shows** *isint* (*Sub* (*C c*) *e*) *bs*
⟨*proof*⟩

lemma *isint-add*: **assumes**
ai:*isint a bs* **and** *bi*: *isint b bs* **shows** *isint* (*Add a b*) *bs*
⟨*proof*⟩

lemma *isint-c*: *isint* (*C j*) *bs*
⟨*proof*⟩

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* ⇒ *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*
⟨*proof*⟩

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

consts *prep* :: *fm* ⇒ *fm*
recdef *prep measure fmsize*
  prep (*E T*) = *T*
  prep (*E F*) = *F*
  prep (*E* (*Or p q*)) = *Or* (*prep* (*E p*)) (*prep* (*E q*))
  prep (*E* (*Imp p q*)) = *Or* (*prep* (*E* (*NOT p*))) (*prep* (*E q*))
  prep (*E* (*Iff p q*)) = *Or* (*prep* (*E* (*And p q*))) (*prep* (*E* (*And* (*NOT p*) (*NOT q*))))
  prep (*E* (*NOT* (*And p q*))) = *Or* (*prep* (*E* (*NOT p*))) (*prep* (*E*(*NOT q*)))
  prep (*E* (*NOT* (*Imp p q*))) = *prep* (*E* (*And p* (*NOT q*)))
  prep (*E* (*NOT* (*Iff p q*))) = *Or* (*prep* (*E* (*And p* (*NOT q*)))) (*prep* (*E*(*And* (*NOT p*) *q*)))
  prep (*E p*) = *E* (*prep p*)
  prep (*A* (*And p q*)) = *And* (*prep* (*A p*)) (*prep* (*A q*))
  prep (*A p*) = *prep* (*NOT* (*E* (*NOT p*)))
  prep (*NOT* (*NOT p*)) = *prep p*
  prep (*NOT* (*And p q*)) = *Or* (*prep* (*NOT p*)) (*prep* (*NOT q*))
  prep (*NOT* (*A p*)) = *prep* (*E* (*NOT p*))
  prep (*NOT* (*Or p q*)) = *And* (*prep* (*NOT p*)) (*prep* (*NOT q*))
  prep (*NOT* (*Imp p q*)) = *And* (*prep p*) (*prep* (*NOT q*))
  prep (*NOT* (*Iff p q*)) = *Or* (*prep* (*And p* (*NOT q*))) (*prep* (*And* (*NOT p*) *q*))
  prep (*NOT p*) = *NOT* (*prep p*)
  prep (*Or p q*) = *Or* (*prep p*) (*prep q*)
  prep (*And p q*) = *And* (*prep p*) (*prep q*)
  prep (*Imp p q*) = *prep* (*Or* (*NOT p*) *q*)
  prep (*Iff p q*) = *Or* (*prep* (*And p q*)) (*prep* (*And* (*NOT p*) (*NOT q*)))
  prep p = *p*
(**hints** *simp add*: *fmsize-pos*)
lemma *prep*: ⋀ *bs*. *Ifm bs* (*prep p*) = *Ifm bs p*

⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *numbound0-gen*:
  assumes *nb*: *numbound0 t* **and** *ti*: *isint t* (*x*#*bs*)
  shows ∀ *y*. *isint t* (*y*#*bs*)
⟨*proof*⟩

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*)

| *bound0 (Or p q) = (bound0 p ∧ bound0 q)*
| *bound0 (Imp p q) = ((bound0 p) ∧ (bound0 q))*
| *bound0 (Iff p q) = (bound0 p ∧ bound0 q)*
| *bound0 (E p) = False*
| *bound0 (A p) = False*

lemma *bound0-I*:
assumes *bp*: *bound0 p*
shows *Ifm (b#bs) p = Ifm (b'#bs) p*
⟨*proof*⟩

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

lemma *numsubst0-I*:
shows *Inum (b#bs) (numsubst0 a t) = Inum ((Inum (b#bs) a)#bs) t*
⟨*proof*⟩

lemma *numsubst0-I'*:
assumes *nb*: *numbound0 a*
shows *Inum (b#bs) (numsubst0 a t) = Inum ((Inum (b'#bs) a)#bs) t*
⟨*proof*⟩

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

lemma *subst0-I*: **assumes** *qfp*: *qfree p*
shows *Ifm* (*b*#*bs*) (*subst0 a p*) = *Ifm* ((*Inum* (*b*#*bs*) *a*)#*bs*) *p*
⟨*proof*⟩

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

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

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

lemma *decrnum*: **assumes** *nb*: *numbound0 t*
shows *Inum* (*x*#*bs*) *t* = *Inum bs* (*decrnum t*)
⟨*proof*⟩

lemma *decr*: **assumes** *nb*: *bound0 p*
shows *Ifm* (*x*#*bs*) *p* = *Ifm bs* (*decr p*)
⟨*proof*⟩

lemma *decr-qf*: *bound0 p* ⟹ *qfree* (*decr p*)
⟨*proof*⟩

consts
isatom :: *fm* ⇒ *bool*
recdef *isatom measure size*

isatom T = True
isatom F = True
isatom (Lt a) = True
isatom (Le a) = True
isatom (Gt a) = True
isatom (Ge a) = True
isatom (Eq a) = True
isatom (NEq a) = True
isatom (Dvd i b) = True
isatom (NDvd i b) = True
isatom p = False

lemma *numsubst0-numbound0*: **assumes** *nb*: *numbound0 t*
shows *numbound0 (numsubst0 t a)*
⟨*proof*⟩

lemma *subst0-bound0*: **assumes** *qf*: *qfree p* **and** *nb*: *numbound0 t*
shows *bound0 (subst0 t p)*
⟨*proof*⟩

lemma *bound0-qf*: *bound0 p ⟹ qfree p*
⟨*proof*⟩

definition *djf*:: *('a ⇒ fm) ⇒ 'a ⇒ fm ⇒ 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 ⇒ T | F ⇒ q | - ⇒ Or fp q))

definition *evaldjf*:: *('a ⇒ fm) ⇒ 'a list ⇒ 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)*
⟨*proof*⟩

lemma *evaldjf-ex*: *Ifm bs (evaldjf f ps) = (∃ p ∈ set ps. Ifm bs (f p))*
⟨*proof*⟩

lemma *evaldjf-bound0*:
assumes *nb*: *∀ x∈ set xs. bound0 (f x)*
shows *bound0 (evaldjf f xs)*
⟨*proof*⟩

lemma *evaldjf-qf*:
assumes *nb*: *∀ x∈ set xs. qfree (f x)*
shows *qfree (evaldjf f xs)*
⟨*proof*⟩

consts
disjuncts :: *fm ⇒ fm list*

conjuncts :: *fm* ⇒ *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*: (∃ *q*∈ *set* (*disjuncts p*). *Ifm bs q*) = *Ifm bs p*
⟨*proof*⟩
lemma *conjuncts*: (∀ *q*∈ *set* (*conjuncts p*). *Ifm bs q*) = *Ifm bs p*
⟨*proof*⟩

lemma *disjuncts-nb*: *bound0 p* ⟹ ∀ *q*∈ *set* (*disjuncts p*). *bound0 q*
⟨*proof*⟩
lemma *conjuncts-nb*: *bound0 p* ⟹ ∀ *q*∈ *set* (*conjuncts p*). *bound0 q*
⟨*proof*⟩

lemma *disjuncts-qf*: *qfree p* ⟹ ∀ *q*∈ *set* (*disjuncts p*). *qfree q*
⟨*proof*⟩
lemma *conjuncts-qf*: *qfree p* ⟹ ∀ *q*∈ *set* (*conjuncts p*). *qfree q*
⟨*proof*⟩

constdefs *DJ* :: (*fm* ⇒ *fm*) ⇒ *fm* ⇒ *fm*
DJ f p ≡ *evaldjf f* (*disjuncts p*)

lemma *DJ*: **assumes** *fdj*: ∀ *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*⟩

lemma *DJ-qf*: **assumes**
fqf: ∀ *p*. *qfree p* ⟶ *qfree* (*f p*)
shows ∀ *p*. *qfree p* ⟶ *qfree* (*DJ f p*)
⟨*proof*⟩

lemma *DJ-qe*: **assumes** *qe*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ∀ *bs p*. *qfree p* ⟶ *qfree* (*DJ qe p*) ∧ (*Ifm bs* ((*DJ qe p*)) = *Ifm bs* (*E p*))
⟨*proof*⟩

consts *bnds*:: *num* ⇒ *nat list*
lex-ns:: *nat list* × *nat list* ⇒ *bool*
recdef *bnds measure size*

bnds (Bound n) = [n]
bnds (CN n c a) = n#(bnds a)
bnds (Neg a) = bnds a
bnds (Add a b) = (bnds a)@(bnds b)
bnds (Sub a b) = (bnds a)@(bnds b)
bnds (Mul i a) = bnds a
bnds (Floor a) = bnds a
bnds (CF c a b) = (bnds a)@(bnds b)
bnds a = []

recdef *lex-ns measure* (λ (*xs*,*ys*). *length xs* + *length ys*)
lex-ns ([], *ms*) = *True*
lex-ns (*ns*, []) = *False*
lex-ns (*n*#*ns*, *m*#*ms*) = (*n*<*m* ∨ ((*n* = *m*) ∧ *lex-ns* (*ns*,*ms*)))

constdefs *lex-bnd* :: *num* ⇒ *num* ⇒ *bool*
lex-bnd t s ≡ *lex-ns* (*bnds t*, *bnds s*)

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

consts *maxcoeff*:: *num* ⇒ *int*

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

lemma *maxcoeff-pos*: *maxcoeff t* ≥ *0*
⟨*proof*⟩

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

definition
numgcd :: *num* ⇒ *int*
where
numgcd-def: *numgcd t* = *numgcdh t* (*maxcoeff t*)

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

definition
reducecoeff :: *num* ⇒ *num*

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

recdef *dvdnumcoeff measure size*
dvdnumcoeff (*C i*) = (λ *g*. *g dvd i*)
dvdnumcoeff (*CN n c t*) = (λ *g*. *g dvd c* ∧ (*dvdnumcoeff t g*))
dvdnumcoeff (*CF c s 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*
⟨*proof*⟩

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

lemma *natabs0*: (*nat* (*abs x*) = *0*) = (*x* = *0*)
⟨*proof*⟩

lemma *numgcd0*:
assumes *g0*: *numgcd t* = *0*
shows *Inum bs t* = *0*
⟨*proof*⟩

lemma *numgcdh-pos*: **assumes** *gp*: *g* ≥ *0* **shows** *numgcdh t g* ≥ *0*
⟨*proof*⟩

lemma *numgcd-pos*: *numgcd t* ≥*0*
⟨*proof*⟩

lemma *reducecoeffh*:
assumes *gt*: *dvdnumcoeff t g* **and** *gp*: *g* > *0*
shows *real g* ∗(*Inum bs* (*reducecoeffh t g*)) = *Inum bs t*
⟨*proof*⟩
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′*
⟨*proof*⟩

lemma *maxcoeff-ismaxcoeff*: *ismaxcoeff t* (*maxcoeff t*)
⟨*proof*⟩

lemma *igcd-gt1*: *igcd i j > 1* $\Longrightarrow$ *((abs i > 1* $\wedge$ *abs j > 1)* $\vee$ *(abs i = 0* $\wedge$ *abs j > 1)* $\vee$ *(abs i > 1* $\wedge$ *abs j = 0))*
⟨*proof*⟩
lemma *numgcdh0*:*numgcdh t m = 0* $\Longrightarrow$ *m =0*
⟨*proof*⟩

lemma *dvdnumcoeff-aux*:
assumes *ismaxcoeff t m* **and** *mp*:*m* $\geq$ *0* **and** *numgcdh t m > 1*
shows *dvdnumcoeff t (numgcdh t m)*
⟨*proof*⟩

lemma *dvdnumcoeff-aux2*:
assumes *numgcd t > 1* **shows** *dvdnumcoeff t (numgcd t)* $\wedge$ *numgcd t > 0*
⟨*proof*⟩

lemma *reducecoeff*: *real (numgcd t)* $*$ *(Inum bs (reducecoeff t)) = Inum bs t*
⟨*proof*⟩

lemma *reducecoeffh-numbound0*: *numbound0 t* $\Longrightarrow$ *numbound0 (reducecoeffh t g)*
⟨*proof*⟩

lemma *reducecoeff-numbound0*: *numbound0 t* $\Longrightarrow$ *numbound0 (reducecoeff t)*
⟨*proof*⟩

consts
simpnum:: *num* $\Rightarrow$ *num*
numadd:: *num* $\times$ *num* $\Rightarrow$ *num*
nummul:: *num* $\Rightarrow$ *int* $\Rightarrow$ *num*

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

lemma *numadd*[*simp*]: *Inum bs (numadd (t,s)) = Inum bs (Add t s)*

⟨*proof*⟩

lemma *numadd-nb*[*simp*]: ⟦ *numbound0 t* ; *numbound0 s*⟧ ⟹ *numbound0* (*numadd* (*t*,*s*))
⟨*proof*⟩

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

lemma *nummul*[*simp*]: ⋀ *i*. *Inum bs* (*nummul t i*) = *Inum bs* (*Mul i t*)
⟨*proof*⟩

lemma *nummul-nb*[*simp*]: ⋀ *i*. *numbound0 t* ⟹ *numbound0* (*nummul t i*)
⟨*proof*⟩

constdefs *numneg* :: *num* ⇒ *num*
  numneg t ≡ *nummul t* (− *1*)

constdefs *numsub* :: *num* ⇒ *num* ⇒ *num*
  numsub s t ≡ (*if s* = *t then C 0 else numadd* (*s*,*numneg t*))

lemma *numneg*[*simp*]: *Inum bs* (*numneg t*) = *Inum bs* (*Neg t*)
⟨*proof*⟩

lemma *numneg-nb*[*simp*]: *numbound0 t* ⟹ *numbound0* (*numneg t*)
⟨*proof*⟩

lemma *numsub*[*simp*]: *Inum bs* (*numsub a b*) = *Inum bs* (*Sub a b*)
⟨*proof*⟩

lemma *numsub-nb*[*simp*]: ⟦ *numbound0 t* ; *numbound0 s*⟧ ⟹ *numbound0* (*numsub t s*)
⟨*proof*⟩

lemma *isint-CF*: **assumes** *si*: *isint s bs* **shows** *isint* (*CF c t s*) *bs*
⟨*proof*⟩

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

in (*bv*, *CF c a bi*))
split-int a = (*a*,*C 0*)

lemma *split-int*:⋀ *tv ti*. *split-int t* = (*tv*,*ti*) ⟹ (*Inum bs* (*Add tv ti*) = *Inum bs t*) ∧ *isint ti bs*
⟨*proof*⟩

lemma *split-int-nb*: *numbound0 t* ⟹ *numbound0* (*fst* (*split-int t*)) ∧ *numbound0* (*snd* (*split-int t*))
⟨*proof*⟩

definition
numfloor:: *num* ⇒ *num*
where
numfloor-def: *numfloor t* = (*let* (*tv*,*ti*) = *split-int t in*
(*case tv of C i* ⇒ *numadd* (*tv*,*ti*)
| - ⇒ *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*⟩

lemma *numfloor-nb*[*simp*]: *numbound0 t* ⟹ *numbound0* (*numfloor t*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *simpnum-numbound0*[*simp*]:
numbound0 t ⟹ *numbound0* (*simpnum t*)
⟨*proof*⟩

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

nozerocoeff t = True

lemma *numadd-nz* : *nozerocoeff a ⟹ nozerocoeff b ⟹ nozerocoeff* (*numadd* (*a*,*b*))
⟨*proof*⟩

lemma *nummul-nz* : ⋀ *i*. *i*≠*0 ⟹ nozerocoeff a ⟹ nozerocoeff* (*nummul a i*)
⟨*proof*⟩

lemma *numneg-nz* : *nozerocoeff a ⟹ nozerocoeff* (*numneg a*)
⟨*proof*⟩

lemma *numsub-nz*: *nozerocoeff a ⟹ nozerocoeff b ⟹ nozerocoeff* (*numsub a b*)
⟨*proof*⟩

lemma *split-int-nz*: *nozerocoeff t ⟹ nozerocoeff* (*fst* (*split-int t*)) ∧ *nozerocoeff* (*snd* (*split-int t*))
⟨*proof*⟩

lemma *numfloor-nz*: *nozerocoeff t ⟹ nozerocoeff* (*numfloor t*)
⟨*proof*⟩

lemma *simpnum-nz*: *nozerocoeff* (*simpnum t*)
⟨*proof*⟩

lemma *maxcoeff-nz*: *nozerocoeff t ⟹ maxcoeff t = 0 ⟹ t = C 0*
⟨*proof*⟩

lemma *numgcd-nz*: **assumes** *nz*: *nozerocoeff t* **and** *g0*: *numgcd t = 0* **shows** *t = C 0*
⟨*proof*⟩

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

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

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*⟩

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*)
⟨*proof*⟩
lemma *not-qf*[*simp*]: *qfree p* ⟹ *qfree* (*not p*)
⟨*proof*⟩
lemma *not-nb*[*simp*]: *bound0 p* ⟹ *bound0* (*not p*)
⟨*proof*⟩

constdefs *conj* :: *fm* ⇒ *fm* ⇒ *fm*
conj p q ≡ (*if* (*p* = *F* ∨ *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*)
⟨*proof*⟩

lemma *conj-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*conj p q*)
⟨*proof*⟩
lemma *conj-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*conj p q*)
⟨*proof*⟩

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

lemma *disj*[*simp*]: *Ifm bs* (*disj p q*) = *Ifm bs* (*Or p q*)
⟨*proof*⟩
lemma *disj-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*disj p q*)
⟨*proof*⟩
lemma *disj-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*disj p q*)
⟨*proof*⟩

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

not p
    else Imp p q)
lemma *imp*[*simp*]: *Ifm bs* (*imp p q*) = *Ifm bs* (*Imp p q*)
⟨*proof*⟩
lemma *imp-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*imp p q*)
⟨*proof*⟩
lemma *imp-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*imp p q*)
⟨*proof*⟩

constdefs *iff* :: *fm* ⇒ *fm* ⇒ *fm*
  iff p q ≡ (*if* (*p* = *q*) *then T else if* (*p* = *not q* ∨ *not p* = *q*) *then F else*
      if p=*F then not q else if q*=*F then not p else if p*=*T then q else if q*=*T then*
p else
  Iff p q)
lemma *iff*[*simp*]: *Ifm bs* (*iff p q*) = *Ifm bs* (*Iff p q*)
  ⟨*proof*⟩
lemma *iff-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*iff p q*)
  ⟨*proof*⟩
lemma *iff-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*iff p q*)
⟨*proof*⟩

consts *check-int*:: *num* ⇒ *bool*
recdef *check-int measure size*
  check-int (*C i*) = *True*
  check-int (*Floor t*) = *True*
  check-int (*Mul i t*) = *check-int t*
  check-int (*Add t s*) = (*check-int t* ∧ *check-int s*)
  check-int (*Neg t*) = *check-int t*
  check-int (*CF c t s*) = *check-int s*
  check-int t = *False*
lemma *check-int*: *check-int t* ⟹ *isint t bs*
⟨*proof*⟩

lemma *rdvd-left1-int*: *real* ⌊*t*⌋ = *t* ⟹ *1 rdvd t*
  ⟨*proof*⟩

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*⟩

constdefs *simpdvd*:: *int* ⇒ *num* ⇒ (*int* × *num*)
  simpdvd d t ≡
   (*let g* = *numgcd t in*
      if g > *1 then* (*let g*′ = *igcd d g in*
        if g′ = *1 then* (*d*, *t*)
        else (*d div g*′,*reducecoeffh t g*′))
      else (*d*, *t*))
lemma *simpdvd*:

assumes *tnz*: *nozerocoeff t* **and** *dnz*: $d \neq 0$
shows *Ifm bs* (*Dvd* (*fst* (*simpdvd d t*)) (*snd* (*simpdvd d t*))) = *Ifm bs* (*Dvd d t*)
⟨*proof*⟩

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

lemma *simpfm*[*simp*]: *Ifm bs* (*simpfm p*) = *Ifm bs p*
⟨*proof*⟩

lemma *simpdvd-numbound0*: *numbound0 t* ⟹ *numbound0* (*snd* (*simpdvd d t*))
⟨*proof*⟩

lemma *simpfm-bound0*[*simp*]: *bound0 p* ⟹ *bound0* (*simpfm p*)
⟨*proof*⟩

lemma *simpfm-qf*[*simp*]: *qfree p* ⟹ *qfree* (*simpfm p*)
⟨*proof*⟩

constdefs *list-conj* :: *fm list* ⇒ *fm*
list-conj ps ≡ *foldr conj ps T*
lemma *list-conj*: *Ifm bs* (*list-conj ps*) = (∀ *p*∈ *set ps*. *Ifm bs p*)
⟨*proof*⟩
lemma *list-conj-qf*: ∀ *p*∈ *set ps*. *qfree p* ⟹ *qfree* (*list-conj ps*)
⟨*proof*⟩
lemma *list-conj-nb*: ∀ *p*∈ *set ps*. *bound0 p* ⟹ *bound0* (*list-conj ps*)
⟨*proof*⟩
constdefs *CJNB*:: (*fm* ⇒ *fm*) ⇒ *fm* ⇒ *fm*
CJNB f p ≡ (*let cjs* = *conjuncts p* ; (*yes*,*no*) = *partition bound0 cjs*
in conj (*decr* (*list-conj yes*)) (*f* (*list-conj no*)))

lemma *CJNB-qe*:
assumes *qe*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ∀ *bs p*. *qfree p* ⟶ *qfree* (*CJNB qe p*) ∧ (*Ifm bs* ((*CJNB qe p*)) = *Ifm bs* (*E p*))
⟨*proof*⟩

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

lemma *qelim-ci*:
assumes *qe-inv*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ⋀ *bs*. *qfree* (*qelim p qe*) ∧ (*Ifm bs* (*qelim p qe*) = *Ifm bs p*)
⟨*proof*⟩

The ℤ Part

Linearity for fm where Bound 0 ranges over ℤ

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

$(ib,b') = \ zsplit0\ b$
$in\ (ia-ib,\ Sub\ a'\ b'))$
$zsplit0\ (Mul\ i\ a) = (let\ (i',a') = \ zsplit0\ a\ in\ (i*i',\ Mul\ i\ a'))$
$zsplit0\ (Floor\ a) = (let\ (i',a') = \ zsplit0\ a\ in\ (i',Floor\ a'))$
(**hints** *simp add*: *Let-def*)

lemma *zsplit0-I*:
shows $\bigwedge\ n\ a.\ zsplit0\ t = (n,a) \Longrightarrow (Inum\ ((real\ (x::int))\ \#bs)\ (CN\ 0\ n\ a) = Inum\ (real\ x\ \#bs)\ t) \wedge numbound0\ a$
(**is** $\bigwedge\ n\ a.\ ?S\ t = (n,a) \Longrightarrow (?I\ x\ (CN\ 0\ n\ a) = ?I\ x\ t) \wedge ?N\ a$)
⟨*proof*⟩

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

lemma *zlin-qfree*: $iszlfm\ p\ bs \Longrightarrow qfree\ p$
⟨*proof*⟩

lemma *iszlfm-gen*:
assumes *lp*: $iszlfm\ p\ (x\#bs)$
shows $\forall\ y.\ iszlfm\ p\ (y\#bs)$
⟨*proof*⟩

lemma *conj-zl*[*simp*]: $iszlfm\ p\ bs \Longrightarrow iszlfm\ q\ bs \Longrightarrow iszlfm\ (conj\ p\ q)\ bs$
⟨*proof*⟩
lemma *disj-zl*[*simp*]: $iszlfm\ p\ bs \Longrightarrow iszlfm\ q\ bs \Longrightarrow iszlfm\ (disj\ p\ q)\ bs$
⟨*proof*⟩
lemma *not-zl*[*simp*]: $iszlfm\ p\ bs \Longrightarrow iszlfm\ (not\ p)\ bs$
⟨*proof*⟩

recdef *zlfm measure fmsize*
$zlfm\ (And\ p\ q) = conj\ (zlfm\ p)\ (zlfm\ q)$
$zlfm\ (Or\ p\ q) = disj\ (zlfm\ p)\ (zlfm\ q)$
$zlfm\ (Imp\ p\ q) = disj\ (zlfm\ (NOT\ p))\ (zlfm\ q)$

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

$zlfm\ (NOT\ (Or\ p\ q)) = conj\ (zlfm\ (NOT\ p))\ (zlfm\ (NOT\ q))$
$zlfm\ (NOT\ (Imp\ p\ q)) = conj\ (zlfm\ p)\ (zlfm\ (NOT\ q))$
$zlfm\ (NOT\ (Iff\ p\ q)) = disj\ (conj(zlfm\ p)\ (zlfm(NOT\ q)))\ (conj\ (zlfm(NOT\ p))$
$(zlfm\ q))$
$zlfm\ (NOT\ (NOT\ p)) = zlfm\ p$
$zlfm\ (NOT\ T) = F$
$zlfm\ (NOT\ F) = T$
$zlfm\ (NOT\ (Lt\ a)) = zlfm\ (Ge\ a)$
$zlfm\ (NOT\ (Le\ a)) = zlfm\ (Gt\ a)$
$zlfm\ (NOT\ (Gt\ a)) = zlfm\ (Le\ a)$
$zlfm\ (NOT\ (Ge\ a)) = zlfm\ (Lt\ a)$
$zlfm\ (NOT\ (Eq\ a)) = zlfm\ (NEq\ a)$
$zlfm\ (NOT\ (NEq\ a)) = zlfm\ (Eq\ a)$
$zlfm\ (NOT\ (Dvd\ i\ a)) = zlfm\ (NDvd\ i\ a)$
$zlfm\ (NOT\ (NDvd\ i\ a)) = zlfm\ (Dvd\ i\ a)$
$zlfm\ p = p$ (**hints** *simp add*: *fmsize-pos*)

lemma *split-int-less-real*:
$(real\ (a::int) < b) = (a < floor\ b \vee (a = floor\ b \wedge real\ (floor\ b) < b))$
⟨*proof*⟩

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*⟩

lemma *split-int-gt-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*⟩

lemma *split-int-le-real*:
$(real\ (a::int) \leq b) = (a \leq floor\ b \vee (a = floor\ b \wedge real\ (floor\ b) < b))$
⟨*proof*⟩

lemma *split-int-le-real'*:
$(real\ (a::int) + b \leq 0) = (real\ a - real\ (floor(-b)) \leq 0 \vee (real\ a - real\ (floor$
$(-b)) = 0 \wedge real\ (floor\ (-b)) + b < 0))$
⟨*proof*⟩

lemma *split-int-ge-real'*:
$(real\ (a::int) + b \geq 0) = (real\ a + real\ (floor\ b) \geq 0 \vee (real\ a + real\ (floor\ b)$
$= 0 \wedge real\ (floor\ b) - b < 0))$
⟨*proof*⟩

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

lemma *split-int-eq-real'*: $(real\ (a::int) + b = 0) = (\ a - floor\ (-b) = 0 \wedge real\ (floor\ (-b)) + b = 0)$ (**is** *?l = ?r*)
⟨*proof*⟩

lemma *zlfm-I*:
assumes *qfp*: *qfree p*
shows $(Ifm\ (real\ i\ \#bs)\ (zlfm\ p) = Ifm\ (real\ i\#\ bs)\ p) \wedge iszlfm\ (zlfm\ p)\ (real\ (i::int)\ \#bs)$
(**is** $(?I\ (?l\ p) = ?I\ p) \wedge ?L\ (?l\ p)$)
⟨*proof*⟩

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

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

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

lemma *minusinf-qfree*: *qfree p* ⟹ *qfree* (*minusinf p*)
⟨*proof*⟩

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

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

δ (*NDvd i* (*CN 0 c e*)) = *i*
δ *p* = *1*

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

lemma *delta-mono*:
assumes *lin*: *iszlfm p bs*
and *d*: *d dvd d'*
and *ad*: *d*δ *p d*
shows *d*δ *p d'*
⟨*proof*⟩

lemma δ : **assumes** *lin*:*iszlfm p bs*
shows *d*δ *p* (δ *p*) $\wedge$ δ *p* >*0*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *minusinf-repeats*:
assumes *d*: *d*δ *p d* **and** *linp*: *iszlfm p* (*a* # *bs*)
shows *Ifm* ((*real*(*x* − *k*∗*d*))#*bs*) (*minusinf p*) = *Ifm* (*real x* #*bs*) (*minusinf p*)
⟨*proof*⟩

lemma *minusinf-ex*:
assumes *lin*: *iszlfm p* (*real* (*a*::*int*) #*bs*)
and *exmi*: $\exists$ (*x*::*int*). *Ifm* (*real x*#*bs*) (*minusinf p*) (**is** $\exists$ *x*. *?P1 x*)
shows $\exists$ (*x*::*int*). *Ifm* (*real x*#*bs*) *p* (**is** $\exists$ *x*. *?P x*)
⟨*proof*⟩

lemma *minusinf-bex*:
assumes *lin*: *iszlfm p* (*real* (*a*::*int*) #*bs*)
shows ($\exists$ (*x*::*int*). *Ifm* (*real x*#*bs*) (*minusinf p*)) =
($\exists$ (*x*::*int*)∈ {*1*..δ *p*}. *Ifm* (*real x*#*bs*) (*minusinf p*))
(**is** ($\exists$ *x*. *?P x*) = -)
⟨*proof*⟩

lemma *dvd1-eq1*: *x* >*0* $\Longrightarrow$ (*x*::*int*) *dvd 1* = (*x* = *1*) ⟨*proof*⟩

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 $(Inum\ (real\ (i::int)\#bs))\ `\ set\ (\alpha\ p) = (Inum\ (real\ i\#bs))\ `\ set\ (\beta\ (mirror\ p))$
⟨*proof*⟩

lemma *mirror*:
assumes *lp*: *iszlfm p* $(a\#bs)$
shows *Ifm* $(real\ (x::int)\#bs)\ (mirror\ p) = Ifm\ (real\ (-\ x)\#bs)\ p$
⟨*proof*⟩

lemma *mirror-l*: *iszlfm p* $(a\#bs) \Longrightarrow$ *iszlfm* $(mirror\ p)\ (a\#bs)$
⟨*proof*⟩

lemma *mirror-d*β: *iszlfm p* $(a\#bs) \wedge d\beta\ p\ 1$
$\Longrightarrow$ *iszlfm* $(mirror\ p)\ (a\#bs) \wedge d\beta\ (mirror\ p)\ 1$
⟨*proof*⟩

lemma *mirror-δ*: *iszlfm p (a#bs) ⟹ δ (mirror p) = δ p*
⟨*proof*⟩

lemma *mirror-ex*:
assumes *lp*: *iszlfm p (real (i::int)#bs)*
shows *(∃ (x::int). Ifm (real x#bs) (mirror p)) = (∃ (x::int). Ifm (real x#bs) p)*
(is *(∃ x. ?I x ?mp) = (∃ x. ?I x p))*
⟨*proof*⟩

lemma *β-numbound0*: **assumes** *lp*: *iszlfm p bs*
shows *∀ b∈ set (β p). numbound0 b*
⟨*proof*⟩

lemma *dβ-mono*:
assumes *linp*: *iszlfm p (a #bs)*
and *dr*: *dβ p l*
and *d*: *l dvd l′*
shows *dβ p l′*
⟨*proof*⟩

lemma *α-l*: **assumes** *lp*: *iszlfm p (a#bs)*
shows *∀ b∈ set (α p). numbound0 b ∧ isint b (a#bs)*
⟨*proof*⟩

lemma *ζ*:
assumes *linp*: *iszlfm p (a #bs)*
shows *ζ p > 0 ∧ dβ p (ζ p)*
⟨*proof*⟩

lemma *aβ*: **assumes** *linp*: *iszlfm p (a #bs)* **and** *d*: *dβ p l* **and** *lp*: *l > 0*
shows *iszlfm (aβ p l) (a #bs) ∧ dβ (aβ p l) 1 ∧ (Ifm (real (l ∗ x) #bs) (aβ p l) = Ifm ((real x)#bs) p)*
⟨*proof*⟩

lemma *aβ-ex*: **assumes** *linp*: *iszlfm p (a#bs)* **and** *d*: *dβ p l* **and** *lp*: *l>0*
shows *(∃ x. l dvd x ∧ Ifm (real x #bs) (aβ p l)) = (∃ (x::int). Ifm (real x#bs) p)*
(is *(∃ x. l dvd x ∧ ?P x) = (∃ x. ?P′ x))*
⟨*proof*⟩

lemma *β*:
assumes *lp*: *iszlfm p (a#bs)*
and *u*: *dβ p 1*
and *d*: *dδ p d*
and *dp*: *d > 0*
and *nob*: *¬(∃(j::int) ∈ {1 .. d}. ∃ b∈ (Inum (a#bs)) ‘ set(β p). real x = b +*

real j)
and *p*: *Ifm* (*real x*#*bs*) *p* (**is** *?P x*)
shows *?P* (*x* − *d*)
⟨*proof*⟩

lemma *β′*:
assumes *lp*: *iszlfm p* (*a* #*bs*)
and *u*: *dβ p 1*
and *d*: *dδ p d*
and *dp*: *d* > *0*
shows ∀ *x*. ¬(∃ (*j*::*int*) ∈ {*1* .. *d*}. ∃ *b*∈ *set*(*β p*). *Ifm* ((*Inum* (*a*#*bs*) *b* + *real j*) #*bs*) *p*) ⟶ *Ifm* (*real x*#*bs*) *p* ⟶ *Ifm* (*real* (*x* − *d*)#*bs*) *p* (**is** ∀ *x*. *?b* ⟶ *?P x* ⟶ *?P* (*x* − *d*))
⟨*proof*⟩

lemma *β-int*: **assumes** *lp*: *iszlfm p bs*
shows ∀ *b*∈ *set* (*β p*). *isint b bs*
⟨*proof*⟩

lemma *cpmi-eq*: *0* < *D* ⟹ (*EX z*::*int*. *ALL x*. *x* < *z* −−> (*P x* = *P1 x*))
==> *ALL x*.~(*EX* (*j*::*int*) : {*1*..*D*}. *EX* (*b*::*int*) : *B*. *P*(*b*+*j*)) −−> *P* (*x*) −−> *P* (*x* − *D*)
==> (*ALL* (*x*::*int*). *ALL* (*k*::*int*). ((*P1 x*)= (*P1* (*x*−*k*∗*D*))))
==> (*EX* (*x*::*int*). *P*(*x*)) = ((*EX* (*j*::*int*) : {*1*..*D*} . (*P1*(*j*))) | (*EX* (*j*::*int*) : {*1*..*D*}. *EX* (*b*::*int*) : *B*. *P* (*b*+*j*)))
⟨*proof*⟩

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*⟩

consts
ϱ :: *fm* ⇒ (*num* × *int*) *list*
σϱ:: *fm* ⇒ *num* × *int* ⇒ *fm*
αϱ :: *fm* ⇒ (*num*×*int*) *list*
aϱ :: *fm* ⇒ *int* ⇒ *fm*
recdef *ϱ measure size*
ϱ (*And p q*) = (*ϱ p* @ *ϱ q*)

```
  ϱ (Or p q) = (ϱ p @ ϱ q)
  ϱ (Eq  (CN 0 c e)) = [(Sub (C −1) e,c)]
  ϱ (NEq (CN 0 c e)) = [(Neg e,c)]
  ϱ (Lt  (CN 0 c e)) = []
  ϱ (Le  (CN 0 c e)) = []
  ϱ (Gt  (CN 0 c e)) = [(Neg e, c)]
  ϱ (Ge  (CN 0 c e)) = [(Sub (C (−1)) e, c)]
  ϱ p = []

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

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

recdef aϱ measure size
  aϱ (And p q) = (λ k. And (aϱ p k) (aϱ q k))
  aϱ (Or p q) = (λ k. Or (aϱ p k) (aϱ q k))
  aϱ (Eq (CN 0 c e)) = (λ k. if k dvd c then (Eq (CN 0 (c div k) e))
                                   else (Eq (CN 0 c (Mul k e))))
```

$a\varrho\ (NEq\ (CN\ 0\ c\ e)) = (\lambda\ k.\ if\ k\ dvd\ c\ then\ (NEq\ (CN\ 0\ (c\ div\ k)\ e))$
$else\ (NEq\ (CN\ 0\ c\ (Mul\ k\ e))))$
$a\varrho\ (Lt\ (CN\ 0\ c\ e)) = (\lambda\ k.\ if\ k\ dvd\ c\ then\ (Lt\ (CN\ 0\ (c\ div\ k)\ e))$
$else\ (Lt\ (CN\ 0\ c\ (Mul\ k\ e))))$
$a\varrho\ (Le\ (CN\ 0\ c\ e)) = (\lambda\ k.\ if\ k\ dvd\ c\ then\ (Le\ (CN\ 0\ (c\ div\ k)\ e))$
$else\ (Le\ (CN\ 0\ c\ (Mul\ k\ e))))$
$a\varrho\ (Gt\ (CN\ 0\ c\ e)) = (\lambda\ k.\ if\ k\ dvd\ c\ then\ (Gt\ (CN\ 0\ (c\ div\ k)\ e))$
$else\ (Gt\ (CN\ 0\ c\ (Mul\ k\ e))))$
$a\varrho\ (Ge\ (CN\ 0\ c\ e)) = (\lambda\ k.\ if\ k\ dvd\ c\ then\ (Ge\ (CN\ 0\ (c\ div\ k)\ e))$
$else\ (Ge\ (CN\ 0\ c\ (Mul\ k\ e))))$
$a\varrho\ (Dvd\ i\ (CN\ 0\ c\ e)) = (\lambda\ k.\ if\ k\ dvd\ c\ then\ (Dvd\ i\ (CN\ 0\ (c\ div\ k)\ e))$
$else\ (Dvd\ (i{*}k)\ (CN\ 0\ c\ (Mul\ k\ e))))$
$a\varrho\ (NDvd\ i\ (CN\ 0\ c\ e)) = (\lambda\ k.\ if\ k\ dvd\ c\ then\ (NDvd\ i\ (CN\ 0\ (c\ div\ k)\ e))$
$else\ (NDvd\ (i{*}k)\ (CN\ 0\ c\ (Mul\ k\ e))))$
$a\varrho\ p = (\lambda\ k.\ p)$

constdefs $\sigma :: fm \Rightarrow int \Rightarrow num \Rightarrow fm$
$\sigma\ p\ k\ t \equiv And\ (Dvd\ k\ t)\ (\sigma\varrho\ p\ (t,k))$

lemma $\sigma\varrho$:
assumes *linp*: $iszlfm\ p\ (real\ (x::int)\#bs)$
and *kpos*: $real\ k > 0$
and *tnb*: $numbound0\ t$
and *tint*: $isint\ t\ (real\ x\#bs)$
and *kdt*: $k\ dvd\ floor\ (Inum\ (b'\#bs)\ t)$
shows $Ifm\ (real\ x\#bs)\ (\sigma\varrho\ p\ (t,k)) =$
$(Ifm\ ((real\ ((floor\ (Inum\ (b'\#bs)\ t))\ div\ k))\#bs)\ p)$
(**is** $?I\ (real\ x)\ (?s\ p) = (?I\ (real\ ((floor\ (?N\ b'\ t))\ div\ k))\ p)$ **is** - = $(?I\ ?tk\ p)$)
⟨*proof*⟩

lemma $a\varrho$:
assumes *lp*: $iszlfm\ p\ (real\ (x::int)\#bs)$ **and** *kp*: $real\ k > 0$
shows $Ifm\ (real\ (x{*}k)\#bs)\ (a\varrho\ p\ k) = Ifm\ (real\ x\#bs)\ p$ (**is** $?I\ (x{*}k)\ (?f\ p\ k) = ?I\ x\ p$)
⟨*proof*⟩

lemma $a\varrho$-*ex*:
assumes *lp*: $iszlfm\ p\ (real\ (x::int)\#bs)$ **and** *kp*: $k > 0$
shows $(\exists\ (x::int).\ real\ k\ rdvd\ real\ x \wedge Ifm\ (real\ x\#bs)\ (a\varrho\ p\ k)) =$
$(\exists\ (x::int).\ Ifm\ (real\ x\#bs)\ p)$ (**is** $(\exists\ x.\ ?D\ x \wedge ?P'\ x) = (\exists\ x.\ ?P\ x)$)
⟨*proof*⟩

lemma $\sigma\varrho'$: **assumes** *lp*: $iszlfm\ p\ (real\ (x::int)\#bs)$ **and** *kp*: $k > 0$ **and** *nb*: $numbound0\ t$
shows $Ifm\ (real\ x\#bs)\ (\sigma\varrho\ p\ (t,k)) = Ifm\ ((Inum\ (real\ x\#bs)\ t)\#bs)\ (a\varrho\ p\ k)$
⟨*proof*⟩

lemma $\sigma\varrho$-*nb*: **assumes** *lp*:$iszlfm\ p\ (a\#bs)$ **and** *nb*: $numbound0\ t$

shows *bound0* ($\sigma\varrho$ *p* (*t*,*k*))
⟨*proof*⟩

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

lemma $\alpha\varrho$-*l*:
assumes *lp*: *iszlfm p* (*real* (*i*::*int*)#*bs*)
shows ∀ (*b*,*k*) ∈ *set* ($\alpha\varrho$ *p*). *k* >*0* ∧ *numbound0 b* ∧ *isint b* (*real i*#*bs*)
⟨*proof*⟩

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

lemma σ-*And*: *Ifm bs* (σ (*And p q*) *k t*) = *Ifm bs* (*And* (σ *p k t*) (σ *q k t*))
⟨*proof*⟩
lemma σ-*Or*: *Ifm bs* (σ (*Or p q*) *k t*) = *Ifm bs* (*Or* (σ *p k t*) (σ *q k t*))
⟨*proof*⟩

lemma ϱ: **assumes** *lp*: *iszlfm p* (*real* (*i*::*int*) #*bs*)
and *pi*: *Ifm* (*real i*#*bs*) *p*
and *d*: *d*δ *p d*
and *dp*: *d* > *0*
and *nob*: ∀(*e*,*c*) ∈ *set* (ϱ *p*). ∀ *j*∈ {*1* .. *c*∗*d*}. *real* (*c*∗*i*) ≠ *Inum* (*real i*#*bs*) *e* + *real j*
(**is** ∀(*e*,*c*) ∈ *set* (ϱ *p*). ∀ *j*∈ {*1* .. *c*∗*d*}. - ≠ *?N i e* + -)
shows *Ifm* (*real*(*i* − *d*)#*bs*) *p*
⟨*proof*⟩

lemma σ-*nb*: **assumes** *lp*: *iszlfm p* (*a*#*bs*) **and** *nb*: *numbound0 t*
shows *bound0* (σ *p k t*)
⟨*proof*⟩

lemma ϱ': **assumes** *lp*: *iszlfm p* (*a* #*bs*)
and *d*: *d*δ *p d*
and *dp*: *d* > *0*
shows ∀ *x*. ¬(∃ (*e*,*c*) ∈ *set*(ϱ *p*). ∃(*j*::*int*) ∈ {*1* .. *c*∗*d*}. *Ifm* (*a* #*bs*) (σ *p c* (*Add e* (*C j*)))) ⟶ *Ifm* (*real x*#*bs*) *p* ⟶ *Ifm* (*real* (*x* − *d*)#*bs*) *p* (**is** ∀ *x*. *?b x* ⟶ *?P x* ⟶ *?P* (*x* − *d*))
⟨*proof*⟩

lemma *rl-thm*:
assumes *lp*: *iszlfm p (real (i::int)#bs)*
shows (∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) = ((∃ *j*∈ {*1* .. *δ p*}. *Ifm* (*real j*#*bs*) (*minusinf p*)) ∨ (∃ (*e*,*c*) ∈ *set* (*ϱ p*). ∃ *j*∈ {*1* .. *c*∗(*δ p*)}. *Ifm* (*a*#*bs*) (*σ p c* (*Add e* (*C j*)))))
(**is** (∃(*x*::*int*). *?P x*) = ((∃ *j*∈ {*1*.. *δ p*}. *?MP j*)∨(∃ (*e*,*c*) ∈ *?R*. ∃ *j*∈ -. *?SP c e j*))
is *?lhs* = (*?MD* ∨ *?RD*) **is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *mirror-αϱ*: **assumes** *lp*: *iszlfm p* (*a*#*bs*)
shows (λ (*t*,*k*). (*Inum* (*a*#*bs*) *t*, *k*)) ‘ *set* (*αϱ p*) = (λ (*t*,*k*). (*Inum* (*a*#*bs*) *t*,*k*)) ‘ *set* (*ϱ* (*mirror p*))
⟨*proof*⟩

The $\mathbb{R}$ part

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

consts
isrlfm :: *fm* ⇒ *bool*
recdef *isrlfm measure size*
isrlfm (*And p q*) = (*isrlfm p* ∧ *isrlfm q*)
isrlfm (*Or p q*) = (*isrlfm p* ∧ *isrlfm q*)
isrlfm (*Eq* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*NEq* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*Lt* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*Le* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*Gt* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm (*Ge* (*CN 0 c e*)) = (*c*>*0* ∧ *numbound0 e*)
isrlfm p = (*isatom p* ∧ (*bound0 p*))

constdefs *fp* :: *fm* ⇒ *int* ⇒ *num* ⇒ *int* ⇒ *fm*
fp p n s j ≡ (*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* ⇒ (*fm* × *int* × *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 (λ ((*p*,*n*,*t*),(*q*,*m*,*s*)). (*And p q*, *n*+*m*, *Add t s*)) [(*a*,*b*). *a*←*acs*,*b*←*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*

$(\lambda\ (p,n,s).\ \mathit{if}\ n{=}0\ \mathit{then}\ [(p,0,\mathit{Floor}\ s)]$
$\mathit{else}\ (\mathit{map}\ (\lambda\ j.\ (\mathit{fp}\ p\ n\ s\ j,\ 0,\ \mathit{Add}\ (\mathit{Floor}\ s)\ (C\ j)))\ (\mathit{if}\ n > 0\ \mathit{then}\ \mathit{iupt}\ (0,n)\ \mathit{else}\ \mathit{iupt}(n,0))))$
$(\mathit{rsplit0}\ a))$
$\mathit{rsplit0}\ (\mathit{CN}\ 0\ c\ a) = \mathit{map}\ (\lambda\ (p,n,s).\ (p,n{+}c,s))\ (\mathit{rsplit0}\ a)$
$\mathit{rsplit0}\ (\mathit{CN}\ m\ c\ a) = \mathit{map}\ (\lambda\ (p,n,s).\ (p,n,\mathit{CN}\ m\ c\ s))\ (\mathit{rsplit0}\ a)$
$\mathit{rsplit0}\ (\mathit{CF}\ c\ t\ s) = \mathit{rsplit0}\ (\mathit{Add}\ (\mathit{Mul}\ c\ (\mathit{Floor}\ t))\ s)$
$\mathit{rsplit0}\ (\mathit{Mul}\ c\ a) = \mathit{map}\ (\lambda\ (p,n,s).\ (p,c{*}n,\mathit{Mul}\ c\ s))\ (\mathit{rsplit0}\ a)$
$\mathit{rsplit0}\ t = [(T,0,t)]$

lemma *not-rl*[*simp*]: $\mathit{isrlfm}\ p \Longrightarrow \mathit{isrlfm}\ (\mathit{not}\ p)$
⟨*proof*⟩
lemma *conj-rl*[*simp*]: $\mathit{isrlfm}\ p \Longrightarrow \mathit{isrlfm}\ q \Longrightarrow \mathit{isrlfm}\ (\mathit{conj}\ p\ q)$
⟨*proof*⟩
lemma *disj-rl*[*simp*]: $\mathit{isrlfm}\ p \Longrightarrow \mathit{isrlfm}\ q \Longrightarrow \mathit{isrlfm}\ (\mathit{disj}\ p\ q)$
⟨*proof*⟩

lemma *rsplit0-cs*:
shows $\forall\ (p,n,s) \in \mathit{set}\ (\mathit{rsplit0}\ t).$
$(\mathit{Ifm}\ (x\#bs)\ p \longrightarrow (\mathit{Inum}\ (x\#bs)\ t = \mathit{Inum}\ (x\#bs)\ (\mathit{CN}\ 0\ n\ s))) \wedge \mathit{numbound0}\ s \wedge \mathit{isrlfm}\ p$
(**is** $\forall\ (p,n,s) \in ?SS\ t.\ (?I\ p \longrightarrow ?N\ t = ?N\ (\mathit{CN}\ 0\ n\ s)) \wedge \text{-} \wedge \text{-}$)
⟨*proof*⟩

lemma *real-in-int-intervals*:
assumes *xb*: $\mathit{real}\ m \leq x \wedge x < \mathit{real}\ ((n::\mathit{int}) + 1)$
shows $\exists\ j{\in}\ \{m..\ n\}.\ \mathit{real}\ j \leq x \wedge x < \mathit{real}\ (j{+}1)$ (**is** $\exists\ j{\in}\ ?N.\ ?P\ j$)
⟨*proof*⟩

lemma *rsplit0-complete*:
assumes $xp{:}0 \leq x$ **and** $x1{:}x < 1$
shows $\exists\ (p,n,s) \in \mathit{set}\ (\mathit{rsplit0}\ t).\ \mathit{Ifm}\ (x\#bs)\ p$ (**is** $\exists\ (p,n,s) \in ?SS\ t.\ ?I\ p$)
⟨*proof*⟩

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

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

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

lemma *foldr-disj-map-rlfm*:

assumes *lf*: $\forall\ n\ s.\ numbound0\ s \longrightarrow isrlfm\ (f\ n\ s)$
and φ: $\forall\ (\varphi,n,s) \in set\ xs.\ numbound0\ s \wedge isrlfm\ \varphi$
shows *isrlfm* (*foldr disj* (*map* ($\lambda\ (\varphi,\ n,\ s).\ conj\ \varphi\ (f\ n\ s)$) *xs*) *F*)
⟨*proof*⟩

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

lemma *rsplit-l*: **assumes** *lf*: $\forall\ n\ s.\ numbound0\ s \longrightarrow isrlfm\ (f\ n\ s)$
shows *isrlfm* (*rsplit f a*)
⟨*proof*⟩

lemma *rsplit*:
assumes *xp*: $x \geq 0$ **and** *x1*: $x < 1$
and *f*: $\forall\ a\ n\ s.\ Inum\ (x\#bs)\ a = Inum\ (x\#bs)\ (CN\ 0\ n\ s) \wedge numbound0\ s \longrightarrow (Ifm\ (x\#bs)\ (f\ n\ s) = Ifm\ (x\#bs)\ (g\ a))$
shows *Ifm* (*x*#*bs*) (*rsplit f a*) = *Ifm* (*x*#*bs*) (*g a*)
⟨*proof*⟩

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

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

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

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

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

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

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

lemma *lt-l*: *isrlfm* (*rsplit lt a*)
⟨*proof*⟩

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

lemma *le-l*: *isrlfm* (*rsplit le a*)
⟨*proof*⟩

lemma *gt-mono*: ∀ *a n s*. *Inum* (*x*#*bs*) *a* = *Inum* (*x*#*bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x*#*bs*) (*gt n s*) = *Ifm* (*x*#*bs*) (*Gt a*) (**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*gt n s*) = *?I* (*Gt a*))
⟨*proof*⟩
lemma *gt-l*: *isrlfm* (*rsplit gt a*)
⟨*proof*⟩

lemma *ge-mono*: ∀ *a n s*. *Inum* (*x*#*bs*) *a* = *Inum* (*x*#*bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x*#*bs*) (*ge n s*) = *Ifm* (*x*#*bs*) (*Ge a*) (**is** ∀ *a n s* . *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*ge n s*) = *?I* (*Ge a*))
⟨*proof*⟩
lemma *ge-l*: *isrlfm* (*rsplit ge a*)
⟨*proof*⟩

lemma *eq-mono*: ∀ *a n s*. *Inum* (*x*#*bs*) *a* = *Inum* (*x*#*bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x*#*bs*) (*eq n s*) = *Ifm* (*x*#*bs*) (*Eq a*) (**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*eq n s*) = *?I* (*Eq a*))
⟨*proof*⟩
lemma *eq-l*: *isrlfm* (*rsplit eq a*)
⟨*proof*⟩

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

lemma *neq-l*: *isrlfm* (*rsplit neq a*)
⟨*proof*⟩

lemma *small-le*:
assumes *u0*:*0* ≤ *u* **and** *u1*: *u* < *1*
shows (−*u* ≤ *real* (*n*::*int*)) = (*0* ≤ *n*)
⟨*proof*⟩

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

lemma *rdvd01-cs*:
assumes *up*: $u \geq 0$ **and** *u1*: $u<1$ **and** *np*: *real* $n > 0$
shows (*real* (i::*int*) *rdvd real* (n::*int*) $*$ $u - s$) = ($\exists$ $j \in \{0 .. n - 1\}$. *real* $n *$ $u = s -$ *real* (*floor* s) + *real* $j \wedge$ *real* i *rdvd real* ($j -$ *floor* s)) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

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

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

lemma *DVDJ-DVD*:
assumes *xp*:$x \geq 0$ **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*⟩

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*⟩

lemma *foldr-disj-map-rlfm2*:
assumes *lf*: ∀ *n* . *isrlfm* (*f n*)
shows *isrlfm* (*foldr disj* (*map f xs*) *F*)
⟨*proof*⟩
lemma *foldr-And-map-rlfm2*:
assumes *lf*: ∀ *n* . *isrlfm* (*f n*)
shows *isrlfm* (*foldr conj* (*map f xs*) *T*)
⟨*proof*⟩

lemma *DVDJ-l*: **assumes** *ip*: $i > 0$ **and** *np*: $n > 0$ **and** *nb*: *numbound0 s*
shows *isrlfm* (*DVDJ i n s*)
⟨*proof*⟩

lemma *NDVDJ-l*: **assumes** *ip*: $i > 0$ **and** *np*: $n > 0$ **and** *nb*: *numbound0 s*
shows *isrlfm* (*NDVDJ i n s*)
⟨*proof*⟩

definition *DVD* :: *int* ⇒ *int* ⇒ *num* ⇒ *fm* **where**

DVD-def: *DVD i c t* =
(*if i=0 then eq c t else*
if c = 0 then (*Dvd i t*) *else if c >0 then DVDJ* (*abs i*) *c t else DVDJ* (*abs i*) (−*c*) (*Neg t*))

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

lemma *DVD-mono*:
assumes *xp*: *0*≤ *x* **and** *x1*: *x* < *1*
shows ∀ *a n s*. *Inum* (*x*#*bs*) *a* = *Inum* (*x*#*bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x*#*bs*) (*DVD i n s*) = *Ifm* (*x*#*bs*) (*Dvd i a*)
(**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*DVD i n s*) = *?I* (*Dvd i a*))
⟨*proof*⟩

lemma *NDVD-mono*: **assumes** *xp*: *0*≤ *x* **and** *x1*: *x* < *1*
shows ∀ *a n s*. *Inum* (*x*#*bs*) *a* = *Inum* (*x*#*bs*) (*CN 0 n s*) ∧ *numbound0 s* ⟶ *Ifm* (*x*#*bs*) (*NDVD i n s*) = *Ifm* (*x*#*bs*) (*NDvd i a*)
(**is** ∀ *a n s*. *?N a* = *?N* (*CN 0 n s*) ∧ - ⟶ *?I* (*NDVD i n s*) = *?I* (*NDvd i a*))
⟨*proof*⟩

lemma *DVD-l*: *isrlfm* (*rsplit* (*DVD i*) *a*)
⟨*proof*⟩

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

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

$(rlfm\ q))$
$rlfm\ (NOT\ (NOT\ p)) = rlfm\ p$
$rlfm\ (NOT\ T) = F$
$rlfm\ (NOT\ F) = T$
$rlfm\ (NOT\ (Lt\ a)) = simpfm\ (rlfm\ (Ge\ a))$
$rlfm\ (NOT\ (Le\ a)) = simpfm\ (rlfm\ (Gt\ a))$
$rlfm\ (NOT\ (Gt\ a)) = simpfm\ (rlfm\ (Le\ a))$
$rlfm\ (NOT\ (Ge\ a)) = simpfm\ (rlfm\ (Lt\ a))$
$rlfm\ (NOT\ (Eq\ a)) = simpfm\ (rlfm\ (NEq\ a))$
$rlfm\ (NOT\ (NEq\ a)) = simpfm\ (rlfm\ (Eq\ a))$
$rlfm\ (NOT\ (Dvd\ i\ a)) = simpfm\ (rlfm\ (NDvd\ i\ a))$
$rlfm\ (NOT\ (NDvd\ i\ a)) = simpfm\ (rlfm\ (Dvd\ i\ a))$
$rlfm\ p = p$ (**hints** *simp add*: *fmsize-pos*)

lemma *bound0at-l* : $[\![isatom\ p\ ;\ bound0\ p]\!] \Longrightarrow isrlfm\ p$
⟨*proof*⟩
lemma *igcd-le1*: **assumes** *ip*: $0 < i$ **shows** $igcd\ i\ j \leq i$
⟨*proof*⟩

lemma *simpfm-rl*: $isrlfm\ p \Longrightarrow isrlfm\ (simpfm\ p)$
⟨*proof*⟩

lemma *rlfm-I*:
assumes *qfp*: *qfree p*
and *xp*: $0 \leq x$ **and** *x1*: $x < 1$
shows $(Ifm\ (x\#bs)\ (rlfm\ p) = Ifm\ (x\#\ bs)\ p) \wedge isrlfm\ (rlfm\ p)$
⟨*proof*⟩
lemma *rlfm-l*:
assumes *qfp*: *qfree p*
shows $isrlfm\ (rlfm\ p)$
⟨*proof*⟩

lemma *rminusinf-inf*:
assumes *lp*: $isrlfm\ p$
shows $\exists\ z.\ \forall\ x < z.\ Ifm\ (x\#bs)\ (minusinf\ p) = Ifm\ (x\#bs)\ p$ (**is** $\exists\ z.\ \forall\ x.$ *?P z x p*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *rminusinf-bound0*:
assumes *lp*: $isrlfm\ p$
shows $bound0\ (minusinf\ p)$

⟨*proof*⟩

lemma *rplusinf-bound0*:
assumes *lp*: *isrlfm p*
shows *bound0* (*plusinf p*)
⟨*proof*⟩

lemma *rminusinf-ex*:
assumes *lp*: *isrlfm p*
and *ex*: *Ifm* (*a*#*bs*) (*minusinf p*)
shows ∃ *x*. *Ifm* (*x*#*bs*) *p*
⟨*proof*⟩

lemma *rplusinf-ex*:
assumes *lp*: *isrlfm p*
and *ex*: *Ifm* (*a*#*bs*) (*plusinf p*)
shows ∃ *x*. *Ifm* (*x*#*bs*) *p*
⟨*proof*⟩

consts
ϒ:: *fm* ⇒ (*num* × *int*) *list*
υ :: *fm* ⇒ (*num* × *int*) ⇒ *fm*
recdef ϒ *measure size*
ϒ (*And p q*) = (ϒ *p* @ ϒ *q*)
ϒ (*Or p q*) = (ϒ *p* @ ϒ *q*)
ϒ (*Eq* (*CN 0 c e*)) = [(*Neg e*,*c*)]
ϒ (*NEq* (*CN 0 c e*)) = [(*Neg e*,*c*)]
ϒ (*Lt* (*CN 0 c e*)) = [(*Neg e*,*c*)]
ϒ (*Le* (*CN 0 c e*)) = [(*Neg e*,*c*)]
ϒ (*Gt* (*CN 0 c e*)) = [(*Neg e*,*c*)]
ϒ (*Ge* (*CN 0 c e*)) = [(*Neg e*,*c*)]
ϒ *p* = []

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

lemma *υ-I*: **assumes** *lp*: *isrlfm p*
and *np*: *real n* > *0* **and** *nbt*: *numbound0 t*
shows (*Ifm* (*x*#*bs*) (υ *p* (*t*,*n*)) = *Ifm* (((*Inum* (*x*#*bs*) *t*)/(*real n*))#*bs*) *p*) ∧ *bound0* (υ *p* (*t*,*n*)) (**is** (*?I x* (υ *p* (*t*,*n*)) = *?I ?u p*) ∧ *?B p* **is** (- = *?I* (*?t*/*?n*) *p*) ∧ - **is** (- = *?I* (*?N x t* /-) *p*) ∧ -)

⟨*proof*⟩

lemma *Υ-l*:
assumes *lp*: *isrlfm p*
shows ∀ (*t*,*k*) ∈ *set* (*Υ p*). *numbound0 t* ∧ *k* >*0*
⟨*proof*⟩

lemma *rminusinf-Υ*:
assumes *lp*: *isrlfm p*
and *nmi*: ¬ (*Ifm* (*a*#*bs*) (*minusinf p*)) (**is** ¬ (*Ifm* (*a*#*bs*) (*?M p*)))
and *ex*: *Ifm* (*x*#*bs*) *p* (**is** *?I x p*)
shows ∃ (*s*,*m*) ∈ *set* (*Υ p*). *x* ≥ *Inum* (*a*#*bs*) *s* / *real m* (**is** ∃ (*s*,*m*) ∈ *?U p*. *x* ≥ *?N a s* / *real m*)
⟨*proof*⟩

lemma *rplusinf-Υ*:
assumes *lp*: *isrlfm p*
and *nmi*: ¬ (*Ifm* (*a*#*bs*) (*plusinf p*)) (**is** ¬ (*Ifm* (*a*#*bs*) (*?M p*)))
and *ex*: *Ifm* (*x*#*bs*) *p* (**is** *?I x p*)
shows ∃ (*s*,*m*) ∈ *set* (*Υ p*). *x* ≤ *Inum* (*a*#*bs*) *s* / *real m* (**is** ∃ (*s*,*m*) ∈ *?U p*. *x* ≤ *?N a s* / *real m*)
⟨*proof*⟩

lemma *lin-dense*:
assumes *lp*: *isrlfm p*
and *noS*: ∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ (λ (*t*,*n*). *Inum* (*x*#*bs*) *t* / *real n*) ‘ *set* (*Υ p*)
(**is** ∀ *t*. - ∧ - ⟶ *t* ∉ (λ (*t*,*n*). *?N x t* / *real n*) ‘ (*?U p*))
and *lx*: *l* < *x* **and** *xu*:*x* < *u* **and** *px*: *Ifm* (*x*#*bs*) *p*
and *ly*: *l* < *y* **and** *yu*: *y* < *u*
shows *Ifm* (*y*#*bs*) *p*
⟨*proof*⟩

lemma *finite-set-intervals*:
assumes *px*: *P* (*x*::*real*)
and *lx*: *l* ≤ *x* **and** *xu*: *x* ≤ *u*
and *linS*: *l*∈ *S* **and** *uinS*: *u* ∈ *S*
and *fS*:*finite S* **and** *lS*: ∀ *x*∈ *S*. *l* ≤ *x* **and** *Su*: ∀ *x*∈ *S*. *x* ≤ *u*
shows ∃ *a* ∈ *S*. ∃ *b* ∈ *S*. (∀ *y*. *a* < *y* ∧ *y* < *b* ⟶ *y* ∉ *S*) ∧ *a* ≤ *x* ∧ *x* ≤ *b* ∧ *P x*
⟨*proof*⟩

lemma *finite-set-intervals2*:
assumes *px*: *P* (*x*::*real*)
and *lx*: *l* ≤ *x* **and** *xu*: *x* ≤ *u*
and *linS*: *l*∈ *S* **and** *uinS*: *u* ∈ *S*
and *fS*:*finite S* **and** *lS*: ∀ *x*∈ *S*. *l* ≤ *x* **and** *Su*: ∀ *x*∈ *S*. *x* ≤ *u*
shows (∃ *s*∈ *S*. *P s*) ∨ (∃ *a* ∈ *S*. ∃ *b* ∈ *S*. (∀ *y*. *a* < *y* ∧ *y* < *b* ⟶ *y* ∉ *S*) ∧ *a* < *x* ∧ *x* < *b* ∧ *P x*)

⟨*proof*⟩

lemma *rinf-*ϒ:
assumes *lp*: *isrlfm p*
and *nmi*: ¬ (*Ifm* (*x*#*bs*) (*minusinf p*)) (**is** ¬ (*Ifm* (*x*#*bs*) (*?M p*)))
and *npi*: ¬ (*Ifm* (*x*#*bs*) (*plusinf p*)) (**is** ¬ (*Ifm* (*x*#*bs*) (*?P p*)))
and *ex*: ∃ *x*. *Ifm* (*x*#*bs*) *p* (**is** ∃ *x*. *?I x p*)
shows ∃ (*l*,*n*) ∈ *set* (ϒ *p*). ∃ (*s*,*m*) ∈ *set* (ϒ *p*). *?I* ((*Inum* (*x*#*bs*) *l* / *real n* + *Inum* (*x*#*bs*) *s* / *real m*) / *2*) *p*
⟨*proof*⟩

theorem *fr-eq*:
assumes *lp*: *isrlfm p*
shows (∃ *x*. *Ifm* (*x*#*bs*) *p*) = ((*Ifm* (*x*#*bs*) (*minusinf p*)) ∨ (*Ifm* (*x*#*bs*) (*plusinf p*)) ∨ (∃ (*t*,*n*) ∈ *set* (ϒ *p*). ∃ (*s*,*m*) ∈ *set* (ϒ *p*). *Ifm* ((((*Inum* (*x*#*bs*) *t*)/ *real n* + (*Inum* (*x*#*bs*) *s*) / *real m*) /*2*)#*bs*) *p*))
(**is** (∃ *x*. *?I x p*) = (*?M* ∨ *?P* ∨ *?F*) **is** *?E* = *?D*)
⟨*proof*⟩

lemma *fr-eqv*:
assumes *lp*: *isrlfm p*
shows (∃ *x*. *Ifm* (*x*#*bs*) *p*) = ((*Ifm* (*x*#*bs*) (*minusinf p*)) ∨ (*Ifm* (*x*#*bs*) (*plusinf p*)) ∨ (∃ (*t*,*k*) ∈ *set* (ϒ *p*). ∃ (*s*,*l*) ∈ *set* (ϒ *p*). *Ifm* (*x*#*bs*) (υ *p* (*Add*(*Mul l t*) (*Mul k s*) , *2*∗*k*∗*l*))))
(**is** (∃ *x*. *?I x p*) = (*?M* ∨ *?P* ∨ *?F*) **is** *?E* = *?D*)
⟨*proof*⟩

The overall Part

lemma *real-ex-int-real01*:
shows (∃ (*x*::*real*). *P x*) = (∃ (*i*::*int*) (*u*::*real*). *0*≤ *u* ∧ *u*< *1* ∧ *P* (*real i* + *u*))
⟨*proof*⟩

consts *exsplitnum* :: *num* ⇒ *num*
exsplit :: *fm* ⇒ *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
⟨*proof*⟩

lemma *exsplit*:
assumes *qfp*: *qfree p*
shows *Ifm (x#y#bs) (exsplit p) = Ifm ((x+y)#bs) p*
⟨*proof*⟩

lemma *splitex*:
assumes *qf*: *qfree p*
shows (*Ifm bs (E p)*) = (∃ (*i*::*int*). *Ifm* (*real i*#*bs*) (*E* (*And* (*And* (*Ge*(*CN 0 1 (C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) (*exsplit p*)))) (**is** *?lhs = ?rhs*)
⟨*proof*⟩

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

lemma *fr-eq-01*:
assumes *qf*: *qfree p*
shows (∃ *x*. *Ifm* (*x*#*bs*) (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*)) = (∃ (*t*,*n*) ∈ *set* (ϒ (*rlfm* (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*))). ∃ (*s*,*m*) ∈ *set* (ϒ (*rlfm* (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*))). *Ifm* (*x*#*bs*) (υ (*rlfm* (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*)) (*Add* (*Mul m t*) (*Mul n s*), *2*∗*n*∗*m*)))
(**is** (∃ *x*. *?I x ?q*) = *?F*)

⟨*proof*⟩

lemma ϒ-*cong-aux*:
assumes *Ul*: ∀ (*t*,*n*) ∈ *set U*. *numbound0 t* ∧ *n* >*0*
shows ((λ (*t*,*n*). *Inum* (*x*#*bs*) *t* /*real n*) ‘ (*set* (*map* (λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , *2*∗*n*∗*m*)) (*alluopairs U*)))) = ((λ ((*t*,*n*),(*s*,*m*)). (*Inum* (*x*#*bs*) *t* /*real n* + *Inum* (*x*#*bs*) *s* /*real m*)/*2*) ‘ (*set U* × *set U*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma ϒ-*cong*:
assumes *lp*: *isrlfm p*
and *UU′*: ((λ (*t*,*n*). *Inum* (*x*#*bs*) *t* /*real n*) ‘ *U′*) = ((λ ((*t*,*n*),(*s*,*m*)). (*Inum* (*x*#*bs*) *t* /*real n* + *Inum* (*x*#*bs*) *s* /*real m*)/*2*) ‘ (*U* × *U*)) (**is** *?f* ‘ *U′* = *?g* ‘ (*U*×*U*))
and *U*: ∀ (*t*,*n*) ∈ *U*. *numbound0 t* ∧ *n* > *0*
and *U′*: ∀ (*t*,*n*) ∈ *U′*. *numbound0 t* ∧ *n* > *0*
shows (∃ (*t*,*n*) ∈ *U*. ∃ (*s*,*m*) ∈ *U*. *Ifm* (*x*#*bs*) (υ *p* (*Add* (*Mul m t*) (*Mul n s*),*2*∗*n*∗*m*))) = (∃ (*t*,*n*) ∈ *U′*. *Ifm* (*x*#*bs*) (υ *p* (*t*,*n*)))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *ferrack01*:
assumes *qf*: *qfree p*
shows ((∃ *x*. *Ifm* (*x*#*bs*) (*And* (*And* (*Ge*(*CN 0 1* (*C 0*))) (*Lt* (*CN 0 1* (*C* (− *1*))))) *p*)) = (*Ifm bs* (*ferrack01 p*))) ∧ *qfree* (*ferrack01 p*) (**is** (*?lhs* = *?rhs*) ∧ -)
⟨*proof*⟩

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*))
⟨*proof*⟩

constdefs *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 ≤ *length a then* (*q*,*B*,*d*) *else* (*mirror q*, *a*,*d*))

lemma *unit*: **assumes** *qf*: *qfree p*
shows ⋀ *q B d*. *unit p* = (*q*,*B*,*d*) ⟹ ((∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) = (∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *q*)) ∧ (*Inum* (*real i*#*bs*)) ‘ *set B* = (*Inum* (*real i*#*bs*)) ‘ *set* (*β q*) ∧ *dβ q 1* ∧ *dδ q d* ∧ *d* >*0* ∧ *iszlfm q* (*real* (*i*::*int*)#*bs*) ∧ (∀ *b*∈ *set B*. *numbound0 b*)
⟨*proof*⟩

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

(**is** (*?lhs* = *?rhs*) ∧ -)
⟨*proof*⟩

lemma *DJcooper*:
assumes *qf*: *qfree p*
shows ((∃ (*x*::*int*). *Ifm* (*real x*#*bs*) *p*) = (*Ifm bs* (*DJ cooper p*))) ∧ *qfree* (*DJ cooper p*)
⟨*proof*⟩

lemma *σϱ-cong*: **assumes** *lp*: *iszlfm p* (*a*#*bs*) **and** *tt′*: *Inum* (*a*#*bs*) *t* = *Inum* (*a*#*bs*) *t′*
shows *Ifm* (*a*#*bs*) (*σϱ p* (*t*,*c*)) = *Ifm* (*a*#*bs*) (*σϱ p* (*t′*,*c*))
⟨*proof*⟩

lemma *σ-cong*: **assumes** *lp*: *iszlfm p* (*a*#*bs*) **and** *tt′*: *Inum* (*a*#*bs*) *t* = *Inum* (*a*#*bs*) *t′*
shows *Ifm* (*a*#*bs*) (*σ p c t*) = *Ifm* (*a*#*bs*) (*σ p c t′*)
⟨*proof*⟩

lemma *ϱ-cong*: **assumes** *lp*: *iszlfm p* (*a*#*bs*)
and *RR*: (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *R* = (λ(*b*,*k*). (*Inum* (*a*#*bs*) *b*,*k*)) ‘ *set* (*ϱ p*)
shows (∃ (*e*,*c*) ∈ *R*. ∃ *j*∈ {*1*.. *c*∗(*δ p*)}. *Ifm* (*a*#*bs*) (*σ p c* (*Add e* (*C j*)))) = (∃ (*e*,*c*) ∈ *set* (*ϱ p*). ∃ *j*∈ {*1*.. *c*∗(*δ p*)}. *Ifm* (*a*#*bs*) (*σ p c* (*Add e* (*C j*))))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

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

$(minusinf\ p)) \vee (\exists\ (e,c) \in R.\ \exists\ j{\in}\ \{1..\ c{*}(\delta\ p)\}.\ Ifm\ (a\#bs)\ (\sigma\ p\ c\ (Add\ e\ (C\ j)))))$
⟨*proof*⟩

constdefs *chooset*:: $fm \Rightarrow fm \times ((num{\times}int)\ list) \times int$
$chooset\ p \equiv (let\ q = zlfm\ p\ ;\ d = \delta\ q;$
$B = remdups\ (map\ (\lambda\ (t,k).\ (simpnum\ t,k))\ (\varrho\ q))\ ;$
$a = remdups\ (map\ (\lambda\ (t,k).\ (simpnum\ t,k))\ (\alpha\varrho\ q))$
$in\ if\ length\ B \leq length\ a\ then\ (q,B,d)\ else\ (mirror\ q,\ a,d))$

lemma *chooset*: **assumes** *qf*: *qfree p*
shows $\bigwedge\ q\ B\ d.\ chooset\ p = (q,B,d) \Longrightarrow ((\exists\ (x{::}int).\ Ifm\ (real\ x\#bs)\ p) = (\exists\ (x{::}int).\ Ifm\ (real\ x\#bs)\ q)) \wedge ((\lambda(t,k).\ (Inum\ (real\ i\#bs)\ t,k))\ `\ set\ B = (\lambda(t,k).\ (Inum\ (real\ i\#bs)\ t,k))\ `\ set\ (\varrho\ q)) \wedge (\delta\ q = d) \wedge d > 0 \wedge iszlfm\ q\ (real\ (i{::}int)\#bs) \wedge (\forall\ (e,c){\in}\ set\ B.\ numbound0\ e \wedge c{>}0)$
⟨*proof*⟩

constdefs *stage*:: $fm \Rightarrow int \Rightarrow (num \times int) \Rightarrow fm$
$stage\ p\ d \equiv (\lambda\ (e,c).\ evaldjf\ (\lambda\ j.\ simpfm\ (\sigma\ p\ c\ (Add\ e\ (C\ j))))\ (iupt\ (1,c{*}d)))$
lemma *stage*:
shows $Ifm\ bs\ (stage\ p\ d\ (e,c)) = (\exists\ j{\in}\{1\ ..\ c{*}d\}.\ Ifm\ bs\ (\sigma\ p\ c\ (Add\ e\ (C\ j))))$
⟨*proof*⟩

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*⟩

constdefs *redlove*:: $fm \Rightarrow fm$
$redlove\ p \equiv$
$(let\ (q,B,d) = chooset\ p;$
$mq = simpfm\ (minusinf\ q);$
$md = evaldjf\ (\lambda\ j.\ simpfm\ (subst0\ (C\ j)\ mq))\ (iupt\ (1,d))$
$in\ if\ md = T\ then\ T\ else$
$(let\ qd = evaldjf\ (stage\ q\ d)\ B$
$in\ decr\ (disj\ md\ qd)))$

lemma *redlove*: **assumes** *qf*: *qfree p*
shows $((\exists\ (x{::}int).\ Ifm\ (real\ x\#bs)\ p) = (Ifm\ bs\ (redlove\ p))) \wedge qfree\ (redlove\ p)$
(**is** $(?lhs = ?rhs) \wedge$ -)
⟨*proof*⟩

lemma *DJredlove*:
assumes *qf*: *qfree p*
shows $((\exists\ (x{::}int).\ Ifm\ (real\ x\#bs)\ p) = (Ifm\ bs\ (DJ\ redlove\ p))) \wedge qfree\ (DJ\ redlove\ p)$
⟨*proof*⟩

lemma *exsplit-qf*: **assumes** *qf*: *qfree p*
shows *qfree* (*exsplit p*)
⟨*proof*⟩

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

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

lemma *mircfr*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*mircfr p*) ∧ *Ifm bs* (*mircfr p*) = *Ifm bs* (*E p*)
⟨*proof*⟩

lemma *mirlfr*: ∀ *bs p*. *qfree p* ⟶ *qfree*(*mirlfr p*) ∧ *Ifm bs* (*mirlfr p*) = *Ifm bs* (*E p*)
⟨*proof*⟩

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*)
⟨*proof*⟩

theorem *mirlfrqe*: (*Ifm bs* (*mirlfrqe p*) = *Ifm bs p*) ∧ *qfree* (*mirlfrqe p*)
⟨*proof*⟩

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*⟩

lemma *ALL* (*x*::*real*). (⌊*x*⌋ = ⌈*x*⌉ = (*x* = *real* ⌊*x*⌋))
⟨*proof*⟩

lemma *ALL* (*x*::*real*). *real* (*2*::*int*)∗*x* − (*real* (*1*::*int*)) < *real* ⌊*x*⌋ + *real* ⌈*x*⌉ ∧ *real* ⌊*x*⌋ + *real* ⌈*x*⌉ ≤ *real* (*2*::*int*)∗*x* + (*real* (*1*::*int*))
⟨*proof*⟩

lemma *ALL* (*x*::*real*). *2*∗⌊*x*⌋ ≤ ⌊*2*∗*x*⌋ ∧ ⌊*2*∗*x*⌋ ≤ *2*∗⌊*x*+*1*⌋
⟨*proof*⟩

lemma *ALL* (*x*::*real*). ∃ *y* ≤ *x*. (⌊*x*⌋ = ⌈*y*⌉)
⟨*proof*⟩

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* ⟨*proof*⟩

lemma *index-of-nat-nat-of-index* [*simp*]:
index-of-nat (*nat-of-index k*) = *k*
⟨*proof*⟩

lemma *nat-of-index-index-of-nat* [*simp*]:
nat-of-index (*index-of-nat n*) = *n*
⟨*proof*⟩

lemma *index*:
(⋀*n*::*index*. *PROP P n*) ≡ (⋀*n*::*nat*. *PROP P* (*index-of-nat n*))
⟨*proof*⟩

lemma *index-case*:
assumes ⋀*n*. *k* = *index-of-nat n* ⟹ *P*
shows *P*
⟨*proof*⟩

lemma *index-induct-raw*:
assumes ⋀*n*. *P* (*index-of-nat n*)
shows *P k*
⟨*proof*⟩

lemma *nat-of-index-inject* [*simp*]:
nat-of-index k = *nat-of-index l* ⟷ *k* = *l*
⟨*proof*⟩

lemma *index-of-nat-inject* [*simp*]:
index-of-nat n = *index-of-nat m* ⟷ *n* = *m*
⟨*proof*⟩

instantiation *index* :: *zero*
begin

definition [*simp*, *code func del*]:
0 = *index-of-nat 0*

instance ⟨*proof*⟩

end

definition [*simp*]:
Suc-index k = *index-of-nat* (*Suc* (*nat-of-index k*))

lemma *index-induct*: *P 0* ⟹ (⋀*k*. *P k* ⟹ *P* (*Suc-index k*)) ⟹ *P k*
⟨*proof*⟩

lemma *Suc-not-Zero-index*: *Suc-index k* ≠ *0*
⟨*proof*⟩

lemma *Zero-not-Suc-index*: *0* ≠ *Suc-index k*
⟨*proof*⟩

lemma *Suc-Suc-index-eq*: *Suc-index k* = *Suc-index l* ⟷ *k* = *l*
⟨*proof*⟩

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*⟩

lemma [*code func*]:
size = *nat-of-index*
⟨*proof*⟩

lemma [*code func*]:
k = *l* ⟷ *nat-of-index k* = *nat-of-index l*
⟨*proof*⟩

13.2 Indices as datatype of ints

instantiation *index* :: *number*
begin

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

instance ⟨*proof*⟩

end

lemma *nat-of-index-number* [*simp*]:
nat-of-index (*number-of k*) = *number-of k*
⟨*proof*⟩

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

13.3 Basic arithmetic

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

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

⟨*proof*⟩
lemma [*code post*]: *Numeral0* = (*0*::*index*)
⟨*proof*⟩

definition [*simp*, *code func del*]:
(*1*::*index*) = *index-of-nat 1*

lemma *one-index-code* [*code inline*, *code func*]:
(*1*::*index*) = *Numeral1*
⟨*proof*⟩
lemma [*code post*]: *Numeral1* = (*1*::*index*)
⟨*proof*⟩

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

lemma *plus-index-code* [*code func*]:
index-of-nat n + *index-of-nat m* = *index-of-nat* (*n* + *m*)
⟨*proof*⟩

definition [*simp*, *code func del*]:
n − *m* = *index-of-nat* (*nat-of-index n* − *nat-of-index m*)

definition [*simp*, *code func del*]:
n ∗ *m* = *index-of-nat* (*nat-of-index n* ∗ *nat-of-index m*)

lemma *times-index-code* [*code func*]:
index-of-nat n ∗ *index-of-nat m* = *index-of-nat* (*n* ∗ *m*)
⟨*proof*⟩

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

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

lemma *div-index-code* [*code func*]:
index-of-nat n div index-of-nat m = *index-of-nat* (*n div m*)
⟨*proof*⟩

lemma *mod-index-code* [*code func*]:
index-of-nat n mod index-of-nat m = *index-of-nat* (*n mod m*)
⟨*proof*⟩

definition [*simp*, *code func del*]:
n ≤ *m* ⟷ *nat-of-index n* ≤ *nat-of-index m*

definition [*simp*, *code func del*]:
n < *m* ⟷ *nat-of-index n* < *nat-of-index m*

lemma *less-eq-index-code* [*code func*]:
index-of-nat n ≤ *index-of-nat m* ⟷ *n* ≤ *m*
⟨*proof*⟩

lemma *less-index-code* [*code func*]:
index-of-nat n < *index-of-nat m* ⟷ *n* < *m*
⟨*proof*⟩

instance ⟨*proof*⟩

end

lemma *Suc-index-minus-one*: *Suc-index n* − *1* = *n* ⟨*proof*⟩

lemma *index-of-nat-code* [*code*]:
index-of-nat = *of-nat*
⟨*proof*⟩

lemma *index-not-eq-zero*: *i* ≠ *index-of-nat 0* ⟷ *i* ≥ *1*
⟨*proof*⟩

definition
nat-of-index-aux :: *index* ⇒ *nat* ⇒ *nat*
where
nat-of-index-aux i n = *nat-of-index i* + *n*

lemma *nat-of-index-aux-code* [*code*]:
nat-of-index-aux i n = (*if i* = *0 then n else nat-of-index-aux* (*i* − *1*) (*Suc n*))
⟨*proof*⟩

lemma *nat-of-index-code* [*code*]:
nat-of-index i = *nat-of-index-aux i 0*
⟨*proof*⟩

13.4 ML interface

⟨*ML*⟩

13.5 Specialized *op* −, *op div* and *op mod* operations

definition
minus-index-aux :: *index* ⇒ *index* ⇒ *index*
where
[*code func del*]: *minus-index-aux* = *op* −

lemma [*code func*]: *op* − = *minus-index-aux*
⟨*proof*⟩

definition

div-mod-index :: *index* ⇒ *index* ⇒ *index* × *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*))
⟨*proof*⟩

lemma [*code func*]:
n div m = *fst* (*div-mod-index n m*)
⟨*proof*⟩

lemma [*code func*]:
n mod m = *snd* (*div-mod-index n m*)
⟨*proof*⟩

13.6 Code serialization

Implementation of indices by bounded integers

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

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

⟨*ML*⟩

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 representation using *0* and *Suc* is unsuitable for computations involving large numbers. The efficiency of the generated code can be improved drastically by implementing natural numbers by 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 = (*Numeral0* :: *nat*)
⟨*proof*⟩
lemmas [*code post*] = *zero-nat-code* [*symmetric*]

lemma *one-nat-code* [*code*, *code unfold*]:
1 = (*Numeral1* :: *nat*)

⟨*proof*⟩
lemmas [*code post*] = *one-nat-code* [*symmetric*]

lemma *Suc-code* [*code*]:
Suc n = *n* + *1*
⟨*proof*⟩

lemma *plus-nat-code* [*code*]:
n + *m* = *nat* (*of-nat n* + *of-nat m*)
⟨*proof*⟩

lemma *minus-nat-code* [*code*]:
n − *m* = *nat* (*of-nat n* − *of-nat m*)
⟨*proof*⟩

lemma *times-nat-code* [*code*]:
n ∗ *m* = *nat* (*of-nat n* ∗ *of-nat m*)
⟨*proof*⟩

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

definition
divmod-aux :: *nat* ⇒ *nat* ⇒ *nat* × *nat*
where
[*code func del*]: *divmod-aux* = *divmod*

lemma [*code func*]:
divmod n m = (*if m* = *0 then* (*0*, *n*) *else divmod-aux n m*)
⟨*proof*⟩

lemma *divmod-aux-code* [*code*]:
divmod-aux n m = (*nat* (*of-nat n div of-nat m*), *nat* (*of-nat n mod of-nat m*))
⟨*proof*⟩

lemma *eq-nat-code* [*code*]:
n = *m* ⟷ (*of-nat n* :: *int*) = *of-nat m*
⟨*proof*⟩

lemma *less-eq-nat-code* [*code*]:
n ≤ *m* ⟷ (*of-nat n* :: *int*) ≤ *of-nat m*
⟨*proof*⟩

lemma *less-nat-code* [*code*]:
n < *m* ⟷ (*of-nat n* :: *int*) < *of-nat m*
⟨*proof*⟩

14.2 Case analysis

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

lemma [*code func*, *code unfold*]:
nat-case = (λ*f g n*. *if n* = *0 then f else g* (*n* − *1*))
⟨*proof*⟩

14.3 Preprocessors

In contrast to *Suc n*, the term *n* + *1* is no longer a constructor term. Therefore, all occurrences of this term in a position where a pattern is expected (i.e. on the left-hand side of a recursion equation or in the arguments of an inductive relation in an introduction rule) must be eliminated. This can be accomplished by applying the following transformation rules:

lemma *Suc-if-eq*: (⋀*n*. *f* (*Suc n*) = *h n*) ⟹ *f 0* = *g* ⟹
f n = (*if n* = *0 then g else h* (*n* − *1*))
⟨*proof*⟩

lemma *Suc-clause*: (⋀*n*. *P n* (*Suc n*)) ⟹ *n* ≠ *0* ⟹ *P* (*n* − *1*) *n*
⟨*proof*⟩

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.

⟨*ML*⟩

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 = *HOLogic.mk-number HOLogic.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)*
⟨*proof*⟩

⟨*ML*⟩

Since natural numbers are implemented using integers in ML, the coercion

function *of-nat* of type *nat* ⇒ *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* ⇒ *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*)
  ⟨*proof*⟩

lemma *nat-code′* [*code func*]:
  nat (*number-of l*) = (*if neg* (*number-of l* :: *int*) *then 0 else number-of l*)
  ⟨*proof*⟩

lemma *of-nat-int* [*code unfold*]:
  of-nat = *int* ⟨*proof*⟩
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,+,¡)

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*}
⟨*proof*⟩

lemma *alluopairs-set*:
⟦*x*∈ *set xs* ; *y* ∈ *set xs*⟧ ⟹ (*x*,*y*) ∈ *set* (*alluopairs xs*) ∨ (*y*,*x*) ∈ *set* (*alluopairs xs*)
⟨*proof*⟩

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*⟩

lemma *nth-pos2*: *0* < *n* ⟹ (*x*#*xs*) ! *n* = *xs* ! (*n* − *1*)
⟨*proof*⟩

lemma *filter-length*: *length* (*List.filter P xs*) < *Suc* (*length xs*)

⟨*proof*⟩

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*
⟨*proof*⟩

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

consts *num-size* :: *num* ⇒ *nat*
primrec
num-size (*C c*) = *1*
num-size (*Bound n*) = *1*
num-size (*Neg a*) = *1* + *num-size a*
num-size (*Add a b*) = *1* + *num-size a* + *num-size b*
num-size (*Sub a b*) = *3* + *num-size a* + *num-size b*
num-size (*Mul c a*) = *1* + *num-size a*
num-size (*CN n c a*) = *3* + *num-size a*

consts *Inum* :: *real list* ⇒ *num* ⇒ *real*
primrec
Inum bs (*C c*) = (*real c*)
Inum bs (*Bound n*) = *bs*!*n*
Inum bs (*CN n c a*) = (*real c*) ∗ (*bs*!*n*) + (*Inum bs a*)
Inum bs (*Neg a*) = −(*Inum bs a*)
Inum bs (*Add a b*) = *Inum bs a* + *Inum bs b*
Inum bs (*Sub a b*) = *Inum bs a* − *Inum bs b*
Inum bs (*Mul c a*) = (*real c*) ∗ *Inum bs a*

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

consts *fmsize* :: *fm* ⇒ *nat*
recdef *fmsize measure size*
fmsize (*NOT p*) = *1* + *fmsize p*
fmsize (*And p q*) = *1* + *fmsize p* + *fmsize q*
fmsize (*Or p q*) = *1* + *fmsize p* + *fmsize q*
fmsize (*Imp p q*) = *3* + *fmsize p* + *fmsize q*
fmsize (*Iff p q*) = *3* + *2*∗(*fmsize p* + *fmsize q*)
fmsize (*E p*) = *1* + *fmsize p*
fmsize (*A p*) = *4*+ *fmsize p*
fmsize p = *1*

lemma *fmsize-pos*: *fmsize p* > *0*
⟨*proof*⟩

consts *Ifm* ::*real list* ⇒ *fm* ⇒ *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* ≤ *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 (*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*)

lemma *IfmLeSub*: ⟦ *Inum bs s* = *s'* ; *Inum bs t* = *t'* ⟧ ⟹ *Ifm bs* (*Le* (*Sub s t*))
= (*s'* ≤ *t'*)
⟨*proof*⟩

lemma *IfmLtSub*: ⟦ *Inum bs s* = *s'* ; *Inum bs t* = *t'* ⟧ ⟹ *Ifm bs* (*Lt* (*Sub s t*))
= (*s'* < *t'*)
⟨*proof*⟩
lemma *IfmEqSub*: ⟦ *Inum bs s* = *s'* ; *Inum bs t* = *t'* ⟧ ⟹ *Ifm bs* (*Eq* (*Sub s t*))
= (*s'* = *t'*)
⟨*proof*⟩
lemma *IfmNOT*: (*Ifm bs p* = *P*) ⟹ (*Ifm bs* (*NOT p*) = (¬*P*))
⟨*proof*⟩
lemma *IfmAnd*: ⟦ *Ifm bs p* = *P* ; *Ifm bs q* = *Q*⟧ ⟹ (*Ifm bs* (*And p q*) = (*P* ∧
Q))
⟨*proof*⟩

lemma *IfmOr*: $[\![Ifm\ bs\ p = P\ ;\ Ifm\ bs\ q = Q]\!] \Longrightarrow (Ifm\ bs\ (Or\ p\ q) = (P \vee Q))$
⟨*proof*⟩
lemma *IfmImp*: $[\![Ifm\ bs\ p = P\ ;\ Ifm\ bs\ q = Q]\!] \Longrightarrow (Ifm\ bs\ (Imp\ p\ q) = (P \longrightarrow Q))$
⟨*proof*⟩
lemma *IfmIff*: $[\![Ifm\ bs\ p = P\ ;\ Ifm\ bs\ q = Q]\!] \Longrightarrow (Ifm\ bs\ (Iff\ p\ q) = (P = Q))$
⟨*proof*⟩

lemma *IfmE*: $(!!\ x.\ Ifm\ (x\#bs)\ p = P\ x) \Longrightarrow (Ifm\ bs\ (E\ p) = (\exists\ x.\ P\ x))$
⟨*proof*⟩
lemma *IfmA*: $(!!\ x.\ Ifm\ (x\#bs)\ p = P\ x) \Longrightarrow (Ifm\ bs\ (A\ p) = (\forall\ x.\ P\ x))$
⟨*proof*⟩

consts *not*:: *fm* ⇒ *fm*
recdef *not measure size*
not (*NOT p*) = *p*
not T = *F*
not F = *T*
not p = *NOT p*
lemma *not*[*simp*]: *Ifm bs* (*not p*) = *Ifm bs* (*NOT p*)
⟨*proof*⟩

constdefs *conj* :: *fm* ⇒ *fm* ⇒ *fm*
conj p q ≡ (*if* (*p* = *F* ∨ *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*)
⟨*proof*⟩

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

lemma *disj*[*simp*]: *Ifm bs* (*disj p q*) = *Ifm bs* (*Or p q*)
⟨*proof*⟩

constdefs *imp* :: *fm* ⇒ *fm* ⇒ *fm*
imp p q ≡ (*if* (*p* = *F* ∨ *q*=*T* ∨ *p*=*q*) *then T else if p*=*T then q else if q*=*F then not p*
else Imp p q)
lemma *imp*[*simp*]: *Ifm bs* (*imp p q*) = *Ifm bs* (*Imp p q*)
⟨*proof*⟩

constdefs *iff* :: *fm* ⇒ *fm* ⇒ *fm*
iff p q ≡ (*if* (*p* = *q*) *then T else if* (*p* = *NOT q* ∨ *NOT p* = *q*) *then F else if p*=*F then not q else if q*=*F then not p else if p*=*T then q else if q*=*T then p else*
Iff p q)
lemma *iff*[*simp*]: *Ifm bs* (*iff p q*) = *Ifm bs* (*Iff p q*)
⟨*proof*⟩

lemma *conj-simps*:
conj F Q = F
conj P F = F
conj T Q = Q
conj P T = P
conj P P = P
$P \neq T \Longrightarrow P \neq F \Longrightarrow Q \neq T \Longrightarrow Q \neq F \Longrightarrow P \neq Q \Longrightarrow$ *conj P Q = And P Q*
⟨*proof*⟩

lemma *disj-simps*:
disj T Q = T
disj P T = T
disj F Q = Q
disj P F = P
disj P P = P
$P \neq T \Longrightarrow P \neq F \Longrightarrow Q \neq T \Longrightarrow Q \neq F \Longrightarrow P \neq Q \Longrightarrow$ *disj P Q = Or P Q*
⟨*proof*⟩
lemma *imp-simps*:
imp F Q = T
imp P T = T
imp T Q = Q
imp P F = not P
imp P P = T
$P \neq T \Longrightarrow P \neq F \Longrightarrow P \neq Q \Longrightarrow Q \neq T \Longrightarrow Q \neq F \Longrightarrow$ *imp P Q = Imp P Q*
⟨*proof*⟩
lemma *trivNOT*: $p \neq NOT\ p\ NOT\ p \neq p$
⟨*proof*⟩

lemma *iff-simps*:
iff p p = T
iff p (NOT p) = F
iff (NOT p) p = F
iff p F = not p
iff F p = not p
$p \neq NOT\ T \Longrightarrow$ *iff T p = p*
$p \neq NOT\ T \Longrightarrow$ *iff p T = p*
$p \neq q \Longrightarrow p \neq NOT\ q \Longrightarrow q \neq NOT\ p \Longrightarrow p \neq F \Longrightarrow q \neq F \Longrightarrow p \neq T \Longrightarrow q \neq T \Longrightarrow$ *iff p q = Iff p q*
⟨*proof*⟩

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

$qfree\ (Or\ \ p\ q) = (qfree\ p \wedge qfree\ q)$
$qfree\ (Imp\ p\ q) = (qfree\ p \wedge qfree\ q)$
$qfree\ (Iff\ p\ q) = (qfree\ p \wedge qfree\ q)$
$qfree\ p = True$

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 $Inum\ (b\#bs)\ a = Inum\ (b'\#bs)\ a$
⟨*proof*⟩

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 $Ifm\ (b\#bs)\ p = Ifm\ (b'\#bs)\ p$
⟨*proof*⟩

lemma *not-qf*[*simp*]: $qfree\ p \Longrightarrow qfree\ (not\ p)$
⟨*proof*⟩
lemma *not-bn*[*simp*]: $bound0\ p \Longrightarrow bound0\ (not\ p)$
⟨*proof*⟩

lemma *conj-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*conj p q*)
⟨*proof*⟩
lemma *conj-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*conj p q*)
⟨*proof*⟩

lemma *disj-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*disj p q*)
⟨*proof*⟩
lemma *disj-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*disj p q*)
⟨*proof*⟩

lemma *imp-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*imp p q*)
⟨*proof*⟩
lemma *imp-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*imp p q*)
⟨*proof*⟩

lemma *iff-qf*[*simp*]: ⟦*qfree p* ; *qfree q*⟧ ⟹ *qfree* (*iff p q*)
⟨*proof*⟩
lemma *iff-nb*[*simp*]: ⟦*bound0 p* ; *bound0 q*⟧ ⟹ *bound0* (*iff p q*)
⟨*proof*⟩

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

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

recdef *decr measure size*
decr (*Lt a*) = *Lt* (*decrnum a*)
decr (*Le a*) = *Le* (*decrnum a*)
decr (*Gt a*) = *Gt* (*decrnum a*)
decr (*Ge a*) = *Ge* (*decrnum a*)
decr (*Eq a*) = *Eq* (*decrnum a*)
decr (*NEq a*) = *NEq* (*decrnum a*)
decr (*NOT p*) = *NOT* (*decr p*)
decr (*And p q*) = *conj* (*decr p*) (*decr q*)
decr (*Or p q*) = *disj* (*decr p*) (*decr q*)
decr (*Imp p q*) = *imp* (*decr p*) (*decr q*)
decr (*Iff p q*) = *iff* (*decr p*) (*decr q*)
decr p = *p*

lemma *decrnum*: **assumes** *nb*: *numbound0 t*

shows *Inum* $(x\#bs)$ t = *Inum bs* (*decrnum t*)
⟨*proof*⟩

lemma *decr*: **assumes** *nb*: *bound0 p*
shows *Ifm* $(x\#bs)$ p = *Ifm bs* (*decr p*)
⟨*proof*⟩

lemma *decr-qf*: *bound0 p* ⟹ *qfree* (*decr p*)
⟨*proof*⟩

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

lemma *bound0-qf*: *bound0 p* ⟹ *qfree p*
⟨*proof*⟩

constdefs *djf*:: ('*a* ⇒ *fm*) ⇒ '*a* ⇒ *fm* ⇒ *fm*
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* ⇒ *T* | *F* ⇒ *q* | - ⇒ *Or* (*f p*) *q*))
constdefs *evaldjf*:: ('*a* ⇒ *fm*) ⇒ '*a list* ⇒ *fm*
evaldjf f ps ≡ *foldr* (*djf f*) *ps F*

lemma *djf-Or*: *Ifm bs* (*djf f p q*) = *Ifm bs* (*Or* (*f p*) *q*)
⟨*proof*⟩

lemma *djf-simps*:
djf f p T = *T*
djf f p F = *f p*
q≠*T* ⟹ *q*≠*F* ⟹ *djf f p q* = (*let fp* = *f p in case fp of T* ⇒ *T* | *F* ⇒ *q* | - ⇒
Or (*f p*) *q*)
⟨*proof*⟩

lemma *evaldjf-ex*: *Ifm bs* (*evaldjf f ps*) = (∃ *p* ∈ *set ps*. *Ifm bs* (*f p*))
⟨*proof*⟩

lemma *evaldjf-bound0*:
assumes *nb*: ∀ *x*∈ *set xs*. *bound0* (*f x*)
shows *bound0* (*evaldjf f xs*)

⟨*proof*⟩

lemma *evaldjf-qf*:
assumes *nb*: ∀ *x*∈ *set xs*. *qfree* (*f x*)
shows *qfree* (*evaldjf f xs*)
⟨*proof*⟩

consts *disjuncts* :: *fm* ⇒ *fm list*
recdef *disjuncts measure size*
disjuncts (*Or p q*) = (*disjuncts p*) @ (*disjuncts q*)
disjuncts F = []
disjuncts p = [*p*]

lemma *disjuncts*: (∃ *q*∈ *set* (*disjuncts p*). *Ifm bs q*) = *Ifm bs p*
⟨*proof*⟩

lemma *disjuncts-nb*: *bound0 p* ⟹ ∀ *q*∈ *set* (*disjuncts p*). *bound0 q*
⟨*proof*⟩

lemma *disjuncts-qf*: *qfree p* ⟹ ∀ *q*∈ *set* (*disjuncts p*). *qfree q*
⟨*proof*⟩

constdefs *DJ* :: (*fm* ⇒ *fm*) ⇒ *fm* ⇒ *fm*
DJ f p ≡ *evaldjf f* (*disjuncts p*)

lemma *DJ*: **assumes** *fdj*: ∀ *p q*. *Ifm bs* (*f* (*Or p q*)) = *Ifm bs* (*Or* (*f p*) (*f q*))
and *fF*: *f F* = *F*
shows *Ifm bs* (*DJ f p*) = *Ifm bs* (*f p*)
⟨*proof*⟩

lemma *DJ-qf*: **assumes**
fqf: ∀ *p*. *qfree p* ⟶ *qfree* (*f p*)
shows ∀ *p*. *qfree p* ⟶ *qfree* (*DJ f p*)
⟨*proof*⟩

lemma *DJ-qe*: **assumes** *qe*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ∀ *bs p*. *qfree p* ⟶ *qfree* (*DJ qe p*) ∧ (*Ifm bs* ((*DJ qe p*)) = *Ifm bs* (*E p*))
⟨*proof*⟩

consts
numgcd :: *num* ⇒ *int*
numgcdh:: *num* ⇒ *int* ⇒ *int*
reducecoeffh:: *num* ⇒ *int* ⇒ *num*
reducecoeff :: *num* ⇒ *num*
dvdnumcoeff:: *num* ⇒ *int* ⇒ *bool*
consts *maxcoeff*:: *num* ⇒ *int*
recdef *maxcoeff measure size*

maxcoeff (*C i*) = *abs i*
maxcoeff (*CN n c t*) = *max* (*abs c*) (*maxcoeff t*)
maxcoeff t = *1*

lemma *maxcoeff-pos*: *maxcoeff t* ≥ *0*
⟨*proof*⟩

recdef *numgcdh measure size*
numgcdh (*C i*) = (λ*g*. *igcd i g*)
numgcdh (*CN n c t*) = (λ*g*. *igcd c* (*numgcdh t g*))
numgcdh t = (λ*g*. *1*)
defs *numgcd-def* [*code func*]: *numgcd t* ≡ *numgcdh t* (*maxcoeff t*)

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

defs *reducecoeff-def*: *reducecoeff t* ≡
(*let g* = *numgcd t in*
if g = *0 then C 0 else if g*=*1 then t else reducecoeffh t g*)

recdef *dvdnumcoeff measure size*
dvdnumcoeff (*C i*) = (λ *g*. *g dvd i*)
dvdnumcoeff (*CN n c t*) = (λ *g*. *g dvd c* ∧ (*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*
⟨*proof*⟩

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

lemma *natabs0*: (*nat* (*abs x*) = *0*) = (*x* = *0*)
⟨*proof*⟩

lemma *numgcd0*:
assumes *g0*: *numgcd t* = *0*
shows *Inum bs t* = *0*
⟨*proof*⟩

lemma *numgcdh-pos*: **assumes** *gp*: *g* ≥ *0* **shows** *numgcdh t g* ≥ *0*
⟨*proof*⟩

lemma *numgcd-pos*: *numgcd t* ≥*0*
⟨*proof*⟩

lemma *reducecoeffh*:

assumes *gt*: *dvdnumcoeff t g* **and** *gp*: *g* > *0*
shows *real g* ∗(*Inum bs* (*reducecoeffh t g*)) = *Inum bs t*
⟨*proof*⟩
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*′
⟨*proof*⟩

lemma *maxcoeff-ismaxcoeff*: *ismaxcoeff t* (*maxcoeff t*)
⟨*proof*⟩

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*))
⟨*proof*⟩
lemma *numgcdh0*:*numgcdh t m* = *0* ⟹ *m* =*0*
⟨*proof*⟩

lemma *dvdnumcoeff-aux*:
assumes *ismaxcoeff t m* **and** *mp*:*m* ≥ *0* **and** *numgcdh t m* > *1*
shows *dvdnumcoeff t* (*numgcdh t m*)
⟨*proof*⟩

lemma *dvdnumcoeff-aux2*:
assumes *numgcd t* > *1* **shows** *dvdnumcoeff t* (*numgcd t*) ∧ *numgcd t* > *0*
⟨*proof*⟩

lemma *reducecoeff*: *real* (*numgcd t*) ∗ (*Inum bs* (*reducecoeff t*)) = *Inum bs t*
⟨*proof*⟩

lemma *reducecoeffh-numbound0*: *numbound0 t* ⟹ *numbound0* (*reducecoeffh t g*)
⟨*proof*⟩

lemma *reducecoeff-numbound0*: *numbound0 t* ⟹ *numbound0* (*reducecoeff t*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *numadd-nb*[*simp*]: ⟦ *numbound0 t* ; *numbound0 s*⟧ ⟹ *numbound0* (*numadd* (*t*,*s*))
⟨*proof*⟩

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

lemma *nummul*[*simp*]: ⋀ *i*. *Inum bs* (*nummul t i*) = *Inum bs* (*Mul i t*)
⟨*proof*⟩

lemma *nummul-nb*[*simp*]: ⋀ *i*. *numbound0 t* ⟹ *numbound0* (*nummul t i*)
⟨*proof*⟩

constdefs *numneg* :: *num* ⇒ *num*
numneg t ≡ *nummul t* (− *1*)

constdefs *numsub* :: *num* ⇒ *num* ⇒ *num*
numsub s t ≡ (*if s* = *t then C 0 else numadd* (*s*,*numneg t*))

lemma *numneg*[*simp*]: *Inum bs* (*numneg t*) = *Inum bs* (*Neg t*)
⟨*proof*⟩

lemma *numneg-nb*[*simp*]: *numbound0 t* ⟹ *numbound0* (*numneg t*)
⟨*proof*⟩

lemma *numsub*[*simp*]: *Inum bs* (*numsub a b*) = *Inum bs* (*Sub a b*)
⟨*proof*⟩

lemma *numsub-nb*[*simp*]: ⟦ *numbound0 t* ; *numbound0 s*⟧ ⟹ *numbound0* (*numsub t s*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *simpnum-numbound0*[*simp*]:
numbound0 t ⟹ *numbound0* (*simpnum t*)
⟨*proof*⟩

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

lemma *numadd-nz* : *nozerocoeff a* ⟹ *nozerocoeff b* ⟹ *nozerocoeff* (*numadd* (*a*,*b*))
⟨*proof*⟩

lemma *nummul-nz* : ⋀ *i*. *i*≠*0* ⟹ *nozerocoeff a* ⟹ *nozerocoeff* (*nummul a i*)
⟨*proof*⟩

lemma *numneg-nz* : *nozerocoeff a* ⟹ *nozerocoeff* (*numneg a*)
⟨*proof*⟩

lemma *numsub-nz*: *nozerocoeff a* ⟹ *nozerocoeff b* ⟹ *nozerocoeff* (*numsub a b*)
⟨*proof*⟩

lemma *simpnum-nz*: *nozerocoeff* (*simpnum t*)
⟨*proof*⟩

lemma *maxcoeff-nz*: *nozerocoeff t* ⟹ *maxcoeff t* = *0* ⟹ *t* = *C 0*
⟨*proof*⟩

lemma *numgcd-nz*: **assumes** *nz*: *nozerocoeff t* **and** *g0*: *numgcd t* = *0* **shows** *t* = *C 0*
⟨*proof*⟩

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

lemma *simp-num-pair-ci*:

shows $((\lambda\ (t,n).\ Inum\ bs\ t\ /\ real\ n)\ (simp\text{-}num\text{-}pair\ (t,n))) = ((\lambda\ (t,n).\ Inum\ bs\ t\ /\ real\ n)\ (t,n))$
(**is** *?lhs = ?rhs*)
⟨*proof*⟩

lemma *simp-num-pair-l*: **assumes** *tnb*: *numbound0 t* **and** *np*: $n > 0$ **and** *tn*: *simp-num-pair* $(t,n) = (t',n')$
shows *numbound0* $t' \wedge n' > 0$
⟨*proof*⟩

consts *simpfm* :: *fm* ⇒ *fm*
recdef *simpfm measure fmsize*
simpfm (*And p q*) = *conj* (*simpfm p*) (*simpfm q*)
simpfm (*Or p q*) = *disj* (*simpfm p*) (*simpfm q*)
simpfm (*Imp p q*) = *imp* (*simpfm p*) (*simpfm q*)
simpfm (*Iff p q*) = *iff* (*simpfm p*) (*simpfm q*)
simpfm (*NOT p*) = *not* (*simpfm p*)
simpfm (*Lt a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* $(v < 0)$ *then T else F*
| - ⇒ *Lt* a')
simpfm (*Le a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* $(v \leq 0)$ *then T else F* | - ⇒ *Le* a')
simpfm (*Gt a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* $(v > 0)$ *then T else F* | - ⇒ *Gt* a')
simpfm (*Ge a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* $(v \geq 0)$ *then T else F* | - ⇒ *Ge* a')
simpfm (*Eq a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* $(v = 0)$ *then T else F* | - ⇒ *Eq* a')
simpfm (*NEq a*) = (*let* a' = *simpnum a in case* a' *of C v* ⇒ *if* $(v \neq 0)$ *then T else F* | - ⇒ *NEq* a')
simpfm p = *p*
lemma *simpfm*: *Ifm bs* (*simpfm p*) = *Ifm bs p*
⟨*proof*⟩

lemma *simpfm-bound0*: *bound0 p* ⟹ *bound0* (*simpfm p*)
⟨*proof*⟩

lemma *simpfm-qf*: *qfree p* ⟹ *qfree* (*simpfm p*)
⟨*proof*⟩

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

prep (*E* (*NOT* (*And p q*))) = *disj* (*prep* (*E* (*NOT p*))) (*prep* (*E*(*NOT q*)))
prep (*E* (*NOT* (*Imp p q*))) = *prep* (*E* (*And p* (*NOT q*)))
prep (*E* (*NOT* (*Iff p q*))) = *disj* (*prep* (*E* (*And p* (*NOT q*)))) (*prep* (*E*(*And* (*NOT p*) *q*)))
prep (*E p*) = *E* (*prep p*)
prep (*A* (*And p q*)) = *conj* (*prep* (*A p*)) (*prep* (*A q*))
prep (*A p*) = *prep* (*NOT* (*E* (*NOT p*)))
prep (*NOT* (*NOT p*)) = *prep p*
prep (*NOT* (*And p q*)) = *disj* (*prep* (*NOT p*)) (*prep* (*NOT q*))
prep (*NOT* (*A p*)) = *prep* (*E* (*NOT p*))
prep (*NOT* (*Or p q*)) = *conj* (*prep* (*NOT p*)) (*prep* (*NOT q*))
prep (*NOT* (*Imp p q*)) = *conj* (*prep p*) (*prep* (*NOT q*))
prep (*NOT* (*Iff p q*)) = *disj* (*prep* (*And p* (*NOT q*))) (*prep* (*And* (*NOT p*) *q*))
prep (*NOT p*) = *not* (*prep p*)
prep (*Or p q*) = *disj* (*prep p*) (*prep q*)
prep (*And p q*) = *conj* (*prep p*) (*prep q*)
prep (*Imp p q*) = *prep* (*Or* (*NOT p*) *q*)
prep (*Iff p q*) = *disj* (*prep* (*And p q*)) (*prep* (*And* (*NOT p*) (*NOT q*)))
prep p = *p*
(**hints** *simp add*: *fmsize-pos*)
lemma *prep*: ⋀ *bs*. *Ifm bs* (*prep p*) = *Ifm bs p*
⟨*proof*⟩

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

lemma *qelim-ci*:
assumes *qe-inv*: ∀ *bs p*. *qfree p* ⟶ *qfree* (*qe p*) ∧ (*Ifm bs* (*qe p*) = *Ifm bs* (*E p*))
shows ⋀ *bs*. *qfree* (*qelim p qe*) ∧ (*Ifm bs* (*qelim p qe*) = *Ifm bs p*)
⟨*proof*⟩

consts
plusinf:: *fm* ⇒ *fm*
minusinf:: *fm* ⇒ *fm*
recdef *minusinf measure size*
minusinf (*And p q*) = *conj* (*minusinf p*) (*minusinf q*)
minusinf (*Or p q*) = *disj* (*minusinf p*) (*minusinf q*)
minusinf (*Eq* (*CN 0 c e*)) = *F*
minusinf (*NEq* (*CN 0 c e*)) = *T*

minusinf (*Lt* (*CN* 0 *c* *e*)) = *T*
minusinf (*Le* (*CN* 0 *c* *e*)) = *T*
minusinf (*Gt* (*CN* 0 *c* *e*)) = *F*
minusinf (*Ge* (*CN* 0 *c* *e*)) = *F*
minusinf *p* = *p*

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

consts
isrlfm :: *fm* ⇒ *bool*
recdef *isrlfm* *measure size*
isrlfm (*And* *p* *q*) = (*isrlfm* *p* ∧ *isrlfm* *q*)
isrlfm (*Or* *p* *q*) = (*isrlfm* *p* ∧ *isrlfm* *q*)
isrlfm (*Eq* (*CN* 0 *c* *e*)) = (*c*>0 ∧ *numbound0* *e*)
isrlfm (*NEq* (*CN* 0 *c* *e*)) = (*c*>0 ∧ *numbound0* *e*)
isrlfm (*Lt* (*CN* 0 *c* *e*)) = (*c*>0 ∧ *numbound0* *e*)
isrlfm (*Le* (*CN* 0 *c* *e*)) = (*c*>0 ∧ *numbound0* *e*)
isrlfm (*Gt* (*CN* 0 *c* *e*)) = (*c*>0 ∧ *numbound0* *e*)
isrlfm (*Ge* (*CN* 0 *c* *e*)) = (*c*>0 ∧ *numbound0* *e*)
isrlfm *p* = (*isatom* *p* ∧ (*bound0* *p*))

consts *rsplit0* :: *num* ⇒ *int* × *num*
recdef *rsplit0* *measure num-size*
rsplit0 (*Bound* 0) = (1,*C* 0)
rsplit0 (*Add* *a* *b*) = (*let* (*ca*,*ta*) = *rsplit0* *a* ; (*cb*,*tb*) = *rsplit0* *b*
in (*ca*+*cb*, *Add* *ta* *tb*))
rsplit0 (*Sub* *a* *b*) = *rsplit0* (*Add* *a* (*Neg* *b*))
rsplit0 (*Neg* *a*) = (*let* (*c*,*t*) = *rsplit0* *a* *in* (−*c*,*Neg* *t*))
rsplit0 (*Mul* *c* *a*) = (*let* (*ca*,*ta*) = *rsplit0* *a* *in* (*c*∗*ca*,*Mul* *c* *ta*))
rsplit0 (*CN* 0 *c* *a*) = (*let* (*ca*,*ta*) = *rsplit0* *a* *in* (*c*+*ca*,*ta*))
rsplit0 (*CN* *n* *c* *a*) = (*let* (*ca*,*ta*) = *rsplit0* *a* *in* (*ca*,*CN* *n* *c* *ta*))
rsplit0 *t* = (0,*t*)
lemma *rsplit0*:
shows *Inum* *bs* ((*split* (*CN* 0)) (*rsplit0* *t*)) = *Inum* *bs* *t* ∧ *numbound0* (*snd* (*rsplit0* *t*))
⟨*proof*⟩

definition

$lt :: int \Rightarrow num \Rightarrow fm$
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
$le :: int \Rightarrow num \Rightarrow fm$
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
$gt :: int \Rightarrow num \Rightarrow fm$
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
$ge :: int \Rightarrow num \Rightarrow fm$
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
$eq :: int \Rightarrow num \Rightarrow fm$
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
$neq :: int \Rightarrow num \Rightarrow fm$
where
$neq\ c\ t = (if\ c = 0\ then\ (NEq\ t)\ else\ if\ c > 0\ then\ (NEq\ (CN\ 0\ c\ t))$
$else\ (NEq\ (CN\ 0\ (-c)\ (Neg\ t))))$

lemma *lt*: $numnoabs\ t \Longrightarrow Ifm\ bs\ (split\ lt\ (rsplit0\ t)) = Ifm\ bs\ (Lt\ t) \wedge isrlfm\ (split\ lt\ (rsplit0\ t))$
⟨*proof*⟩

lemma *le*: $numnoabs\ t \Longrightarrow Ifm\ bs\ (split\ le\ (rsplit0\ t)) = Ifm\ bs\ (Le\ t) \wedge isrlfm\ (split\ le\ (rsplit0\ t))$
⟨*proof*⟩

lemma *gt*: $numnoabs\ t \Longrightarrow Ifm\ bs\ (split\ gt\ (rsplit0\ t)) = Ifm\ bs\ (Gt\ t) \wedge isrlfm\ (split\ gt\ (rsplit0\ t))$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *eq*: *numnoabs t* ⟹ *Ifm bs* (*split eq* (*rsplit0 t*)) = *Ifm bs* (*Eq t*) ∧ *isrlfm* (*split eq* (*rsplit0 t*))
⟨*proof*⟩

lemma *neq*: *numnoabs t* ⟹ *Ifm bs* (*split neq* (*rsplit0 t*)) = *Ifm bs* (*NEq t*) ∧ *isrlfm* (*split neq* (*rsplit0 t*))
⟨*proof*⟩

lemma *conj-lin*: *isrlfm p* ⟹ *isrlfm q* ⟹ *isrlfm* (*conj p q*)
⟨*proof*⟩
lemma *disj-lin*: *isrlfm p* ⟹ *isrlfm q* ⟹ *isrlfm* (*disj p q*)
⟨*proof*⟩

consts *rlfm* :: *fm* ⇒ *fm*
recdef *rlfm measure fmsize*
rlfm (*And p q*) = *conj* (*rlfm p*) (*rlfm q*)
rlfm (*Or p q*) = *disj* (*rlfm p*) (*rlfm q*)
rlfm (*Imp p q*) = *disj* (*rlfm* (*NOT p*)) (*rlfm q*)
rlfm (*Iff p q*) = *disj* (*conj* (*rlfm p*) (*rlfm q*)) (*conj* (*rlfm* (*NOT p*)) (*rlfm* (*NOT q*)))
rlfm (*Lt a*) = *split lt* (*rsplit0 a*)
rlfm (*Le a*) = *split le* (*rsplit0 a*)
rlfm (*Gt a*) = *split gt* (*rsplit0 a*)
rlfm (*Ge a*) = *split ge* (*rsplit0 a*)
rlfm (*Eq a*) = *split eq* (*rsplit0 a*)
rlfm (*NEq a*) = *split neq* (*rsplit0 a*)
rlfm (*NOT* (*And p q*)) = *disj* (*rlfm* (*NOT p*)) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Or p q*)) = *conj* (*rlfm* (*NOT p*)) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Imp p q*)) = *conj* (*rlfm p*) (*rlfm* (*NOT q*))
rlfm (*NOT* (*Iff p q*)) = *disj* (*conj*(*rlfm p*) (*rlfm*(*NOT q*))) (*conj*(*rlfm*(*NOT p*)) (*rlfm q*))
rlfm (*NOT* (*NOT p*)) = *rlfm p*
rlfm (*NOT T*) = *F*
rlfm (*NOT F*) = *T*
rlfm (*NOT* (*Lt a*)) = *rlfm* (*Ge a*)
rlfm (*NOT* (*Le a*)) = *rlfm* (*Gt a*)
rlfm (*NOT* (*Gt a*)) = *rlfm* (*Le a*)
rlfm (*NOT* (*Ge a*)) = *rlfm* (*Lt a*)
rlfm (*NOT* (*Eq a*)) = *rlfm* (*NEq a*)
rlfm (*NOT* (*NEq a*)) = *rlfm* (*Eq a*)
rlfm p = *p* (**hints** *simp add*: *fmsize-pos*)

lemma *rlfm-I*:
assumes *qfp*: *qfree p*
shows (*Ifm bs* (*rlfm p*) = *Ifm bs p*) ∧ *isrlfm* (*rlfm p*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *rplusinf-inf*:
assumes *lp*: *isrlfm p*
shows ∃ *z*. ∀ *x* > *z*. *Ifm* (*x*#*bs*) (*plusinf p*) = *Ifm* (*x*#*bs*) *p* (**is** ∃ *z*. ∀ *x*. *?P z x p*)
⟨*proof*⟩

lemma *rminusinf-bound0*:
assumes *lp*: *isrlfm p*
shows *bound0* (*minusinf p*)
⟨*proof*⟩

lemma *rplusinf-bound0*:
assumes *lp*: *isrlfm p*
shows *bound0* (*plusinf p*)
⟨*proof*⟩

lemma *rminusinf-ex*:
assumes *lp*: *isrlfm p*
and *ex*: *Ifm* (*a*#*bs*) (*minusinf p*)
shows ∃ *x*. *Ifm* (*x*#*bs*) *p*
⟨*proof*⟩

lemma *rplusinf-ex*:
assumes *lp*: *isrlfm p*
and *ex*: *Ifm* (*a*#*bs*) (*plusinf p*)
shows ∃ *x*. *Ifm* (*x*#*bs*) *p*
⟨*proof*⟩

consts
uset:: *fm* ⇒ (*num* × *int*) *list*
usubst :: *fm* ⇒ (*num* × *int*) ⇒ *fm*
recdef *uset measure size*
uset (*And p q*) = (*uset p* @ *uset q*)
uset (*Or p q*) = (*uset p* @ *uset q*)
uset (*Eq* (*CN 0 c e*)) = [(*Neg e*,*c*)]
uset (*NEq* (*CN 0 c e*)) = [(*Neg e*,*c*)]
uset (*Lt* (*CN 0 c e*)) = [(*Neg e*,*c*)]
uset (*Le* (*CN 0 c e*)) = [(*Neg e*,*c*)]
uset (*Gt* (*CN 0 c e*)) = [(*Neg e*,*c*)]
uset (*Ge* (*CN 0 c e*)) = [(*Neg e*,*c*)]
uset p = []
recdef *usubst measure size*

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

lemma *usubst-I*: **assumes** *lp*: *isrlfm p*
and *np*: *real n* > *0* **and** *nbt*: *numbound0 t*
shows (*Ifm* (*x*#*bs*) (*usubst p* (*t*,*n*)) = *Ifm* (((*Inum* (*x*#*bs*) *t*)/(*real n*))#*bs*) *p*)
∧ *bound0* (*usubst p* (*t*,*n*)) (**is** (*?I x* (*usubst p* (*t*,*n*)) = *?I ?u p*) ∧ *?B p* **is** (- = *?I*
(*?t*/*?n*) *p*) ∧ - **is** (- = *?I* (*?N x t* /-) *p*) ∧ -)
⟨*proof*⟩

lemma *uset-l*:
assumes *lp*: *isrlfm p*
shows ∀ (*t*,*k*) ∈ *set* (*uset p*). *numbound0 t* ∧ *k* >*0*
⟨*proof*⟩

lemma *rminusinf-uset*:
assumes *lp*: *isrlfm p*
and *nmi*: ¬ (*Ifm* (*a*#*bs*) (*minusinf p*)) (**is** ¬ (*Ifm* (*a*#*bs*) (*?M p*)))
and *ex*: *Ifm* (*x*#*bs*) *p* (**is** *?I x p*)
shows ∃ (*s*,*m*) ∈ *set* (*uset p*). *x* ≥ *Inum* (*a*#*bs*) *s* / *real m* (**is** ∃ (*s*,*m*) ∈ *?U*
p. *x* ≥ *?N a s* / *real m*)
⟨*proof*⟩

lemma *rplusinf-uset*:
assumes *lp*: *isrlfm p*
and *nmi*: ¬ (*Ifm* (*a*#*bs*) (*plusinf p*)) (**is** ¬ (*Ifm* (*a*#*bs*) (*?M p*)))
and *ex*: *Ifm* (*x*#*bs*) *p* (**is** *?I x p*)
shows ∃ (*s*,*m*) ∈ *set* (*uset p*). *x* ≤ *Inum* (*a*#*bs*) *s* / *real m* (**is** ∃ (*s*,*m*) ∈ *?U*
p. *x* ≤ *?N a s* / *real m*)
⟨*proof*⟩

lemma *lin-dense*:
assumes *lp*: *isrlfm p*
and *noS*: ∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ (λ (*t*,*n*). *Inum* (*x*#*bs*) *t* / *real n*) ‘ *set*
(*uset p*)
(**is** ∀ *t*. - ∧ - ⟶ *t* ∉ (λ (*t*,*n*). *?N x t* / *real n*) ‘ (*?U p*))
and *lx*: *l* < *x* **and** *xu*:*x* < *u* **and** *px*: *Ifm* (*x*#*bs*) *p*
and *ly*: *l* < *y* **and** *yu*: *y* < *u*
shows *Ifm* (*y*#*bs*) *p*
⟨*proof*⟩

lemma *finite-set-intervals*:

assumes *px*: $P\ (x::real)$
and *lx*: $l \leq x$ **and** *xu*: $x \leq u$
and *linS*: $l \in S$ **and** *uinS*: $u \in S$
and *fS*:*finite* S **and** *lS*: $\forall\ x \in S.\ l \leq x$ **and** *Su*: $\forall\ x \in S.\ x \leq u$
shows $\exists\ a \in S.\ \exists\ b \in S.\ (\forall\ y.\ a < y \wedge y < b \longrightarrow y \notin S) \wedge a \leq x \wedge x \leq b \wedge P\ x$
⟨*proof*⟩

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*⟩

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*⟩

theorem *fr-eq*:
assumes *lp*: *isrlfm p*
shows (∃ *x*. *Ifm* (*x*#*bs*) *p*) = ((*Ifm* (*x*#*bs*) (*minusinf p*)) ∨ (*Ifm* (*x*#*bs*) (*plusinf p*)) ∨ (∃ (*t*,*n*) ∈ *set* (*uset p*). ∃ (*s*,*m*) ∈ *set* (*uset p*). *Ifm* ((((*Inum* (*x*#*bs*) *t*)/ *real n* + (*Inum* (*x*#*bs*) *s*) / *real m*) /*2*)#*bs*) *p*))
(**is** (∃ *x*. *?I x p*) = (*?M* ∨ *?P* ∨ *?F*) **is** *?E* = *?D*)
⟨*proof*⟩

lemma *fr-equsubst*:
assumes *lp*: *isrlfm p*
shows (∃ *x*. *Ifm* (*x*#*bs*) *p*) = ((*Ifm* (*x*#*bs*) (*minusinf p*)) ∨ (*Ifm* (*x*#*bs*) (*plusinf p*)) ∨ (∃ (*t*,*k*) ∈ *set* (*uset p*). ∃ (*s*,*l*) ∈ *set* (*uset p*). *Ifm* (*x*#*bs*) (*usubst p* (*Add*(*Mul l t*) (*Mul k s*) , *2*∗*k*∗*l*))))
(**is** (∃ *x*. *?I x p*) = (*?M* ∨ *?P* ∨ *?F*) **is** *?E* = *?D*)
⟨*proof*⟩

constdefs *ferrack*:: *fm* ⇒ *fm*
ferrack p ≡ (*let p*′ = *rlfm* (*simpfm p*); *mp* = *minusinf p*′; *pp* = *plusinf p*′
in if (*mp* = *T* ∨ *pp* = *T*) *then T else*

(*let* U = *remdps*(*map simp-num-pair*
(*map* (λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , 2∗*n*∗*m*))
(*alluopairs* (*uset p*′))))
in decr (*disj mp* (*disj pp* (*evaldjf* (*simpfm o* (*usubst p*′)) *U*)))))

lemma *uset-cong-aux*:
assumes *Ul*: ∀ (*t*,*n*) ∈ *set U*. *numbound0 t* ∧ *n* >*0*
shows ((λ (*t*,*n*). *Inum* (*x*#*bs*) *t* /*real n*) ‘ (*set* (*map* (λ ((*t*,*n*),(*s*,*m*)). (*Add* (*Mul m t*) (*Mul n s*) , 2∗*n*∗*m*)) (*alluopairs U*)))) = ((λ ((*t*,*n*),(*s*,*m*)). (*Inum* (*x*#*bs*) *t* /*real n* + *Inum* (*x*#*bs*) *s* /*real m*)/*2*) ‘ (*set U* × *set U*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *uset-cong*:
assumes *lp*: *isrlfm p*
and *UU*′: ((λ (*t*,*n*). *Inum* (*x*#*bs*) *t* /*real n*) ‘ *U*′) = ((λ ((*t*,*n*),(*s*,*m*)). (*Inum* (*x*#*bs*) *t* /*real n* + *Inum* (*x*#*bs*) *s* /*real m*)/*2*) ‘ (*U* × *U*)) (**is** *?f* ‘ *U*′ = *?g* ‘ (*U*×*U*))
and *U*: ∀ (*t*,*n*) ∈ *U*. *numbound0 t* ∧ *n* > *0*
and *U*′: ∀ (*t*,*n*) ∈ *U*′. *numbound0 t* ∧ *n* > *0*
shows (∃ (*t*,*n*) ∈ *U*. ∃ (*s*,*m*) ∈ *U*. *Ifm* (*x*#*bs*) (*usubst p* (*Add* (*Mul m t*) (*Mul n s*),2∗*n*∗*m*))) = (∃ (*t*,*n*) ∈ *U*′. *Ifm* (*x*#*bs*) (*usubst p* (*t*,*n*)))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *ferrack*:
assumes *qf*: *qfree p*
shows *qfree* (*ferrack p*) ∧ ((*Ifm bs* (*ferrack p*)) = (∃ *x*. *Ifm* (*x*#*bs*) *p*))
(**is** - ∧ (*?rhs* = *?lhs*))
⟨*proof*⟩

constdefs *linrqe*:: *fm* ⇒ *fm*
linrqe ≡ (λ *p*. *qelim* (*prep p*) *ferrack*)

theorem *linrqe*: (*Ifm bs* (*linrqe p*) = *Ifm bs p*) ∧ *qfree* (*linrqe p*)
⟨*proof*⟩

definition
ferrack-test :: *unit* ⇒ *fm*
where
ferrack-test u = *linrqe* (*A* (*A* (*Imp* (*Lt* (*Sub* (*Bound 1*) (*Bound 0*)))
(*E* (*Eq* (*Sub* (*Add* (*Bound 0*) (*Bound 2*)) (*Bound 1*)))))))

export-code *linrqe ferrack-test* **in** *SML* **module-name** *Ferrack*

⟨*ML*⟩

end