Miscellaneous HOL-Complex Examples

October 1, 2005

Contents

1 Binary arithmetic examples

theory *BinEx*
imports *Complex-Main*
begin

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

1.1 Real Arithmetic

1.1.1 Addition

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

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

1.1.2 Negation

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

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

1.1.3 Multiplication

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

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

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

1.1.4 Inequalities

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

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

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

lemma $(13557456::real) < 18678654$

by *simp*

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

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

1.1.5 Powers

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

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

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

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

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

1.1.6 Tests

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

by *arith*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.2 Complex Arithmetic

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

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

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

lemmas *distrib* = *left-distrib right-distrib left-diff-distrib right-diff-distrib*

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

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

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

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

No powers (not supported yet)

end

2 Square roots of primes are irrational

theory *Sqrt*

```
imports Primes Complex-Main
begin
```

2.1 Preliminaries

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

```
constdefs
  rationals :: real set    (ℚ)
  ℚ ≡ {x. ∃ m n. n ≠ 0 ∧ |x| = real (m::nat) / real (n::nat)}

theorem rationals-rep: x ∈ ℚ ⟹
  ∃ m n. n ≠ 0 ∧ |x| = real m / real n ∧ gcd (m, n) = 1
proof −
  assume x ∈ ℚ
  then obtain m n :: nat where
      n: n ≠ 0 and x-rat: |x| = real m / real n
    by (unfold rationals-def) blast
  let ?gcd = gcd (m, n)
  from n have gcd: ?gcd ≠ 0 by (simp add: gcd-zero)
  let ?k = m div ?gcd
  let ?l = n div ?gcd
  let ?gcd′ = gcd (?k, ?l)
  have ?gcd dvd m .. then have gcd-k: ?gcd ∗ ?k = m
    by (rule dvd-mult-div-cancel)
  have ?gcd dvd n .. then have gcd-l: ?gcd ∗ ?l = n
    by (rule dvd-mult-div-cancel)

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

```
lemma [elim?]: r ∈ ℚ ⟹
  (⋀m n. n ≠ 0 ⟹ |r| = real m / real n ⟹ gcd (m, n) = 1 ⟹ C)
    ⟹ C
  using rationals-rep by blast
```

2.2 Main theorem

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

```
theorem sqrt-prime-irrational: prime p ⟹ sqrt (real p) ∉ ℚ
proof
  assume p-prime: prime p
  then have p: 1 < p by (simp add: prime-def)
  assume sqrt (real p) ∈ ℚ
  then obtain m n where
      n: n ≠ 0 and sqrt-rat: |sqrt (real p)| = real m / real n
    and gcd: gcd (m, n) = 1 ..
  have eq: m^2 = p ∗ n^2
  proof −
    from n and sqrt-rat have real m = |sqrt (real p)| ∗ real n by simp
    then have real (m^2) = (sqrt (real p))^2 ∗ real (n^2)
      by (auto simp add: power2-eq-square)
    also have (sqrt (real p))^2 = real p by simp
    also have . . . ∗ real (n^2) = real (p ∗ n^2) by simp
    finally show ?thesis ..
  qed
  have p dvd m ∧ p dvd n
  proof
    from eq have p dvd m^2 ..
    with p-prime show p dvd m by (rule prime-dvd-power-two)
    then obtain k where m = p ∗ k ..
    with eq have p ∗ n^2 = p^2 ∗ k^2 by (auto simp add: power2-eq-square mult-ac)
    with p have n^2 = p ∗ k^2 by (simp add: power2-eq-square)
    then have p dvd n^2 ..
    with p-prime show p dvd n by (rule prime-dvd-power-two)
  qed
  then have p dvd gcd (m, n) ..
  with gcd have p dvd 1 by simp
  then have p ≤ 1 by (simp add: dvd-imp-le)
  with p show False by simp
qed

corollary sqrt (real (2::nat)) ∉ ℚ
  by (rule sqrt-prime-irrational) (rule two-is-prime)
```

2.3 Variations

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

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

end

3 Square roots of primes are irrational (script version)

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

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

3.1 Preliminaries

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

lemma *prime-dvd-other-side*:

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

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

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

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

3.2 The set of rational numbers

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

3.3 Main theorem

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

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

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

end

4 The Nonstandard Primes as an Extension of the Prime Numbers

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

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

constdefs
hdvd :: [*hypnat*, *hypnat*] => *bool* (**infixl** *hdvd 50*)
(*M*::*hypnat*) *hdvd N* == (**p2** (*op dvd*)) *M N*

declare *hdvd-def* [*transfer-unfold*]

constdefs
starprime :: *hypnat set*
starprime == (**s** {*p*. *prime p*})

declare *starprime-def* [*transfer-unfold*]

constdefs
choicefun :: ′*a set* => ′*a*
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*})

A "choice" theorem for ultrafilters, like almost everywhere quantification

lemma *UF-choice*: {*n*. ∃ *m*. *Q n m*} : *FreeUltrafilterNat*
==> ∃ *f*. {*n*. *Q n* (*f n*)} : *FreeUltrafilterNat*
apply (*rule-tac x* = %*n*. (@*x*. *Q n x*) **in** *exI*)
apply (*ultra*, *rule someI*, *auto*)
done

lemma *UF-if*: ({*n*. *P n*} : *FreeUltrafilterNat* −−> {*n*. *Q n*} : *FreeUltrafilterNat*)
=
({*n*. *P n* −−> *Q n*} : *FreeUltrafilterNat*)
apply *auto*
apply *ultra*+
done

lemma *UF-conj*: $(\{n.\ P\ n\} : \mathit{FreeUltrafilterNat}\ \&\ \{n.\ Q\ n\} : \mathit{FreeUltrafilterNat})$ $=$
$(\{n.\ P\ n\ \&\ Q\ n\} : \mathit{FreeUltrafilterNat})$
apply *auto*
apply *ultra+*
done

lemma *UF-choice-ccontr*: $(\forall f.\ \{n.\ Q\ n\ (f\ n)\} : \mathit{FreeUltrafilterNat}) =$
$(\{n.\ \forall m.\ Q\ n\ m\} : \mathit{FreeUltrafilterNat})$
apply *auto*
prefer *2* **apply** *ultra*
apply (*rule ccontr*)
apply (*rule contrapos-np*)
apply (*erule-tac* [*2*] *asm-rl*)
apply (*simp* (*no-asm*) *add*: *FreeUltrafilterNat-Compl-iff1 Collect-neg-eq* [*symmetric*])
apply (*rule UF-choice*, *ultra*)
done

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

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

lemma *lemma-hypnat-P-EX*: $(\exists (x{::}hypnat).\ P\ x) = (\exists f.\ P\ (star\text{-}n\ f))$
apply *auto*
apply (*rule-tac* $x = x$ **in** *star-cases*, *auto*)
done

lemma *lemma-hypnat-P-ALL*: $(\forall (x{::}hypnat).\ P\ x) = (\forall f.\ P\ (star\text{-}n\ f))$
apply *auto*
apply (*rule-tac* $x = x$ **in** *star-cases*, *auto*)
done

lemma *hdvd*:
$(star\text{-}n\ X\ hdvd\ star\text{-}n\ Y) =$
$(\{n.\ X\ n\ dvd\ Y\ n\} : \mathit{FreeUltrafilterNat})$
by (*simp add*: *hdvd-def starP2*)

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

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

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

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

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

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

lemma *prime-two*: *prime 2*
apply (*unfold prime-def*, *auto*)
apply (*frule dvd-imp-le*)
apply (*auto dest*: *dvd-0-left*)
apply (*case-tac m*, *simp*, *arith*)
done
declare *prime-two* [*simp*]

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

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

lemma *NatStar-hypnat-of-nat*: *finite A* ==> ∗*s*∗ *A* = *hypnat-of-nat* ' *A*
apply (*rule-tac P* = %*x*. ∗*s*∗ *x* = *hypnat-of-nat* ' *x* **in** *finite-induct*)
apply *auto*
done

lemma *FreeUltrafilterNat-singleton-not-mem*: $\{x\} \notin$ *FreeUltrafilterNat*
by (*auto intro*!: *FreeUltrafilterNat-finite*)
declare *FreeUltrafilterNat-singleton-not-mem* [*simp*]

4.1 Another characterization of infinite set of natural numbers

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

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

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

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

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

lemma *finite-nat-set-bounded-iff2*: *finite N* = (∃ *n*. (∀ *i* ∈ *N*. *i*<=(*n*::*nat*)))
by (*blast intro*: *finite-nat-set-bounded2 bounded-nat-set-is-finite2*)

lemma *not-finite-nat-set-iff2*: (~ *finite N*) = (∀ *n*. ∃ *i* ∈ *N*. *n* < (*i*::*nat*))
by (*auto simp add*: *finite-nat-set-bounded-iff2 le-def*)

4.2 An injective function cannot define an embedded natural number

lemma *lemma-infinite-set-singleton*: ∀ *m n*. *m* ≠ *n* −−> *f n* ≠ *f m*
==> {*n*. *f n* = *N*} = {} | (∃ *m*. {*n*. *f n* = *N*} = {*m*})
apply *auto*
apply (*drule-tac x* = *x* **in** *spec*, *auto*)

apply (*subgoal-tac* $\forall n. (f\ n = f\ x) = (x = n)$)
apply *auto*
done

lemma *inj-fun-not-hypnat-in-SHNat*: *inj* (*f*::*nat*=>*nat*) ==> *star-n f* $\notin$ *Nats*
apply (*auto simp add*: *SHNat-eq hypnat-of-nat-eq star-n-eq-iff*)
apply (*subgoal-tac* $\forall m\ n.\ m \neq n$ −−> $f\ n \neq f\ m$, *auto*)
apply (*drule-tac* [*2*] *injD*)
prefer *2* **apply** *assumption*
apply (*drule-tac* $N = N$ **in** *lemma-infinite-set-singleton*, *auto*)
done

lemma *range-subset-mem-starsetNat*:
  range f <= *A* ==> *star-n f* $\in$ *s* *A*
apply (*simp add*: *starset-def star-of-def Iset-star-n*)
apply (*subgoal-tac* $\forall n.\ f\ n \in A$, *auto*)
done

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

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

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

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

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

lemma *inj-injf-max*: $\forall x.\ \exists y \in E.\ x < y ==> inj\ (\%n.\ injf\text{-}max\ n\ E)$
apply (*rule inj-onI*)
apply (*rule ccontr*, *auto*)
apply (*drule injf-max-order-preserving2*)
apply (*cut-tac* $m = x$ **and** $n = y$ **in** *less-linear*, *auto*)
apply (*auto dest*!: *spec*)
done

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

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

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

lemma *starsetNat-eq-hypnat-of-nat-image-finite*: $*s*\ A =\ hypnat\text{-}of\text{-}nat\ `\ A ==>$
$finite\ A$
apply (*rule ccontr*)
apply (*auto dest*: *hypnat-infinite-has-nonstandard*)
done

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

by (*blast intro*!: *starsetNat-eq-hypnat-of-nat-image-finite NatStar-hypnat-of-nat*)

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

4.3 Existence of Infinitely Many Primes: a Nonstandard Proof

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

lemma *lemma-not-dvd-hypnat-one2*: ∃ *n* ∈ − {*0*}. ~ *hypnat-of-nat n hdvd 1*
apply (*cut-tac lemma-not-dvd-hypnat-one*)
apply (*auto simp del*: *lemma-not-dvd-hypnat-one*)
done
declare *lemma-not-dvd-hypnat-one2* [*simp*]

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

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

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

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

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

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

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

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

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

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

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

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

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

end

5 Big O notation – continued

```
theory BigO-Complex
imports BigO Complex
begin
```

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

```
lemma bigo-LIMSEQ1: f =o O(g) ==> g ----> 0 ==> f ----> 0
  apply (simp add: LIMSEQ-def bigo-alt-def)
  apply clarify
  apply (drule-tac x = r / c in spec)
  apply (drule mp)
  apply (erule divide-pos-pos)
  apply assumption
  apply clarify
  apply (rule-tac x = no in exI)
  apply (rule allI)
  apply (drule-tac x = n in spec)+
  apply (rule impI)
  apply (drule mp)
  apply assumption
  apply (rule order-le-less-trans)
  apply assumption
  apply (rule order-less-le-trans)
  apply (subgoal-tac c * abs(g n) < c * (r / c))
  apply assumption
  apply (erule mult-strict-left-mono)
  apply assumption
  apply simp
done

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

end
```