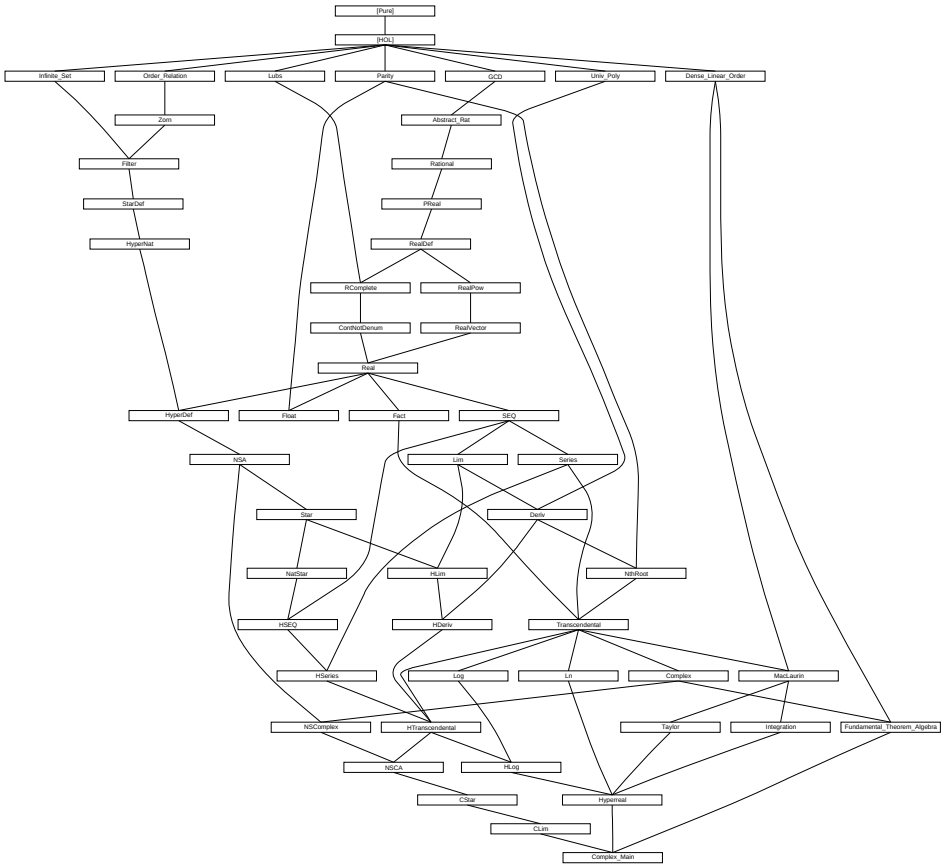
Isabelle/HOL-Complex — Higher-Order Logic with Complex Numbers

June 8, 2008

Contents



1 Lubs: Definitions of Upper Bounds and Least Upper Bounds

theory *Lubs*
imports *Main*
begin

Thanks to suggestions by James Margetson

definition
setle :: ['*a set*, '*a*::*ord*] => *bool* (**infixl** *<= 70) **where**
S *<= *x* = (*ALL y*: *S*. *y* <= *x*)

definition
setge :: ['*a*::*ord*, '*a set*] => *bool* (**infixl** <=* 70) **where**
x <=* *S* = (*ALL y*: *S*. *x* <= *y*)

definition
leastP :: ['*a* =>*bool*,'*a*::*ord*] => *bool* **where**
leastP P x = (*P x* & *x* <=* *Collect P*)

definition
isUb :: ['*a set*, '*a set*, '*a*::*ord*] => *bool* **where**
isUb R S x = (*S* *<= *x* & *x*: *R*)

definition
isLub :: ['*a set*, '*a set*, '*a*::*ord*] => *bool* **where**
isLub R S x = *leastP* (*isUb R S*) *x*

definition
ubs :: ['*a set*, '*a*::*ord set*] => '*a set* **where**
ubs R S = *Collect* (*isUb R S*)

1.1 Rules for the Relations *<= and <=*

lemma *setleI*: *ALL y*: *S*. *y* <= *x* ==> *S* *<= *x*
by (*simp add*: *setle-def*)

lemma *setleD*: [| *S* *<= *x*; *y*: *S* |] ==> *y* <= *x*
by (*simp add*: *setle-def*)

lemma *setgeI*: *ALL y*: *S*. *x*<= *y* ==> *x* <=* *S*
by (*simp add*: *setge-def*)

lemma *setgeD*: [| *x* <=* *S*; *y*: *S* |] ==> *x* <= *y*
by (*simp add*: *setge-def*)

1.2 Rules about the Operators *leastP*, *ub* and *lub*

lemma *leastPD1*: *leastP P x* ==> *P x*

by (*simp add*: *leastP-def*)

lemma *leastPD2*: *leastP P x ==> x <=∗ Collect P*
by (*simp add*: *leastP-def*)

lemma *leastPD3*: [| *leastP P x*; *y*: *Collect P* |] ==> *x <= y*
by (*blast dest*!: *leastPD2 setgeD*)

lemma *isLubD1*: *isLub R S x ==> S ∗<= x*
by (*simp add*: *isLub-def isUb-def leastP-def*)

lemma *isLubD1a*: *isLub R S x ==> x*: *R*
by (*simp add*: *isLub-def isUb-def leastP-def*)

lemma *isLub-isUb*: *isLub R S x ==> isUb R S x*
apply (*simp add*: *isUb-def*)
apply (*blast dest*: *isLubD1 isLubD1a*)
done

lemma *isLubD2*: [| *isLub R S x*; *y* : *S* |] ==> *y <= x*
by (*blast dest*!: *isLubD1 setleD*)

lemma *isLubD3*: *isLub R S x ==> leastP*(*isUb R S*) *x*
by (*simp add*: *isLub-def*)

lemma *isLubI1*: *leastP*(*isUb R S*) *x ==> isLub R S x*
by (*simp add*: *isLub-def*)

lemma *isLubI2*: [| *isUb R S x*; *x <=∗ Collect* (*isUb R S*) |] ==> *isLub R S x*
by (*simp add*: *isLub-def leastP-def*)

lemma *isUbD*: [| *isUb R S x*; *y* : *S* |] ==> *y <= x*
by (*simp add*: *isUb-def setle-def*)

lemma *isUbD2*: *isUb R S x ==> S ∗<= x*
by (*simp add*: *isUb-def*)

lemma *isUbD2a*: *isUb R S x ==> x*: *R*
by (*simp add*: *isUb-def*)

lemma *isUbI*: [| *S ∗<= x*; *x*: *R* |] ==> *isUb R S x*
by (*simp add*: *isUb-def*)

lemma *isLub-le-isUb*: [| *isLub R S x*; *isUb R S y* |] ==> *x <= y*
apply (*simp add*: *isLub-def*)
apply (*blast intro*!: *leastPD3*)
done

lemma *isLub-ubs*: *isLub R S x ==> x <=∗ ubs R S*

apply (*simp add*: *ubs-def isLub-def*)
apply (*erule leastPD2*)
done

end

2 GCD: The Greatest Common Divisor

theory *GCD*
imports *ATP-Linkup*
begin

See [**?**].

2.1 Specification of GCD on nats

definition
is-gcd :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where** — *gcd* as a relation
is-gcd p m n ⟷ *p dvd m* ∧ *p dvd n* ∧
(∀ *d*. *d dvd m* ⟶ *d dvd n* ⟶ *d dvd p*)

Uniqueness

lemma *is-gcd-unique*: *is-gcd m a b* ⟹ *is-gcd n a b* ⟹ *m* = *n*
by (*simp add*: *is-gcd-def*) (*blast intro*: *dvd-anti-sym*)

Connection to divides relation

lemma *is-gcd-dvd*: *is-gcd m a b* ⟹ *k dvd a* ⟹ *k dvd b* ⟹ *k dvd m*
by (*auto simp add*: *is-gcd-def*)

Commutativity

lemma *is-gcd-commute*: *is-gcd k m n* = *is-gcd k n m*
by (*auto simp add*: *is-gcd-def*)

2.2 GCD on nat by Euclid's algorithm

fun
gcd :: *nat* × *nat* => *nat*
where
gcd (*m*, *n*) = (*if n* = *0 then m else gcd* (*n*, *m mod n*))

lemma *gcd-induct*:
fixes *m n* :: *nat*
assumes ⋀*m*. *P m 0*
and ⋀*m n*. *0* < *n* ⟹ *P n* (*m mod n*) ⟹ *P m n*
shows *P m n*
apply (*rule gcd.induct* [*of split P* (*m*, *n*), *unfolded Product-Type.split*])

apply (*case-tac* $n = 0$)
apply *simp-all*
using *assms* **apply** *simp-all*
done

lemma *gcd-0* [*simp*]: *gcd* $(m, 0) = m$
by *simp*

lemma *gcd-0-left* [*simp*]: *gcd* $(0, m) = m$
by *simp*

lemma *gcd-non-0*: $n > 0 \Longrightarrow$ *gcd* $(m, n) =$ *gcd* $(n, m \bmod n)$
by *simp*

lemma *gcd-1* [*simp*]: *gcd* $(m,$ *Suc* $0) = 1$
by *simp*

declare *gcd.simps* [*simp del*]

gcd (m, n) divides m and n. The conjunctions don't seem provable separately.

lemma *gcd-dvd1* [*iff*]: *gcd* (m, n) *dvd* m
and *gcd-dvd2* [*iff*]: *gcd* (m, n) *dvd* n
apply (*induct m n rule*: *gcd-induct*)
apply (*simp-all add*: *gcd-non-0*)
apply (*blast dest*: *dvd-mod-imp-dvd*)
done

Maximality: for all m, n, k naturals, if k divides m and k divides n then k divides *gcd* (m, n).

lemma *gcd-greatest*: k *dvd* $m \Longrightarrow k$ *dvd* $n \Longrightarrow k$ *dvd gcd* (m, n)
by (*induct m n rule*: *gcd-induct*) (*simp-all add*: *gcd-non-0 dvd-mod*)

Function gcd yields the Greatest Common Divisor.

lemma *is-gcd*: *is-gcd* (*gcd* $(m, n))$ m n
by (*simp add*: *is-gcd-def gcd-greatest*)

2.3 Derived laws for GCD

lemma *gcd-greatest-iff* [*iff*]: k *dvd gcd* $(m, n) \longleftrightarrow k$ *dvd* $m \wedge k$ *dvd* n
by (*blast intro*!: *gcd-greatest intro*: *dvd-trans*)

lemma *gcd-zero*: *gcd* $(m, n) = 0 \longleftrightarrow m = 0 \wedge n = 0$
by (*simp only*: *dvd-0-left-iff* [*symmetric*] *gcd-greatest-iff*)

lemma *gcd-commute*: *gcd* $(m, n) =$ *gcd* (n, m)
apply (*rule is-gcd-unique*)

apply (*rule is-gcd*)
apply (*subst is-gcd-commute*)
apply (*simp add*: *is-gcd*)
done

lemma *gcd-assoc*: *gcd* (*gcd* (*k*, *m*), *n*) = *gcd* (*k*, *gcd* (*m*, *n*))
apply (*rule is-gcd-unique*)
apply (*rule is-gcd*)
apply (*simp add*: *is-gcd-def*)
apply (*blast intro*: *dvd-trans*)
done

lemma *gcd-1-left* [*simp*]: *gcd* (*Suc 0*, *m*) = *1*
by (*simp add*: *gcd-commute*)

Multiplication laws

lemma *gcd-mult-distrib2*: *k* ∗ *gcd* (*m*, *n*) = *gcd* (*k* ∗ *m*, *k* ∗ *n*)
— [**?**, page 27]
apply (*induct m n rule*: *gcd-induct*)
apply *simp*
apply (*case-tac k* = *0*)
apply (*simp-all add*: *mod-geq gcd-non-0 mod-mult-distrib2*)
done

lemma *gcd-mult* [*simp*]: *gcd* (*k*, *k* ∗ *n*) = *k*
apply (*rule gcd-mult-distrib2* [*of k 1 n*, *simplified*, *symmetric*])
done

lemma *gcd-self* [*simp*]: *gcd* (*k*, *k*) = *k*
apply (*rule gcd-mult* [*of k 1*, *simplified*])
done

lemma *relprime-dvd-mult*: *gcd* (*k*, *n*) = *1* ==> *k dvd m* ∗ *n* ==> *k dvd m*
apply (*insert gcd-mult-distrib2* [*of m k n*])
apply *simp*
apply (*erule-tac t* = *m* **in** *ssubst*)
apply *simp*
done

lemma *relprime-dvd-mult-iff*: *gcd* (*k*, *n*) = *1* ==> (*k dvd m* ∗ *n*) = (*k dvd m*)
apply (*blast intro*: *relprime-dvd-mult dvd-trans*)
done

lemma *gcd-mult-cancel*: *gcd* (*k*, *n*) = *1* ==> *gcd* (*k* ∗ *m*, *n*) = *gcd* (*m*, *n*)
apply (*rule dvd-anti-sym*)
apply (*rule gcd-greatest*)
apply (*rule-tac n* = *k* **in** *relprime-dvd-mult*)
apply (*simp add*: *gcd-assoc*)
apply (*simp add*: *gcd-commute*)

apply (*simp-all add*: *mult-commute*)
apply (*blast intro*: *dvd-trans*)
done

Addition laws

lemma *gcd-add1* [*simp*]: *gcd* (*m* + *n*, *n*) = *gcd* (*m*, *n*)
apply (*case-tac n* = *0*)
apply (*simp-all add*: *gcd-non-0*)
done

lemma *gcd-add2* [*simp*]: *gcd* (*m*, *m* + *n*) = *gcd* (*m*, *n*)
proof −
have *gcd* (*m*, *m* + *n*) = *gcd* (*m* + *n*, *m*) **by** (*rule gcd-commute*)
also have ... = *gcd* (*n* + *m*, *m*) **by** (*simp add*: *add-commute*)
also have ... = *gcd* (*n*, *m*) **by** *simp*
also have ... = *gcd* (*m*, *n*) **by** (*rule gcd-commute*)
finally show *?thesis* **.**
qed

lemma *gcd-add2′* [*simp*]: *gcd* (*m*, *n* + *m*) = *gcd* (*m*, *n*)
apply (*subst add-commute*)
apply (*rule gcd-add2*)
done

lemma *gcd-add-mult*: *gcd* (*m*, *k* ∗ *m* + *n*) = *gcd* (*m*, *n*)
by (*induct k*) (*simp-all add*: *add-assoc*)

lemma *gcd-dvd-prod*: *gcd* (*m*, *n*) *dvd m* ∗ *n*
using *mult-dvd-mono* [*of 1*] **by** *auto*

Division by gcd yields rrelatively primes.

lemma *div-gcd-relprime*:
assumes *nz*: *a* ≠ *0* ∨ *b* ≠ *0*
shows *gcd* (*a div gcd*(*a*,*b*), *b div gcd*(*a*,*b*)) = *1*
proof −
let *?g* = *gcd* (*a*, *b*)
let *?a′* = *a div ?g*
let *?b′* = *b div ?g*
let *?g′* = *gcd* (*?a′*, *?b′*)
have *dvdg*: *?g dvd a ?g dvd b* **by** *simp-all*
have *dvdg′*: *?g′ dvd ?a′ ?g′ dvd ?b′* **by** *simp-all*
from *dvdg dvdg′* **obtain** *ka kb ka′ kb′* **where**
kab: *a* = *?g* ∗ *ka b* = *?g* ∗ *kb ?a′* = *?g′* ∗ *ka′ ?b′* = *?g′* ∗ *kb′*
unfolding *dvd-def* **by** *blast*
then have *?g* ∗ *?a′* = (*?g* ∗ *?g′*) ∗ *ka′ ?g* ∗ *?b′* = (*?g* ∗ *?g′*) ∗ *kb′* **by** *simp-all*
then have *dvdgg′*:*?g* ∗ *?g′ dvd a ?g*∗ *?g′ dvd b*
by (*auto simp add*: *dvd-mult-div-cancel* [*OF dvdg*(*1*)]
dvd-mult-div-cancel [*OF dvdg*(*2*)] *dvd-def*)

have *?g ≠ 0* **using** *nz* **by** (*simp add*: *gcd-zero*)
then have *gp*: *?g > 0* **by** *simp*
from *gcd-greatest* [*OF dvdgg'*] **have** *?g * ?g' dvd ?g* **.**
with *dvd-mult-cancel1* [*OF gp*] **show** *?g' = 1* **by** *simp*
qed

2.4 LCM defined by GCD

definition
lcm :: *nat* × *nat* ⇒ *nat*
where
lcm-prim-def: *lcm* = (λ(*m*, *n*). *m * n div gcd* (*m*, *n*))

lemma *lcm-def*:
lcm (*m*, *n*) = *m * n div gcd* (*m*, *n*)
unfolding *lcm-prim-def* **by** *simp*

lemma *prod-gcd-lcm*:
*m * n* = *gcd* (*m*, *n*) * *lcm* (*m*, *n*)
unfolding *lcm-def* **by** (*simp add*: *dvd-mult-div-cancel* [*OF gcd-dvd-prod*])

lemma *lcm-0* [*simp*]: *lcm* (*m*, *0*) = *0*
unfolding *lcm-def* **by** *simp*

lemma *lcm-1* [*simp*]: *lcm* (*m*, *1*) = *m*
unfolding *lcm-def* **by** *simp*

lemma *lcm-0-left* [*simp*]: *lcm* (*0*, *n*) = *0*
unfolding *lcm-def* **by** *simp*

lemma *lcm-1-left* [*simp*]: *lcm* (*1*, *m*) = *m*
unfolding *lcm-def* **by** *simp*

lemma *dvd-pos*:
fixes *n m* :: *nat*
assumes *n > 0* **and** *m dvd n*
shows *m > 0*
using *assms* **by** (*cases m*) *auto*

lemma *lcm-least*:
assumes *m dvd k* **and** *n dvd k*
shows *lcm* (*m*, *n*) *dvd k*
proof (*cases k*)
case *0* **then show** *?thesis* **by** *auto*
next
case (*Suc* -) **then have** *pos-k*: *k > 0* **by** *auto*
from *assms dvd-pos* [*OF this*] **have** *pos-mn*: *m > 0 n > 0* **by** *auto*
with *gcd-zero* [*of m n*] **have** *pos-gcd*: *gcd* (*m*, *n*) > *0* **by** *simp*
from *assms* **obtain** *p* **where** *k-m*: *k = m * p* **using** *dvd-def* **by** *blast*

from *assms* **obtain** q **where** *k-n*: $k = n * q$ **using** *dvd-def* **by** *blast*
from *pos-k k-m* **have** *pos-p*: $p > 0$ **by** *auto*
from *pos-k k-n* **have** *pos-q*: $q > 0$ **by** *auto*
have $k * k * gcd\ (q, p) = k * gcd\ (k * q, k * p)$
by (*simp add*: *mult-ac gcd-mult-distrib2*)
also have $\ldots = k * gcd\ (m * p * q, n * q * p)$
by (*simp add*: *k-m* [*symmetric*] *k-n* [*symmetric*])
also have $\ldots = k * p * q * gcd\ (m, n)$
by (*simp add*: *mult-ac gcd-mult-distrib2*)
finally have $(m * p) * (n * q) * gcd\ (q, p) = k * p * q * gcd\ (m, n)$
by (*simp only*: *k-m* [*symmetric*] *k-n* [*symmetric*])
then have $p * q * m * n * gcd\ (q, p) = p * q * k * gcd\ (m, n)$
by (*simp add*: *mult-ac*)
with *pos-p pos-q* **have** $m * n * gcd\ (q, p) = k * gcd\ (m, n)$
by *simp*
with *prod-gcd-lcm* [*of m n*]
have $lcm\ (m, n) * gcd\ (q, p) * gcd\ (m, n) = k * gcd\ (m, n)$
by (*simp add*: *mult-ac*)
with *pos-gcd* **have** $lcm\ (m, n) * gcd\ (q, p) = k$ **by** *simp*
then show *?thesis* **using** *dvd-def* **by** *auto*
qed

lemma *lcm-dvd1* [*iff*]:
$m\ dvd\ lcm\ (m, n)$
proof (*cases m*)
case *0* **then show** *?thesis* **by** *simp*
next
case (*Suc* -)
then have *mpos*: $m > 0$ **by** *simp*
show *?thesis*
proof (*cases n*)
case *0* **then show** *?thesis* **by** *simp*
next
case (*Suc* -)
then have *npos*: $n > 0$ **by** *simp*
have $gcd\ (m, n)\ dvd\ n$ **by** *simp*
then obtain k **where** $n = gcd\ (m, n) * k$ **using** *dvd-def* **by** *auto*
then have $m * n\ div\ gcd\ (m, n) = m * (gcd\ (m, n) * k)\ div\ gcd\ (m, n)$ **by** (*simp add*: *mult-ac*)
also have $\ldots = m * k$ **using** *mpos npos gcd-zero* **by** *simp*
finally show *?thesis* **by** (*simp add*: *lcm-def*)
qed
qed

lemma *lcm-dvd2* [*iff*]:
$n\ dvd\ lcm\ (m, n)$
proof (*cases n*)
case *0* **then show** *?thesis* **by** *simp*
next

case (*Suc* -)
then have *npos*: $n > 0$ **by** *simp*
show *?thesis*
proof (*cases m*)
case *0* **then show** *?thesis* **by** *simp*
next
case (*Suc* -)
then have *mpos*: $m > 0$ **by** *simp*
have *gcd* (m, n) *dvd m* **by** *simp*
then obtain *k* **where** $m = gcd\ (m, n) * k$ **using** *dvd-def* **by** *auto*
then have $m * n\ div\ gcd\ (m, n) = (gcd\ (m, n) * k) * n\ div\ gcd\ (m, n)$ **by** (*simp add*: *mult-ac*)
also have $\ldots = n * k$ **using** *mpos npos gcd-zero* **by** *simp*
finally show *?thesis* **by** (*simp add*: *lcm-def*)
qed
qed

2.5 GCD and LCM on integers

definition
$igcd :: int \Rightarrow int \Rightarrow int$ **where**
$igcd\ i\ j = int\ (gcd\ (nat\ (abs\ i), nat\ (abs\ j)))$

lemma *igcd-dvd1* [*simp*]: *igcd i j dvd i*
by (*simp add*: *igcd-def int-dvd-iff*)

lemma *igcd-dvd2* [*simp*]: *igcd i j dvd j*
by (*simp add*: *igcd-def int-dvd-iff*)

lemma *igcd-pos*: $igcd\ i\ j \geq 0$
by (*simp add*: *igcd-def*)

lemma *igcd0* [*simp*]: $(igcd\ i\ j = 0) = (i = 0 \wedge j = 0)$
by (*simp add*: *igcd-def gcd-zero*) *arith*

lemma *igcd-commute*: $igcd\ i\ j = igcd\ j\ i$
unfolding *igcd-def* **by** (*simp add*: *gcd-commute*)

lemma *igcd-neg1* [*simp*]: $igcd\ (-i)\ j = igcd\ i\ j$
unfolding *igcd-def* **by** *simp*

lemma *igcd-neg2* [*simp*]: $igcd\ i\ (-j) = igcd\ i\ j$
unfolding *igcd-def* **by** *simp*

lemma *zrelprime-dvd-mult*: $igcd\ i\ j = 1 \Longrightarrow i\ dvd\ k * j \Longrightarrow i\ dvd\ k$
unfolding *igcd-def*
proof −
assume $int\ (gcd\ (nat\ |i|, nat\ |j|)) = 1\ i\ dvd\ k * j$
then have *g*: $gcd\ (nat\ |i|, nat\ |j|) = 1$ **by** *simp*

from ‹*i dvd k * j*› **obtain** *h* **where** *h*: *k*j = i*h* **unfolding** *dvd-def* **by** *blast*
have *th*: *nat |i| dvd nat |k| * nat |j|*
unfolding *dvd-def*
by (*rule-tac x*= *nat |h|* **in** *exI*, *simp add*: *h nat-abs-mult-distrib* [*symmetric*])
from *relprime-dvd-mult* [*OF g th*] **obtain** *h'* **where** *h'*: *nat |k| = nat |i| * h'*
unfolding *dvd-def* **by** *blast*
from *h'* **have** *int* (*nat |k|*) = *int* (*nat |i| * h'*) **by** *simp*
then have *|k| = |i| * int h'* **by** (*simp add*: *int-mult*)
then show *?thesis*
apply (*subst zdvd-abs1* [*symmetric*])
apply (*subst zdvd-abs2* [*symmetric*])
apply (*unfold dvd-def*)
apply (*rule-tac x = int h'* **in** *exI*, *simp*)
done
qed

lemma *int-nat-abs*: *int* (*nat* (*abs x*)) = *abs x* **by** *arith*

lemma *igcd-greatest*:
assumes *k dvd m* **and** *k dvd n*
shows *k dvd igcd m n*
proof −
let *?k'* = *nat |k|*
let *?m'* = *nat |m|*
let *?n'* = *nat |n|*
from ‹*k dvd m*› **and** ‹*k dvd n*› **have** *dvd'*: *?k' dvd ?m' ?k' dvd ?n'*
unfolding *zdvd-int* **by** (*simp-all only*: *int-nat-abs zdvd-abs1 zdvd-abs2*)
from *gcd-greatest* [*OF dvd'*] **have** *int* (*nat |k|*) *dvd igcd m n*
unfolding *igcd-def* **by** (*simp only*: *zdvd-int*)
then have *|k| dvd igcd m n* **by** (*simp only*: *int-nat-abs*)
then show *k dvd igcd m n* **by** (*simp add*: *zdvd-abs1*)
qed

lemma *div-igcd-relprime*:
assumes *nz*: *a ≠ 0 ∨ b ≠ 0*
shows *igcd* (*a div* (*igcd a b*)) (*b div* (*igcd a b*)) = *1*
proof −
from *nz* **have** *nz'*: *nat |a| ≠ 0 ∨ nat |b| ≠ 0* **by** *arith*
let *?g* = *igcd a b*
let *?a'* = *a div ?g*
let *?b'* = *b div ?g*
let *?g'* = *igcd ?a' ?b'*
have *dvdg*: *?g dvd a ?g dvd b* **by** (*simp-all add*: *igcd-dvd1 igcd-dvd2*)
have *dvdg'*: *?g' dvd ?a' ?g' dvd ?b'* **by** (*simp-all add*: *igcd-dvd1 igcd-dvd2*)
from *dvdg dvdg'* **obtain** *ka kb ka' kb'* **where**
kab: *a = ?g*ka b = ?g*kb ?a' = ?g'*ka' ?b' = ?g' * kb'*
unfolding *dvd-def* **by** *blast*
then have *?g* ?a' = (?g * ?g') * ka' ?g* ?b' = (?g * ?g') * kb'* **by** *simp-all*
then have *dvdgg'*:*?g * ?g' dvd a ?g* ?g' dvd b*

```
    by (auto simp add: zdvd-mult-div-cancel [OF dvdg(1)]
      zdvd-mult-div-cancel [OF dvdg(2)] dvd-def)
  have ?g ≠ 0 using nz by simp
  then have gp: ?g ≠ 0 using igcd-pos[where i=a and j=b] by arith
  from igcd-greatest [OF dvdgg'] have ?g * ?g' dvd ?g .
  with zdvd-mult-cancel1 [OF gp] have |?g'| = 1 by simp
  with igcd-pos show ?g' = 1 by simp
qed

definition ilcm = (λi j. int (lcm(nat(abs i),nat(abs j))))

lemma dvd-ilcm-self1[simp]: i dvd ilcm i j
by(simp add:ilcm-def dvd-int-iff)

lemma dvd-ilcm-self2[simp]: j dvd ilcm i j
by(simp add:ilcm-def dvd-int-iff)


lemma dvd-imp-dvd-ilcm1:
  assumes k dvd i shows k dvd (ilcm i j)
proof −
  have nat(abs k) dvd nat(abs i) using ‹k dvd i›
    by(simp add:int-dvd-iff[symmetric] dvd-int-iff[symmetric] zdvd-abs1)
  thus ?thesis by(simp add:ilcm-def dvd-int-iff)(blast intro: dvd-trans)
qed

lemma dvd-imp-dvd-ilcm2:
  assumes k dvd j shows k dvd (ilcm i j)
proof −
  have nat(abs k) dvd nat(abs j) using ‹k dvd j›
    by(simp add:int-dvd-iff[symmetric] dvd-int-iff[symmetric] zdvd-abs1)
  thus ?thesis by(simp add:ilcm-def dvd-int-iff)(blast intro: dvd-trans)
qed


lemma zdvd-self-abs1: (d::int) dvd (abs d)
by (case-tac d <0, simp-all)

lemma zdvd-self-abs2: (abs (d::int)) dvd d
by (case-tac d<0, simp-all)



lemma lcm-pos:
  assumes mpos: m > 0
  and npos: n>0
  shows lcm (m,n) > 0
proof(rule ccontr, simp add: lcm-def gcd-zero)
assume h:m*n div gcd(m,n) = 0
```

from *mpos npos* **have** *gcd* (*m*,*n*) ≠ *0* **using** *gcd-zero* **by** *simp*
hence *gcdp*: *gcd*(*m*,*n*) > *0* **by** *simp*
with *h*
have *m*∗*n* < *gcd*(*m*,*n*)
by (*cases m* ∗ *n* < *gcd* (*m*, *n*)) (*auto simp add*: *div-if*[*OF gcdp*, **where** *m*=*m*∗*n*])
moreover
have *gcd*(*m*,*n*) *dvd m* **by** *simp*
with *mpos dvd-imp-le* **have** *t1*:*gcd*(*m*,*n*) ≤ *m* **by** *simp*
with *npos* **have** *t1*:*gcd*(*m*,*n*)∗*n* ≤ *m*∗*n* **by** *simp*
have *gcd*(*m*,*n*) ≤ *gcd*(*m*,*n*)∗*n* **using** *npos* **by** *simp*
with *t1* **have** *gcd*(*m*,*n*) ≤ *m*∗*n* **by** *arith*
ultimately show *False* **by** *simp*
qed

lemma *ilcm-pos*:
assumes *anz*: *a* ≠ *0*
and *bnz*: *b* ≠ *0*
shows *0* < *ilcm a b*
proof−
let *?na* = *nat* (*abs a*)
let *?nb* = *nat* (*abs b*)
have *nap*: *?na* >*0* **using** *anz* **by** *simp*
have *nbp*: *?nb* >*0* **using** *bnz* **by** *simp*
have *0* < *lcm* (*?na*,*?nb*) **by** (*rule lcm-pos*[*OF nap nbp*])
thus *?thesis* **by** (*simp add*: *ilcm-def*)
qed

end

3 Abstract-Rat: Abstract rational numbers

theory *Abstract-Rat*
imports *GCD*
begin

types *Num* = *int* × *int*

abbreviation
Num0-syn :: *Num* (*0*$_N$)
where *0*$_N$ ≡ (*0*, *0*)

abbreviation
Numi-syn :: *int* ⇒ *Num* (-$_N$)
where *i*$_N$ ≡ (*i*, *1*)

definition
isnormNum :: *Num* ⇒ *bool*
where

isnormNum = (λ(*a*,*b*). (*if a = 0 then b = 0 else b > 0 ∧ igcd a b = 1*))

definition
normNum :: *Num* ⇒ *Num*
where
normNum = (λ(*a*,*b*). (*if a=0 ∨ b = 0 then* (*0*,*0*) *else*
(*let g = igcd a b*
in if b > 0 then (*a div g*, *b div g*) *else* (− (*a div g*), − (*b div g*)))))

lemma *normNum-isnormNum* [*simp*]: *isnormNum* (*normNum x*)
proof −
have ∃ *a b*. *x* = (*a*,*b*) **by** *auto*
then obtain *a b* **where** *x*[*simp*]: *x* = (*a*,*b*) **by** *blast*
{assume *a=0 ∨ b = 0* **hence** *?thesis* **by** (*simp add*: *normNum-def isnormNum-def*)**}**

moreover
{assume *anz*: *a* ≠ *0* **and** *bnz*: *b* ≠ *0*
let *?g* = *igcd a b*
let *?a'* = *a div ?g*
let *?b'* = *b div ?g*
let *?g'* = *igcd ?a' ?b'*
from *anz bnz* **have** *?g* ≠ *0* **by** *simp* **with** *igcd-pos*[*of a b*]
have *gpos*: *?g* > *0* **by** *arith*
have *gdvd*: *?g dvd a ?g dvd b* **by** (*simp-all add*: *igcd-dvd1 igcd-dvd2*)
from *zdvd-mult-div-cancel*[*OF gdvd*(*1*)] *zdvd-mult-div-cancel*[*OF gdvd*(*2*)]
anz bnz
have *nz'*:*?a'* ≠ *0 ?b'* ≠ *0*
by − (*rule notI*,*simp add*:*igcd-def*)+
from *anz bnz* **have** *stupid*: *a* ≠ *0* ∨ *b* ≠ *0* **by** *blast*
from *div-igcd-relprime*[*OF stupid*] **have** *gp1*: *?g'* = *1* **.**
from *bnz* **have** *b* < *0* ∨ *b* > *0* **by** *arith*
moreover
{assume *b*: *b* > *0*
from *pos-imp-zdiv-nonneg-iff*[*OF gpos*] *b*
have *?b'* ≥ *0* **by** *simp*
with *nz'* **have** *b'*: *?b'* > *0* **by** *simp*
from *b b' anz bnz nz' gp1* **have** *?thesis*
by (*simp add*: *isnormNum-def normNum-def Let-def split-def fst-conv snd-conv*)**}**
moreover {assume *b*: *b* < *0*
{assume *b'*: *?b'* ≥ *0*
from *gpos* **have** *th*: *?g* ≥ *0* **by** *arith*
from *mult-nonneg-nonneg*[*OF th b'*] *zdvd-mult-div-cancel*[*OF gdvd*(*2*)]
have *False* **using** *b* **by** *simp* **}**
hence *b'*: *?b'* < *0* **by** (*presburger add*: *linorder-not-le*[*symmetric*])
from *anz bnz nz' b b' gp1* **have** *?thesis*
by (*simp add*: *isnormNum-def normNum-def Let-def split-def fst-conv snd-conv*)**}**
ultimately have *?thesis* **by** *blast*

}
ultimately show *?thesis* **by** *blast*
qed

Arithmetic over Num

definition
Nadd :: *Num* ⇒ *Num* ⇒ *Num* (**infixl** $+_N$ *60*)
where
Nadd = (λ(*a*,*b*) (*a*′,*b*′). *if a* = *0* ∨ *b* = *0 then normNum*(*a*′,*b*′)
else if a′=*0* ∨ *b*′ = *0 then normNum*(*a*,*b*)
else normNum(*a*∗*b*′ + *b*∗*a*′, *b*∗*b*′))

definition
Nmul :: *Num* ⇒ *Num* ⇒ *Num* (**infixl** $*_N$ *60*)
where
Nmul = (λ(*a*,*b*) (*a*′,*b*′). *let g* = *igcd* (*a*∗*a*′) (*b*∗*b*′)
in (*a*∗*a*′ *div g*, *b*∗*b*′ *div g*))

definition
Nneg :: *Num* ⇒ *Num* ($\sim_N$)
where
Nneg ≡ (λ(*a*,*b*). (−*a*,*b*))

definition
Nsub :: *Num* ⇒ *Num* ⇒ *Num* (**infixl** $-_N$ *60*)
where
Nsub = (λ*a b*. *a* $+_N$ $\sim_N$ *b*)

definition
Ninv :: *Num* ⇒ *Num*
where
Ninv ≡ λ(*a*,*b*). *if a* < *0 then* (−*b*, |*a*|) *else* (*b*,*a*)

definition
Ndiv :: *Num* ⇒ *Num* ⇒ *Num* (**infixl** $\div_N$ *60*)
where
Ndiv ≡ λ*a b*. *a* $*_N$ *Ninv b*

lemma *Nneg-normN*[*simp*]: *isnormNum x* ⟹ *isnormNum* ($\sim_N$ *x*)
by(*simp add*: *isnormNum-def Nneg-def split-def*)
lemma *Nadd-normN*[*simp*]: *isnormNum* (*x* $+_N$ *y*)
by (*simp add*: *Nadd-def split-def*)
lemma *Nsub-normN*[*simp*]: ⟦ *isnormNum y*⟧ ⟹ *isnormNum* (*x* $-_N$ *y*)
by (*simp add*: *Nsub-def split-def*)
lemma *Nmul-normN*[*simp*]: **assumes** *xn*:*isnormNum x* **and** *yn*: *isnormNum y*
shows *isnormNum* (*x* $*_N$ *y*)
proof−
have ∃ *a b*. *x* = (*a*,*b*) **and** ∃ *a*′ *b*′. *y* = (*a*′,*b*′) **by** *auto*
then obtain *a b a*′ *b*′ **where** *ab*: *x* = (*a*,*b*) **and** *ab*′: *y* = (*a*′,*b*′) **by** *blast*

{assume $a = 0$
hence *?thesis* **using** *xn ab ab'*
by (*simp add*: *igcd-def isnormNum-def Let-def Nmul-def split-def*)**}**
moreover
{assume $a' = 0$
hence *?thesis* **using** *yn ab ab'*
by (*simp add*: *igcd-def isnormNum-def Let-def Nmul-def split-def*)**}**
moreover
{assume *a*: $a \neq 0$ **and** *a'*: $a' \neq 0$
hence *bp*: $b > 0$ $b' > 0$ **using** *xn yn ab ab'* **by** (*simp-all add*: *isnormNum-def*)
from *mult-pos-pos*[*OF bp*] **have** $x *_N y = normNum\ (a*a', b*b')$
using *ab ab' a a' bp* **by** (*simp add*: *Nmul-def Let-def split-def normNum-def*)
hence *?thesis* **by** *simp***}**
ultimately show *?thesis* **by** *blast*
qed

lemma *Ninv-normN*[*simp*]: $isnormNum\ x \Longrightarrow isnormNum\ (Ninv\ x)$
by (*simp add*: *Ninv-def isnormNum-def split-def*)
(*cases fst* $x = 0$, *auto simp add*: *igcd-commute*)

lemma *isnormNum-int*[*simp*]:
$isnormNum\ 0_N\ isnormNum\ (1::int)_N\ i \neq 0 \Longrightarrow isnormNum\ i_N$
by (*simp-all add*: *isnormNum-def igcd-def*)

Relations over Num

definition
$Nlt0 :: Num \Rightarrow bool\ (0>_N)$
where
$Nlt0 = (\lambda(a,b).\ a < 0)$

definition
$Nle0 :: Num \Rightarrow bool\ (0\geq_N)$
where
$Nle0 = (\lambda(a,b).\ a \leq 0)$

definition
$Ngt0 :: Num \Rightarrow bool\ (0<_N)$
where
$Ngt0 = (\lambda(a,b).\ a > 0)$

definition
$Nge0 :: Num \Rightarrow bool\ (0\leq_N)$
where
$Nge0 = (\lambda(a,b).\ a \geq 0)$

definition
$Nlt :: Num \Rightarrow Num \Rightarrow bool$ (**infix** $<_N$ *55*)
where
$Nlt = (\lambda a\ b.\ 0>_N\ (a -_N b))$

definition
Nle :: *Num* ⇒ *Num* ⇒ *bool* (**infix** $\leq_N$ *55*)
where
Nle = (λ*a b*. *0*$\geq_N$ (*a* −$_N$ *b*))

definition
INum = (λ(*a*,*b*). *of-int a* / *of-int b*)

lemma *INum-int* [*simp*]: *INum i*$_N$ = ((*of-int i*) ::′*a*::*field*) *INum 0*$_N$ = (*0*::′*a*::*field*)
by (*simp-all add*: *INum-def*)

lemma *isnormNum-unique*[*simp*]:
assumes *na*: *isnormNum x* **and** *nb*: *isnormNum y*
shows ((*INum x* ::′*a*::{*ring-char-0*,*field*, *division-by-zero*}) = *INum y*) = (*x* = *y*) (**is** *?lhs* = *?rhs*)
proof
have ∃ *a b a*′ *b*′. *x* = (*a*,*b*) ∧ *y* = (*a*′,*b*′) **by** *auto*
then obtain *a b a*′ *b*′ **where** *xy*[*simp*]: *x* = (*a*,*b*) *y*=(*a*′,*b*′) **by** *blast*
assume *H*: *?lhs*
{assume *a* = *0* ∨ *b* = *0* ∨ *a*′ = *0* ∨ *b*′ = *0* **hence** *?rhs*
using *na nb H*
apply (*simp add*: *INum-def split-def isnormNum-def*)
apply (*cases a* = *0*, *simp-all*)
apply (*cases b* = *0*, *simp-all*)
apply (*cases a*′ = *0*, *simp-all*)
apply (*cases a*′ = *0*, *simp-all add*: *of-int-eq-0-iff*)
done}
moreover
{ assume *az*: *a* ≠ *0* **and** *bz*: *b* ≠ *0* **and** *a*′*z*: *a*′≠*0* **and** *b*′*z*: *b*′≠*0*
from *az bz a*′*z b*′*z na nb* **have** *pos*: *b* > *0 b*′ > *0* **by** (*simp-all add*: *isnormNum-def*)
from *prems* **have** *eq*:*a* ∗ *b*′ = *a*′∗*b*
by (*simp add*: *INum-def eq-divide-eq divide-eq-eq of-int-mult*[*symmetric*] *del*: *of-int-mult*)
from *prems* **have** *gcd1*: *igcd a b* = *1 igcd b a* = *1 igcd a*′ *b*′ = *1 igcd b*′ *a*′ = *1*
by (*simp-all add*: *isnormNum-def add*: *igcd-commute*)
from *eq* **have** *raw-dvd*: *a dvd a*′∗*b b dvd b*′∗*a a*′ *dvd a*∗*b*′ *b*′ *dvd b*∗*a*′
apply(*unfold dvd-def*)
apply (*rule-tac x*=*b*′ **in** *exI*, *simp add*: *mult-ac*)
apply (*rule-tac x*=*a*′ **in** *exI*, *simp add*: *mult-ac*)
apply (*rule-tac x*=*b* **in** *exI*, *simp add*: *mult-ac*)
apply (*rule-tac x*=*a* **in** *exI*, *simp add*: *mult-ac*)
done
from *zdvd-dvd-eq*[*OF bz zrelprime-dvd-mult*[*OF gcd1*(*2*) *raw-dvd*(*2*)]
zrelprime-dvd-mult[*OF gcd1*(*4*) *raw-dvd*(*4*)]]
have *eq1*: *b* = *b*′ **using** *pos* **by** *simp-all*
with *eq* **have** *a* = *a*′ **using** *pos* **by** *simp*
with *eq1* **have** *?rhs* **by** *simp***}**

ultimately show *?rhs* **by** *blast*
next
assume *?rhs* **thus** *?lhs* **by** *simp*
qed

lemma *isnormNum0*[*simp*]: *isnormNum x* ⟹ (*INum x* = (*0*::'*a*::{*ring-char-0*, *field*,*division-by-zero*})) = (*x* = 0_N)
unfolding *INum-int*(*2*)[*symmetric*]
by (*rule isnormNum-unique*, *simp-all*)

lemma *of-int-div-aux*: *d* ~= *0* ==> ((*of-int x*)::'*a*::{*field*, *ring-char-0*}) / (*of-int d*) =
of-int (*x div d*) + (*of-int* (*x mod d*)) / ((*of-int d*)::'*a*)
proof −
assume *d* ~= *0*
hence *dz*: *of-int d* ≠ (*0*::'*a*) **by** (*simp add*: *of-int-eq-0-iff*)
let *?t* = *of-int* (*x div d*) ∗ ((*of-int d*)::'*a*) + *of-int*(*x mod d*)
let *?f* = λ*x*. *x* / *of-int d*
have *x* = (*x div d*) ∗ *d* + *x mod d*
by *auto*
then have *eq*: *of-int x* = *?t*
by (*simp only*: *of-int-mult*[*symmetric*] *of-int-add* [*symmetric*])
then have *of-int x* / *of-int d* = *?t* / *of-int d*
using *cong*[*OF refl*[*of ?f*] *eq*] **by** *simp*
then show *?thesis* **by** (*simp add*: *add-divide-distrib ring-simps prems*)
qed

lemma *of-int-div*: (*d*::*int*) ~= *0* ==> *d dvd n* ==>
(*of-int*(*n div d*)::'*a*::{*field*, *ring-char-0*}) = *of-int n* / *of-int d*
apply (*frule of-int-div-aux* [*of d n*, **where** *?'a* = '*a*])
apply *simp*
apply (*simp add*: *zdvd-iff-zmod-eq-0*)
done

lemma *normNum*[*simp*]: *INum* (*normNum x*) = (*INum x* :: '*a*::{*ring-char-0*,*field*, *division-by-zero*})
proof−
have ∃ *a b*. *x* = (*a*,*b*) **by** *auto*
then obtain *a b* **where** *x*[*simp*]: *x* = (*a*,*b*) **by** *blast*
{assume *a=0* ∨ *b* = *0* **hence** *?thesis*
by (*simp add*: *INum-def normNum-def split-def Let-def*)**}**
moreover
{assume *a*: *a≠0* **and** *b*: *b≠0*
let *?g* = *igcd a b*
from *a b* **have** *g*: *?g* ≠ *0***by** *simp*
from *of-int-div*[*OF g*, **where** *?'a* = '*a*]
have *?thesis* **by** (*auto simp add*: *INum-def normNum-def split-def Let-def*)**}**

ultimately show *?thesis* **by** *blast*
qed

lemma *INum-normNum-iff*: (*INum x* ::'*a*::{*field*, *division-by-zero*, *ring-char-0*})
= *INum y* ⟷ *normNum x* = *normNum y* (**is** *?lhs* = *?rhs*)
proof −
have *normNum x* = *normNum y* ⟷ (*INum* (*normNum x*) :: '*a*) = *INum* (*normNum y*)
by (*simp del*: *normNum*)
also have ... = *?lhs* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *Nadd*[*simp*]: *INum* (*x* +$_N$ *y*) = *INum x* + (*INum y* :: '*a* :: {*ring-char-0*,*division-by-zero*,*field*})
proof−
let *?z* = *0*:: '*a*
have ∃ *a b*. *x* = (*a*,*b*) ∃ *a' b'*. *y* = (*a'*,*b'*) **by** *auto*
then obtain *a b a' b'* **where** *x*[*simp*]: *x* = (*a*,*b*)
and *y*[*simp*]: *y* = (*a'*,*b'*) **by** *blast*
{assume *a=0* ∨ *a'*= *0* ∨ *b* =*0* ∨ *b'* = *0* **hence** *?thesis*
apply (*cases a=0*,*simp-all add*: *Nadd-def*)
apply (*cases b*= *0*,*simp-all add*: *INum-def*)
apply (*cases a'*= *0*,*simp-all*)
apply (*cases b'*= *0*,*simp-all*)
done **}**
moreover
{assume *aa'*:*a* ≠ *0 a'*≠ *0* **and** *bb'*: *b* ≠ *0 b'* ≠ *0*
{assume *z*: *a* ∗ *b'* + *b* ∗ *a'* = *0*
hence *of-int* (*a*∗*b'* + *b*∗*a'*) / (*of-int b*∗ *of-int b'*) = *?z* **by** *simp*
hence *of-int b'* ∗ *of-int a* / (*of-int b* ∗ *of-int b'*) + *of-int b* ∗ *of-int a'* / (*of-int b* ∗ *of-int b'*) = *?z* **by** (*simp add*:*add-divide-distrib*)
hence *th*: *of-int a* / *of-int b* + *of-int a'* / *of-int b'* = *?z* **using** *bb' aa'* **by** *simp*
from *z aa' bb'* **have** *?thesis*
by (*simp add*: *th Nadd-def normNum-def INum-def split-def*)**}**
moreover {assume *z*: *a* ∗ *b'* + *b* ∗ *a'* ≠ *0*
let *?g* = *igcd* (*a* ∗ *b'* + *b* ∗ *a'*) (*b*∗*b'*)
have *gz*: *?g* ≠ *0* **using** *z* **by** *simp*
have *?thesis* **using** *aa' bb' z gz*
of-int-div[**where** *?'a* = '*a*,
OF gz igcd-dvd1[**where** *i*=*a* ∗ *b'* + *b* ∗ *a'* **and** *j*=*b*∗*b'*]]
of-int-div[**where** *?'a* = '*a*,
OF gz igcd-dvd2[**where** *i*=*a* ∗ *b'* + *b* ∗ *a'* **and** *j*=*b*∗*b'*]]
by (*simp add*: *x y Nadd-def INum-def normNum-def Let-def add-divide-distrib*)**}**
ultimately have *?thesis* **using** *aa' bb'*
by (*simp add*: *Nadd-def INum-def normNum-def x y Let-def*) **}**
ultimately show *?thesis* **by** *blast*
qed

lemma *Nmul*[*simp*]: *INum* $(x *_N y) = INum\ x * (INum\ y :: 'a :: \{ring\text{-}char\text{-}0, division\text{-}by\text{-}zero, field\})$

proof−
let *?z* = *0*::'*a*
have ∃ *a b*. *x* = (*a*,*b*) ∃ *a' b'*. *y* = (*a'*,*b'*) **by** *auto*
then obtain *a b a' b'* **where** *x*: *x* = (*a*,*b*) **and** *y*: *y* = (*a'*,*b'*) **by** *blast*
{assume *a*=*0* ∨ *a'*= *0* ∨ *b* = *0* ∨ *b'* = *0* **hence** *?thesis*
apply (*cases a*=*0*,*simp-all add*: *x y Nmul-def INum-def Let-def*)
apply (*cases b*=*0*,*simp-all*)
apply (*cases a'*=*0*,*simp-all*)
done **}**
moreover
{assume *z*: *a* ≠ *0 a'* ≠ *0 b* ≠ *0 b'* ≠ *0*
let *?g*=*igcd* (*a*∗*a'*) (*b*∗*b'*)
have *gz*: *?g* ≠ *0* **using** *z* **by** *simp*
from *z of-int-div*[**where** *?'a* = '*a*, *OF gz igcd-dvd1*[**where** *i*=*a*∗*a'* **and** *j*=*b*∗*b'*]]

of-int-div[**where** *?'a* = '*a* , *OF gz igcd-dvd2*[**where** *i*=*a*∗*a'* **and** *j*=*b*∗*b'*]]
have *?thesis* **by** (*simp add*: *Nmul-def x y Let-def INum-def*)**}**
ultimately show *?thesis* **by** *blast*
qed

lemma *Nneg*[*simp*]: *INum* $(\sim_N x) = -\ (INum\ x :: 'a :: field)$
by (*simp add*: *Nneg-def split-def INum-def*)

lemma *Nsub*[*simp*]: **shows** *INum* $(x -_N y) = INum\ x - (INum\ y :: 'a :: \{ring\text{-}char\text{-}0, division\text{-}by\text{-}zero, field\})$
by (*simp add*: *Nsub-def split-def*)

lemma *Ninv*[*simp*]: *INum* (*Ninv x*) = (*1*::'*a* :: {*division-by-zero*,*field*}) / (*INum x*)
by (*simp add*: *Ninv-def INum-def split-def*)

lemma *Ndiv*[*simp*]: *INum* $(x \div_N y) = INum\ x$ / (*INum y* ::'*a* :: {*ring-char-0*, *division-by-zero*,*field*}) **by** (*simp add*: *Ndiv-def*)

lemma *Nlt0-iff*[*simp*]: **assumes** *nx*: *isnormNum x*
shows ((*INum x* :: '*a* :: {*ring-char-0*,*division-by-zero*,*ordered-field*})< *0*) = *0*$>_N$ *x*
proof−
have ∃ *a b*. *x* = (*a*,*b*) **by** *simp*
then obtain *a b* **where** *x*[*simp*]:*x* = (*a*,*b*) **by** *blast*
{assume *a* = *0* **hence** *?thesis* **by** (*simp add*: *Nlt0-def INum-def*) **}**
moreover
{assume *a*: *a*≠*0* **hence** *b*: (*of-int b*::'*a*) > *0* **using** *nx* **by** (*simp add*: *isnormNum-def*)
from *pos-divide-less-eq*[*OF b*, **where** *b*=*of-int a* **and** *a*=*0*::'*a*]
have *?thesis* **by** (*simp add*: *Nlt0-def INum-def*)**}**
ultimately show *?thesis* **by** *blast*
qed

lemma *Nle0-iff*[*simp*]:**assumes** *nx*: *isnormNum x*
shows ((*INum x* :: *'a* :: {*ring-char-0*,*division-by-zero*,*ordered-field*}) ≤ *0*) = *0*≥$_N$ *x*
proof−
have ∃ *a b*. *x* = (*a*,*b*) **by** *simp*
then obtain *a b* **where** *x*[*simp*]:*x* = (*a*,*b*) **by** *blast*
{assume *a* = *0* **hence** *?thesis* **by** (*simp add*: *Nle0-def INum-def*) **}**
moreover
{assume *a*: *a*≠*0* **hence** *b*: (*of-int b* :: *'a*) > *0* **using** *nx* **by** (*simp add*: *isnormNum-def*)
from *pos-divide-le-eq*[*OF b*, **where** *b*=*of-int a* **and** *a*=*0*::*'a*]
have *?thesis* **by** (*simp add*: *Nle0-def INum-def*)**}**
ultimately show *?thesis* **by** *blast*
qed

lemma *Ngt0-iff*[*simp*]:**assumes** *nx*: *isnormNum x* **shows** ((*INum x* :: *'a* :: {*ring-char-0*,*division-by-zero*,*ordere*
0) = *0*<$_N$ *x*
proof−
have ∃ *a b*. *x* = (*a*,*b*) **by** *simp*
then obtain *a b* **where** *x*[*simp*]:*x* = (*a*,*b*) **by** *blast*
{assume *a* = *0* **hence** *?thesis* **by** (*simp add*: *Ngt0-def INum-def*) **}**
moreover
{assume *a*: *a*≠*0* **hence** *b*: (*of-int b*::*'a*) > *0* **using** *nx* **by** (*simp add*: *isnormNum-def*)
from *pos-less-divide-eq*[*OF b*, **where** *b*=*of-int a* **and** *a*=*0*::*'a*]
have *?thesis* **by** (*simp add*: *Ngt0-def INum-def*)**}**
ultimately show *?thesis* **by** *blast*
qed
lemma *Nge0-iff*[*simp*]:**assumes** *nx*: *isnormNum x*
shows ((*INum x* :: *'a* :: {*ring-char-0*,*division-by-zero*,*ordered-field*}) ≥ *0*) = *0*≤$_N$ *x*
proof−
have ∃ *a b*. *x* = (*a*,*b*) **by** *simp*
then obtain *a b* **where** *x*[*simp*]:*x* = (*a*,*b*) **by** *blast*
{assume *a* = *0* **hence** *?thesis* **by** (*simp add*: *Nge0-def INum-def*) **}**
moreover
{assume *a*: *a*≠*0* **hence** *b*: (*of-int b*::*'a*) > *0* **using** *nx* **by** (*simp add*: *isnormNum-def*)
from *pos-le-divide-eq*[*OF b*, **where** *b*=*of-int a* **and** *a*=*0*::*'a*]
have *?thesis* **by** (*simp add*: *Nge0-def INum-def*)**}**
ultimately show *?thesis* **by** *blast*
qed

lemma *Nlt-iff*[*simp*]: **assumes** *nx*: *isnormNum x* **and** *ny*: *isnormNum y*
shows ((*INum x* :: *'a* :: {*ring-char-0*,*division-by-zero*,*ordered-field*}) < *INum y*)
= (*x* <$_N$ *y*)
proof−
let *?z* = *0*::*'a*
have ((*INum x* ::*'a*) < *INum y*) = (*INum* (*x* −$_N$ *y*) < *?z*) **using** *nx ny* **by** *simp*
also have ... = (*0*>$_N$ (*x* −$_N$ *y*)) **using** *Nlt0-iff*[*OF Nsub-normN*[*OF ny*]] **by** *simp*
finally show *?thesis* **by** (*simp add*: *Nlt-def*)

qed

lemma *Nle-iff*[*simp*]: **assumes** *nx*: *isnormNum x* **and** *ny*: *isnormNum y*
shows ((*INum x* :: *'a* :: {*ring-char-0*,*division-by-zero*,*ordered-field*})≤ *INum y*)
= (*x* ≤$_N$ *y*)
proof−
have ((*INum x* ::*'a*) ≤ *INum y*) = (*INum* (*x* −$_N$ *y*) ≤ (*0*::*'a*)) **using** *nx ny* **by** *simp*
also have ... = (*0*≥$_N$ (*x* −$_N$ *y*)) **using** *Nle0-iff*[*OF Nsub-normN*[*OF ny*]] **by** *simp*
finally show *?thesis* **by** (*simp add*: *Nle-def*)
qed

lemma *Nadd-commute*: *x* +$_N$ *y* = *y* +$_N$ *x*
proof−
have *n*: *isnormNum* (*x* +$_N$ *y*) *isnormNum* (*y* +$_N$ *x*) **by** *simp-all*
have (*INum* (*x* +$_N$ *y*)::*'a* :: {*ring-char-0*,*division-by-zero*,*field*}) = *INum* (*y* +$_N$ *x*) **by** *simp*
with *isnormNum-unique*[*OF n*] **show** *?thesis* **by** *simp*
qed

lemma[*simp*]: (*0*, *b*) +$_N$ *y* = *normNum y* (*a*, *0*) +$_N$ *y* = *normNum y*
x +$_N$ (*0*, *b*) = *normNum x x* +$_N$ (*a*, *0*) = *normNum x*
apply (*simp add*: *Nadd-def split-def*, *simp add*: *Nadd-def split-def*)
apply (*subst Nadd-commute*,*simp add*: *Nadd-def split-def*)
apply (*subst Nadd-commute*,*simp add*: *Nadd-def split-def*)
done

lemma *normNum-nilpotent-aux*[*simp*]: **assumes** *nx*: *isnormNum x*
shows *normNum x* = *x*
proof−
let *?a* = *normNum x*
have *n*: *isnormNum ?a* **by** *simp*
have *th*:*INum ?a* = (*INum x* ::*'a* :: {*ring-char-0*, *division-by-zero*,*field*}) **by** *simp*
with *isnormNum-unique*[*OF n nx*]
show *?thesis* **by** *simp*
qed

lemma *normNum-nilpotent*[*simp*]: *normNum* (*normNum x*) = *normNum x*
by *simp*
lemma *normNum0*[*simp*]: *normNum* (*0*,*b*) = *0*$_N$ *normNum* (*a*,*0*) = *0*$_N$
by (*simp-all add*: *normNum-def*)
lemma *normNum-Nadd*: *normNum* (*x* +$_N$ *y*) = *x* +$_N$ *y* **by** *simp*
lemma *Nadd-normNum1*[*simp*]: *normNum x* +$_N$ *y* = *x* +$_N$ *y*
proof−
have *n*: *isnormNum* (*normNum x* +$_N$ *y*) *isnormNum* (*x* +$_N$ *y*) **by** *simp-all*
have *INum* (*normNum x* +$_N$ *y*) = *INum x* + (*INum y* :: *'a* :: {*ring-char-0*, *division-by-zero*,*field*}) **by** *simp*
also have ... = *INum* (*x* +$_N$ *y*) **by** *simp*

finally show *?thesis* **using** *isnormNum-unique*[*OF n*] **by** *simp*
qed
lemma *Nadd-normNum2*[*simp*]: $x +_N normNum\ y = x +_N y$
proof−
have *n*: *isnormNum* $(x +_N normNum\ y)$ *isnormNum* $(x +_N y)$ **by** *simp-all*
have *INum* $(x +_N normNum\ y) = INum\ x + (INum\ y :: 'a :: \{$*ring-char-0*, *division-by-zero*,*field*$\})$ **by** *simp*
also have $\ldots = INum\ (x +_N y)$ **by** *simp*
finally show *?thesis* **using** *isnormNum-unique*[*OF n*] **by** *simp*
qed

lemma *Nadd-assoc*: $x +_N y +_N z = x +_N (y +_N z)$
proof−
have *n*: *isnormNum* $(x +_N y +_N z)$ *isnormNum* $(x +_N (y +_N z))$ **by** *simp-all*
have *INum* $(x +_N y +_N z) = (INum\ (x +_N (y +_N z)) :: 'a :: \{$*ring-char-0*, *division-by-zero*,*field*$\})$ **by** *simp*
with *isnormNum-unique*[*OF n*] **show** *?thesis* **by** *simp*
qed

lemma *Nmul-commute*: *isnormNum* $x \Longrightarrow$ *isnormNum* $y \Longrightarrow x *_N y = y *_N x$
by (*simp add*: *Nmul-def split-def Let-def igcd-commute mult-commute*)

lemma *Nmul-assoc*: **assumes** *nx*: *isnormNum x* **and** *ny*:*isnormNum y* **and** *nz*:*isnormNum z*
shows $x *_N y *_N z = x *_N (y *_N z)$
proof−
from *nx ny nz* **have** *n*: *isnormNum* $(x *_N y *_N z)$ *isnormNum* $(x *_N (y *_N z))$

by *simp-all*
have *INum* $(x +_N y +_N z) = (INum\ (x +_N (y +_N z)) :: 'a :: \{$*ring-char-0*, *division-by-zero*,*field*$\})$ **by** *simp*
with *isnormNum-unique*[*OF n*] **show** *?thesis* **by** *simp*
qed

lemma *Nsub0*: **assumes** *x*: *isnormNum x* **and** *y*:*isnormNum y* **shows** $(x -_N y = 0_N) = (x = y)$
proof−
{fix $h :: 'a :: \{$*ring-char-0*,*division-by-zero*,*ordered-field*$\}$
from *isnormNum-unique*[**where** $?'a = 'a$, *OF Nsub-normN*[*OF y*], **where** $y=0_N$]
have $(x -_N y = 0_N) = (INum\ (x -_N y) = (INum\ 0_N :: 'a))$ **by** *simp*
also have $\ldots = (INum\ x = (INum\ y :: 'a))$ **by** *simp*
also have $\ldots = (x = y)$ **using** *x y* **by** *simp*
finally show *?thesis* **.}**
qed

lemma *Nmul0*[*simp*]: $c *_N 0_N = 0_N \quad 0_N *_N c = 0_N$
by (*simp-all add*: *Nmul-def Let-def split-def*)

lemma *Nmul-eq0*[*simp*]: **assumes** *nx*:*isnormNum x* **and** *ny*: *isnormNum y*
shows $(x*_N\ y = 0_N) = (x = 0_N \vee y = 0_N)$
proof−
{fix *h* :: *'a* :: {*ring-char-0*,*division-by-zero*,*ordered-field*}
have $\exists\ a\ b\ a'\ b'.\ x = (a,b) \wedge y= (a',b')$ **by** *auto*
then obtain *a b a' b'* **where** *xy*[*simp*]: $x = (a,b)\ y = (a',b')$ **by** *blast*
have *n0*: *isnormNum* 0_N **by** *simp*
show *?thesis* **using** *nx ny*
apply (*simp only*: *isnormNum-unique*[**where** *?'a* = *'a*, *OF Nmul-normN*[*OF nx ny*] *n0*, *symmetric*] *Nmul*[**where** *?'a* = *'a*])
apply (*simp add*: *INum-def split-def isnormNum-def fst-conv snd-conv*)
apply (*cases a=0*,*simp-all*)
apply (*cases a'=0*,*simp-all*)
done }
qed
lemma *Nneg-Nneg*[*simp*]: $\sim_N\ (\sim_N\ c) = c$
by (*simp add*: *Nneg-def split-def*)

lemma *Nmul1*[*simp*]:
$isnormNum\ c \Longrightarrow 1_N *_N c = c$
$isnormNum\ c \Longrightarrow c *_N 1_N = c$
apply (*simp-all add*: *Nmul-def Let-def split-def isnormNum-def*)
by (*cases fst c = 0*, *simp-all*,*cases c*, *simp-all*)+

end

4 Rational: Rational numbers

theory *Rational*
imports ~~/*src*/*HOL*/*Library*/*Abstract-Rat*
uses (*rat-arith.ML*)
begin

4.1 Rational numbers

4.1.1 Equivalence of fractions

definition
fraction :: (*int* × *int*) *set* **where**
fraction = {*x*. *snd x* ≠ *0*}

definition
ratrel :: ((*int* × *int*) × (*int* × *int*)) *set* **where**
ratrel = {(*x*,*y*). *snd x* ≠ *0* ∧ *snd y* ≠ *0* ∧ *fst x* ∗ *snd y* = *fst y* ∗ *snd x*}

lemma *fraction-iff* [*simp*]: (*x* ∈ *fraction*) = (*snd x* ≠ *0*)
by (*simp add*: *fraction-def*)

lemma *ratrel-iff* [*simp*]:

$((x,y) \in ratrel) =$
$(snd\ x \neq 0 \wedge snd\ y \neq 0 \wedge fst\ x * snd\ y = fst\ y * snd\ x)$
by (*simp add*: *ratrel-def*)

lemma *refl-ratrel*: *refl fraction ratrel*
by (*auto simp add*: *refl-def fraction-def ratrel-def*)

lemma *sym-ratrel*: *sym ratrel*
by (*simp add*: *ratrel-def sym-def*)

lemma *trans-ratrel-lemma*:
assumes *1*: $a * b' = a' * b$
assumes *2*: $a' * b'' = a'' * b'$
assumes *3*: $b' \neq (0::int)$
shows $a * b'' = a'' * b$
proof −
have $b' * (a * b'') = b'' * (a * b')$ **by** *simp*
also note *1*
also have $b'' * (a' * b) = b * (a' * b'')$ **by** *simp*
also note *2*
also have $b * (a'' * b') = b' * (a'' * b)$ **by** *simp*
finally have $b' * (a * b'') = b' * (a'' * b)$ **.**
with *3* **show** $a * b'' = a'' * b$ **by** *simp*
qed

lemma *trans-ratrel*: *trans ratrel*
by (*auto simp add*: *trans-def elim*: *trans-ratrel-lemma*)

lemma *equiv-ratrel*: *equiv fraction ratrel*
by (*rule equiv.intro* [*OF refl-ratrel sym-ratrel trans-ratrel*])

lemmas *equiv-ratrel-iff* [*iff*] = *eq-equiv-class-iff* [*OF equiv-ratrel*]

lemma *equiv-ratrel-iff2*:
$[\![snd\ x \neq 0; snd\ y \neq 0]\!]$
$\Longrightarrow (ratrel\ ``\ \{x\} = ratrel\ ``\ \{y\}) = ((x,y) \in ratrel)$
by (*rule eq-equiv-class-iff* [*OF equiv-ratrel*], *simp-all*)

4.1.2 The type of rational numbers

typedef (*Rat*) *rat* = *fraction*//*ratrel*
proof
have $(0,1) \in fraction$ **by** (*simp add*: *fraction-def*)
thus $ratrel``\{(0,1)\} \in fraction//ratrel$ **by** (*rule quotientI*)
qed

lemma *ratrel-in-Rat* [*simp*]: $snd\ x \neq 0 \Longrightarrow ratrel``\{x\} \in Rat$
by (*simp add*: *Rat-def quotientI*)

declare *Abs-Rat-inject* [*simp*] *Abs-Rat-inverse* [*simp*]

definition
Fract :: *int* ⇒ *int* ⇒ *rat* **where**
[*code func del*]: *Fract a b* = *Abs-Rat* (*ratrel*``{(*a*,*b*)})

lemma *Fract-zero*:
Fract k 0 = *Fract l 0*
by (*simp add*: *Fract-def ratrel-def*)

theorem *Rat-cases* [*case-names Fract*, *cases type*: *rat*]:
(!!*a b*. *q* = *Fract a b* ==> *b* ≠ *0* ==> *C*) ==> *C*
by (*cases q*) (*clarsimp simp add*: *Fract-def Rat-def fraction-def quotient-def*)

theorem *Rat-induct* [*case-names Fract*, *induct type*: *rat*]:
(!!*a b*. *b* ≠ *0* ==> *P* (*Fract a b*)) ==> *P q*
by (*cases q*) *simp*

4.1.3 Congruence lemmas

lemma *add-congruent2*:
(λ*x y*. *ratrel*``{(*fst x* ∗ *snd y* + *fst y* ∗ *snd x*, *snd x* ∗ *snd y*)})
respects2 ratrel
apply (*rule equiv-ratrel* [*THEN congruent2-commuteI*])
apply (*simp-all add*: *left-distrib*)
done

lemma *minus-congruent*:
(λ*x*. *ratrel*``{(− *fst x*, *snd x*)}) *respects ratrel*
by (*simp add*: *congruent-def*)

lemma *mult-congruent2*:
(λ*x y*. *ratrel*``{(*fst x* ∗ *fst y*, *snd x* ∗ *snd y*)}) *respects2 ratrel*
by (*rule equiv-ratrel* [*THEN congruent2-commuteI*], *simp-all*)

lemma *inverse-congruent*:
(λ*x*. *ratrel*``{*if fst x=0 then* (*0*,*1*) *else* (*snd x*, *fst x*)}) *respects ratrel*
by (*auto simp add*: *congruent-def mult-commute*)

lemma *le-congruent2*:
(λ*x y*. {(*fst x* ∗ *snd y*)∗(*snd x* ∗ *snd y*) ≤ (*fst y* ∗ *snd x*)∗(*snd x* ∗ *snd y*)})
respects2 ratrel
proof (*clarsimp simp add*: *congruent2-def*)
fix *a b a′ b′ c d c′ d′*::*int*
assume *neq*: *b* ≠ *0* *b′* ≠ *0* *d* ≠ *0* *d′* ≠ *0*
assume *eq1*: *a* ∗ *b′* = *a′* ∗ *b*
assume *eq2*: *c* ∗ *d′* = *c′* ∗ *d*

let *?le* = $\lambda a\ b\ c\ d.\ ((a * d) * (b * d) \leq (c * b) * (b * d))$
{
fix *a b c d x* :: *int* **assume** *x*: $x \neq 0$
have *?le a b c d* = *?le* $(a * x)$ $(b * x)$ *c d*
proof −
from *x* **have** $0 < x * x$ **by** (*auto simp add*: *zero-less-mult-iff*)
hence *?le a b c d* =
$((a * d) * (b * d) * (x * x) \leq (c * b) * (b * d) * (x * x))$
by (*simp add*: *mult-le-cancel-right*)
also have ... = *?le* $(a * x)$ $(b * x)$ *c d*
by (*simp add*: *mult-ac*)
finally show *?thesis* **.**
qed
} note *le-factor* = *this*

let *?D* = $b * d$ **and** *?D'* = $b' * d'$
from *neq* **have** *D*: $?D \neq 0$ **by** *simp*
from *neq* **have** $?D' \neq 0$ **by** *simp*
hence *?le a b c d* = *?le* $(a * ?D')$ $(b * ?D')$ *c d*
by (*rule le-factor*)
also have ... = $((a * b') * ?D * ?D' * d * d' \leq (c * d') * ?D * ?D' * b * b')$
by (*simp add*: *mult-ac*)
also have ... = $((a' * b) * ?D * ?D' * d * d' \leq (c' * d) * ?D * ?D' * b * b')$
by (*simp only*: *eq1 eq2*)
also have ... = *?le* $(a' * ?D)$ $(b' * ?D)$ *c' d'*
by (*simp add*: *mult-ac*)
also from *D* **have** ... = *?le a' b' c' d'*
by (*rule le-factor* [*symmetric*])
finally show *?le a b c d* = *?le a' b' c' d'* **.**
qed

lemmas *UN-ratrel* = *UN-equiv-class* [*OF equiv-ratrel*]
lemmas *UN-ratrel2* = *UN-equiv-class2* [*OF equiv-ratrel equiv-ratrel*]

4.1.4 Standard operations on rational numbers

instantiation *rat* :: {*zero*, *one*, *plus*, *minus*, *uminus*, *times*, *inverse*, *ord*, *abs*, *sgn*}
begin

definition
Zero-rat-def [*code func del*]: *0* = *Fract 0 1*

definition
One-rat-def [*code func del*]: *1* = *Fract 1 1*

definition
add-rat-def [*code func del*]:
q + *r* =

$$\mathit{Abs\text{-}Rat}\ (\bigcup x \in \mathit{Rep\text{-}Rat}\ q.\ \bigcup y \in \mathit{Rep\text{-}Rat}\ r.\ \mathit{ratrel}``\{(\mathit{fst}\ x * \mathit{snd}\ y + \mathit{fst}\ y * \mathit{snd}\ x,\ \mathit{snd}\ x * \mathit{snd}\ y)\})$$

definition
minus-rat-def [*code func del*]:
$-\ q = \mathit{Abs\text{-}Rat}\ (\bigcup x \in \mathit{Rep\text{-}Rat}\ q.\ \mathit{ratrel}``\{(-\ \mathit{fst}\ x,\ \mathit{snd}\ x)\})$

definition
diff-rat-def [*code func del*]: $q - r = q + -\ (r::\mathit{rat})$

definition
mult-rat-def [*code func del*]:
$q * r =$
$\mathit{Abs\text{-}Rat}\ (\bigcup x \in \mathit{Rep\text{-}Rat}\ q.\ \bigcup y \in \mathit{Rep\text{-}Rat}\ r.\ \mathit{ratrel}``\{(\mathit{fst}\ x * \mathit{fst}\ y,\ \mathit{snd}\ x * \mathit{snd}\ y)\})$

definition
inverse-rat-def [*code func del*]:
$\mathit{inverse}\ q =$
$\mathit{Abs\text{-}Rat}\ (\bigcup x \in \mathit{Rep\text{-}Rat}\ q.\ \mathit{ratrel}``\{\mathit{if}\ \mathit{fst}\ x{=}0\ \mathit{then}\ (0,1)\ \mathit{else}\ (\mathit{snd}\ x,\ \mathit{fst}\ x)\})$

definition
divide-rat-def [*code func del*]: $q\ /\ r = q * \mathit{inverse}\ (r::\mathit{rat})$

definition
le-rat-def [*code func del*]:
$q \le r \longleftrightarrow \mathit{contents}\ (\bigcup x \in \mathit{Rep\text{-}Rat}\ q.\ \bigcup y \in \mathit{Rep\text{-}Rat}\ r.\ \{(\mathit{fst}\ x * \mathit{snd}\ y){*}(\mathit{snd}\ x * \mathit{snd}\ y) \le (\mathit{fst}\ y * \mathit{snd}\ x){*}(\mathit{snd}\ x * \mathit{snd}\ y)\})$

definition
less-rat-def [*code func del*]: $z < (w::\mathit{rat}) \longleftrightarrow z \le w \wedge z \ne w$

definition
abs-rat-def: $|q| = (\mathit{if}\ q < 0\ \mathit{then}\ -q\ \mathit{else}\ (q::\mathit{rat}))$

definition
sgn-rat-def: $\mathit{sgn}\ (q::\mathit{rat}) = (\mathit{if}\ q{=}0\ \mathit{then}\ 0\ \mathit{else}\ \mathit{if}\ 0{<}q\ \mathit{then}\ 1\ \mathit{else}\ -\ 1)$

instance ..

end

instantiation *rat* :: *power*
begin

primrec *power-rat*
where
rat-power-0: $q\ \hat{}\ 0 = (1::\mathit{rat})$

| *rat-power-Suc*: $q \hat{} (Suc\ n) = (q::rat) * (q \hat{} n)$

instance ..

end

theorem *eq-rat*: $b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow$
$(Fract\ a\ b = Fract\ c\ d) = (a * d = c * b)$
by (*simp add*: *Fract-def*)

theorem *add-rat*: $b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow$
$Fract\ a\ b + Fract\ c\ d = Fract\ (a * d + c * b)\ (b * d)$
by (*simp add*: *Fract-def add-rat-def add-congruent2 UN-ratrel2*)

theorem *minus-rat*: $b \neq 0 \Longrightarrow -(Fract\ a\ b) = Fract\ (-a)\ b$
by (*simp add*: *Fract-def minus-rat-def minus-congruent UN-ratrel*)

theorem *diff-rat*: $b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow$
$Fract\ a\ b - Fract\ c\ d = Fract\ (a * d - c * b)\ (b * d)$
by (*simp add*: *diff-rat-def add-rat minus-rat*)

theorem *mult-rat*: $b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow$
$Fract\ a\ b * Fract\ c\ d = Fract\ (a * c)\ (b * d)$
by (*simp add*: *Fract-def mult-rat-def mult-congruent2 UN-ratrel2*)

theorem *inverse-rat*: $a \neq 0 \Longrightarrow b \neq 0 \Longrightarrow$
$inverse\ (Fract\ a\ b) = Fract\ b\ a$
by (*simp add*: *Fract-def inverse-rat-def inverse-congruent UN-ratrel*)

theorem *divide-rat*: $c \neq 0 \Longrightarrow b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow$
$Fract\ a\ b\ /\ Fract\ c\ d = Fract\ (a * d)\ (b * c)$
by (*simp add*: *divide-rat-def inverse-rat mult-rat*)

theorem *le-rat*: $b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow$
$(Fract\ a\ b \leq Fract\ c\ d) = ((a * d) * (b * d) \leq (c * b) * (b * d))$
by (*simp add*: *Fract-def le-rat-def le-congruent2 UN-ratrel2*)

theorem *less-rat*: $b \neq 0 \Longrightarrow d \neq 0 \Longrightarrow$
$(Fract\ a\ b < Fract\ c\ d) = ((a * d) * (b * d) < (c * b) * (b * d))$
by (*simp add*: *less-rat-def le-rat eq-rat order-less-le*)

theorem *abs-rat*: $b \neq 0 \Longrightarrow |Fract\ a\ b| = Fract\ |a|\ |b|$
by (*simp add*: *abs-rat-def minus-rat Zero-rat-def less-rat eq-rat*)
(*auto simp add*: *mult-less-0-iff zero-less-mult-iff order-le-less*
split: *abs-split*)

4.1.5 The ordered field of rational numbers

instance *rat* :: *field*

```
proof
  fix q r s :: rat
  show (q + r) + s = q + (r + s)
    by (induct q, induct r, induct s)
       (simp add: add-rat add-ac mult-ac int-distrib)
  show q + r = r + q
    by (induct q, induct r) (simp add: add-rat add-ac mult-ac)
  show 0 + q = q
    by (induct q) (simp add: Zero-rat-def add-rat)
  show (−q) + q = 0
    by (induct q) (simp add: Zero-rat-def minus-rat add-rat eq-rat)
  show q − r = q + (−r)
    by (induct q, induct r) (simp add: add-rat minus-rat diff-rat)
  show (q * r) * s = q * (r * s)
    by (induct q, induct r, induct s) (simp add: mult-rat mult-ac)
  show q * r = r * q
    by (induct q, induct r) (simp add: mult-rat mult-ac)
  show 1 * q = q
    by (induct q) (simp add: One-rat-def mult-rat)
  show (q + r) * s = q * s + r * s
    by (induct q, induct r, induct s)
       (simp add: add-rat mult-rat eq-rat int-distrib)
  show q ≠ 0 ==> inverse q * q = 1
    by (induct q) (simp add: inverse-rat mult-rat One-rat-def Zero-rat-def eq-rat)
  show q / r = q * inverse r
    by (simp add: divide-rat-def)
  show 0 ≠ (1::rat)
    by (simp add: Zero-rat-def One-rat-def eq-rat)
qed

instance rat :: linorder
proof
  fix q r s :: rat
  {
    assume q ≤ r and r ≤ s
    show q ≤ s
    proof (insert prems, induct q, induct r, induct s)
      fix a b c d e f :: int
      assume neq: b ≠ 0  d ≠ 0  f ≠ 0
      assume 1: Fract a b ≤ Fract c d and 2: Fract c d ≤ Fract e f
      show Fract a b ≤ Fract e f
      proof −
        from neq obtain bb: 0 < b * b and dd: 0 < d * d and ff: 0 < f * f
          by (auto simp add: zero-less-mult-iff linorder-neq-iff)
        have (a * d) * (b * d) * (f * f) ≤ (c * b) * (b * d) * (f * f)
        proof −
          from neq 1 have (a * d) * (b * d) ≤ (c * b) * (b * d)
            by (simp add: le-rat)
          with ff show ?thesis by (simp add: mult-le-cancel-right)
```

```
      qed
      also have ... = (c * f) * (d * f) * (b * b)
        by (simp only: mult-ac)
      also have ... ≤ (e * d) * (d * f) * (b * b)
      proof −
        from neq 2 have (c * f) * (d * f) ≤ (e * d) * (d * f)
          by (simp add: le-rat)
        with bb show ?thesis by (simp add: mult-le-cancel-right)
      qed
      finally have (a * f) * (b * f) * (d * d) ≤ e * b * (b * f) * (d * d)
        by (simp only: mult-ac)
      with dd have (a * f) * (b * f) ≤ (e * b) * (b * f)
        by (simp add: mult-le-cancel-right)
      with neq show ?thesis by (simp add: le-rat)
    qed
  qed
 next
  assume q ≤ r and r ≤ q
  show q = r
  proof (insert prems, induct q, induct r)
    fix a b c d :: int
    assume neq: b ≠ 0  d ≠ 0
    assume 1: Fract a b ≤ Fract c d and 2: Fract c d ≤ Fract a b
    show Fract a b = Fract c d
    proof −
      from neq 1 have (a * d) * (b * d) ≤ (c * b) * (b * d)
        by (simp add: le-rat)
      also have ... ≤ (a * d) * (b * d)
      proof −
        from neq 2 have (c * b) * (d * b) ≤ (a * d) * (d * b)
          by (simp add: le-rat)
        thus ?thesis by (simp only: mult-ac)
      qed
      finally have (a * d) * (b * d) = (c * b) * (b * d) .
      moreover from neq have b * d ≠ 0 by simp
      ultimately have a * d = c * b by simp
      with neq show ?thesis by (simp add: eq-rat)
    qed
  qed
 next
  show q ≤ q
    by (induct q) (simp add: le-rat)
  show (q < r) = (q ≤ r ∧ q ≠ r)
    by (simp only: less-rat-def)
  show q ≤ r ∨ r ≤ q
    by (induct q, induct r)
       (simp add: le-rat mult-commute, rule linorder-linear)
 }
qed
```

instantiation *rat* :: *distrib-lattice*
begin

definition
(*inf* :: *rat* ⇒ *rat* ⇒ *rat*) = *min*

definition
(*sup* :: *rat* ⇒ *rat* ⇒ *rat*) = *max*

instance
by *default* (*auto simp add*: *min-max.sup-inf-distrib1 inf-rat-def sup-rat-def*)

end

instance *rat* :: *ordered-field*
proof
fix *q r s* :: *rat*
show *q* ≤ *r* ==> *s* + *q* ≤ *s* + *r*
proof (*induct q*, *induct r*, *induct s*)
fix *a b c d e f* :: *int*
assume *neq*: *b* ≠ *0* *d* ≠ *0* *f* ≠ *0*
assume *le*: *Fract a b* ≤ *Fract c d*
show *Fract e f* + *Fract a b* ≤ *Fract e f* + *Fract c d*
proof −
let *?F* = *f* ∗ *f* **from** *neq* **have** *F*: *0* < *?F*
by (*auto simp add*: *zero-less-mult-iff*)
from *neq le* **have** (*a* ∗ *d*) ∗ (*b* ∗ *d*) ≤ (*c* ∗ *b*) ∗ (*b* ∗ *d*)
by (*simp add*: *le-rat*)
with *F* **have** (*a* ∗ *d*) ∗ (*b* ∗ *d*) ∗ *?F* ∗ *?F* ≤ (*c* ∗ *b*) ∗ (*b* ∗ *d*) ∗ *?F* ∗ *?F*
by (*simp add*: *mult-le-cancel-right*)
with *neq* **show** *?thesis* **by** (*simp add*: *add-rat le-rat mult-ac int-distrib*)
qed
qed
show *q* < *r* ==> *0* < *s* ==> *s* ∗ *q* < *s* ∗ *r*
proof (*induct q*, *induct r*, *induct s*)
fix *a b c d e f* :: *int*
assume *neq*: *b* ≠ *0* *d* ≠ *0* *f* ≠ *0*
assume *le*: *Fract a b* < *Fract c d*
assume *gt*: *0* < *Fract e f*
show *Fract e f* ∗ *Fract a b* < *Fract e f* ∗ *Fract c d*
proof −
let *?E* = *e* ∗ *f* **and** *?F* = *f* ∗ *f*
from *neq gt* **have** *0* < *?E*
by (*auto simp add*: *Zero-rat-def less-rat le-rat order-less-le eq-rat*)
moreover from *neq* **have** *0* < *?F*
by (*auto simp add*: *zero-less-mult-iff*)
moreover from *neq le* **have** (*a* ∗ *d*) ∗ (*b* ∗ *d*) < (*c* ∗ *b*) ∗ (*b* ∗ *d*)
by (*simp add*: *less-rat*)

```
    ultimately have (a * d) * (b * d) * ?E * ?F < (c * b) * (b * d) * ?E * ?F
      by (simp add: mult-less-cancel-right)
    with neq show ?thesis
      by (simp add: less-rat mult-rat mult-ac)
   qed
  qed
  show |q| = (if q < 0 then -q else q)
   by (simp only: abs-rat-def)
qed (auto simp: sgn-rat-def)

instance rat :: division-by-zero
proof
  show inverse 0 = (0::rat)
   by (simp add: Zero-rat-def Fract-def inverse-rat-def
                 inverse-congruent UN-ratrel)
qed

instance rat :: recpower
proof
  fix q :: rat
  fix n :: nat
  show q ^ 0 = 1 by simp
  show q ^ (Suc n) = q * (q ^ n) by simp
qed
```

4.2 Various Other Results

```
lemma minus-rat-cancel [simp]: b ≠ 0 ==> Fract (-a) (-b) = Fract a b
by (simp add: eq-rat)

theorem Rat-induct-pos [case-names Fract, induct type: rat]:
  assumes step: !!a b. 0 < b ==> P (Fract a b)
   shows P q
proof (cases q)
  have step': !!a b. b < 0 ==> P (Fract a b)
  proof -
   fix a::int and b::int
   assume b: b < 0
   hence 0 < -b by simp
   hence P (Fract (-a) (-b)) by (rule step)
   thus P (Fract a b) by (simp add: order-less-imp-not-eq [OF b])
  qed
  case (Fract a b)
  thus P q by (force simp add: linorder-neq-iff step step')
qed

lemma zero-less-Fract-iff:
    0 < b ==> (0 < Fract a b) = (0 < a)
by (simp add: Zero-rat-def less-rat order-less-imp-not-eq2 zero-less-mult-iff)
```

lemma *Fract-add-one*: $n \neq 0 \Longrightarrow$ *Fract* $(m + n)$ $n =$ *Fract* m $n + 1$
apply (*insert add-rat* [*of concl*: m n 1 1])
apply (*simp add*: *One-rat-def* [*symmetric*])
done

lemma *of-nat-rat*: *of-nat* $k =$ *Fract* (*of-nat* k) 1
by (*induct* k) (*simp-all add*: *Zero-rat-def One-rat-def add-rat*)

lemma *of-int-rat*: *of-int* $k =$ *Fract* k 1
by (*cases* k *rule*: *int-diff-cases*, *simp add*: *of-nat-rat diff-rat*)

lemma *Fract-of-nat-eq*: *Fract* (*of-nat* k) 1 = *of-nat* k
by (*rule of-nat-rat* [*symmetric*])

lemma *Fract-of-int-eq*: *Fract* k 1 = *of-int* k
by (*rule of-int-rat* [*symmetric*])

lemma *Fract-of-int-quotient*: *Fract* k l = (*if* $l = 0$ *then Fract* 1 0 *else of-int* k / *of-int* l)
by (*auto simp add*: *Fract-zero Fract-of-int-eq* [*symmetric*] *divide-rat*)

4.3 Numerals and Arithmetic

instantiation *rat* :: *number-ring*
begin

definition
rat-number-of-def [*code func del*]: *number-of* w = (*of-int* w :: *rat*)

instance
by *default* (*simp add*: *rat-number-of-def*)

end

use *rat-arith.ML*
declaration ⟪ *K rat-arith-setup* ⟫

4.4 Embedding from Rationals to other Fields

class *field-char-0* = *field* + *ring-char-0*

instance *ordered-field* < *field-char-0* **..**

definition
of-rat :: *rat* ⇒ 'a::*field-char-0*
where
[*code func del*]: *of-rat* q = *contents* ($\bigcup(a,b) \in$ *Rep-Rat* q. {*of-int* a / *of-int* b})

lemma *of-rat-congruent*:

```
  (λ(a, b). {of-int a / of-int b::'a::field-char-0}) respects ratrel
apply (rule congruent.intro)
apply (clarsimp simp add: nonzero-divide-eq-eq nonzero-eq-divide-eq)
apply (simp only: of-int-mult [symmetric])
done

lemma of-rat-rat:
  b ≠ 0 ⟹ of-rat (Fract a b) = of-int a / of-int b
unfolding Fract-def of-rat-def
by (simp add: UN-ratrel of-rat-congruent)

lemma of-rat-0 [simp]: of-rat 0 = 0
by (simp add: Zero-rat-def of-rat-rat)

lemma of-rat-1 [simp]: of-rat 1 = 1
by (simp add: One-rat-def of-rat-rat)

lemma of-rat-add: of-rat (a + b) = of-rat a + of-rat b
by (induct a, induct b, simp add: add-rat of-rat-rat add-frac-eq)

lemma of-rat-minus: of-rat (- a) = - of-rat a
by (induct a, simp add: minus-rat of-rat-rat)

lemma of-rat-diff: of-rat (a - b) = of-rat a - of-rat b
by (simp only: diff-minus of-rat-add of-rat-minus)

lemma of-rat-mult: of-rat (a * b) = of-rat a * of-rat b
apply (induct a, induct b, simp add: mult-rat of-rat-rat)
apply (simp add: divide-inverse nonzero-inverse-mult-distrib mult-ac)
done

lemma nonzero-of-rat-inverse:
  a ≠ 0 ⟹ of-rat (inverse a) = inverse (of-rat a)
apply (rule inverse-unique [symmetric])
apply (simp add: of-rat-mult [symmetric])
done

lemma of-rat-inverse:
  (of-rat (inverse a)::'a::{field-char-0,division-by-zero}) =
   inverse (of-rat a)
by (cases a = 0, simp-all add: nonzero-of-rat-inverse)

lemma nonzero-of-rat-divide:
  b ≠ 0 ⟹ of-rat (a / b) = of-rat a / of-rat b
by (simp add: divide-inverse of-rat-mult nonzero-of-rat-inverse)

lemma of-rat-divide:
  (of-rat (a / b)::'a::{field-char-0,division-by-zero})
   = of-rat a / of-rat b
```

by (*cases b = 0*, *simp-all add*: *nonzero-of-rat-divide*)

lemma *of-rat-power*:
(*of-rat* (*a ^ n*)::*'a*::{*field-char-0*,*recpower*}) = *of-rat a ^ n*
by (*induct n*) (*simp-all add*: *of-rat-mult power-Suc*)

lemma *of-rat-eq-iff* [*simp*]: (*of-rat a = of-rat b*) = (*a = b*)
apply (*induct a*, *induct b*)
apply (*simp add*: *of-rat-rat eq-rat*)
apply (*simp add*: *nonzero-divide-eq-eq nonzero-eq-divide-eq*)
apply (*simp only*: *of-int-mult* [*symmetric*] *of-int-eq-iff*)
done

lemmas *of-rat-eq-0-iff* [*simp*] = *of-rat-eq-iff* [*of - 0*, *simplified*]

lemma *of-rat-eq-id* [*simp*]: *of-rat* = (*id* :: *rat* ⇒ *rat*)
proof
fix *a*
show *of-rat a = id a*
by (*induct a*)
(*simp add*: *of-rat-rat divide-rat Fract-of-int-eq* [*symmetric*])
qed

Collapse nested embeddings

lemma *of-rat-of-nat-eq* [*simp*]: *of-rat* (*of-nat n*) = *of-nat n*
by (*induct n*) (*simp-all add*: *of-rat-add*)

lemma *of-rat-of-int-eq* [*simp*]: *of-rat* (*of-int z*) = *of-int z*
by (*cases z rule*: *int-diff-cases*, *simp add*: *of-rat-diff*)

lemma *of-rat-number-of-eq* [*simp*]:
of-rat (*number-of w*) = (*number-of w* :: *'a*::{*number-ring*,*field-char-0*})
by (*simp add*: *number-of-eq*)

lemmas *zero-rat* = *Zero-rat-def*
lemmas *one-rat* = *One-rat-def*

abbreviation
rat-of-nat :: *nat* ⇒ *rat*
where
rat-of-nat ≡ *of-nat*

abbreviation
rat-of-int :: *int* ⇒ *rat*
where
rat-of-int ≡ *of-int*

4.5 Implementation of rational numbers as pairs of integers

definition
Rational :: *int* × *int* ⇒ *rat*
where
Rational = *INum*

code-datatype *Rational*

lemma *Rational-simp*:
Rational (*k*, *l*) = *rat-of-int k* / *rat-of-int l*
unfolding *Rational-def INum-def* **by** *simp*

lemma *Rational-zero* [*simp*]: *Rational* 0_N = *0*
by (*simp add*: *Rational-simp*)

lemma *Rational-lit* [*simp*]: *Rational* i_N = *rat-of-int i*
by (*simp add*: *Rational-simp*)

lemma *zero-rat-code* [*code*, *code unfold*]:
0 = *Rational* 0_N **by** *simp*
declare *zero-rat-code* [*symmetric*, *code post*]

lemma *one-rat-code* [*code*, *code unfold*]:
1 = *Rational* 1_N **by** *simp*
declare *one-rat-code* [*symmetric*, *code post*]

lemma [*code unfold*, *symmetric*, *code post*]:
number-of k = *rat-of-int* (*number-of k*)
by (*simp add*: *number-of-is-id rat-number-of-def*)

definition
[*code func del*]: *Fract'* (*b*::*bool*) *k l* = *Fract k l*

lemma [*code*]:
Fract k l = *Fract'* (*l* ≠ *0*) *k l*
unfolding *Fract'-def* **..**

lemma [*code*]:
Fract' True k l = (*if l* ≠ *0 then Rational* (*k*, *l*) *else Fract 1 0*)
by (*simp add*: *Fract'-def Rational-simp Fract-of-int-quotient* [*of k l*])

lemma [*code*]:
of-rat (*Rational* (*k*, *l*)) = (*if l* ≠ *0 then of-int k* / *of-int l else 0*)
by (*cases l* = *0*)
(*auto simp add*: *Rational-simp of-rat-rat* [*simplified Fract-of-int-quotient* [*of k l*], *symmetric*])

instantiation *rat* :: *eq*
begin

definition [*code func del*]: *eq-class.eq* (*r*::*rat*) *s* ⟷ *r* = *s*

instance by *default* (*simp add*: *eq-rat-def*)

lemma *rat-eq-code* [*code*]: *eq-class.eq* (*Rational x*) (*Rational y*) ⟷ *eq-class.eq* (*normNum x*) (*normNum y*)
unfolding *Rational-def INum-normNum-iff eq* **..**

end

lemma *rat-less-eq-code* [*code*]: *Rational x* ≤ *Rational y* ⟷ *normNum x* ≤$_N$ *normNum y*
proof −
have *normNum x* ≤$_N$ *normNum y* ⟷ *Rational* (*normNum x*) ≤ *Rational* (*normNum y*)
by (*simp add*: *Rational-def del*: *normNum*)
also have ... = (*Rational x* ≤ *Rational y*) **by** (*simp add*: *Rational-def*)
finally show *?thesis* **by** *simp*
qed

lemma *rat-less-code* [*code*]: *Rational x* < *Rational y* ⟷ *normNum x* <$_N$ *normNum y*
proof −
have *normNum x* <$_N$ *normNum y* ⟷ *Rational* (*normNum x*) < *Rational* (*normNum y*)
by (*simp add*: *Rational-def del*: *normNum*)
also have ... = (*Rational x* < *Rational y*) **by** (*simp add*: *Rational-def*)
finally show *?thesis* **by** *simp*
qed

lemma *rat-add-code* [*code*]: *Rational x* + *Rational y* = *Rational* (*x* +$_N$ *y*)
unfolding *Rational-def* **by** *simp*

lemma *rat-mul-code* [*code*]: *Rational x* ∗ *Rational y* = *Rational* (*x* ∗$_N$ *y*)
unfolding *Rational-def* **by** *simp*

lemma *rat-neg-code* [*code*]: − *Rational x* = *Rational* (~$_N$ *x*)
unfolding *Rational-def* **by** *simp*

lemma *rat-sub-code* [*code*]: *Rational x* − *Rational y* = *Rational* (*x* −$_N$ *y*)
unfolding *Rational-def* **by** *simp*

lemma *rat-inv-code* [*code*]: *inverse* (*Rational x*) = *Rational* (*Ninv x*)
unfolding *Rational-def Ninv divide-rat-def* **by** *simp*

lemma *rat-div-code* [*code*]: *Rational x* / *Rational y* = *Rational* (*x* ÷$_N$ *y*)
unfolding *Rational-def* **by** *simp*

Setup for SML code generator

```
types-code
  rat ((int */ int))
attach (term-of) ⟪
fun term-of-rat (p, q) =
  let
    val rT = Type (Rational.rat, [])
  in
    if q = 1 orelse p = 0 then HOLogic.mk-number rT p
    else @{term op / :: rat ⇒ rat ⇒ rat} $
      HOLogic.mk-number rT p $ HOLogic.mk-number rT q
  end;
⟫
attach (test) ⟪
fun gen-rat i =
  let
    val p = random-range 0 i;
    val q = random-range 1 (i + 1);
    val g = Integer.gcd p q;
    val p' = p div g;
    val q' = q div g;
    val r = (if one-of [true, false] then p' else ~ p',
      if p' = 0 then 0 else q')
  in
    (r, fn () => term-of-rat r)
  end;
⟫

consts-code
  Rational ((-))

consts-code
  of-int :: int ⇒ rat (⟨module⟩rat'-of'-int)
attach ⟪
fun rat-of-int 0 = (0, 0)
  | rat-of-int i = (i, 1);
⟫

end
```

5 PReal: Positive real numbers

```
theory PReal
imports Rational
begin
```

Could be generalized and moved to *Ring-and-Field*

```
lemma add-eq-exists: ∃x. a+x = (b::rat)
by (rule-tac x=b−a in exI, simp)
```

definition
cut :: *rat set* => *bool* **where**
cut A = ({} ⊂ *A* &
A < {*r*. *0* < *r*} &
(∀ *y* ∈ *A*. ((∀ *z*. *0*<*z* & *z* < *y* −−> *z* ∈ *A*) & (∃ *u* ∈ *A*. *y* < *u*))))

lemma *cut-of-rat*:
assumes *q*: *0* < *q* **shows** *cut* {*r*::*rat*. *0* < *r* & *r* < *q*} (**is** *cut ?A*)
proof −
from *q* **have** *pos*: *?A* < {*r*. *0* < *r*} **by** *force*
have *nonempty*: {} ⊂ *?A*
proof
show {} ⊆ *?A* **by** *simp*
show {} ≠ *?A*
by (*force simp only*: *q eq-commute* [*of* {}] *interval-empty-iff*)
qed
show *?thesis*
by (*simp add*: *cut-def pos nonempty*,
blast dest: *dense intro*: *order-less-trans*)
qed

typedef *preal* = {*A*. *cut A*}
by (*blast intro*: *cut-of-rat* [*OF zero-less-one*])

definition
preal-of-rat :: *rat* => *preal* **where**
preal-of-rat q = *Abs-preal* {*x*::*rat*. *0* < *x* & *x* < *q*}

definition
psup :: *preal set* => *preal* **where**
psup P = *Abs-preal* (⋃ *X* ∈ *P*. *Rep-preal X*)

definition
add-set :: [*rat set*,*rat set*] => *rat set* **where**
add-set A B = {*w*. ∃ *x* ∈ *A*. ∃ *y* ∈ *B*. *w* = *x* + *y*}

definition
diff-set :: [*rat set*,*rat set*] => *rat set* **where**
diff-set A B = {*w*. ∃ *x*. *0* < *w* & *0* < *x* & *x* ∉ *B* & *x* + *w* ∈ *A*}

definition
mult-set :: [*rat set*,*rat set*] => *rat set* **where**
mult-set A B = {*w*. ∃ *x* ∈ *A*. ∃ *y* ∈ *B*. *w* = *x* ∗ *y*}

definition
inverse-set :: *rat set* => *rat set* **where**
inverse-set A = {*x*. ∃ *y*. *0* < *x* & *x* < *y* & *inverse y* ∉ *A*}

instantiation *preal* :: {*ord*, *plus*, *minus*, *times*, *inverse*, *one*}
begin

definition
preal-less-def:
$R < S == \textit{Rep-preal}\ R < \textit{Rep-preal}\ S$

definition
preal-le-def:
$R \leq S == \textit{Rep-preal}\ R \subseteq \textit{Rep-preal}\ S$

definition
preal-add-def:
$R + S == \textit{Abs-preal}\ (\textit{add-set}\ (\textit{Rep-preal}\ R)\ (\textit{Rep-preal}\ S))$

definition
preal-diff-def:
$R - S == \textit{Abs-preal}\ (\textit{diff-set}\ (\textit{Rep-preal}\ R)\ (\textit{Rep-preal}\ S))$

definition
preal-mult-def:
$R * S == \textit{Abs-preal}\ (\textit{mult-set}\ (\textit{Rep-preal}\ R)\ (\textit{Rep-preal}\ S))$

definition
preal-inverse-def:
$\textit{inverse}\ R == \textit{Abs-preal}\ (\textit{inverse-set}\ (\textit{Rep-preal}\ R))$

definition $R\ /\ S = R * \textit{inverse}\ (S::\textit{preal})$

definition
preal-one-def:
$1 == \textit{preal-of-rat}\ 1$

instance ..

end

Reduces equality on abstractions to equality on representatives

declare *Abs-preal-inject* [*simp*]
declare *Abs-preal-inverse* [*simp*]

lemma *rat-mem-preal*: $0 < q \Longrightarrow \{r::\textit{rat}.\ 0 < r\ \&\ r < q\} \in \textit{preal}$
by (*simp add*: *preal-def cut-of-rat*)

lemma *preal-nonempty*: $A \in \textit{preal} \Longrightarrow \exists x \in A.\ 0 < x$
by (*unfold preal-def cut-def*, *blast*)

lemma *preal-Ex-mem*: $A \in \textit{preal} \Longrightarrow \exists x.\ x \in A$

by (*drule preal-nonempty*, *fast*)

lemma *preal-imp-psubset-positives*: $A \in preal \Longrightarrow A < \{r.\ 0 < r\}$
by (*force simp add*: *preal-def cut-def*)

lemma *preal-exists-bound*: $A \in preal \Longrightarrow \exists x.\ 0 < x\ \&\ x \notin A$
by (*drule preal-imp-psubset-positives*, *auto*)

lemma *preal-exists-greater*: $[|\ A \in preal;\ y \in A\ |] \Longrightarrow \exists u \in A.\ y < u$
by (*unfold preal-def cut-def*, *blast*)

lemma *preal-downwards-closed*: $[|\ A \in preal;\ y \in A;\ 0 < z;\ z < y\ |] \Longrightarrow z \in A$
by (*unfold preal-def cut-def*, *blast*)

Relaxing the final premise

lemma *preal-downwards-closed'*:
$[|\ A \in preal;\ y \in A;\ 0 < z;\ z \leq y\ |] \Longrightarrow z \in A$
apply (*simp add*: *order-le-less*)
apply (*blast intro*: *preal-downwards-closed*)
done

A positive fraction not in a positive real is an upper bound. Gleason p. 122 - Remark (1)

lemma *not-in-preal-ub*:
assumes *A*: $A \in preal$
and *notx*: $x \notin A$
and *y*: $y \in A$
and *pos*: $0 < x$
shows $y < x$
proof (*cases rule*: *linorder-cases*)
assume $x<y$
with *notx* **show** *?thesis*
by (*simp add*: *preal-downwards-closed* [*OF A y*] *pos*)
next
assume $x=y$
with *notx* **and** *y* **show** *?thesis* **by** *simp*
next
assume $y<x$
thus *?thesis* **.**
qed

preal lemmas instantiated to *Rep-preal X*

lemma *mem-Rep-preal-Ex*: $\exists x.\ x \in Rep\text{-}preal\ X$
by (*rule preal-Ex-mem* [*OF Rep-preal*])

lemma *Rep-preal-exists-bound*: $\exists x>0.\ x \notin Rep\text{-}preal\ X$
by (*rule preal-exists-bound* [*OF Rep-preal*])

lemmas *not-in-Rep-preal-ub* = *not-in-preal-ub* [*OF Rep-preal*]

5.1 *preal-of-prat*: the Injection from prat to preal

lemma *rat-less-set-mem-preal*: $0 < y \Longrightarrow \{u::rat.\ 0 < u\ \&\ u < y\} \in preal$
by (*simp add*: *preal-def cut-of-rat*)

lemma *rat-subset-imp-le*:
$[|\{u::rat.\ 0 < u\ \&\ u < x\} \subseteq \{u.\ 0 < u\ \&\ u < y\};\ 0<x|] \Longrightarrow x \leq y$
apply (*simp add*: *linorder-not-less* [*symmetric*])
apply (*blast dest*: *dense intro*: *order-less-trans*)
done

lemma *rat-set-eq-imp-eq*:
$[|\{u::rat.\ 0 < u\ \&\ u < x\} = \{u.\ 0 < u\ \&\ u < y\};$
$0 < x;\ 0 < y|] \Longrightarrow x = y$
by (*blast intro*: *rat-subset-imp-le order-antisym*)

5.2 Properties of Ordering

lemma *preal-le-refl*: $w \leq (w::preal)$
by (*simp add*: *preal-le-def*)

lemma *preal-le-trans*: $[|\ i \leq j;\ j \leq k\ |] \Longrightarrow i \leq (k::preal)$
by (*force simp add*: *preal-le-def*)

lemma *preal-le-anti-sym*: $[|\ z \leq w;\ w \leq z\ |] \Longrightarrow z = (w::preal)$
apply (*simp add*: *preal-le-def*)
apply (*rule Rep-preal-inject* [*THEN iffD1*], *blast*)
done

lemma *preal-less-le*: $((w::preal) < z) = (w \leq z\ \&\ w \neq z)$
by (*simp add*: *preal-le-def preal-less-def Rep-preal-inject less-le*)

instance *preal* :: *order*
by *intro-classes*
(*assumption* |
rule preal-le-refl preal-le-trans preal-le-anti-sym preal-less-le)+

lemma *preal-imp-pos*: $[|A \in preal;\ r \in A|] \Longrightarrow 0 < r$
by (*insert preal-imp-psubset-positives*, *blast*)

lemma *preal-le-linear*: $x <= y \mid y <= (x::preal)$
apply (*auto simp add*: *preal-le-def*)
apply (*rule ccontr*)
apply (*blast dest*: *not-in-Rep-preal-ub intro*: *preal-imp-pos* [*OF Rep-preal*]
elim: *order-less-asym*)
done

instance *preal* :: *linorder*
by *intro-classes* (*rule preal-le-linear*)

instantiation *preal* :: *distrib-lattice*
begin

definition
(*inf* :: *preal* ⇒ *preal* ⇒ *preal*) = *min*

definition
(*sup* :: *preal* ⇒ *preal* ⇒ *preal*) = *max*

instance
by *intro-classes*
(*auto simp add*: *inf-preal-def sup-preal-def min-max.sup-inf-distrib1*)

end

5.3 Properties of Addition

lemma *preal-add-commute*: (*x*::*preal*) + *y* = *y* + *x*
apply (*unfold preal-add-def add-set-def*)
apply (*rule-tac f* = *Abs-preal* **in** *arg-cong*)
apply (*force simp add*: *add-commute*)
done

Lemmas for proving that addition of two positive reals gives a positive real

lemma *empty-psubset-nonempty*: *a* ∈ *A* ==> {} ⊂ *A*
by *blast*

Part 1 of Dedekind sections definition

lemma *add-set-not-empty*:
[|*A* ∈ *preal*; *B* ∈ *preal*|] ==> {} ⊂ *add-set A B*
apply (*drule preal-nonempty*)+
apply (*auto simp add*: *add-set-def*)
done

Part 2 of Dedekind sections definition. A structured version of this proof is *preal-not-mem-mult-set-Ex* below.

lemma *preal-not-mem-add-set-Ex*:
[|*A* ∈ *preal*; *B* ∈ *preal*|] ==> ∃ *q*>*0*. *q* ∉ *add-set A B*
apply (*insert preal-exists-bound* [*of A*] *preal-exists-bound* [*of B*], *auto*)
apply (*rule-tac x* = *x*+*xa* **in** *exI*)
apply (*simp add*: *add-set-def*, *clarify*)
apply (*drule* (*3*) *not-in-preal-ub*)+
apply (*force dest*: *add-strict-mono*)
done

lemma *add-set-not-rat-set*:
assumes *A*: *A* ∈ *preal*
and *B*: *B* ∈ *preal*

shows *add-set A B* < {*r*. *0* < *r*}
proof
from *preal-imp-pos* [*OF A*] *preal-imp-pos* [*OF B*]
show *add-set A B* ⊆ {*r*. *0* < *r*} **by** (*force simp add*: *add-set-def*)
next
show *add-set A B* ≠ {*r*. *0* < *r*}
by (*insert preal-not-mem-add-set-Ex* [*OF A B*], *blast*)
qed

Part 3 of Dedekind sections definition

lemma *add-set-lemma3*:
[|*A* ∈ *preal*; *B* ∈ *preal*; *u* ∈ *add-set A B*; *0* < *z*; *z* < *u*|]
==> *z* ∈ *add-set A B*
proof (*unfold add-set-def*, *clarify*)
fix *x*::*rat* **and** *y*::*rat*
assume *A*: *A* ∈ *preal*
and *B*: *B* ∈ *preal*
and [*simp*]: *0* < *z*
and *zless*: *z* < *x* + *y*
and *x*: *x* ∈ *A*
and *y*: *y* ∈ *B*
have *xpos* [*simp*]: *0*<*x* **by** (*rule preal-imp-pos* [*OF A x*])
have *ypos* [*simp*]: *0*<*y* **by** (*rule preal-imp-pos* [*OF B y*])
have *xypos* [*simp*]: *0* < *x*+*y* **by** (*simp add*: *pos-add-strict*)
let *?f* = *z*/(*x*+*y*)
have *fless*: *?f* < *1* **by** (*simp add*: *zless pos-divide-less-eq*)
show ∃ *x'* ∈ *A*. ∃ *y'*∈*B*. *z* = *x'* + *y'*
proof (*intro bexI*)
show *z* = *x*∗*?f* + *y*∗*?f*
by (*simp add*: *left-distrib* [*symmetric*] *divide-inverse mult-ac*
order-less-imp-not-eq2)
next
show *y* ∗ *?f* ∈ *B*
proof (*rule preal-downwards-closed* [*OF B y*])
show *0* < *y* ∗ *?f*
by (*simp add*: *divide-inverse zero-less-mult-iff*)
next
show *y* ∗ *?f* < *y*
by (*insert mult-strict-left-mono* [*OF fless ypos*], *simp*)
qed
next
show *x* ∗ *?f* ∈ *A*
proof (*rule preal-downwards-closed* [*OF A x*])
show *0* < *x* ∗ *?f*
by (*simp add*: *divide-inverse zero-less-mult-iff*)
next
show *x* ∗ *?f* < *x*
by (*insert mult-strict-left-mono* [*OF fless xpos*], *simp*)
qed

qed
qed

Part 4 of Dedekind sections definition

lemma *add-set-lemma4*:
[|$A \in$ *preal*; $B \in$ *preal*; $y \in$ *add-set* A B|] ==> $\exists u \in$ *add-set* A B. $y < u$
apply (*auto simp add*: *add-set-def*)
apply (*frule preal-exists-greater* [*of* A], *auto*)
apply (*rule-tac* x=$u + y$ **in** *exI*)
apply (*auto intro*: *add-strict-left-mono*)
done

lemma *mem-add-set*:
[|$A \in$ *preal*; $B \in$ *preal*|] ==> *add-set* A $B \in$ *preal*
apply (*simp* (*no-asm-simp*) *add*: *preal-def cut-def*)
apply (*blast intro*!: *add-set-not-empty add-set-not-rat-set*
add-set-lemma3 add-set-lemma4)
done

lemma *preal-add-assoc*: $((x::preal) + y) + z = x + (y + z)$
apply (*simp add*: *preal-add-def mem-add-set Rep-preal*)
apply (*force simp add*: *add-set-def add-ac*)
done

instance *preal* :: *ab-semigroup-add*
proof
fix a b c :: *preal*
show $(a + b) + c = a + (b + c)$ **by** (*rule preal-add-assoc*)
show $a + b = b + a$ **by** (*rule preal-add-commute*)
qed

lemma *preal-add-left-commute*: $x + (y + z) = y + ((x + z)::preal)$
by (*rule add-left-commute*)

Positive Real addition is an AC operator

lemmas *preal-add-ac = preal-add-assoc preal-add-commute preal-add-left-commute*

5.4 Properties of Multiplication

Proofs essentially same as for addition

lemma *preal-mult-commute*: $(x::preal) * y = y * x$
apply (*unfold preal-mult-def mult-set-def*)
apply (*rule-tac f = Abs-preal* **in** *arg-cong*)
apply (*force simp add*: *mult-commute*)
done

Multiplication of two positive reals gives a positive real.

Lemmas for proving positive reals multiplication set in *preal*

Part 1 of Dedekind sections definition

lemma *mult-set-not-empty*:
  [|*A* ∈ *preal*; *B* ∈ *preal*|] ==> {} ⊂ *mult-set A B*
apply (*insert preal-nonempty* [*of A*] *preal-nonempty* [*of B*])
apply (*auto simp add*: *mult-set-def*)
done

Part 2 of Dedekind sections definition

lemma *preal-not-mem-mult-set-Ex*:
  assumes *A*: *A* ∈ *preal*
    and *B*: *B* ∈ *preal*
   shows ∃ *q*. *0* < *q* & *q* ∉ *mult-set A B*
proof −
 from *preal-exists-bound* [*OF A*]
 obtain *x* **where** [*simp*]: *0* < *x x* ∉ *A* **by** *blast*
 from *preal-exists-bound* [*OF B*]
 obtain *y* **where** [*simp*]: *0* < *y y* ∉ *B* **by** *blast*
 show *?thesis*
 proof (*intro exI conjI*)
  show *0* < *x*∗*y* **by** (*simp add*: *mult-pos-pos*)
  show *x* ∗ *y* ∉ *mult-set A B*
  proof −
   { fix *u*::*rat* **and** *v*::*rat*
        assume *u* ∈ *A* **and** *v* ∈ *B* **and** *x*∗*y* = *u*∗*v*
        moreover
        with *prems* **have** *u*<*x* **and** *v*<*y* **by** (*blast dest*: *not-in-preal-ub*)+
        moreover
        with *prems* **have** *0*≤*v*
          by (*blast intro*: *preal-imp-pos* [*OF B*] *order-less-imp-le prems*)
        moreover
     from *calculation*
        have *u*∗*v* < *x*∗*y* **by** (*blast intro*: *mult-strict-mono prems*)
        ultimately have *False* **by** *force* **}**
   thus *?thesis* **by** (*auto simp add*: *mult-set-def*)
  qed
 qed
qed

lemma *mult-set-not-rat-set*:
 assumes *A*: *A* ∈ *preal*
  and *B*: *B* ∈ *preal*
 shows *mult-set A B* < {*r*. *0* < *r*}
proof
 show *mult-set A B* ⊆ {*r*. *0* < *r*}
  by (*force simp add*: *mult-set-def*
    intro: *preal-imp-pos* [*OF A*] *preal-imp-pos* [*OF B*] *mult-pos-pos*)
 show *mult-set A B* ≠ {*r*. *0* < *r*}
  using *preal-not-mem-mult-set-Ex* [*OF A B*] **by** *blast*
qed

Part 3 of Dedekind sections definition

lemma *mult-set-lemma3*:
[|$A \in$ *preal*; $B \in$ *preal*; $u \in$ *mult-set* A B; $0 < z$; $z < u$|]
==> $z \in$ *mult-set* A B
proof (*unfold mult-set-def*, *clarify*)
fix x::*rat* **and** y::*rat*
assume A: $A \in$ *preal*
and B: $B \in$ *preal*
and [*simp*]: $0 < z$
and *zless*: $z < x * y$
and x: $x \in A$
and y: $y \in B$
have [*simp*]: $0<y$ **by** (*rule preal-imp-pos* [*OF* B y])
show $\exists x' \in A.\ \exists y' \in B.\ z = x' * y'$
proof
show $\exists y' \in B.\ z = (z/y) * y'$
proof
show $z = (z/y)*y$
by (*simp add*: *divide-inverse mult-commute* [*of* y] *mult-assoc*
order-less-imp-not-eq2)
show $y \in B$ **by** *fact*
qed
next
show $z/y \in A$
proof (*rule preal-downwards-closed* [*OF* A x])
show $0 < z/y$
by (*simp add*: *zero-less-divide-iff*)
show $z/y < x$ **by** (*simp add*: *pos-divide-less-eq zless*)
qed
qed
qed

Part 4 of Dedekind sections definition

lemma *mult-set-lemma4*:
[|$A \in$ *preal*; $B \in$ *preal*; $y \in$ *mult-set* A B|] ==> $\exists u \in$ *mult-set* A $B.\ y < u$
apply (*auto simp add*: *mult-set-def*)
apply (*frule preal-exists-greater* [*of* A], *auto*)
apply (*rule-tac* x=$u * y$ **in** *exI*)
apply (*auto intro*: *preal-imp-pos* [*of* A] *preal-imp-pos* [*of* B]
mult-strict-right-mono)
done

lemma *mem-mult-set*:
[|$A \in$ *preal*; $B \in$ *preal*|] ==> *mult-set* A $B \in$ *preal*
apply (*simp* (*no-asm-simp*) *add*: *preal-def cut-def*)
apply (*blast intro*!: *mult-set-not-empty mult-set-not-rat-set*
mult-set-lemma3 mult-set-lemma4)
done

lemma *preal-mult-assoc*: $((x::preal) * y) * z = x * (y * z)$
apply (*simp add*: *preal-mult-def mem-mult-set Rep-preal*)
apply (*force simp add*: *mult-set-def mult-ac*)
done

instance *preal* :: *ab-semigroup-mult*
proof
  fix *a b c* :: *preal*
  show $(a * b) * c = a * (b * c)$ **by** (*rule preal-mult-assoc*)
  show $a * b = b * a$ **by** (*rule preal-mult-commute*)
qed

lemma *preal-mult-left-commute*: $x * (y * z) = y * ((x * z)::preal)$
by (*rule mult-left-commute*)

Positive Real multiplication is an AC operator

lemmas *preal-mult-ac* =
    preal-mult-assoc preal-mult-commute preal-mult-left-commute

Positive real 1 is the multiplicative identity element

lemma *preal-mult-1*: $(1::preal) * z = z$
unfolding *preal-one-def*
proof (*induct z*)
  fix *A* :: *rat set*
  assume *A*: $A \in preal$
  have $\{w.\ \exists u.\ 0 < u \wedge u < 1\ \&\ (\exists v \in A.\ w = u * v)\} = A$ (**is** *?lhs* = *A*)
  proof
    show *?lhs* $\subseteq A$
    proof *clarify*
      fix *x*::*rat* **and** *u*::*rat* **and** *v*::*rat*
      assume *upos*: $0<u$ **and** $u<1$ **and** *v*: $v \in A$
      have *vpos*: $0<v$ **by** (*rule preal-imp-pos* [*OF A v*])
      hence $u*v < 1*v$ **by** (*simp only*: *mult-strict-right-mono prems*)
      thus $u * v \in A$
        by (*force intro*: *preal-downwards-closed* [*OF A v*] *mult-pos-pos*
          upos vpos)
    qed
  next
    show *A* $\subseteq$ *?lhs*
    proof *clarify*
      fix *x*::*rat*
      assume *x*: $x \in A$
      have *xpos*: $0<x$ **by** (*rule preal-imp-pos* [*OF A x*])
      from *preal-exists-greater* [*OF A x*]
      obtain *v* **where** *v*: $v \in A$ **and** *xlessv*: $x < v$ **..**
      have *vpos*: $0<v$ **by** (*rule preal-imp-pos* [*OF A v*])
      show $\exists u.\ 0 < u \wedge u < 1 \wedge (\exists v{\in}A.\ x = u * v)$
      proof (*intro exI conjI*)

```
      show 0 < x/v
        by (simp add: zero-less-divide-iff xpos vpos)
      show x / v < 1
        by (simp add: pos-divide-less-eq vpos xlessv)
      show ∃ v'∈A. x = (x / v) * v'
      proof
        show x = (x/v)*v
          by (simp add: divide-inverse mult-assoc vpos
                        order-less-imp-not-eq2)
        show v ∈ A by fact
      qed
    qed
  qed
  thus preal-of-rat 1 * Abs-preal A = Abs-preal A
    by (simp add: preal-of-rat-def preal-mult-def mult-set-def
                  rat-mem-preal A)
qed

instance preal :: comm-monoid-mult
by intro-classes (rule preal-mult-1)

lemma preal-mult-1-right: z * (1::preal) = z
by (rule mult-1-right)
```

5.5 Distribution of Multiplication across Addition

```
lemma mem-Rep-preal-add-iff:
     (z ∈ Rep-preal(R+S)) = (∃ x ∈ Rep-preal R. ∃ y ∈ Rep-preal S. z = x + y)
apply (simp add: preal-add-def mem-add-set Rep-preal)
apply (simp add: add-set-def)
done

lemma mem-Rep-preal-mult-iff:
     (z ∈ Rep-preal(R*S)) = (∃ x ∈ Rep-preal R. ∃ y ∈ Rep-preal S. z = x * y)
apply (simp add: preal-mult-def mem-mult-set Rep-preal)
apply (simp add: mult-set-def)
done

lemma distrib-subset1:
    Rep-preal (w * (x + y)) ⊆ Rep-preal (w * x + w * y)
apply (auto simp add: Bex-def mem-Rep-preal-add-iff mem-Rep-preal-mult-iff)
apply (force simp add: right-distrib)
done

lemma preal-add-mult-distrib-mean:
  assumes a: a ∈ Rep-preal w
    and b: b ∈ Rep-preal w
    and d: d ∈ Rep-preal x
```

```
    and e: e ∈ Rep-preal y
  shows ∃ c ∈ Rep-preal w. a * d + b * e = c * (d + e)
proof
  let ?c = (a*d + b*e)/(d+e)
  have [simp]: 0<a 0<b 0<d 0<e 0<d+e
    by (blast intro: preal-imp-pos [OF Rep-preal] a b d e pos-add-strict)+
  have cpos: 0 < ?c
    by (simp add: zero-less-divide-iff zero-less-mult-iff pos-add-strict)
  show a * d + b * e = ?c * (d + e)
    by (simp add: divide-inverse mult-assoc order-less-imp-not-eq2)
  show ?c ∈ Rep-preal w
  proof (cases rule: linorder-le-cases)
    assume a ≤ b
    hence ?c ≤ b
      by (simp add: pos-divide-le-eq right-distrib mult-right-mono
                    order-less-imp-le)
    thus ?thesis by (rule preal-downwards-closed' [OF Rep-preal b cpos])
  next
    assume b ≤ a
    hence ?c ≤ a
      by (simp add: pos-divide-le-eq right-distrib mult-right-mono
                    order-less-imp-le)
    thus ?thesis by (rule preal-downwards-closed' [OF Rep-preal a cpos])
  qed
qed

lemma distrib-subset2:
     Rep-preal (w * x + w * y) ⊆ Rep-preal (w * (x + y))
apply (auto simp add: Bex-def mem-Rep-preal-add-iff mem-Rep-preal-mult-iff)
apply (drule-tac w=w and x=x and y=y in preal-add-mult-distrib-mean, auto)
done

lemma preal-add-mult-distrib2: (w * ((x::preal) + y)) = (w * x) + (w * y)
apply (rule Rep-preal-inject [THEN iffD1])
apply (rule equalityI [OF distrib-subset1 distrib-subset2])
done

lemma preal-add-mult-distrib: (((x::preal) + y) * w) = (x * w) + (y * w)
by (simp add: preal-mult-commute preal-add-mult-distrib2)

instance preal :: comm-semiring
by intro-classes (rule preal-add-mult-distrib)
```

5.6 Existence of Inverse, a Positive Real

```
lemma mem-inv-set-ex:
  assumes A: A ∈ preal shows ∃ x y. 0 < x & x < y & inverse y ∉ A
proof -
  from preal-exists-bound [OF A]
```

```
  obtain x where [simp]: 0<x x ∉ A by blast
  show ?thesis
  proof (intro exI conjI)
    show 0 < inverse (x+1)
      by (simp add: order-less-trans [OF - less-add-one])
    show inverse(x+1) < inverse x
      by (simp add: less-imp-inverse-less less-add-one)
    show inverse (inverse x) ∉ A
      by (simp add: order-less-imp-not-eq2)
  qed
qed
```

Part 1 of Dedekind sections definition

```
lemma inverse-set-not-empty:
     A ∈ preal ==> {} ⊂ inverse-set A
apply (insert mem-inv-set-ex [of A])
apply (auto simp add: inverse-set-def)
done
```

Part 2 of Dedekind sections definition

```
lemma preal-not-mem-inverse-set-Ex:
   assumes A: A ∈ preal  shows ∃ q. 0 < q & q ∉ inverse-set A
proof −
  from preal-nonempty [OF A]
  obtain x where x: x ∈ A and  xpos [simp]: 0<x ..
  show ?thesis
  proof (intro exI conjI)
    show 0 < inverse x by simp
    show inverse x ∉ inverse-set A
    proof −
      { fix y::rat
        assume ygt: inverse x < y
        have [simp]: 0 < y by (simp add: order-less-trans [OF - ygt])
        have iyless: inverse y < x
          by (simp add: inverse-less-imp-less [of x] ygt)
        have inverse y ∈ A
          by (simp add: preal-downwards-closed [OF A x] iyless)}
     thus ?thesis by (auto simp add: inverse-set-def)
    qed
  qed
qed

lemma inverse-set-not-rat-set:
   assumes A: A ∈ preal  shows inverse-set A < {r. 0 < r}
proof
  show inverse-set A ⊆ {r. 0 < r}  by (force simp add: inverse-set-def)
next
  show inverse-set A ≠ {r. 0 < r}
    by (insert preal-not-mem-inverse-set-Ex [OF A], blast)
```

qed

Part 3 of Dedekind sections definition

lemma *inverse-set-lemma3*:
  [|$A \in$ *preal*; $u \in$ *inverse-set* A; $0 < z$; $z < u$|]
  ==> $z \in$ *inverse-set* A
apply (*auto simp add*: *inverse-set-def*)
apply (*auto intro*: *order-less-trans*)
done

Part 4 of Dedekind sections definition

lemma *inverse-set-lemma4*:
  [|$A \in$ *preal*; $y \in$ *inverse-set* A|] ==> $\exists u \in$ *inverse-set* A. $y < u$
apply (*auto simp add*: *inverse-set-def*)
apply (*drule dense* [*of y*])
apply (*blast intro*: *order-less-trans*)
done

lemma *mem-inverse-set*:
  $A \in$ *preal* ==> *inverse-set* $A \in$ *preal*
apply (*simp* (*no-asm-simp*) *add*: *preal-def cut-def*)
apply (*blast intro*!: *inverse-set-not-empty inverse-set-not-rat-set*
    inverse-set-lemma3 inverse-set-lemma4)
done

5.7 Gleason's Lemma 9-3.4, page 122

lemma *Gleason9-34-exists*:
 assumes A: $A \in$ *preal*
  and $\forall x \in A.\ x + u \in A$
  and $0 \leq z$
 shows $\exists b \in A.\ b + (\textit{of-int}\ z) * u \in A$
proof (*cases z rule*: *int-cases*)
 case (*nonneg n*)
 show *?thesis*
 proof (*simp add*: *prems*, *induct n*)
  case *0*
   from *preal-nonempty* [*OF A*]
   show *?case* **by** *force*
  case (*Suc k*)
   from *this* **obtain** b **where** $b \in A$ $b + \textit{of-nat}\ k * u \in A$ **..**
   hence $b + \textit{of-int}\ (\textit{int}\ k) * u + u \in A$ **by** (*simp add*: *prems*)
   thus *?case* **by** (*force simp add*: *left-distrib add-ac prems*)
 qed
next
 case (*neg n*)
 with *prems* **show** *?thesis* **by** *simp*
qed

lemma *Gleason9-34-contra*:
assumes *A*: $A \in preal$
shows $[|\forall x{\in}A.\ x + u \in A;\ 0 < u;\ 0 < y;\ y \notin A|] ==> False$
proof (*induct u*, *induct y*)
fix *a*::*int* **and** *b*::*int*
fix *c*::*int* **and** *d*::*int*
assume *bpos* [*simp*]: $0 < b$
and *dpos* [*simp*]: $0 < d$
and *closed*: $\forall x{\in}A.\ x + (Fract\ c\ d) \in A$
and *upos*: $0 < Fract\ c\ d$
and *ypos*: $0 < Fract\ a\ b$
and *notin*: $Fract\ a\ b \notin A$
have *cpos* [*simp*]: $0 < c$
by (*simp add*: *zero-less-Fract-iff* [*OF dpos*, *symmetric*] *upos*)
have *apos* [*simp*]: $0 < a$
by (*simp add*: *zero-less-Fract-iff* [*OF bpos*, *symmetric*] *ypos*)
let *?k* = $a*d$
have *frle*: $Fract\ a\ b \le Fract\ ?k\ 1 * (Fract\ c\ d)$
proof −
have *?thesis* = $((a * d * b * d) \le c * b * (a * d * b * d))$
by (*simp add*: *mult-rat le-rat order-less-imp-not-eq2 mult-ac*)
moreover
have $(1 * (a * d * b * d)) \le c * b * (a * d * b * d)$
by (*rule mult-mono*,
simp-all add: *int-one-le-iff-zero-less zero-less-mult-iff*
order-less-imp-le)
ultimately
show *?thesis* **by** *simp*
qed
have *k*: $0 \le ?k$ **by** (*simp add*: *order-less-imp-le zero-less-mult-iff*)
from *Gleason9-34-exists* [*OF A closed k*]
obtain *z* **where** *z*: $z \in A$
and *mem*: $z + of\text{-}int\ ?k * Fract\ c\ d \in A$ **..**
have *less*: $z + of\text{-}int\ ?k * Fract\ c\ d < Fract\ a\ b$
by (*rule not-in-preal-ub* [*OF A notin mem ypos*])
have $0<z$ **by** (*rule preal-imp-pos* [*OF A z*])
with *frle* **and** *less* **show** *False* **by** (*simp add*: *Fract-of-int-eq*)
qed

lemma *Gleason9-34*:
assumes *A*: $A \in preal$
and *upos*: $0 < u$
shows $\exists r \in A.\ r + u \notin A$
proof (*rule ccontr*, *simp*)
assume *closed*: $\forall r{\in}A.\ r + u \in A$
from *preal-exists-bound* [*OF A*]
obtain *y* **where** *y*: $y \notin A$ **and** *ypos*: $0 < y$ **by** *blast*

```
  show False
    by (rule Gleason9-34-contra [OF A closed upos ypos y])
qed
```

5.8 Gleason's Lemma 9-3.6

```
lemma lemma-gleason9-36:
  assumes A: A ∈ preal
    and x: 1 < x
  shows ∃ r ∈ A. r*x ∉ A
proof −
  from preal-nonempty [OF A]
  obtain y where y: y ∈ A and  ypos: 0<y ..
  show ?thesis
  proof (rule classical)
    assume ~(∃ r∈A. r * x ∉ A)
    with y have ymem: y * x ∈ A by blast
    from ypos mult-strict-left-mono [OF x]
    have yless: y < y*x by simp
    let ?d = y*x − y
    from yless have dpos: 0 < ?d and eq: y + ?d = y*x by auto
    from Gleason9-34 [OF A dpos]
    obtain r where r: r∈A and notin: r + ?d ∉ A ..
    have rpos: 0<r by (rule preal-imp-pos [OF A r])
    with dpos have rdpos: 0 < r + ?d by arith
    have ~ (r + ?d ≤ y + ?d)
    proof
      assume le: r + ?d ≤ y + ?d
      from ymem have yd: y + ?d ∈ A by (simp add: eq)
      have r + ?d ∈ A by (rule preal-downwards-closed' [OF A yd rdpos le])
      with notin show False by simp
    qed
    hence y < r by simp
    with ypos have  dless: ?d < (r * ?d)/y
      by (simp add: pos-less-divide-eq mult-commute [of ?d]
                    mult-strict-right-mono dpos)
    have r + ?d < r*x
    proof −
      have r + ?d < r + (r * ?d)/y by (simp add: dless)
      also with ypos have ... = (r/y) * (y + ?d)
        by (simp only: right-distrib divide-inverse mult-ac, simp)
      also have ... = r*x using ypos
        by (simp add: times-divide-eq-left)
      finally show r + ?d < r*x .
    qed
    with r notin rdpos
    show ∃ r∈A. r * x ∉ A by (blast dest:  preal-downwards-closed [OF A])
  qed
qed
```

5.9 Existence of Inverse: Part 2

lemma *mem-Rep-preal-inverse-iff*:
(*z* ∈ *Rep-preal*(*inverse R*)) =
(*0* < *z* ∧ (∃ *y*. *z* < *y* ∧ *inverse y* ∉ *Rep-preal R*))
apply (*simp add*: *preal-inverse-def mem-inverse-set Rep-preal*)
apply (*simp add*: *inverse-set-def*)
done

lemma *Rep-preal-of-rat*:
0 < *q* ==> *Rep-preal* (*preal-of-rat q*) = {*x*. *0* < *x* ∧ *x* < *q*}
by (*simp add*: *preal-of-rat-def rat-mem-preal*)

lemma *subset-inverse-mult-lemma*:
assumes *xpos*: *0* < *x* **and** *xless*: *x* < *1*
shows ∃ *r u y*. *0* < *r* & *r* < *y* & *inverse y* ∉ *Rep-preal R* &
u ∈ *Rep-preal R* & *x* = *r* ∗ *u*
proof −
from *xpos* **and** *xless* **have** *1* < *inverse x* **by** (*simp add*: *one-less-inverse-iff*)
from *lemma-gleason9-36* [*OF Rep-preal this*]
obtain *r* **where** *r*: *r* ∈ *Rep-preal R*
and *notin*: *r* ∗ (*inverse x*) ∉ *Rep-preal R* **..**
have *rpos*: *0*<*r* **by** (*rule preal-imp-pos* [*OF Rep-preal r*])
from *preal-exists-greater* [*OF Rep-preal r*]
obtain *u* **where** *u*: *u* ∈ *Rep-preal R* **and** *rless*: *r* < *u* **..**
have *upos*: *0*<*u* **by** (*rule preal-imp-pos* [*OF Rep-preal u*])
show *?thesis*
proof (*intro exI conjI*)
show *0* < *x*/*u* **using** *xpos upos*
by (*simp add*: *zero-less-divide-iff*)
show *x*/*u* < *x*/*r* **using** *xpos upos rpos*
by (*simp add*: *divide-inverse mult-less-cancel-left rless*)
show *inverse* (*x* / *r*) ∉ *Rep-preal R* **using** *notin*
by (*simp add*: *divide-inverse mult-commute*)
show *u* ∈ *Rep-preal R* **by** (*rule u*)
show *x* = *x* / *u* ∗ *u* **using** *upos*
by (*simp add*: *divide-inverse mult-commute*)
qed
qed

lemma *subset-inverse-mult*:
Rep-preal(*preal-of-rat 1*) ⊆ *Rep-preal*(*inverse R* ∗ *R*)
apply (*auto simp add*: *Bex-def Rep-preal-of-rat mem-Rep-preal-inverse-iff*
mem-Rep-preal-mult-iff)
apply (*blast dest*: *subset-inverse-mult-lemma*)
done

lemma *inverse-mult-subset-lemma*:
assumes *rpos*: *0* < *r*
and *rless*: *r* < *y*

```
    and notin: inverse y ∉ Rep-preal R
    and q: q ∈ Rep-preal R
  shows r*q < 1
proof −
  have q < inverse y using rpos rless
    by (simp add: not-in-preal-ub [OF Rep-preal notin] q)
  hence r * q < r/y using rpos
    by (simp add: divide-inverse mult-less-cancel-left)
  also have ... ≤ 1 using rpos rless
    by (simp add: pos-divide-le-eq)
  finally show ?thesis .
qed

lemma inverse-mult-subset:
     Rep-preal(inverse R * R) ⊆ Rep-preal(preal-of-rat 1)
apply (auto simp add: Bex-def Rep-preal-of-rat mem-Rep-preal-inverse-iff
                  mem-Rep-preal-mult-iff)
apply (simp add: zero-less-mult-iff preal-imp-pos [OF Rep-preal])
apply (blast intro: inverse-mult-subset-lemma)
done

lemma preal-mult-inverse: inverse R * R = (1::preal)
unfolding preal-one-def
apply (rule Rep-preal-inject [THEN iffD1])
apply (rule equalityI [OF inverse-mult-subset subset-inverse-mult])
done

lemma preal-mult-inverse-right: R * inverse R = (1::preal)
apply (rule preal-mult-commute [THEN subst])
apply (rule preal-mult-inverse)
done
```

Theorems needing *Gleason9-34*

```
lemma Rep-preal-self-subset: Rep-preal (R) ⊆ Rep-preal(R + S)
proof
  fix r
  assume r: r ∈ Rep-preal R
  have rpos: 0<r by (rule preal-imp-pos [OF Rep-preal r])
  from mem-Rep-preal-Ex
  obtain y where y: y ∈ Rep-preal S ..
  have ypos: 0<y by (rule preal-imp-pos [OF Rep-preal y])
  have ry: r+y ∈ Rep-preal(R + S) using r y
    by (auto simp add: mem-Rep-preal-add-iff)
  show r ∈ Rep-preal(R + S) using r ypos rpos
    by (simp add:  preal-downwards-closed [OF Rep-preal ry])
qed

lemma Rep-preal-sum-not-subset: ~ Rep-preal (R + S) ⊆ Rep-preal(R)
proof −
```

from *mem-Rep-preal-Ex*
obtain *y* **where** *y*: $y \in$ *Rep-preal S* **..**
have *ypos*: $0<y$ **by** (*rule preal-imp-pos* [*OF Rep-preal y*])
from *Gleason9-34* [*OF Rep-preal ypos*]
obtain *r* **where** *r*: $r \in$ *Rep-preal R* **and** *notin*: $r + y \notin$ *Rep-preal R* **..**
have $r + y \in$ *Rep-preal* $(R + S)$ **using** *r y*
by (*auto simp add*: *mem-Rep-preal-add-iff*)
thus *?thesis* **using** *notin* **by** *blast*
qed

lemma *Rep-preal-sum-not-eq*: *Rep-preal* $(R + S) \neq$ *Rep-preal*(R)
by (*insert Rep-preal-sum-not-subset*, *blast*)

at last, Gleason prop. 9-3.5(iii) page 123

lemma *preal-self-less-add-left*: $(R$::*preal*$) < R + S$
apply (*unfold preal-less-def less-le*)
apply (*simp add*: *Rep-preal-self-subset Rep-preal-sum-not-eq* [*THEN not-sym*])
done

lemma *preal-self-less-add-right*: $(R$::*preal*$) < S + R$
by (*simp add*: *preal-add-commute preal-self-less-add-left*)

lemma *preal-not-eq-self*: $x \neq x +$ $(y$::*preal*$)$
by (*insert preal-self-less-add-left* [*of x y*], *auto*)

5.10 Subtraction for Positive Reals

Gleason prop. 9-3.5(iv), page 123: proving $A < B \Longrightarrow \exists\, D.\ A + D = B$. We define the claimed D and show that it is a positive real

Part 1 of Dedekind sections definition

lemma *diff-set-not-empty*:
$R < S$ ==> {} $\subset$ *diff-set* (*Rep-preal S*) (*Rep-preal R*)
apply (*auto simp add*: *preal-less-def diff-set-def elim*!: *equalityE*)
apply (*frule-tac x1* $= S$ **in** *Rep-preal* [*THEN preal-exists-greater*])
apply (*drule preal-imp-pos* [*OF Rep-preal*], *clarify*)
apply (*cut-tac a*=*x* **and** *b*=*u* **in** *add-eq-exists*, *force*)
done

Part 2 of Dedekind sections definition

lemma *diff-set-nonempty*:
$\exists\, q.\ 0 < q$ & $q \notin$ *diff-set* (*Rep-preal S*) (*Rep-preal R*)
apply (*cut-tac X* $= S$ **in** *Rep-preal-exists-bound*)
apply (*erule exE*)
apply (*rule-tac x* $= x$ **in** *exI*, *auto*)
apply (*simp add*: *diff-set-def*)
apply (*auto dest*: *Rep-preal* [*THEN preal-downwards-closed*])
done

lemma *diff-set-not-rat-set*:
diff-set (*Rep-preal S*) (*Rep-preal R*) < {*r*. *0* < *r*} (**is** *?lhs* < *?rhs*)
proof
show *?lhs* ⊆ *?rhs* **by** (*auto simp add*: *diff-set-def*)
show *?lhs* ≠ *?rhs* **using** *diff-set-nonempty* **by** *blast*
qed

Part 3 of Dedekind sections definition

lemma *diff-set-lemma3*:
[|*R* < *S*; *u* ∈ *diff-set* (*Rep-preal S*) (*Rep-preal R*); *0* < *z*; *z* < *u*|]
==> *z* ∈ *diff-set* (*Rep-preal S*) (*Rep-preal R*)
apply (*auto simp add*: *diff-set-def*)
apply (*rule-tac x*=*x* **in** *exI*)
apply (*drule Rep-preal* [*THEN preal-downwards-closed*], *auto*)
done

Part 4 of Dedekind sections definition

lemma *diff-set-lemma4*:
[|*R* < *S*; *y* ∈ *diff-set* (*Rep-preal S*) (*Rep-preal R*)|]
==> ∃ *u* ∈ *diff-set* (*Rep-preal S*) (*Rep-preal R*). *y* < *u*
apply (*auto simp add*: *diff-set-def*)
apply (*drule Rep-preal* [*THEN preal-exists-greater*], *clarify*)
apply (*cut-tac a*=*x*+*y* **and** *b*=*u* **in** *add-eq-exists*, *clarify*)
apply (*rule-tac x*=*y*+*xa* **in** *exI*)
apply (*auto simp add*: *add-ac*)
done

lemma *mem-diff-set*:
R < *S* ==> *diff-set* (*Rep-preal S*) (*Rep-preal R*) ∈ *preal*
apply (*unfold preal-def cut-def*)
apply (*blast intro*!: *diff-set-not-empty diff-set-not-rat-set*
diff-set-lemma3 diff-set-lemma4)
done

lemma *mem-Rep-preal-diff-iff*:
R < *S* ==>
(*z* ∈ *Rep-preal*(*S*−*R*)) =
(∃ *x*. *0* < *x* & *0* < *z* & *x* ∉ *Rep-preal R* & *x* + *z* ∈ *Rep-preal S*)
apply (*simp add*: *preal-diff-def mem-diff-set Rep-preal*)
apply (*force simp add*: *diff-set-def*)
done

proving that *R* + *D* ≤ *S*

lemma *less-add-left-lemma*:
assumes *Rless*: *R* < *S*
and *a*: *a* ∈ *Rep-preal R*
and *cb*: *c* + *b* ∈ *Rep-preal S*
and *c* ∉ *Rep-preal R*

and $0 < b$
and $0 < c$
shows $a + b \in$ *Rep-preal S*
proof −
have $0<a$ **by** (*rule preal-imp-pos* [*OF Rep-preal a*])
moreover
have $a < c$ **using** *prems*
by (*blast intro*: *not-in-Rep-preal-ub*)
ultimately show *?thesis* **using** *prems*
by (*simp add*: *preal-downwards-closed* [*OF Rep-preal cb*])
qed

lemma *less-add-left-le1*:
$R < (S::preal) ==> R + (S-R) \leq S$
apply (*auto simp add*: *Bex-def preal-le-def mem-Rep-preal-add-iff*
mem-Rep-preal-diff-iff)
apply (*blast intro*: *less-add-left-lemma*)
done

5.11 proving that $S \leq R + D$ — trickier

lemma *lemma-sum-mem-Rep-preal-ex*:
$x \in$ *Rep-preal S* $==> \exists e.\ 0 < e\ \&\ x + e \in$ *Rep-preal S*
apply (*drule Rep-preal* [*THEN preal-exists-greater*], *clarify*)
apply (*cut-tac a=x* **and** *b=u* **in** *add-eq-exists*, *auto*)
done

lemma *less-add-left-lemma2*:
assumes *Rless*: $R < S$
and *x*: $x \in$ *Rep-preal S*
and *xnot*: $x \notin$ *Rep-preal R*
shows $\exists u\ v\ z.\ 0 < v\ \&\ 0 < z\ \&\ u \in$ *Rep-preal R* $\&\ z \notin$ *Rep-preal R* $\&$
$z + v \in$ *Rep-preal S* $\&\ x = u + v$
proof −
have *xpos*: $0<x$ **by** (*rule preal-imp-pos* [*OF Rep-preal x*])
from *lemma-sum-mem-Rep-preal-ex* [*OF x*]
obtain *e* **where** *epos*: $0 < e$ **and** *xe*: $x + e \in$ *Rep-preal S* **by** *blast*
from *Gleason9-34* [*OF Rep-preal epos*]
obtain *r* **where** *r*: $r \in$ *Rep-preal R* **and** *notin*: $r + e \notin$ *Rep-preal R* **..**
with *x xnot xpos* **have** *rless*: $r < x$ **by** (*blast intro*: *not-in-Rep-preal-ub*)
from *add-eq-exists* [*of r x*]
obtain *y* **where** *eq*: $x = r+y$ **by** *auto*
show *?thesis*
proof (*intro exI conjI*)
show $r \in$ *Rep-preal R* **by** (*rule r*)
show $r + e \notin$ *Rep-preal R* **by** (*rule notin*)
show $r + e + y \in$ *Rep-preal S* **using** *xe eq* **by** (*simp add*: *add-ac*)
show $x = r + y$ **by** (*simp add*: *eq*)
show $0 < r + e$ **using** *epos preal-imp-pos* [*OF Rep-preal r*]

by *simp*
show $0 < y$ **using** *rless eq* **by** *arith*
qed
qed

lemma *less-add-left-le2*: $R < (S::preal) ==> S \leq R + (S-R)$
apply (*auto simp add*: *preal-le-def*)
apply (*case-tac* $x \in Rep\text{-}preal\ R$)
apply (*cut-tac Rep-preal-self-subset* [*of R*], *force*)
apply (*auto simp add*: *Bex-def mem-Rep-preal-add-iff mem-Rep-preal-diff-iff*)
apply (*blast dest*: *less-add-left-lemma2*)
done

lemma *less-add-left*: $R < (S::preal) ==> R + (S-R) = S$
by (*blast intro*: *preal-le-anti-sym* [*OF less-add-left-le1 less-add-left-le2*])

lemma *less-add-left-Ex*: $R < (S::preal) ==> \exists D.\ R + D = S$
by (*fast dest*: *less-add-left*)

lemma *preal-add-less2-mono1*: $R < (S::preal) ==> R + T < S + T$
apply (*auto dest*!: *less-add-left-Ex simp add*: *preal-add-assoc*)
apply (*rule-tac y1* $= D$ **in** *preal-add-commute* [*THEN subst*])
apply (*auto intro*: *preal-self-less-add-left simp add*: *preal-add-assoc* [*symmetric*])
done

lemma *preal-add-less2-mono2*: $R < (S::preal) ==> T + R < T + S$
by (*auto intro*: *preal-add-less2-mono1 simp add*: *preal-add-commute* [*of T*])

lemma *preal-add-right-less-cancel*: $R + T < S + T ==> R < (S::preal)$
apply (*insert linorder-less-linear* [*of R S*], *auto*)
apply (*drule-tac* $R = S$ **and** $T = T$ **in** *preal-add-less2-mono1*)
apply (*blast dest*: *order-less-trans*)
done

lemma *preal-add-left-less-cancel*: $T + R < T + S ==> R < (S::preal)$
by (*auto elim*: *preal-add-right-less-cancel simp add*: *preal-add-commute* [*of T*])

lemma *preal-add-less-cancel-right*: $((R::preal) + T < S + T) = (R < S)$
by (*blast intro*: *preal-add-less2-mono1 preal-add-right-less-cancel*)

lemma *preal-add-less-cancel-left*: $(T + (R::preal) < T + S) = (R < S)$
by (*blast intro*: *preal-add-less2-mono2 preal-add-left-less-cancel*)

lemma *preal-add-le-cancel-right*: $((R::preal) + T \leq S + T) = (R \leq S)$
by (*simp add*: *linorder-not-less* [*symmetric*] *preal-add-less-cancel-right*)

lemma *preal-add-le-cancel-left*: $(T + (R::preal) \leq T + S) = (R \leq S)$
by (*simp add*: *linorder-not-less* [*symmetric*] *preal-add-less-cancel-left*)

lemma *preal-add-less-mono*:
[| *x1* < *y1*; *x2* < *y2* |] ==> *x1* + *x2* < *y1* + (*y2*::*preal*)
apply (*auto dest*!: *less-add-left-Ex simp add*: *preal-add-ac*)
apply (*rule preal-add-assoc* [*THEN subst*])
apply (*rule preal-self-less-add-right*)
done

lemma *preal-add-right-cancel*: (*R*::*preal*) + *T* = *S* + *T* ==> *R* = *S*
apply (*insert linorder-less-linear* [*of R S*], *safe*)
apply (*drule-tac* [!] *T* = *T* **in** *preal-add-less2-mono1*, *auto*)
done

lemma *preal-add-left-cancel*: *C* + *A* = *C* + *B* ==> *A* = (*B*::*preal*)
by (*auto intro*: *preal-add-right-cancel simp add*: *preal-add-commute*)

lemma *preal-add-left-cancel-iff*: (*C* + *A* = *C* + *B*) = ((*A*::*preal*) = *B*)
by (*fast intro*: *preal-add-left-cancel*)

lemma *preal-add-right-cancel-iff*: (*A* + *C* = *B* + *C*) = ((*A*::*preal*) = *B*)
by (*fast intro*: *preal-add-right-cancel*)

lemmas *preal-cancels* =
preal-add-less-cancel-right preal-add-less-cancel-left
preal-add-le-cancel-right preal-add-le-cancel-left
preal-add-left-cancel-iff preal-add-right-cancel-iff

instance *preal* :: *ordered-cancel-ab-semigroup-add*
proof
fix *a b c* :: *preal*
show *a* + *b* = *a* + *c* ⟹ *b* = *c* **by** (*rule preal-add-left-cancel*)
show *a* ≤ *b* ⟹ *c* + *a* ≤ *c* + *b* **by** (*simp only*: *preal-add-le-cancel-left*)
qed

5.12 Completeness of type *preal*

Prove that supremum is a cut

Part 1 of Dedekind sections definition

lemma *preal-sup-set-not-empty*:
P ≠ {} ==> {} ⊂ (⋃ *X* ∈ *P*. *Rep-preal*(*X*))
apply *auto*
apply (*cut-tac X* = *x* **in** *mem-Rep-preal-Ex*, *auto*)
done

Part 2 of Dedekind sections definition

lemma *preal-sup-not-exists*:
∀ *X* ∈ *P*. *X* ≤ *Y* ==> ∃ *q*. *0* < *q* & *q* ∉ (⋃ *X* ∈ *P*. *Rep-preal*(*X*))
apply (*cut-tac X* = *Y* **in** *Rep-preal-exists-bound*)
apply (*auto simp add*: *preal-le-def*)

done

lemma *preal-sup-set-not-rat-set*:
$\forall X \in P.\ X \leq Y \Longrightarrow (\bigcup X \in P.\ \mathit{Rep\text{-}preal}(X)) < \{r.\ 0 < r\}$
apply (*drule preal-sup-not-exists*)
apply (*blast intro*: *preal-imp-pos* [*OF Rep-preal*])
done

Part 3 of Dedekind sections definition

lemma *preal-sup-set-lemma3*:
$[|P \neq \{\};\ \forall X \in P.\ X \leq Y;\ u \in (\bigcup X \in P.\ \mathit{Rep\text{-}preal}(X));\ 0 < z;\ z < u|]$
$\Longrightarrow z \in (\bigcup X \in P.\ \mathit{Rep\text{-}preal}(X))$
by (*auto elim*: *Rep-preal* [*THEN preal-downwards-closed*])

Part 4 of Dedekind sections definition

lemma *preal-sup-set-lemma4*:
$[|P \neq \{\};\ \forall X \in P.\ X \leq Y;\ y \in (\bigcup X \in P.\ \mathit{Rep\text{-}preal}(X))\ |]$
$\Longrightarrow \exists u \in (\bigcup X \in P.\ \mathit{Rep\text{-}preal}(X)).\ y < u$
by (*blast dest*: *Rep-preal* [*THEN preal-exists-greater*])

lemma *preal-sup*:
$[|P \neq \{\};\ \forall X \in P.\ X \leq Y|] \Longrightarrow (\bigcup X \in P.\ \mathit{Rep\text{-}preal}(X)) \in \mathit{preal}$
apply (*unfold preal-def cut-def*)
apply (*blast intro*!: *preal-sup-set-not-empty preal-sup-set-not-rat-set*
preal-sup-set-lemma3 preal-sup-set-lemma4)
done

lemma *preal-psup-le*:
$[|\ \forall X \in P.\ X \leq Y;\ \ x \in P\ |] \Longrightarrow x \leq \mathit{psup}\ P$
apply (*simp* (*no-asm-simp*) *add*: *preal-le-def*)
apply (*subgoal-tac* $P \neq \{\}$)
apply (*auto simp add*: *psup-def preal-sup*)
done

lemma *psup-le-ub*: $[|\ P \neq \{\};\ \forall X \in P.\ X \leq Y\ |] \Longrightarrow \mathit{psup}\ P \leq Y$
apply (*simp* (*no-asm-simp*) *add*: *preal-le-def*)
apply (*simp add*: *psup-def preal-sup*)
apply (*auto simp add*: *preal-le-def*)
done

Supremum property

lemma *preal-complete*:
$[|\ P \neq \{\};\ \forall X \in P.\ X \leq Y\ |] \Longrightarrow (\exists X \in P.\ Z < X) = (Z < \mathit{psup}\ P)$
apply (*simp add*: *preal-less-def psup-def preal-sup*)
apply (*auto simp add*: *preal-le-def*)
apply (*rename-tac U*)
apply (*cut-tac* $x = U$ **and** $y = Z$ **in** *linorder-less-linear*)
apply (*auto simp add*: *preal-less-def*)
done

5.13 The Embedding from *rat* into *preal*

lemma *preal-of-rat-add-lemma1*:
  $[|x < y + z;\ 0 < x;\ 0 < y|] ==> x * y * inverse\ (y + z) < (y::rat)$
apply (*frule-tac* $c = y * inverse\ (y + z)$ **in** *mult-strict-right-mono*)
apply (*simp add*: *zero-less-mult-iff*)
apply (*simp add*: *mult-ac*)
done

lemma *preal-of-rat-add-lemma2*:
  assumes $u < x + y$
    and $0 < x$
    and $0 < y$
    and $0 < u$
  shows $\exists v\ w::rat.\ w < y\ \&\ 0 < v\ \&\ v < x\ \&\ 0 < w\ \&\ u = v + w$
proof (*intro exI conjI*)
  show $u * x * inverse(x+y) < x$ **using** *prems*
    by (*simp add*: *preal-of-rat-add-lemma1*)
  show $u * y * inverse(x+y) < y$ **using** *prems*
    by (*simp add*: *preal-of-rat-add-lemma1 add-commute* [*of x*])
  show $0 < u * x * inverse\ (x + y)$ **using** *prems*
    by (*simp add*: *zero-less-mult-iff*)
  show $0 < u * y * inverse\ (x + y)$ **using** *prems*
    by (*simp add*: *zero-less-mult-iff*)
  show $u = u * x * inverse\ (x + y) + u * y * inverse\ (x + y)$ **using** *prems*
    by (*simp add*: *left-distrib* [*symmetric*] *right-distrib* [*symmetric*] *mult-ac*)
qed

lemma *preal-of-rat-add*:
  $[|\ 0 < x;\ 0 < y|]$
   $==>$ *preal-of-rat* $((x::rat) + y) =$ *preal-of-rat* $x +$ *preal-of-rat* y
apply (*unfold preal-of-rat-def preal-add-def*)
apply (*simp add*: *rat-mem-preal*)
apply (*rule-tac* $f = Abs\text{-}preal$ **in** *arg-cong*)
apply (*auto simp add*: *add-set-def*)
apply (*blast dest*: *preal-of-rat-add-lemma2*)
done

lemma *preal-of-rat-mult-lemma1*:
  $[|x < y;\ 0 < x;\ 0 < z|] ==> x * z * inverse\ y < (z::rat)$
apply (*frule-tac* $c = z * inverse\ y$ **in** *mult-strict-right-mono*)
apply (*simp add*: *zero-less-mult-iff*)
apply (*subgoal-tac* $y * (z * inverse\ y) = z * (y * inverse\ y)$)
apply (*simp-all add*: *mult-ac*)
done

lemma *preal-of-rat-mult-lemma2*:
  assumes *xless*: $x < y * z$
    and *xpos*: $0 < x$
    and *ypos*: $0 < y$

shows $x * z * inverse\ y * inverse\ z < (z::rat)$
proof −
have $0 < y * z$ **using** *prems* **by** *simp*
hence *zpos*: $0 < z$ **using** *prems* **by** (*simp add*: *zero-less-mult-iff*)
have $x * z * inverse\ y * inverse\ z = x * inverse\ y * (z * inverse\ z)$
by (*simp add*: *mult-ac*)
also have $... = x/y$ **using** *zpos*
by (*simp add*: *divide-inverse*)
also from *xless* **have** $... < z$
by (*simp add*: *pos-divide-less-eq* [*OF ypos*] *mult-commute*)
finally show *?thesis* **.**
qed

lemma *preal-of-rat-mult-lemma3*:
assumes *uless*: $u < x * y$
and $0 < x$
and $0 < y$
and $0 < u$
shows $\exists v\ w::rat.\ v < x\ \&\ w < y\ \&\ 0 < v\ \&\ 0 < w\ \&\ u = v * w$
proof −
from *dense* [*OF uless*]
obtain r **where** $u < r\ r < x * y$ **by** *blast*
thus *?thesis*
proof (*intro exI conjI*)
show $u * x * inverse\ r < x$ **using** *prems*
by (*simp add*: *preal-of-rat-mult-lemma1*)
show $r * y * inverse\ x * inverse\ y < y$ **using** *prems*
by (*simp add*: *preal-of-rat-mult-lemma2*)
show $0 < u * x * inverse\ r$ **using** *prems*
by (*simp add*: *zero-less-mult-iff*)
show $0 < r * y * inverse\ x * inverse\ y$ **using** *prems*
by (*simp add*: *zero-less-mult-iff*)
have $u * x * inverse\ r * (r * y * inverse\ x * inverse\ y) =$
$u * (r * inverse\ r) * (x * inverse\ x) * (y * inverse\ y)$
by (*simp only*: *mult-ac*)
thus $u = u * x * inverse\ r * (r * y * inverse\ x * inverse\ y)$ **using** *prems*
by *simp*
qed
qed

lemma *preal-of-rat-mult*:
$[|\ 0 < x;\ 0 < y|]$
$==>\ preal\text{-}of\text{-}rat\ ((x::rat) * y) = preal\text{-}of\text{-}rat\ x * preal\text{-}of\text{-}rat\ y$
apply (*unfold preal-of-rat-def preal-mult-def*)
apply (*simp add*: *rat-mem-preal*)
apply (*rule-tac f* = *Abs-preal* **in** *arg-cong*)
apply (*auto simp add*: *zero-less-mult-iff mult-strict-mono mult-set-def*)
apply (*blast dest*: *preal-of-rat-mult-lemma3*)
done

lemma *preal-of-rat-less-iff*:
[| $0 < x$; $0 < y$|] ==> (*preal-of-rat* $x <$ *preal-of-rat* $y) = (x < y)$
by (*force simp add*: *preal-of-rat-def preal-less-def rat-mem-preal*)

lemma *preal-of-rat-le-iff*:
[| $0 < x$; $0 < y$|] ==> (*preal-of-rat* $x \leq$ *preal-of-rat* $y) = (x \leq y)$
by (*simp add*: *preal-of-rat-less-iff linorder-not-less* [*symmetric*])

lemma *preal-of-rat-eq-iff*:
[| $0 < x$; $0 < y$|] ==> (*preal-of-rat* $x =$ *preal-of-rat* $y) = (x = y)$
by (*simp add*: *preal-of-rat-le-iff order-eq-iff*)

end

6 RealDef: Defining the Reals from the Positive Reals

theory *RealDef*
imports *PReal*
uses (*real-arith.ML*)
begin

definition
realrel :: ((*preal* * *preal*) * (*preal* * *preal*)) *set* **where**
realrel = {*p*. ∃ *x1 y1 x2 y2*. *p* = ((*x1*,*y1*),(*x2*,*y2*)) & *x1*+*y2* = *x2*+*y1*}

typedef (*Real*) *real* = *UNIV*//*realrel*
by (*auto simp add*: *quotient-def*)

definition

real-of-preal :: *preal* => *real* **where**
real-of-preal m = *Abs-Real*(*realrel*``{(*m* + *1*, *1*)})

instantiation *real* :: {*zero*, *one*, *plus*, *minus*, *uminus*, *times*, *inverse*, *ord*, *abs*, *sgn*}
begin

definition
real-zero-def [*code func del*]: *0* = *Abs-Real*(*realrel*``{(*1*, *1*)})

definition
real-one-def [*code func del*]: *1* = *Abs-Real*(*realrel*``{(*1* + *1*, *1*)})

definition
real-add-def [*code func del*]: *z* + *w* =

contents ($\bigcup(x,y) \in$ *Rep-Real*(z). $\bigcup(u,v) \in$ *Rep-Real*(w).
{ *Abs-Real*(*realrel*``{(x+u, y+v)}) })

definition
real-minus-def [*code func del*]: $-\ r =$ *contents* ($\bigcup(x,y) \in$ *Rep-Real*(r). { *Abs-Real*(*realrel*``{(y,x)}) })

definition
real-diff-def [*code func del*]: $r - (s$::*real*$) = r + - s$

definition
real-mult-def [*code func del*]:
$z * w =$
contents ($\bigcup(x,y) \in$ *Rep-Real*(z). $\bigcup(u,v) \in$ *Rep-Real*(w).
{ *Abs-Real*(*realrel*``{(x*u + y*v, x*v + y*u)}) })

definition
real-inverse-def [*code func del*]: *inverse* (R::*real*) = (*THE* S. ($R = 0$ & $S = 0$) | $S * R = 1$)

definition
real-divide-def [*code func del*]: R / (S::*real*) = $R *$ *inverse* S

definition
real-le-def [*code func del*]: $z \leq (w$::*real*) $\longleftrightarrow$
($\exists x\ y\ u\ v.\ x$+$v \leq u$+y & (x,y) $\in$ *Rep-Real* z & (u,v) $\in$ *Rep-Real* w)

definition
real-less-def [*code func del*]: $x < (y$::*real*) $\longleftrightarrow x \leq y \wedge x \neq y$

definition
real-abs-def: *abs* (r::*real*) = (*if* $r < 0$ *then* $- r$ *else* r)

definition
real-sgn-def: *sgn* (x::*real*) = (*if* x=0 *then* 0 *else if* 0<x *then* 1 *else* $- 1$)

instance ..

end

6.1 Equivalence relation over positive reals

lemma *preal-trans-lemma*:
assumes $x + y1 = x1 + y$
and $x + y2 = x2 + y$
shows $x1 + y2 = x2 + (y1$::*preal*)
proof −
have ($x1 + y2$) + x = ($x + y2$) + $x1$ **by** (*simp add*: *add-ac*)
also have ... = ($x2 + y$) + $x1$ **by** (*simp add*: *prems*)

also have ... = *x2* + (*x1* + *y*) **by** (*simp add*: *add-ac*)
also have ... = *x2* + (*x* + *y1*) **by** (*simp add*: *prems*)
also have ... = (*x2* + *y1*) + *x* **by** (*simp add*: *add-ac*)
finally have (*x1* + *y2*) + *x* = (*x2* + *y1*) + *x* **.**
thus *?thesis* **by** (*rule add-right-imp-eq*)
qed

lemma *realrel-iff* [*simp*]: (((*x1*,*y1*),(*x2*,*y2*)) ∈ *realrel*) = (*x1* + *y2* = *x2* + *y1*)
by (*simp add*: *realrel-def*)

lemma *equiv-realrel*: *equiv UNIV realrel*
apply (*auto simp add*: *equiv-def refl-def sym-def trans-def realrel-def*)
apply (*blast dest*: *preal-trans-lemma*)
done

Reduces equality of equivalence classes to the *realrel* relation: (*realrel* `` {*x*} = *realrel* `` {*y*}) = ((*x*, *y*) ∈ *realrel*)

lemmas *equiv-realrel-iff* =
eq-equiv-class-iff [*OF equiv-realrel UNIV-I UNIV-I*]

declare *equiv-realrel-iff* [*simp*]

lemma *realrel-in-real* [*simp*]: *realrel*``{(*x*,*y*)}: *Real*
by (*simp add*: *Real-def realrel-def quotient-def*, *blast*)

declare *Abs-Real-inject* [*simp*]
declare *Abs-Real-inverse* [*simp*]

Case analysis on the representation of a real number as an equivalence class of pairs of positive reals.

lemma *eq-Abs-Real* [*case-names Abs-Real*, *cases type*: *real*]:
(!!*x y*. *z* = *Abs-Real*(*realrel*``{(*x*,*y*)}) ==> *P*) ==> *P*
apply (*rule Rep-Real* [*of z*, *unfolded Real-def*, *THEN quotientE*])
apply (*drule arg-cong* [**where** *f*=*Abs-Real*])
apply (*auto simp add*: *Rep-Real-inverse*)
done

6.2 Addition and Subtraction

lemma *real-add-congruent2-lemma*:
[|*a* + *ba* = *aa* + *b*; *ab* + *bc* = *ac* + *bb*|]
==> *a* + *ab* + (*ba* + *bc*) = *aa* + *ac* + (*b* + (*bb*::*preal*))
apply (*simp add*: *add-assoc*)
apply (*rule add-left-commute* [*of ab*, *THEN ssubst*])
apply (*simp add*: *add-assoc* [*symmetric*])
apply (*simp add*: *add-ac*)
done

lemma *real-add*:
  Abs-Real (*realrel*``{(*x*,*y*)}) + *Abs-Real* (*realrel*``{(*u*,*v*)}) =
   Abs-Real (*realrel*``{(*x*+*u*, *y*+*v*)})
proof −
  have (λ*z* *w*. (λ(*x*,*y*). (λ(*u*,*v*). {*Abs-Real* (*realrel* `` {(*x*+*u*, *y*+*v*)})}) *w*) *z*)
      respects2 realrel
    by (*simp add*: *congruent2-def*, *blast intro*: *real-add-congruent2-lemma*)
  thus *?thesis*
    by (*simp add*: *real-add-def UN-UN-split-split-eq*
            UN-equiv-class2 [*OF equiv-realrel equiv-realrel*])
qed

lemma *real-minus*: − *Abs-Real*(*realrel*``{(*x*,*y*)}) = *Abs-Real*(*realrel* `` {(*y*,*x*)})
proof −
  have (λ(*x*,*y*). {*Abs-Real* (*realrel*``{(*y*,*x*)})}) *respects realrel*
    by (*simp add*: *congruent-def add-commute*)
  thus *?thesis*
    by (*simp add*: *real-minus-def UN-equiv-class* [*OF equiv-realrel*])
qed

instance *real* :: *ab-group-add*
proof
  fix *x y z* :: *real*
  show (*x* + *y*) + *z* = *x* + (*y* + *z*)
    by (*cases x*, *cases y*, *cases z*, *simp add*: *real-add add-assoc*)
  show *x* + *y* = *y* + *x*
    by (*cases x*, *cases y*, *simp add*: *real-add add-commute*)
  show *0* + *x* = *x*
    by (*cases x*, *simp add*: *real-add real-zero-def add-ac*)
  show − *x* + *x* = *0*
    by (*cases x*, *simp add*: *real-minus real-add real-zero-def add-commute*)
  show *x* − *y* = *x* + − *y*
    by (*simp add*: *real-diff-def*)
qed

6.3 Multiplication

lemma *real-mult-congruent2-lemma*:
    !!(*x1*::*preal*). [| *x1* + *y2* = *x2* + *y1* |] ==>
        x ∗ *x1* + *y* ∗ *y1* + (*x* ∗ *y2* + *y* ∗ *x2*) =
        x ∗ *x2* + *y* ∗ *y2* + (*x* ∗ *y1* + *y* ∗ *x1*)
apply (*simp add*: *add-left-commute add-assoc* [*symmetric*])
apply (*simp add*: *add-assoc right-distrib* [*symmetric*])
apply (*simp add*: *add-commute*)
done

lemma *real-mult-congruent2*:
  (%*p1 p2*.

```
      (%(x1,y1). (%(x2,y2).
        { Abs-Real (realrel``{(x1*x2 + y1*y2, x1*y2+y1*x2)}) }) p2) p1)
    respects2 realrel
apply (rule congruent2-commuteI [OF equiv-realrel], clarify)
apply (simp add: mult-commute add-commute)
apply (auto simp add: real-mult-congruent2-lemma)
done

lemma real-mult:
     Abs-Real((realrel``{(x1,y1)})) * Abs-Real((realrel``{(x2,y2)})) =
      Abs-Real(realrel `` {(x1*x2+y1*y2,x1*y2+y1*x2)})
by (simp add: real-mult-def UN-UN-split-split-eq
       UN-equiv-class2 [OF equiv-realrel equiv-realrel real-mult-congruent2])

lemma real-mult-commute: (z::real) * w = w * z
by (cases z, cases w, simp add: real-mult add-ac mult-ac)

lemma real-mult-assoc: ((z1::real) * z2) * z3 = z1 * (z2 * z3)
apply (cases z1, cases z2, cases z3)
apply (simp add: real-mult right-distrib add-ac mult-ac)
done

lemma real-mult-1: (1::real) * z = z
apply (cases z)
apply (simp add: real-mult real-one-def right-distrib
                 mult-1-right mult-ac add-ac)
done

lemma real-add-mult-distrib: ((z1::real) + z2) * w = (z1 * w) + (z2 * w)
apply (cases z1, cases z2, cases w)
apply (simp add: real-add real-mult right-distrib add-ac mult-ac)
done
```

one and zero are distinct

```
lemma real-zero-not-eq-one: 0 ≠ (1::real)
proof -
  have (1::preal) < 1 + 1
    by (simp add: preal-self-less-add-left)
  thus ?thesis
    by (simp add: real-zero-def real-one-def)
qed

instance real :: comm-ring-1
proof
  fix x y z :: real
  show (x * y) * z = x * (y * z) by (rule real-mult-assoc)
  show x * y = y * x by (rule real-mult-commute)
  show 1 * x = x by (rule real-mult-1)
  show (x + y) * z = x * z + y * z by (rule real-add-mult-distrib)
```

show $0 \neq (1::real)$ **by** (*rule real-zero-not-eq-one*)
qed

6.4 Inverse and Division

lemma *real-zero-iff*: *Abs-Real* (*realrel* `` {(*x*, *x*)}) = *0*
by (*simp add*: *real-zero-def add-commute*)

Instead of using an existential quantifier and constructing the inverse within the proof, we could define the inverse explicitly.

lemma *real-mult-inverse-left-ex*: $x \neq 0 \Longrightarrow \exists y.\ y*x = (1::real)$
apply (*simp add*: *real-zero-def real-one-def*, *cases x*)
apply (*cut-tac x* = *xa* **and** *y* = *y* **in** *linorder-less-linear*)
apply (*auto dest*!: *less-add-left-Ex simp add*: *real-zero-iff*)
apply (*rule-tac*
x = *Abs-Real* (*realrel*``{(*1*, *inverse* (*D*) + *1*)})
in *exI*)
apply (*rule-tac* [*2*]
x = *Abs-Real* (*realrel*``{(*inverse* (*D*) + *1*, *1*)})
in *exI*)
apply (*auto simp add*: *real-mult preal-mult-inverse-right ring-simps*)
done

lemma *real-mult-inverse-left*: $x \neq 0 \Longrightarrow inverse(x)*x = (1::real)$
apply (*simp add*: *real-inverse-def*)
apply (*drule real-mult-inverse-left-ex*, *safe*)
apply (*rule theI*, *assumption*, *rename-tac z*)
apply (*subgoal-tac* (*z* * *x*) * *y* = *z* * (*x* * *y*))
apply (*simp add*: *mult-commute*)
apply (*rule mult-assoc*)
done

6.5 The Real Numbers form a Field

instance *real* :: *field*
proof
fix *x y z* :: *real*
show $x \neq 0 \Longrightarrow inverse\ x * x = 1$ **by** (*rule real-mult-inverse-left*)
show *x* / *y* = *x* * *inverse y* **by** (*simp add*: *real-divide-def*)
qed

Inverse of zero! Useful to simplify certain equations

lemma *INVERSE-ZERO*: *inverse 0* = (*0*::*real*)
by (*simp add*: *real-inverse-def*)

instance *real* :: *division-by-zero*
proof
show *inverse 0* = (*0*::*real*) **by** (*rule INVERSE-ZERO*)
qed

6.6 The $\leq$ Ordering

lemma *real-le-refl*: $w \leq (w::real)$
by (*cases w*, *force simp add*: *real-le-def*)

The arithmetic decision procedure is not set up for type preal. This lemma is currently unused, but it could simplify the proofs of the following two lemmas.

lemma *preal-eq-le-imp-le*:
  assumes *eq*: $a+b = c+d$ **and** *le*: $c \leq a$
  shows $b \leq (d::preal)$
proof −
  have $c+d \leq a+d$ **by** (*simp add*: *prems*)
  hence $a+b \leq a+d$ **by** (*simp add*: *prems*)
  thus $b \leq d$ **by** *simp*
qed

lemma *real-le-lemma*:
  assumes *l*: $u1 + v2 \leq u2 + v1$
      and $x1 + v1 = u1 + y1$
      and $x2 + v2 = u2 + y2$
  shows $x1 + y2 \leq x2 + (y1::preal)$
proof −
  have $(x1+v1) + (u2+y2) = (u1+y1) + (x2+v2)$ **by** (*simp add*: *prems*)
  hence $(x1+y2) + (u2+v1) = (x2+y1) + (u1+v2)$ **by** (*simp add*: *add-ac*)
  also have ... $\leq (x2+y1) + (u2+v1)$ **by** (*simp add*: *prems*)
  finally show *?thesis* **by** *simp*
qed

lemma *real-le*:
     $(Abs\text{-}Real(realrel``\{(x1,y1)\}) \leq Abs\text{-}Real(realrel``\{(x2,y2)\})) =$
      $(x1 + y2 \leq x2 + y1)$
apply (*simp add*: *real-le-def*)
apply (*auto intro*: *real-le-lemma*)
done

lemma *real-le-anti-sym*: $[|\ z \leq w;\ w \leq z\ |] ==> z = (w::real)$
by (*cases z*, *cases w*, *simp add*: *real-le*)

lemma *real-trans-lemma*:
  assumes $x + v \leq u + y$
      and $u + v' \leq u' + v$
      and $x2 + v2 = u2 + y2$
  shows $x + v' \leq u' + (y::preal)$
proof −
  have $(x+v') + (u+v) = (x+v) + (u+v')$ **by** (*simp add*: *add-ac*)
  also have ... $\leq (u+y) + (u+v')$ **by** (*simp add*: *prems*)
  also have ... $\leq (u+y) + (u'+v)$ **by** (*simp add*: *prems*)
  also have ... $= (u'+y) + (u+v)$ **by** (*simp add*: *add-ac*)

finally show *?thesis* **by** *simp*
qed

lemma *real-le-trans*: [| $i \leq j$; $j \leq k$ |] ==> $i \leq (k::real)$
apply (*cases i*, *cases j*, *cases k*)
apply (*simp add*: *real-le*)
apply (*blast intro*: *real-trans-lemma*)
done

lemma *real-less-le*: $((w::real) < z) = (w \leq z \ \& \ w \neq z)$
by (*simp add*: *real-less-def*)

instance *real* :: *order*
proof qed
(*assumption* |
rule real-le-refl real-le-trans real-le-anti-sym real-less-le)+

lemma *real-le-linear*: $(z::real) \leq w \mid w \leq z$
apply (*cases z*, *cases w*)
apply (*auto simp add*: *real-le real-zero-def add-ac*)
done

instance *real* :: *linorder*
by (*intro-classes*, *rule real-le-linear*)

lemma *real-le-eq-diff*: $(x \leq y) = (x-y \leq (0::real))$
apply (*cases x*, *cases y*)
apply (*auto simp add*: *real-le real-zero-def real-diff-def real-add real-minus*
add-ac)
apply (*simp-all add*: *add-assoc* [*symmetric*])
done

lemma *real-add-left-mono*:
assumes *le*: $x \leq y$ **shows** $z + x \leq z + (y::real)$
proof −
have $z + x - (z + y) = (z + -z) + (x - y)$
by (*simp add*: *diff-minus add-ac*)
with *le* **show** *?thesis*
by (*simp add*: *real-le-eq-diff* [*of x*] *real-le-eq-diff* [*of z+x*] *diff-minus*)
qed

lemma *real-sum-gt-zero-less*: $(0 < S + (-W::real)) ==> (W < S)$
by (*simp add*: *linorder-not-le* [*symmetric*] *real-le-eq-diff* [*of S*] *diff-minus*)

lemma *real-less-sum-gt-zero*: $(W < S) ==> (0 < S + (-W::real))$

by (*simp add*: *linorder-not-le* [*symmetric*] *real-le-eq-diff* [*of S*] *diff-minus*)

lemma *real-mult-order*: [| *0* < *x*; *0* < *y* |] ==> (*0*::*real*) < *x* * *y*
apply (*cases x*, *cases y*)
apply (*simp add*: *linorder-not-le* [**where** *'a* = *real*, *symmetric*]
linorder-not-le [**where** *'a* = *preal*]
real-zero-def real-le real-mult)
— Reduce to the (simpler) ≤ relation
apply (*auto dest*!: *less-add-left-Ex*
simp add: *add-ac mult-ac*
right-distrib preal-self-less-add-left)
done

lemma *real-mult-less-mono2*: [| (*0*::*real*) < *z*; *x* < *y* |] ==> *z* * *x* < *z* * *y*
apply (*rule real-sum-gt-zero-less*)
apply (*drule real-less-sum-gt-zero* [*of x y*])
apply (*drule real-mult-order*, *assumption*)
apply (*simp add*: *right-distrib*)
done

instantiation *real* :: *distrib-lattice*
begin

definition
(*inf* :: *real* ⇒ *real* ⇒ *real*) = *min*

definition
(*sup* :: *real* ⇒ *real* ⇒ *real*) = *max*

instance
by *default* (*auto simp add*: *inf-real-def sup-real-def min-max.sup-inf-distrib1*)

end

6.7 The Reals Form an Ordered Field

instance *real* :: *ordered-field*
proof
fix *x y z* :: *real*
show *x* ≤ *y* ==> *z* + *x* ≤ *z* + *y* **by** (*rule real-add-left-mono*)
show *x* < *y* ==> *0* < *z* ==> *z* * *x* < *z* * *y* **by** (*rule real-mult-less-mono2*)
show |*x*| = (*if x* < *0 then* −*x else x*) **by** (*simp only*: *real-abs-def*)
show *sgn x* = (*if x=0 then 0 else if 0<x then 1 else* − *1*)
by (*simp only*: *real-sgn-def*)
qed

instance *real* :: *lordered-ab-group-add* **..**

The function *real-of-preal* requires many proofs, but it seems to be essential for proving completeness of the reals from that of the positive reals.

lemma *real-of-preal-add*:
real-of-preal ((*x*::*preal*) + *y*) = *real-of-preal x* + *real-of-preal y*
by (*simp add*: *real-of-preal-def real-add left-distrib add-ac*)

lemma *real-of-preal-mult*:
real-of-preal ((*x*::*preal*) ∗ *y*) = *real-of-preal x*∗ *real-of-preal y*
by (*simp add*: *real-of-preal-def real-mult right-distrib add-ac mult-ac*)

Gleason prop 9-4.4 p 127

lemma *real-of-preal-trichotomy*:
∃ *m*. (*x*::*real*) = *real-of-preal m* | *x* = *0* | *x* = −(*real-of-preal m*)
apply (*simp add*: *real-of-preal-def real-zero-def*, *cases x*)
apply (*auto simp add*: *real-minus add-ac*)
apply (*cut-tac x* = *x* **and** *y* = *y* **in** *linorder-less-linear*)
apply (*auto dest*!: *less-add-left-Ex simp add*: *add-assoc* [*symmetric*])
done

lemma *real-of-preal-leD*:
real-of-preal m1 ≤ *real-of-preal m2* ==> *m1* ≤ *m2*
by (*simp add*: *real-of-preal-def real-le*)

lemma *real-of-preal-lessI*: *m1* < *m2* ==> *real-of-preal m1* < *real-of-preal m2*
by (*auto simp add*: *real-of-preal-leD linorder-not-le* [*symmetric*])

lemma *real-of-preal-lessD*:
real-of-preal m1 < *real-of-preal m2* ==> *m1* < *m2*
by (*simp add*: *real-of-preal-def real-le linorder-not-le* [*symmetric*])

lemma *real-of-preal-less-iff* [*simp*]:
(*real-of-preal m1* < *real-of-preal m2*) = (*m1* < *m2*)
by (*blast intro*: *real-of-preal-lessI real-of-preal-lessD*)

lemma *real-of-preal-le-iff*:
(*real-of-preal m1* ≤ *real-of-preal m2*) = (*m1* ≤ *m2*)
by (*simp add*: *linorder-not-less* [*symmetric*])

lemma *real-of-preal-zero-less*: *0* < *real-of-preal m*
apply (*insert preal-self-less-add-left* [*of 1 m*])
apply (*auto simp add*: *real-zero-def real-of-preal-def*
real-less-def real-le-def add-ac)
apply (*rule-tac x*=*m* + *1* **in** *exI*, *rule-tac x*=*1* **in** *exI*)
apply (*simp add*: *add-ac*)
done

lemma *real-of-preal-minus-less-zero*: − *real-of-preal m* < *0*
by (*simp add*: *real-of-preal-zero-less*)

lemma *real-of-preal-not-minus-gt-zero*: ~ *0* < − *real-of-preal m*
proof −

from *real-of-preal-minus-less-zero*
show *?thesis* **by** (*blast dest*: *order-less-trans*)
qed

6.8 Theorems About the Ordering

lemma *real-gt-zero-preal-Ex*: $(0 < x) = (\exists y.\ x = real\text{-}of\text{-}preal\ y)$
apply (*auto simp add*: *real-of-preal-zero-less*)
apply (*cut-tac* $x = x$ **in** *real-of-preal-trichotomy*)
apply (*blast elim*!: *real-of-preal-not-minus-gt-zero* [*THEN notE*])
done

lemma *real-gt-preal-preal-Ex*:
$real\text{-}of\text{-}preal\ z < x \Longrightarrow \exists y.\ x = real\text{-}of\text{-}preal\ y$
by (*blast dest*!: *real-of-preal-zero-less* [*THEN order-less-trans*]
intro: *real-gt-zero-preal-Ex* [*THEN iffD1*])

lemma *real-ge-preal-preal-Ex*:
$real\text{-}of\text{-}preal\ z \leq x \Longrightarrow \exists y.\ x = real\text{-}of\text{-}preal\ y$
by (*blast dest*: *order-le-imp-less-or-eq real-gt-preal-preal-Ex*)

lemma *real-less-all-preal*: $y \leq 0 \Longrightarrow \forall x.\ y < real\text{-}of\text{-}preal\ x$
by (*auto elim*: *order-le-imp-less-or-eq* [*THEN disjE*]
intro: *real-of-preal-zero-less* [*THEN* [*2*] *order-less-trans*]
simp add: *real-of-preal-zero-less*)

lemma *real-less-all-real2*: $\sim\ 0 < y \Longrightarrow \forall x.\ y < real\text{-}of\text{-}preal\ x$
by (*blast intro*!: *real-less-all-preal linorder-not-less* [*THEN iffD1*])

6.9 More Lemmas

lemma *real-mult-left-cancel*: $(c::real) \neq 0 \Longrightarrow (c*a=c*b) = (a=b)$
by *auto*

lemma *real-mult-right-cancel*: $(c::real) \neq 0 \Longrightarrow (a*c=b*c) = (a=b)$
by *auto*

lemma *real-mult-less-iff1* [*simp*]: $(0::real) < z \Longrightarrow (x*z < y*z) = (x < y)$
by (*force elim*: *order-less-asym*
simp add: *Ring-and-Field.mult-less-cancel-right*)

lemma *real-mult-le-cancel-iff1* [*simp*]: $(0::real) < z \Longrightarrow (x*z \leq y*z) = (x \leq y)$
apply (*simp add*: *mult-le-cancel-right*)
apply (*blast intro*: *elim*: *order-less-asym*)
done

lemma *real-mult-le-cancel-iff2* [*simp*]: $(0::real) < z \Longrightarrow (z*x \leq z*y) = (x \leq y)$
by(*simp add*:*mult-commute*)

lemma *real-inverse-gt-one*: $[|\ (0::real) < x;\ x < 1\ |] \Longrightarrow 1 < inverse\ x$

by (*simp add*: *one-less-inverse-iff*)

6.10 Embedding numbers into the Reals

abbreviation
real-of-nat :: *nat* ⇒ *real*
where
real-of-nat ≡ *of-nat*

abbreviation
real-of-int :: *int* ⇒ *real*
where
real-of-int ≡ *of-int*

abbreviation
real-of-rat :: *rat* ⇒ *real*
where
real-of-rat ≡ *of-rat*

consts

real :: *'a* => *real*

defs (**overloaded**)
real-of-nat-def [*code inline*]: *real* == *real-of-nat*
real-of-int-def [*code inline*]: *real* == *real-of-int*

lemma *real-eq-of-nat*: *real* = *of-nat*
unfolding *real-of-nat-def* **..**

lemma *real-eq-of-int*: *real* = *of-int*
unfolding *real-of-int-def* **..**

lemma *real-of-int-zero* [*simp*]: *real* (*0*::*int*) = *0*
by (*simp add*: *real-of-int-def*)

lemma *real-of-one* [*simp*]: *real* (*1*::*int*) = (*1*::*real*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-add* [*simp*]: *real*(*x* + *y*) = *real* (*x*::*int*) + *real y*
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-minus* [*simp*]: *real*(−*x*) = −*real* (*x*::*int*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-diff* [*simp*]: *real*(*x* − *y*) = *real* (*x*::*int*) − *real y*
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-mult* [*simp*]: *real*(*x* ∗ *y*) = *real* (*x*::*int*) ∗ *real y*

by (*simp add*: *real-of-int-def*)

lemma *real-of-int-setsum* [*simp*]: *real* ((*SUM x*:*A*. *f x*)::*int*) = (*SUM x*:*A*. *real*(*f x*))
  apply (*subst real-eq-of-int*)+
  apply (*rule of-int-setsum*)
done

lemma *real-of-int-setprod* [*simp*]: *real* ((*PROD x*:*A*. *f x*)::*int*) =
    (*PROD x*:*A*. *real*(*f x*))
  apply (*subst real-eq-of-int*)+
  apply (*rule of-int-setprod*)
done

lemma *real-of-int-zero-cancel* [*simp*]: (*real x* = *0*) = (*x* = (*0*::*int*))
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-inject* [*iff*]: (*real* (*x*::*int*) = *real y*) = (*x* = *y*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-less-iff* [*iff*]: (*real* (*x*::*int*) < *real y*) = (*x* < *y*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-le-iff* [*simp*]: (*real* (*x*::*int*) ≤ *real y*) = (*x* ≤ *y*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-gt-zero-cancel-iff* [*simp*]: (*0* < *real* (*n*::*int*)) = (*0* < *n*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-ge-zero-cancel-iff* [*simp*]: (*0* <= *real* (*n*::*int*)) = (*0* <= *n*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-lt-zero-cancel-iff* [*simp*]: (*real* (*n*::*int*) < *0*) = (*n* < *0*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-le-zero-cancel-iff* [*simp*]: (*real* (*n*::*int*) <= *0*) = (*n* <= *0*)
by (*simp add*: *real-of-int-def*)

lemma *real-of-int-abs* [*simp*]: *real* (*abs x*) = *abs*(*real* (*x*::*int*))
by (*auto simp add*: *abs-if*)

lemma *int-less-real-le*: ((*n*::*int*) < *m*) = (*real n* + *1* <= *real m*)
  apply (*subgoal-tac real n* + *1* = *real* (*n* + *1*))
  apply (*simp del*: *real-of-int-add*)
  apply *auto*
done

lemma *int-le-real-less*: ((*n*::*int*) <= *m*) = (*real n* < *real m* + *1*)
  apply (*subgoal-tac real m* + *1* = *real* (*m* + *1*))

```
  apply (simp del: real-of-int-add)
  apply simp
done

lemma real-of-int-div-aux: d ~= 0 ==> (real (x::int)) / (real d) =
    real (x div d) + (real (x mod d)) / (real d)
proof -
  assume d ~= 0
  have x = (x div d) * d + x mod d
    by auto
  then have real x = real (x div d) * real d + real(x mod d)
    by (simp only: real-of-int-mult [THEN sym] real-of-int-add [THEN sym])
  then have real x / real d = ... / real d
    by simp
  then show ?thesis
    by (auto simp add: add-divide-distrib ring-simps prems)
qed

lemma real-of-int-div: (d::int) ~= 0 ==> d dvd n ==>
    real(n div d) = real n / real d
  apply (frule real-of-int-div-aux [of d n])
  apply simp
  apply (simp add: zdvd-iff-zmod-eq-0)
done

lemma real-of-int-div2:
  0 <= real (n::int) / real (x) - real (n div x)
  apply (case-tac x = 0)
  apply simp
  apply (case-tac 0 < x)
  apply (simp add: compare-rls)
  apply (subst real-of-int-div-aux)
  apply simp
  apply simp
  apply (subst zero-le-divide-iff)
  apply auto
  apply (simp add: compare-rls)
  apply (subst real-of-int-div-aux)
  apply simp
  apply simp
  apply (subst zero-le-divide-iff)
  apply auto
done

lemma real-of-int-div3:
  real (n::int) / real (x) - real (n div x) <= 1
  apply(case-tac x = 0)
  apply simp
  apply (simp add: compare-rls)
```

apply (*subst real-of-int-div-aux*)
apply *assumption*
apply *simp*
apply (*subst divide-le-eq*)
apply *clarsimp*
apply (*rule conjI*)
apply (*rule impI*)
apply (*rule order-less-imp-le*)
apply *simp*
apply (*rule impI*)
apply (*rule order-less-imp-le*)
apply *simp*
done

lemma *real-of-int-div4*: *real* (*n div x*) <= *real* (*n*::*int*) / *real x*
by (*insert real-of-int-div2* [*of n x*], *simp*)

6.11 Embedding the Naturals into the Reals

lemma *real-of-nat-zero* [*simp*]: *real* (*0*::*nat*) = *0*
by (*simp add*: *real-of-nat-def*)

lemma *real-of-nat-one* [*simp*]: *real* (*Suc 0*) = (*1*::*real*)
by (*simp add*: *real-of-nat-def*)

lemma *real-of-nat-add* [*simp*]: *real* (*m* + *n*) = *real* (*m*::*nat*) + *real n*
by (*simp add*: *real-of-nat-def*)

lemma *real-of-nat-Suc*: *real* (*Suc n*) = *real n* + (*1*::*real*)
by (*simp add*: *real-of-nat-def*)

lemma *real-of-nat-less-iff* [*iff*]:
(*real* (*n*::*nat*) < *real m*) = (*n* < *m*)
by (*simp add*: *real-of-nat-def*)

lemma *real-of-nat-le-iff* [*iff*]: (*real* (*n*::*nat*) ≤ *real m*) = (*n* ≤ *m*)
by (*simp add*: *real-of-nat-def*)

lemma *real-of-nat-ge-zero* [*iff*]: *0* ≤ *real* (*n*::*nat*)
by (*simp add*: *real-of-nat-def zero-le-imp-of-nat*)

lemma *real-of-nat-Suc-gt-zero*: *0* < *real* (*Suc n*)
by (*simp add*: *real-of-nat-def del*: *of-nat-Suc*)

lemma *real-of-nat-mult* [*simp*]: *real* (*m* * *n*) = *real* (*m*::*nat*) * *real n*
by (*simp add*: *real-of-nat-def of-nat-mult*)

lemma *real-of-nat-setsum* [*simp*]: *real* ((*SUM x*:*A*. *f x*)::*nat*) =

```
    (SUM x:A. real(f x))
  apply (subst real-eq-of-nat)+
  apply (rule of-nat-setsum)
done

lemma real-of-nat-setprod [simp]: real ((PROD x:A. f x)::nat) =
    (PROD x:A. real(f x))
  apply (subst real-eq-of-nat)+
  apply (rule of-nat-setprod)
done

lemma real-of-card: real (card A) = setsum (%x.1) A
  apply (subst card-eq-setsum)
  apply (subst real-of-nat-setsum)
  apply simp
done

lemma real-of-nat-inject [iff]: (real (n::nat) = real m) = (n = m)
by (simp add: real-of-nat-def)

lemma real-of-nat-zero-iff [iff]: (real (n::nat) = 0) = (n = 0)
by (simp add: real-of-nat-def)

lemma real-of-nat-diff: n ≤ m ==> real (m - n) = real (m::nat) - real n
by (simp add: add: real-of-nat-def of-nat-diff)

lemma real-of-nat-gt-zero-cancel-iff [simp]: (0 < real (n::nat)) = (0 < n)
by (auto simp: real-of-nat-def)

lemma real-of-nat-le-zero-cancel-iff [simp]: (real (n::nat) ≤ 0) = (n = 0)
by (simp add: add: real-of-nat-def)

lemma not-real-of-nat-less-zero [simp]: ~ real (n::nat) < 0
by (simp add: add: real-of-nat-def)

lemma real-of-nat-ge-zero-cancel-iff [simp]: (0 ≤ real (n::nat))
by (simp add: add: real-of-nat-def)

lemma nat-less-real-le: ((n::nat) < m) = (real n + 1 <= real m)
  apply (subgoal-tac real n + 1 = real (Suc n))
  apply simp
  apply (auto simp add: real-of-nat-Suc)
done

lemma nat-le-real-less: ((n::nat) <= m) = (real n < real m + 1)
  apply (subgoal-tac real m + 1 = real (Suc m))
  apply (simp add: less-Suc-eq-le)
  apply (simp add: real-of-nat-Suc)
done
```

lemma *real-of-nat-div-aux*: *0 < d ==> (real (x::nat)) / (real d) =*
real (x div d) + (real (x mod d)) / (real d)
proof −
assume *0 < d*
have *x = (x div d) * d + x mod d*
by *auto*
then have *real x = real (x div d) * real d + real(x mod d)*
by (*simp only*: *real-of-nat-mult* [*THEN sym*] *real-of-nat-add* [*THEN sym*])
then have *real x / real d = ... / real d*
by *simp*
then show *?thesis*
by (*auto simp add*: *add-divide-distrib ring-simps prems*)
qed

lemma *real-of-nat-div*: *0 < (d::nat) ==> d dvd n ==>*
real(n div d) = real n / real d
apply (*frule real-of-nat-div-aux* [*of d n*])
apply *simp*
apply (*subst dvd-eq-mod-eq-0* [*THEN sym*])
apply *assumption*
done

lemma *real-of-nat-div2*:
0 <= real (n::nat) / real (x) − real (n div x)
apply(*case-tac x = 0*)
apply (*simp*)
apply (*simp add*: *compare-rls*)
apply (*subst real-of-nat-div-aux*)
apply *simp*
apply *simp*
apply (*subst zero-le-divide-iff*)
apply *simp*
done

lemma *real-of-nat-div3*:
real (n::nat) / real (x) − real (n div x) <= 1
apply(*case-tac x = 0*)
apply (*simp*)
apply (*simp add*: *compare-rls*)
apply (*subst real-of-nat-div-aux*)
apply *simp*
apply *simp*
done

lemma *real-of-nat-div4*: *real (n div x) <= real (n::nat) / real x*
by (*insert real-of-nat-div2* [*of n x*], *simp*)

lemma *real-of-int-real-of-nat*: *real (int n) = real n*

by (*simp add*: *real-of-nat-def real-of-int-def int-eq-of-nat*)

lemma *real-of-int-of-nat-eq* [*simp*]: *real* (*of-nat n* :: *int*) = *real n*
by (*simp add*: *real-of-int-def real-of-nat-def*)

lemma *real-nat-eq-real* [*simp*]: *0* <= *x* ==> *real*(*nat x*) = *real x*
apply (*subgoal-tac real*(*int*(*nat x*)) = *real*(*nat x*))
apply *force*
apply (*simp only*: *real-of-int-real-of-nat*)
done

6.12 Numerals and Arithmetic

instantiation *real* :: *number-ring*
begin

definition
real-number-of-def [*code func del*]: *number-of w* = *real-of-int w*

instance
by *intro-classes* (*simp add*: *real-number-of-def*)

end

lemma [*code unfold*, *symmetric*, *code post*]:
number-of k = *real-of-int* (*number-of k*)
unfolding *number-of-is-id real-number-of-def* **..**

Collapse applications of *real* to *number-of*

lemma *real-number-of* [*simp*]: *real* (*number-of v* :: *int*) = *number-of v*
by (*simp add*: *real-of-int-def of-int-number-of-eq*)

lemma *real-of-nat-number-of* [*simp*]:
real (*number-of v* :: *nat*) =
(*if neg* (*number-of v* :: *int*) *then 0*
else (*number-of v* :: *real*))
by (*simp add*: *real-of-int-real-of-nat* [*symmetric*] *int-nat-number-of*)

use *real-arith.ML*
declaration ⟪ *K real-arith-setup* ⟫

6.13 Simprules combining x+y and 0: ARE THEY NEEDED?

Needed in this non-standard form by Hyperreal/Transcendental

lemma *real-0-le-divide-iff*:
((*0*::*real*) ≤ *x*/*y*) = ((*x* ≤ *0* | *0* ≤ *y*) & (*0* ≤ *x* | *y* ≤ *0*))
by (*simp add*: *real-divide-def zero-le-mult-iff*, *auto*)

lemma *real-add-minus-iff* [*simp*]: $(x + -a = (0::real)) = (x=a)$
by *arith*

lemma *real-add-eq-0-iff*: $(x+y = (0::real)) = (y = -x)$
by *auto*

lemma *real-add-less-0-iff*: $(x+y < (0::real)) = (y < -x)$
by *auto*

lemma *real-0-less-add-iff*: $((0::real) < x+y) = (-x < y)$
by *auto*

lemma *real-add-le-0-iff*: $(x+y \leq (0::real)) = (y \leq -x)$
by *auto*

lemma *real-0-le-add-iff*: $((0::real) \leq x+y) = (-x \leq y)$
by *auto*

6.13.1 Density of the Reals

lemma *real-lbound-gt-zero*:
$[| (0::real) < d1; 0 < d2 |] ==> \exists e.\ 0 < e \ \& \ e < d1 \ \& \ e < d2$
apply (*rule-tac x* = $(min\ d1\ d2) / 2$ **in** *exI*)
apply (*simp add*: *min-def*)
done

Similar results are proved in *Ring-and-Field*

lemma *real-less-half-sum*: $x < y ==> x < (x+y) / (2::real)$
by *auto*

lemma *real-gt-half-sum*: $x < y ==> (x+y)/(2::real) < y$
by *auto*

6.14 Absolute Value Function for the Reals

lemma *abs-minus-add-cancel*: $abs(x + (-y)) = abs\ (y + (-(x::real)))$
by (*simp add*: *abs-if*)

lemma *abs-le-interval-iff*: $(abs\ x \leq r) = (-r \leq x \ \& \ x \leq (r::real))$
by (*force simp add*: *OrderedGroup.abs-le-iff*)

lemma *abs-add-one-gt-zero* [*simp*]: $(0::real) < 1 + abs(x)$
by (*simp add*: *abs-if*)

lemma *abs-real-of-nat-cancel* [*simp*]: $abs\ (real\ x) = real\ (x::nat)$
by (*rule abs-of-nonneg* [*OF real-of-nat-ge-zero*])

lemma *abs-add-one-not-less-self* [*simp*]: $\sim abs(x) + (1::real) < x$

by *simp*

lemma *abs-sum-triangle-ineq*: *abs* ((*x*::*real*) + *y* + (−*l* + −*m*)) ≤ *abs*(*x* + −*l*) + *abs*(*y* + −*m*)
by *simp*

instance *real* :: *lordered-ring*
proof
  fix *a*::*real*
  show *abs a* = *sup a* (−*a*)
    by (*auto simp add*: *real-abs-def sup-real-def*)
qed

6.15 Implementation of rational real numbers as pairs of integers

definition
  Ratreal :: *int* × *int* ⇒ *real*
where
  Ratreal = *INum*

code-datatype *Ratreal*

lemma *Ratreal-simp*:
  Ratreal (*k*, *l*) = *real-of-int k* / *real-of-int l*
  unfolding *Ratreal-def INum-def* **by** *simp*

lemma *Ratreal-zero* [*simp*]: *Ratreal* 0_N = *0*
  by (*simp add*: *Ratreal-simp*)

lemma *Ratreal-lit* [*simp*]: *Ratreal* i_N = *real-of-int i*
  by (*simp add*: *Ratreal-simp*)

lemma *zero-real-code* [*code*, *code unfold*]:
  0 = *Ratreal* 0_N **by** *simp*
declare *zero-real-code* [*symmetric*, *code post*]

lemma *one-real-code* [*code*, *code unfold*]:
  1 = *Ratreal* 1_N **by** *simp*
declare *one-real-code* [*symmetric*, *code post*]

instantiation *real* :: *eq*
begin

definition *eq-class.eq* (*x*::*real*) *y* ⟷ *x* = *y*

instance by *default* (*simp add*: *eq-real-def*)

lemma *real-eq-code* [*code*]: *eq-class.eq* (*Ratreal x*) (*Ratreal y*) ⟷ *eq-class.eq* (*normNum*

x) ($normNum\ y$)
unfolding *Ratreal-def INum-normNum-iff eq* **..**

end

lemma *real-less-eq-code* [*code*]: $Ratreal\ x \leq Ratreal\ y \longleftrightarrow normNum\ x \leq_N normNum\ y$
proof −
have $normNum\ x \leq_N normNum\ y \longleftrightarrow Ratreal\ (normNum\ x) \leq Ratreal\ (normNum\ y)$
by (*simp add*: *Ratreal-def del*: *normNum*)
also have $\ldots = (Ratreal\ x \leq Ratreal\ y)$ **by** (*simp add*: *Ratreal-def*)
finally show *?thesis* **by** *simp*
qed

lemma *real-less-code* [*code*]: $Ratreal\ x < Ratreal\ y \longleftrightarrow normNum\ x <_N normNum\ y$
proof −
have $normNum\ x <_N normNum\ y \longleftrightarrow Ratreal\ (normNum\ x) < Ratreal\ (normNum\ y)$
by (*simp add*: *Ratreal-def del*: *normNum*)
also have $\ldots = (Ratreal\ x < Ratreal\ y)$ **by** (*simp add*: *Ratreal-def*)
finally show *?thesis* **by** *simp*
qed

lemma *real-add-code* [*code*]: $Ratreal\ x + Ratreal\ y = Ratreal\ (x +_N y)$
unfolding *Ratreal-def* **by** *simp*

lemma *real-mul-code* [*code*]: $Ratreal\ x * Ratreal\ y = Ratreal\ (x *_N y)$
unfolding *Ratreal-def* **by** *simp*

lemma *real-neg-code* [*code*]: $-\ Ratreal\ x = Ratreal\ (\sim_N x)$
unfolding *Ratreal-def* **by** *simp*

lemma *real-sub-code* [*code*]: $Ratreal\ x - Ratreal\ y = Ratreal\ (x -_N y)$
unfolding *Ratreal-def* **by** *simp*

lemma *real-inv-code* [*code*]: $inverse\ (Ratreal\ x) = Ratreal\ (Ninv\ x)$
unfolding *Ratreal-def Ninv real-divide-def* **by** *simp*

lemma *real-div-code* [*code*]: $Ratreal\ x\ /\ Ratreal\ y = Ratreal\ (x \div_N y)$
unfolding *Ratreal-def* **by** *simp*

Setup for SML code generator

types-code
real ((*int* */ int*))
attach (*term-of*) ⟪
fun term-of-real (*p*, *q*) =
let

```
    val rT = HOLogic.realT
  in
    if q = 1 orelse p = 0 then HOLogic.mk-number rT p
    else @{term op / :: real ⇒ real ⇒ real} $
      HOLogic.mk-number rT p $ HOLogic.mk-number rT q
  end;
⟩⟩
attach (test) ⟨⟨
fun gen-real i =
  let
    val p = random-range 0 i;
    val q = random-range 1 (i + 1);
    val g = Integer.gcd p q;
    val p' = p div g;
    val q' = q div g;
    val r = (if one-of [true, false] then p' else ~ p',
      if p' = 0 then 0 else q')
  in
    (r, fn () => term-of-real r)
  end;
⟩⟩

consts-code
  Ratreal ((-))

consts-code
  of-int :: int ⇒ real (⟨module⟩real'-of'-int)
attach ⟨⟨
fun real-of-int 0 = (0, 0)
  | real-of-int i = (i, 1);
⟩⟩

declare real-of-int-of-nat-eq [symmetric, code]

end
```

7 RComplete: Completeness of the Reals; Floor and Ceiling Functions

```
theory RComplete
imports Lubs RealDef
begin

lemma real-sum-of-halves: x/2 + x/2 = (x::real)
  by simp
```

7.1 Completeness of Positive Reals

Supremum property for the set of positive reals

Let P be a non-empty set of positive reals, with an upper bound y. Then P has a least upper bound (written S).

FIXME: Can the premise be weakened to $\forall x \in P.\ x \leq y$?

lemma *posreal-complete*:
  assumes *positive-P*: $\forall x \in P.\ (0::real) < x$
    and *not-empty-P*: $\exists x.\ x \in P$
    and *upper-bound-Ex*: $\exists y.\ \forall x \in P.\ x<y$
  shows $\exists S.\ \forall y.\ (\exists x \in P.\ y < x) = (y < S)$
proof (*rule exI*, *rule allI*)
  fix y
  let *?pP* $= \{w.\ real\text{-}of\text{-}preal\ w \in P\}$

  show $(\exists x{\in}P.\ y < x) = (y < real\text{-}of\text{-}preal\ (psup\ ?pP))$
  proof (*cases* $0 < y$)
    assume *neg-y*: $\neg\ 0 < y$
    show *?thesis*
    proof
      assume $\exists x{\in}P.\ y < x$
      have $\forall x.\ y < real\text{-}of\text{-}preal\ x$
        using *neg-y* **by** (*rule real-less-all-real2*)
      thus $y < real\text{-}of\text{-}preal\ (psup\ ?pP)$ **..**
    next
      assume $y < real\text{-}of\text{-}preal\ (psup\ ?pP)$
      obtain x **where** *x-in-P*: $x \in P$ **using** *not-empty-P* **..**
      hence $0 < x$ **using** *positive-P* **by** *simp*
      hence $y < x$ **using** *neg-y* **by** *simp*
      thus $\exists x \in P.\ y < x$ **using** *x-in-P* **..**
    qed
  next
    assume *pos-y*: $0 < y$

    then obtain *py* **where** *y-is-py*: $y = real\text{-}of\text{-}preal\ py$
      by (*auto simp add*: *real-gt-zero-preal-Ex*)

    obtain a **where** $a \in P$ **using** *not-empty-P* **..**
    with *positive-P* **have** *a-pos*: $0 < a$ **..**
    then obtain *pa* **where** $a = real\text{-}of\text{-}preal\ pa$
      by (*auto simp add*: *real-gt-zero-preal-Ex*)
    hence $pa \in ?pP$ **using** ‹$a \in P$› **by** *auto*
    hence *pP-not-empty*: $?pP \neq \{\}$ **by** *auto*

    obtain *sup* **where** *sup*: $\forall x \in P.\ x < sup$
      using *upper-bound-Ex* **..**
    from *this* **and** ‹$a \in P$› **have** $a < sup$ **..**
    hence $0 < sup$ **using** *a-pos* **by** *arith*

```
    then obtain possup where sup = real-of-preal possup
      by (auto simp add: real-gt-zero-preal-Ex)
    hence ∀ X ∈ ?pP. X ≤ possup
      using sup by (auto simp add: real-of-preal-lessI)
    with pP-not-empty have psup: ⋀Z. (∃ X ∈ ?pP. Z < X) = (Z < psup ?pP)
      by (rule preal-complete)

    show ?thesis
    proof
      assume ∃ x ∈ P. y < x
      then obtain x where x-in-P: x ∈ P and y-less-x: y < x ..
      hence 0 < x using pos-y by arith
      then obtain px where x-is-px: x = real-of-preal px
        by (auto simp add: real-gt-zero-preal-Ex)

      have py-less-X: ∃ X ∈ ?pP. py < X
      proof
        show py < px using y-is-py and x-is-px and y-less-x
          by (simp add: real-of-preal-lessI)
        show px ∈ ?pP using x-in-P and x-is-px by simp
      qed

      have (∃ X ∈ ?pP. py < X) ==> (py < psup ?pP)
        using psup by simp
      hence py < psup ?pP using py-less-X by simp
      thus y < real-of-preal (psup {w. real-of-preal w ∈ P})
        using y-is-py and pos-y by (simp add: real-of-preal-lessI)
    next
      assume y-less-psup: y < real-of-preal (psup ?pP)

      hence py < psup ?pP using y-is-py
        by (simp add: real-of-preal-lessI)
      then obtain X where py-less-X: py < X and X-in-pP: X ∈ ?pP
        using psup by auto
      then obtain x where x-is-X: x = real-of-preal X
        by (simp add: real-gt-zero-preal-Ex)
      hence y < x using py-less-X and y-is-py
        by (simp add: real-of-preal-lessI)

      moreover have x ∈ P using x-is-X and X-in-pP by simp

      ultimately show ∃ x ∈ P. y < x ..
    qed
  qed
qed
```

Completeness properties using *isUb*, *isLub* etc.

```
lemma real-isLub-unique: [| isLub R S x; isLub R S y |] ==> x = (y::real)
  apply (frule isLub-isUb)
```

apply (*frule-tac* $x = y$ **in** *isLub-isUb*)
apply (*blast intro*!: *order-antisym dest*!: *isLub-le-isUb*)
done

Completeness theorem for the positive reals (again).

lemma *posreals-complete*:
assumes *positive-S*: $\forall x \in S.\ 0 < x$
and *not-empty-S*: $\exists x.\ x \in S$
and *upper-bound-Ex*: $\exists u.\ isUb\ (UNIV::real\ set)\ S\ u$
shows $\exists t.\ isLub\ (UNIV::real\ set)\ S\ t$
proof
let *?pS* $= \{w.\ real\text{-}of\text{-}preal\ w \in S\}$

obtain u **where** *isUb UNIV S u* **using** *upper-bound-Ex* **..**
hence *sup*: $\forall x \in S.\ x \leq u$ **by** (*simp add*: *isUb-def setle-def*)

obtain x **where** *x-in-S*: $x \in S$ **using** *not-empty-S* **..**
hence *x-gt-zero*: $0 < x$ **using** *positive-S* **by** *simp*
have $x \leq u$ **using** *sup* **and** *x-in-S* **..**
hence $0 < u$ **using** *x-gt-zero* **by** *arith*

then obtain *pu* **where** *u-is-pu*: $u = real\text{-}of\text{-}preal\ pu$
by (*auto simp add*: *real-gt-zero-preal-Ex*)

have *pS-less-pu*: $\forall pa \in ?pS.\ pa \leq pu$
proof
fix *pa*
assume $pa \in ?pS$
then obtain a **where** $a \in S$ **and** $a = real\text{-}of\text{-}preal\ pa$
by *simp*
moreover hence $a \leq u$ **using** *sup* **by** *simp*
ultimately show $pa \leq pu$
using *sup* **and** *u-is-pu* **by** (*simp add*: *real-of-preal-le-iff*)
qed

have $\forall y \in S.\ y \leq real\text{-}of\text{-}preal\ (psup\ ?pS)$
proof
fix y
assume *y-in-S*: $y \in S$
hence $0 < y$ **using** *positive-S* **by** *simp*
then obtain *py* **where** *y-is-py*: $y = real\text{-}of\text{-}preal\ py$
by (*auto simp add*: *real-gt-zero-preal-Ex*)
hence *py-in-pS*: $py \in ?pS$ **using** *y-in-S* **by** *simp*
with *pS-less-pu* **have** $py \leq psup\ ?pS$
by (*rule preal-psup-le*)
thus $y \leq real\text{-}of\text{-}preal\ (psup\ ?pS)$
using *y-is-py* **by** (*simp add*: *real-of-preal-le-iff*)
qed

```
  moreover {
    fix x
    assume x-ub-S: ∀ y∈S. y ≤ x
    have real-of-preal (psup ?pS) ≤ x
    proof −
      obtain s where s-in-S: s ∈ S using not-empty-S ..
      hence s-pos: 0 < s using positive-S by simp

      hence ∃ ps. s = real-of-preal ps by (simp add: real-gt-zero-preal-Ex)
      then obtain ps where s-is-ps: s = real-of-preal ps ..
      hence ps-in-pS: ps ∈ {w. real-of-preal w ∈ S} using s-in-S by simp

      from x-ub-S have s ≤ x using s-in-S ..
      hence 0 < x using s-pos by simp
      hence ∃ px. x = real-of-preal px by (simp add: real-gt-zero-preal-Ex)
      then obtain px where x-is-px: x = real-of-preal px ..

      have ∀ pe ∈ ?pS. pe ≤ px
      proof
        fix pe
        assume pe ∈ ?pS
        hence real-of-preal pe ∈ S by simp
        hence real-of-preal pe ≤ x using x-ub-S by simp
        thus pe ≤ px using x-is-px by (simp add: real-of-preal-le-iff)
      qed

      moreover have ?pS ≠ {} using ps-in-pS by auto
      ultimately have (psup ?pS) ≤ px by (simp add: psup-le-ub)
    thus real-of-preal (psup ?pS) ≤ x using x-is-px by (simp add: real-of-preal-le-iff)
    qed
  }
  ultimately show isLub UNIV S (real-of-preal (psup ?pS))
    by (simp add: isLub-def leastP-def isUb-def setle-def setge-def)
qed
```

reals Completeness (again!)

```
lemma reals-complete:
  assumes notempty-S: ∃ X. X ∈ S
    and exists-Ub: ∃ Y. isUb (UNIV::real set) S Y
  shows ∃ t. isLub (UNIV :: real set) S t
proof −
  obtain X where X-in-S: X ∈ S using notempty-S ..
  obtain Y where Y-isUb: isUb (UNIV::real set) S Y
    using exists-Ub ..
  let ?SHIFT = {z. ∃ x ∈S. z = x + (−X) + 1} ∩ {x. 0 < x}

  {
    fix x
    assume isUb (UNIV::real set) S x
```

```
  hence S-le-x: ∀ y ∈ S. y <= x
    by (simp add: isUb-def setle-def)
  {
    fix s
    assume s ∈ {z. ∃x∈S. z = x + - X + 1}
    hence ∃ x ∈ S. s = x + -X + 1 ..
    then obtain x1 where x1 ∈ S and s = x1 + (-X) + 1 ..
    moreover hence x1 ≤ x using S-le-x by simp
    ultimately have s ≤ x + - X + 1 by arith
  }
  then have isUb (UNIV::real set) ?SHIFT (x + (-X) + 1)
    by (auto simp add: isUb-def setle-def)
} note S-Ub-is-SHIFT-Ub = this

hence isUb UNIV ?SHIFT (Y + (-X) + 1) using Y-isUb by simp
hence ∃ Z. isUb UNIV ?SHIFT Z ..
moreover have ∀ y ∈ ?SHIFT. 0 < y by auto
moreover have shifted-not-empty: ∃ u. u ∈ ?SHIFT
  using X-in-S and Y-isUb by auto
ultimately obtain t where t-is-Lub: isLub UNIV ?SHIFT t
  using posreals-complete [of ?SHIFT] by blast

show ?thesis
proof
  show isLub UNIV S (t + X + (-1))
  proof (rule isLubI2)
    {
      fix x
      assume isUb (UNIV::real set) S x
      hence isUb (UNIV::real set) (?SHIFT) (x + (-X) + 1)
        using S-Ub-is-SHIFT-Ub by simp
      hence t ≤ (x + (-X) + 1)
        using t-is-Lub by (simp add: isLub-le-isUb)
      hence t + X + -1 ≤ x by arith
    }
    then show (t + X + -1) <=* Collect (isUb UNIV S)
      by (simp add: setgeI)
  next
    show isUb UNIV S (t + X + -1)
    proof -
      {
        fix y
        assume y-in-S: y ∈ S
        have y ≤ t + X + -1
        proof -
          obtain u where u-in-shift: u ∈ ?SHIFT using shifted-not-empty ..
          hence ∃ x ∈ S. u = x + - X + 1 by simp
          then obtain x where x-and-u: u = x + - X + 1 ..
        have u-le-t: u ≤ t using u-in-shift and t-is-Lub by (simp add: isLubD2)
```

```
          show ?thesis
          proof cases
            assume y ≤ x
            moreover have x = u + X + − 1 using x-and-u by arith
            moreover have u + X + − 1 ≤ t + X + −1 using u-le-t by arith
            ultimately show y ≤ t + X + −1 by arith
          next
            assume ~(y ≤ x)
            hence x-less-y: x < y by arith

           have x + (−X) + 1 ∈ ?SHIFT using x-and-u and u-in-shift by simp
            hence 0 < x + (−X) + 1 by simp
            hence 0 < y + (−X) + 1 using x-less-y by arith
            hence y + (−X) + 1 ∈ ?SHIFT using y-in-S by simp
            hence y + (−X) + 1 ≤ t using t-is-Lub  by (simp add: isLubD2)
            thus ?thesis by simp
          qed
        qed
       }
       then show ?thesis by (simp add: isUb-def setle-def)
     qed
   qed
  qed
qed
```

7.2 The Archimedean Property of the Reals

```
theorem reals-Archimedean:
  assumes x-pos: 0 < x
  shows ∃ n. inverse (real (Suc n)) < x
proof (rule ccontr)
  assume contr: ¬ ?thesis
  have ∀ n. x * real (Suc n) <= 1
  proof
    fix n
    from contr have x ≤ inverse (real (Suc n))
      by (simp add: linorder-not-less)
    hence x ≤ (1 / (real (Suc n)))
      by (simp add: inverse-eq-divide)
    moreover have 0 ≤ real (Suc n)
      by (rule real-of-nat-ge-zero)
    ultimately have x * real (Suc n) ≤ (1 / real (Suc n)) * real (Suc n)
      by (rule mult-right-mono)
    thus x * real (Suc n) ≤ 1 by simp
  qed
  hence {z. ∃ n. z = x * (real (Suc n))} *<= 1
    by (simp add: setle-def, safe, rule spec)
  hence isUb (UNIV::real set) {z. ∃ n. z = x * (real (Suc n))} 1
```

```
    by (simp add: isUbI)
  hence ∃ Y. isUb (UNIV::real set) {z. ∃ n. z = x* (real (Suc n))} Y ..
  moreover have ∃ X. X ∈ {z. ∃ n. z = x* (real (Suc n))} by auto
  ultimately have ∃ t. isLub UNIV {z. ∃ n. z = x * real (Suc n)} t
    by (simp add: reals-complete)
  then obtain t where
    t-is-Lub: isLub UNIV {z. ∃ n. z = x * real (Suc n)} t ..

  have ∀ n::nat. x * real n ≤ t + - x
  proof
    fix n
    from t-is-Lub have x * real (Suc n) ≤ t
      by (simp add: isLubD2)
    hence  x * (real n) + x ≤ t
      by (simp add: right-distrib real-of-nat-Suc)
    thus  x * (real n) ≤ t + - x by arith
  qed

  hence ∀ m. x * real (Suc m) ≤ t + - x by simp
  hence {z. ∃ n. z = x * (real (Suc n))}  *<= (t + - x)
    by (auto simp add: setle-def)
  hence isUb (UNIV::real set) {z. ∃ n. z = x * (real (Suc n))} (t + (-x))
    by (simp add: isUbI)
  hence t ≤ t + - x
    using t-is-Lub by (simp add: isLub-le-isUb)
  thus False using x-pos by arith
qed
```

There must be other proofs, e.g. *Suc* of the largest integer in the cut representing x.

```
lemma reals-Archimedean2: ∃ n. (x::real) < real (n::nat)
proof cases
  assume x ≤ 0
  hence x < real (1::nat) by simp
  thus ?thesis ..
next
  assume ¬ x ≤ 0
  hence x-greater-zero: 0 < x by simp
  hence 0 < inverse x by simp
  then obtain n where inverse (real (Suc n)) < inverse x
    using reals-Archimedean by blast
  hence inverse (real (Suc n)) * x < inverse x * x
    using x-greater-zero by (rule mult-strict-right-mono)
  hence inverse (real (Suc n)) * x < 1
    using x-greater-zero by simp
  hence real (Suc n) * (inverse (real (Suc n)) * x) < real (Suc n) * 1
    by (rule mult-strict-left-mono) simp
  hence x < real (Suc n)
    by (simp add: ring-simps)
```

thus $\exists (n::nat).\ x < real\ n$ **..**
qed

lemma *reals-Archimedean3*:
assumes *x-greater-zero*: $0 < x$
shows $\forall (y::real).\ \exists (n::nat).\ y < real\ n * x$
proof
fix y
have *x-not-zero*: $x \neq 0$ **using** *x-greater-zero* **by** *simp*
obtain n **where** $y * inverse\ x < real\ (n::nat)$
using *reals-Archimedean2* **..**
hence $y * inverse\ x * x < real\ n * x$
using *x-greater-zero* **by** (*simp add*: *mult-strict-right-mono*)
hence $x * inverse\ x * y < x * real\ n$
by (*simp add*: *ring-simps*)
hence $y < real\ (n::nat) * x$
using *x-not-zero* **by** (*simp add*: *ring-simps*)
thus $\exists (n::nat).\ y < real\ n * x$ **..**
qed

lemma *reals-Archimedean6*:
$0 \leq r \Longrightarrow \exists (n::nat).\ real\ (n - 1) \leq r\ \&\ r < real\ (n)$
apply (*insert reals-Archimedean2* [*of r*], *safe*)
apply (*subgoal-tac* $\exists x::nat.\ r < real\ x \wedge (\forall y.\ r < real\ y \longrightarrow x \leq y)$, *auto*)
apply (*rule-tac* $x = x$ **in** *exI*)
apply (*case-tac* x, *simp*)
apply (*rename-tac* x')
apply (*drule-tac* $x = x'$ **in** *spec*, *simp*)
apply (*rule-tac* x=*LEAST* $n.\ r < real\ n$ **in** *exI*, *safe*)
apply (*erule LeastI*, *erule Least-le*)
done

lemma *reals-Archimedean6a*: $0 \leq r \Longrightarrow \exists n.\ real\ (n) \leq r\ \&\ r < real\ (Suc\ n)$
by (*drule reals-Archimedean6*) *auto*

lemma *reals-Archimedean-6b-int*:
$0 \leq r \Longrightarrow \exists n::int.\ real\ n \leq r\ \&\ r < real\ (n+1)$
apply (*drule reals-Archimedean6a*, *auto*)
apply (*rule-tac* $x = int\ n$ **in** *exI*)
apply (*simp add*: *real-of-int-real-of-nat real-of-nat-Suc*)
done

lemma *reals-Archimedean-6c-int*:
$r < 0 \Longrightarrow \exists n::int.\ real\ n \leq r\ \&\ r < real\ (n+1)$
apply (*rule reals-Archimedean-6b-int* [*of* $-r$, *THEN exE*], *simp*, *auto*)
apply (*rename-tac n*)
apply (*drule order-le-imp-less-or-eq*, *auto*)
apply (*rule-tac* $x = -\ n - 1$ **in** *exI*)
apply (*rule-tac* [*2*] $x = -\ n$ **in** *exI*, *auto*)

done

7.3 Floor and Ceiling Functions from the Reals to the Integers

definition
floor :: *real* => *int* **where**
floor r = (*LEAST n*::*int*. *r* < *real* (*n*+*1*))

definition
ceiling :: *real* => *int* **where**
ceiling r = − *floor* (− *r*)

notation (*xsymbols*)
floor (⌊-⌋) **and**
ceiling (⌈-⌉)

notation (*HTML* **output**)
floor (⌊-⌋) **and**
ceiling (⌈-⌉)

lemma *number-of-less-real-of-int-iff* [*simp*]:
((*number-of n*) < *real* (*m*::*int*)) = (*number-of n* < *m*)
apply *auto*
apply (*rule real-of-int-less-iff* [*THEN iffD1*])
apply (*drule-tac* [*2*] *real-of-int-less-iff* [*THEN iffD2*], *auto*)
done

lemma *number-of-less-real-of-int-iff2* [*simp*]:
(*real* (*m*::*int*) < (*number-of n*)) = (*m* < *number-of n*)
apply *auto*
apply (*rule real-of-int-less-iff* [*THEN iffD1*])
apply (*drule-tac* [*2*] *real-of-int-less-iff* [*THEN iffD2*], *auto*)
done

lemma *number-of-le-real-of-int-iff* [*simp*]:
((*number-of n*) ≤ *real* (*m*::*int*)) = (*number-of n* ≤ *m*)
by (*simp add*: *linorder-not-less* [*symmetric*])

lemma *number-of-le-real-of-int-iff2* [*simp*]:
(*real* (*m*::*int*) ≤ (*number-of n*)) = (*m* ≤ *number-of n*)
by (*simp add*: *linorder-not-less* [*symmetric*])

lemma *floor-zero* [*simp*]: *floor 0* = *0*
apply (*simp add*: *floor-def del*: *real-of-int-add*)
apply (*rule Least-equality*)
apply *simp-all*
done

lemma *floor-real-of-nat-zero* [*simp*]: *floor* (*real* (*0*::*nat*)) = *0*
by *auto*

lemma *floor-real-of-nat* [*simp*]: *floor* (*real* (*n*::*nat*)) = *int n*
apply (*simp only*: *floor-def*)
apply (*rule Least-equality*)
apply (*drule-tac* [*2*] *real-of-int-of-nat-eq* [*THEN ssubst*])
apply (*drule-tac* [*2*] *real-of-int-less-iff* [*THEN iffD1*])
apply *simp-all*
done

lemma *floor-minus-real-of-nat* [*simp*]: *floor* (− *real* (*n*::*nat*)) = − *int n*
apply (*simp only*: *floor-def*)
apply (*rule Least-equality*)
apply (*drule-tac* [*2*] *real-of-int-of-nat-eq* [*THEN ssubst*])
apply (*drule-tac* [*2*] *real-of-int-minus* [*THEN sym*, *THEN subst*])
apply (*drule-tac* [*2*] *real-of-int-less-iff* [*THEN iffD1*])
apply *simp-all*
done

lemma *floor-real-of-int* [*simp*]: *floor* (*real* (*n*::*int*)) = *n*
apply (*simp only*: *floor-def*)
apply (*rule Least-equality*)
apply *auto*
done

lemma *floor-minus-real-of-int* [*simp*]: *floor* (− *real* (*n*::*int*)) = − *n*
apply (*simp only*: *floor-def*)
apply (*rule Least-equality*)
apply (*drule-tac* [*2*] *real-of-int-minus* [*THEN sym*, *THEN subst*])
apply *auto*
done

lemma *real-lb-ub-int*: ∃ *n*::*int*. *real n* ≤ *r* & *r* < *real* (*n*+*1*)
apply (*case-tac r* < *0*)
apply (*blast intro*: *reals-Archimedean-6c-int*)
apply (*simp only*: *linorder-not-less*)
apply (*blast intro*: *reals-Archimedean-6b-int reals-Archimedean-6c-int*)
done

lemma *lemma-floor*:
  assumes *a1*: *real m* ≤ *r* **and** *a2*: *r* < *real n* + *1*
  shows *m* ≤ (*n*::*int*)
proof −
  have *real m* < *real n* + *1* **using** *a1 a2* **by** (*rule order-le-less-trans*)
  also have ... = *real* (*n* + *1*) **by** *simp*
  finally have *m* < *n* + *1* **by** (*simp only*: *real-of-int-less-iff*)
  thus *?thesis* **by** *arith*

qed

lemma *real-of-int-floor-le* [*simp*]: *real* (*floor r*) ≤ *r*
apply (*simp add*: *floor-def Least-def*)
apply (*insert real-lb-ub-int* [*of r*], *safe*)
apply (*rule theI2*)
apply *auto*
done

lemma *floor-mono*: *x* < *y* ==> *floor x* ≤ *floor y*
apply (*simp add*: *floor-def Least-def*)
apply (*insert real-lb-ub-int* [*of x*])
apply (*insert real-lb-ub-int* [*of y*], *safe*)
apply (*rule theI2*)
apply (*rule-tac* [*3*] *theI2*)
apply *simp*
apply (*erule conjI*)
apply (*auto simp add*: *order-eq-iff int-le-real-less*)
done

lemma *floor-mono2*: *x* ≤ *y* ==> *floor x* ≤ *floor y*
by (*auto dest*: *order-le-imp-less-or-eq simp add*: *floor-mono*)

lemma *lemma-floor2*: *real n* < *real* (*x*::*int*) + *1* ==> *n* ≤ *x*
by (*auto intro*: *lemma-floor*)

lemma *real-of-int-floor-cancel* [*simp*]:
(*real* (*floor x*) = *x*) = (∃ *n*::*int*. *x* = *real n*)
apply (*simp add*: *floor-def Least-def*)
apply (*insert real-lb-ub-int* [*of x*], *erule exE*)
apply (*rule theI2*)
apply (*auto intro*: *lemma-floor*)
done

lemma *floor-eq*: [| *real n* < *x*; *x* < *real n* + *1* |] ==> *floor x* = *n*
apply (*simp add*: *floor-def*)
apply (*rule Least-equality*)
apply (*auto intro*: *lemma-floor*)
done

lemma *floor-eq2*: [| *real n* ≤ *x*; *x* < *real n* + *1* |] ==> *floor x* = *n*
apply (*simp add*: *floor-def*)
apply (*rule Least-equality*)
apply (*auto intro*: *lemma-floor*)
done

lemma *floor-eq3*: [| *real n* < *x*; *x* < *real* (*Suc n*) |] ==> *nat*(*floor x*) = *n*
apply (*rule inj-int* [*THEN injD*])
apply (*simp add*: *real-of-nat-Suc*)

apply (*simp add*: *real-of-nat-Suc floor-eq floor-eq* [**where** *n* = *int n*])
done

lemma *floor-eq4*: [| *real n* ≤ *x*; *x* < *real* (*Suc n*) |] ==> *nat*(*floor x*) = *n*
apply (*drule order-le-imp-less-or-eq*)
apply (*auto intro*: *floor-eq3*)
done

lemma *floor-number-of-eq* [*simp*]:
    floor(*number-of n* :: *real*) = (*number-of n* :: *int*)
apply (*subst real-number-of* [*symmetric*])
apply (*rule floor-real-of-int*)
done

lemma *floor-one* [*simp*]: *floor 1* = *1*
  apply (*rule trans*)
  prefer *2*
  apply (*rule floor-real-of-int*)
  apply *simp*
done

lemma *real-of-int-floor-ge-diff-one* [*simp*]: *r* − *1* ≤ *real*(*floor r*)
apply (*simp add*: *floor-def Least-def*)
apply (*insert real-lb-ub-int* [*of r*], *safe*)
apply (*rule theI2*)
apply (*auto intro*: *lemma-floor*)
done

lemma *real-of-int-floor-gt-diff-one* [*simp*]: *r* − *1* < *real*(*floor r*)
apply (*simp add*: *floor-def Least-def*)
apply (*insert real-lb-ub-int* [*of r*], *safe*)
apply (*rule theI2*)
apply (*auto intro*: *lemma-floor*)
done

lemma *real-of-int-floor-add-one-ge* [*simp*]: *r* ≤ *real*(*floor r*) + *1*
apply (*insert real-of-int-floor-ge-diff-one* [*of r*])
apply (*auto simp del*: *real-of-int-floor-ge-diff-one*)
done

lemma *real-of-int-floor-add-one-gt* [*simp*]: *r* < *real*(*floor r*) + *1*
apply (*insert real-of-int-floor-gt-diff-one* [*of r*])
apply (*auto simp del*: *real-of-int-floor-gt-diff-one*)
done

lemma *le-floor*: *real a* <= *x* ==> *a* <= *floor x*
  apply (*subgoal-tac a* < *floor x* + *1*)
  apply *arith*
  apply (*subst real-of-int-less-iff* [*THEN sym*])

```
  apply simp
  apply (insert real-of-int-floor-add-one-gt [of x])
  apply arith
done

lemma real-le-floor: a <= floor x ==> real a <= x
  apply (rule order-trans)
  prefer 2
  apply (rule real-of-int-floor-le)
  apply (subst real-of-int-le-iff)
  apply assumption
done

lemma le-floor-eq: (a <= floor x) = (real a <= x)
  apply (rule iffI)
  apply (erule real-le-floor)
  apply (erule le-floor)
done

lemma le-floor-eq-number-of [simp]:
    (number-of n <= floor x) = (number-of n <= x)
by (simp add: le-floor-eq)

lemma le-floor-eq-zero [simp]: (0 <= floor x) = (0 <= x)
by (simp add: le-floor-eq)

lemma le-floor-eq-one [simp]: (1 <= floor x) = (1 <= x)
by (simp add: le-floor-eq)

lemma floor-less-eq: (floor x < a) = (x < real a)
  apply (subst linorder-not-le [THEN sym])+
  apply simp
  apply (rule le-floor-eq)
done

lemma floor-less-eq-number-of [simp]:
    (floor x < number-of n) = (x < number-of n)
by (simp add: floor-less-eq)

lemma floor-less-eq-zero [simp]: (floor x < 0) = (x < 0)
by (simp add: floor-less-eq)

lemma floor-less-eq-one [simp]: (floor x < 1) = (x < 1)
by (simp add: floor-less-eq)

lemma less-floor-eq: (a < floor x) = (real a + 1 <= x)
  apply (insert le-floor-eq [of a + 1 x])
  apply auto
done
```

lemma *less-floor-eq-number-of* [*simp*]:
(*number-of n* < *floor x*) = (*number-of n* + *1* <= *x*)
by (*simp add*: *less-floor-eq*)

lemma *less-floor-eq-zero* [*simp*]: (*0* < *floor x*) = (*1* <= *x*)
by (*simp add*: *less-floor-eq*)

lemma *less-floor-eq-one* [*simp*]: (*1* < *floor x*) = (*2* <= *x*)
by (*simp add*: *less-floor-eq*)

lemma *floor-le-eq*: (*floor x* <= *a*) = (*x* < *real a* + *1*)
apply (*insert floor-less-eq* [*of x a* + *1*])
apply *auto*
done

lemma *floor-le-eq-number-of* [*simp*]:
(*floor x* <= *number-of n*) = (*x* < *number-of n* + *1*)
by (*simp add*: *floor-le-eq*)

lemma *floor-le-eq-zero* [*simp*]: (*floor x* <= *0*) = (*x* < *1*)
by (*simp add*: *floor-le-eq*)

lemma *floor-le-eq-one* [*simp*]: (*floor x* <= *1*) = (*x* < *2*)
by (*simp add*: *floor-le-eq*)

lemma *floor-add* [*simp*]: *floor* (*x* + *real a*) = *floor x* + *a*
apply (*subst order-eq-iff*)
apply (*rule conjI*)
prefer *2*
apply (*subgoal-tac floor x* + *a* < *floor* (*x* + *real a*) + *1*)
apply *arith*
apply (*subst real-of-int-less-iff* [*THEN sym*])
apply *simp*
apply (*subgoal-tac x* + *real a* < *real*(*floor*(*x* + *real a*)) + *1*)
apply (*subgoal-tac real* (*floor x*) <= *x*)
apply *arith*
apply (*rule real-of-int-floor-le*)
apply (*rule real-of-int-floor-add-one-gt*)
apply (*subgoal-tac floor* (*x* + *real a*) < *floor x* + *a* + *1*)
apply *arith*
apply (*subst real-of-int-less-iff* [*THEN sym*])
apply *simp*
apply (*subgoal-tac real*(*floor*(*x* + *real a*)) <= *x* + *real a*)
apply (*subgoal-tac x* < *real*(*floor x*) + *1*)
apply *arith*
apply (*rule real-of-int-floor-add-one-gt*)
apply (*rule real-of-int-floor-le*)
done

lemma *floor-add-number-of* [*simp*]:
  floor (*x* + *number-of n*) = *floor x* + *number-of n*
 apply (*subst floor-add* [*THEN sym*])
 apply *simp*
done

lemma *floor-add-one* [*simp*]: *floor* (*x* + *1*) = *floor x* + *1*
 apply (*subst floor-add* [*THEN sym*])
 apply *simp*
done

lemma *floor-subtract* [*simp*]: *floor* (*x* − *real a*) = *floor x* − *a*
 apply (*subst diff-minus*)+
 apply (*subst real-of-int-minus* [*THEN sym*])
 apply (*rule floor-add*)
done

lemma *floor-subtract-number-of* [*simp*]: *floor* (*x* − *number-of n*) =
  floor x − *number-of n*
 apply (*subst floor-subtract* [*THEN sym*])
 apply *simp*
done

lemma *floor-subtract-one* [*simp*]: *floor* (*x* − *1*) = *floor x* − *1*
 apply (*subst floor-subtract* [*THEN sym*])
 apply *simp*
done

lemma *ceiling-zero* [*simp*]: *ceiling 0* = *0*
by (*simp add*: *ceiling-def*)

lemma *ceiling-real-of-nat* [*simp*]: *ceiling* (*real* (*n*::*nat*)) = *int n*
by (*simp add*: *ceiling-def*)

lemma *ceiling-real-of-nat-zero* [*simp*]: *ceiling* (*real* (*0*::*nat*)) = *0*
by *auto*

lemma *ceiling-floor* [*simp*]: *ceiling* (*real* (*floor r*)) = *floor r*
by (*simp add*: *ceiling-def*)

lemma *floor-ceiling* [*simp*]: *floor* (*real* (*ceiling r*)) = *ceiling r*
by (*simp add*: *ceiling-def*)

lemma *real-of-int-ceiling-ge* [*simp*]: *r* ≤ *real* (*ceiling r*)
apply (*simp add*: *ceiling-def*)
apply (*subst le-minus-iff*, *simp*)
done

lemma *ceiling-mono*: $x < y \Longrightarrow \mathit{ceiling}\ x \leq \mathit{ceiling}\ y$
by (*simp add*: *floor-mono ceiling-def*)

lemma *ceiling-mono2*: $x \leq y \Longrightarrow \mathit{ceiling}\ x \leq \mathit{ceiling}\ y$
by (*simp add*: *floor-mono2 ceiling-def*)

lemma *real-of-int-ceiling-cancel* [*simp*]:
    $(\mathit{real}\ (\mathit{ceiling}\ x) = x) = (\exists\, n{::}\mathit{int}.\ x = \mathit{real}\ n)$
apply (*auto simp add*: *ceiling-def*)
apply (*drule arg-cong* [**where** $f = \mathit{uminus}$], *auto*)
apply (*rule-tac* $x = -n$ **in** *exI*, *auto*)
done

lemma *ceiling-eq*: $[\!|\ \mathit{real}\ n < x;\ x < \mathit{real}\ n + 1\ |\!] \Longrightarrow \mathit{ceiling}\ x = n + 1$
apply (*simp add*: *ceiling-def*)
apply (*rule minus-equation-iff* [*THEN iffD1*])
apply (*simp add*: *floor-eq* [**where** $n = -(n+1)$])
done

lemma *ceiling-eq2*: $[\!|\ \mathit{real}\ n < x;\ x \leq \mathit{real}\ n + 1\ |\!] \Longrightarrow \mathit{ceiling}\ x = n + 1$
by (*simp add*: *ceiling-def floor-eq2* [**where** $n = -(n+1)$])

lemma *ceiling-eq3*: $[\!|\ \mathit{real}\ n - 1 < x;\ x \leq \mathit{real}\ n\ |\!] \Longrightarrow \mathit{ceiling}\ x = n$
by (*simp add*: *ceiling-def floor-eq2* [**where** $n = -n$])

lemma *ceiling-real-of-int* [*simp*]: $\mathit{ceiling}\ (\mathit{real}\ (n{::}\mathit{int})) = n$
by (*simp add*: *ceiling-def*)

lemma *ceiling-number-of-eq* [*simp*]:
    $\mathit{ceiling}\ (\mathit{number\text{-}of}\ n :: \mathit{real}) = (\mathit{number\text{-}of}\ n)$
apply (*subst real-number-of* [*symmetric*])
apply (*rule ceiling-real-of-int*)
done

lemma *ceiling-one* [*simp*]: $\mathit{ceiling}\ 1 = 1$
  by (*unfold ceiling-def*, *simp*)

lemma *real-of-int-ceiling-diff-one-le* [*simp*]: $\mathit{real}\ (\mathit{ceiling}\ r) - 1 \leq r$
apply (*rule neg-le-iff-le* [*THEN iffD1*])
apply (*simp add*: *ceiling-def diff-minus*)
done

lemma *real-of-int-ceiling-le-add-one* [*simp*]: $\mathit{real}\ (\mathit{ceiling}\ r) \leq r + 1$
apply (*insert real-of-int-ceiling-diff-one-le* [*of r*])
apply (*simp del*: *real-of-int-ceiling-diff-one-le*)
done

lemma *ceiling-le*: $x <= \mathit{real}\ a \Longrightarrow \mathit{ceiling}\ x <= a$
  apply (*unfold ceiling-def*)

apply (*subgoal-tac* $-a <= \mathit{floor}(-x)$)
apply *simp*
apply (*rule le-floor*)
apply *simp*
done

lemma *ceiling-le-real*: *ceiling* $x <= a \Longrightarrow x <= \mathit{real}\ a$
apply (*unfold ceiling-def*)
apply (*subgoal-tac* $\mathit{real}(-a) <= -x$)
apply *simp*
apply (*rule real-le-floor*)
apply *simp*
done

lemma *ceiling-le-eq*: $(\mathit{ceiling}\ x <= a) = (x <= \mathit{real}\ a)$
apply (*rule iffI*)
apply (*erule ceiling-le-real*)
apply (*erule ceiling-le*)
done

lemma *ceiling-le-eq-number-of* [*simp*]:
$(\mathit{ceiling}\ x <= \mathit{number\text{-}of}\ n) = (x <= \mathit{number\text{-}of}\ n)$
by (*simp add*: *ceiling-le-eq*)

lemma *ceiling-le-zero-eq* [*simp*]: $(\mathit{ceiling}\ x <= 0) = (x <= 0)$
by (*simp add*: *ceiling-le-eq*)

lemma *ceiling-le-eq-one* [*simp*]: $(\mathit{ceiling}\ x <= 1) = (x <= 1)$
by (*simp add*: *ceiling-le-eq*)

lemma *less-ceiling-eq*: $(a < \mathit{ceiling}\ x) = (\mathit{real}\ a < x)$
apply (*subst linorder-not-le* [*THEN sym*])+
apply *simp*
apply (*rule ceiling-le-eq*)
done

lemma *less-ceiling-eq-number-of* [*simp*]:
$(\mathit{number\text{-}of}\ n < \mathit{ceiling}\ x) = (\mathit{number\text{-}of}\ n < x)$
by (*simp add*: *less-ceiling-eq*)

lemma *less-ceiling-eq-zero* [*simp*]: $(0 < \mathit{ceiling}\ x) = (0 < x)$
by (*simp add*: *less-ceiling-eq*)

lemma *less-ceiling-eq-one* [*simp*]: $(1 < \mathit{ceiling}\ x) = (1 < x)$
by (*simp add*: *less-ceiling-eq*)

lemma *ceiling-less-eq*: $(\mathit{ceiling}\ x < a) = (x <= \mathit{real}\ a - 1)$
apply (*insert ceiling-le-eq* [*of x a* $- 1$])
apply *auto*

done

lemma *ceiling-less-eq-number-of* [*simp*]:
(*ceiling x* < *number-of n*) = (*x* <= *number-of n* − *1*)
by (*simp add*: *ceiling-less-eq*)

lemma *ceiling-less-eq-zero* [*simp*]: (*ceiling x* < *0*) = (*x* <= −*1*)
by (*simp add*: *ceiling-less-eq*)

lemma *ceiling-less-eq-one* [*simp*]: (*ceiling x* < *1*) = (*x* <= *0*)
by (*simp add*: *ceiling-less-eq*)

lemma *le-ceiling-eq*: (*a* <= *ceiling x*) = (*real a* − *1* < *x*)
apply (*insert less-ceiling-eq* [*of a* − *1 x*])
apply *auto*
done

lemma *le-ceiling-eq-number-of* [*simp*]:
(*number-of n* <= *ceiling x*) = (*number-of n* − *1* < *x*)
by (*simp add*: *le-ceiling-eq*)

lemma *le-ceiling-eq-zero* [*simp*]: (*0* <= *ceiling x*) = (−*1* < *x*)
by (*simp add*: *le-ceiling-eq*)

lemma *le-ceiling-eq-one* [*simp*]: (*1* <= *ceiling x*) = (*0* < *x*)
by (*simp add*: *le-ceiling-eq*)

lemma *ceiling-add* [*simp*]: *ceiling* (*x* + *real a*) = *ceiling x* + *a*
apply (*unfold ceiling-def*, *simp*)
apply (*subst real-of-int-minus* [*THEN sym*])
apply (*subst floor-add*)
apply *simp*
done

lemma *ceiling-add-number-of* [*simp*]: *ceiling* (*x* + *number-of n*) =
ceiling x + *number-of n*
apply (*subst ceiling-add* [*THEN sym*])
apply *simp*
done

lemma *ceiling-add-one* [*simp*]: *ceiling* (*x* + *1*) = *ceiling x* + *1*
apply (*subst ceiling-add* [*THEN sym*])
apply *simp*
done

lemma *ceiling-subtract* [*simp*]: *ceiling* (*x* − *real a*) = *ceiling x* − *a*
apply (*subst diff-minus*)+
apply (*subst real-of-int-minus* [*THEN sym*])
apply (*rule ceiling-add*)

done

lemma *ceiling-subtract-number-of* [*simp*]: *ceiling* (*x* − *number-of n*) =
ceiling x − *number-of n*
apply (*subst ceiling-subtract* [*THEN sym*])
apply *simp*
done

lemma *ceiling-subtract-one* [*simp*]: *ceiling* (*x* − *1*) = *ceiling x* − *1*
apply (*subst ceiling-subtract* [*THEN sym*])
apply *simp*
done

7.4 Versions for the natural numbers

definition
natfloor :: *real* => *nat* **where**
natfloor x = *nat*(*floor x*)

definition
natceiling :: *real* => *nat* **where**
natceiling x = *nat*(*ceiling x*)

lemma *natfloor-zero* [*simp*]: *natfloor 0* = *0*
by (*unfold natfloor-def*, *simp*)

lemma *natfloor-one* [*simp*]: *natfloor 1* = *1*
by (*unfold natfloor-def*, *simp*)

lemma *zero-le-natfloor* [*simp*]: *0* <= *natfloor x*
by (*unfold natfloor-def*, *simp*)

lemma *natfloor-number-of-eq* [*simp*]: *natfloor* (*number-of n*) = *number-of n*
by (*unfold natfloor-def*, *simp*)

lemma *natfloor-real-of-nat* [*simp*]: *natfloor*(*real n*) = *n*
by (*unfold natfloor-def*, *simp*)

lemma *real-natfloor-le*: *0* <= *x* ==> *real*(*natfloor x*) <= *x*
by (*unfold natfloor-def*, *simp*)

lemma *natfloor-neg*: *x* <= *0* ==> *natfloor x* = *0*
apply (*unfold natfloor-def*)
apply (*subgoal-tac floor x* <= *floor 0*)
apply *simp*
apply (*erule floor-mono2*)
done

lemma *natfloor-mono*: *x* <= *y* ==> *natfloor x* <= *natfloor y*

apply (*case-tac* 0 <= *x*)
apply (*subst natfloor-def*)+
apply (*subst nat-le-eq-zle*)
apply *force*
apply (*erule floor-mono2*)
apply (*subst natfloor-neg*)
apply *simp*
apply *simp*
done

lemma *le-natfloor*: *real x* <= *a* ==> *x* <= *natfloor a*
apply (*unfold natfloor-def*)
apply (*subst nat-int* [*THEN sym*])
apply (*subst nat-le-eq-zle*)
apply *simp*
apply (*rule le-floor*)
apply *simp*
done

lemma *le-natfloor-eq*: *0* <= *x* ==> (*a* <= *natfloor x*) = (*real a* <= *x*)
apply (*rule iffI*)
apply (*rule order-trans*)
prefer *2*
apply (*erule real-natfloor-le*)
apply (*subst real-of-nat-le-iff*)
apply *assumption*
apply (*erule le-natfloor*)
done

lemma *le-natfloor-eq-number-of* [*simp*]:
~ *neg*((*number-of n*)::*int*) ==> *0* <= *x* ==>
(*number-of n* <= *natfloor x*) = (*number-of n* <= *x*)
apply (*subst le-natfloor-eq*, *assumption*)
apply *simp*
done

lemma *le-natfloor-eq-one* [*simp*]: (*1* <= *natfloor x*) = (*1* <= *x*)
apply (*case-tac* 0 <= *x*)
apply (*subst le-natfloor-eq*, *assumption*, *simp*)
apply (*rule iffI*)
apply (*subgoal-tac natfloor x* <= *natfloor* 0)
apply *simp*
apply (*rule natfloor-mono*)
apply *simp*
apply *simp*
done

lemma *natfloor-eq*: *real n* <= *x* ==> *x* < *real n* + *1* ==> *natfloor x* = *n*
apply (*unfold natfloor-def*)

apply (*subst nat-int* [*THEN sym*])**back**
apply (*subst eq-nat-nat-iff*)
apply *simp*
apply *simp*
apply (*rule floor-eq2*)
apply *auto*
done

lemma *real-natfloor-add-one-gt*: *x* < *real*(*natfloor x*) + *1*
apply (*case-tac 0* <= *x*)
apply (*unfold natfloor-def*)
apply *simp*
apply *simp-all*
done

lemma *real-natfloor-gt-diff-one*: *x* − *1* < *real*(*natfloor x*)
apply (*simp add*: *compare-rls*)
apply (*rule real-natfloor-add-one-gt*)
done

lemma *ge-natfloor-plus-one-imp-gt*: *natfloor z* + *1* <= *n* ==> *z* < *real n*
apply (*subgoal-tac z* < *real*(*natfloor z*) + *1*)
apply *arith*
apply (*rule real-natfloor-add-one-gt*)
done

lemma *natfloor-add* [*simp*]: *0* <= *x* ==> *natfloor* (*x* + *real a*) = *natfloor x* + *a*
apply (*unfold natfloor-def*)
apply (*subgoal-tac real a* = *real* (*int a*))
apply (*erule ssubst*)
apply (*simp add*: *nat-add-distrib del*: *real-of-int-of-nat-eq*)
apply *simp*
done

lemma *natfloor-add-number-of* [*simp*]:
~*neg* ((*number-of n*)::*int*) ==> *0* <= *x* ==>
natfloor (*x* + *number-of n*) = *natfloor x* + *number-of n*
apply (*subst natfloor-add* [*THEN sym*])
apply *simp-all*
done

lemma *natfloor-add-one*: *0* <= *x* ==> *natfloor*(*x* + *1*) = *natfloor x* + *1*
apply (*subst natfloor-add* [*THEN sym*])
apply *assumption*
apply *simp*
done

lemma *natfloor-subtract* [*simp*]: *real a* <= *x* ==>
natfloor(*x* − *real a*) = *natfloor x* − *a*

apply (*unfold natfloor-def*)
apply (*subgoal-tac real a = real* (*int a*))
apply (*erule ssubst*)
apply (*simp del*: *real-of-int-of-nat-eq*)
apply *simp*
done

lemma *natceiling-zero* [*simp*]: *natceiling 0 = 0*
by (*unfold natceiling-def*, *simp*)

lemma *natceiling-one* [*simp*]: *natceiling 1 = 1*
by (*unfold natceiling-def*, *simp*)

lemma *zero-le-natceiling* [*simp*]: *0 <= natceiling x*
by (*unfold natceiling-def*, *simp*)

lemma *natceiling-number-of-eq* [*simp*]: *natceiling* (*number-of n*) = *number-of n*
by (*unfold natceiling-def*, *simp*)

lemma *natceiling-real-of-nat* [*simp*]: *natceiling*(*real n*) = *n*
by (*unfold natceiling-def*, *simp*)

lemma *real-natceiling-ge*: *x <= real*(*natceiling x*)
apply (*unfold natceiling-def*)
apply (*case-tac x < 0*)
apply *simp*
apply (*subst real-nat-eq-real*)
apply (*subgoal-tac ceiling 0 <= ceiling x*)
apply *simp*
apply (*rule ceiling-mono2*)
apply *simp*
apply *simp*
done

lemma *natceiling-neg*: *x <= 0 ==> natceiling x = 0*
apply (*unfold natceiling-def*)
apply *simp*
done

lemma *natceiling-mono*: *x <= y ==> natceiling x <= natceiling y*
apply (*case-tac 0 <= x*)
apply (*subst natceiling-def*)+
apply (*subst nat-le-eq-zle*)
apply (*rule disjI2*)
apply (*subgoal-tac real* (*0*::*int*) <= *real*(*ceiling y*))
apply *simp*
apply (*rule order-trans*)
apply *simp*
apply (*erule order-trans*)

apply *simp*
apply (*erule ceiling-mono2*)
apply (*subst natceiling-neg*)
apply *simp-all*
done

lemma *natceiling-le*: *x <= real a ==> natceiling x <= a*
apply (*unfold natceiling-def*)
apply (*case-tac x < 0*)
apply *simp*
apply (*subst nat-int* [*THEN sym*])**back**
apply (*subst nat-le-eq-zle*)
apply *simp*
apply (*rule ceiling-le*)
apply *simp*
done

lemma *natceiling-le-eq*: *0 <= x ==> (natceiling x <= a) = (x <= real a)*
apply (*rule iffI*)
apply (*rule order-trans*)
apply (*rule real-natceiling-ge*)
apply (*subst real-of-nat-le-iff*)
apply *assumption*
apply (*erule natceiling-le*)
done

lemma *natceiling-le-eq-number-of* [*simp*]:
~ *neg*((*number-of n*)::*int*) *==> 0 <= x ==>*
(*natceiling x <= number-of n*) = (*x <= number-of n*)
apply (*subst natceiling-le-eq*, *assumption*)
apply *simp*
done

lemma *natceiling-le-eq-one*: (*natceiling x <= 1*) = (*x <= 1*)
apply (*case-tac 0 <= x*)
apply (*subst natceiling-le-eq*)
apply *assumption*
apply *simp*
apply (*subst natceiling-neg*)
apply *simp*
apply *simp*
done

lemma *natceiling-eq*: *real n < x ==> x <= real n + 1 ==> natceiling x = n + 1*
apply (*unfold natceiling-def*)
apply (*simplesubst nat-int* [*THEN sym*]) **back back**
apply (*subgoal-tac nat*(*int n*) + *1* = *nat*(*int n* + *1*))
apply (*erule ssubst*)

```
  apply (subst eq-nat-nat-iff)
  apply (subgoal-tac ceiling 0 <= ceiling x)
  apply simp
  apply (rule ceiling-mono2)
  apply force
  apply force
  apply (rule ceiling-eq2)
  apply (simp, simp)
  apply (subst nat-add-distrib)
  apply auto
done

lemma natceiling-add [simp]: 0 <= x ==>
    natceiling (x + real a) = natceiling x + a
  apply (unfold natceiling-def)
  apply (subgoal-tac real a = real (int a))
  apply (erule ssubst)
  apply (simp del: real-of-int-of-nat-eq)
  apply (subst nat-add-distrib)
  apply (subgoal-tac 0 = ceiling 0)
  apply (erule ssubst)
  apply (erule ceiling-mono2)
  apply simp-all
done

lemma natceiling-add-number-of [simp]:
    ~ neg ((number-of n)::int) ==> 0 <= x ==>
      natceiling (x + number-of n) = natceiling x + number-of n
  apply (subst natceiling-add [THEN sym])
  apply simp-all
done

lemma natceiling-add-one: 0 <= x ==> natceiling(x + 1) = natceiling x + 1
  apply (subst natceiling-add [THEN sym])
  apply assumption
  apply simp
done

lemma natceiling-subtract [simp]: real a <= x ==>
    natceiling(x - real a) = natceiling x - a
  apply (unfold natceiling-def)
  apply (subgoal-tac real a = real (int a))
  apply (erule ssubst)
  apply (simp del: real-of-int-of-nat-eq)
  apply simp
done

lemma natfloor-div-nat: 1 <= x ==> y > 0 ==>
  natfloor (x / real y) = natfloor x div y
```

proof −
assume *1 <= (x::real)* **and** *(y::nat) > 0*
have *natfloor x = (natfloor x) div y * y + (natfloor x) mod y*
by *simp*
then have *a*: *real(natfloor x) = real ((natfloor x) div y) * real y +*
real((natfloor x) mod y)
by (*simp only*: *real-of-nat-add* [*THEN sym*] *real-of-nat-mult* [*THEN sym*])
have *x = real(natfloor x) + (x − real(natfloor x))*
by *simp*
then have *x = real ((natfloor x) div y) * real y +*
real((natfloor x) mod y) + (x − real(natfloor x))
by (*simp add*: *a*)
then have *x / real y = ... / real y*
by *simp*
also have *... = real((natfloor x) div y) + real((natfloor x) mod y) /*
real y + (x − real(natfloor x)) / real y
by (*auto simp add*: *ring-simps add-divide-distrib*
diff-divide-distrib prems)
finally have *natfloor (x / real y) = natfloor(...)* **by** *simp*
also have *... = natfloor(real((natfloor x) mod y) /*
real y + (x − real(natfloor x)) / real y + real((natfloor x) div y))
by (*simp add*: *add-ac*)
also have *... = natfloor(real((natfloor x) mod y) /*
real y + (x − real(natfloor x)) / real y) + (natfloor x) div y
apply (*rule natfloor-add*)
apply (*rule add-nonneg-nonneg*)
apply (*rule divide-nonneg-pos*)
apply *simp*
apply (*simp add*: *prems*)
apply (*rule divide-nonneg-pos*)
apply (*simp add*: *compare-rls*)
apply (*rule real-natfloor-le*)
apply (*insert prems*, *auto*)
done
also have *natfloor(real((natfloor x) mod y) /*
real y + (x − real(natfloor x)) / real y) = 0
apply (*rule natfloor-eq*)
apply *simp*
apply (*rule add-nonneg-nonneg*)
apply (*rule divide-nonneg-pos*)
apply *force*
apply (*force simp add*: *prems*)
apply (*rule divide-nonneg-pos*)
apply (*simp add*: *compare-rls*)
apply (*rule real-natfloor-le*)
apply (*auto simp add*: *prems*)
apply (*insert prems*, *arith*)
apply (*simp add*: *add-divide-distrib* [*THEN sym*])
apply (*subgoal-tac real y = real y − 1 + 1*)

apply (*erule ssubst*)
apply (*rule add-le-less-mono*)
apply (*simp add*: *compare-rls*)
apply (*subgoal-tac real*(*natfloor x mod y*) + *1* =
real(*natfloor x mod y* + *1*))
apply (*erule ssubst*)
apply (*subst real-of-nat-le-iff*)
apply (*subgoal-tac natfloor x mod y* < *y*)
apply *arith*
apply (*rule mod-less-divisor*)
apply *auto*
apply (*simp add*: *compare-rls*)
apply (*subst add-commute*)
apply (*rule real-natfloor-add-one-gt*)
done
finally show *?thesis* **by** *simp*
qed

end

8 ContNotDenum: Non-denumerability of the Continuum.

theory *ContNotDenum*
imports *RComplete*
begin

8.1 Abstract

The following document presents a proof that the Continuum is uncountable. It is formalised in the Isabelle/Isar theorem proving system.

Theorem: The Continuum $\mathbb{R}$ is not denumerable. In other words, there does not exist a function f:$\mathbb{N}\Rightarrow\mathbb{R}$ such that f is surjective.

Outline: An elegant informal proof of this result uses Cantor's Diagonalisation argument. The proof presented here is not this one. First we formalise some properties of closed intervals, then we prove the Nested Interval Property. This property relies on the completeness of the Real numbers and is the foundation for our argument. Informally it states that an intersection of countable closed intervals (where each successive interval is a subset of the last) is non-empty. We then assume a surjective function f:$\mathbb{N}\Rightarrow\mathbb{R}$ exists and find a real x such that x is not in the range of f by generating a sequence of closed intervals then using the NIP.

8.2 Closed Intervals

This section formalises some properties of closed intervals.

8.2.1 Definition

definition
closed-int :: *real* ⇒ *real* ⇒ *real set* **where**
closed-int x y = {*z*. *x* ≤ *z* ∧ *z* ≤ *y*}

8.2.2 Properties

lemma *closed-int-subset*:
assumes *xy*: *x1* ≥ *x0 y1* ≤ *y0*
shows *closed-int x1 y1* ⊆ *closed-int x0 y0*
proof −
{
fix *x*::*real*
assume *x* ∈ *closed-int x1 y1*
hence *x* ≥ *x1* ∧ *x* ≤ *y1* **by** (*simp add*: *closed-int-def*)
with *xy* **have** *x* ≥ *x0* ∧ *x* ≤ *y0* **by** *auto*
hence *x* ∈ *closed-int x0 y0* **by** (*simp add*: *closed-int-def*)
}
thus *?thesis* **by** *auto*
qed

lemma *closed-int-least*:
assumes *a*: *a* ≤ *b*
shows *a* ∈ *closed-int a b* ∧ (∀ *x* ∈ *closed-int a b*. *a* ≤ *x*)
proof
from *a* **have** *a*∈{*x*. *a*≤*x* ∧ *x*≤*b*} **by** *simp*
thus *a* ∈ *closed-int a b* **by** (*unfold closed-int-def*)
next
have ∀ *x*∈{*x*. *a*≤*x* ∧ *x*≤*b*}. *a*≤*x* **by** *simp*
thus ∀ *x* ∈ *closed-int a b*. *a* ≤ *x* **by** (*unfold closed-int-def*)
qed

lemma *closed-int-most*:
assumes *a*: *a* ≤ *b*
shows *b* ∈ *closed-int a b* ∧ (∀ *x* ∈ *closed-int a b*. *x* ≤ *b*)
proof
from *a* **have** *b*∈{*x*. *a*≤*x* ∧ *x*≤*b*} **by** *simp*
thus *b* ∈ *closed-int a b* **by** (*unfold closed-int-def*)
next
have ∀ *x*∈{*x*. *a*≤*x* ∧ *x*≤*b*}. *x*≤*b* **by** *simp*
thus ∀ *x* ∈ *closed-int a b*. *x*≤*b* **by** (*unfold closed-int-def*)
qed

lemma *closed-not-empty*:

shows $a \leq b \Longrightarrow \exists\, x.\ x \in$ *closed-int a b*
by (*auto dest*: *closed-int-least*)

lemma *closed-mem*:
assumes $a \leq c$ **and** $c \leq b$
shows $c \in$ *closed-int a b*
using *assms* **unfolding** *closed-int-def* **by** *auto*

lemma *closed-subset*:
assumes *ac*: $a \leq b \;\; c \leq d$
assumes *closed*: *closed-int a b* $\subseteq$ *closed-int c d*
shows $b \geq c$
proof −
from *closed* **have** $\forall\, x{\in}$*closed-int a b*. $x{\in}$*closed-int c d* **by** *auto*
hence $\forall\, x.\ a{\leq}x \wedge x{\leq}b \longrightarrow c{\leq}x \wedge x{\leq}d$ **by** (*unfold closed-int-def*, *auto*)
with *ac* **have** $c{\leq}b \wedge b{\leq}d$ **by** *simp*
thus *?thesis* **by** *auto*
qed

8.3 Nested Interval Property

theorem *NIP*:
fixes *f*::*nat* ⇒ *real set*
assumes *subset*: $\forall\, n.\ f\ (Suc\ n) \subseteq f\ n$
and *closed*: $\forall\, n.\ \exists\, a\ b.\ f\ n =$ *closed-int a b* $\wedge\ a \leq b$
shows $(\bigcap n.\ f\ n) \neq \{\}$
proof −
let *?g* = $\lambda n.$ (*SOME c*. $c{\in}(f\ n) \wedge (\forall\, x{\in}(f\ n).\ c \leq x)$)
have *ne*: $\forall\, n.\ \exists\, x.\ x{\in}(f\ n)$
proof
fix *n*
from *closed* **have** $\exists\, a\ b.\ f\ n =$ *closed-int a b* $\wedge\ a \leq b$ **by** *simp*
then obtain *a* **and** *b* **where** *fn*: $f\ n =$ *closed-int a b* $\wedge\ a \leq b$ **by** *auto*
hence $a \leq b$ **..**
with *closed-not-empty* **have** $\exists\, x.\ x{\in}$*closed-int a b* **by** *simp*
with *fn* **show** $\exists\, x.\ x{\in}(f\ n)$ **by** *simp*
qed

have *gdef*: $\forall\, n.\ (?g\ n){\in}(f\ n) \wedge (\forall\, x{\in}(f\ n).\ (?g\ n){\leq}x)$
proof
fix *n*
from *closed* **have** $\exists\, a\ b.\ f\ n =$ *closed-int a b* $\wedge\ a \leq b$ **..**
then obtain *a* **and** *b* **where** *ff*: $f\ n =$ *closed-int a b* **and** $a \leq b$ **by** *auto*
hence $a \leq b$ **by** *simp*
hence $a{\in}$*closed-int a b* $\wedge$ $(\forall\, x{\in}$*closed-int a b*. $a \leq x)$ **by** (*rule closed-int-least*)
with *ff* **have** $a{\in}(f\ n) \wedge (\forall\, x{\in}(f\ n).\ a \leq x)$ **by** *simp*
hence $\exists\, c.\ c{\in}(f\ n) \wedge (\forall\, x{\in}(f\ n).\ c \leq x)$ **..**
thus $(?g\ n){\in}(f\ n) \wedge (\forall\, x{\in}(f\ n).\ (?g\ n){\leq}x)$ **by** (*rule someI-ex*)
qed

— A denotes the set of all left-most points of all the intervals ...
moreover obtain *A* **where** *Adef*: $A = ?g\ `\ \mathbb{N}$ **by** *simp*
ultimately have $\exists x.\ x{\in}A$
proof −
have $(0{::}nat) \in \mathbb{N}$ **by** *simp*
moreover have $?g\ 0 = ?g\ 0$ **by** *simp*
ultimately have $?g\ 0 \in ?g\ `\ \mathbb{N}$ **by** (*rule rev-image-eqI*)
with *Adef* **have** $?g\ 0 \in A$ **by** *simp*
thus *?thesis* **..**
qed

— Now show that A is bounded above ...
moreover have $\exists y.\ isUb\ (UNIV{::}real\ set)\ A\ y$
proof −
{
fix *n*
from *ne* **have** *ex*: $\exists x.\ x{\in}(f\ n)$ **..**
from *gdef* **have** $(?g\ n){\in}(f\ n) \wedge (\forall x{\in}(f\ n).\ (?g\ n){\leq}x)$ **by** *simp*
moreover
from *closed* **have** $\exists a\ b.\ f\ n = closed\text{-}int\ a\ b \wedge a \leq b$ **..**
then obtain *a* **and** *b* **where** $f\ n = closed\text{-}int\ a\ b \wedge a \leq b$ **by** *auto*
hence $b{\in}(f\ n) \wedge (\forall x{\in}(f\ n).\ x \leq b)$ **using** *closed-int-most* **by** *blast*
ultimately have $\forall x{\in}(f\ n).\ (?g\ n) \leq b$ **by** *simp*
with *ex* **have** $(?g\ n) \leq b$ **by** *auto*
hence $\exists b.\ (?g\ n) \leq b$ **by** *auto*
}
hence *aux*: $\forall n.\ \exists b.\ (?g\ n) \leq b$ **..**

have *fs*: $\forall n{::}nat.\ f\ n \subseteq f\ 0$
proof (*rule allI*, *induct-tac n*)
show $f\ 0 \subseteq f\ 0$ **by** *simp*
next
fix *n*
assume $f\ n \subseteq f\ 0$
moreover from *subset* **have** $f\ (Suc\ n) \subseteq f\ n$ **..**
ultimately show $f\ (Suc\ n) \subseteq f\ 0$ **by** *simp*
qed
have $\forall n.\ (?g\ n){\in}(f\ 0)$
proof
fix *n*
from *gdef* **have** $(?g\ n){\in}(f\ n) \wedge (\forall x{\in}(f\ n).\ (?g\ n){\leq}x)$ **by** *simp*
hence $?g\ n \in f\ n$ **..**
with *fs* **show** $?g\ n \in f\ 0$ **by** *auto*
qed
moreover from *closed*
obtain *a* **and** *b* **where** $f\ 0 = closed\text{-}int\ a\ b$ **and** *alb*: $a \leq b$ **by** *blast*
ultimately have $\forall n.\ ?g\ n \in closed\text{-}int\ a\ b$ **by** *auto*
with *alb* **have** $\forall n.\ ?g\ n \leq b$ **using** *closed-int-most* **by** *blast*

```
  with Adef have ∀ y∈A. y≤b by auto
  hence A *<= b by (unfold setle-def)
  moreover have b ∈ (UNIV::real set) by simp
  ultimately have A *<= b ∧ b ∈ (UNIV::real set) by simp
  hence isUb (UNIV::real set) A b by (unfold isUb-def)
  thus ?thesis by auto
qed
— by the Axiom Of Completeness, A has a least upper bound ...
ultimately have ∃ t. isLub UNIV A t by (rule reals-complete)

— denote this least upper bound as t ...
then obtain t where tdef: isLub UNIV A t ..

— and finally show that this least upper bound is in all the intervals...
have ∀ n. t ∈ f n
proof
  fix n::nat
  from closed obtain a and b where
    int: f n = closed-int a b and alb: a ≤ b by blast

  have t ≥ a
  proof −
    have a ∈ A
    proof −

      from alb int have ain: a∈f n ∧ (∀ x∈f n. a ≤ x)
        using closed-int-least by blast
      moreover have ∀ e. e∈f n ∧ (∀ x∈f n. e ≤ x) ⟶ e = a
      proof clarsimp
        fix e
        assume ein: e ∈ f n and lt: ∀ x∈f n. e ≤ x
        from lt ain have aux: ∀ x∈f n. a ≤ x ∧ e ≤ x by auto

        from ein aux have a ≤ e ∧ e ≤ e by auto
        moreover from ain aux have a ≤ a ∧ e ≤ a by auto
        ultimately show e = a by simp
      qed
      hence ⋀e.  e∈f n ∧ (∀ x∈f n. e ≤ x) ⟹ e = a by simp
      ultimately have (?g n) = a by (rule some-equality)
      moreover
      {
        have n = of-nat n by simp
        moreover have of-nat n ∈ ℕ by simp
        ultimately have n ∈ ℕ
          apply −
          apply (subst(asm) eq-sym-conv)
          apply (erule subst)
          .
      }
```

```
    with Adef have (?g n) ∈ A by auto
    ultimately show ?thesis by simp
  qed
  with tdef show a ≤ t by (rule isLubD2)
qed
moreover have t ≤ b
proof −
  have isUb UNIV A b
  proof −
    {
      from alb int have
        ain: b∈f n ∧ (∀ x∈f n. x ≤ b) using closed-int-most by blast

      have subsetd: ∀ m. ∀ n. f (n + m) ⊆ f n
      proof (rule allI, induct-tac m)
        show ∀ n. f (n + 0) ⊆ f n by simp
      next
        fix m n
        assume pp: ∀ p. f (p + n) ⊆ f p
        {
          fix p
          from pp have f (p + n) ⊆ f p by simp
          moreover from subset have f (Suc (p + n)) ⊆ f (p + n) by auto
          hence f (p + (Suc n)) ⊆ f (p + n) by simp
          ultimately have f (p + (Suc n)) ⊆ f p by simp
        }
        thus ∀ p. f (p + Suc n) ⊆ f p ..
      qed
      have subsetm: ∀ α β. α ≥ β ⟶ (f α) ⊆ (f β)
      proof ((rule allI)+, rule impI)
        fix α::nat and β::nat
        assume β ≤ α
        hence ∃ k. α = β + k by (simp only: le-iff-add)
        then obtain k where α = β + k ..
        moreover
        from subsetd have f (β + k) ⊆ f β by simp
        ultimately show f α ⊆ f β by auto
      qed

      fix m
      {
        assume m ≥ n
        with subsetm have f m ⊆ f n by simp
        with ain have ∀ x∈f m. x ≤ b by auto
        moreover
        from gdef have ?g m ∈ f m ∧ (∀ x∈f m. ?g m ≤ x) by simp
        ultimately have ?g m ≤ b by auto
      }
      moreover
```

{
assume $\neg(m \geq n)$
hence $m < n$ **by** *simp*
with *subsetm* **have** *sub*: $(f\ n) \subseteq (f\ m)$ **by** *simp*
from *closed* **obtain** *ma* **and** *mb* **where**
$f\ m = closed\text{-}int\ ma\ mb \wedge ma \leq mb$ **by** *blast*
hence *one*: $ma \leq mb$ **and** *fm*: $f\ m = closed\text{-}int\ ma\ mb$ **by** *auto*
from *one alb sub fm int* **have** $ma \leq b$ **using** *closed-subset* **by** *blast*
moreover have $(?g\ m) = ma$
proof −
from *gdef* **have** $?g\ m \in f\ m \wedge (\forall x{\in}f\ m.\ ?g\ m \leq x)$ **..**
moreover from *one* **have**
$ma \in closed\text{-}int\ ma\ mb \wedge (\forall x{\in}closed\text{-}int\ ma\ mb.\ ma \leq x)$
by (*rule closed-int-least*)
with *fm* **have** $ma{\in}f\ m \wedge (\forall x{\in}f\ m.\ ma \leq x)$ **by** *simp*
ultimately have $ma \leq ?g\ m \wedge ?g\ m \leq ma$ **by** *auto*
thus $?g\ m = ma$ **by** *auto*
qed
ultimately have $?g\ m \leq b$ **by** *simp*
}
ultimately have $?g\ m \leq b$ **by** (*rule case-split*)
}
with *Adef* **have** $\forall y{\in}A.\ y{\leq}b$ **by** *auto*
hence $A\ *<=\ b$ **by** (*unfold setle-def*)
moreover have $b \in (UNIV{::}real\ set)$ **by** *simp*
ultimately have $A\ *<=\ b \wedge b \in (UNIV{::}real\ set)$ **by** *simp*
thus $isUb\ (UNIV{::}real\ set)\ A\ b$ **by** (*unfold isUb-def*)
qed
with *tdef* **show** $t \leq b$ **by** (*rule isLub-le-isUb*)
qed
ultimately have $t \in closed\text{-}int\ a\ b$ **by** (*rule closed-mem*)
with *int* **show** $t \in f\ n$ **by** *simp*
qed
hence $t \in (\bigcap n.\ f\ n)$ **by** *auto*
thus *?thesis* **by** *auto*
qed

8.4 Generating the intervals

8.4.1 Existence of non-singleton closed intervals

This lemma asserts that given any non-singleton closed interval (a,b) and any element c, there exists a closed interval that is a subset of (a,b) and that does not contain c and is a non-singleton itself.

lemma *closed-subset-ex*:
fixes *c*::*real*
assumes *alb*: $a < b$
shows
$\exists ka\ kb.\ ka < kb \wedge closed\text{-}int\ ka\ kb \subseteq closed\text{-}int\ a\ b \wedge c \notin (closed\text{-}int\ ka\ kb)$

proof −
 {
  assume *clb*: $c < b$
  {
   assume *cla*: $c < a$
   from *alb cla clb* **have** $c \notin$ *closed-int a b* **by** (*unfold closed-int-def*, *auto*)
   with *alb* **have**
    $a < b \wedge$ *closed-int a b* $\subseteq$ *closed-int a b* $\wedge\ c \notin$ *closed-int a b*
    by *auto*
   hence
    $\exists$ *ka kb*. *ka* $<$ *kb* $\wedge$ *closed-int ka kb* $\subseteq$ *closed-int a b* $\wedge\ c \notin$ (*closed-int ka kb*)
    by *auto*
  }
  moreover
  {
   assume *ncla*: $\neg(c < a)$
   with *clb* **have** *cdef*: $a \leq c \wedge c < b$ **by** *simp*
   obtain *ka* **where** *kadef*: $ka = (c + b)/2$ **by** *blast*

   from *kadef clb* **have** *kalb*: $ka < b$ **by** *auto*
   moreover from *kadef cdef* **have** *kagc*: $ka > c$ **by** *simp*
   ultimately have $c \notin$(*closed-int ka b*) **by** (*unfold closed-int-def*, *auto*)
   moreover from *cdef kagc* **have** $ka \geq a$ **by** *simp*
   hence *closed-int ka b* $\subseteq$ *closed-int a b* **by** (*unfold closed-int-def*, *auto*)
   ultimately have
    ka $< b\ \wedge$ *closed-int ka b* $\subseteq$ *closed-int a b* $\wedge\ c \notin$ *closed-int ka b*
    using *kalb* **by** *auto*
   hence
    $\exists$ *ka kb*. *ka* $<$ *kb* $\wedge$ *closed-int ka kb* $\subseteq$ *closed-int a b* $\wedge\ c \notin$ (*closed-int ka kb*)
    by *auto*

  }
  ultimately have
   $\exists$ *ka kb*. *ka* $<$ *kb* $\wedge$ *closed-int ka kb* $\subseteq$ *closed-int a b* $\wedge\ c \notin$ (*closed-int ka kb*)
   by (*rule case-split*)
 }
 moreover
 {
  assume $\neg\ (c < b)$
  hence *cgeb*: $c \geq b$ **by** *simp*

  obtain *kb* **where** *kbdef*: $kb = (a + b)/2$ **by** *blast*
  with *alb* **have** *kblb*: $kb < b$ **by** *auto*
  with *kbdef cgeb* **have** $a < kb \wedge kb < c$ **by** *auto*
  moreover hence $c \notin$ (*closed-int a kb*) **by** (*unfold closed-int-def*, *auto*)
  moreover from *kblb* **have**
   closed-int a kb $\subseteq$ *closed-int a b* **by** (*unfold closed-int-def*, *auto*)
  ultimately have
   $a < kb \wedge$ *closed-int a kb* $\subseteq$ *closed-int a b* $\wedge\ c \notin$*closed-int a kb*

```
    by simp
  hence
    ∃ka kb. ka < kb ∧ closed-int ka kb ⊆ closed-int a b ∧ c ∉ (closed-int ka kb)
    by auto
 }
 ultimately show ?thesis by (rule case-split)
qed
```

8.5 newInt: Interval generation

Given a function f:ℕ⇒ℝ, newInt (Suc n) f returns a closed interval such that *newInt (Suc n) f* ⊆ *newInt n f* and does not contain *f (Suc n)*. With the base case defined such that *(f 0)*∉*newInt 0 f*.

8.5.1 Definition

```
consts newInt :: nat ⇒ (nat ⇒ real) ⇒ (real set)
primrec
newInt 0 f = closed-int (f 0 + 1) (f 0 + 2)
newInt (Suc n) f =
 (SOME e. (∃ e1 e2.
  e1 < e2 ∧
  e = closed-int e1 e2 ∧
  e ⊆ (newInt n f) ∧
  (f (Suc n)) ∉ e)
 )
```

8.5.2 Properties

We now show that every application of newInt returns an appropriate interval.

```
lemma newInt-ex:
 ∃ a b. a < b ∧
  newInt (Suc n) f = closed-int a b ∧
  newInt (Suc n) f ⊆ newInt n f ∧
  f (Suc n) ∉ newInt (Suc n) f
proof (induct n)
 case 0

 let ?e = SOME e. ∃ e1 e2.
  e1 < e2 ∧
  e = closed-int e1 e2 ∧
  e ⊆ closed-int (f 0 + 1) (f 0 + 2) ∧
  f (Suc 0) ∉ e

 have newInt (Suc 0) f = ?e by auto
 moreover
 have f 0 + 1 < f 0 + 2 by simp
```

```
  with closed-subset-ex have
   ∃ka kb. ka < kb ∧ closed-int ka kb ⊆ closed-int (f 0 + 1) (f 0 + 2) ∧
    f (Suc 0) ∉ (closed-int ka kb) .
  hence
   ∃ e. ∃ ka kb. ka < kb ∧ e = closed-int ka kb ∧
    e ⊆ closed-int (f 0 + 1) (f 0 + 2) ∧ f (Suc 0) ∉ e by simp
  hence
   ∃ ka kb. ka < kb ∧ ?e = closed-int ka kb ∧
    ?e ⊆ closed-int (f 0 + 1) (f 0 + 2) ∧ f (Suc 0) ∉ ?e
   by (rule someI-ex)
  ultimately have ∃ e1 e2. e1 < e2 ∧
   newInt (Suc 0) f = closed-int e1 e2 ∧
   newInt (Suc 0) f ⊆ closed-int (f 0 + 1) (f 0 + 2) ∧
   f (Suc 0) ∉ newInt (Suc 0) f by simp
  thus
   ∃ a b. a < b ∧ newInt (Suc 0) f = closed-int a b ∧
    newInt (Suc 0) f ⊆ newInt 0 f ∧ f (Suc 0) ∉ newInt (Suc 0) f
   by simp
next
  case (Suc n)
  hence ∃ a b.
   a < b ∧
   newInt (Suc n) f = closed-int a b ∧
   newInt (Suc n) f ⊆ newInt n f ∧
   f (Suc n) ∉ newInt (Suc n) f by simp
  then obtain a and b where ab: a < b ∧
   newInt (Suc n) f = closed-int a b ∧
   newInt (Suc n) f ⊆ newInt n f ∧
   f (Suc n) ∉ newInt (Suc n) f by auto
  hence cab: closed-int a b = newInt (Suc n) f by simp

  let ?e = SOME e. ∃ e1 e2.
    e1 < e2 ∧
    e = closed-int e1 e2 ∧
    e ⊆ closed-int a b ∧
    f (Suc (Suc n)) ∉ e
  from cab have ni: newInt (Suc (Suc n)) f = ?e by auto

  from ab have a < b by simp
  with closed-subset-ex have
   ∃ka kb. ka < kb ∧ closed-int ka kb ⊆ closed-int a b ∧
    f (Suc (Suc n)) ∉ closed-int ka kb .
  hence
   ∃ e. ∃ ka kb. ka < kb ∧ e = closed-int ka kb ∧
    closed-int ka kb ⊆ closed-int a b ∧ f (Suc (Suc n)) ∉ closed-int ka kb
   by simp
  hence
   ∃ e.  ∃ ka kb. ka < kb ∧ e = closed-int ka kb ∧
    e ⊆ closed-int a b ∧ f (Suc (Suc n)) ∉ e by simp
```

```
  hence
    ∃ ka kb. ka < kb ∧ ?e = closed-int ka kb ∧
     ?e ⊆ closed-int a b ∧ f (Suc (Suc n)) ∉ ?e by (rule someI-ex)
  with ab ni show
    ∃ ka kb. ka < kb ∧
     newInt (Suc (Suc n)) f = closed-int ka kb ∧
     newInt (Suc (Suc n)) f ⊆ newInt (Suc n) f ∧
     f (Suc (Suc n)) ∉ newInt (Suc (Suc n)) f by auto
qed

lemma newInt-subset:
  newInt (Suc n) f ⊆ newInt n f
  using newInt-ex by auto
```

Another fundamental property is that no element in the range of f is in the intersection of all closed intervals generated by newInt.

```
lemma newInt-inter:
  ∀ n. f n ∉ (⋂ n. newInt n f)
proof
  fix n::nat
  {
    assume n0: n = 0
    moreover have newInt 0 f = closed-int (f 0 + 1) (f 0 + 2) by simp
    ultimately have f n ∉ newInt n f by (unfold closed-int-def, simp)
  }
  moreover
  {
    assume ¬ n = 0
    hence n > 0 by simp
    then obtain m where ndef: n = Suc m by (auto simp add: gr0-conv-Suc)

    from newInt-ex have
      ∃ a b. a < b ∧ (newInt (Suc m) f) = closed-int a b ∧
       newInt (Suc m) f ⊆ newInt m f ∧ f (Suc m) ∉ newInt (Suc m) f .
    then have f (Suc m) ∉ newInt (Suc m) f by auto
    with ndef have f n ∉ newInt n f by simp
  }
  ultimately have f n ∉ newInt n f by (rule case-split)
  thus f n ∉ (⋂ n. newInt n f) by auto
qed


lemma newInt-notempty:
  (⋂ n. newInt n f) ≠ {}
proof −
  let ?g = λn. newInt n f
  have ∀ n. ?g (Suc n) ⊆ ?g n
  proof
    fix n
```

```
    show ?g (Suc n) ⊆ ?g n by (rule newInt-subset)
  qed
  moreover have ∀ n. ∃ a b. ?g n = closed-int a b ∧ a ≤ b
  proof
    fix n::nat
    {
      assume n = 0
      then have
        ?g n = closed-int (f 0 + 1) (f 0 + 2) ∧ (f 0 + 1 ≤ f 0 + 2)
        by simp
      hence ∃ a b. ?g n = closed-int a b ∧ a ≤ b by blast
    }
    moreover
    {
      assume ¬ n = 0
      then have n > 0 by simp
      then obtain m where nd: n = Suc m by (auto simp add: gr0-conv-Suc)

      have
        ∃ a b. a < b ∧ (newInt (Suc m) f) = closed-int a b ∧
        (newInt (Suc m) f) ⊆ (newInt m f) ∧ (f (Suc m)) ∉ (newInt (Suc m) f)
        by (rule newInt-ex)
      then obtain a and b where
        a < b ∧ (newInt (Suc m) f) = closed-int a b by auto
      with nd have ?g n = closed-int a b ∧ a ≤ b by auto
      hence ∃ a b. ?g n = closed-int a b ∧ a ≤ b by blast
    }
    ultimately show ∃ a b. ?g n = closed-int a b ∧ a ≤ b by (rule case-split)
  qed
  ultimately show ?thesis by (rule NIP)
qed
```

8.6 Final Theorem

```
theorem real-non-denum:
  shows ¬ (∃f::nat⇒real. surj f)
proof — by contradiction
  assume ∃f::nat⇒real. surj f
  then obtain f::nat⇒real where surj f by auto
  hence rangeF: range f = UNIV by (rule surj-range)
  — We now produce a real number x that is not in the range of f, using the
properties of newInt.
  have ∃x. x ∈ (⋂n. newInt n f) using newInt-notempty by blast
  moreover have ∀ n. f n ∉ (⋂n. newInt n f) by (rule newInt-inter)
  ultimately obtain x where x ∈ (⋂n. newInt n f) and ∀ n. f n ≠ x by blast
  moreover from rangeF have x ∈ range f by simp
  ultimately show False by blast
qed
```

end

9 RealPow: Natural powers theory

theory *RealPow*
imports *RealDef*
begin

declare *abs-mult-self* [*simp*]

instantiation *real* :: *recpower*
begin

primrec *power-real* **where**
  realpow-0: $r \hat{} 0 = (1::real)$
  | *realpow-Suc*: $r \hat{} Suc\ n = (r::real) * r \hat{} n$

instance proof
  fix z :: *real*
  fix n :: *nat*
  show $z\hat{}0 = 1$ **by** *simp*
  show $z\hat{}(Suc\ n) = z * (z\hat{}n)$ **by** *simp*
qed

end

lemma *two-realpow-ge-one* [*simp*]: $(1::real) \leq 2 \hat{} n$
by *simp*

lemma *two-realpow-gt* [*simp*]: *real* $(n::nat) < 2 \hat{} n$
apply (*induct n*)
apply (*auto simp add*: *real-of-nat-Suc*)
apply (*subst mult-2*)
apply (*rule add-less-le-mono*)
apply (*auto simp add*: *two-realpow-ge-one*)
done

lemma *realpow-Suc-le-self*: $[|\ 0 \leq r;\ r \leq (1::real)\ |] ==> r \hat{} Suc\ n \leq r$
by (*insert power-decreasing* [*of 1 Suc n r*], *simp*)

lemma *realpow-minus-mult* [*rule-format*]:
    $0 < n --> (x::real) \hat{} (n - 1) * x = x \hat{} n$
apply (*simp split add*: *nat-diff-split*)
done

lemma *realpow-two-mult-inverse* [*simp*]:
    $r \neq 0 ==> r * inverse\ r \hat{}Suc\ (Suc\ 0) = inverse\ (r::real)$

by (*simp add*: *real-mult-assoc* [*symmetric*])

lemma *realpow-two-minus* [*simp*]: $(-x)\hat{}Suc\ (Suc\ 0) = (x::real)\hat{}Suc\ (Suc\ 0)$
by *simp*

lemma *realpow-two-diff*:
$(x::real)\hat{}Suc\ (Suc\ 0) - y\hat{}Suc\ (Suc\ 0) = (x - y) * (x + y)$
apply (*unfold real-diff-def*)
apply (*simp add*: *ring-simps*)
done

lemma *realpow-two-disj*:
$((x::real)\hat{}Suc\ (Suc\ 0) = y\hat{}Suc\ (Suc\ 0)) = (x = y \mid x = -y)$
apply (*cut-tac* $x = x$ **and** $y = y$ **in** *realpow-two-diff*)
apply (*auto simp del*: *realpow-Suc*)
done

lemma *realpow-real-of-nat*: $real\ (m::nat)\ \hat{}\ n = real\ (m\ \hat{}\ n)$
apply (*induct n*)
apply (*auto simp add*: *real-of-nat-one real-of-nat-mult*)
done

lemma *realpow-real-of-nat-two-pos* [*simp*] : $0 < real\ (Suc\ (Suc\ 0)\ \hat{}\ n)$
apply (*induct n*)
apply (*auto simp add*: *real-of-nat-mult zero-less-mult-iff*)
done

lemma *realpow-increasing*:
$[|(0::real) \leq x;\ 0 \leq y;\ x\ \hat{}\ Suc\ n \leq y\ \hat{}\ Suc\ n|] ==> x \leq y$
by (*rule power-le-imp-le-base*)

9.1 Literal Arithmetic Involving Powers, Type *real*

lemma *real-of-int-power*: $real\ (x::int)\ \hat{}\ n = real\ (x\ \hat{}\ n)$
apply (*induct n*)
apply (*simp-all add*: *nat-mult-distrib*)
done
declare *real-of-int-power* [*symmetric*, *simp*]

lemma *power-real-number-of*:
$(number\text{-}of\ v :: real)\ \hat{}\ n = real\ ((number\text{-}of\ v :: int)\ \hat{}\ n)$
by (*simp only*: *real-number-of* [*symmetric*] *real-of-int-power*)

declare *power-real-number-of* [*of - number-of w*, *standard*, *simp*]

9.2 Properties of Squares

lemma *sum-squares-ge-zero*:
fixes $x\ y :: 'a::ordered\text{-}ring\text{-}strict$

shows $0 \leq x * x + y * y$
by (*intro add-nonneg-nonneg zero-le-square*)

lemma *not-sum-squares-lt-zero*:
fixes x y :: $'a$::*ordered-ring-strict*
shows $\neg\ x * x + y * y < 0$
by (*simp add*: *linorder-not-less sum-squares-ge-zero*)

lemma *sum-nonneg-eq-zero-iff*:
fixes x y :: $'a$::*pordered-ab-group-add*
assumes *x*: $0 \leq x$ **and** *y*: $0 \leq y$
shows $(x + y = 0) = (x = 0 \wedge y = 0)$
proof (*auto*)
from y **have** $x + 0 \leq x + y$ **by** (*rule add-left-mono*)
also assume $x + y = 0$
finally have $x \leq 0$ **by** *simp*
thus $x = 0$ **using** x **by** (*rule order-antisym*)
next
from x **have** $0 + y \leq x + y$ **by** (*rule add-right-mono*)
also assume $x + y = 0$
finally have $y \leq 0$ **by** *simp*
thus $y = 0$ **using** y **by** (*rule order-antisym*)
qed

lemma *sum-squares-eq-zero-iff*:
fixes x y :: $'a$::*ordered-ring-strict*
shows $(x * x + y * y = 0) = (x = 0 \wedge y = 0)$
by (*simp add*: *sum-nonneg-eq-zero-iff*)

lemma *sum-squares-le-zero-iff*:
fixes x y :: $'a$::*ordered-ring-strict*
shows $(x * x + y * y \leq 0) = (x = 0 \wedge y = 0)$
by (*simp add*: *order-le-less not-sum-squares-lt-zero sum-squares-eq-zero-iff*)

lemma *sum-squares-gt-zero-iff*:
fixes x y :: $'a$::*ordered-ring-strict*
shows $(0 < x * x + y * y) = (x \neq 0 \vee y \neq 0)$
by (*simp add*: *order-less-le sum-squares-ge-zero sum-squares-eq-zero-iff*)

lemma *sum-power2-ge-zero*:
fixes x y :: $'a$::{*ordered-idom*,*recpower*}
shows $0 \leq x^2 + y^2$
unfolding *power2-eq-square* **by** (*rule sum-squares-ge-zero*)

lemma *not-sum-power2-lt-zero*:
fixes x y :: $'a$::{*ordered-idom*,*recpower*}
shows $\neg\ x^2 + y^2 < 0$
unfolding *power2-eq-square* **by** (*rule not-sum-squares-lt-zero*)

lemma *sum-power2-eq-zero-iff*:
fixes $x\ y :: 'a::\{ordered\text{-}idom,recpower\}$
shows $(x^2 + y^2 = 0) = (x = 0 \wedge y = 0)$
unfolding *power2-eq-square* **by** (*rule sum-squares-eq-zero-iff*)

lemma *sum-power2-le-zero-iff*:
fixes $x\ y :: 'a::\{ordered\text{-}idom,recpower\}$
shows $(x^2 + y^2 \leq 0) = (x = 0 \wedge y = 0)$
unfolding *power2-eq-square* **by** (*rule sum-squares-le-zero-iff*)

lemma *sum-power2-gt-zero-iff*:
fixes $x\ y :: 'a::\{ordered\text{-}idom,recpower\}$
shows $(0 < x^2 + y^2) = (x \neq 0 \vee y \neq 0)$
unfolding *power2-eq-square* **by** (*rule sum-squares-gt-zero-iff*)

9.3 Squares of Reals

lemma *real-two-squares-add-zero-iff* [*simp*]:
$(x * x + y * y = 0) = ((x::real) = 0 \wedge y = 0)$
by (*rule sum-squares-eq-zero-iff*)

lemma *real-sum-squares-cancel*: $x * x + y * y = 0 ==> x = (0::real)$
by *simp*

lemma *real-sum-squares-cancel2*: $x * x + y * y = 0 ==> y = (0::real)$
by *simp*

lemma *real-mult-self-sum-ge-zero*: $(0::real) \leq x*x + y*y$
by (*rule sum-squares-ge-zero*)

lemma *real-sum-squares-cancel-a*: $x * x = -(y * y) ==> x = (0::real)$ & $y=0$
by (*simp add*: *real-add-eq-0-iff* [*symmetric*])

lemma *real-squared-diff-one-factored*: $x*x - (1::real) = (x + 1)*(x - 1)$
by (*simp add*: *left-distrib right-diff-distrib*)

lemma *real-mult-is-one* [*simp*]: $(x*x = (1::real)) = (x = 1 \mid x = -1)$
apply *auto*
apply (*drule right-minus-eq* [*THEN iffD2*])
apply (*auto simp add*: *real-squared-diff-one-factored*)
done

lemma *real-sum-squares-not-zero*: $x \sim= 0 ==> x * x + y * y \sim= (0::real)$
by *simp*

lemma *real-sum-squares-not-zero2*: $y \sim= 0 ==> x * x + y * y \sim= (0::real)$
by *simp*

lemma *realpow-two-sum-zero-iff* [*simp*]:

$(x \hat{} 2 + y \hat{} 2 = (0::real)) = (x = 0 \ \& \ y = 0)$
by (*rule sum-power2-eq-zero-iff*)

lemma *realpow-two-le-add-order* [*simp*]: $(0::real) \leq u \hat{} 2 + v \hat{} 2$
by (*rule sum-power2-ge-zero*)

lemma *realpow-two-le-add-order2* [*simp*]: $(0::real) \leq u \hat{} 2 + v \hat{} 2 + w \hat{} 2$
by (*intro add-nonneg-nonneg zero-le-power2*)

lemma *real-sum-square-gt-zero*: $x \sim= 0 \Longrightarrow (0::real) < x * x + y * y$
by (*simp add*: *sum-squares-gt-zero-iff*)

lemma *real-sum-square-gt-zero2*: $y \sim= 0 \Longrightarrow (0::real) < x * x + y * y$
by (*simp add*: *sum-squares-gt-zero-iff*)

lemma *real-minus-mult-self-le* [*simp*]: $-(u * u) \leq (x * (x::real))$
by (*rule-tac j = 0* **in** *real-le-trans*, *auto*)

lemma *realpow-square-minus-le* [*simp*]: $-(u \hat{} 2) \leq (x::real) \hat{} 2$
by (*auto simp add*: *power2-eq-square*)

lemma *real-sq-order*:
fixes *x*::*real*
assumes *xgt0*: $0 \leq x$ **and** *ygt0*: $0 \leq y$ **and** *sq*: $x\hat{}2 \leq y\hat{}2$
shows $x \leq y$
proof −
from *sq* **have** $x \hat{} Suc\ (Suc\ 0) \leq y \hat{} Suc\ (Suc\ 0)$
by (*simp only*: *numeral-2-eq-2*)
thus $x \leq y$ **using** *ygt0*
by (*rule power-le-imp-le-base*)
qed

9.4 Various Other Theorems

lemma *real-le-add-half-cancel*: $(x + y/2 \leq (y::real)) = (x \leq y\ /2)$
by *auto*

lemma *real-minus-half-eq* [*simp*]: $(x::real) - x/2 = x/2$
by *auto*

lemma *real-mult-inverse-cancel*:
$[|(0::real) < x;\ 0 < x1;\ x1 * y < x * u\ |]$
$\Longrightarrow inverse\ x * y < inverse\ x1 * u$
apply (*rule-tac c=x* **in** *mult-less-imp-less-left*)
apply (*auto simp add*: *real-mult-assoc* [*symmetric*])
apply (*simp* (*no-asm*) *add*: *mult-ac*)
apply (*rule-tac c=x1* **in** *mult-less-imp-less-right*)
apply (*auto simp add*: *mult-ac*)

done

lemma *real-mult-inverse-cancel2*:
[|(*0*::*real*) < *x*;*0* < *x1*; *x1* * *y* < *x* * *u* |] ==> *y* * *inverse x* < *u* * *inverse x1*
apply (*auto dest*: *real-mult-inverse-cancel simp add*: *mult-ac*)
done

lemma *inverse-real-of-nat-gt-zero* [*simp*]: *0* < *inverse* (*real* (*Suc n*))
by *simp*

lemma *inverse-real-of-nat-ge-zero* [*simp*]: *0* ≤ *inverse* (*real* (*Suc n*))
by *simp*

lemma *realpow-num-eq-if*: (*m*::*real*) ^ *n* = (*if n=0 then 1 else m* * *m* ^ (*n* − *1*))
by (*case-tac n*, *auto*)

end

10 RealVector: Vector Spaces and Algebras over the Reals

theory *RealVector*
imports *RealPow*
begin

10.1 Locale for additive functions

locale *additive* =
fixes *f* :: 'a::*ab-group-add* ⇒ 'b::*ab-group-add*
assumes *add*: *f* (*x* + *y*) = *f x* + *f y*

lemma (**in** *additive*) *zero*: *f 0* = *0*
proof −
have *f 0* = *f* (*0* + *0*) **by** *simp*
also have ... = *f 0* + *f 0* **by** (*rule add*)
finally show *f 0* = *0* **by** *simp*
qed

lemma (**in** *additive*) *minus*: *f* (− *x*) = − *f x*
proof −
have *f* (− *x*) + *f x* = *f* (− *x* + *x*) **by** (*rule add* [*symmetric*])
also have ... = − *f x* + *f x* **by** (*simp add*: *zero*)
finally show *f* (− *x*) = − *f x* **by** (*rule add-right-imp-eq*)
qed

lemma (**in** *additive*) *diff*: *f* (*x* − *y*) = *f x* − *f y*
by (*simp add*: *diff-def add minus*)

lemma (**in** *additive*) *setsum*: $f\ (setsum\ g\ A) = (\sum x{\in}A.\ f\ (g\ x))$
apply (*cases finite A*)
apply (*induct set*: *finite*)
apply (*simp add*: *zero*)
apply (*simp add*: *add*)
apply (*simp add*: *zero*)
done

10.2 Real vector spaces

class *scaleR* = *type* +
  fixes *scaleR* :: *real* ⇒ *'a* ⇒ *'a* (**infixr** $*_R$ *75*)
begin

abbreviation
  divideR :: *'a* ⇒ *real* ⇒ *'a* (**infixl** $'/_R$ *70*)
where
  $x\ /_R\ r == scaleR\ (inverse\ r)\ x$

end

instantiation *real* :: *scaleR*
begin

definition
  real-scaleR-def [*simp*]: $scaleR\ a\ x = a * x$

instance ..

end

class *real-vector* = *scaleR* + *ab-group-add* +
  assumes *scaleR-right-distrib*: $scaleR\ a\ (x + y) = scaleR\ a\ x + scaleR\ a\ y$
  and *scaleR-left-distrib*: $scaleR\ (a + b)\ x = scaleR\ a\ x + scaleR\ b\ x$
  and *scaleR-scaleR* [*simp*]: $scaleR\ a\ (scaleR\ b\ x) = scaleR\ (a * b)\ x$
  and *scaleR-one* [*simp*]: $scaleR\ 1\ x = x$

class *real-algebra* = *real-vector* + *ring* +
  assumes *mult-scaleR-left* [*simp*]: $scaleR\ a\ x * y = scaleR\ a\ (x * y)$
  and *mult-scaleR-right* [*simp*]: $x * scaleR\ a\ y = scaleR\ a\ (x * y)$

class *real-algebra-1* = *real-algebra* + *ring-1*

class *real-div-algebra* = *real-algebra-1* + *division-ring*

class *real-field* = *real-div-algebra* + *field*

instance *real* :: *real-field*

apply (*intro-classes*, *unfold real-scaleR-def*)
apply (*rule right-distrib*)
apply (*rule left-distrib*)
apply (*rule mult-assoc* [*symmetric*])
apply (*rule mult-1-left*)
apply (*rule mult-assoc*)
apply (*rule mult-left-commute*)
done

lemma *scaleR-left-commute*:
fixes *x* :: *'a*::*real-vector*
shows *scaleR a* (*scaleR b x*) = *scaleR b* (*scaleR a x*)
by (*simp add*: *mult-commute*)

interpretation *scaleR-left*: *additive* [(λ*a*. *scaleR a x*::*'a*::*real-vector*)]
by *unfold-locales* (*rule scaleR-left-distrib*)

interpretation *scaleR-right*: *additive* [(λ*x*. *scaleR a x*::*'a*::*real-vector*)]
by *unfold-locales* (*rule scaleR-right-distrib*)

lemmas *scaleR-zero-left* [*simp*] = *scaleR-left.zero*

lemmas *scaleR-zero-right* [*simp*] = *scaleR-right.zero*

lemmas *scaleR-minus-left* [*simp*] = *scaleR-left.minus*

lemmas *scaleR-minus-right* [*simp*] = *scaleR-right.minus*

lemmas *scaleR-left-diff-distrib* = *scaleR-left.diff*

lemmas *scaleR-right-diff-distrib* = *scaleR-right.diff*

lemma *scaleR-eq-0-iff* [*simp*]:
fixes *x* :: *'a*::*real-vector*
shows (*scaleR a x* = *0*) = (*a* = *0* ∨ *x* = *0*)
proof *cases*
assume *a* = *0* **thus** *?thesis* **by** *simp*
next
assume *anz* [*simp*]: *a* ≠ *0*
{ assume *scaleR a x* = *0*
hence *scaleR* (*inverse a*) (*scaleR a x*) = *0* **by** *simp*
hence *x* = *0* **by** *simp* **}**
thus *?thesis* **by** *force*
qed

lemma *scaleR-left-imp-eq*:
fixes *x y* :: *'a*::*real-vector*
shows ⟦*a* ≠ *0*; *scaleR a x* = *scaleR a y*⟧ ⟹ *x* = *y*
proof −

assume *nonzero*: $a \neq 0$
assume $scaleR\ a\ x = scaleR\ a\ y$
hence $scaleR\ a\ (x - y) = 0$
by (*simp add*: *scaleR-right-diff-distrib*)
hence $x - y = 0$ **by** (*simp add*: *nonzero*)
thus $x = y$ **by** *simp*
qed

lemma *scaleR-right-imp-eq*:
fixes $x\ y :: 'a$::*real-vector*
shows $[\![x \neq 0;\ scaleR\ a\ x = scaleR\ b\ x]\!] \Longrightarrow a = b$
proof −
assume *nonzero*: $x \neq 0$
assume $scaleR\ a\ x = scaleR\ b\ x$
hence $scaleR\ (a - b)\ x = 0$
by (*simp add*: *scaleR-left-diff-distrib*)
hence $a - b = 0$ **by** (*simp add*: *nonzero*)
thus $a = b$ **by** *simp*
qed

lemma *scaleR-cancel-left*:
fixes $x\ y :: 'a$::*real-vector*
shows $(scaleR\ a\ x = scaleR\ a\ y) = (x = y \vee a = 0)$
by (*auto intro*: *scaleR-left-imp-eq*)

lemma *scaleR-cancel-right*:
fixes $x\ y :: 'a$::*real-vector*
shows $(scaleR\ a\ x = scaleR\ b\ x) = (a = b \vee x = 0)$
by (*auto intro*: *scaleR-right-imp-eq*)

lemma *nonzero-inverse-scaleR-distrib*:
fixes $x :: 'a$::*real-div-algebra* **shows**
$[\![a \neq 0;\ x \neq 0]\!] \Longrightarrow inverse\ (scaleR\ a\ x) = scaleR\ (inverse\ a)\ (inverse\ x)$
by (*rule inverse-unique*, *simp*)

lemma *inverse-scaleR-distrib*:
fixes $x :: 'a$::{*real-div-algebra*,*division-by-zero*}
shows $inverse\ (scaleR\ a\ x) = scaleR\ (inverse\ a)\ (inverse\ x)$
apply (*case-tac* $a = 0$, *simp*)
apply (*case-tac* $x = 0$, *simp*)
apply (*erule* (*1*) *nonzero-inverse-scaleR-distrib*)
done

10.3 Embedding of the Reals into any *real-algebra-1*: *of-real*

definition
$of\text{-}real :: real \Rightarrow 'a$::*real-algebra-1* **where**
$of\text{-}real\ r = scaleR\ r\ 1$

lemma *scaleR-conv-of-real*: *scaleR r x = of-real r * x*
by (*simp add*: *of-real-def*)

lemma *of-real-0* [*simp*]: *of-real 0 = 0*
by (*simp add*: *of-real-def*)

lemma *of-real-1* [*simp*]: *of-real 1 = 1*
by (*simp add*: *of-real-def*)

lemma *of-real-add* [*simp*]: *of-real (x + y) = of-real x + of-real y*
by (*simp add*: *of-real-def scaleR-left-distrib*)

lemma *of-real-minus* [*simp*]: *of-real (- x) = - of-real x*
by (*simp add*: *of-real-def*)

lemma *of-real-diff* [*simp*]: *of-real (x - y) = of-real x - of-real y*
by (*simp add*: *of-real-def scaleR-left-diff-distrib*)

lemma *of-real-mult* [*simp*]: *of-real (x * y) = of-real x * of-real y*
by (*simp add*: *of-real-def mult-commute*)

lemma *nonzero-of-real-inverse*:
x ≠ 0 ⟹ of-real (inverse x) =
inverse (of-real x :: 'a::real-div-algebra)
by (*simp add*: *of-real-def nonzero-inverse-scaleR-distrib*)

lemma *of-real-inverse* [*simp*]:
of-real (inverse x) =
inverse (of-real x :: 'a::{real-div-algebra,division-by-zero})
by (*simp add*: *of-real-def inverse-scaleR-distrib*)

lemma *nonzero-of-real-divide*:
y ≠ 0 ⟹ of-real (x / y) =
(of-real x / of-real y :: 'a::real-field)
by (*simp add*: *divide-inverse nonzero-of-real-inverse*)

lemma *of-real-divide* [*simp*]:
of-real (x / y) =
(of-real x / of-real y :: 'a::{real-field,division-by-zero})
by (*simp add*: *divide-inverse*)

lemma *of-real-power* [*simp*]:
of-real (x ^ n) = (of-real x :: 'a::{real-algebra-1,recpower}) ^ n
by (*induct n*) (*simp-all add*: *power-Suc*)

lemma *of-real-eq-iff* [*simp*]: *(of-real x = of-real y) = (x = y)*
by (*simp add*: *of-real-def scaleR-cancel-right*)

lemmas *of-real-eq-0-iff* [*simp*] = *of-real-eq-iff* [*of - 0*, *simplified*]

lemma *of-real-eq-id* [*simp*]: *of-real* = (*id* :: *real* ⇒ *real*)
proof
  fix *r*
  show *of-real r* = *id r*
    by (*simp add*: *of-real-def*)
qed

Collapse nested embeddings

lemma *of-real-of-nat-eq* [*simp*]: *of-real* (*of-nat n*) = *of-nat n*
by (*induct n*) *auto*

lemma *of-real-of-int-eq* [*simp*]: *of-real* (*of-int z*) = *of-int z*
by (*cases z rule*: *int-diff-cases*, *simp*)

lemma *of-real-number-of-eq*:
  of-real (*number-of w*) = (*number-of w* :: *'a*::{*number-ring*,*real-algebra-1*})
by (*simp add*: *number-of-eq*)

Every real algebra has characteristic zero

instance *real-algebra-1* < *ring-char-0*
proof
  fix *m n* :: *nat*
  have (*of-real* (*of-nat m*) = (*of-real* (*of-nat n*)::*'a*)) = (*m* = *n*)
    by (*simp only*: *of-real-eq-iff of-nat-eq-iff*)
  thus (*of-nat m* = (*of-nat n*::*'a*)) = (*m* = *n*)
    by (*simp only*: *of-real-of-nat-eq*)
qed

10.4 The Set of Real Numbers

definition
  Reals :: *'a*::*real-algebra-1 set* **where**
  Reals ≡ *range of-real*

notation (*xsymbols*)
  Reals (ℝ)

lemma *Reals-of-real* [*simp*]: *of-real r* ∈ *Reals*
by (*simp add*: *Reals-def*)

lemma *Reals-of-int* [*simp*]: *of-int z* ∈ *Reals*
by (*subst of-real-of-int-eq* [*symmetric*], *rule Reals-of-real*)

lemma *Reals-of-nat* [*simp*]: *of-nat n* ∈ *Reals*
by (*subst of-real-of-nat-eq* [*symmetric*], *rule Reals-of-real*)

lemma *Reals-number-of* [*simp*]:
  (*number-of w*::*'a*::{*number-ring*,*real-algebra-1*}) ∈ *Reals*

by (*subst of-real-number-of-eq* [*symmetric*], *rule Reals-of-real*)

lemma *Reals-0* [*simp*]: $0 \in Reals$
apply (*unfold Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-0* [*symmetric*])
done

lemma *Reals-1* [*simp*]: $1 \in Reals$
apply (*unfold Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-1* [*symmetric*])
done

lemma *Reals-add* [*simp*]: $[\![a \in Reals; b \in Reals]\!] \Longrightarrow a + b \in Reals$
apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-add* [*symmetric*])
done

lemma *Reals-minus* [*simp*]: $a \in Reals \Longrightarrow - a \in Reals$
apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-minus* [*symmetric*])
done

lemma *Reals-diff* [*simp*]: $[\![a \in Reals; b \in Reals]\!] \Longrightarrow a - b \in Reals$
apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-diff* [*symmetric*])
done

lemma *Reals-mult* [*simp*]: $[\![a \in Reals; b \in Reals]\!] \Longrightarrow a * b \in Reals$
apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-mult* [*symmetric*])
done

lemma *nonzero-Reals-inverse*:
fixes a :: $'a$::*real-div-algebra*
shows $[\![a \in Reals; a \neq 0]\!] \Longrightarrow inverse\ a \in Reals$
apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*erule nonzero-of-real-inverse* [*symmetric*])
done

lemma *Reals-inverse* [*simp*]:
fixes a :: $'a$::{*real-div-algebra*,*division-by-zero*}
shows $a \in Reals \Longrightarrow inverse\ a \in Reals$

apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-inverse* [*symmetric*])
done

lemma *nonzero-Reals-divide*:
fixes *a b* :: *'a*::*real-field*
shows ⟦*a* ∈ *Reals*; *b* ∈ *Reals*; *b* ≠ *0*⟧ ⟹ *a* / *b* ∈ *Reals*
apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*erule nonzero-of-real-divide* [*symmetric*])
done

lemma *Reals-divide* [*simp*]:
fixes *a b* :: *'a*::{*real-field*,*division-by-zero*}
shows ⟦*a* ∈ *Reals*; *b* ∈ *Reals*⟧ ⟹ *a* / *b* ∈ *Reals*
apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-divide* [*symmetric*])
done

lemma *Reals-power* [*simp*]:
fixes *a* :: *'a*::{*real-algebra-1*,*recpower*}
shows *a* ∈ *Reals* ⟹ *a* ^ *n* ∈ *Reals*
apply (*auto simp add*: *Reals-def*)
apply (*rule range-eqI*)
apply (*rule of-real-power* [*symmetric*])
done

lemma *Reals-cases* [*cases set*: *Reals*]:
assumes *q* ∈ ℝ
obtains (*of-real*) *r* **where** *q* = *of-real r*
unfolding *Reals-def*
proof −
from ‹*q* ∈ ℝ› **have** *q* ∈ *range of-real* **unfolding** *Reals-def* **.**
then obtain *r* **where** *q* = *of-real r* **..**
then show *thesis* **..**
qed

lemma *Reals-induct* [*case-names of-real*, *induct set*: *Reals*]:
q ∈ ℝ ⟹ (⋀*r*. *P* (*of-real r*)) ⟹ *P q*
by (*rule Reals-cases*) *auto*

10.5 Real normed vector spaces

class *norm* = *type* +
fixes *norm* :: *'a* ⇒ *real*

instantiation *real* :: *norm*

begin

definition
real-norm-def [*simp*]: *norm r* ≡ |*r*|

instance ..

end

class *sgn-div-norm* = *scaleR* + *norm* + *sgn* +
assumes *sgn-div-norm*: *sgn x* = *x* /$_R$ *norm x*

class *real-normed-vector* = *real-vector* + *sgn-div-norm* +
assumes *norm-ge-zero* [*simp*]: *0* ≤ *norm x*
and *norm-eq-zero* [*simp*]: *norm x* = *0* ⟷ *x* = *0*
and *norm-triangle-ineq*: *norm* (*x* + *y*) ≤ *norm x* + *norm y*
and *norm-scaleR*: *norm* (*scaleR a x*) = |*a*| ∗ *norm x*

class *real-normed-algebra* = *real-algebra* + *real-normed-vector* +
assumes *norm-mult-ineq*: *norm* (*x* ∗ *y*) ≤ *norm x* ∗ *norm y*

class *real-normed-algebra-1* = *real-algebra-1* + *real-normed-algebra* +
assumes *norm-one* [*simp*]: *norm 1* = *1*

class *real-normed-div-algebra* = *real-div-algebra* + *real-normed-vector* +
assumes *norm-mult*: *norm* (*x* ∗ *y*) = *norm x* ∗ *norm y*

class *real-normed-field* = *real-field* + *real-normed-div-algebra*

instance *real-normed-div-algebra* < *real-normed-algebra-1*
proof
fix *x y* :: *'a*
show *norm* (*x* ∗ *y*) ≤ *norm x* ∗ *norm y*
by (*simp add*: *norm-mult*)
next
have *norm* (*1* ∗ *1*::*'a*) = *norm* (*1*::*'a*) ∗ *norm* (*1*::*'a*)
by (*rule norm-mult*)
thus *norm* (*1*::*'a*) = *1* **by** *simp*
qed

instance *real* :: *real-normed-field*
apply (*intro-classes*, *unfold real-norm-def real-scaleR-def*)
apply (*simp add*: *real-sgn-def*)
apply (*rule abs-ge-zero*)
apply (*rule abs-eq-0*)
apply (*rule abs-triangle-ineq*)
apply (*rule abs-mult*)
apply (*rule abs-mult*)
done

lemma *norm-zero* [*simp*]: *norm* (*0*::*'a*::*real-normed-vector*) = *0*
by *simp*

lemma *zero-less-norm-iff* [*simp*]:
fixes *x* :: *'a*::*real-normed-vector*
shows (*0* < *norm x*) = (*x* ≠ *0*)
by (*simp add*: *order-less-le*)

lemma *norm-not-less-zero* [*simp*]:
fixes *x* :: *'a*::*real-normed-vector*
shows ¬ *norm x* < *0*
by (*simp add*: *linorder-not-less*)

lemma *norm-le-zero-iff* [*simp*]:
fixes *x* :: *'a*::*real-normed-vector*
shows (*norm x* ≤ *0*) = (*x* = *0*)
by (*simp add*: *order-le-less*)

lemma *norm-minus-cancel* [*simp*]:
fixes *x* :: *'a*::*real-normed-vector*
shows *norm* (− *x*) = *norm x*
proof −
have *norm* (− *x*) = *norm* (*scaleR* (− *1*) *x*)
by (*simp only*: *scaleR-minus-left scaleR-one*)
also have ... = |− *1*| ∗ *norm x*
by (*rule norm-scaleR*)
finally show *?thesis* **by** *simp*
qed

lemma *norm-minus-commute*:
fixes *a b* :: *'a*::*real-normed-vector*
shows *norm* (*a* − *b*) = *norm* (*b* − *a*)
proof −
have *norm* (− (*b* − *a*)) = *norm* (*b* − *a*)
by (*rule norm-minus-cancel*)
thus *?thesis* **by** *simp*
qed

lemma *norm-triangle-ineq2*:
fixes *a b* :: *'a*::*real-normed-vector*
shows *norm a* − *norm b* ≤ *norm* (*a* − *b*)
proof −
have *norm* (*a* − *b* + *b*) ≤ *norm* (*a* − *b*) + *norm b*
by (*rule norm-triangle-ineq*)
thus *?thesis* **by** *simp*
qed

lemma *norm-triangle-ineq3*:

fixes *a b* :: *'a*::*real-normed-vector*
shows $|norm\ a - norm\ b| \leq norm\ (a - b)$
apply (*subst abs-le-iff*)
apply *auto*
apply (*rule norm-triangle-ineq2*)
apply (*subst norm-minus-commute*)
apply (*rule norm-triangle-ineq2*)
done

lemma *norm-triangle-ineq4*:
fixes *a b* :: *'a*::*real-normed-vector*
shows $norm\ (a - b) \leq norm\ a + norm\ b$
proof −
have $norm\ (a + - b) \leq norm\ a + norm\ (- b)$
by (*rule norm-triangle-ineq*)
thus *?thesis*
by (*simp only*: *diff-minus norm-minus-cancel*)
qed

lemma *norm-diff-ineq*:
fixes *a b* :: *'a*::*real-normed-vector*
shows $norm\ a - norm\ b \leq norm\ (a + b)$
proof −
have $norm\ a - norm\ (- b) \leq norm\ (a - - b)$
by (*rule norm-triangle-ineq2*)
thus *?thesis* **by** *simp*
qed

lemma *norm-diff-triangle-ineq*:
fixes *a b c d* :: *'a*::*real-normed-vector*
shows $norm\ ((a + b) - (c + d)) \leq norm\ (a - c) + norm\ (b - d)$
proof −
have $norm\ ((a + b) - (c + d)) = norm\ ((a - c) + (b - d))$
by (*simp add*: *diff-minus add-ac*)
also have $\ldots \leq norm\ (a - c) + norm\ (b - d)$
by (*rule norm-triangle-ineq*)
finally show *?thesis* **.**
qed

lemma *abs-norm-cancel* [*simp*]:
fixes *a* :: *'a*::*real-normed-vector*
shows $|norm\ a| = norm\ a$
by (*rule abs-of-nonneg* [*OF norm-ge-zero*])

lemma *norm-add-less*:
fixes *x y* :: *'a*::*real-normed-vector*
shows $[\![norm\ x < r; norm\ y < s]\!] \Longrightarrow norm\ (x + y) < r + s$
by (*rule order-le-less-trans* [*OF norm-triangle-ineq add-strict-mono*])

lemma *norm-mult-less*:
fixes *x y* :: *'a*::*real-normed-algebra*
shows ⟦*norm x* < *r*; *norm y* < *s*⟧ ⟹ *norm* (*x* ∗ *y*) < *r* ∗ *s*
apply (*rule order-le-less-trans* [*OF norm-mult-ineq*])
apply (*simp add*: *mult-strict-mono'*)
done

lemma *norm-of-real* [*simp*]:
norm (*of-real r* :: *'a*::*real-normed-algebra-1*) = |*r*|
unfolding *of-real-def* **by** (*simp add*: *norm-scaleR*)

lemma *norm-number-of* [*simp*]:
norm (*number-of w*::*'a*::{*number-ring*,*real-normed-algebra-1*})
= |*number-of w*|
by (*subst of-real-number-of-eq* [*symmetric*], *rule norm-of-real*)

lemma *norm-of-int* [*simp*]:
norm (*of-int z*::*'a*::*real-normed-algebra-1*) = |*of-int z*|
by (*subst of-real-of-int-eq* [*symmetric*], *rule norm-of-real*)

lemma *norm-of-nat* [*simp*]:
norm (*of-nat n*::*'a*::*real-normed-algebra-1*) = *of-nat n*
apply (*subst of-real-of-nat-eq* [*symmetric*])
apply (*subst norm-of-real*, *simp*)
done

lemma *nonzero-norm-inverse*:
fixes *a* :: *'a*::*real-normed-div-algebra*
shows *a* ≠ *0* ⟹ *norm* (*inverse a*) = *inverse* (*norm a*)
apply (*rule inverse-unique* [*symmetric*])
apply (*simp add*: *norm-mult* [*symmetric*])
done

lemma *norm-inverse*:
fixes *a* :: *'a*::{*real-normed-div-algebra*,*division-by-zero*}
shows *norm* (*inverse a*) = *inverse* (*norm a*)
apply (*case-tac a* = *0*, *simp*)
apply (*erule nonzero-norm-inverse*)
done

lemma *nonzero-norm-divide*:
fixes *a b* :: *'a*::*real-normed-field*
shows *b* ≠ *0* ⟹ *norm* (*a* / *b*) = *norm a* / *norm b*
by (*simp add*: *divide-inverse norm-mult nonzero-norm-inverse*)

lemma *norm-divide*:
fixes *a b* :: *'a*::{*real-normed-field*,*division-by-zero*}
shows *norm* (*a* / *b*) = *norm a* / *norm b*
by (*simp add*: *divide-inverse norm-mult norm-inverse*)

lemma *norm-power-ineq*:
fixes $x :: 'a::\{real\text{-}normed\text{-}algebra\text{-}1,recpower\}$
shows $norm\ (x\ \hat{}\ n) \leq norm\ x\ \hat{}\ n$
proof (*induct n*)
case *0* **show** $norm\ (x\ \hat{}\ 0) \leq norm\ x\ \hat{}\ 0$ **by** *simp*
next
case (*Suc n*)
have $norm\ (x * x\ \hat{}\ n) \leq norm\ x * norm\ (x\ \hat{}\ n)$
by (*rule norm-mult-ineq*)
also from *Suc* **have** $\ldots \leq norm\ x * norm\ x\ \hat{}\ n$
using *norm-ge-zero* **by** (*rule mult-left-mono*)
finally show $norm\ (x\ \hat{}\ Suc\ n) \leq norm\ x\ \hat{}\ Suc\ n$
by (*simp add*: *power-Suc*)
qed

lemma *norm-power*:
fixes $x :: 'a::\{real\text{-}normed\text{-}div\text{-}algebra,recpower\}$
shows $norm\ (x\ \hat{}\ n) = norm\ x\ \hat{}\ n$
by (*induct n*) (*simp-all add*: *power-Suc norm-mult*)

10.6 Sign function

lemma *norm-sgn*:
$norm\ (sgn(x::'a::real\text{-}normed\text{-}vector)) = (if\ x = 0\ then\ 0\ else\ 1)$
by (*simp add*: *sgn-div-norm norm-scaleR*)

lemma *sgn-zero* [*simp*]: $sgn(0::'a::real\text{-}normed\text{-}vector) = 0$
by (*simp add*: *sgn-div-norm*)

lemma *sgn-zero-iff*: $(sgn(x::'a::real\text{-}normed\text{-}vector) = 0) = (x = 0)$
by (*simp add*: *sgn-div-norm*)

lemma *sgn-minus*: $sgn\ (-\ x) = -\ sgn(x::'a::real\text{-}normed\text{-}vector)$
by (*simp add*: *sgn-div-norm*)

lemma *sgn-scaleR*:
$sgn\ (scaleR\ r\ x) = scaleR\ (sgn\ r)\ (sgn(x::'a::real\text{-}normed\text{-}vector))$
by (*simp add*: *sgn-div-norm norm-scaleR mult-ac*)

lemma *sgn-one* [*simp*]: $sgn\ (1::'a::real\text{-}normed\text{-}algebra\text{-}1) = 1$
by (*simp add*: *sgn-div-norm*)

lemma *sgn-of-real*:
$sgn\ (of\text{-}real\ r::'a::real\text{-}normed\text{-}algebra\text{-}1) = of\text{-}real\ (sgn\ r)$
unfolding *of-real-def* **by** (*simp only*: *sgn-scaleR sgn-one*)

lemma *sgn-mult*:
fixes $x\ y :: 'a::real\text{-}normed\text{-}div\text{-}algebra$

shows *sgn* $(x * y) =$ *sgn* $x *$ *sgn* y
by (*simp add*: *sgn-div-norm norm-mult mult-commute*)

lemma *real-sgn-eq*: *sgn* $(x$::*real*$) =$ x / $|x|$
by (*simp add*: *sgn-div-norm divide-inverse*)

lemma *real-sgn-pos*: $0 <$ $(x$::*real*$) \Longrightarrow$ *sgn* $x = 1$
unfolding *real-sgn-eq* **by** *simp*

lemma *real-sgn-neg*: $(x$::*real*$) < 0 \Longrightarrow$ *sgn* $x = -1$
unfolding *real-sgn-eq* **by** *simp*

10.7 Bounded Linear and Bilinear Operators

locale *bounded-linear* = *additive* +
constrains f :: $'a$::*real-normed-vector* $\Rightarrow$ $'b$::*real-normed-vector*
assumes *scaleR*: f (*scaleR* r x) = *scaleR* r (f x)
assumes *bounded*: $\exists K.\ \forall x.$ *norm* (f x) $\leq$ *norm* $x * K$

lemma (**in** *bounded-linear*) *pos-bounded*:
$\exists K{>}0.\ \forall x.$ *norm* (f x) $\leq$ *norm* $x * K$
proof −
obtain K **where** K: $\bigwedge x.$ *norm* (f x) $\leq$ *norm* $x * K$
using *bounded* **by** *fast*
show *?thesis*
proof (*intro exI impI conjI allI*)
show $0 <$ *max* 1 K
by (*rule order-less-le-trans* [*OF zero-less-one le-maxI1*])
next
fix x
have *norm* (f x) $\leq$ *norm* $x * K$ **using** K **.**
also have $\ldots \leq$ *norm* $x *$ *max* 1 K
by (*rule mult-left-mono* [*OF le-maxI2 norm-ge-zero*])
finally show *norm* (f x) $\leq$ *norm* $x *$ *max* 1 K **.**
qed
qed

lemma (**in** *bounded-linear*) *nonneg-bounded*:
$\exists K{\geq}0.\ \forall x.$ *norm* (f x) $\leq$ *norm* $x * K$
proof −
from *pos-bounded*
show *?thesis* **by** (*auto intro*: *order-less-imp-le*)
qed

locale *bounded-bilinear* =
fixes *prod* :: [$'a$::*real-normed-vector*, $'b$::*real-normed-vector*]
$\Rightarrow$ $'c$::*real-normed-vector*
(**infixl** $**$ *70*)
assumes *add-left*: *prod* (a + a') b = *prod* a b + *prod* a' b

assumes *add-right*: $prod\ a\ (b + b') = prod\ a\ b + prod\ a\ b'$
assumes *scaleR-left*: $prod\ (scaleR\ r\ a)\ b = scaleR\ r\ (prod\ a\ b)$
assumes *scaleR-right*: $prod\ a\ (scaleR\ r\ b) = scaleR\ r\ (prod\ a\ b)$
assumes *bounded*: $\exists K.\ \forall a\ b.\ norm\ (prod\ a\ b) \leq norm\ a * norm\ b * K$

lemma (**in** *bounded-bilinear*) *pos-bounded*:
$\exists K{>}0.\ \forall a\ b.\ norm\ (a ** b) \leq norm\ a * norm\ b * K$
apply (*cut-tac bounded*, *erule exE*)
apply (*rule-tac x=max 1 K* **in** *exI*, *safe*)
apply (*rule order-less-le-trans* [*OF zero-less-one le-maxI1*])
apply (*drule spec*, *drule spec*, *erule order-trans*)
apply (*rule mult-left-mono* [*OF le-maxI2*])
apply (*intro mult-nonneg-nonneg norm-ge-zero*)
done

lemma (**in** *bounded-bilinear*) *nonneg-bounded*:
$\exists K{\geq}0.\ \forall a\ b.\ norm\ (a ** b) \leq norm\ a * norm\ b * K$
proof −
from *pos-bounded*
show *?thesis* **by** (*auto intro*: *order-less-imp-le*)
qed

lemma (**in** *bounded-bilinear*) *additive-right*: $additive\ (\lambda b.\ prod\ a\ b)$
by (*rule additive.intro*, *rule add-right*)

lemma (**in** *bounded-bilinear*) *additive-left*: $additive\ (\lambda a.\ prod\ a\ b)$
by (*rule additive.intro*, *rule add-left*)

lemma (**in** *bounded-bilinear*) *zero-left*: $prod\ 0\ b = 0$
by (*rule additive.zero* [*OF additive-left*])

lemma (**in** *bounded-bilinear*) *zero-right*: $prod\ a\ 0 = 0$
by (*rule additive.zero* [*OF additive-right*])

lemma (**in** *bounded-bilinear*) *minus-left*: $prod\ (-\ a)\ b = -\ prod\ a\ b$
by (*rule additive.minus* [*OF additive-left*])

lemma (**in** *bounded-bilinear*) *minus-right*: $prod\ a\ (-\ b) = -\ prod\ a\ b$
by (*rule additive.minus* [*OF additive-right*])

lemma (**in** *bounded-bilinear*) *diff-left*:
$prod\ (a - a')\ b = prod\ a\ b - prod\ a'\ b$
by (*rule additive.diff* [*OF additive-left*])

lemma (**in** *bounded-bilinear*) *diff-right*:
$prod\ a\ (b - b') = prod\ a\ b - prod\ a\ b'$
by (*rule additive.diff* [*OF additive-right*])

lemma (**in** *bounded-bilinear*) *bounded-linear-left*:

bounded-linear ($\lambda a.\ a ** b$)
apply (*unfold-locales*)
apply (*rule add-left*)
apply (*rule scaleR-left*)
apply (*cut-tac bounded*, *safe*)
apply (*rule-tac x=norm b * K* **in** *exI*)
apply (*simp add*: *mult-ac*)
done

lemma (**in** *bounded-bilinear*) *bounded-linear-right*:
bounded-linear ($\lambda b.\ a ** b$)
apply (*unfold-locales*)
apply (*rule add-right*)
apply (*rule scaleR-right*)
apply (*cut-tac bounded*, *safe*)
apply (*rule-tac x=norm a * K* **in** *exI*)
apply (*simp add*: *mult-ac*)
done

lemma (**in** *bounded-bilinear*) *prod-diff-prod*:
$(x ** y - a ** b) = (x - a) ** (y - b) + (x - a) ** b + a ** (y - b)$
by (*simp add*: *diff-left diff-right*)

interpretation *mult*:
bounded-bilinear [$op * :: 'a \Rightarrow 'a \Rightarrow 'a$::*real-normed-algebra*]
apply (*rule bounded-bilinear.intro*)
apply (*rule left-distrib*)
apply (*rule right-distrib*)
apply (*rule mult-scaleR-left*)
apply (*rule mult-scaleR-right*)
apply (*rule-tac x=1* **in** *exI*)
apply (*simp add*: *norm-mult-ineq*)
done

interpretation *mult-left*:
bounded-linear [(λx::'a::*real-normed-algebra*. $x * y$)]
by (*rule mult.bounded-linear-left*)

interpretation *mult-right*:
bounded-linear [(λy::'a::*real-normed-algebra*. $x * y$)]
by (*rule mult.bounded-linear-right*)

interpretation *divide*:
bounded-linear [(λx::'a::*real-normed-field*. x / y)]
unfolding *divide-inverse* **by** (*rule mult.bounded-linear-left*)

interpretation *scaleR*: *bounded-bilinear* [*scaleR*]
apply (*rule bounded-bilinear.intro*)
apply (*rule scaleR-left-distrib*)

apply (*rule scaleR-right-distrib*)
apply *simp*
apply (*rule scaleR-left-commute*)
apply (*rule-tac x=1* **in** *exI*)
apply (*simp add*: *norm-scaleR*)
done

interpretation *scaleR-left*: *bounded-linear* [$\lambda r.$ *scaleR r x*]
by (*rule scaleR.bounded-linear-left*)

interpretation *scaleR-right*: *bounded-linear* [$\lambda x.$ *scaleR r x*]
by (*rule scaleR.bounded-linear-right*)

interpretation *of-real*: *bounded-linear* [$\lambda r.$ *of-real r*]
unfolding *of-real-def* **by** (*rule scaleR.bounded-linear-left*)

end

theory *Real*
imports *ContNotDenum RealVector*
begin
end

11 Float: Floating Point Representation of the Reals

theory *Float*
imports *Real Parity*
uses ~~/*src*/*Tools*/*float.ML* (*float-arith.ML*)
begin

definition
pow2 :: *int* ⇒ *real* **where**
pow2 a = (*if* (*0* <= *a*) *then* (*2*^(*nat a*)) *else* (*inverse* (*2*^(*nat* (−*a*)))))

definition
float :: *int* ∗ *int* ⇒ *real* **where**
float x = *real* (*fst x*) ∗ *pow2* (*snd x*)

lemma *pow2-0*[*simp*]: *pow2 0* = *1*
by (*simp add*: *pow2-def*)

lemma *pow2-1*[*simp*]: *pow2 1* = *2*
by (*simp add*: *pow2-def*)

lemma *pow2-neg*: *pow2 x = inverse (pow2 (−x))*
by (*simp add*: *pow2-def*)

lemma *pow2-add1*: *pow2 (1 + a) = 2 ∗ (pow2 a)*
proof −
  have *h*: ! *n*. *nat (2 + int n) − Suc 0 = nat (1 + int n)* **by** *arith*
  have *g*: ! *a b*. *a − −1 = a + (1*::*int)* **by** *arith*
  have *pos*: ! *n*. *pow2 (int n + 1) = 2 ∗ pow2 (int n)*
    apply (*auto*, *induct-tac n*)
    apply (*simp-all add*: *pow2-def*)
   apply (*rule-tac m1=2* **and** *n1=nat (2 + int na)* **in** *ssubst*[*OF realpow-num-eq-if*])
    by (*auto simp add*: *h*)
  show *?thesis*
  proof (*induct a*)
    case (*1 n*)
    from *pos* **show** *?case* **by** (*simp add*: *ring-simps*)
  next
    case (*2 n*)
    show *?case*
      apply (*auto*)
      apply (*subst pow2-neg*[*of − int n*])
      apply (*subst pow2-neg*[*of −1 − int n*])
      apply (*auto simp add*: *g pos*)
      done
  qed
qed

lemma *pow2-add*: *pow2 (a+b) = (pow2 a) ∗ (pow2 b)*
proof (*induct b*)
  case (*1 n*)
  show *?case*
  proof (*induct n*)
    case *0*
    show *?case* **by** *simp*
  next
    case (*Suc m*)
    show *?case* **by** (*auto simp add*: *ring-simps pow2-add1 prems*)
  qed
next
  case (*2 n*)
  show *?case*
  proof (*induct n*)
    case *0*
    show *?case*
      apply (*auto*)
      apply (*subst pow2-neg*[*of a + −1*])
      apply (*subst pow2-neg*[*of −1*])
      apply (*simp*)

```
    apply (insert pow2-add1[of −a])
    apply (simp add: ring-simps)
    apply (subst pow2-neg[of −a])
    apply (simp)
    done
  case (Suc m)
  have a: int m − (a + −2) =  1 + (int m − a + 1) by arith
  have b: int m − −2 = 1 + (int m + 1) by arith
  show ?case
    apply (auto)
    apply (subst pow2-neg[of a + (−2 − int m)])
    apply (subst pow2-neg[of −2 − int m])
    apply (auto simp add: ring-simps)
    apply (subst a)
    apply (subst b)
    apply (simp only: pow2-add1)
    apply (subst pow2-neg[of int m − a + 1])
    apply (subst pow2-neg[of int m + 1])
    apply auto
    apply (insert prems)
    apply (auto simp add: ring-simps)
    done
 qed
qed

lemma float (a, e) + float (b, e) = float (a + b, e)
by (simp add: float-def ring-simps)

definition
  int-of-real :: real ⇒ int where
  int-of-real x = (SOME y. real y = x)

definition
  real-is-int :: real ⇒ bool where
  real-is-int x = (EX (u::int). x = real u)

lemma real-is-int-def2: real-is-int x = (x = real (int-of-real x))
by (auto simp add: real-is-int-def int-of-real-def)

lemma float-transfer: real-is-int ((real a)*(pow2 c)) ⟹ float (a, b) = float (int-of-real
((real a)*(pow2 c)), b − c)
by (simp add: float-def real-is-int-def2 pow2-add[symmetric])

lemma pow2-int: pow2 (int c) = 2^c
by (simp add: pow2-def)

lemma float-transfer-nat: float (a, b) = float (a * 2^c, b − int c)
by (simp add: float-def pow2-int[symmetric] pow2-add[symmetric])
```

lemma *real-is-int-real*[*simp*]: *real-is-int* (*real* (*x*::*int*))
by (*auto simp add*: *real-is-int-def int-of-real-def*)

lemma *int-of-real-real*[*simp*]: *int-of-real* (*real x*) = *x*
by (*simp add*: *int-of-real-def*)

lemma *real-int-of-real*[*simp*]: *real-is-int x* ⟹ *real* (*int-of-real x*) = *x*
by (*auto simp add*: *int-of-real-def real-is-int-def*)

lemma *real-is-int-add-int-of-real*: *real-is-int a* ⟹ *real-is-int b* ⟹ (*int-of-real* (*a*+*b*)) = (*int-of-real a*) + (*int-of-real b*)
by (*auto simp add*: *int-of-real-def real-is-int-def*)

lemma *real-is-int-add*[*simp*]: *real-is-int a* ⟹ *real-is-int b* ⟹ *real-is-int* (*a*+*b*)
apply (*subst real-is-int-def2*)
apply (*simp add*: *real-is-int-add-int-of-real real-int-of-real*)
done

lemma *int-of-real-sub*: *real-is-int a* ⟹ *real-is-int b* ⟹ (*int-of-real* (*a*−*b*)) = (*int-of-real a*) − (*int-of-real b*)
by (*auto simp add*: *int-of-real-def real-is-int-def*)

lemma *real-is-int-sub*[*simp*]: *real-is-int a* ⟹ *real-is-int b* ⟹ *real-is-int* (*a*−*b*)
apply (*subst real-is-int-def2*)
apply (*simp add*: *int-of-real-sub real-int-of-real*)
done

lemma *real-is-int-rep*: *real-is-int x* ⟹ *?*! (*a*::*int*). *real a* = *x*
by (*auto simp add*: *real-is-int-def*)

lemma *int-of-real-mult*:
  assumes *real-is-int a real-is-int b*
  shows (*int-of-real* (*a*∗*b*)) = (*int-of-real a*) ∗ (*int-of-real b*)
proof −
  from *prems* **have** *a*: *?*! (*a'*::*int*). *real a'* = *a* **by** (*rule-tac real-is-int-rep*, *auto*)
  from *prems* **have** *b*: *?*! (*b'*::*int*). *real b'* = *b* **by** (*rule-tac real-is-int-rep*, *auto*)
  from *a* **obtain** *a'*::*int* **where** *a'*:*a* = *real a'* **by** *auto*
  from *b* **obtain** *b'*::*int* **where** *b'*:*b* = *real b'* **by** *auto*
  have *r*: *real a'* ∗ *real b'* = *real* (*a'* ∗ *b'*) **by** *auto*
  show *?thesis*
    apply (*simp add*: *a' b'*)
    apply (*subst r*)
    apply (*simp only*: *int-of-real-real*)
    done
qed

lemma *real-is-int-mult*[*simp*]: *real-is-int a* ⟹ *real-is-int b* ⟹ *real-is-int* (*a*∗*b*)
apply (*subst real-is-int-def2*)
apply (*simp add*: *int-of-real-mult*)

done

lemma *real-is-int-0*[*simp*]: *real-is-int* (*0*::*real*)
by (*simp add*: *real-is-int-def int-of-real-def*)

lemma *real-is-int-1*[*simp*]: *real-is-int* (*1*::*real*)
proof −
  have *real-is-int* (*1*::*real*) = *real-is-int*(*real* (*1*::*int*)) **by** *auto*
  also have ... = *True* **by** (*simp only*: *real-is-int-real*)
  ultimately show *?thesis* **by** *auto*
qed

lemma *real-is-int-n1*: *real-is-int* (−*1*::*real*)
proof −
  have *real-is-int* (−*1*::*real*) = *real-is-int*(*real* (−*1*::*int*)) **by** *auto*
  also have ... = *True* **by** (*simp only*: *real-is-int-real*)
  ultimately show *?thesis* **by** *auto*
qed

lemma *real-is-int-number-of*[*simp*]: *real-is-int* ((*number-of* :: *int* ⇒ *real*) *x*)
proof −
  have *neg1*: *real-is-int* (−*1*::*real*)
  proof −
    have *real-is-int* (−*1*::*real*) = *real-is-int*(*real* (−*1*::*int*)) **by** *auto*
    also have ... = *True* **by** (*simp only*: *real-is-int-real*)
    ultimately show *?thesis* **by** *auto*
  qed

  {
    fix *x* :: *int*
    have *real-is-int* ((*number-of* :: *int* ⇒ *real*) *x*)
      unfolding *number-of-eq*
      apply (*induct x*)
      apply (*induct-tac n*)
      apply (*simp*)
      apply (*simp*)
      apply (*induct-tac n*)
      apply (*simp add*: *neg1*)
    proof −
      fix *n* :: *nat*
      assume *rn*: (*real-is-int* (*of-int* (− (*int* (*Suc n*)))))
      have *s*: −(*int* (*Suc* (*Suc n*))) = −*1* + − (*int* (*Suc n*)) **by** *simp*
      show *real-is-int* (*of-int* (− (*int* (*Suc* (*Suc n*)))))
        apply (*simp only*: *s of-int-add*)
        apply (*rule real-is-int-add*)
        apply (*simp add*: *neg1*)
        apply (*simp only*: *rn*)
        done
    qed

```
  }
  note Abs-Bin = this
  {
    fix x :: int
    have ? u. x = u
      apply (rule exI[where x = x])
      apply (simp)
      done
  }
  then obtain u::int where x = u by auto
  with Abs-Bin show ?thesis by auto
qed

lemma int-of-real-0[simp]: int-of-real (0::real) = (0::int)
by (simp add: int-of-real-def)

lemma int-of-real-1[simp]: int-of-real (1::real) = (1::int)
proof −
  have 1: (1::real) = real (1::int) by auto
  show ?thesis by (simp only: 1 int-of-real-real)
qed

lemma int-of-real-number-of[simp]: int-of-real (number-of b) = number-of b
proof −
  have real-is-int (number-of b) by simp
  then have uu: ?! u::int. number-of b = real u by (auto simp add: real-is-int-rep)
  then obtain u::int where u:number-of b = real u by auto
  have number-of b = real ((number-of b)::int)
    by (simp add: number-of-eq real-of-int-def)
  have ub: number-of b = real ((number-of b)::int)
    by (simp add: number-of-eq real-of-int-def)
  from uu u ub have unb: u = number-of b
    by blast
  have int-of-real (number-of b) = u by (simp add: u)
  with unb show ?thesis by simp
qed

lemma float-transfer-even: even a ⟹ float (a, b) = float (a div 2, b+1)
  apply (subst float-transfer[where a=a and b=b and c=−1, simplified])
  apply (simp-all add: pow2-def even-def real-is-int-def ring-simps)
  apply (auto)
proof −
  fix q::int
  have a:b − (−1::int) = (1::int) + b by arith
  show (float (q, (b − (−1::int)))) = (float (q, ((1::int) + b)))
    by (simp add: a)
qed

consts
```

norm-float :: *int*∗*int* ⇒ *int*∗*int*

lemma *int-div-zdiv*: *int* (*a div b*) = (*int a*) *div* (*int b*)
by (*rule zdiv-int*)

lemma *int-mod-zmod*: *int* (*a mod b*) = (*int a*) *mod* (*int b*)
by (*rule zmod-int*)

lemma *abs-div-2-less*: *a* ≠ *0* ⟹ *a* ≠ −*1* ⟹ *abs*((*a*::*int*) *div 2*) < *abs a*
by *arith*

lemma *terminating-norm-float*: ∀ *a*. (*a*::*int*) ≠ *0* ∧ *even a* ⟶ *a* ≠ *0* ∧ |*a div 2*|
< |*a*|
apply (*auto*)
apply (*rule abs-div-2-less*)
apply (*auto*)
done

declare [[*simp-depth-limit* = *2*]]
recdef *norm-float measure* (% (*a*,*b*). *nat* (*abs a*))
norm-float (*a*,*b*) = (*if* (*a* ≠ *0*) & (*even a*) *then norm-float* (*a div 2*, *b*+*1*) *else*
(*if a*=*0 then* (*0*,*0*) *else* (*a*,*b*)))
(**hints** *simp*: *even-def terminating-norm-float*)
declare [[*simp-depth-limit* = *100*]]

lemma *norm-float*: *float x* = *float* (*norm-float x*)
proof −
{
fix *a b* :: *int*
have *norm-float-pair*: *float* (*a*,*b*) = *float* (*norm-float* (*a*,*b*))
proof (*induct a b rule*: *norm-float.induct*)
case (*1 u v*)
show *?case*
proof *cases*
assume *u*: *u* ≠ *0* ∧ *even u*
with *prems* **have** *ind*: *float* (*u div 2*, *v* + *1*) = *float* (*norm-float* (*u div 2*,
v + *1*)) **by** *auto*
with *u* **have** *float* (*u*,*v*) = *float* (*u div 2*, *v*+*1*) **by** (*simp add*: *float-transfer-even*)
then show *?thesis*
apply (*subst norm-float.simps*)
apply (*simp add*: *ind*)
done
next
assume ~(*u* ≠ *0* ∧ *even u*)
then show *?thesis*
by (*simp add*: *prems float-def*)
qed
qed
}

```
  note helper = this
  have ? a b. x = (a,b) by auto
  then obtain a b where x = (a, b) by blast
  then show ?thesis by (simp only: helper)
qed

lemma float-add-l0: float (0, e) + x = x
  by (simp add: float-def)

lemma float-add-r0: x + float (0, e) = x
  by (simp add: float-def)

lemma float-add:
  float (a1, e1) + float (a2, e2) =
  (if e1<=e2 then float (a1+a2*2^(nat(e2-e1)), e1)
  else float (a1*2^(nat (e1-e2))+a2, e2))
  apply (simp add: float-def ring-simps)
  apply (auto simp add: pow2-int[symmetric] pow2-add[symmetric])
  done

lemma float-add-assoc1:
  (x + float (y1, e1)) + float (y2, e2) = (float (y1, e1) + float (y2, e2)) + x
  by simp

lemma float-add-assoc2:
  (float (y1, e1) + x) + float (y2, e2) = (float (y1, e1) + float (y2, e2)) + x
  by simp

lemma float-add-assoc3:
  float (y1, e1) + (x + float (y2, e2)) = (float (y1, e1) + float (y2, e2)) + x
  by simp

lemma float-add-assoc4:
  float (y1, e1) + (float (y2, e2) + x) = (float (y1, e1) + float (y2, e2)) + x
  by simp

lemma float-mult-l0: float (0, e) * x = float (0, 0)
  by (simp add: float-def)

lemma float-mult-r0: x * float (0, e) = float (0, 0)
  by (simp add: float-def)

definition
  lbound :: real ⇒ real
where
  lbound x = min 0 x

definition
  ubound :: real ⇒ real
```

where
ubound x = max 0 x

lemma *lbound*: *lbound x ≤ x*
by (*simp add*: *lbound-def*)

lemma *ubound*: *x ≤ ubound x*
by (*simp add*: *ubound-def*)

lemma *float-mult*:
*float (a1, e1) * float (a2, e2) =*
*(float (a1 * a2, e1 + e2))*
by (*simp add*: *float-def pow2-add*)

lemma *float-minus*:
- (float (a,b)) = float (-a, b)
by (*simp add*: *float-def*)

lemma *zero-less-pow2*:
0 < pow2 x
proof −
{
fix *y*
have *0 <= y ⟹ 0 < pow2 y*
by (*induct y*, *induct-tac n*, *simp-all add*: *pow2-add*)
}
note *helper=this*
show *?thesis*
apply (*case-tac 0 <= x*)
apply (*simp add*: *helper*)
apply (*subst pow2-neg*)
apply (*simp add*: *helper*)
done
qed

lemma *zero-le-float*:
(0 <= float (a,b)) = (0 <= a)
apply (*auto simp add*: *float-def*)
apply (*auto simp add*: *zero-le-mult-iff zero-less-pow2*)
apply (*insert zero-less-pow2*[*of b*])
apply (*simp-all*)
done

lemma *float-le-zero*:
(float (a,b) <= 0) = (a <= 0)
apply (*auto simp add*: *float-def*)
apply (*auto simp add*: *mult-le-0-iff*)
apply (*insert zero-less-pow2*[*of b*])
apply *auto*

done

lemma *float-abs*:
abs (*float* (*a*,*b*)) = (*if* 0 <= *a* *then* (*float* (*a*,*b*)) *else* (*float* (−*a*,*b*)))
apply (*auto simp add*: *abs-if*)
apply (*simp-all add*: *zero-le-float*[*symmetric*, *of a b*] *float-minus*)
done

lemma *float-zero*:
float (0, *b*) = 0
by (*simp add*: *float-def*)

lemma *float-pprt*:
pprt (*float* (*a*, *b*)) = (*if* 0 <= *a* *then* (*float* (*a*,*b*)) *else* (*float* (0, *b*)))
by (*auto simp add*: *zero-le-float float-le-zero float-zero*)

lemma *pprt-lbound*: *pprt* (*lbound x*) = *float* (0, 0)
apply (*simp add*: *float-def*)
apply (*rule pprt-eq-0*)
apply (*simp add*: *lbound-def*)
done

lemma *nprt-ubound*: *nprt* (*ubound x*) = *float* (0, 0)
apply (*simp add*: *float-def*)
apply (*rule nprt-eq-0*)
apply (*simp add*: *ubound-def*)
done

lemma *float-nprt*:
nprt (*float* (*a*, *b*)) = (*if* 0 <= *a* *then* (*float* (0,*b*)) *else* (*float* (*a*, *b*)))
by (*auto simp add*: *zero-le-float float-le-zero float-zero*)

lemma *norm-0-1*: (0::-::*number-ring*) = *Numeral0* & (1::-::*number-ring*) = *Numeral1*
by *auto*

lemma *add-left-zero*: 0 + *a* = (*a*::'*a*::*comm-monoid-add*)
by *simp*

lemma *add-right-zero*: *a* + 0 = (*a*::'*a*::*comm-monoid-add*)
by *simp*

lemma *mult-left-one*: 1 * *a* = (*a*::'*a*::*semiring-1*)
by *simp*

lemma *mult-right-one*: *a* * 1 = (*a*::'*a*::*semiring-1*)
by *simp*

lemma *int-pow-0*: (*a*::*int*) ^(*Numeral0*) = 1
by *simp*

lemma *int-pow-1*: $(a::int)\hat{}(Numeral1) = a$
by *simp*

lemma *zero-eq-Numeral0-nring*: $(0::'a::number\text{-}ring) = Numeral0$
by *simp*

lemma *one-eq-Numeral1-nring*: $(1::'a::number\text{-}ring) = Numeral1$
by *simp*

lemma *zero-eq-Numeral0-nat*: $(0::nat) = Numeral0$
by *simp*

lemma *one-eq-Numeral1-nat*: $(1::nat) = Numeral1$
by *simp*

lemma *zpower-Pls*: $(z::int)\hat{}Numeral0 = Numeral1$
by *simp*

lemma *zpower-Min*: $(z::int)\hat{}((-1)::nat) = Numeral1$
proof −
have *1*:$((-1)::nat) = 0$
by *simp*
show *?thesis* **by** (*simp add*: *1*)
qed

lemma *fst-cong*: $a=a' \Longrightarrow fst\ (a,b) = fst\ (a',b)$
by *simp*

lemma *snd-cong*: $b=b' \Longrightarrow snd\ (a,b) = snd\ (a,b')$
by *simp*

lemma *lift-bool*: $x \Longrightarrow x=True$
by *simp*

lemma *nlift-bool*: $\sim x \Longrightarrow x=False$
by *simp*

lemma *not-false-eq-true*: $(\sim\ False) = True$ **by** *simp*

lemma *not-true-eq-false*: $(\sim\ True) = False$ **by** *simp*

lemmas *binarith* =
normalize-bin-simps
pred-bin-simps succ-bin-simps
add-bin-simps minus-bin-simps mult-bin-simps

lemma *int-eq-number-of-eq*:
$(((number\text{-}of\ v)::int)=(number\text{-}of\ w)) = iszero\ ((number\text{-}of\ (v + uminus\ w))::int)$

by *simp*

lemma *int-iszero-number-of-Pls*: *iszero* (*Numeral0*::*int*)
by (*simp only*: *iszero-number-of-Pls*)

lemma *int-nonzero-number-of-Min*: ~(*iszero* ((−*1*)::*int*))
by *simp*

lemma *int-iszero-number-of-Bit0*: *iszero* ((*number-of* (*Int.Bit0 w*))::*int*) = *iszero* ((*number-of w*)::*int*)
by *simp*

lemma *int-iszero-number-of-Bit1*: ¬ *iszero* ((*number-of* (*Int.Bit1 w*))::*int*)
by *simp*

lemma *int-less-number-of-eq-neg*: (((*number-of x*)::*int*) < *number-of y*) = *neg* ((*number-of* (*x* + (*uminus y*)))::*int*)
by *simp*

lemma *int-not-neg-number-of-Pls*: ¬ (*neg* (*Numeral0*::*int*))
by *simp*

lemma *int-neg-number-of-Min*: *neg* (−*1*::*int*)
by *simp*

lemma *int-neg-number-of-Bit0*: *neg* ((*number-of* (*Int.Bit0 w*))::*int*) = *neg* ((*number-of w*)::*int*)
by *simp*

lemma *int-neg-number-of-Bit1*: *neg* ((*number-of* (*Int.Bit1 w*))::*int*) = *neg* ((*number-of w*)::*int*)
by *simp*

lemma *int-le-number-of-eq*: (((*number-of x*)::*int*) ≤ *number-of y*) = (¬ *neg* ((*number-of* (*y* + (*uminus x*)))::*int*))
by *simp*

lemmas *intarithrel* =
int-eq-number-of-eq
lift-bool[*OF int-iszero-number-of-Pls*] *nlift-bool*[*OF int-nonzero-number-of-Min*]
int-iszero-number-of-Bit0
lift-bool[*OF int-iszero-number-of-Bit1*] *int-less-number-of-eq-neg nlift-bool*[*OF int-not-neg-number-of-Pls*]
lift-bool[*OF int-neg-number-of-Min*]
int-neg-number-of-Bit0 int-neg-number-of-Bit1 int-le-number-of-eq

lemma *int-number-of-add-sym*: ((*number-of v*)::*int*) + *number-of w* = *number-of* (*v* + *w*)
by *simp*

lemma *int-number-of-diff-sym*: $((number\text{-}of\ v)::int) - number\text{-}of\ w = number\text{-}of\ (v + (uminus\ w))$
by *simp*

lemma *int-number-of-mult-sym*: $((number\text{-}of\ v)::int) * number\text{-}of\ w = number\text{-}of\ (v * w)$
by *simp*

lemma *int-number-of-minus-sym*: $-((number\text{-}of\ v)::int) = number\text{-}of\ (uminus\ v)$
by *simp*

lemmas *intarith* = *int-number-of-add-sym int-number-of-minus-sym int-number-of-diff-sym int-number-of-mult-sym*

lemmas *natarith* = *add-nat-number-of diff-nat-number-of mult-nat-number-of eq-nat-number-of less-nat-number-of*

lemmas *powerarith* = *nat-number-of zpower-number-of-even*
zpower-number-of-odd[*simplified zero-eq-Numeral0-nring one-eq-Numeral1-nring*]
zpower-Pls zpower-Min

lemmas *floatarith*[*simplified norm-0-1*] = *float-add float-add-l0 float-add-r0 float-mult float-mult-l0 float-mult-r0*
float-minus float-abs zero-le-float float-pprt float-nprt pprt-lbound nprt-ubound

lemmas *arith* = *binarith intarith intarithrel natarith powerarith floatarith not-false-eq-true not-true-eq-false*

use *float-arith.ML*

end

12 Univ-Poly: Univariate Polynomials

theory *Univ-Poly*
imports *Main*
begin

Application of polynomial as a function.

primrec (**in** *semiring-0*) *poly* :: *'a list* => *'a* => *'a* **where**
poly-Nil: *poly* [] *x* = *0*
| *poly-Cons*: *poly* (*h*#*t*) *x* = *h* + *x* * *poly t x*

12.1 Arithmetic Operations on Polynomials

addition

primrec (**in** *semiring-0*) *padd* :: *'a list* ⇒ *'a list* ⇒ *'a list* (**infixl** *+++ 65*)
where
padd-Nil: [] *+++ l2* = *l2*
| *padd-Cons*: (*h#t*) *+++ l2* = (*if l2* = [] *then h#t*
else (*h* + *hd l2*)#(*t +++ tl l2*))

Multiplication by a constant

primrec (**in** *semiring-0*) *cmult* :: *'a* ⇒ *'a list* ⇒ *'a list* (**infixl** *%** *70*) **where**
cmult-Nil: *c %** [] = []
| *cmult-Cons*: *c %** (*h#t*) = (*c* * *h*)#(*c %* t*)

Multiplication by a polynomial

primrec (**in** *semiring-0*) *pmult* :: *'a list* ⇒ *'a list* ⇒ *'a list* (**infixl** ***** *70*)
where
pmult-Nil: [] **** l2* = []
| *pmult-Cons*: (*h#t*) **** l2* = (*if t* = [] *then h %* l2*
else (*h %* l2*) *+++* ((*0*) # (*t *** l2*)))

Repeated multiplication by a polynomial

primrec (**in** *semiring-0*) *mulexp* :: *nat* ⇒ *'a list* ⇒ *'a list* ⇒ *'a list* **where**
mulexp-zero: *mulexp 0 p q* = *q*
| *mulexp-Suc*: *mulexp* (*Suc n*) *p q* = *p *** mulexp n p q*

Exponential

primrec (**in** *semiring-1*) *pexp* :: *'a list* ⇒ *nat* ⇒ *'a list* (**infixl** *%^ 80*) **where**
pexp-0: *p %^ 0* = [*1*]
| *pexp-Suc*: *p %^* (*Suc n*) = *p **** (*p %^ n*)

Quotient related value of dividing a polynomial by x + a

primrec (**in** *field*) *pquot* :: *'a list* ⇒ *'a* ⇒ *'a list* **where**
pquot-Nil: *pquot* [] *a*= []
| *pquot-Cons*: *pquot* (*h#t*) *a* = (*if t* = [] *then* [*h*]
else (*inverse*(*a*) * (*h* − *hd*(*pquot t a*)))#(*pquot t a*))

normalization of polynomials (remove extra 0 coeff)

primrec (**in** *semiring-0*) *pnormalize* :: *'a list* ⇒ *'a list* **where**
pnormalize-Nil: *pnormalize* [] = []
| *pnormalize-Cons*: *pnormalize* (*h#p*) = (*if* ((*pnormalize p*) = [])
then (*if* (*h* = *0*) *then* [] *else* [*h*])
else (*h*#(*pnormalize p*)))

definition (**in** *semiring-0*) *pnormal p* = ((*pnormalize p* = *p*) ∧ *p* ≠ [])
definition (**in** *semiring-0*) *nonconstant p* = (*pnormal p* ∧ (∀ *x*. *p* ≠ [*x*]))

Other definitions

definition (**in** *ring-1*)
poly-minus :: *'a list* => *'a list* (−− - [*80*] *80*) **where**
−− *p* = (− *1*) *%* *p*

definition (**in** *semiring-0*)
divides :: *'a list* ⇒ *'a list* ⇒ *bool* (**infixl** *divides 70*) **where**
p1 divides p2 = (∃ *q*. *poly p2* = *poly*(*p1* ∗∗∗ *q*))

— order of a polynomial
definition (**in** *ring-1*) *order* :: *'a* => *'a list* => *nat* **where**
order a p = (*SOME n*. ([−*a*, *1*] %^ *n*) *divides p* &
~ (([−*a*, *1*] %^ (*Suc n*)) *divides p*))

— degree of a polynomial
definition (**in** *semiring-0*) *degree* :: *'a list* => *nat* **where**
degree p = *length* (*pnormalize p*) − *1*

— squarefree polynomials — NB with respect to real roots only.
definition (**in** *ring-1*)
rsquarefree :: *'a list* => *bool* **where**
rsquarefree p = (*poly p* ≠ *poly* [] &
(∀ *a*. (*order a p* = *0*) | (*order a p* = *1*)))

context *semiring-0*
begin

lemma *padd-Nil2*[*simp*]: *p* +++ [] = *p*
by (*induct p*) *auto*

lemma *padd-Cons-Cons*: (*h1* # *p1*) +++ (*h2* # *p2*) = (*h1* + *h2*) # (*p1* +++ *p2*)
by *auto*

lemma *pminus-Nil*[*simp*]: −− [] = []
by (*simp add*: *poly-minus-def*)

lemma *pmult-singleton*: [*h1*] ∗∗∗ *p1* = *h1* %∗ *p1* **by** *simp*
end

lemma (**in** *semiring-1*) *poly-ident-mult*[*simp*]: *1* %∗ *t* = *t* **by** (*induct t*, *auto*)

lemma (**in** *semiring-0*) *poly-simple-add-Cons*[*simp*]: [*a*] +++ ((*0*)#*t*) = (*a*#*t*)
by *simp*

Handy general properties

lemma (**in** *comm-semiring-0*) *padd-commut*: *b* +++ *a* = *a* +++ *b*
proof(*induct b arbitrary*: *a*)
case *Nil* **thus** *?case* **by** *auto*
next
case (*Cons b bs a*) **thus** *?case* **by** (*cases a*, *simp-all add*: *add-commute*)
qed

lemma (**in** *comm-semiring-0*) *padd-assoc*: $\forall\ b\ c.\ (a\ +\!+\!+\ b)\ +\!+\!+\ c = a\ +\!+\!+\ (b\ +\!+\!+\ c)$
apply (*induct a arbitrary*: *b c*)
apply (*simp*, *clarify*)
apply (*case-tac b*, *simp-all add*: *add-ac*)
done

lemma (**in** *semiring-0*) *poly-cmult-distr*: $a\ \%*\ (\ p\ +\!+\!+\ q) = (a\ \%*\ p\ +\!+\!+\ a\ \%*\ q)$
apply (*induct p arbitrary*: *q*,*simp*)
apply (*case-tac q*, *simp-all add*: *right-distrib*)
done

lemma (**in** *ring-1*) *pmult-by-x*[*simp*]: $[0,\ 1]\ ***\ t = ((0)\#t)$
apply (*induct t*, *simp*)
apply (*auto simp add*: *mult-zero-left poly-ident-mult padd-commut*)
apply (*case-tac t*, *auto*)
done

properties of evaluation of polynomials.

lemma (**in** *semiring-0*) *poly-add*: *poly* (*p1* $+\!+\!+$ *p2*) *x* = *poly p1 x* + *poly p2 x*
proof(*induct p1 arbitrary*: *p2*)
  case *Nil* **thus** *?case* **by** *simp*
next
  case (*Cons a as p2*) **thus** *?case*
    by (*cases p2*, *simp-all add*: *add-ac right-distrib*)
qed

lemma (**in** *comm-semiring-0*) *poly-cmult*: *poly* (*c* $\%*$ *p*) *x* = *c* $*$ *poly p x*
apply (*induct p*)
apply (*case-tac* [*2*] *x*=*zero*)
apply (*auto simp add*: *right-distrib mult-ac*)
done

lemma (**in** *comm-semiring-0*) *poly-cmult-map*: *poly* (*map* (*op* $*$ *c*) *p*) *x* = *c*$*$*poly p x*
  by (*induct p*, *auto simp add*: *right-distrib mult-ac*)

lemma (**in** *comm-ring-1*) *poly-minus*: *poly* ($--$ *p*) *x* = $-$ (*poly p x*)
apply (*simp add*: *poly-minus-def*)
apply (*auto simp add*: *poly-cmult minus-mult-left*[*symmetric*])
done

lemma (**in** *comm-semiring-0*) *poly-mult*: *poly* (*p1* $***$ *p2*) *x* = *poly p1 x* $*$ *poly p2 x*
proof(*induct p1 arbitrary*: *p2*)
  case *Nil* **thus** *?case* **by** *simp*
next
  case (*Cons a as p2*)

thus *?case* **by** (*cases as*,
simp-all add: *poly-cmult poly-add left-distrib right-distrib mult-ac*)
qed

class *recpower-semiring* = *semiring* + *recpower*
class *recpower-semiring-1* = *semiring-1* + *recpower*
class *recpower-semiring-0* = *semiring-0* + *recpower*
class *recpower-ring* = *ring* + *recpower*
class *recpower-ring-1* = *ring-1* + *recpower*
subclass (**in** *recpower-ring-1*) *recpower-ring* **by** *unfold-locales*
class *recpower-comm-semiring-1* = *recpower* + *comm-semiring-1*
class *recpower-comm-ring-1* = *recpower* + *comm-ring-1*
subclass (**in** *recpower-comm-ring-1*) *recpower-comm-semiring-1* **by** *unfold-locales*
class *recpower-idom* = *recpower* + *idom*
subclass (**in** *recpower-idom*) *recpower-comm-ring-1* **by** *unfold-locales*
class *idom-char-0* = *idom* + *ring-char-0*
class *recpower-idom-char-0* = *recpower* + *idom-char-0*
subclass (**in** *recpower-idom-char-0*) *recpower-idom* **by** *unfold-locales*

lemma (**in** *recpower-comm-ring-1*) *poly-exp*: *poly* (*p* %^ *n*) *x* = (*poly p x*) ^ *n*
apply (*induct n*)
apply (*auto simp add*: *poly-cmult poly-mult power-Suc*)
done

More Polynomial Evaluation Lemmas

lemma (**in** *semiring-0*) *poly-add-rzero*[*simp*]: *poly* (*a* +++ []) *x* = *poly a x*
by *simp*

lemma (**in** *comm-semiring-0*) *poly-mult-assoc*: *poly* ((*a* *** *b*) *** *c*) *x* = *poly* (*a* *** (*b* *** *c*)) *x*
by (*simp add*: *poly-mult mult-assoc*)

lemma (**in** *semiring-0*) *poly-mult-Nil2*[*simp*]: *poly* (*p* *** []) *x* = *0*
by (*induct p*, *auto*)

lemma (**in** *comm-semiring-1*) *poly-exp-add*: *poly* (*p* %^ (*n* + *d*)) *x* = *poly*(*p* %^ *n* *** *p* %^ *d*) *x*
apply (*induct n*)
apply (*auto simp add*: *poly-mult mult-assoc*)
done

12.2 Key Property: if *f a* = (*0*::'*a*) then *x* − *a* divides *p x*

lemma (**in** *comm-ring-1*) *lemma-poly-linear-rem*: ∀ *h*. ∃ *q r*. *h*#*t* = [*r*] +++ [−*a*, *1*] *** *q*
proof(*induct t*)
case *Nil*
{fix *h* **have** [*h*] = [*h*] +++ [− *a*, *1*] *** [] **by** *simp***}**
thus *?case* **by** *blast*

next
case (*Cons x xs*)
{fix *h*
from *Cons.hyps*[*rule-format*, *of x*]
obtain *q r* **where** *qr*: *x*#*xs* = [*r*] +++ [− *a*, *1*] *** *q* **by** *blast*
have *h*#*x*#*xs* = [*a*∗*r* + *h*] +++ [−*a*, *1*] *** (*r*#*q*)
using *qr* **by**(*cases q*, *simp-all add*: *ring-simps diff-def*[*symmetric*]
minus-mult-left[*symmetric*] *right-minus*)
hence ∃ *q r*. *h*#*x*#*xs* = [*r*] +++ [−*a*, *1*] *** *q* **by** *blast***}**
thus *?case* **by** *blast*
qed

lemma (**in** *comm-ring-1*) *poly-linear-rem*: ∃ *q r*. *h*#*t* = [*r*] +++ [−*a*, *1*] *** *q*
by (*cut-tac t* = *t* **and** *a* = *a* **in** *lemma-poly-linear-rem*, *auto*)

lemma (**in** *comm-ring-1*) *poly-linear-divides*: (*poly p a* = *0*) = ((*p* = []) | (∃ *q*. *p* = [−*a*, *1*] *** *q*))
proof−
{assume *p*: *p* = [] **hence** *?thesis* **by** *simp***}**
moreover
{fix *x xs* **assume** *p*: *p* = *x*#*xs*
{fix *q* **assume** *p* = [−*a*, *1*] *** *q* **hence** *poly p a* = *0*
by (*simp add*: *poly-add poly-cmult minus-mult-left*[*symmetric*])**}**
moreover
{assume *p0*: *poly p a* = *0*
from *poly-linear-rem*[*of x xs a*] **obtain** *q r*
where *qr*: *x*#*xs* = [*r*] +++ [− *a*, *1*] *** *q* **by** *blast*
have *r* = *0* **using** *p0* **by** (*simp only*: *p qr poly-mult poly-add*) *simp*
hence ∃ *q*. *p* = [− *a*, *1*] *** *q* **using** *p qr* **apply** − **apply** (*rule exI*[**where** *x*=*q*])**apply** *auto* **apply** (*cases q*) **apply** *auto* **done}**
ultimately have *?thesis* **using** *p* **by** *blast***}**
ultimately show *?thesis* **by** (*cases p*, *auto*)
qed

lemma (**in** *semiring-0*) *lemma-poly-length-mult*[*simp*]: ∀ *h k a*. *length* (*k* %∗ *p* +++ (*h* # (*a* %∗ *p*))) = *Suc* (*length p*)
by (*induct p*, *auto*)

lemma (**in** *semiring-0*) *lemma-poly-length-mult2*[*simp*]: ∀ *h k*. *length* (*k* %∗ *p* +++ (*h* # *p*)) = *Suc* (*length p*)
by (*induct p*, *auto*)

lemma (**in** *ring-1*) *poly-length-mult*[*simp*]: *length*([−*a*,*1*] *** *q*) = *Suc* (*length q*)
by *auto*

12.3 Polynomial length

lemma (**in** *semiring-0*) *poly-cmult-length*[*simp*]: *length* (*a* %∗ *p*) = *length p*

by (*induct p*, *auto*)

lemma (**in** *semiring-0*) *poly-add-length*: *length* (*p1* +++ *p2*) = *max* (*length p1*) (*length p2*)
apply (*induct p1 arbitrary*: *p2*, *simp-all*)
apply *arith*
done

lemma (**in** *semiring-0*) *poly-root-mult-length*[*simp*]: *length*([*a*,*b*] *** *p*) = *Suc* (*length p*)
by (*simp add*: *poly-add-length*)

lemma (**in** *idom*) *poly-mult-not-eq-poly-Nil*[*simp*]:
poly (*p* *** *q*) *x* ≠ *poly* [] *x* ⟷ *poly p x* ≠ *poly* [] *x* ∧ *poly q x* ≠ *poly* [] *x*
by (*auto simp add*: *poly-mult*)

lemma (**in** *idom*) *poly-mult-eq-zero-disj*: *poly* (*p* *** *q*) *x* = *0* ⟷ *poly p x* = *0* ∨ *poly q x* = *0*
by (*auto simp add*: *poly-mult*)

Normalisation Properties

lemma (**in** *semiring-0*) *poly-normalized-nil*: (*pnormalize p* = []) −−> (*poly p x* = *0*)
by (*induct p*, *auto*)

A nontrivial polynomial of degree n has no more than n roots

lemma (**in** *idom*) *poly-roots-index-lemma*:
assumes *p*: *poly p x* ≠ *poly* [] *x* **and** *n*: *length p* = *n*
shows ∃ *i*. ∀ *x*. *poly p x* = *0* ⟶ (∃ *m*≤*n*. *x* = *i m*)
using *p n*
proof(*induct n arbitrary*: *p x*)
case *0* **thus** *?case* **by** *simp*
next
case (*Suc n p x*)
{assume *C*: ⋀*i*. ∃ *x*. *poly p x* = *0* ∧ (∀ *m*≤*Suc n*. *x* ≠ *i m*)
from *Suc.prems* **have** *p0*: *poly p x* ≠ *0 p*≠ [] **by** *auto*
from *p0*(*1*)[*unfolded poly-linear-divides*[*of p x*]]
have ∀ *q*. *p* ≠ [− *x*, *1*] *** *q* **by** *blast*
from *C* **obtain** *a* **where** *a*: *poly p a* = *0* **by** *blast*
from *a*[*unfolded poly-linear-divides*[*of p a*]] *p0*(*2*)
obtain *q* **where** *q*: *p* = [−*a*, *1*] *** *q* **by** *blast*
have *lg*: *length q* = *n* **using** *q Suc.prems*(*2*) **by** *simp*
from *q p0* **have** *qx*: *poly q x* ≠ *poly* [] *x*
by (*auto simp add*: *poly-mult poly-add poly-cmult*)
from *Suc.hyps*[*OF qx lg*] **obtain** *i* **where**
i: ∀ *x*. *poly q x* = *0* ⟶ (∃ *m*≤*n*. *x* = *i m*) **by** *blast*
let *?i* = λ*m*. *if m* = *Suc n then a else i m*
from *C*[*of ?i*] **obtain** *y* **where** *y*: *poly p y* = *0* ∀ *m*≤ *Suc n*. *y* ≠ *?i m*
by *blast*

```
    from y have y = a ∨ poly q y = 0
      by (simp only: q poly-mult-eq-zero-disj poly-add) (simp add: ring-simps)
    with i[rule-format, of y] y(1) y(2) have False apply auto
      apply (erule-tac x=m in allE)
      apply auto
      done}
  thus ?case by blast
qed


lemma (in idom) poly-roots-index-length: poly p x ≠ poly [] x ==>
      ∃ i. ∀ x. (poly p x = 0) --> (∃ n. n ≤ length p & x = i n)
by (blast intro: poly-roots-index-lemma)

lemma (in idom) poly-roots-finite-lemma1: poly p x ≠ poly [] x ==>
      ∃ N i. ∀ x. (poly p x = 0) --> (∃ n. (n::nat) < N & x = i n)
apply (drule poly-roots-index-length, safe)
apply (rule-tac x = Suc (length p) in exI)
apply (rule-tac x = i in exI)
apply (simp add: less-Suc-eq-le)
done


lemma (in idom) idom-finite-lemma:
  assumes P: ∀ x. P x --> (∃ n. n < length j & x = j!n)
  shows finite {x. P x}
proof-
  let ?M = {x. P x}
  let ?N = set j
  have ?M ⊆ ?N using P by auto
  thus ?thesis using finite-subset by auto
qed


lemma (in idom) poly-roots-finite-lemma2: poly p x ≠ poly [] x ==>
      ∃ i. ∀ x. (poly p x = 0) --> x ∈ set i
apply (drule poly-roots-index-length, safe)
apply (rule-tac x=map (λn. i n) [0 ..< Suc (length p)] in exI)
apply (auto simp add: image-iff)
apply (erule-tac x=x in allE, clarsimp)
by (case-tac n=length p, auto simp add: order-le-less)

lemma UNIV-nat-infinite: ¬ finite (UNIV :: nat set)
  unfolding finite-conv-nat-seg-image
proof(auto simp add: expand-set-eq image-iff)
  fix n::nat and f:: nat ⇒ nat
  let ?N = {i. i < n}
  let ?fN = f ' ?N
  let ?y = Max ?fN + 1
```

```
  from nat-seg-image-imp-finite[of ?fN f n]
  have thfN: finite ?fN by simp
  {assume n =0 hence ∃ x. ∀ xa<n. x ≠ f xa by auto}
  moreover
  {assume nz: n ≠ 0
    hence thne: ?fN ≠ {} by (auto simp add: neq0-conv)
    have ∀ x∈ ?fN. Max ?fN ≥ x using nz Max-ge-iff[OF thfN thne] by auto
    hence ∀ x∈ ?fN. ?y > x by auto
    hence ?y ∉ ?fN by auto
    hence ∃ x. ∀ xa<n. x ≠ f xa by auto }
  ultimately show ∃ x. ∀ xa<n. x ≠ f xa by blast
qed

lemma (in ring-char-0) UNIV-ring-char-0-infinte:
  ¬ (finite (UNIV:: 'a set))
proof
  assume F: finite (UNIV :: 'a set)
  have th0: of-nat ' UNIV ⊆ UNIV by simp
  from finite-subset[OF th0] have th: finite (of-nat ' UNIV :: 'a set) .
  have th': inj-on (of-nat::nat ⇒ 'a) (UNIV)
    unfolding inj-on-def by auto
  from finite-imageD[OF th th'] UNIV-nat-infinite
  show False by blast
qed

lemma (in idom-char-0) poly-roots-finite: (poly p ≠ poly []) =
  finite {x. poly p x = 0}
proof
  assume H: poly p ≠ poly []
  show finite {x. poly p x = (0::'a)}
    using H
    apply −
    apply (erule contrapos-np, rule ext)
    apply (rule ccontr)
    apply (clarify dest!: poly-roots-finite-lemma2)
    using finite-subset
  proof−
    fix x i
    assume F: ¬ finite {x. poly p x = (0::'a)}
      and P: ∀ x. poly p x = (0::'a) ⟶ x ∈ set i
    let ?M= {x. poly p x = (0::'a)}
    from P have ?M ⊆ set i by auto
    with finite-subset F show False by auto
  qed
next
  assume F: finite {x. poly p x = (0::'a)}
  show poly p ≠ poly [] using F UNIV-ring-char-0-infinte by auto
qed
```

Entirety and Cancellation for polynomials

lemma (**in** *idom-char-0*) *poly-entire-lemma2*:
  assumes *p0*: *poly p* ≠ *poly* [] **and** *q0*: *poly q* ≠ *poly* []
  shows *poly* (*p*∗∗∗*q*) ≠ *poly* []
proof−
  let *?S* = λ*p*. {*x*. *poly p x* = *0*}
  have *?S* (*p* ∗∗∗ *q*) = *?S p* ∪ *?S q* **by** (*auto simp add*: *poly-mult*)
  with *p0 q0* **show** *?thesis* **unfolding** *poly-roots-finite* **by** *auto*
qed

lemma (**in** *idom-char-0*) *poly-entire*:
  poly (*p* ∗∗∗ *q*) = *poly* [] ⟷ *poly p* = *poly* [] ∨ *poly q* = *poly* []
using *poly-entire-lemma2*[*of p q*]
by *auto* (*rule ext*, *simp add*: *poly-mult*)+

lemma (**in** *idom-char-0*) *poly-entire-neg*: (*poly* (*p* ∗∗∗ *q*) ≠ *poly* []) = ((*poly p* ≠ *poly* []) & (*poly q* ≠ *poly* []))
by (*simp add*: *poly-entire*)

lemma *fun-eq*: (*f* = *g*) = (∀ *x*. *f x* = *g x*)
by (*auto intro*!: *ext*)

lemma (**in** *comm-ring-1*) *poly-add-minus-zero-iff*: (*poly* (*p* +++ −− *q*) = *poly* []) = (*poly p* = *poly q*)
by (*auto simp add*: *ring-simps poly-add poly-minus-def fun-eq poly-cmult minus-mult-left*[*symmetric*])

lemma (**in** *comm-ring-1*) *poly-add-minus-mult-eq*: *poly* (*p* ∗∗∗ *q* +++ −−(*p* ∗∗∗ *r*)) = *poly* (*p* ∗∗∗ (*q* +++ −− *r*))
by (*auto simp add*: *poly-add poly-minus-def fun-eq poly-mult poly-cmult right-distrib minus-mult-left*[*symmetric*] *minus-mult-right*[*symmetric*])

subclass (**in** *idom-char-0*) *comm-ring-1* **by** *unfold-locales*
lemma (**in** *idom-char-0*) *poly-mult-left-cancel*: (*poly* (*p* ∗∗∗ *q*) = *poly* (*p* ∗∗∗ *r*)) = (*poly p* = *poly* [] | *poly q* = *poly r*)
proof−
  have *poly* (*p* ∗∗∗ *q*) = *poly* (*p* ∗∗∗ *r*) ⟷ *poly* (*p* ∗∗∗ *q* +++ −− (*p* ∗∗∗ *r*)) = *poly* [] **by** (*simp only*: *poly-add-minus-zero-iff*)
  also have . . . ⟷ *poly p* = *poly* [] | *poly q* = *poly r*
   by (*auto intro*: *ext simp add*: *poly-add-minus-mult-eq poly-entire poly-add-minus-zero-iff*)
  finally show *?thesis* **.**
qed

lemma (**in** *recpower-idom*) *poly-exp-eq-zero*[*simp*]:
    (*poly* (*p* %ˆ *n*) = *poly* []) = (*poly p* = *poly* [] & *n* ≠ *0*)
apply (*simp only*: *fun-eq add*: *all-simps* [*symmetric*])
apply (*rule arg-cong* [**where** *f* = *All*])
apply (*rule ext*)
apply (*induct n*)
apply (*auto simp add*: *poly-exp poly-mult*)
done

lemma (**in** *semiring-1*) *one-neq-zero*[*simp*]: *1 ≠ 0* **using** *zero-neq-one* **by** *blast*
lemma (**in** *comm-ring-1*) *poly-prime-eq-zero*[*simp*]: *poly* [*a*,*1*] ≠ *poly* []
apply (*simp add*: *fun-eq*)
apply (*rule-tac x* = *minus one a* **in** *exI*)
apply (*unfold diff-minus*)
apply (*subst add-commute*)
apply (*subst add-assoc*)
apply *simp*
done

lemma (**in** *recpower-idom*) *poly-exp-prime-eq-zero*: (*poly* ([*a*, *1*] % ^ *n*) ≠ *poly* [])
by *auto*

A more constructive notion of polynomials being trivial

lemma (**in** *idom-char-0*) *poly-zero-lemma'*: *poly* (*h* # *t*) = *poly* [] ==> *h* = *0* & *poly t* = *poly* []
apply(*simp add*: *fun-eq*)
apply (*case-tac h* = *zero*)
apply (*drule-tac* [*2*] *x* = *zero* **in** *spec*, *auto*)
apply (*cases poly t* = *poly* [], *simp*)
proof−
fix *x*
assume *H*: ∀ *x*. *x* = (*0*::'*a*) ∨ *poly t x* = (*0*::'*a*) **and** *pnz*: *poly t* ≠ *poly* []
let *?S* = {*x*. *poly t x* = *0*}
from *H* **have** ∀ *x*. *x* ≠*0* ⟶ *poly t x* = *0* **by** *blast*
hence *th*: *?S* ⊇ *UNIV* − {*0*} **by** *auto*
from *poly-roots-finite pnz* **have** *th'*: *finite ?S* **by** *blast*
from *finite-subset*[*OF th th'*] *UNIV-ring-char-0-infinte*
show *poly t x* = (*0*::'*a*) **by** *simp*
qed

lemma (**in** *idom-char-0*) *poly-zero*: (*poly p* = *poly* []) = *list-all* (%*c*. *c* = *0*) *p*
apply (*induct p*, *simp*)
apply (*rule iffI*)
apply (*drule poly-zero-lemma'*, *auto*)
done

lemma (**in** *idom-char-0*) *poly-0*: *list-all* (λ*c*. *c* = *0*) *p* ⟹ *poly p x* = *0*
unfolding *poly-zero*[*symmetric*] **by** *simp*

Basics of divisibility.

lemma (**in** *idom*) *poly-primes*: ([*a*, *1*] *divides* (*p* ∗∗∗ *q*)) = ([*a*, *1*] *divides p* | [*a*, *1*] *divides q*)
apply (*auto simp add*: *divides-def fun-eq poly-mult poly-add poly-cmult left-distrib* [*symmetric*])
apply (*drule-tac x* = *uminus a* **in** *spec*)
apply (*simp add*: *poly-linear-divides poly-add poly-cmult left-distrib* [*symmetric*])
apply (*cases p* = [])

apply (*rule exI*[**where** *x*=[]])
apply *simp*
apply (*cases q* = [])
apply (*erule allE*[**where** *x*=[]], *simp*)

apply *clarsimp*
apply (*cases* ∃ *q*::'*a list*. *p* = *a* %* *q* +++ ((*0*::'*a*) # *q*))
apply (*clarsimp simp add*: *poly-add poly-cmult*)
apply (*rule-tac x*=*qa* **in** *exI*)
apply (*simp add*: *left-distrib* [*symmetric*])
apply *clarsimp*

apply (*auto simp add*: *right-minus poly-linear-divides poly-add poly-cmult left-distrib* [*symmetric*])
apply (*rule-tac x* = *pmult qa q* **in** *exI*)
apply (*rule-tac* [*2*] *x* = *pmult p qa* **in** *exI*)
apply (*auto simp add*: *poly-add poly-mult poly-cmult mult-ac*)
done

lemma (**in** *comm-semiring-1*) *poly-divides-refl*[*simp*]: *p divides p*
apply (*simp add*: *divides-def*)
apply (*rule-tac x* = [*one*] **in** *exI*)
apply (*auto simp add*: *poly-mult fun-eq*)
done

lemma (**in** *comm-semiring-1*) *poly-divides-trans*: [| *p divides q*; *q divides r* |] ==> *p divides r*
apply (*simp add*: *divides-def*, *safe*)
apply (*rule-tac x* = *pmult qa qaa* **in** *exI*)
apply (*auto simp add*: *poly-mult fun-eq mult-assoc*)
done

lemma (**in** *recpower-comm-semiring-1*) *poly-divides-exp*: *m* ≤ *n* ==> (*p* %^ *m*) *divides* (*p* %^ *n*)
apply (*auto simp add*: *le-iff-add*)
apply (*induct-tac k*)
apply (*rule-tac* [*2*] *poly-divides-trans*)
apply (*auto simp add*: *divides-def*)
apply (*rule-tac x* = *p* **in** *exI*)
apply (*auto simp add*: *poly-mult fun-eq mult-ac*)
done

lemma (**in** *recpower-comm-semiring-1*) *poly-exp-divides*: [| (*p* %^ *n*) *divides q*; *m*≤*n* |] ==> (*p* %^ *m*) *divides q*
by (*blast intro*: *poly-divides-exp poly-divides-trans*)

lemma (**in** *comm-semiring-0*) *poly-divides-add*:
[| *p divides q*; *p divides r* |] ==> *p divides* (*q* +++ *r*)

apply (*simp add*: *divides-def*, *auto*)
apply (*rule-tac x* = *padd qa qaa* **in** *exI*)
apply (*auto simp add*: *poly-add fun-eq poly-mult right-distrib*)
done

lemma (**in** *comm-ring-1*) *poly-divides-diff*:
[| *p divides q*; *p divides* (*q* +++ *r*) |] ==> *p divides r*
apply (*simp add*: *divides-def*, *auto*)
apply (*rule-tac x* = *padd qaa* (*poly-minus qa*) **in** *exI*)
apply (*auto simp add*: *poly-add fun-eq poly-mult poly-minus right-diff-distrib compare-rls add-ac*)
done

lemma (**in** *comm-ring-1*) *poly-divides-diff2*: [| *p divides r*; *p divides* (*q* +++ *r*) |] ==> *p divides q*
apply (*erule poly-divides-diff*)
apply (*auto simp add*: *poly-add fun-eq poly-mult divides-def add-ac*)
done

lemma (**in** *semiring-0*) *poly-divides-zero*: *poly p* = *poly* [] ==> *q divides p*
apply (*simp add*: *divides-def*)
apply (*rule exI*[**where** *x*=[]])
apply (*auto simp add*: *fun-eq poly-mult*)
done

lemma (**in** *semiring-0*) *poly-divides-zero2*[*simp*]: *q divides* []
apply (*simp add*: *divides-def*)
apply (*rule-tac x* = [] **in** *exI*)
apply (*auto simp add*: *fun-eq*)
done

At last, we can consider the order of a root.

lemma (**in** *idom-char-0*) *poly-order-exists-lemma*:
assumes *lp*: *length p* = *d* **and** *p*: *poly p* ≠ *poly* []
shows ∃ *n q*. *p* = *mulexp n* [−*a*, *1*] *q* ∧ *poly q a* ≠ *0*
using *lp p*
proof(*induct d arbitrary*: *p*)
case *0* **thus** *?case* **by** *simp*
next
case (*Suc n p*)
{assume *p0*: *poly p a* = *0*
from *Suc.prems* **have** *h*: *length p* = *Suc n poly p* ≠ *poly* [] **by** *blast*
hence *pN*: *p* ≠ [] **by** − (*rule ccontr*, *simp*)
from *p0*[*unfolded poly-linear-divides*] *pN* **obtain** *q* **where**
q: *p* = [−*a*, *1*] *** *q* **by** *blast*
from *q h p0* **have** *qh*: *length q* = *n poly q* ≠ *poly* []
apply −
apply *simp*
apply (*simp only*: *fun-eq*)

apply (*rule ccontr*)
apply (*simp add*: *fun-eq poly-add poly-cmult minus-mult-left*[*symmetric*])
done
from *Suc.hyps*[*OF qh*] **obtain** *m r* **where**
mr: *q* = *mulexp m* [−*a*,*1*] *r poly r a* ≠ *0* **by** *blast*
from *mr q* **have** *p* = *mulexp* (*Suc m*) [−*a*,*1*] *r* ∧ *poly r a* ≠ *0* **by** *simp*
hence *?case* **by** *blast***}**
moreover
{assume *p0*: *poly p a* ≠ *0*
hence *?case* **using** *Suc.prems* **apply** *simp* **by** (*rule exI*[**where** *x*=*0*::*nat*], *simp*)**}**
ultimately show *?case* **by** *blast*
qed

lemma (**in** *recpower-comm-semiring-1*) *poly-mulexp*: *poly* (*mulexp n p q*) *x* = (*poly p x*) ^ *n* ∗ *poly q x*
by(*induct n*, *auto simp add*: *poly-mult power-Suc mult-ac*)

lemma (**in** *comm-semiring-1*) *divides-left-mult*:
assumes *d*:(*p*∗∗∗*q*) *divides r* **shows** *p divides r* ∧ *q divides r*
proof−
from *d* **obtain** *t* **where** *r*:*poly r* = *poly* (*p*∗∗∗*q* ∗∗∗ *t*)
unfolding *divides-def* **by** *blast*
hence *poly r* = *poly* (*p* ∗∗∗ (*q* ∗∗∗ *t*))
poly r = *poly* (*q* ∗∗∗ (*p*∗∗∗*t*)) **by**(*auto simp add*: *fun-eq poly-mult mult-ac*)
thus *?thesis* **unfolding** *divides-def* **by** *blast*
qed

lemma (**in** *recpower-semiring-1*)
zero-power-iff: *0* ^ *n* = (*if n* = *0 then 1 else 0*)
by (*induct n*, *simp-all add*: *power-Suc*)

lemma (**in** *recpower-idom-char-0*) *poly-order-exists*:
assumes *lp*: *length p* = *d* **and** *p0*: *poly p* ≠ *poly* []
shows ∃*n*. ([−*a*, *1*] %^ *n*) *divides p* & ~(([−*a*, *1*] %^ (*Suc n*)) *divides p*)
proof−
let *?poly* = *poly*
let *?mulexp* = *mulexp*
let *?pexp* = *pexp*
from *lp p0*
show *?thesis*
apply −
apply (*drule poly-order-exists-lemma* [**where** *a*=*a*], *assumption*, *clarify*)
apply (*rule-tac x* = *n* **in** *exI*, *safe*)

apply (*unfold divides-def*)
apply (*rule-tac x = q* **in** *exI*)
apply (*induct-tac n, simp*)
apply (*simp* (*no-asm-simp*) *add*: *poly-add poly-cmult poly-mult right-distrib mult-ac*)
apply *safe*
apply (*subgoal-tac ?poly* (*?mulexp n* [*uminus a, one*] *q*) ≠ *?poly* (*pmult* (*?pexp* [*uminus a, one*] (*Suc n*)) *qa*))
apply *simp*
apply (*induct-tac n*)
apply (*simp del*: *pmult-Cons pexp-Suc*)
apply (*erule-tac Q = ?poly q a = zero* **in** *contrapos-np*)
apply (*simp add*: *poly-add poly-cmult minus-mult-left*[*symmetric*])
apply (*rule pexp-Suc* [*THEN ssubst*])
apply (*rule ccontr*)
apply (*simp add*: *poly-mult-left-cancel poly-mult-assoc del*: *pmult-Cons pexp-Suc*)
done
qed

lemma (**in** *semiring-1*) *poly-one-divides*[*simp*]: [*1*] *divides p*
by (*simp add*: *divides-def, auto*)

lemma (**in** *recpower-idom-char-0*) *poly-order*: *poly p* ≠ *poly* []
==> *EX*! *n*. ([−*a, 1*] % ^ *n*) *divides p* &
~(([−*a, 1*] % ^ (*Suc n*)) *divides p*)
apply (*auto intro*: *poly-order-exists simp add*: *less-linear simp del*: *pmult-Cons pexp-Suc*)
apply (*cut-tac x = y* **and** *y = n* **in** *less-linear*)
apply (*drule-tac m = n* **in** *poly-exp-divides*)
apply (*auto dest*: *Suc-le-eq* [*THEN iffD2, THEN* [*2*] *poly-exp-divides*]
simp del: *pmult-Cons pexp-Suc*)
done

Order

lemma *some1-equalityD*: [| *n* = (@*n*. *P n*); *EX*! *n*. *P n* |] ==> *P n*
by (*blast intro*: *someI2*)

lemma (**in** *recpower-idom-char-0*) *order*:
(([−*a, 1*] % ^ *n*) *divides p* &
~(([−*a, 1*] % ^ (*Suc n*)) *divides p*)) =
((*n* = *order a p*) & ~(*poly p* = *poly* []))
apply (*unfold order-def*)
apply (*rule iffI*)
apply (*blast dest*: *poly-divides-zero intro*!: *some1-equality* [*symmetric*] *poly-order*)
apply (*blast intro*!: *poly-order* [*THEN* [*2*] *some1-equalityD*])
done

lemma (**in** *recpower-idom-char-0*) *order2*: [| *poly p* ≠ *poly* [] |]
==> ([−*a, 1*] % ^ (*order a p*)) *divides p* &

```
          ~(([−a, 1] %^ (Suc(order a p))) divides p)
by (simp add: order del: pexp-Suc)

lemma (in recpower-idom-char-0) order-unique: [| poly p ≠ poly []; ([−a, 1] %^
n) divides p;
        ~(([−a, 1] %^ (Suc n)) divides p)
     |] ==> (n = order a p)
by (insert order [of a n p], auto)

lemma (in recpower-idom-char-0) order-unique-lemma: (poly p ≠ poly [] & ([−a,
1] %^ n) divides p &
        ~(([−a, 1] %^ (Suc n)) divides p))
     ==> (n = order a p)
by (blast intro: order-unique)

lemma (in ring-1) order-poly: poly p = poly q ==> order a p = order a q
by (auto simp add: fun-eq divides-def poly-mult order-def)

lemma (in semiring-1) pexp-one[simp]: p %^ (Suc 0) = p
apply (induct p)
apply (auto simp add: numeral-1-eq-1)
done

lemma (in comm-ring-1) lemma-order-root:
     0 < n & [− a, 1] %^ n divides p & ~ [− a, 1] %^ (Suc n) divides p
          ⟹ poly p a = 0
apply (induct n arbitrary: a p, blast)
apply (auto simp add: divides-def poly-mult simp del: pmult-Cons)
done

lemma (in recpower-idom-char-0) order-root: (poly p a = 0) = ((poly p = poly
[]) | order a p ≠ 0)
proof−
  let ?poly = poly
  show ?thesis
apply (case-tac ?poly p = ?poly [], auto)
apply (simp add: poly-linear-divides del: pmult-Cons, safe)
apply (drule-tac [!] a = a in order2)
apply (rule ccontr)
apply (simp add: divides-def poly-mult fun-eq del: pmult-Cons, blast)
using neq0-conv
apply (blast intro: lemma-order-root)
done
qed

lemma (in recpower-idom-char-0) order-divides: (([−a, 1] %^ n) divides p) =
((poly p = poly []) | n ≤ order a p)
proof−
  let ?poly = poly
```

show *?thesis*
apply (*case-tac ?poly p = ?poly* [], *auto*)
apply (*simp add*: *divides-def fun-eq poly-mult*)
apply (*rule-tac x* = [] **in** *exI*)
apply (*auto dest*!: *order2* [**where** *a*=*a*]
intro: *poly-exp-divides simp del*: *pexp-Suc*)
done
qed

lemma (**in** *recpower-idom-char-0*) *order-decomp*:
poly p ≠ *poly* []
==> ∃ *q*. (*poly p* = *poly* (([−*a*, *1*] % ^ (*order a p*)) *** *q*)) &
~([−*a*, *1*] *divides q*)
apply (*unfold divides-def*)
apply (*drule order2* [**where** *a* = *a*])
apply (*simp add*: *divides-def del*: *pexp-Suc pmult-Cons*, *safe*)
apply (*rule-tac x* = *q* **in** *exI*, *safe*)
apply (*drule-tac x* = *qa* **in** *spec*)
apply (*auto simp add*: *poly-mult fun-eq poly-exp mult-ac simp del*: *pmult-Cons*)
done

Important composition properties of orders.

lemma *order-mult*: *poly* (*p* *** *q*) ≠ *poly* []
==> *order a* (*p* *** *q*) = *order a p* + *order* (*a*::'*a*::{*recpower-idom-char-0*})
q
apply (*cut-tac a* = *a* **and** *p* = *p* *** *q* **and** *n* = *order a p* + *order a q* **in** *order*)
apply (*auto simp add*: *poly-entire simp del*: *pmult-Cons*)
apply (*drule-tac a* = *a* **in** *order2*)+
apply *safe*
apply (*simp add*: *divides-def fun-eq poly-exp-add poly-mult del*: *pmult-Cons*, *safe*)
apply (*rule-tac x* = *qa* *** *qaa* **in** *exI*)
apply (*simp add*: *poly-mult mult-ac del*: *pmult-Cons*)
apply (*drule-tac a* = *a* **in** *order-decomp*)+
apply *safe*
apply (*subgoal-tac* [−*a*,*1*] *divides* (*qa* *** *qaa*))
apply (*simp add*: *poly-primes del*: *pmult-Cons*)
apply (*auto simp add*: *divides-def simp del*: *pmult-Cons*)
apply (*rule-tac x* = *qb* **in** *exI*)
apply (*subgoal-tac poly* (([−*a*, *1*] % ^ (*order a p*)) *** (*qa* *** *qaa*)) = *poly* (([−*a*, *1*] % ^ (*order a p*)) *** ([−*a*, *1*] *** *qb*)))
apply (*drule poly-mult-left-cancel* [*THEN iffD1*], *force*)
apply (*subgoal-tac poly* (([−*a*, *1*] % ^ (*order a q*)) *** ([−*a*, *1*] % ^ (*order a p*)) *** (*qa* *** *qaa*))) = *poly* (([−*a*, *1*] % ^ (*order a q*)) *** ([−*a*, *1*] % ^ (*order a p*)) *** ([−*a*, *1*] *** *qb*))))
apply (*drule poly-mult-left-cancel* [*THEN iffD1*], *force*)
apply (*simp add*: *fun-eq poly-exp-add poly-mult mult-ac del*: *pmult-Cons*)
done

lemma (**in** *recpower-idom-char-0*) *order-mult*:

assumes *pq0*: *poly* (*p* *** *q*) ≠ *poly* []
shows *order a* (*p* *** *q*) = *order a p* + *order a q*
proof−
let *?order* = *order*
let *?divides* = *op divides*
let *?poly* = *poly*
from *pq0*
show *?thesis*
apply (*cut-tac a* = *a* **and** *p* = *pmult p q* **and** *n* = *?order a p* + *?order a q* **in** *order*)
apply (*auto simp add*: *poly-entire simp del*: *pmult-Cons*)
apply (*drule-tac a* = *a* **in** *order2*)+
apply *safe*
apply (*simp add*: *divides-def fun-eq poly-exp-add poly-mult del*: *pmult-Cons*, *safe*)
apply (*rule-tac x* = *pmult qa qaa* **in** *exI*)
apply (*simp add*: *poly-mult mult-ac del*: *pmult-Cons*)
apply (*drule-tac a* = *a* **in** *order-decomp*)+
apply *safe*
apply (*subgoal-tac ?divides* [*uminus a*,*one*] (*pmult qa qaa*))
apply (*simp add*: *poly-primes del*: *pmult-Cons*)
apply (*auto simp add*: *divides-def simp del*: *pmult-Cons*)
apply (*rule-tac x* = *qb* **in** *exI*)
apply (*subgoal-tac ?poly* (*pmult* (*pexp* [*uminus a*, *one*] (*?order a p*)) (*pmult qa qaa*)) = *?poly* (*pmult* (*pexp* [*uminus a*, *one*] (*?order a p*)) (*pmult* [*uminus a*, *one*] *qb*)))
apply (*drule poly-mult-left-cancel* [*THEN iffD1*], *force*)
apply (*subgoal-tac ?poly* (*pmult* (*pexp* [*uminus a*, *one*] (*order a q*)) (*pmult* (*pexp* [*uminus a*, *one*] (*order a p*)) (*pmult qa qaa*))) = *?poly* (*pmult* (*pexp* [*uminus a*, *one*] (*order a q*)) (*pmult* (*pexp* [*uminus a*, *one*] (*order a p*)) (*pmult* [*uminus a*, *one*] *qb*))))
apply (*drule poly-mult-left-cancel* [*THEN iffD1*], *force*)
apply (*simp add*: *fun-eq poly-exp-add poly-mult mult-ac del*: *pmult-Cons*)
done
qed

lemma (**in** *recpower-idom-char-0*) *order-root2*: *poly p* ≠ *poly* [] ==> (*poly p a* = *0*) = (*order a p* ≠ *0*)
by (*rule order-root* [*THEN ssubst*], *auto*)

lemma (**in** *semiring-1*) *pmult-one*[*simp*]: [*1*] *** *p* = *p* **by** *auto*

lemma (**in** *semiring-0*) *poly-Nil-zero*: *poly* [] = *poly* [*0*]
by (*simp add*: *fun-eq*)

lemma (**in** *recpower-idom-char-0*) *rsquarefree-decomp*:
[| *rsquarefree p*; *poly p a* = *0* |]
==> ∃ *q*. (*poly p* = *poly* ([−*a*, *1*] *** *q*)) & *poly q a* ≠ *0*
apply (*simp add*: *rsquarefree-def*, *safe*)
apply (*frule-tac a* = *a* **in** *order-decomp*)

apply (*drule-tac x = a* **in** *spec*)
apply (*drule-tac a = a* **in** *order-root2* [*symmetric*])
apply (*auto simp del*: *pmult-Cons*)
apply (*rule-tac x = q* **in** *exI*, *safe*)
apply (*simp add*: *poly-mult fun-eq*)
apply (*drule-tac p1 = q* **in** *poly-linear-divides* [*THEN iffD1*])
apply (*simp add*: *divides-def del*: *pmult-Cons*, *safe*)
apply (*drule-tac x = []* **in** *spec*)
apply (*auto simp add*: *fun-eq*)
done

Normalization of a polynomial.

lemma (**in** *semiring-0*) *poly-normalize*[*simp*]: *poly* (*pnormalize p*) = *poly p*
apply (*induct p*)
apply (*auto simp add*: *fun-eq*)
done

The degree of a polynomial.

lemma (**in** *semiring-0*) *lemma-degree-zero*:
    list-all (%*c*. *c = 0*) *p* ⟷ *pnormalize p = []*
by (*induct p*, *auto*)

lemma (**in** *idom-char-0*) *degree-zero*:
  assumes *pN*: *poly p = poly []* **shows***degree p = 0*
proof−
  let *?pn = pnormalize*
  from *pN*
  show *?thesis*
    apply (*simp add*: *degree-def*)
    apply (*case-tac ?pn p = []*)
    apply (*auto simp add*: *poly-zero lemma-degree-zero*)
    done
qed

lemma (**in** *semiring-0*) *pnormalize-sing*: (*pnormalize* [*x*] = [*x*]) ⟷ *x ≠ 0* **by** *simp*
lemma (**in** *semiring-0*) *pnormalize-pair*: *y ≠ 0* ⟷ (*pnormalize* [*x*, *y*] = [*x*, *y*])
by *simp*
lemma (**in** *semiring-0*) *pnormal-cons*: *pnormal p* ⟹ *pnormal* (*c*#*p*)
  unfolding *pnormal-def* **by** *simp*
lemma (**in** *semiring-0*) *pnormal-tail*: *p≠[]* ⟹ *pnormal* (*c*#*p*) ⟹ *pnormal p*
  unfolding *pnormal-def*
  apply (*cases pnormalize p = []*, *auto*)
  by (*cases c = 0*, *auto*)

lemma (**in** *semiring-0*) *pnormal-last-nonzero*: *pnormal p* ==> *last p ≠ 0*
proof(*induct p*)
  case *Nil* **thus** *?case* **by** (*simp add*: *pnormal-def*)

next
case (*Cons a as*) **thus** *?case*
apply (*simp add*: *pnormal-def*)
apply (*cases pnormalize as = []*, *simp-all*)
apply (*cases as = []*, *simp-all*)
apply (*cases a=0*, *simp-all*)
apply (*cases a=0*, *simp-all*)
done
qed

lemma (**in** *semiring-0*) *pnormal-length*: *pnormal p ⟹ 0 < length p*
unfolding *pnormal-def length-greater-0-conv* **by** *blast*

lemma (**in** *semiring-0*) *pnormal-last-length*: *⟦0 < length p ; last p ≠ 0⟧ ⟹ pnormal p*
apply (*induct p*, *auto*)
apply (*case-tac p = []*, *auto*)
apply (*simp add*: *pnormal-def*)
by (*rule pnormal-cons*, *auto*)

lemma (**in** *semiring-0*) *pnormal-id*: *pnormal p ⟷ (0 < length p ∧ last p ≠ 0)*
using *pnormal-last-length pnormal-length pnormal-last-nonzero* **by** *blast*

lemma (**in** *idom-char-0*) *poly-Cons-eq*: *poly (c#cs) = poly (d#ds) ⟷ c=d ∧ poly cs = poly ds* (**is** *?lhs ⟷ ?rhs*)
proof
assume *eq*: *?lhs*
hence *⋀x. poly ((c#cs) +++ −− (d#ds)) x = 0*
by (*simp only*: *poly-minus poly-add ring-simps*) *simp*
hence *poly ((c#cs) +++ −− (d#ds)) = poly []* **by** *−* (*rule ext*, *simp*)
hence *c = d ∧ list-all (λx. x=0) ((cs +++ −− ds))*
unfolding *poly-zero* **by** (*simp add*: *poly-minus-def ring-simps minus-mult-left*[*symmetric*])
hence *c = d ∧ (∀x. poly (cs +++ −− ds) x = 0)*
unfolding *poly-zero*[*symmetric*] **by** *simp*
thus *?rhs* **apply** (*simp add*: *poly-minus poly-add ring-simps*) **apply** (*rule ext*, *simp*) **done**
next
assume *?rhs* **then show** *?lhs* **by** *−* (*rule ext*,*simp*)
qed

lemma (**in** *idom-char-0*) *pnormalize-unique*: *poly p = poly q ⟹ pnormalize p = pnormalize q*
proof(*induct q arbitrary*: *p*)
case *Nil* **thus** *?case* **by** (*simp only*: *poly-zero lemma-degree-zero*) *simp*
next
case (*Cons c cs p*)
thus *?case*
proof(*induct p*)
case *Nil*

hence *poly* [] = *poly* (*c*#*cs*) **by** *blast*
then have *poly* (*c*#*cs*) = *poly* [] **by** *simp*
thus *?case* **by** (*simp only*: *poly-zero lemma-degree-zero*) *simp*
next
case (*Cons d ds*)
hence *eq*: *poly* (*d* # *ds*) = *poly* (*c* # *cs*) **by** *blast*
hence *eq'*: ⋀*x*. *poly* (*d* # *ds*) *x* = *poly* (*c* # *cs*) *x* **by** *simp*
hence *poly* (*d* # *ds*) *0* = *poly* (*c* # *cs*) *0* **by** *blast*
hence *dc*: *d* = *c* **by** *auto*
with *eq* **have** *poly ds* = *poly cs*
unfolding *poly-Cons-eq* **by** *simp*
with *Cons.prems* **have** *pnormalize ds* = *pnormalize cs* **by** *blast*
with *dc* **show** *?case* **by** *simp*
qed
qed

lemma (**in** *idom-char-0*) *degree-unique*: **assumes** *pq*: *poly p* = *poly q*
shows *degree p* = *degree q*
using *pnormalize-unique*[*OF pq*] **unfolding** *degree-def* **by** *simp*

lemma (**in** *semiring-0*) *pnormalize-length*: *length* (*pnormalize p*) ≤ *length p* **by** (*induct p*, *auto*)

lemma (**in** *semiring-0*) *last-linear-mul-lemma*:
last ((*a* %* *p*) +++ (*x*#(*b* %* *p*))) = (*if p*=[] *then x else b*∗*last p*)

apply (*induct p arbitrary*: *a x b*, *auto*)
apply (*subgoal-tac padd* (*cmult aa p*) (*times b a* # *cmult b p*) ≠ [], *simp*)
apply (*induct-tac p*, *auto*)
done

lemma (**in** *semiring-1*) *last-linear-mul*: **assumes** *p*:*p*≠[] **shows** *last* ([*a*,*1*] *** *p*) = *last p*
proof−
from *p* **obtain** *c cs* **where** *cs*: *p* = *c*#*cs* **by** (*cases p*, *auto*)
from *cs* **have** *eq*:[*a*,*1*] *** *p* = (*a* %* (*c*#*cs*)) +++ (*0*#(*1* %* (*c*#*cs*)))
by (*simp add*: *poly-cmult-distr*)
show *?thesis* **using** *cs*
unfolding *eq last-linear-mul-lemma* **by** *simp*
qed

lemma (**in** *semiring-0*) *pnormalize-eq*: *last p* ≠ *0* ⟹ *pnormalize p* = *p*
apply (*induct p*, *auto*)
apply (*case-tac p*, *auto*)+
done

lemma (**in** *semiring-0*) *last-pnormalize*: *pnormalize p* ≠ [] ⟹ *last* (*pnormalize p*) ≠ *0*
by (*induct p*, *auto*)

lemma (**in** *semiring-0*) *pnormal-degree*: *last p ≠ 0 ⟹ degree p = length p − 1*
using *pnormalize-eq*[*of p*] **unfolding** *degree-def* **by** *simp*

lemma (**in** *semiring-0*) *poly-Nil-ext*: *poly* [] = (*λx. 0*) **by** (*rule ext*) *simp*

lemma (**in** *idom-char-0*) *linear-mul-degree*: **assumes** *p*: *poly p ≠ poly* []
shows *degree* ([*a*,*1*] ***** *p*) = *degree p + 1*
proof−
from *p* **have** *pnz*: *pnormalize p ≠* []
unfolding *poly-zero lemma-degree-zero* **.**

from *last-linear-mul*[*OF pnz*, *of a*] *last-pnormalize*[*OF pnz*]
have *l0*: *last* ([*a*, *1*] ***** *pnormalize p*) *≠ 0* **by** *simp*
from *last-pnormalize*[*OF pnz*] *last-linear-mul*[*OF pnz*, *of a*]
pnormal-degree[*OF l0*] *pnormal-degree*[*OF last-pnormalize*[*OF pnz*]] *pnz*

have *th*: *degree* ([*a*,*1*] ***** *pnormalize p*) = *degree* (*pnormalize p*) + *1*
by (*auto simp add*: *poly-length-mult*)

have *eqs*: *poly* ([*a*,*1*] ***** *pnormalize p*) = *poly* ([*a*,*1*] ***** *p*)
by (*rule ext*) (*simp add*: *poly-mult poly-add poly-cmult*)
from *degree-unique*[*OF eqs*] *th*
show *?thesis* **by** (*simp add*: *degree-unique*[*OF poly-normalize*])
qed

lemma (**in** *idom-char-0*) *linear-pow-mul-degree*:
degree([*a*,*1*] *%^n *** p*) = (*if poly p = poly* [] *then 0 else degree p + n*)
proof(*induct n arbitrary*: *a p*)
case (*0 a p*)
{assume *p*: *poly p = poly* []
hence *?case* **using** *degree-unique*[*OF p*] **by** (*simp add*: *degree-def*)**}**
moreover
{assume *p*: *poly p ≠ poly* [] **hence** *?case* **by** (*auto simp add*: *poly-Nil-ext*) **}**
ultimately show *?case* **by** *blast*
next
case (*Suc n a p*)
have *eq*: *poly* ([*a*,*1*] *%^*(*Suc n*) ***** *p*) = *poly* ([*a*,*1*] *%^ n **** ([*a*,*1*] ***** *p*))
apply (*rule ext*, *simp add*: *poly-mult poly-add poly-cmult*)
by (*simp add*: *mult-ac add-ac right-distrib*)
note *deq* = *degree-unique*[*OF eq*]
{assume *p*: *poly p = poly* []
with *eq* **have** *eq′*: *poly* ([*a*,*1*] *%^*(*Suc n*) ***** *p*) = *poly* []
by − (*rule ext*,*simp add*: *poly-mult poly-cmult poly-add*)
from *degree-unique*[*OF eq′*] *p* **have** *?case* **by** (*simp add*: *degree-def*)**}**
moreover
{assume *p*: *poly p ≠ poly* []
from *p* **have** *ap*: *poly* ([*a*,*1*] ***** *p*) *≠ poly* []

using *poly-mult-not-eq-poly-Nil* **unfolding** *poly-entire* **by** *auto*
have *eq*: *poly* ([*a*,*1*] %^(*Suc n*) *** *p*) = *poly* ([*a*,*1*]%^*n* *** ([*a*,*1*] *** *p*))
by (*rule ext*, *simp add*: *poly-mult poly-add poly-exp poly-cmult mult-ac add-ac right-distrib*)
from *ap* **have** *ap'*: (*poly* ([*a*,*1*] *** *p*) = *poly* []) = *False* **by** *blast*
have *th0*: *degree* ([*a*,*1*]%^*n* *** ([*a*,*1*] *** *p*)) = *degree* ([*a*,*1*] *** *p*) + *n*
apply (*simp only*: *Suc.hyps*[*of a pmult* [*a*,*one*] *p*] *ap'*)
by *simp*

from *degree-unique*[*OF eq*] *ap p th0 linear-mul-degree*[*OF p*, *of a*]
have *?case* **by** (*auto simp del*: *poly.simps*)**}**
ultimately show *?case* **by** *blast*
qed

lemma (**in** *recpower-idom-char-0*) *order-degree*:
assumes *p0*: *poly p* ≠ *poly* []
shows *order a p* ≤ *degree p*
proof−
from *order2*[*OF p0*, *unfolded divides-def*]
obtain *q* **where** *q*: *poly p* = *poly* ([− *a*, *1*]%^ (*order a p*) *** *q*) **by** *blast*
{assume *poly q* = *poly* []
with *q p0* **have** *False* **by** (*simp add*: *poly-mult poly-entire*)**}**
with *degree-unique*[*OF q*, *unfolded linear-pow-mul-degree*]
show *?thesis* **by** *auto*
qed

Tidier versions of finiteness of roots.

lemma (**in** *idom-char-0*) *poly-roots-finite-set*: *poly p* ≠ *poly* [] ==> *finite* {*x*. *poly p x* = *0*}
unfolding *poly-roots-finite* **.**

bound for polynomial.

lemma *poly-mono*: *abs*(*x*) ≤ *k* ==> *abs*(*poly p* (*x*::'*a*::{*ordered-idom*})) ≤ *poly* (*map abs p*) *k*
apply (*induct p*, *auto*)
apply (*rule-tac y* = *abs a* + *abs* (*x* ∗ *poly p x*) **in** *order-trans*)
apply (*rule abs-triangle-ineq*)
apply (*auto intro*!: *mult-mono simp add*: *abs-mult*)
done

lemma (**in** *semiring-0*) *poly-Sing*: *poly* [*c*] *x* = *c* **by** *simp*

end

13 Dense-Linear-Order: Dense linear order without endpoints and a quantifier elimination procedure in Ferrante and Rackoff style

theory *Dense-Linear-Order*
imports *Arith-Tools*
uses
~~/*src*/*HOL*/*Tools*/*Qelim*/*qelim.ML*
~~/*src*/*HOL*/*Tools*/*Qelim*/*langford-data.ML*
~~/*src*/*HOL*/*Tools*/*Qelim*/*ferrante-rackoff-data.ML*
(~~/*src*/*HOL*/*Tools*/*Qelim*/*langford.ML*)
(~~/*src*/*HOL*/*Tools*/*Qelim*/*ferrante-rackoff.ML*)
begin

setup *Langford-Data.setup*
setup *Ferrante-Rackoff-Data.setup*

context *linorder*
begin

lemma *less-not-permute*: $\neg\ (x < y \wedge y < x)$ **by** (*simp add*: *not-less linear*)

lemma *gather-simps*:
shows
$(\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in U.\ x < y) \wedge x < u \wedge P\ x) \longleftrightarrow (\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in (\mathit{insert}\ u\ U).\ x < y) \wedge P\ x)$
and $(\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in U.\ x < y) \wedge l < x \wedge P\ x) \longleftrightarrow (\exists x.\ (\forall y \in (\mathit{insert}\ l\ L).\ y < x) \wedge (\forall y \in U.\ x < y) \wedge P\ x)$
$(\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in U.\ x < y) \wedge x < u) \longleftrightarrow (\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in (\mathit{insert}\ u\ U).\ x < y))$
and $(\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in U.\ x < y) \wedge l < x) \longleftrightarrow (\exists x.\ (\forall y \in (\mathit{insert}\ l\ L).\ y < x) \wedge (\forall y \in U.\ x < y))$ **by** *auto*

lemma
gather-start: $(\exists x.\ P\ x) \equiv (\exists x.\ (\forall y \in \{\}.\ y < x) \wedge (\forall y \in \{\}.\ x < y) \wedge P\ x)$
by *simp*

Theorems for $\exists z.\ \forall x.\ x < z \longrightarrow (P\ x \longleftrightarrow P_{-\infty})$

lemma *minf-lt*: $\exists z\ .\ \forall x.\ x < z \longrightarrow (x < t \longleftrightarrow \mathit{True})$ **by** *auto*
lemma *minf-gt*: $\exists z\ .\ \forall x.\ x < z \longrightarrow (t < x \longleftrightarrow \mathit{False})$
by (*simp add*: *not-less*) (*rule exI*[**where** x=t], *auto simp add*: *less-le*)

lemma *minf-le*: $\exists z.\ \forall x.\ x < z \longrightarrow (x \leq t \longleftrightarrow \mathit{True})$ **by** (*auto simp add*: *less-le*)
lemma *minf-ge*: $\exists z.\ \forall x.\ x < z \longrightarrow (t \leq x \longleftrightarrow \mathit{False})$
by (*auto simp add*: *less-le not-less not-le*)
lemma *minf-eq*: $\exists z.\ \forall x.\ x < z \longrightarrow (x = t \longleftrightarrow \mathit{False})$ **by** *auto*
lemma *minf-neq*: $\exists z.\ \forall x.\ x < z \longrightarrow (x \neq t \longleftrightarrow \mathit{True})$ **by** *auto*
lemma *minf-P*: $\exists z.\ \forall x.\ x < z \longrightarrow (P \longleftrightarrow P)$ **by** *blast*

Theorems for $\exists\, z.\ \forall\, x.\ x < z \longrightarrow (P\ x \longleftrightarrow P_{+\infty})$

lemma *pinf-gt*: $\exists\, z\ .\ \forall\, x.\ z < x \longrightarrow (t < x \longleftrightarrow \mathit{True})$ **by** *auto*
lemma *pinf-lt*: $\exists\, z\ .\ \forall\, x.\ z < x \longrightarrow (x < t \longleftrightarrow \mathit{False})$
by (*simp add*: *not-less*) (*rule exI*[**where** x=t], *auto simp add*: *less-le*)

lemma *pinf-ge*: $\exists\, z.\ \forall\, x.\ z < x \longrightarrow (t \le x \longleftrightarrow \mathit{True})$ **by** (*auto simp add*: *less-le*)
lemma *pinf-le*: $\exists\, z.\ \forall\, x.\ z < x \longrightarrow (x \le t \longleftrightarrow \mathit{False})$
by (*auto simp add*: *less-le not-less not-le*)
lemma *pinf-eq*: $\exists\, z.\ \forall\, x.\ z < x \longrightarrow (x = t \longleftrightarrow \mathit{False})$ **by** *auto*
lemma *pinf-neq*: $\exists\, z.\ \forall\, x.\ z < x \longrightarrow (x \neq t \longleftrightarrow \mathit{True})$ **by** *auto*
lemma *pinf-P*: $\exists\, z.\ \forall\, x.\ z < x \longrightarrow (P \longleftrightarrow P)$ **by** *blast*

lemma *nmi-lt*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{True} \wedge x < t \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$ **by** *auto*
lemma *nmi-gt*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{False} \wedge t < x \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$
by (*auto simp add*: *le-less*)
lemma *nmi-le*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{True} \wedge x{\le}\ t \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$ **by** *auto*
lemma *nmi-ge*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{False} \wedge t{\le}\ x \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$ **by** *auto*
lemma *nmi-eq*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{False} \wedge\ x = t \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$ **by** *auto*
lemma *nmi-neq*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{True} \wedge x \neq t \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$ **by** *auto*
lemma *nmi-P*: $\forall\ x.\ {}^{\sim}P \wedge P \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$ **by** *auto*
lemma *nmi-conj*: $[\![\forall\, x.\ \neg P1' \wedge P1\ x \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)\ ;$
$\forall\, x.\ \neg P2' \wedge P2\ x \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)]\!] \Longrightarrow$
$\forall\, x.\ \neg(P1' \wedge P2') \wedge (P1\ x \wedge P2\ x) \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$ **by** *auto*
lemma *nmi-disj*: $[\![\forall\, x.\ \neg P1' \wedge P1\ x \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)\ ;$
$\forall\, x.\ \neg P2' \wedge P2\ x \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)]\!] \Longrightarrow$
$\forall\, x.\ \neg(P1' \vee P2') \wedge (P1\ x \vee P2\ x) \longrightarrow\ (\exists\ u{\in}\ U.\ u \le x)$ **by** *auto*

lemma *npi-lt*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{False} \wedge\ x < t \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)$ **by** (*auto simp add*: *le-less*)
lemma *npi-gt*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{True} \wedge t < x \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)$ **by** *auto*
lemma *npi-le*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{False} \wedge\ x \le t \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)$ **by** *auto*
lemma *npi-ge*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{True} \wedge t \le x \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)$ **by** *auto*
lemma *npi-eq*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{False} \wedge\ x = t \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)$ **by** *auto*
lemma *npi-neq*: $t \in U \Longrightarrow \forall\, x.\ \neg \mathit{True} \wedge x \neq t \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u\ )$ **by** *auto*
lemma *npi-P*: $\forall\ x.\ {}^{\sim}P \wedge P \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)$ **by** *auto*
lemma *npi-conj*: $[\![\forall\, x.\ \neg P1' \wedge P1\ x \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)\ ;\ \forall\, x.\ \neg P2' \wedge P2\ x$
$\longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)]\!]$
$\Longrightarrow\ \forall\, x.\ \neg(P1' \wedge P2') \wedge (P1\ x \wedge P2\ x) \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)$ **by** *auto*
lemma *npi-disj*: $[\![\forall\, x.\ \neg P1' \wedge P1\ x \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)\ ; \forall\, x.\ \neg P2' \wedge P2\ x$
$\longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)]\!]$
$\Longrightarrow \forall\, x.\ \neg(P1' \vee P2') \wedge (P1\ x \vee P2\ x) \longrightarrow\ (\exists\ u{\in}\ U.\ x \le u)$ **by** *auto*

lemma *lin-dense-lt*: $t \in U \Longrightarrow \forall\, x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l{<}\ x \wedge$
$x < u \wedge x < t \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow y < t)$
proof(*clarsimp*)
fix $x\ l\ u\ y$ **assume** tU: $t \in U$ **and** noU: $\forall\, t.\ l < t \wedge t < u \longrightarrow t \notin U$ **and** lx: $l < x$
and xu: $x{<}u$ **and** px: $x < t$ **and** ly: $l{<}y$ **and** yu:$y < u$
from tU noU ly yu **have** tny: $t{\neq}y$ **by** *auto*

{assume H: $t < y$
from *less-trans*[*OF lx px*] *less-trans*[*OF H yu*]
have $l < t \wedge t < u$ **by** *simp*
with *tU noU* **have** *False* **by** *auto***}**
hence $\neg\ t < y$ **by** *auto* **hence** $y \leq t$ **by** (*simp add*: *not-less*)
thus $y < t$ **using** *tny* **by** (*simp add*: *less-le*)
qed

lemma *lin-dense-gt*: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t < u \longrightarrow t \notin U) \wedge l < x \wedge x < u \wedge t < x \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow t < y)$
proof(*clarsimp*)
fix $x\ l\ u\ y$
assume *tU*: $t \in U$ **and** *noU*: $\forall t.\ l < t \wedge t < u \longrightarrow t \notin U$ **and** *lx*: $l < x$ **and** *xu*: $x<u$
and *px*: $t < x$ **and** *ly*: $l<y$ **and** *yu*:$y < u$
from *tU noU ly yu* **have** *tny*: $t\neq y$ **by** *auto*
{assume H: $y< t$
from *less-trans*[*OF ly H*] *less-trans*[*OF px xu*] **have** $l < t \wedge t < u$ **by** *simp*
with *tU noU* **have** *False* **by** *auto***}**
hence $\neg\ y<t$ **by** *auto* **hence** $t \leq y$ **by** (*auto simp add*: *not-less*)
thus $t < y$ **using** *tny* **by** (*simp add*:*less-le*)
qed

lemma *lin-dense-le*: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t< u \longrightarrow t \notin U) \wedge l< x \wedge x < u \wedge x \leq t \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow y\leq t)$
proof(*clarsimp*)
fix $x\ l\ u\ y$
assume *tU*: $t \in U$ **and** *noU*: $\forall t.\ l < t \wedge t < u \longrightarrow t \notin U$ **and** *lx*: $l < x$ **and** *xu*: $x<u$
and *px*: $x \leq t$ **and** *ly*: $l<y$ **and** *yu*:$y < u$
from *tU noU ly yu* **have** *tny*: $t\neq y$ **by** *auto*
{assume H: $t < y$
from *less-le-trans*[*OF lx px*] *less-trans*[*OF H yu*]
have $l < t \wedge t < u$ **by** *simp*
with *tU noU* **have** *False* **by** *auto***}**
hence $\neg\ t < y$ **by** *auto* **thus** $y \leq t$ **by** (*simp add*: *not-less*)
qed

lemma *lin-dense-ge*: $t \in U \Longrightarrow \forall x\ l\ u.\ (\forall\ t.\ l < t \wedge t< u \longrightarrow t \notin U) \wedge l< x \wedge x < u \wedge t \leq x \longrightarrow (\forall\ y.\ l < y \wedge y < u \longrightarrow t \leq y)$
proof(*clarsimp*)
fix $x\ l\ u\ y$
assume *tU*: $t \in U$ **and** *noU*: $\forall t.\ l < t \wedge t < u \longrightarrow t \notin U$ **and** *lx*: $l < x$ **and** *xu*: $x<u$
and *px*: $t \leq x$ **and** *ly*: $l<y$ **and** *yu*:$y < u$
from *tU noU ly yu* **have** *tny*: $t\neq y$ **by** *auto*
{assume H: $y< t$
from *less-trans*[*OF ly H*] *le-less-trans*[*OF px xu*]
have $l < t \wedge t < u$ **by** *simp*

with *tU noU* **have** *False* **by** *auto*}
hence ¬ *y*<*t* **by** *auto* **thus** *t* ≤ *y* **by** (*simp add*: *not-less*)
qed
lemma *lin-dense-eq*: *t* ∈ *U* ⟹ ∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ *x* = *t* ⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ *y*= *t*) **by** *auto*
lemma *lin-dense-neq*: *t* ∈ *U* ⟹ ∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ *x* ≠ *t* ⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ *y*≠ *t*) **by** *auto*
lemma *lin-dense-P*: ∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ *P* ⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ *P*) **by** *auto*

lemma *lin-dense-conj*:
⟦∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ *P1 x*
⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ *P1 y*) ;
∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ *P2 x*
⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ *P2 y*)⟧ ⟹
∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ (*P1 x* ∧ *P2 x*)
⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ (*P1 y* ∧ *P2 y*))
by *blast*
lemma *lin-dense-disj*:
⟦∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ *P1 x*
⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ *P1 y*) ;
∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ *P2 x*
⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ *P2 y*)⟧ ⟹
∀ *x l u*. (∀ *t*. *l* < *t* ∧ *t*< *u* ⟶ *t* ∉ *U*) ∧ *l*< *x* ∧ *x* < *u* ∧ (*P1 x* ∨ *P2 x*)
⟶ (∀ *y*. *l* < *y* ∧ *y* < *u* ⟶ (*P1 y* ∨ *P2 y*))
by *blast*

lemma *npmibnd*: ⟦∀ *x*. ¬ *MP* ∧ *P x* ⟶ (∃ *u*∈ *U*. *u* ≤ *x*); ∀ *x*. ¬*PP* ∧ *P x* ⟶ (∃ *u*∈ *U*. *x* ≤ *u*)⟧
⟹ ∀ *x*. ¬ *MP* ∧ ¬*PP* ∧ *P x* ⟶ (∃ *u*∈ *U*. ∃ *u'* ∈ *U*. *u* ≤ *x* ∧ *x* ≤ *u'*)
by *auto*

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

from *xu uinS* **have** *linxM*: $u \in ?xM$ **by** *blast*
hence *xMne*: $?xM \neq \{\}$ **by** *blast*
have *ax*:$?a \leq x$ **using** *Mxne fMx* **by** *auto*
have *xb*:$x \leq ?b$ **using** *xMne fxM* **by** *auto*
have $?a \in ?Mx$ **using** *Max-in*[*OF fMx Mxne*] **by** *simp* **hence** *ainS*: $?a \in S$ **using** *MxS* **by** *blast*
have $?b \in ?xM$ **using** *Min-in*[*OF fxM xMne*] **by** *simp* **hence** *binS*: $?b \in S$ **using** *xMS* **by** *blast*
have *noy*:$\forall\ y.\ ?a < y \wedge y < ?b \longrightarrow y \notin S$
proof(*clarsimp*)
fix y **assume** *ay*: $?a < y$ **and** *yb*: $y < ?b$ **and** *yS*: $y \in S$
from *yS* **have** $y \in ?Mx \vee y \in ?xM$ **by** (*auto simp add*: *linear*)
moreover {assume $y \in ?Mx$ **hence** $y \leq ?a$ **using** *Mxne fMx* **by** *auto* **with** *ay* **have** *False* **by** (*simp add*: *not-le*[*symmetric*])**}**
moreover {assume $y \in ?xM$ **hence** $?b \leq y$ **using** *xMne fxM* **by** *auto* **with** *yb* **have** *False* **by** (*simp add*: *not-le*[*symmetric*])**}**
ultimately show *False* **by** *blast*
qed
from *ainS binS noy ax xb px* **show** *?thesis* **by** *blast*
qed

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

end

14 The classical QE after Langford for dense linear orders

context *dense-linear-order*
begin

lemma *dlo-qe-bnds*:
assumes *ne*: $L \neq \{\}$ **and** *neU*: $U \neq \{\}$ **and** *fL*: *finite L* **and** *fU*: *finite U*
shows $(\exists\, x.\ (\forall\, y \in L.\ y < x) \wedge (\forall\, y \in U.\ x < y)) \equiv (\forall\ l \in L.\ \forall\, u \in U.\ l < u)$
proof (*simp only*: *atomize-eq*, *rule iffI*)

assume H: $\exists x.\ (\forall y{\in}L.\ y < x) \wedge (\forall y{\in}U.\ x < y)$
then obtain x **where** xL: $\forall y{\in}L.\ y < x$ **and** xU: $\forall y{\in}U.\ x < y$ **by** *blast*
{fix $l\ u$ **assume** l: $l \in L$ **and** u: $u \in U$
have $l < x$ **using** $xL\ l$ **by** *blast*
also have $x < u$ **using** $xU\ u$ **by** *blast*
finally (*less-trans*) **have** $l < u$ **.}**
thus $\forall l{\in}L.\ \forall u{\in}U.\ l < u$ **by** *blast*
next
assume H: $\forall l{\in}L.\ \forall u{\in}U.\ l < u$
let *?ML* $=$ *Max L*
let *?MU* $=$ *Min U*
from *fL ne* **have** *th1*: *?ML* $\in L$ **and** *th1'*: $\forall l{\in}L.\ l \leq$ *?ML* **by** *auto*
from *fU neU* **have** *th2*: *?MU* $\in U$ **and** *th2'*: $\forall u{\in}U.$ *?MU* $\leq u$ **by** *auto*
from *th1 th2 H* **have** *?ML* $<$ *?MU* **by** *auto*
with *dense* **obtain** w **where** *th3*: *?ML* $< w$ **and** *th4*: $w <$ *?MU* **by** *blast*
from *th3 th1'* **have** $\forall l \in L.\ l < w$ **by** *auto*
moreover from *th4 th2'* **have** $\forall u \in U.\ w < u$ **by** *auto*
ultimately show $\exists x.\ (\forall y{\in}L.\ y < x) \wedge (\forall y{\in}U.\ x < y)$ **by** *auto*
qed

lemma *dlo-qe-noub*:
assumes *ne*: $L \neq \{\}$ **and** *fL*: *finite L*
shows $(\exists x.\ (\forall y \in L.\ y < x) \wedge (\forall y \in \{\}.\ x < y)) \equiv$ *True*
proof(*simp add*: *atomize-eq*)
from *gt-ex*[*of Max L*] **obtain** M **where** M: *Max L* $< M$ **by** *blast*
from *ne fL* **have** $\forall x \in L.\ x \leq$ *Max L* **by** *simp*
with M **have** $\forall x{\in}L.\ x < M$ **by** (*auto intro*: *le-less-trans*)
thus $\exists x.\ \forall y{\in}L.\ y < x$ **by** *blast*
qed

lemma *dlo-qe-nolb*:
assumes *ne*: $U \neq \{\}$ **and** *fU*: *finite U*
shows $(\exists x.\ (\forall y \in \{\}.\ y < x) \wedge (\forall y \in U.\ x < y)) \equiv$ *True*
proof(*simp add*: *atomize-eq*)
from *lt-ex*[*of Min U*] **obtain** M **where** M: $M <$ *Min U* **by** *blast*
from *ne fU* **have** $\forall x \in U.$ *Min U* $\leq x$ **by** *simp*
with M **have** $\forall x{\in}U.\ M < x$ **by** (*auto intro*: *less-le-trans*)
thus $\exists x.\ \forall y{\in}U.\ x < y$ **by** *blast*
qed

lemma *exists-neq*: $\exists (x::'a).\ x \neq t\ \exists (x::'a).\ t \neq x$
using *gt-ex*[*of t*] **by** *auto*

lemmas *dlo-simps* $=$ *order-refl less-irrefl not-less not-le exists-neq*
le-less neq-iff linear less-not-permute

lemma *axiom*: *dense-linear-order* $(op \leq)$ $(op <)$ **by** (*rule dense-linear-order-axioms*)
lemma *atoms*:
includes *meta-term-syntax*

shows *TERM* (*less* :: *'a* ⇒ -)
and *TERM* (*less-eq* :: *'a* ⇒ -)
and *TERM* (*op* = :: *'a* ⇒ -) **.**

declare *axiom*[*langford qe*: *dlo-qe-bnds dlo-qe-nolb dlo-qe-noub gather*: *gather-start gather-simps atoms*: *atoms*]
declare *dlo-simps*[*langfordsimp*]

end

lemma *dnf*:
(*P* & (*Q* | *R*)) = ((*P*&*Q*) | (*P*&*R*))
((*Q* | *R*) & *P*) = ((*Q*&*P*) | (*R*&*P*))
by *blast+*

lemmas *weak-dnf-simps* = *simp-thms dnf*

lemma *nnf-simps*:
(¬(*P* ∧ *Q*)) = (¬*P* ∨ ¬*Q*) (¬(*P* ∨ *Q*)) = (¬*P* ∧ ¬*Q*) (*P* ⟶ *Q*) = (¬*P* ∨ *Q*)
(*P* = *Q*) = ((*P* ∧ *Q*) ∨ (¬*P* ∧ ¬ *Q*)) (¬ ¬(*P*)) = *P*
by *blast+*

lemma *ex-distrib*: (∃ *x*. *P x* ∨ *Q x*) ⟷ ((∃ *x*. *P x*) ∨ (∃ *x*. *Q x*)) **by** *blast*

lemmas *dnf-simps* = *weak-dnf-simps nnf-simps ex-distrib*

use ~~/*src*/*HOL*/*Tools*/*Qelim*/*langford.ML*
method-setup *dlo* = ⟪
Method.ctxt-args (*Method.SIMPLE-METHOD'* o *LangfordQE.dlo-tac*)
⟫ *Langford's algorithm for quantifier elimination in dense linear orders*

15 Contructive dense linear orders yield QE for linear arithmetic over ordered Fields – see *Arith-Tools.thy*

Linear order without upper bounds

locale *linorder-stupid-syntax* = *linorder*
begin
notation
less-eq (*op* ⊑) **and**
less-eq ((-/ ⊑ -) [*51*, *51*] *50*) **and**
less (*op* ⊏) **and**
less ((-/ ⊏ -) [*51*, *51*] *50*)

end

locale *linorder-no-ub* = *linorder-stupid-syntax* +
assumes *gt-ex*: ∃ *y*. *less x y*

begin
lemma *ge-ex*: $\exists\, y.\ x \sqsubseteq y$ **using** *gt-ex* **by** *auto*

Theorems for $\exists\, z.\ \forall\, x.\ z \sqsubset x \longrightarrow (P\ x \longleftrightarrow P_{+\infty})$

lemma *pinf-conj*:
assumes *ex1*: $\exists\, z1.\ \forall\, x.\ z1 \sqsubset x \longrightarrow (P1\ x \longleftrightarrow P1')$
and *ex2*: $\exists\, z2.\ \forall\, x.\ z2 \sqsubset x \longrightarrow (P2\ x \longleftrightarrow P2')$
shows $\exists\, z.\ \forall\, x.\ z \sqsubset\ x \longrightarrow ((P1\ x \wedge P2\ x) \longleftrightarrow (P1' \wedge P2'))$
proof−
from *ex1 ex2* **obtain** *z1* **and** *z2* **where** *z1*: $\forall\, x.\ z1 \sqsubset x \longrightarrow (P1\ x \longleftrightarrow P1')$
and *z2*: $\forall\, x.\ z2 \sqsubset x \longrightarrow (P2\ x \longleftrightarrow P2')$ **by** *blast*
from *gt-ex* **obtain** *z* **where** *z*:*ord.max less-eq z1 z2* $\sqsubset z$ **by** *blast*
from *z* **have** *zz1*: $z1 \sqsubset z$ **and** *zz2*: $z2 \sqsubset z$ **by** *simp-all*
{fix *x* **assume** *H*: $z \sqsubset x$
from *less-trans*[*OF zz1 H*] *less-trans*[*OF zz2 H*]
have $(P1\ x \wedge P2\ x) \longleftrightarrow (P1' \wedge P2')$ **using** *z1 zz1 z2 zz2* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *pinf-disj*:
assumes *ex1*: $\exists\, z1.\ \forall\, x.\ z1 \sqsubset x \longrightarrow (P1\ x \longleftrightarrow P1')$
and *ex2*: $\exists\, z2.\ \forall\, x.\ z2 \sqsubset x \longrightarrow (P2\ x \longleftrightarrow P2')$
shows $\exists\, z.\ \forall\, x.\ z \sqsubset\ x \longrightarrow ((P1\ x \vee P2\ x) \longleftrightarrow (P1' \vee P2'))$
proof−
from *ex1 ex2* **obtain** *z1* **and** *z2* **where** *z1*: $\forall\, x.\ z1 \sqsubset x \longrightarrow (P1\ x \longleftrightarrow P1')$
and *z2*: $\forall\, x.\ z2 \sqsubset x \longrightarrow (P2\ x \longleftrightarrow P2')$ **by** *blast*
from *gt-ex* **obtain** *z* **where** *z*:*ord.max less-eq z1 z2* $\sqsubset z$ **by** *blast*
from *z* **have** *zz1*: $z1 \sqsubset z$ **and** *zz2*: $z2 \sqsubset z$ **by** *simp-all*
{fix *x* **assume** *H*: $z \sqsubset x$
from *less-trans*[*OF zz1 H*] *less-trans*[*OF zz2 H*]
have $(P1\ x \vee P2\ x) \longleftrightarrow (P1' \vee P2')$ **using** *z1 zz1 z2 zz2* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *pinf-ex*: **assumes** *ex*:$\exists\, z.\ \forall\, x.\ z \sqsubset x \longrightarrow (P\ x \longleftrightarrow P1)$ **and** *p1*: *P1* **shows** $\exists\ x.\ P\ x$
proof−
from *ex* **obtain** *z* **where** *z*: $\forall\, x.\ z \sqsubset x \longrightarrow (P\ x \longleftrightarrow P1)$ **by** *blast*
from *gt-ex* **obtain** *x* **where** *x*: $z \sqsubset x$ **by** *blast*
from *z x p1* **show** *?thesis* **by** *blast*
qed

end

Linear order without upper bounds

locale *linorder-no-lb* = *linorder-stupid-syntax* +
assumes *lt-ex*: $\exists\, y.$ *less y x*

begin
lemma *le-ex*: $\exists\, y.\ y \sqsubseteq x$ **using** *lt-ex* **by** *auto*

Theorems for $\exists\, z.\ \forall\, x.\ x \sqsubset z \longrightarrow (P\ x \longleftrightarrow P_{-\infty})$

lemma *minf-conj*:
assumes *ex1*: $\exists\, z1.\ \forall\, x.\ x \sqsubset z1 \longrightarrow (P1\ x \longleftrightarrow P1')$
and *ex2*: $\exists\, z2.\ \forall\, x.\ x \sqsubset z2 \longrightarrow (P2\ x \longleftrightarrow P2')$
shows $\exists\, z.\ \forall\, x.\ x \sqsubset z \longrightarrow ((P1\ x \wedge P2\ x) \longleftrightarrow (P1' \wedge P2'))$
proof−
from *ex1 ex2* **obtain** *z1* **and** *z2* **where** *z1*: $\forall\, x.\ x \sqsubset z1 \longrightarrow (P1\ x \longleftrightarrow P1')$**and** *z2*: $\forall\, x.\ x \sqsubset z2 \longrightarrow (P2\ x \longleftrightarrow P2')$ **by** *blast*
from *lt-ex* **obtain** *z* **where** *z*:$z \sqsubset$ *ord.min less-eq z1 z2* **by** *blast*
from *z* **have** *zz1*: $z \sqsubset z1$ **and** *zz2*: $z \sqsubset z2$ **by** *simp-all*
{fix *x* **assume** *H*: $x \sqsubset z$
from *less-trans*[*OF H zz1*] *less-trans*[*OF H zz2*]
have $(P1\ x \wedge P2\ x) \longleftrightarrow (P1' \wedge P2')$ **using** *z1 zz1 z2 zz2* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *minf-disj*:
assumes *ex1*: $\exists\, z1.\ \forall\, x.\ x \sqsubset z1 \longrightarrow (P1\ x \longleftrightarrow P1')$
and *ex2*: $\exists\, z2.\ \forall\, x.\ x \sqsubset z2 \longrightarrow (P2\ x \longleftrightarrow P2')$
shows $\exists\, z.\ \forall\, x.\ x \sqsubset z \longrightarrow ((P1\ x \vee P2\ x) \longleftrightarrow (P1' \vee P2'))$
proof−
from *ex1 ex2* **obtain** *z1* **and** *z2* **where** *z1*: $\forall\, x.\ x \sqsubset z1 \longrightarrow (P1\ x \longleftrightarrow P1')$**and** *z2*: $\forall\, x.\ x \sqsubset z2 \longrightarrow (P2\ x \longleftrightarrow P2')$ **by** *blast*
from *lt-ex* **obtain** *z* **where** *z*:$z \sqsubset$ *ord.min less-eq z1 z2* **by** *blast*
from *z* **have** *zz1*: $z \sqsubset z1$ **and** *zz2*: $z \sqsubset z2$ **by** *simp-all*
{fix *x* **assume** *H*: $x \sqsubset z$
from *less-trans*[*OF H zz1*] *less-trans*[*OF H zz2*]
have $(P1\ x \vee P2\ x) \longleftrightarrow (P1' \vee P2')$ **using** *z1 zz1 z2 zz2* **by** *auto*
}
thus *?thesis* **by** *blast*
qed

lemma *minf-ex*: **assumes** *ex*:$\exists\, z.\ \forall\, x.\ x \sqsubset z \longrightarrow (P\ x \longleftrightarrow P1)$ **and** *p1*: *P1*
shows $\exists\ x.\ P\ x$
proof−
from *ex* **obtain** *z* **where** *z*: $\forall\, x.\ x \sqsubset z \longrightarrow (P\ x \longleftrightarrow P1)$ **by** *blast*
from *lt-ex* **obtain** *x* **where** *x*: $x \sqsubset z$ **by** *blast*
from *z x p1* **show** *?thesis* **by** *blast*
qed

end

locale *constr-dense-linear-order* = *linorder-no-lb* + *linorder-no-ub* +
fixes *between*

assumes *between-less*: *less x y* $\Longrightarrow$ *less x* (*between x y*) $\wedge$ *less* (*between x y*) *y*
and *between-same*: *between x x* = *x*

interpretation *constr-dense-linear-order* < *dense-linear-order*
apply *unfold-locales*
using *gt-ex lt-ex between-less*
by (*auto*, *rule-tac x*=*between x y* **in** *exI*, *simp*)

context *constr-dense-linear-order*
begin

lemma *rinf-U*:
assumes *fU*: *finite U*
and *lin-dense*: $\forall x\ l\ u.\ (\forall\ t.\ l \sqsubset t \wedge t \sqsubset u \longrightarrow t \notin U) \wedge l \sqsubset x \wedge x \sqsubset u \wedge P\ x \longrightarrow (\forall\ y.\ l \sqsubset y \wedge y \sqsubset u \longrightarrow P\ y\ )$
and *nmpiU*: $\forall x.\ \neg\ MP \wedge \neg PP \wedge P\ x \longrightarrow (\exists\ u{\in}\ U.\ \exists\ u' \in U.\ u \sqsubseteq x \wedge x \sqsubseteq u')$
and *nmi*: $\neg\ MP$ **and** *npi*: $\neg\ PP$ **and** *ex*: $\exists\ x.\ \ P\ x$
shows $\exists\ u{\in}\ U.\ \exists\ u' \in U.\ P$ (*between u u'*)
proof−
from *ex* **obtain** *x* **where** *px*: *P x* **by** *blast*
from *px nmi npi nmpiU* **have** $\exists\ u{\in}\ U.\ \exists\ u' \in U.\ u \sqsubseteq x \wedge x \sqsubseteq u'$ **by** *auto*
then obtain *u* **and** *u'* **where** *uU*:$u{\in}\ U$ **and** *uU'*: $u' \in U$ **and** *ux*:$u \sqsubseteq x$ **and** *xu'*:$x \sqsubseteq u'$ **by** *auto*
from *uU* **have** *Une*: $U \neq \{\}$ **by** *auto*
term *linorder.Min less-eq*
let *?l* = *linorder.Min less-eq U*
let *?u* = *linorder.Max less-eq U*
have *linM*: *?l* $\in$ *U* **using** *fU Une* **by** *simp*
have *uinM*: *?u* $\in$ *U* **using** *fU Une* **by** *simp*
have *lM*: $\forall\ t{\in}\ U.\ ?l \sqsubseteq t$ **using** *Une fU* **by** *auto*
have *Mu*: $\forall\ t{\in}\ U.\ t \sqsubseteq ?u$ **using** *Une fU* **by** *auto*
have *th*:*?l* $\sqsubseteq$ *u* **using** *uU Une lM* **by** *auto*
from *order-trans*[*OF th ux*] **have** *lx*: *?l* $\sqsubseteq$ *x* **.**
have *th*: $u' \sqsubseteq ?u$ **using** uU' *Une Mu* **by** *simp*
from *order-trans*[*OF xu' th*] **have** *xu*: *x* $\sqsubseteq$ *?u* **.**
from *finite-set-intervals2*[**where** *P*=*P*,*OF px lx xu linM uinM fU lM Mu*]
have $(\exists\ s{\in}\ U.\ P\ s) \vee$
$(\exists\ t1{\in}\ U.\ \exists\ t2 \in U.\ (\forall\ y.\ t1 \sqsubset y \wedge y \sqsubset t2 \longrightarrow y \notin U) \wedge t1 \sqsubset x \wedge x \sqsubset t2 \wedge P\ x)$ **.**
moreover { fix *u* **assume** *um*: *u*∈*U* **and** *pu*: *P u*
have *between u u* = *u* **by** (*simp add*: *between-same*)
with *um pu* **have** *P* (*between u u*) **by** *simp*
with *um* **have** *?thesis* **by** *blast*}
moreover{
assume $\exists\ t1{\in}\ U.\ \exists\ t2 \in U.\ (\forall\ y.\ t1 \sqsubset y \wedge y \sqsubset t2 \longrightarrow y \notin U) \wedge t1 \sqsubset x \wedge x \sqsubset t2 \wedge P\ x$
then obtain *t1* **and** *t2* **where** *t1M*: *t1* $\in$ *U* **and** *t2M*: *t2*$\in$ *U*
and *noM*: $\forall\ y.\ t1 \sqsubset y \wedge y \sqsubset t2 \longrightarrow y \notin U$ **and** *t1x*: *t1* $\sqsubset$ *x* **and** *xt2*: *x*

$\sqsubset$ *t2* **and** *px*: *P x*
by *blast*
from *less-trans*[*OF t1x xt2*] **have** *t1t2*: *t1* $\sqsubset$ *t2* **.**
let *?u* = *between t1 t2*
from *between-less t1t2* **have** *t1lu*: *t1* $\sqsubset$ *?u* **and** *ut2*: *?u* $\sqsubset$ *t2* **by** *auto*
from *lin-dense noM t1x xt2 px t1lu ut2* **have** *P ?u* **by** *blast*
with *t1M t2M* **have** *?thesis* **by** *blast*}
ultimately show *?thesis* **by** *blast*
qed

theorem *fr-eq*:
assumes *fU*: *finite U*
and *lin-dense*: $\forall x\ l\ u.\ (\forall\ t.\ l \sqsubset t \wedge t \sqsubset u \longrightarrow t \notin U) \wedge l \sqsubset x \wedge x \sqsubset u \wedge P\ x \longrightarrow (\forall\ y.\ l \sqsubset y \wedge y \sqsubset u \longrightarrow P\ y\ )$
and *nmibnd*: $\forall x.\ \neg\ MP \wedge P\ x \longrightarrow (\exists\ u{\in}\ U.\ u \sqsubseteq x)$
and *npibnd*: $\forall x.\ \neg PP \wedge P\ x \longrightarrow (\exists\ u{\in}\ U.\ x \sqsubseteq u)$
and *mi*: $\exists z.\ \forall x.\ x \sqsubset z \longrightarrow (P\ x = MP)$ **and** *pi*: $\exists z.\ \forall x.\ z \sqsubset x \longrightarrow (P\ x = PP)$
shows $(\exists\ x.\ P\ x) \equiv (MP \vee PP \vee (\exists\ u \in U.\ \exists\ u'{\in}\ U.\ P\ (between\ u\ u')))$
(**is** - $\equiv$ (- $\vee$ - $\vee$ *?F*) **is** *?E* $\equiv$ *?D*)
proof−
{
assume *px*: $\exists\ x.\ P\ x$
have $MP \vee PP \vee (\neg\ MP \wedge \neg\ PP)$ **by** *blast*
moreover {**assume** $MP \vee PP$ **hence** *?D* **by** *blast*}
moreover {**assume** *nmi*: $\neg\ MP$ **and** *npi*: $\neg\ PP$
from *npmibnd*[*OF nmibnd npibnd*]
have *nmpiU*: $\forall x.\ \neg\ MP \wedge \neg PP \wedge P\ x \longrightarrow (\exists\ u{\in}\ U.\ \exists\ u' \in U.\ u \sqsubseteq x \wedge x \sqsubseteq u')$ **.**
from *rinf-U*[*OF fU lin-dense nmpiU nmi npi px*] **have** *?D* **by** *blast*}
ultimately have *?D* **by** *blast*}
moreover
{ **assume** *?D*
moreover {**assume** *m*:*MP* **from** *minf-ex*[*OF mi m*] **have** *?E* **.**}
moreover {**assume** *p*: *PP* **from** *pinf-ex*[*OF pi p*] **have** *?E* **. }**
moreover {**assume** *f*:*?F* **hence** *?E* **by** *blast*}
ultimately have *?E* **by** *blast*}
ultimately have *?E* = *?D* **by** *blast* **thus** *?E* $\equiv$ *?D* **by** *simp*
qed

lemmas *minf-thms* = *minf-conj minf-disj minf-eq minf-neq minf-lt minf-le minf-gt minf-ge minf-P*
lemmas *pinf-thms* = *pinf-conj pinf-disj pinf-eq pinf-neq pinf-lt pinf-le pinf-gt pinf-ge pinf-P*

lemmas *nmi-thms* = *nmi-conj nmi-disj nmi-eq nmi-neq nmi-lt nmi-le nmi-gt nmi-ge nmi-P*
lemmas *npi-thms* = *npi-conj npi-disj npi-eq npi-neq npi-lt npi-le npi-gt npi-ge npi-P*

lemmas *lin-dense-thms = lin-dense-conj lin-dense-disj lin-dense-eq lin-dense-neq lin-dense-lt lin-dense-le lin-dense-gt lin-dense-ge lin-dense-P*

lemma *ferrack-axiom*: *constr-dense-linear-order less-eq less between*
by (*rule constr-dense-linear-order-axioms*)
lemma *atoms*:
includes *meta-term-syntax*
shows *TERM* (*less* :: *'a* ⇒ -)
and *TERM* (*less-eq* :: *'a* ⇒ -)
and *TERM* (*op =* :: *'a* ⇒ -) **.**

declare *ferrack-axiom* [*ferrack minf*: *minf-thms pinf*: *pinf-thms*
nmi: *nmi-thms npi*: *npi-thms lindense*:
lin-dense-thms qe: *fr-eq atoms*: *atoms*]

declaration ⟪
let
fun simps phi = map (*Morphism.thm phi*) [@{*thm not-less*}, @{*thm not-le*}]
fun generic-whatis phi =
let
val [*lt*, *le*] = *map* (*Morphism.term phi*) [@{*term op* ⊏}, @{*term op* ⊑}]
fun h x t =
case term-of t of
Const(*op =*, -)$*y*$*z => if term-of x aconv y then Ferrante-Rackoff-Data.Eq*
else Ferrante-Rackoff-Data.Nox
| @{*term Not*}$(*Const*(*op =*, -)$*y*$*z*) => *if term-of x aconv y then Ferrante-Rackoff-Data.NEq*
else Ferrante-Rackoff-Data.Nox
| *b*$*y*$*z => if Term.could-unify* (*b*, *lt*) *then*
if term-of x aconv y then Ferrante-Rackoff-Data.Lt
else if term-of x aconv z then Ferrante-Rackoff-Data.Gt
else Ferrante-Rackoff-Data.Nox
else if Term.could-unify (*b*, *le*) *then*
if term-of x aconv y then Ferrante-Rackoff-Data.Le
else if term-of x aconv z then Ferrante-Rackoff-Data.Ge
else Ferrante-Rackoff-Data.Nox
else Ferrante-Rackoff-Data.Nox
| - => *Ferrante-Rackoff-Data.Nox*
in h end
fun ss phi = HOL-ss addsimps (*simps phi*)
in
Ferrante-Rackoff-Data.funs @{*thm ferrack-axiom*}
{*isolate-conv = K* (*K* (*K Thm.reflexive*)), *whatis = generic-whatis*, *simpset =*
ss}
end
⟫

end

use ~~/*src*/*HOL*/*Tools*/*Qelim*/*ferrante-rackoff.ML*

method-setup *ferrack* = ⟪
Method.ctxt-args (*Method.SIMPLE-METHOD' o FerranteRackoff.dlo-tac*)
⟫ *Ferrante and Rackoff's algorithm for quantifier elimination in dense linear orders*

15.1 Ferrante and Rackoff algorithm over ordered fields

lemma *neg-prod-lt*:(*c*::'*a*::*ordered-field*) < *0* ⟹ ((*c*∗*x* < *0*) == (*x* > *0*))
proof−
assume *H*: *c* < *0*
have *c*∗*x* < *0* = (*0*/*c* < *x*) **by** (*simp only*: *neg-divide-less-eq*[*OF H*] *ring-simps*)
also have ... = (*0* < *x*) **by** *simp*
finally show (*c*∗*x* < *0*) == (*x* > *0*) **by** *simp*
qed

lemma *pos-prod-lt*:(*c*::'*a*::*ordered-field*) > *0* ⟹ ((*c*∗*x* < *0*) == (*x* < *0*))
proof−
assume *H*: *c* > *0*
hence *c*∗*x* < *0* = (*0*/*c* > *x*) **by** (*simp only*: *pos-less-divide-eq*[*OF H*] *ring-simps*)
also have ... = (*0* > *x*) **by** *simp*
finally show (*c*∗*x* < *0*) == (*x* < *0*) **by** *simp*
qed

lemma *neg-prod-sum-lt*: (*c*::'*a*::*ordered-field*) < *0* ⟹ ((*c*∗*x* + *t*< *0*) == (*x* > (− *1*/*c*)∗*t*))
proof−
assume *H*: *c* < *0*
have *c*∗*x* + *t*< *0* = (*c*∗*x* < −*t*) **by** (*subst less-iff-diff-less-0* [*of c*∗*x* −*t*], *simp*)
also have ... = (−*t*/*c* < *x*) **by** (*simp only*: *neg-divide-less-eq*[*OF H*] *ring-simps*)
also have ... = ((− *1*/*c*)∗*t* < *x*) **by** *simp*
finally show (*c*∗*x* + *t* < *0*) == (*x* > (− *1*/*c*)∗*t*) **by** *simp*
qed

lemma *pos-prod-sum-lt*:(*c*::'*a*::*ordered-field*) > *0* ⟹ ((*c*∗*x* + *t* < *0*) == (*x* < (− *1*/*c*)∗*t*))
proof−
assume *H*: *c* > *0*
have *c*∗*x* + *t*< *0* = (*c*∗*x* < −*t*) **by** (*subst less-iff-diff-less-0* [*of c*∗*x* −*t*], *simp*)
also have ... = (−*t*/*c* > *x*) **by** (*simp only*: *pos-less-divide-eq*[*OF H*] *ring-simps*)
also have ... = ((− *1*/*c*)∗*t* > *x*) **by** *simp*
finally show (*c*∗*x* + *t* < *0*) == (*x* < (− *1*/*c*)∗*t*) **by** *simp*
qed

lemma *sum-lt*:((*x*::'*a*::*pordered-ab-group-add*) + *t* < *0*) == (*x* < − *t*)
using *less-diff-eq*[**where** *a*= *x* **and** *b*=*t* **and** *c*=*0*] **by** *simp*

lemma *neg-prod-le*:(*c*::'*a*::*ordered-field*) < *0* ⟹ ((*c*∗*x* <= *0*) == (*x* >= *0*))
proof−
assume *H*: *c* < *0*

have $c*x <= 0 = (0/c <= x)$ **by** (*simp only*: *neg-divide-le-eq*[*OF H*] *ring-simps*)
also have $\ldots = (0 <= x)$ **by** *simp*
finally show $(c*x <= 0) == (x >= 0)$ **by** *simp*
qed

lemma *pos-prod-le*:$(c::'a::ordered\text{-}field) > 0 \Longrightarrow ((c*x <= 0) == (x <= 0))$
proof−
assume *H*: $c > 0$
hence $c*x <= 0 = (0/c >= x)$ **by** (*simp only*: *pos-le-divide-eq*[*OF H*] *ring-simps*)
also have $\ldots = (0 >= x)$ **by** *simp*
finally show $(c*x <= 0) == (x <= 0)$ **by** *simp*
qed

lemma *neg-prod-sum-le*: $(c::'a::ordered\text{-}field) < 0 \Longrightarrow ((c*x + t <= 0) == (x >= (-1/c)*t))$
proof−
assume *H*: $c < 0$
have $c*x + t <= 0 = (c*x <= -t)$ **by** (*subst le-iff-diff-le-0* [*of* $c*x$ $-t$], *simp*)
also have $\ldots = (-t/c <= x)$ **by** (*simp only*: *neg-divide-le-eq*[*OF H*] *ring-simps*)
also have $\ldots = ((-1/c)*t <= x)$ **by** *simp*
finally show $(c*x + t <= 0) == (x >= (-1/c)*t)$ **by** *simp*
qed

lemma *pos-prod-sum-le*:$(c::'a::ordered\text{-}field) > 0 \Longrightarrow ((c*x + t <= 0) == (x <= (-1/c)*t))$
proof−
assume *H*: $c > 0$
have $c*x + t <= 0 = (c*x <= -t)$ **by** (*subst le-iff-diff-le-0* [*of* $c*x$ $-t$], *simp*)
also have $\ldots = (-t/c >= x)$ **by** (*simp only*: *pos-le-divide-eq*[*OF H*] *ring-simps*)
also have $\ldots = ((-1/c)*t >= x)$ **by** *simp*
finally show $(c*x + t <= 0) == (x <= (-1/c)*t)$ **by** *simp*
qed

lemma *sum-le*:$((x::'a::pordered\text{-}ab\text{-}group\text{-}add) + t <= 0) == (x <= -t)$
using *le-diff-eq*[**where** *a*= *x* **and** *b*=*t* **and** *c*=*0*] **by** *simp*

lemma *nz-prod-eq*:$(c::'a::ordered\text{-}field) \neq 0 \Longrightarrow ((c*x = 0) == (x = 0))$ **by** *simp*
lemma *nz-prod-sum-eq*: $(c::'a::ordered\text{-}field) \neq 0 \Longrightarrow ((c*x + t = 0) == (x = (-1/c)*t))$
proof−
assume *H*: $c \neq 0$
have $c*x + t = 0 = (c*x = -t)$ **by** (*subst eq-iff-diff-eq-0* [*of* $c*x$ $-t$], *simp*)
also have $\ldots = (x = -t/c)$ **by** (*simp only*: *nonzero-eq-divide-eq*[*OF H*] *ring-simps*)
finally show $(c*x + t = 0) == (x = (-1/c)*t)$ **by** *simp*
qed
lemma *sum-eq*:$((x::'a::pordered\text{-}ab\text{-}group\text{-}add) + t = 0) == (x = -t)$
using *eq-diff-eq*[**where** *a*= *x* **and** *b*=*t* **and** *c*=*0*] **by** *simp*

```
interpretation class-ordered-field-dense-linear-order: constr-dense-linear-order
 [op <= op <
   λ x y. 1/2 * ((x::'a::{ordered-field,recpower,number-ring}) + y)]
proof (unfold-locales, dlo, dlo, auto)
  fix x y::'a assume lt: x < y
  from  less-half-sum[OF lt] show x < (x + y) /2 by simp
next
  fix x y::'a assume lt: x < y
  from  gt-half-sum[OF lt] show (x + y) /2 < y by simp
qed

declaration⟪
let
fun earlier [] x y = false
        | earlier (h::t) x y =
    if h aconvc y then false else if h aconvc x then true else earlier t x y;

fun dest-frac ct = case term-of ct of
   Const (@{const-name HOL.divide},-) $ a $ b=>
    Rat.rat-of-quotient (snd (HOLogic.dest-number a), snd (HOLogic.dest-number
b))
 | t => Rat.rat-of-int (snd (HOLogic.dest-number t))

fun mk-frac phi cT x =
 let val (a, b) = Rat.quotient-of-rat x
 in if b = 1 then Numeral.mk-cnumber cT a
    else Thm.capply
         (Thm.capply (Drule.cterm-rule (instantiate' [SOME cT] []) @{cpat op /})
                  (Numeral.mk-cnumber cT a))
         (Numeral.mk-cnumber cT b)
 end

fun whatis x ct = case term-of ct of
  Const(@{const-name HOL.plus}, -)$(Const(@{const-name HOL.times},-)$-$y)$-
=>
     if y aconv term-of x then (c*x+t,[(funpow 2 Thm.dest-arg1) ct, Thm.dest-arg
ct])
     else (Nox,[])
| Const(@{const-name HOL.plus}, -)$y$- =>
     if y aconv term-of x then (x+t,[Thm.dest-arg ct])
     else (Nox,[])
| Const(@{const-name HOL.times}, -)$-$y =>
     if y aconv term-of x then (c*x,[Thm.dest-arg1 ct])
     else (Nox,[])
| t => if t aconv term-of x then (x,[]) else (Nox,[]);

fun xnormalize-conv ctxt [] ct = reflexive ct
| xnormalize-conv ctxt (vs as (x::-)) ct =
   case term-of ct of
```

```
  Const(@{const-name HOL.less},-)$-$Const(@{const-name HOL.zero},-) =>
   (case whatis x (Thm.dest-arg1 ct) of
   (c*x+t,[c,t]) =>
     let
      val cr = dest-frac c
      val clt = Thm.dest-fun2 ct
      val cz = Thm.dest-arg ct
      val neg = cr </ Rat.zero
      val cthp = Simplifier.rewrite (local-simpset-of ctxt)
            (Thm.capply @{cterm Trueprop}
              (if neg then Thm.capply (Thm.capply clt c) cz
                else Thm.capply (Thm.capply clt cz) c))
      val cth = equal-elim (symmetric cthp) TrueI
       val th = implies-elim (instantiate' [SOME (ctyp-of-term x)] (map SOME
[c,x,t])
           (if neg then @{thm neg-prod-sum-lt} else @{thm pos-prod-sum-lt})) cth
      val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
             (Normalizer.semiring-normalize-ord-conv ctxt (earlier vs)))) th
    in rth end
  | (x+t,[t]) =>
     let
      val T = ctyp-of-term x
      val th = instantiate' [SOME T] [SOME x, SOME t] @{thm sum-lt}
      val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
           (Normalizer.semiring-normalize-ord-conv ctxt (earlier vs)))) th
     in  rth end
  | (c*x,[c]) =>
     let
      val cr = dest-frac c
      val clt = Thm.dest-fun2 ct
      val cz = Thm.dest-arg ct
      val neg = cr </ Rat.zero
      val cthp = Simplifier.rewrite (local-simpset-of ctxt)
            (Thm.capply @{cterm Trueprop}
              (if neg then Thm.capply (Thm.capply clt c) cz
                else Thm.capply (Thm.capply clt cz) c))
      val cth = equal-elim (symmetric cthp) TrueI
       val th = implies-elim (instantiate' [SOME (ctyp-of-term x)] (map SOME
[c,x])
           (if neg then @{thm neg-prod-lt} else @{thm pos-prod-lt})) cth
      val rth = th
    in rth end
  | - => reflexive ct)


| Const(@{const-name HOL.less-eq},-)$-$Const(@{const-name HOL.zero},-) =>
  (case whatis x (Thm.dest-arg1 ct) of
   (c*x+t,[c,t]) =>
     let
```

```
      val T = ctyp-of-term x
      val cr = dest-frac c
      val clt = Drule.cterm-rule (instantiate' [SOME T] []) @{cpat op <}
      val cz = Thm.dest-arg ct
      val neg = cr </ Rat.zero
      val cthp = Simplifier.rewrite (local-simpset-of ctxt)
            (Thm.capply @{cterm Trueprop}
              (if neg then Thm.capply (Thm.capply clt c) cz
               else Thm.capply (Thm.capply clt cz) c))
      val cth = equal-elim (symmetric cthp) TrueI
      val th = implies-elim (instantiate' [SOME T] (map SOME [c,x,t])
          (if neg then @{thm neg-prod-sum-le} else @{thm pos-prod-sum-le})) cth
      val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
              (Normalizer.semiring-normalize-ord-conv ctxt (earlier vs)))) th
    in rth end
  | (x+t,[t]) =>
    let
      val T = ctyp-of-term x
      val th = instantiate' [SOME T] [SOME x, SOME t] @{thm sum-le}
      val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
           (Normalizer.semiring-normalize-ord-conv ctxt (earlier vs)))) th
     in  rth end
  | (c*x,[c]) =>
    let
      val T = ctyp-of-term x
      val cr = dest-frac c
      val clt = Drule.cterm-rule (instantiate' [SOME T] []) @{cpat op <}
      val cz = Thm.dest-arg ct
      val neg = cr </ Rat.zero
      val cthp = Simplifier.rewrite (local-simpset-of ctxt)
            (Thm.capply @{cterm Trueprop}
              (if neg then Thm.capply (Thm.capply clt c) cz
               else Thm.capply (Thm.capply clt cz) c))
      val cth = equal-elim (symmetric cthp) TrueI
       val th = implies-elim (instantiate' [SOME (ctyp-of-term x)] (map SOME
[c,x])
          (if neg then @{thm neg-prod-le} else @{thm pos-prod-le})) cth
      val rth = th
    in rth end
  | - => reflexive ct)

| Const(op =,-)$-$Const(@{const-name HOL.zero},-) =>
  (case whatis x (Thm.dest-arg1 ct) of
   (c*x+t,[c,t]) =>
    let
      val T = ctyp-of-term x
      val cr = dest-frac c
      val ceq = Thm.dest-fun2 ct
      val cz = Thm.dest-arg ct
```

```
        val cthp = Simplifier.rewrite (local-simpset-of ctxt)
           (Thm.capply @{cterm Trueprop}
            (Thm.capply @{cterm Not} (Thm.capply (Thm.capply ceq c) cz)))
        val cth = equal-elim (symmetric cthp) TrueI
        val th = implies-elim
              (instantiate' [SOME T] (map SOME [c,x,t]) @{thm nz-prod-sum-eq})
cth
        val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
                  (Normalizer.semiring-normalize-ord-conv ctxt (earlier vs)))) th
      in rth end
    | (x+t,[t]) =>
      let
        val T = ctyp-of-term x
        val th = instantiate' [SOME T] [SOME x, SOME t] @{thm sum-eq}
        val rth = Conv.fconv-rule (Conv.arg-conv (Conv.binop-conv
             (Normalizer.semiring-normalize-ord-conv ctxt (earlier vs)))) th
      in  rth end
    | (c*x,[c]) =>
      let
        val T = ctyp-of-term x
        val cr = dest-frac c
        val ceq = Thm.dest-fun2 ct
        val cz = Thm.dest-arg ct
        val cthp = Simplifier.rewrite (local-simpset-of ctxt)
           (Thm.capply @{cterm Trueprop}
            (Thm.capply @{cterm Not} (Thm.capply (Thm.capply ceq c) cz)))
        val cth = equal-elim (symmetric cthp) TrueI
        val rth = implies-elim
              (instantiate' [SOME T] (map SOME [c,x]) @{thm nz-prod-eq}) cth
      in rth end
    | - => reflexive ct);

local
  val less-iff-diff-less-0 = mk-meta-eq @{thm less-iff-diff-less-0}
  val le-iff-diff-le-0 = mk-meta-eq @{thm le-iff-diff-le-0}
  val eq-iff-diff-eq-0 = mk-meta-eq @{thm eq-iff-diff-eq-0}
in
fun field-isolate-conv phi ctxt vs ct = case term-of ct of
  Const(@{const-name HOL.less},-)$a$b =>
   let val (ca,cb) = Thm.dest-binop ct
       val T = ctyp-of-term ca
       val th = instantiate' [SOME T] [SOME ca, SOME cb] less-iff-diff-less-0
       val nth = Conv.fconv-rule
         (Conv.arg-conv (Conv.arg1-conv
              (Normalizer.semiring-normalize-ord-conv @{context} (earlier vs)))) th
       val rth = transitive nth (xnormalize-conv ctxt vs (Thm.rhs-of nth))
   in rth end
| Const(@{const-name HOL.less-eq},-)$a$b =>
   let val (ca,cb) = Thm.dest-binop ct
```

```
      val T = ctyp-of-term ca
      val th = instantiate' [SOME T] [SOME ca, SOME cb] le-iff-diff-le-0
      val nth = Conv.fconv-rule
        (Conv.arg-conv (Conv.arg1-conv
            (Normalizer.semiring-normalize-ord-conv @{context} (earlier vs)))) th
      val rth = transitive nth (xnormalize-conv ctxt vs (Thm.rhs-of nth))
  in rth end

| Const(op =,-)$a$b =>
  let val (ca,cb) = Thm.dest-binop ct
      val T = ctyp-of-term ca
      val th = instantiate' [SOME T] [SOME ca, SOME cb] eq-iff-diff-eq-0
      val nth = Conv.fconv-rule
        (Conv.arg-conv (Conv.arg1-conv
            (Normalizer.semiring-normalize-ord-conv @{context} (earlier vs)))) th
      val rth = transitive nth (xnormalize-conv ctxt vs (Thm.rhs-of nth))
  in rth end
| @{term Not} $(Const(op =,-)$a$b) => Conv.arg-conv (field-isolate-conv phi ctxt
vs) ct
| - => reflexive ct
end;

fun classfield-whatis phi =
 let
  fun h x t =
   case term-of t of
     Const(op =, -)$y$z => if term-of x aconv y then Ferrante-Rackoff-Data.Eq
                           else Ferrante-Rackoff-Data.Nox
  | @{term Not}$(Const(op =, -)$y$z) => if term-of x aconv y then Ferrante-Rackoff-Data.NEq
                           else Ferrante-Rackoff-Data.Nox
   | Const(@{const-name HOL.less},-)$y$z =>
      if term-of x aconv y then Ferrante-Rackoff-Data.Lt
       else if term-of x aconv z then Ferrante-Rackoff-Data.Gt
       else Ferrante-Rackoff-Data.Nox
   | Const (@{const-name HOL.less-eq},-)$y$z =>
        if term-of x aconv y then Ferrante-Rackoff-Data.Le
        else if term-of x aconv z then Ferrante-Rackoff-Data.Ge
        else Ferrante-Rackoff-Data.Nox
   | - => Ferrante-Rackoff-Data.Nox
 in h end;
fun class-field-ss phi =
   HOL-basic-ss addsimps ([@{thm linorder-not-less}, @{thm linorder-not-le}])
   addsplits [@{thm abs-split},@{thm split-max}, @{thm split-min}]

in
Ferrante-Rackoff-Data.funs @{thm class-ordered-field-dense-linear-order.ferrack-axiom}
 {isolate-conv = field-isolate-conv, whatis = classfield-whatis, simpset = class-field-ss}
end
⟩⟩
```

end

16 Fact: Factorial Function

theory *Fact*
imports *../Real/Real*
begin

consts *fact* :: *nat* => *nat*
primrec
fact-0: *fact 0 = 1*
fact-Suc: *fact (Suc n) = (Suc n) * fact n*

lemma *fact-gt-zero* [*simp*]: *0 < fact n*
by (*induct n*) *auto*

lemma *fact-not-eq-zero* [*simp*]: *fact n ≠ 0*
by *simp*

lemma *real-of-nat-fact-not-zero* [*simp*]: *real (fact n) ≠ 0*
by *auto*

lemma *real-of-nat-fact-gt-zero* [*simp*]: *0 < real(fact n)*
by *auto*

lemma *real-of-nat-fact-ge-zero* [*simp*]: *0 ≤ real(fact n)*
by *simp*

lemma *fact-ge-one* [*simp*]: *1 ≤ fact n*
by (*induct n*) *auto*

lemma *fact-mono*: *m ≤ n ==> fact m ≤ fact n*
apply (*drule le-imp-less-or-eq*)
apply (*auto dest*!: *less-imp-Suc-add*)
apply (*induct-tac k*, *auto*)
done

Note that *fact 0 = fact 1*

lemma *fact-less-mono*: [| *0 < m*; *m < n* |] ==> *fact m < fact n*
apply (*drule-tac m = m* **in** *less-imp-Suc-add*, *auto*)
apply (*induct-tac k*, *auto*)
done

lemma *inv-real-of-nat-fact-gt-zero* [*simp*]: *0 < inverse (real (fact n))*
by (*auto simp add*: *positive-imp-inverse-positive*)

lemma *inv-real-of-nat-fact-ge-zero* [*simp*]: $0 \leq inverse\ (real\ (fact\ n))$
by (*auto intro*: *order-less-imp-le*)

lemma *fact-diff-Suc* [*rule-format*]:
$n < Suc\ m \Longrightarrow fact\ (Suc\ m - n) = (Suc\ m - n) * fact\ (m - n)$
apply (*induct n arbitrary*: *m*)
apply *auto*
apply (*drule-tac x* = *m* − *1* **in** *meta-spec*, *auto*)
done

lemma *fact-num0* [*simp*]: *fact 0* = *1*
by *auto*

lemma *fact-num-eq-if*: *fact m* = (*if m*=*0 then 1 else m* ∗ *fact* (*m* − *1*))
by (*cases m*) *auto*

lemma *fact-add-num-eq-if*:
$fact\ (m + n) = (if\ m + n = 0\ then\ 1\ else\ (m + n) * fact\ (m + n - 1))$
by (*cases m* + *n*) *auto*

lemma *fact-add-num-eq-if2*:
$fact\ (m + n) = (if\ m = 0\ then\ fact\ n\ else\ (m + n) * fact\ ((m - 1) + n))$
by (*cases m*) *auto*

end

17 SEQ: Sequences and Convergence

theory *SEQ*
imports *../Real/Real*
begin

definition
Zseq :: [*nat* ⇒ *'a*::*real-normed-vector*] ⇒ *bool* **where**
— Standard definition of sequence converging to zero
Zseq X = (∀ *r*>*0*. ∃ *no*. ∀ *n*≥*no*. *norm* (*X n*) < *r*)

definition
LIMSEQ :: [*nat* => *'a*::*real-normed-vector*, *'a*] => *bool*
(((-)/ −−−−> (-)) [*60*, *60*] *60*) **where**
— Standard definition of convergence of sequence
X −−−−> *L* = (∀ *r*. *0* < *r* −−> (∃ *no*. ∀ *n*. *no* ≤ *n* −−> *norm* (*X n* − *L*) < *r*))

definition
lim :: (*nat* => *'a*::*real-normed-vector*) => *'a* **where**
— Standard definition of limit using choice operator

lim X = (THE L. X ----> L)

definition
convergent :: (nat => 'a::real-normed-vector) => bool **where**
— Standard definition of convergence
convergent X = (∃ L. X ----> L)

definition
Bseq :: (nat => 'a::real-normed-vector) => bool **where**
— Standard definition for bounded sequence
Bseq X = (∃ K>0.∀ n. norm (X n) ≤ K)

definition
monoseq :: (nat=>real)=>bool **where**
— Definition for monotonicity
monoseq X = ((∀ m. ∀ n≥m. X m ≤ X n) | (∀ m. ∀ n≥m. X n ≤ X m))

definition
subseq :: (nat => nat) => bool **where**
— Definition of subsequence
subseq f = (∀ m. ∀ n>m. (f m) < (f n))

definition
Cauchy :: (nat => 'a::real-normed-vector) => bool **where**
— Standard definition of the Cauchy condition
Cauchy X = (∀ e>0. ∃ M. ∀ m ≥ M. ∀ n ≥ M. norm (X m − X n) < e)

17.1 Bounded Sequences

lemma *BseqI'*: **assumes** *K*: ⋀*n. norm (X n) ≤ K* **shows** *Bseq X*
unfolding *Bseq-def*
proof (*intro exI conjI allI*)
show *0 < max K 1* **by** *simp*
next
fix *n*::*nat*
have *norm (X n) ≤ K* **by** (*rule K*)
thus *norm (X n) ≤ max K 1* **by** *simp*
qed

lemma *BseqE*: ⟦*Bseq X*; ⋀*K*. ⟦*0 < K*; ∀ *n. norm (X n) ≤ K*⟧ ⟹ *Q*⟧ ⟹ *Q*
unfolding *Bseq-def* **by** *auto*

lemma *BseqI2'*: **assumes** *K*: ∀ *n*≥*N. norm (X n) ≤ K* **shows** *Bseq X*
proof (*rule BseqI'*)
let *?A = norm ‘ X ‘ {..N}*
have *1*: *finite ?A* **by** *simp*
fix *n*::*nat*
show *norm (X n) ≤ max K (Max ?A)*
proof (*cases rule*: *linorder-le-cases*)

assume $n \geq N$
hence *norm* $(X\ n) \leq K$ **using** K **by** *simp*
thus *norm* $(X\ n) \leq max\ K\ (Max\ ?A)$ **by** *simp*
next
assume $n \leq N$
hence *norm* $(X\ n) \in ?A$ **by** *simp*
with *1* **have** *norm* $(X\ n) \leq Max\ ?A$ **by** (*rule Max-ge*)
thus *norm* $(X\ n) \leq max\ K\ (Max\ ?A)$ **by** *simp*
qed
qed

lemma *Bseq-ignore-initial-segment*: $Bseq\ X \Longrightarrow Bseq\ (\lambda n.\ X\ (n + k))$
unfolding *Bseq-def* **by** *auto*

lemma *Bseq-offset*: $Bseq\ (\lambda n.\ X\ (n + k)) \Longrightarrow Bseq\ X$
apply (*erule BseqE*)
apply (*rule-tac N=k* **and** *K=K* **in** *BseqI2'*)
apply *clarify*
apply (*drule-tac x=n − k* **in** *spec*, *simp*)
done

17.2 Sequences That Converge to Zero

lemma *ZseqI*:
$(\bigwedge r.\ 0 < r \Longrightarrow \exists no.\ \forall n \geq no.\ norm\ (X\ n) < r) \Longrightarrow Zseq\ X$
unfolding *Zseq-def* **by** *simp*

lemma *ZseqD*:
$[\![Zseq\ X;\ 0 < r]\!] \Longrightarrow \exists no.\ \forall n \geq no.\ norm\ (X\ n) < r$
unfolding *Zseq-def* **by** *simp*

lemma *Zseq-zero*: $Zseq\ (\lambda n.\ 0)$
unfolding *Zseq-def* **by** *simp*

lemma *Zseq-const-iff*: $Zseq\ (\lambda n.\ k) = (k = 0)$
unfolding *Zseq-def* **by** *force*

lemma *Zseq-norm-iff*: $Zseq\ (\lambda n.\ norm\ (X\ n)) = Zseq\ (\lambda n.\ X\ n)$
unfolding *Zseq-def* **by** *simp*

lemma *Zseq-imp-Zseq*:
assumes X: $Zseq\ X$
assumes Y: $\bigwedge n.\ norm\ (Y\ n) \leq norm\ (X\ n) * K$
shows $Zseq\ (\lambda n.\ Y\ n)$
proof (*cases*)
assume K: $0 < K$
show *?thesis*
proof (*rule ZseqI*)
fix r::*real* **assume** $0 < r$

```
    hence 0 < r / K
      using K by (rule divide-pos-pos)
    then obtain N where ∀ n≥N. norm (X n) < r / K
      using ZseqD [OF X] by fast
    hence ∀ n≥N. norm (X n) * K < r
      by (simp add: pos-less-divide-eq K)
    hence ∀ n≥N. norm (Y n) < r
      by (simp add: order-le-less-trans [OF Y])
    thus ∃ N. ∀ n≥N. norm (Y n) < r ..
  qed
next
  assume ¬ 0 < K
  hence K: K ≤ 0 by (simp only: linorder-not-less)
  {
    fix n::nat
    have norm (Y n) ≤ norm (X n) * K by (rule Y)
    also have ... ≤ norm (X n) * 0
      using K norm-ge-zero by (rule mult-left-mono)
    finally have norm (Y n) = 0 by simp
  }
  thus ?thesis by (simp add: Zseq-zero)
qed

lemma Zseq-le: ⟦Zseq Y; ∀ n. norm (X n) ≤ norm (Y n)⟧ ⟹ Zseq X
by (erule-tac K=1 in Zseq-imp-Zseq, simp)

lemma Zseq-add:
  assumes X: Zseq X
  assumes Y: Zseq Y
  shows Zseq (λn. X n + Y n)
proof (rule ZseqI)
  fix r::real assume 0 < r
  hence r: 0 < r / 2 by simp
  obtain M where M: ∀ n≥M. norm (X n) < r/2
    using ZseqD [OF X r] by fast
  obtain N where N: ∀ n≥N. norm (Y n) < r/2
    using ZseqD [OF Y r] by fast
  show ∃ N. ∀ n≥N. norm (X n + Y n) < r
  proof (intro exI allI impI)
    fix n assume n: max M N ≤ n
    have norm (X n + Y n) ≤ norm (X n) + norm (Y n)
      by (rule norm-triangle-ineq)
    also have ... < r/2 + r/2
    proof (rule add-strict-mono)
      from M n show norm (X n) < r/2 by simp
      from N n show norm (Y n) < r/2 by simp
    qed
    finally show norm (X n + Y n) < r by simp
  qed
```

qed

lemma *Zseq-minus*: *Zseq X* $\Longrightarrow$ *Zseq* ($\lambda n.$ $-$ *X n*)
unfolding *Zseq-def* **by** *simp*

lemma *Zseq-diff*: $[\![$*Zseq X*; *Zseq Y*$]\!] \Longrightarrow$ *Zseq* ($\lambda n.$ *X n* $-$ *Y n*)
by (*simp only*: *diff-minus Zseq-add Zseq-minus*)

lemma (**in** *bounded-linear*) *Zseq*:
assumes *X*: *Zseq X*
shows *Zseq* ($\lambda n.$ *f* (*X n*))
proof $-$
obtain *K* **where** $\bigwedge x.$ *norm* (*f x*) $\leq$ *norm x* $*$ *K*
using *bounded* **by** *fast*
with *X* **show** *?thesis*
by (*rule Zseq-imp-Zseq*)
qed

lemma (**in** *bounded-bilinear*) *Zseq*:
assumes *X*: *Zseq X*
assumes *Y*: *Zseq Y*
shows *Zseq* ($\lambda n.$ *X n* $**$ *Y n*)
proof (*rule ZseqI*)
fix *r*::*real* **assume** *r*: *0* $<$ *r*
obtain *K* **where** *K*: *0* $<$ *K*
and *norm-le*: $\bigwedge x\ y.$ *norm* (*x* $**$ *y*) $\leq$ *norm x* $*$ *norm y* $*$ *K*
using *pos-bounded* **by** *fast*
from *K* **have** *K'*: *0* $<$ *inverse K*
by (*rule positive-imp-inverse-positive*)
obtain *M* **where** *M*: $\forall\, n{\geq}M.$ *norm* (*X n*) $<$ *r*
using *ZseqD* [*OF X r*] **by** *fast*
obtain *N* **where** *N*: $\forall\, n{\geq}N.$ *norm* (*Y n*) $<$ *inverse K*
using *ZseqD* [*OF Y K'*] **by** *fast*
show $\exists\, N.\ \forall\, n{\geq}N.$ *norm* (*X n* $**$ *Y n*) $<$ *r*
proof (*intro exI allI impI*)
fix *n* **assume** *n*: *max M N* $\leq$ *n*
have *norm* (*X n* $**$ *Y n*) $\leq$ *norm* (*X n*) $*$ *norm* (*Y n*) $*$ *K*
by (*rule norm-le*)
also have *norm* (*X n*) $*$ *norm* (*Y n*) $*$ *K* $<$ *r* $*$ *inverse K* $*$ *K*
proof (*intro mult-strict-right-mono mult-strict-mono' norm-ge-zero K*)
from *M n* **show** *Xn*: *norm* (*X n*) $<$ *r* **by** *simp*
from *N n* **show** *Yn*: *norm* (*Y n*) $<$ *inverse K* **by** *simp*
qed
also from *K* **have** *r* $*$ *inverse K* $*$ *K* $=$ *r* **by** *simp*
finally show *norm* (*X n* $**$ *Y n*) $<$ *r* **.**
qed
qed

lemma (**in** *bounded-bilinear*) *Zseq-prod-Bseq*:

```
  assumes X: Zseq X
  assumes Y: Bseq Y
  shows Zseq (λn. X n ** Y n)
proof −
  obtain K where K: 0 ≤ K
    and norm-le: ⋀x y. norm (x ** y) ≤ norm x * norm y * K
    using nonneg-bounded by fast
  obtain B where B: 0 < B
    and norm-Y: ⋀n. norm (Y n) ≤ B
    using Y [unfolded Bseq-def] by fast
  from X show ?thesis
  proof (rule Zseq-imp-Zseq)
    fix n::nat
    have norm (X n ** Y n) ≤ norm (X n) * norm (Y n) * K
      by (rule norm-le)
    also have ... ≤ norm (X n) * B * K
      by (intro mult-mono' order-refl norm-Y norm-ge-zero
            mult-nonneg-nonneg K)
    also have ... = norm (X n) * (B * K)
      by (rule mult-assoc)
    finally show norm (X n ** Y n) ≤ norm (X n) * (B * K) .
  qed
qed

lemma (in bounded-bilinear) Bseq-prod-Zseq:
  assumes X: Bseq X
  assumes Y: Zseq Y
  shows Zseq (λn. X n ** Y n)
proof −
  obtain K where K: 0 ≤ K
    and norm-le: ⋀x y. norm (x ** y) ≤ norm x * norm y * K
    using nonneg-bounded by fast
  obtain B where B: 0 < B
    and norm-X: ⋀n. norm (X n) ≤ B
    using X [unfolded Bseq-def] by fast
  from Y show ?thesis
  proof (rule Zseq-imp-Zseq)
    fix n::nat
    have norm (X n ** Y n) ≤ norm (X n) * norm (Y n) * K
      by (rule norm-le)
    also have ... ≤ B * norm (Y n) * K
      by (intro mult-mono' order-refl norm-X norm-ge-zero
            mult-nonneg-nonneg K)
    also have ... = norm (Y n) * (B * K)
      by (simp only: mult-ac)
    finally show norm (X n ** Y n) ≤ norm (Y n) * (B * K) .
  qed
qed
```

lemma (**in** *bounded-bilinear*) *Zseq-left*:
Zseq X $\Longrightarrow$ *Zseq* ($\lambda n.$ *X n* ** *a*)
by (*rule bounded-linear-left* [*THEN bounded-linear.Zseq*])

lemma (**in** *bounded-bilinear*) *Zseq-right*:
Zseq X $\Longrightarrow$ *Zseq* ($\lambda n.$ *a* ** *X n*)
by (*rule bounded-linear-right* [*THEN bounded-linear.Zseq*])

lemmas *Zseq-mult* = *mult.Zseq*
lemmas *Zseq-mult-right* = *mult.Zseq-right*
lemmas *Zseq-mult-left* = *mult.Zseq-left*

17.3 Limits of Sequences

lemma *LIMSEQ-iff*:
(*X* ----> *L*) = ($\forall r{>}0.\ \exists no.\ \forall n \geq no.$ *norm* (*X n* − *L*) < *r*)
by (*rule LIMSEQ-def*)

lemma *LIMSEQ-Zseq-iff*: (($\lambda n.$ *X n*) ----> *L*) = *Zseq* ($\lambda n.$ *X n* − *L*)
by (*simp only*: *LIMSEQ-def Zseq-def*)

lemma *LIMSEQ-I*:
($\bigwedge r.$ *0* < *r* $\Longrightarrow$ $\exists no.\ \forall n{\geq}no.$ *norm* (*X n* − *L*) < *r*) $\Longrightarrow$ *X* ----> *L*
by (*simp add*: *LIMSEQ-def*)

lemma *LIMSEQ-D*:
⟦*X* ----> *L*; *0* < *r*⟧ $\Longrightarrow$ $\exists no.\ \forall n{\geq}no.$ *norm* (*X n* − *L*) < *r*
by (*simp add*: *LIMSEQ-def*)

lemma *LIMSEQ-const*: ($\lambda n.$ *k*) ----> *k*
by (*simp add*: *LIMSEQ-def*)

lemma *LIMSEQ-const-iff*: ($\lambda n.$ *k*) ----> *l* = (*k* = *l*)
by (*simp add*: *LIMSEQ-Zseq-iff Zseq-const-iff*)

lemma *LIMSEQ-norm*: *X* ----> *a* $\Longrightarrow$ ($\lambda n.$ *norm* (*X n*)) ----> *norm a*
apply (*simp add*: *LIMSEQ-def*, *safe*)
apply (*drule-tac x*=*r* **in** *spec*, *safe*)
apply (*rule-tac x*=*no* **in** *exI*, *safe*)
apply (*drule-tac x*=*n* **in** *spec*, *safe*)
apply (*erule order-le-less-trans* [*OF norm-triangle-ineq3*])
done

lemma *LIMSEQ-ignore-initial-segment*:
f ----> *a* $\Longrightarrow$ ($\lambda n.$ *f* (*n* + *k*)) ----> *a*
apply (*rule LIMSEQ-I*)
apply (*drule* (*1*) *LIMSEQ-D*)
apply (*erule exE*, *rename-tac N*)
apply (*rule-tac x*=*N* **in** *exI*)

apply *simp*
done

lemma *LIMSEQ-offset*:
(λ*n*. *f* (*n* + *k*)) −−−−> *a* ⟹ *f* −−−−> *a*
apply (*rule LIMSEQ-I*)
apply (*drule* (*1*) *LIMSEQ-D*)
apply (*erule exE*, *rename-tac N*)
apply (*rule-tac x*=*N* + *k* **in** *exI*)
apply *clarify*
apply (*drule-tac x*=*n* − *k* **in** *spec*)
apply (*simp add*: *le-diff-conv2*)
done

lemma *LIMSEQ-Suc*: *f* −−−−> *l* ⟹ (λ*n*. *f* (*Suc n*)) −−−−> *l*
by (*drule-tac k*=*1* **in** *LIMSEQ-ignore-initial-segment*, *simp*)

lemma *LIMSEQ-imp-Suc*: (λ*n*. *f* (*Suc n*)) −−−−> *l* ⟹ *f* −−−−> *l*
by (*rule-tac k*=*1* **in** *LIMSEQ-offset*, *simp*)

lemma *LIMSEQ-Suc-iff*: (λ*n*. *f* (*Suc n*)) −−−−> *l* = *f* −−−−> *l*
by (*blast intro*: *LIMSEQ-imp-Suc LIMSEQ-Suc*)

lemma *add-diff-add*:
fixes *a b c d* :: *'a*::*ab-group-add*
shows (*a* + *c*) − (*b* + *d*) = (*a* − *b*) + (*c* − *d*)
by *simp*

lemma *minus-diff-minus*:
fixes *a b* :: *'a*::*ab-group-add*
shows (− *a*) − (− *b*) = − (*a* − *b*)
by *simp*

lemma *LIMSEQ-add*: ⟦*X* −−−−> *a*; *Y* −−−−> *b*⟧ ⟹ (λ*n*. *X n* + *Y n*)
−−−−> *a* + *b*
by (*simp only*: *LIMSEQ-Zseq-iff add-diff-add Zseq-add*)

lemma *LIMSEQ-minus*: *X* −−−−> *a* ⟹ (λ*n*. − *X n*) −−−−> − *a*
by (*simp only*: *LIMSEQ-Zseq-iff minus-diff-minus Zseq-minus*)

lemma *LIMSEQ-minus-cancel*: (λ*n*. − *X n*) −−−−> − *a* ⟹ *X* −−−−> *a*
by (*drule LIMSEQ-minus*, *simp*)

lemma *LIMSEQ-diff*: ⟦*X* −−−−> *a*; *Y* −−−−> *b*⟧ ⟹ (λ*n*. *X n* − *Y n*)
−−−−> *a* − *b*
by (*simp add*: *diff-minus LIMSEQ-add LIMSEQ-minus*)

lemma *LIMSEQ-unique*: ⟦*X* −−−−> *a*; *X* −−−−> *b*⟧ ⟹ *a* = *b*
by (*drule* (*1*) *LIMSEQ-diff*, *simp add*: *LIMSEQ-const-iff*)

lemma (**in** *bounded-linear*) *LIMSEQ*:
  $X \longrightarrow a \Longrightarrow (\lambda n.\ f\ (X\ n)) \longrightarrow f\ a$
by (*simp only*: *LIMSEQ-Zseq-iff diff* [*symmetric*] *Zseq*)

lemma (**in** *bounded-bilinear*) *LIMSEQ*:
  $[\![X \longrightarrow a;\ Y \longrightarrow b]\!] \Longrightarrow (\lambda n.\ X\ n\ **\ Y\ n) \longrightarrow a\ **\ b$
by (*simp only*: *LIMSEQ-Zseq-iff prod-diff-prod*
        Zseq-add Zseq Zseq-left Zseq-right)

lemma *LIMSEQ-mult*:
  fixes *a b* :: *'a*::*real-normed-algebra*
  shows [| $X \longrightarrow a$; $Y \longrightarrow b$ |] ==> $(\%n.\ X\ n * Y\ n) \longrightarrow a * b$
by (*rule mult.LIMSEQ*)

lemma *inverse-diff-inverse*:
  $[\![(a::'a::division\text{-}ring) \neq 0;\ b \neq 0]\!]$
   $\Longrightarrow inverse\ a - inverse\ b = -\ (inverse\ a * (a - b) * inverse\ b)$
by (*simp add*: *ring-simps*)

lemma *Bseq-inverse-lemma*:
  fixes *x* :: *'a*::*real-normed-div-algebra*
  shows $[\![r \leq norm\ x;\ 0 < r]\!] \Longrightarrow norm\ (inverse\ x) \leq inverse\ r$
apply (*subst nonzero-norm-inverse*, *clarsimp*)
apply (*erule* (*1*) *le-imp-inverse-le*)
done

lemma *Bseq-inverse*:
  fixes *a* :: *'a*::*real-normed-div-algebra*
  assumes *X*: $X \longrightarrow a$
  assumes *a*: $a \neq 0$
  shows *Bseq* $(\lambda n.\ inverse\ (X\ n))$
proof −
  from *a* **have** $0 < norm\ a$ **by** *simp*
  hence $\exists r{>}0.\ r < norm\ a$ **by** (*rule dense*)
  then obtain *r* **where** *r1*: $0 < r$ **and** *r2*: $r < norm\ a$ **by** *fast*
  obtain *N* **where** *N*: $\bigwedge n.\ N \leq n \Longrightarrow norm\ (X\ n - a) < r$
    using *LIMSEQ-D* [*OF X r1*] **by** *fast*
  show *?thesis*
  proof (*rule BseqI2'* [*rule-format*])
    fix *n* **assume** *n*: $N \leq n$
    hence *1*: $norm\ (X\ n - a) < r$ **by** (*rule N*)
    hence *2*: $X\ n \neq 0$ **using** *r2* **by** *auto*
    hence $norm\ (inverse\ (X\ n)) = inverse\ (norm\ (X\ n))$
      by (*rule nonzero-norm-inverse*)
    also have $\ldots \leq inverse\ (norm\ a - r)$
    proof (*rule le-imp-inverse-le*)
      show $0 < norm\ a - r$ **using** *r2* **by** *simp*
    next

```
    have norm a − norm (X n) ≤ norm (a − X n)
      by (rule norm-triangle-ineq2)
    also have ... = norm (X n − a)
      by (rule norm-minus-commute)
    also have ... < r using 1 .
    finally show norm a − r ≤ norm (X n) by simp
   qed
   finally show norm (inverse (X n)) ≤ inverse (norm a − r) .
  qed
qed

lemma LIMSEQ-inverse-lemma:
  fixes a :: 'a::real-normed-div-algebra
  shows ⟦X ----> a; a ≠ 0; ∀ n. X n ≠ 0⟧
        ⟹ (λn. inverse (X n)) ----> inverse a
apply (subst LIMSEQ-Zseq-iff)
apply (simp add: inverse-diff-inverse nonzero-imp-inverse-nonzero)
apply (rule Zseq-minus)
apply (rule Zseq-mult-left)
apply (rule mult.Bseq-prod-Zseq)
apply (erule (1) Bseq-inverse)
apply (simp add: LIMSEQ-Zseq-iff)
done

lemma LIMSEQ-inverse:
  fixes a :: 'a::real-normed-div-algebra
  assumes X: X ----> a
  assumes a: a ≠ 0
  shows (λn. inverse (X n)) ----> inverse a
proof −
  from a have 0 < norm a by simp
  then obtain k where ∀ n≥k. norm (X n − a) < norm a
    using LIMSEQ-D [OF X] by fast
  hence ∀ n≥k. X n ≠ 0 by auto
  hence k: ∀ n. X (n + k) ≠ 0 by simp

  from X have (λn. X (n + k)) ----> a
    by (rule LIMSEQ-ignore-initial-segment)
  hence (λn. inverse (X (n + k))) ----> inverse a
    using a k by (rule LIMSEQ-inverse-lemma)
  thus (λn. inverse (X n)) ----> inverse a
    by (rule LIMSEQ-offset)
qed

lemma LIMSEQ-divide:
  fixes a b :: 'a::real-normed-field
  shows ⟦X ----> a; Y ----> b; b ≠ 0⟧ ⟹ (λn. X n / Y n) ----> a
/ b
by (simp add: LIMSEQ-mult LIMSEQ-inverse divide-inverse)
```

lemma *LIMSEQ-pow*:
fixes $a :: 'a$::{*real-normed-algebra*,*recpower*}
shows X ----> $a \Longrightarrow (\lambda n.\ (X\ n)\ \hat{}\ m)$ ----> $a\ \hat{}\ m$
by (*induct m*) (*simp-all add*: *power-Suc LIMSEQ-const LIMSEQ-mult*)

lemma *LIMSEQ-setsum*:
assumes *n*: $\bigwedge n.\ n \in S \Longrightarrow X\ n$ ----> $L\ n$
shows $(\lambda m.\ \sum n{\in}S.\ X\ n\ m)$ ----> $(\sum n{\in}S.\ L\ n)$
proof (*cases finite S*)
case *True*
thus *?thesis* **using** *n*
proof (*induct*)
case *empty*
show *?case*
by (*simp add*: *LIMSEQ-const*)
next
case *insert*
thus *?case*
by (*simp add*: *LIMSEQ-add*)
qed
next
case *False*
thus *?thesis*
by (*simp add*: *LIMSEQ-const*)
qed

lemma *LIMSEQ-setprod*:
fixes $L :: 'a \Rightarrow 'b$::{*real-normed-algebra*,*comm-ring-1*}
assumes *n*: $\bigwedge n.\ n \in S \Longrightarrow X\ n$ ----> $L\ n$
shows $(\lambda m.\ \prod n{\in}S.\ X\ n\ m)$ ----> $(\prod n{\in}S.\ L\ n)$
proof (*cases finite S*)
case *True*
thus *?thesis* **using** *n*
proof (*induct*)
case *empty*
show *?case*
by (*simp add*: *LIMSEQ-const*)
next
case *insert*
thus *?case*
by (*simp add*: *LIMSEQ-mult*)
qed
next
case *False*
thus *?thesis*
by (*simp add*: *setprod-def LIMSEQ-const*)
qed

lemma *LIMSEQ-add-const*: $f \longrightarrow a \Longrightarrow (\%n.(f\ n + b)) \longrightarrow a + b$
by (*simp add*: *LIMSEQ-add LIMSEQ-const*)

lemma *LIMSEQ-add-minus*:
$[\!|\ X \longrightarrow a;\ Y \longrightarrow b\ |\!] \Longrightarrow (\%n.\ X\ n + -Y\ n) \longrightarrow a + -b$
by (*simp only*: *LIMSEQ-add LIMSEQ-minus*)

lemma *LIMSEQ-diff-const*: $f \longrightarrow a \Longrightarrow (\%n.(f\ n\ - b)) \longrightarrow a - b$
by (*simp add*: *LIMSEQ-diff LIMSEQ-const*)

lemma *LIMSEQ-diff-approach-zero*:
$g \longrightarrow L \Longrightarrow (\%x.\ f\ x - g\ x) \longrightarrow 0 \Longrightarrow$
$f \longrightarrow L$
apply (*drule LIMSEQ-add*)
apply *assumption*
apply *simp*
done

lemma *LIMSEQ-diff-approach-zero2*:
$f \longrightarrow L \Longrightarrow (\%x.\ f\ x - g\ x) \longrightarrow 0 \Longrightarrow$
$g \longrightarrow L$
apply (*drule LIMSEQ-diff*)
apply *assumption*
apply *simp*
done

A sequence tends to zero iff its abs does

lemma *LIMSEQ-norm-zero*: $((\lambda n.\ norm\ (X\ n)) \longrightarrow 0) = (X \longrightarrow 0)$
by (*simp add*: *LIMSEQ-def*)

lemma *LIMSEQ-rabs-zero*: $((\%n.\ |f\ n|) \longrightarrow 0) = (f \longrightarrow (0::real))$
by (*simp add*: *LIMSEQ-def*)

lemma *LIMSEQ-imp-rabs*: $f \longrightarrow (l::real) \Longrightarrow (\%n.\ |f\ n|) \longrightarrow |l|$
by (*drule LIMSEQ-norm*, *simp*)

An unbounded sequence's inverse tends to 0

lemma *LIMSEQ-inverse-zero*:
$\forall r::real.\ \exists N.\ \forall n \geq N.\ r < X\ n \Longrightarrow (\lambda n.\ inverse\ (X\ n)) \longrightarrow 0$
apply (*rule LIMSEQ-I*)
apply (*drule-tac x=inverse r* **in** *spec*, *safe*)
apply (*rule-tac x=N* **in** *exI*, *safe*)
apply (*drule-tac x=n* **in** *spec*, *safe*)
apply (*frule positive-imp-inverse-positive*)
apply (*frule (1) less-imp-inverse-less*)
apply (*subgoal-tac 0 < X n*, *simp*)
apply (*erule (1) order-less-trans*)
done

The sequence $(1::'a)\ /\ n$ tends to 0 as n tends to infinity

lemma *LIMSEQ-inverse-real-of-nat*: $(\%n.\ inverse(real(Suc\ n)))\ ----->\ 0$
apply (*rule LIMSEQ-inverse-zero*, *safe*)
apply (*cut-tac* $x = r$ **in** *reals-Archimedean2*)
apply (*safe*, *rule-tac* $x = n$ **in** *exI*)
apply (*auto simp add*: *real-of-nat-Suc*)
done

The sequence $r + (1::'a)\ /\ n$ tends to r as n tends to infinity is now easily proved

lemma *LIMSEQ-inverse-real-of-nat-add*:
$(\%n.\ r + inverse(real(Suc\ n)))\ ----->\ r$
by (*cut-tac LIMSEQ-add* [*OF LIMSEQ-const LIMSEQ-inverse-real-of-nat*], *auto*)

lemma *LIMSEQ-inverse-real-of-nat-add-minus*:
$(\%n.\ r + -inverse(real(Suc\ n)))\ ----->\ r$
by (*cut-tac LIMSEQ-add-minus* [*OF LIMSEQ-const LIMSEQ-inverse-real-of-nat*], *auto*)

lemma *LIMSEQ-inverse-real-of-nat-add-minus-mult*:
$(\%n.\ r*(\ 1 + -inverse(real(Suc\ n))))\ ----->\ r$
by (*cut-tac* $b=1$ **in**
LIMSEQ-mult [*OF LIMSEQ-const LIMSEQ-inverse-real-of-nat-add-minus*], *auto*)

lemma *LIMSEQ-le-const*:
$[\![X\ ----->\ (x::real);\ \exists N.\ \forall n{\geq}N.\ a \leq X\ n]\!] \Longrightarrow a \leq x$
apply (*rule ccontr*, *simp only*: *linorder-not-le*)
apply (*drule-tac* $r=a - x$ **in** *LIMSEQ-D*, *simp*)
apply *clarsimp*
apply (*drule-tac* $x=max\ N\ no$ **in** *spec*, *drule mp*, *rule le-maxI1*)
apply (*drule-tac* $x=max\ N\ no$ **in** *spec*, *drule mp*, *rule le-maxI2*)
apply *simp*
done

lemma *LIMSEQ-le-const2*:
$[\![X\ ----->\ (x::real);\ \exists N.\ \forall n{\geq}N.\ X\ n \leq a]\!] \Longrightarrow x \leq a$
apply (*subgoal-tac* $-\ a \leq -\ x$, *simp*)
apply (*rule LIMSEQ-le-const*)
apply (*erule LIMSEQ-minus*)
apply *simp*
done

lemma *LIMSEQ-le*:
$[\![X\ ----->\ x;\ Y\ ----->\ y;\ \exists N.\ \forall n{\geq}N.\ X\ n \leq Y\ n]\!] \Longrightarrow x \leq (y::real)$
apply (*subgoal-tac* $0 \leq y - x$, *simp*)
apply (*rule LIMSEQ-le-const*)
apply (*erule* (*1*) *LIMSEQ-diff*)
apply (*simp add*: *le-diff-eq*)

done

17.4 Convergence

lemma *limI*: X $----> L$ ==> $lim\ X = L$
apply (*simp add*: *lim-def*)
apply (*blast intro*: *LIMSEQ-unique*)
done

lemma *convergentD*: *convergent* X ==> $\exists L.\ (X$ $----> L)$
by (*simp add*: *convergent-def*)

lemma *convergentI*: $(X$ $----> L)$ ==> *convergent* X
by (*auto simp add*: *convergent-def*)

lemma *convergent-LIMSEQ-iff*: *convergent* $X = (X$ $----> lim\ X)$
by (*auto intro*: *theI LIMSEQ-unique simp add*: *convergent-def lim-def*)

lemma *convergent-minus-iff*: (*convergent* X) = (*convergent* $(\%n.\ -(X\ n))$)
apply (*simp add*: *convergent-def*)
apply (*auto dest*: *LIMSEQ-minus*)
apply (*drule LIMSEQ-minus*, *auto*)
done

17.5 Bounded Monotonic Sequences

Subsequence (alternative definition, (e.g. Hoskins)

lemma *subseq-Suc-iff*: *subseq* $f = (\forall n.\ (f\ n) < (f\ (Suc\ n)))$
apply (*simp add*: *subseq-def*)
apply (*auto dest*!: *less-imp-Suc-add*)
apply (*induct-tac k*)
apply (*auto intro*: *less-trans*)
done

lemma *monoseq-Suc*:
  $monoseq\ X = ((\forall n.\ X\ n \leq X\ (Suc\ n))$
    $| (\forall n.\ X\ (Suc\ n) \leq X\ n))$
apply (*simp add*: *monoseq-def*)
apply (*auto dest*!: *le-imp-less-or-eq*)
apply (*auto intro*!: *lessI* [*THEN less-imp-le*] *dest*!: *less-imp-Suc-add*)
apply (*induct-tac ka*)
apply (*auto intro*: *order-trans*)
apply (*erule contrapos-np*)
apply (*induct-tac k*)
apply (*auto intro*: *order-trans*)
done

lemma *monoI1*: $\forall m.\ \forall\ n \geq m.\ X\ m \leq X\ n$ ==> *monoseq* X
by (*simp add*: *monoseq-def*)

lemma *monoI2*: $\forall\, m.\ \forall\ n \geq m.\ X\ n \leq X\ m \Longrightarrow monoseq\ X$
by (*simp add*: *monoseq-def*)

lemma *mono-SucI1*: $\forall\, n.\ X\ n \leq X\ (Suc\ n) \Longrightarrow monoseq\ X$
by (*simp add*: *monoseq-Suc*)

lemma *mono-SucI2*: $\forall\, n.\ X\ (Suc\ n) \leq X\ n \Longrightarrow monoseq\ X$
by (*simp add*: *monoseq-Suc*)

Bounded Sequence

lemma *BseqD*: $Bseq\ X \Longrightarrow \exists\, K.\ 0 < K\ \&\ (\forall\, n.\ norm\ (X\ n) \leq K)$
by (*simp add*: *Bseq-def*)

lemma *BseqI*: $[|\ 0 < K;\ \forall\, n.\ norm\ (X\ n) \leq K\ |] \Longrightarrow Bseq\ X$
by (*auto simp add*: *Bseq-def*)

lemma *lemma-NBseq-def*:
$(\exists\, K > 0.\ \forall\, n.\ norm\ (X\ n) \leq K) =$
$(\exists\, N.\ \forall\, n.\ norm\ (X\ n) \leq real(Suc\ N))$
apply *auto*
prefer *2* **apply** *force*
apply (*cut-tac x* = *K* **in** *reals-Archimedean2*, *clarify*)
apply (*rule-tac x* = *n* **in** *exI*, *clarify*)
apply (*drule-tac x* = *na* **in** *spec*)
apply (*auto simp add*: *real-of-nat-Suc*)
done

alternative definition for Bseq

lemma *Bseq-iff*: $Bseq\ X = (\exists\, N.\ \forall\, n.\ norm\ (X\ n) \leq real(Suc\ N))$
apply (*simp add*: *Bseq-def*)
apply (*simp* (*no-asm*) *add*: *lemma-NBseq-def*)
done

lemma *lemma-NBseq-def2*:
$(\exists\, K > 0.\ \forall\, n.\ norm\ (X\ n) \leq K) = (\exists\, N.\ \forall\, n.\ norm\ (X\ n) < real(Suc\ N))$
apply (*subst lemma-NBseq-def*, *auto*)
apply (*rule-tac x* = *Suc N* **in** *exI*)
apply (*rule-tac* [*2*] *x* = *N* **in** *exI*)
apply (*auto simp add*: *real-of-nat-Suc*)
prefer *2* **apply** (*blast intro*: *order-less-imp-le*)
apply (*drule-tac x* = *n* **in** *spec*, *simp*)
done

lemma *Bseq-iff1a*: $Bseq\ X = (\exists\, N.\ \forall\, n.\ norm\ (X\ n) < real(Suc\ N))$
by (*simp add*: *Bseq-def lemma-NBseq-def2*)

17.5.1 Upper Bounds and Lubs of Bounded Sequences

lemma *Bseq-isUb*:
!!(*X*::*nat*=>*real*). *Bseq X* ==> ∃ *U*. *isUb* (*UNIV*::*real set*) {*x*. ∃ *n*. *X n* = *x*} *U*
by (*auto intro*: *isUbI setleI simp add*: *Bseq-def abs-le-iff*)

Use completeness of reals (supremum property) to show that any bounded sequence has a least upper bound

lemma *Bseq-isLub*:
!!(*X*::*nat*=>*real*). *Bseq X* ==>
∃ *U*. *isLub* (*UNIV*::*real set*) {*x*. ∃ *n*. *X n* = *x*} *U*
by (*blast intro*: *reals-complete Bseq-isUb*)

17.5.2 A Bounded and Monotonic Sequence Converges

lemma *lemma-converg1*:
!!(*X*::*nat*=>*real*). [| ∀ *m*. ∀ *n* ≥ *m*. *X m* ≤ *X n*;
isLub (*UNIV*::*real set*) {*x*. ∃ *n*. *X n* = *x*} (*X ma*)
|] ==> ∀ *n* ≥ *ma*. *X n* = *X ma*
apply *safe*
apply (*drule-tac y* = *X n* **in** *isLubD2*)
apply (*blast dest*: *order-antisym*)+
done

The best of both worlds: Easier to prove this result as a standard theorem and then use equivalence to "transfer" it into the equivalent nonstandard form if needed!

lemma *Bmonoseq-LIMSEQ*: ∀ *n*. *m* ≤ *n* --> *X n* = *X m* ==> ∃ *L*. (*X* ----> *L*)
apply (*simp add*: *LIMSEQ-def*)
apply (*rule-tac x* = *X m* **in** *exI*, *safe*)
apply (*rule-tac x* = *m* **in** *exI*, *safe*)
apply (*drule spec*, *erule impE*, *auto*)
done

lemma *lemma-converg2*:
!!(*X*::*nat*=>*real*).
[| ∀ *m*. *X m* ~= *U*; *isLub UNIV* {*x*. ∃ *n*. *X n* = *x*} *U* |] ==> ∀ *m*. *X m* < *U*
apply *safe*
apply (*drule-tac y* = *X m* **in** *isLubD2*)
apply (*auto dest*!: *order-le-imp-less-or-eq*)
done

lemma *lemma-converg3*: !!(*X* ::*nat*=>*real*). ∀ *m*. *X m* ≤ *U* ==> *isUb UNIV* {*x*. ∃ *n*. *X n* = *x*} *U*
by (*rule setleI* [*THEN isUbI*], *auto*)

FIXME: *U* − *T* < *U* is redundant

lemma *lemma-converg4*: !!(*X*::*nat*=> *real*).

```
[| ∀ m. X m ~= U;
   isLub UNIV {x. ∃ n. X n = x} U;
   0 < T;
   U + - T < U
|] ==> ∃ m. U + -T < X m & X m < U
apply (drule lemma-converg2, assumption)
apply (rule ccontr, simp)
apply (simp add: linorder-not-less)
apply (drule lemma-converg3)
apply (drule isLub-le-isUb, assumption)
apply (auto dest: order-less-le-trans)
done
```

A standard proof of the theorem for monotone increasing sequence

```
lemma Bseq-mono-convergent:
    [| Bseq X; ∀ m. ∀ n ≥ m. X m ≤ X n |] ==> convergent (X::nat=>real)
apply (simp add: convergent-def)
apply (frule Bseq-isLub, safe)
apply (case-tac ∃ m. X m = U, auto)
apply (blast dest: lemma-converg1 Bmonoseq-LIMSEQ)

apply (rule-tac x = U in exI)
apply (subst LIMSEQ-iff, safe)
apply (frule lemma-converg2, assumption)
apply (drule lemma-converg4, auto)
apply (rule-tac x = m in exI, safe)
apply (subgoal-tac X m ≤ X n)
 prefer 2 apply blast
apply (drule-tac x=n and P=%m. X m < U in spec, arith)
done

lemma Bseq-minus-iff: Bseq (%n. -(X n)) = Bseq X
by (simp add: Bseq-def)
```

Main monotonicity theorem

```
lemma Bseq-monoseq-convergent: [| Bseq X; monoseq X |] ==> convergent X
apply (simp add: monoseq-def, safe)
apply (rule-tac [2] convergent-minus-iff [THEN ssubst])
apply (drule-tac [2] Bseq-minus-iff [THEN ssubst])
apply (auto intro!: Bseq-mono-convergent)
done
```

17.5.3 A Few More Equivalence Theorems for Boundedness

alternative formulation for boundedness

```
lemma Bseq-iff2: Bseq X = (∃ k > 0. ∃ x. ∀ n. norm (X(n) + -x) ≤ k)
apply (unfold Bseq-def, safe)
apply (rule-tac [2] x = k + norm x in exI)
```

apply (*rule-tac* $x = K$ **in** *exI*, *simp*)
apply (*rule exI* [**where** $x = 0$], *auto*)
apply (*erule order-less-le-trans*, *simp*)
apply (*drule-tac* $x{=}n$ **in** *spec*, *fold diff-def*)
apply (*drule order-trans* [*OF norm-triangle-ineq2*])
apply *simp*
done

alternative formulation for boundedness

lemma *Bseq-iff3*: $Bseq\ X = (\exists k > 0.\ \exists N.\ \forall n.\ norm(X(n) + -X(N)) \leq k)$
apply *safe*
apply (*simp add*: *Bseq-def*, *safe*)
apply (*rule-tac* $x = K + norm\ (X\ N)$ **in** *exI*)
apply *auto*
apply (*erule order-less-le-trans*, *simp*)
apply (*rule-tac* $x = N$ **in** *exI*, *safe*)
apply (*drule-tac* $x = n$ **in** *spec*)
apply (*rule order-trans* [*OF norm-triangle-ineq*], *simp*)
apply (*auto simp add*: *Bseq-iff2*)
done

lemma *BseqI2*: $(\forall n.\ k \leq f\ n\ \&\ f\ n \leq (K::real)) ==> Bseq\ f$
apply (*simp add*: *Bseq-def*)
apply (*rule-tac* $x = (|k| + |K|) + 1$ **in** *exI*, *auto*)
apply (*drule-tac* $x = n$ **in** *spec*, *arith*)
done

17.6 Cauchy Sequences

lemma *CauchyI*:
$(\bigwedge e.\ 0 < e \Longrightarrow \exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ norm\ (X\ m - X\ n) < e) \Longrightarrow Cauchy\ X$
by (*simp add*: *Cauchy-def*)

lemma *CauchyD*:
$[\![Cauchy\ X;\ 0 < e]\!] \Longrightarrow \exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ norm\ (X\ m - X\ n) < e$
by (*simp add*: *Cauchy-def*)

17.6.1 Cauchy Sequences are Bounded

A Cauchy sequence is bounded – this is the standard proof mechanization rather than the nonstandard proof

lemma *lemmaCauchy*: $\forall n \geq M.\ norm\ (X\ M - X\ n) < (1::real)$
$==>\ \forall n \geq M.\ norm\ (X\ n :: 'a::real\text{-}normed\text{-}vector) < 1 + norm\ (X\ M)$
apply (*clarify*, *drule spec*, *drule* (*1*) *mp*)
apply (*simp only*: *norm-minus-commute*)
apply (*drule order-le-less-trans* [*OF norm-triangle-ineq2*])
apply *simp*
done

lemma *Cauchy-Bseq*: *Cauchy X ==> Bseq X*
apply (*simp add*: *Cauchy-def*)
apply (*drule spec*, *drule mp*, *rule zero-less-one*, *safe*)
apply (*drule-tac x=M* **in** *spec*, *simp*)
apply (*drule lemmaCauchy*)
apply (*rule-tac k=M* **in** *Bseq-offset*)
apply (*simp add*: *Bseq-def*)
apply (*rule-tac x=1 + norm (X M)* **in** *exI*)
apply (*rule conjI*, *rule order-less-le-trans* [*OF zero-less-one*], *simp*)
apply (*simp add*: *order-less-imp-le*)
done

17.6.2 Cauchy Sequences are Convergent

axclass *banach* ⊆ *real-normed-vector*
Cauchy-convergent: *Cauchy X* ⟹ *convergent X*

theorem *LIMSEQ-imp-Cauchy*:
assumes *X*: *X* −−−−> *a* **shows** *Cauchy X*
proof (*rule CauchyI*)
fix *e*::*real* **assume** *0 < e*
hence *0 < e/2* **by** *simp*
with *X* **have** ∃ *N*. ∀ *n*≥*N*. *norm* (*X n* − *a*) < *e/2* **by** (*rule LIMSEQ-D*)
then obtain *N* **where** *N*: ∀ *n*≥*N*. *norm* (*X n* − *a*) < *e/2* **..**
show ∃ *N*. ∀ *m*≥*N*. ∀ *n*≥*N*. *norm* (*X m* − *X n*) < *e*
proof (*intro exI allI impI*)
fix *m* **assume** *N* ≤ *m*
hence *m*: *norm* (*X m* − *a*) < *e/2* **using** *N* **by** *fast*
fix *n* **assume** *N* ≤ *n*
hence *n*: *norm* (*X n* − *a*) < *e/2* **using** *N* **by** *fast*
have *norm* (*X m* − *X n*) = *norm* ((*X m* − *a*) − (*X n* − *a*)) **by** *simp*
also have ... ≤ *norm* (*X m* − *a*) + *norm* (*X n* − *a*)
by (*rule norm-triangle-ineq4*)
also from *m n* **have** ... < *e* **by**(*simp add*:*field-simps*)
finally show *norm* (*X m* − *X n*) < *e* **.**
qed
qed

lemma *convergent-Cauchy*: *convergent X* ⟹ *Cauchy X*
unfolding *convergent-def*
by (*erule exE*, *erule LIMSEQ-imp-Cauchy*)

Proof that Cauchy sequences converge based on the one from http://pirate.shu.edu/ wachsmut/ira/nu

If sequence *X* is Cauchy, then its limit is the lub of {*r*. ∃ *N*. ∀ *n*≥*N*. *r* < *X*
n}

lemma *isUb-UNIV-I*: (⋀*y*. *y* ∈ *S* ⟹ *y* ≤ *u*) ⟹ *isUb UNIV S u*
by (*simp add*: *isUbI setleI*)

lemma *real-abs-diff-less-iff*:
$(|x - a| < (r::real)) = (a - r < x \wedge x < a + r)$
by *auto*

locale (**open**) *real-Cauchy* =
fixes $X :: nat \Rightarrow real$
assumes X: *Cauchy X*
fixes $S :: real\ set$
defines *S-def*: $S \equiv \{x::real.\ \exists N.\ \forall n \geq N.\ x < X\ n\}$

lemma (**in** *real-Cauchy*) *mem-S*: $\forall n \geq N.\ x < X\ n \Longrightarrow x \in S$
by (*unfold S-def*, *auto*)

lemma (**in** *real-Cauchy*) *bound-isUb*:
assumes N: $\forall n \geq N.\ X\ n < x$
shows *isUb UNIV S x*
proof (*rule isUb-UNIV-I*)
fix $y::real$ **assume** $y \in S$
hence $\exists M.\ \forall n \geq M.\ y < X\ n$
by (*simp add*: *S-def*)
then obtain M **where** $\forall n \geq M.\ y < X\ n$ **..**
hence $y < X\ (max\ M\ N)$ **by** *simp*
also have $\ldots < x$ **using** N **by** *simp*
finally show $y \leq x$
by (*rule order-less-imp-le*)
qed

lemma (**in** *real-Cauchy*) *isLub-ex*: $\exists u.\ isLub\ UNIV\ S\ u$
proof (*rule reals-complete*)
obtain N **where** $\forall m \geq N.\ \forall n \geq N.\ norm\ (X\ m - X\ n) < 1$
using *CauchyD* [*OF X zero-less-one*] **by** *fast*
hence N: $\forall n \geq N.\ norm\ (X\ n - X\ N) < 1$ **by** *simp*
show $\exists x.\ x \in S$
proof
from N **have** $\forall n \geq N.\ X\ N - 1 < X\ n$
by (*simp add*: *real-abs-diff-less-iff*)
thus $X\ N - 1 \in S$ **by** (*rule mem-S*)
qed
show $\exists u.\ isUb\ UNIV\ S\ u$
proof
from N **have** $\forall n \geq N.\ X\ n < X\ N + 1$
by (*simp add*: *real-abs-diff-less-iff*)
thus *isUb UNIV S* $(X\ N + 1)$
by (*rule bound-isUb*)
qed
qed

lemma (**in** *real-Cauchy*) *isLub-imp-LIMSEQ*:
assumes x: *isLub UNIV S x*

```
  shows X ----> x
proof (rule LIMSEQ-I)
  fix r::real assume 0 < r
  hence r: 0 < r/2 by simp
  obtain N where ∀n≥N. ∀m≥N. norm (X n - X m) < r/2
    using CauchyD [OF X r] by fast
  hence ∀n≥N. norm (X n - X N) < r/2 by simp
  hence N: ∀n≥N. X N - r/2 < X n ∧ X n < X N + r/2
    by (simp only: real-norm-def real-abs-diff-less-iff)

  from N have ∀n≥N. X N - r/2 < X n by fast
  hence X N - r/2 ∈ S by (rule mem-S)
  hence 1: X N - r/2 ≤ x using x isLub-isUb isUbD by fast

  from N have ∀n≥N. X n < X N + r/2 by fast
  hence isUb UNIV S (X N + r/2) by (rule bound-isUb)
  hence 2: x ≤ X N + r/2 using x isLub-le-isUb by fast

  show ∃N. ∀n≥N. norm (X n - x) < r
  proof (intro exI allI impI)
    fix n assume n: N ≤ n
    from N n have X n < X N + r/2 and X N - r/2 < X n by simp+
    thus norm (X n - x) < r using 1 2
      by (simp add: real-abs-diff-less-iff)
  qed
qed

lemma (in real-Cauchy) LIMSEQ-ex: ∃x. X ----> x
proof -
  obtain x where isLub UNIV S x
    using isLub-ex by fast
  hence X ----> x
    by (rule isLub-imp-LIMSEQ)
  thus ?thesis ..
qed

lemma real-Cauchy-convergent:
  fixes X :: nat ⇒ real
  shows Cauchy X ⟹ convergent X
unfolding convergent-def by (rule real-Cauchy.LIMSEQ-ex)

instance real :: banach
by intro-classes (rule real-Cauchy-convergent)

lemma Cauchy-convergent-iff:
  fixes X :: nat ⇒ 'a::banach
  shows Cauchy X = convergent X
by (fast intro: Cauchy-convergent convergent-Cauchy)
```

17.7 Power Sequences

The sequence $x \hat{} n$ tends to 0 if $(0::'a) \leq x$ and $x < (1::'a)$. Proof will use (NS) Cauchy equivalence for convergence and also fact that bounded and monotonic sequence converges.

lemma *Bseq-realpow*: [| $0 \leq (x::real)$; $x \leq 1$ |] ==> *Bseq* (%n. $x \hat{} n$)
apply (*simp add*: *Bseq-def*)
apply (*rule-tac* $x = 1$ **in** *exI*)
apply (*simp add*: *power-abs*)
apply (*auto dest*: *power-mono*)
done

lemma *monoseq-realpow*: [| $0 \leq x$; $x \leq 1$ |] ==> *monoseq* (%n. $x \hat{} n$)
apply (*clarify intro*!: *mono-SucI2*)
apply (*cut-tac* $n = n$ **and** $N = Suc\ n$ **and** $a = x$ **in** *power-decreasing*, *auto*)
done

lemma *convergent-realpow*:
[| $0 \leq (x::real)$; $x \leq 1$ |] ==> *convergent* (%n. $x \hat{} n$)
by (*blast intro*!: *Bseq-monoseq-convergent Bseq-realpow monoseq-realpow*)

lemma *LIMSEQ-inverse-realpow-zero-lemma*:
fixes x :: *real*
assumes x: $0 \leq x$
shows *real* $n * x + 1 \leq (x + 1) \hat{} n$
apply (*induct* n)
apply *simp*
apply *simp*
apply (*rule order-trans*)
prefer *2*
apply (*erule mult-left-mono*)
apply (*rule add-increasing* [*OF* x], *simp*)
apply (*simp add*: *real-of-nat-Suc*)
apply (*simp add*: *ring-distribs*)
apply (*simp add*: *mult-nonneg-nonneg* x)
done

lemma *LIMSEQ-inverse-realpow-zero*:
$1 < (x::real) \Longrightarrow (\lambda n.\ inverse\ (x \hat{} n)) ----> 0$
proof (*rule LIMSEQ-inverse-zero* [*rule-format*])
fix y :: *real*
assume x: $1 < x$
hence $0 < x - 1$ **by** *simp*
hence $\forall y.\ \exists N::nat.\ y < real\ N * (x - 1)$
by (*rule reals-Archimedean3*)
hence $\exists N::nat.\ y < real\ N * (x - 1)$ **..**
then obtain $N::nat$ **where** $y < real\ N * (x - 1)$ **..**
also have $\ldots \leq real\ N * (x - 1) + 1$ **by** *simp*
also have $\ldots \leq (x - 1 + 1) \hat{} N$

by (*rule LIMSEQ-inverse-realpow-zero-lemma*, *cut-tac x*, *simp*)
also have $\ldots = x \hat{} N$ **by** *simp*
finally have $y < x \hat{} N$ **.**
hence $\forall n \geq N.\ y < x \hat{} n$
apply *clarify*
apply (*erule order-less-le-trans*)
apply (*erule power-increasing*)
apply (*rule order-less-imp-le* [*OF x*])
done
thus $\exists N.\ \forall n \geq N.\ y < x \hat{} n$ **..**
qed

lemma *LIMSEQ-realpow-zero*:
$[\![0 \leq (x::real);\ x < 1]\!] \Longrightarrow (\lambda n.\ x \hat{} n) \longrightarrow 0$
proof (*cases*)
assume $x = 0$
hence $(\lambda n.\ x \hat{} Suc\ n) \longrightarrow 0$ **by** (*simp add*: *LIMSEQ-const*)
thus *?thesis* **by** (*rule LIMSEQ-imp-Suc*)
next
assume $0 \leq x$ **and** $x \neq 0$
hence *x0*: $0 < x$ **by** *simp*
assume *x1*: $x < 1$
from *x0 x1* **have** $1 < inverse\ x$
by (*rule real-inverse-gt-one*)
hence $(\lambda n.\ inverse\ (inverse\ x \hat{} n)) \longrightarrow 0$
by (*rule LIMSEQ-inverse-realpow-zero*)
thus *?thesis* **by** (*simp add*: *power-inverse*)
qed

lemma *LIMSEQ-power-zero*:
fixes $x :: 'a::\{real\text{-}normed\text{-}algebra\text{-}1, recpower\}$
shows $norm\ x < 1 \Longrightarrow (\lambda n.\ x \hat{} n) \longrightarrow 0$
apply (*drule LIMSEQ-realpow-zero* [*OF norm-ge-zero*])
apply (*simp only*: *LIMSEQ-Zseq-iff*, *erule Zseq-le*)
apply (*simp add*: *power-abs norm-power-ineq*)
done

lemma *LIMSEQ-divide-realpow-zero*:
$1 < (x::real) \Longrightarrow (\%n.\ a\ /\ (x \hat{} n)) \longrightarrow 0$
apply (*cut-tac* $a = a$ **and** *x1* $= inverse\ x$ **in**
LIMSEQ-mult [*OF LIMSEQ-const LIMSEQ-realpow-zero*])
apply (*auto simp add*: *divide-inverse power-inverse*)
apply (*simp add*: *inverse-eq-divide pos-divide-less-eq*)
done

Limit of $c \hat{} n$ for $|c| < (1::'a)$

lemma *LIMSEQ-rabs-realpow-zero*: $|c| < (1::real) \Longrightarrow (\%n.\ |c| \hat{} n) \longrightarrow 0$
by (*rule LIMSEQ-realpow-zero* [*OF abs-ge-zero*])

lemma *LIMSEQ-rabs-realpow-zero2*: $|c| < (1::real) ==> (\%n.\ c\ \hat{}\ n) ----> 0$
apply (*rule LIMSEQ-rabs-zero* [*THEN iffD1*])
apply (*auto intro*: *LIMSEQ-rabs-realpow-zero simp add*: *power-abs*)
done

end

18 Series: Finite Summation and Infinite Series

theory *Series*
imports *SEQ*
begin

definition
  sums :: $(nat \Rightarrow 'a::real\text{-}normed\text{-}vector) \Rightarrow 'a \Rightarrow bool$
    (**infixr** *sums 80*) **where**
  $f\ sums\ s = (\%n.\ setsum\ f\ \{0..<n\}) ----> s$

definition
  summable :: $(nat \Rightarrow 'a::real\text{-}normed\text{-}vector) \Rightarrow bool$ **where**
  $summable\ f = (\exists s.\ f\ sums\ s)$

definition
  suminf :: $(nat \Rightarrow 'a::real\text{-}normed\text{-}vector) \Rightarrow 'a$ **where**
  $suminf\ f = (THE\ s.\ f\ sums\ s)$

syntax
  -suminf :: $idt \Rightarrow 'a \Rightarrow 'a$ $(\sum$-. - [0, 10] 10)
translations
  $\sum i.\ b == CONST\ suminf\ (\%i.\ b)$

lemma *sumr-diff-mult-const*:
 $setsum\ f\ \{0..<n\} - (real\ n*r) = setsum\ (\%i.\ f\ i - r)\ \{0..<n::nat\}$
by (*simp add*: *diff-minus setsum-addf real-of-nat-def*)

lemma *real-setsum-nat-ivl-bounded*:
    $(!!p.\ p < n \Longrightarrow f(p) \leq K)$
     $\Longrightarrow setsum\ f\ \{0..<n::nat\} \leq real\ n * K$
using *setsum-bounded*[**where** $A = \{0..<n\}$]
by (*auto simp*:*real-of-nat-def*)

lemma *sumr-minus-one-realpow-zero* [*simp*]:
  $(\sum i=0..<2*n.\ (-1)\ \hat{}\ Suc\ i) = (0::real)$
by (*induct n*, *auto*)

lemma *sumr-one-lb-realpow-zero* [*simp*]:
(∑ *n*=*Suc 0*..<*n*. *f*(*n*) * (*0*::*real*) ^ *n*) = *0*
by (*rule setsum-0'*, *simp*)

lemma *sumr-group*:
(∑ *m*=*0*..<*n*::*nat*. *setsum f* {*m* * *k* ..< *m**k* + *k*}) = *setsum f* {*0* ..< *n* * *k*}
apply (*subgoal-tac k* = *0* | *0* < *k*, *auto*)
apply (*induct n*)
apply (*simp-all add*: *setsum-add-nat-ivl add-commute*)
done

lemma *sumr-offset3*:
setsum f {*0*::*nat*..<*n*+*k*} = (∑ *m*=*0*..<*n*. *f* (*m*+*k*)) + *setsum f* {*0*..<*k*}
apply (*subst setsum-shift-bounds-nat-ivl* [*symmetric*])
apply (*simp add*: *setsum-add-nat-ivl add-commute*)
done

lemma *sumr-offset*:
fixes *f* :: *nat* ⇒ *'a*::*ab-group-add*
shows (∑ *m*=*0*..<*n*. *f*(*m*+*k*)) = *setsum f* {*0*..<*n*+*k*} − *setsum f* {*0*..<*k*}
by (*simp add*: *sumr-offset3*)

lemma *sumr-offset2*:
∀*f*. (∑ *m*=*0*..<*n*::*nat*. *f*(*m*+*k*)::*real*) = *setsum f* {*0*..<*n*+*k*} − *setsum f* {*0*..<*k*}
by (*simp add*: *sumr-offset*)

lemma *sumr-offset4*:
∀ *n f*. *setsum f* {*0*::*nat*..<*n*+*k*} = (∑ *m*=*0*..<*n*. *f* (*m*+*k*)::*real*) + *setsum f* {*0*..<*k*}
by (*clarify*, *rule sumr-offset3*)

18.1 Infinite Sums, by the Properties of Limits

lemma *sums-summable*: *f sums l* ==> *summable f*
by (*simp add*: *sums-def summable-def*, *blast*)

lemma *summable-sums*: *summable f* ==> *f sums* (*suminf f*)
apply (*simp add*: *summable-def suminf-def sums-def*)
apply (*blast intro*: *theI LIMSEQ-unique*)
done

lemma *summable-sumr-LIMSEQ-suminf*:
summable f ==> (%*n*. *setsum f* {*0*..<*n*}) ----> (*suminf f*)
by (*rule summable-sums* [*unfolded sums-def*])

lemma *sums-unique*: *f sums s* ==> (*s* = *suminf f*)
apply (*frule sums-summable* [*THEN summable-sums*])
apply (*auto intro*!: *LIMSEQ-unique simp add*: *sums-def*)

done

lemma *sums-split-initial-segment*: *f sums s ==>*
(%n. f(n + k)) sums (s − (SUM i = 0..< k. f i))
apply (*unfold sums-def*)
apply (*simp add*: *sumr-offset*)
apply (*rule LIMSEQ-diff-const*)
apply (*rule LIMSEQ-ignore-initial-segment*)
apply *assumption*
done

lemma *summable-ignore-initial-segment*: *summable f ==>*
summable (%n. f(n + k))
apply (*unfold summable-def*)
apply (*auto intro*: *sums-split-initial-segment*)
done

lemma *suminf-minus-initial-segment*: *summable f ==>*
suminf f = s ==> suminf (%n. f(n + k)) = s − (SUM i = 0..< k. f i)
apply (*frule summable-ignore-initial-segment*)
apply (*rule sums-unique* [*THEN sym*])
apply (*frule summable-sums*)
apply (*rule sums-split-initial-segment*)
apply *auto*
done

lemma *suminf-split-initial-segment*: *summable f ==>*
suminf f = (SUM i = 0..< k. f i) + suminf (%n. f(n + k))
by (*auto simp add*: *suminf-minus-initial-segment*)

lemma *series-zero*:
(∀ m. n ≤ m −−> f(m) = 0) ==> f sums (setsum f {0..<n})
apply (*simp add*: *sums-def LIMSEQ-def diff-minus*[*symmetric*], *safe*)
apply (*rule-tac x = n* **in** *exI*)
apply (*clarsimp simp add*:*setsum-diff*[*symmetric*] *cong*:*setsum-ivl-cong*)
done

lemma *sums-zero*: *(λn. 0) sums 0*
unfolding *sums-def* **by** (*simp add*: *LIMSEQ-const*)

lemma *summable-zero*: *summable (λn. 0)*
by (*rule sums-zero* [*THEN sums-summable*])

lemma *suminf-zero*: *suminf (λn. 0) = 0*
by (*rule sums-zero* [*THEN sums-unique*, *symmetric*])

lemma (**in** *bounded-linear*) *sums*:
(λn. X n) sums a ⟹ (λn. f (X n)) sums (f a)
unfolding *sums-def* **by** (*drule LIMSEQ*, *simp only*: *setsum*)

lemma (**in** *bounded-linear*) *summable*:
summable ($\lambda n.\ X\ n$) $\Longrightarrow$ *summable* ($\lambda n.\ f\ (X\ n)$)
unfolding *summable-def* **by** (*auto intro*: *sums*)

lemma (**in** *bounded-linear*) *suminf*:
summable ($\lambda n.\ X\ n$) $\Longrightarrow$ $f\ (\sum n.\ X\ n) = (\sum n.\ f\ (X\ n))$
by (*intro sums-unique sums summable-sums*)

lemma *sums-mult*:
fixes c :: *'a*::*real-normed-algebra*
shows *f sums a* $\Longrightarrow$ ($\lambda n.\ c * f\ n$) *sums* ($c * a$)
by (*rule mult-right.sums*)

lemma *summable-mult*:
fixes c :: *'a*::*real-normed-algebra*
shows *summable f* $\Longrightarrow$ *summable* ($\%n.\ c * f\ n$)
by (*rule mult-right.summable*)

lemma *suminf-mult*:
fixes c :: *'a*::*real-normed-algebra*
shows *summable f* $\Longrightarrow$ *suminf* ($\lambda n.\ c * f\ n$) $= c *$ *suminf f*
by (*rule mult-right.suminf* [*symmetric*])

lemma *sums-mult2*:
fixes c :: *'a*::*real-normed-algebra*
shows *f sums a* $\Longrightarrow$ ($\lambda n.\ f\ n * c$) *sums* ($a * c$)
by (*rule mult-left.sums*)

lemma *summable-mult2*:
fixes c :: *'a*::*real-normed-algebra*
shows *summable f* $\Longrightarrow$ *summable* ($\lambda n.\ f\ n * c$)
by (*rule mult-left.summable*)

lemma *suminf-mult2*:
fixes c :: *'a*::*real-normed-algebra*
shows *summable f* $\Longrightarrow$ *suminf f* $* c = (\sum n.\ f\ n * c)$
by (*rule mult-left.suminf*)

lemma *sums-divide*:
fixes c :: *'a*::*real-normed-field*
shows *f sums a* $\Longrightarrow$ ($\lambda n.\ f\ n\ /\ c$) *sums* ($a\ /\ c$)
by (*rule divide.sums*)

lemma *summable-divide*:
fixes c :: *'a*::*real-normed-field*
shows *summable f* $\Longrightarrow$ *summable* ($\lambda n.\ f\ n\ /\ c$)
by (*rule divide.summable*)

lemma *suminf-divide*:
fixes $c :: 'a$::*real-normed-field*
shows *summable* $f \Longrightarrow$ *suminf* $(\lambda n.\ f\ n\ /\ c) =$ *suminf* $f\ /\ c$
by (*rule divide.suminf* [*symmetric*])

lemma *sums-add*: $[\![X$ *sums* $a;\ Y$ *sums* $b]\!] \Longrightarrow (\lambda n.\ X\ n + Y\ n)$ *sums* $(a + b)$
unfolding *sums-def* **by** (*simp add*: *setsum-addf LIMSEQ-add*)

lemma *summable-add*: $[\![$*summable* X; *summable* $Y]\!] \Longrightarrow$ *summable* $(\lambda n.\ X\ n + Y\ n)$
unfolding *summable-def* **by** (*auto intro*: *sums-add*)

lemma *suminf-add*:
$[\![$*summable* X; *summable* $Y]\!] \Longrightarrow$ *suminf* $X +$ *suminf* $Y = (\sum n.\ X\ n + Y\ n)$
by (*intro sums-unique sums-add summable-sums*)

lemma *sums-diff*: $[\![X$ *sums* $a;\ Y$ *sums* $b]\!] \Longrightarrow (\lambda n.\ X\ n - Y\ n)$ *sums* $(a - b)$
unfolding *sums-def* **by** (*simp add*: *setsum-subtractf LIMSEQ-diff*)

lemma *summable-diff*: $[\![$*summable* X; *summable* $Y]\!] \Longrightarrow$ *summable* $(\lambda n.\ X\ n - Y\ n)$
unfolding *summable-def* **by** (*auto intro*: *sums-diff*)

lemma *suminf-diff*:
$[\![$*summable* X; *summable* $Y]\!] \Longrightarrow$ *suminf* $X -$ *suminf* $Y = (\sum n.\ X\ n - Y\ n)$
by (*intro sums-unique sums-diff summable-sums*)

lemma *sums-minus*: X *sums* a ==> $(\lambda n.\ -\ X\ n)$ *sums* $(-\ a)$
unfolding *sums-def* **by** (*simp add*: *setsum-negf LIMSEQ-minus*)

lemma *summable-minus*: *summable* $X \Longrightarrow$ *summable* $(\lambda n.\ -\ X\ n)$
unfolding *summable-def* **by** (*auto intro*: *sums-minus*)

lemma *suminf-minus*: *summable* $X \Longrightarrow (\sum n.\ -\ X\ n) = -\ (\sum n.\ X\ n)$
by (*intro sums-unique* [*symmetric*] *sums-minus summable-sums*)

lemma *sums-group*:
$[|$*summable* f; $0 < k\ |]$ ==> $(\%n.$ *setsum* $f\ \{n*k..<n*k+k\})$ *sums* $($*suminf* $f)$
apply (*drule summable-sums*)
apply (*simp only*: *sums-def sumr-group*)
apply (*unfold LIMSEQ-def*, *safe*)
apply (*drule-tac* x=r **in** *spec*, *safe*)
apply (*rule-tac* x=no **in** *exI*, *safe*)
apply (*drule-tac* x=$n*k$ **in** *spec*)
apply (*erule mp*)
apply (*erule order-trans*)
apply *simp*
done

A summable series of positive terms has limit that is at least as great as any

partial sum.

lemma *series-pos-le*:
  fixes f :: $nat \Rightarrow real$
  shows $[\![summable\ f; \forall\, m{\geq}n.\ 0 \leq f\ m]\!] \Longrightarrow setsum\ f\ \{0..{<}n\} \leq suminf\ f$
apply (*drule summable-sums*)
apply (*simp add*: *sums-def*)
apply (*cut-tac* $k = setsum\ f\ \{0..{<}n\}$ **in** *LIMSEQ-const*)
apply (*erule LIMSEQ-le*, *blast*)
apply (*rule-tac* $x{=}n$ **in** *exI*, *clarify*)
apply (*rule setsum-mono2*)
apply *auto*
done

lemma *series-pos-less*:
  fixes f :: $nat \Rightarrow real$
  shows $[\![summable\ f; \forall\, m{\geq}n.\ 0 < f\ m]\!] \Longrightarrow setsum\ f\ \{0..{<}n\} < suminf\ f$
apply (*rule-tac* $y{=}setsum\ f\ \{0..{<}Suc\ n\}$ **in** *order-less-le-trans*)
apply *simp*
apply (*erule series-pos-le*)
apply (*simp add*: *order-less-imp-le*)
done

lemma *suminf-gt-zero*:
  fixes f :: $nat \Rightarrow real$
  shows $[\![summable\ f; \forall\, n.\ 0 < f\ n]\!] \Longrightarrow 0 < suminf\ f$
by (*drule-tac* $n{=}0$ **in** *series-pos-less*, *simp-all*)

lemma *suminf-ge-zero*:
  fixes f :: $nat \Rightarrow real$
  shows $[\![summable\ f; \forall\, n.\ 0 \leq f\ n]\!] \Longrightarrow 0 \leq suminf\ f$
by (*drule-tac* $n{=}0$ **in** *series-pos-le*, *simp-all*)

lemma *sumr-pos-lt-pair*:
  fixes f :: $nat \Rightarrow real$
  shows $[\![summable\ f;$
      $\forall\, d.\ 0 < f\ (k + (Suc(Suc\ 0) * d)) + f\ (k + ((Suc(Suc\ 0) * d) + 1))]\!]$
    $\Longrightarrow setsum\ f\ \{0..{<}k\} < suminf\ f$
apply (*subst suminf-split-initial-segment* [**where** $k{=}k$])
apply *assumption*
apply *simp*
apply (*drule-tac* $k{=}k$ **in** *summable-ignore-initial-segment*)
apply (*drule-tac* $k{=}Suc\ (Suc\ 0)$ **in** *sums-group*, *simp*)
apply *simp*
apply (*frule sums-unique*)
apply (*drule sums-summable*)
apply *simp*
apply (*erule suminf-gt-zero*)
apply (*simp add*: *add-ac*)
done

Sum of a geometric progression.

lemmas *sumr-geometric* = *geometric-sum* [**where** *'a* = *real*]

lemma *geometric-sums*:
fixes *x* :: *'a*::{*real-normed-field*,*recpower*}
shows *norm x* < *1* ⟹ (λ*n*. *x* ^ *n*) *sums* (*1* / (*1* − *x*))
proof −
assume *less-1*: *norm x* < *1*
hence *neq-1*: *x* ≠ *1* **by** *auto*
hence *neq-0*: *x* − *1* ≠ *0* **by** *simp*
from *less-1* **have** *lim-0*: (λ*n*. *x* ^ *n*) −−−−> *0*
by (*rule LIMSEQ-power-zero*)
hence (λ*n*. *x* ^ *n* / (*x* − *1*) − *1* / (*x* − *1*)) −−−−> *0* / (*x* − *1*) − *1* / (*x* − *1*)
using *neq-0* **by** (*intro LIMSEQ-divide LIMSEQ-diff LIMSEQ-const*)
hence (λ*n*. (*x* ^ *n* − *1*) / (*x* − *1*)) −−−−> *1* / (*1* − *x*)
by (*simp add*: *nonzero-minus-divide-right* [*OF neq-0*] *diff-divide-distrib*)
thus (λ*n*. *x* ^ *n*) *sums* (*1* / (*1* − *x*))
by (*simp add*: *sums-def geometric-sum neq-1*)
qed

lemma *summable-geometric*:
fixes *x* :: *'a*::{*real-normed-field*,*recpower*}
shows *norm x* < *1* ⟹ *summable* (λ*n*. *x* ^ *n*)
by (*rule geometric-sums* [*THEN sums-summable*])

Cauchy-type criterion for convergence of series (c.f. Harrison)

lemma *summable-convergent-sumr-iff*:
summable f = *convergent* (%*n*. *setsum f* {*0*..<*n*})
by (*simp add*: *summable-def sums-def convergent-def*)

lemma *summable-LIMSEQ-zero*: *summable f* ⟹ *f* −−−−> *0*
apply (*drule summable-convergent-sumr-iff* [*THEN iffD1*])
apply (*drule convergent-Cauchy*)
apply (*simp only*: *Cauchy-def LIMSEQ-def*, *safe*)
apply (*drule-tac x*=*r* **in** *spec*, *safe*)
apply (*rule-tac x*=*M* **in** *exI*, *safe*)
apply (*drule-tac x*=*Suc n* **in** *spec*, *simp*)
apply (*drule-tac x*=*n* **in** *spec*, *simp*)
done

lemma *summable-Cauchy*:
summable (*f*::*nat* ⇒ *'a*::*banach*) =
(∀ *e* > *0*. ∃ *N*. ∀ *m* ≥ *N*. ∀ *n*. *norm* (*setsum f* {*m*..<*n*}) < *e*)
apply (*simp only*: *summable-convergent-sumr-iff Cauchy-convergent-iff* [*symmetric*] *Cauchy-def*, *safe*)
apply (*drule spec*, *drule* (*1*) *mp*)
apply (*erule exE*, *rule-tac x*=*M* **in** *exI*, *clarify*)
apply (*rule-tac x*=*m* **and** *y*=*n* **in** *linorder-le-cases*)

apply (*frule (1) order-trans*)
apply (*drule-tac x=n* **in** *spec*, *drule (1) mp*)
apply (*drule-tac x=m* **in** *spec*, *drule (1) mp*)
apply (*simp add*: *setsum-diff* [*symmetric*])
apply *simp*
apply (*drule spec*, *drule (1) mp*)
apply (*erule exE*, *rule-tac x=N* **in** *exI*, *clarify*)
apply (*rule-tac x=m* **and** *y=n* **in** *linorder-le-cases*)
apply (*subst norm-minus-commute*)
apply (*simp add*: *setsum-diff* [*symmetric*])
apply (*simp add*: *setsum-diff* [*symmetric*])
done

Comparison test

lemma *norm-setsum*:
  fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
  shows *norm* (*setsum f A*) $\leq$ ($\sum i{\in}A.$ *norm* (*f i*))
apply (*case-tac finite A*)
apply (*erule finite-induct*)
apply *simp*
apply *simp*
apply (*erule order-trans* [*OF norm-triangle-ineq add-left-mono*])
apply *simp*
done

lemma *summable-comparison-test*:
  fixes $f :: nat \Rightarrow 'a$::*banach*
  shows $[\![\exists N.\ \forall n{\geq}N.$ *norm* (*f n*) $\leq$ *g n*; *summable g*$]\!] \Longrightarrow$ *summable f*
apply (*simp add*: *summable-Cauchy*, *safe*)
apply (*drule-tac x=e* **in** *spec*, *safe*)
apply (*rule-tac x = N + Na* **in** *exI*, *safe*)
apply (*rotate-tac 2*)
apply (*drule-tac x = m* **in** *spec*)
apply (*auto*, *rotate-tac 2*, *drule-tac x = n* **in** *spec*)
apply (*rule-tac y =* $\sum k{=}m..{<}n.$ *norm* (*f k*) **in** *order-le-less-trans*)
apply (*rule norm-setsum*)
apply (*rule-tac y = setsum g* $\{m..{<}n\}$ **in** *order-le-less-trans*)
apply (*auto intro*: *setsum-mono simp add*: *abs-less-iff*)
done

lemma *summable-norm-comparison-test*:
  fixes $f :: nat \Rightarrow 'a$::*banach*
  shows $[\![\exists N.\ \forall n{\geq}N.$ *norm* (*f n*) $\leq$ *g n*; *summable g*$]\!]$
      $\Longrightarrow$ *summable* ($\lambda n.$ *norm* (*f n*))
apply (*rule summable-comparison-test*)
apply (*auto*)
done

lemma *summable-rabs-comparison-test*:

fixes $f :: nat \Rightarrow real$
shows $[\![\exists N.\ \forall n \geq N.\ |f\ n| \leq g\ n;\ summable\ g]\!] \Longrightarrow summable\ (\lambda n.\ |f\ n|)$
apply (*rule summable-comparison-test*)
apply (*auto*)
done

Summability of geometric series for real algebras

lemma *complete-algebra-summable-geometric*:
fixes $x :: 'a::\{real\text{-}normed\text{-}algebra\text{-}1, banach, recpower\}$
shows $norm\ x < 1 \Longrightarrow summable\ (\lambda n.\ x\ \hat{}\ n)$
proof (*rule summable-comparison-test*)
show $\exists N.\ \forall n \geq N.\ norm\ (x\ \hat{}\ n) \leq norm\ x\ \hat{}\ n$
by (*simp add*: *norm-power-ineq*)
show $norm\ x < 1 \Longrightarrow summable\ (\lambda n.\ norm\ x\ \hat{}\ n)$
by (*simp add*: *summable-geometric*)
qed

Limit comparison property for series (c.f. jrh)

lemma *summable-le*:
fixes $f\ g :: nat \Rightarrow real$
shows $[\![\forall n.\ f\ n \leq g\ n;\ summable\ f;\ summable\ g]\!] \Longrightarrow suminf\ f \leq suminf\ g$
apply (*drule summable-sums*)+
apply (*simp only*: *sums-def*, *erule* (*1*) *LIMSEQ-le*)
apply (*rule exI*)
apply (*auto intro*!: *setsum-mono*)
done

lemma *summable-le2*:
fixes $f\ g :: nat \Rightarrow real$
shows $[\![\forall n.\ |f\ n| \leq g\ n;\ summable\ g]\!] \Longrightarrow summable\ f \wedge suminf\ f \leq suminf\ g$
apply (*subgoal-tac summable f*)
apply (*auto intro*!: *summable-le*)
apply (*simp add*: *abs-le-iff*)
apply (*rule-tac g=g* **in** *summable-comparison-test*, *simp-all*)
done

lemma *suminf-0-le*:
fixes $f::nat \Rightarrow real$
assumes *gt0*: $\forall n.\ 0 \leq f\ n$ **and** *sm*: *summable f*
shows $0 \leq suminf\ f$
proof −
let *?g* = $(\lambda n.\ (0::real))$
from *gt0* **have** $\forall n.\ ?g\ n \leq f\ n$ **by** *simp*
moreover have *summable ?g* **by** (*rule summable-zero*)
moreover from *sm* **have** *summable f* **.**
ultimately have $suminf\ ?g \leq suminf\ f$ **by** (*rule summable-le*)
then show $0 \leq suminf\ f$ **by** (*simp add*: *suminf-zero*)
qed

Absolute convergence imples normal convergence

lemma *summable-norm-cancel*:
fixes $f :: nat \Rightarrow 'a::banach$
shows $summable\ (\lambda n.\ norm\ (f\ n)) \Longrightarrow summable\ f$
apply (*simp only*: *summable-Cauchy*, *safe*)
apply (*drule-tac* $x{=}e$ **in** *spec*, *safe*)
apply (*rule-tac* $x{=}N$ **in** *exI*, *safe*)
apply (*drule-tac* $x{=}m$ **in** *spec*, *safe*)
apply (*rule order-le-less-trans* [*OF norm-setsum*])
apply (*rule order-le-less-trans* [*OF abs-ge-self*])
apply *simp*
done

lemma *summable-rabs-cancel*:
fixes $f :: nat \Rightarrow real$
shows $summable\ (\lambda n.\ |f\ n|) \Longrightarrow summable\ f$
by (*rule summable-norm-cancel*, *simp*)

Absolute convergence of series

lemma *summable-norm*:
fixes $f :: nat \Rightarrow 'a::banach$
shows $summable\ (\lambda n.\ norm\ (f\ n)) \Longrightarrow norm\ (suminf\ f) \leq (\sum n.\ norm\ (f\ n))$
by (*auto intro*: *LIMSEQ-le LIMSEQ-norm summable-norm-cancel*
summable-sumr-LIMSEQ-suminf norm-setsum)

lemma *summable-rabs*:
fixes $f :: nat \Rightarrow real$
shows $summable\ (\lambda n.\ |f\ n|) \Longrightarrow |suminf\ f| \leq (\sum n.\ |f\ n|)$
by (*fold real-norm-def*, *rule summable-norm*)

18.2 The Ratio Test

lemma *norm-ratiotest-lemma*:
fixes $x\ y :: 'a::real\text{-}normed\text{-}vector$
shows $[\![c \leq 0;\ norm\ x \leq c * norm\ y]\!] \Longrightarrow x = 0$
apply (*subgoal-tac* $norm\ x \leq 0$, *simp*)
apply (*erule order-trans*)
apply (*simp add*: *mult-le-0-iff*)
done

lemma *rabs-ratiotest-lemma*: $[|\ c \leq 0;\ abs\ x \leq c * abs\ y\ |] ==> x = (0::real)$
by (*erule norm-ratiotest-lemma*, *simp*)

lemma *le-Suc-ex*: $(k::nat) \leq l ==> (\exists n.\ l = k + n)$
apply (*drule le-imp-less-or-eq*)
apply (*auto dest*: *less-imp-Suc-add*)
done

lemma *le-Suc-ex-iff*: $((k::nat) \leq l) = (\exists n.\ l = k + n)$

by (*auto simp add*: *le-Suc-ex*)

lemma *ratio-test-lemma2*:
fixes $f :: nat \Rightarrow 'a::banach$
shows $[\![\forall n \geq N.\ norm\ (f\ (Suc\ n)) \leq c * norm\ (f\ n)]\!] \Longrightarrow 0 < c \vee summable\ f$
apply (*simp* (*no-asm*) *add*: *linorder-not-le* [*symmetric*])
apply (*simp add*: *summable-Cauchy*)
apply (*safe*, *subgoal-tac* $\forall n.\ N < n \longrightarrow f\ (n) = 0$)
prefer *2*
apply *clarify*
apply(*erule-tac* $x = n - 1$ **in** *allE*)
apply (*simp add*:*diff-Suc split*:*nat.splits*)
apply (*blast intro*: *norm-ratiotest-lemma*)
apply (*rule-tac* $x = Suc\ N$ **in** *exI*, *clarify*)
apply(*simp cong*:*setsum-ivl-cong*)
done

lemma *ratio-test*:
fixes $f :: nat \Rightarrow 'a::banach$
shows $[\![c < 1; \forall n \geq N.\ norm\ (f\ (Suc\ n)) \leq c * norm\ (f\ n)]\!] \Longrightarrow summable\ f$
apply (*frule ratio-test-lemma2*, *auto*)
apply (*rule-tac* $g = \%n.\ (norm\ (f\ N) / (c\ \hat{}\ N)) * c\ \hat{}\ n$
in *summable-comparison-test*)
apply (*rule-tac* $x = N$ **in** *exI*, *safe*)
apply (*drule le-Suc-ex-iff* [*THEN iffD1*])
apply (*auto simp add*: *power-add field-power-not-zero*)
apply (*induct-tac na*, *auto*)
apply (*rule-tac* $y = c * norm\ (f\ (N + n))$ **in** *order-trans*)
apply (*auto intro*: *mult-right-mono simp add*: *summable-def*)
apply (*simp add*: *mult-ac*)
apply (*rule-tac* $x = norm\ (f\ N) * (1 / (1 - c)) / (c\ \hat{}\ N)$ **in** *exI*)
apply (*rule sums-divide*)
apply (*rule sums-mult*)
apply (*auto intro*!: *geometric-sums*)
done

18.3 Cauchy Product Formula

lemma *setsum-triangle-reindex*:
fixes $n :: nat$
shows $(\sum (i,j) \in \{(i,j).\ i+j < n\}.\ f\ i\ j) = (\sum k=0..<n.\ \sum i=0..k.\ f\ i\ (k - i))$
proof −
have $(\sum (i, j) \in \{(i, j).\ i + j < n\}.\ f\ i\ j) =$
$(\sum (k, i) \in (SIGMA\ k:\{0..<n\}.\ \{0..k\}).\ f\ i\ (k - i))$
proof (*rule setsum-reindex-cong*)
show *inj-on* $(\lambda(k,i).\ (i, k - i))$ (*SIGMA* $k:\{0..<n\}.\ \{0..k\}$)
by (*rule inj-on-inverseI* [**where** $g=\lambda(i,j).\ (i+j, i)$], *auto*)
show $\{(i,j).\ i + j < n\} = (\lambda(k,i).\ (i, k - i))$ ' (*SIGMA* $k:\{0..<n\}.\ \{0..k\}$)

```
    by (safe, rule-tac x=(a+b,a) in image-eqI, auto)
  show ⋀a. (λ(k, i). f i (k − i)) a = split f ((λ(k, i). (i, k − i)) a)
    by clarify
 qed
 thus ?thesis by (simp add: setsum-Sigma)
qed

lemma Cauchy-product-sums:
 fixes a b :: nat ⇒ 'a::{real-normed-algebra,banach}
 assumes a: summable (λk. norm (a k))
 assumes b: summable (λk. norm (b k))
 shows (λk. ∑ i=0..k. a i * b (k − i)) sums ((∑ k. a k) * (∑ k. b k))
proof −
 let ?S1 = λn::nat. {0..<n} × {0..<n}
 let ?S2 = λn::nat. {(i,j). i + j < n}
 have S1-mono: ⋀m n. m ≤ n ⟹ ?S1 m ⊆ ?S1 n by auto
 have S2-le-S1: ⋀n. ?S2 n ⊆ ?S1 n by auto
 have S1-le-S2: ⋀n. ?S1 (n div 2) ⊆ ?S2 n by auto
 have finite-S1: ⋀n. finite (?S1 n) by simp
 with S2-le-S1 have finite-S2: ⋀n. finite (?S2 n) by (rule finite-subset)

 let ?g = λ(i,j). a i * b j
 let ?f = λ(i,j). norm (a i) * norm (b j)
 have f-nonneg: ⋀x. 0 ≤ ?f x
  by (auto simp add: mult-nonneg-nonneg)
 hence norm-setsum-f: ⋀A. norm (setsum ?f A) = setsum ?f A
  unfolding real-norm-def
  by (simp only: abs-of-nonneg setsum-nonneg [rule-format])

 have (λn. (∑ k=0..<n. a k) * (∑ k=0..<n. b k))
       ----> (∑ k. a k) * (∑ k. b k)
  by (intro LIMSEQ-mult summable-sumr-LIMSEQ-suminf
      summable-norm-cancel [OF a] summable-norm-cancel [OF b])
 hence 1: (λn. setsum ?g (?S1 n)) ----> (∑ k. a k) * (∑ k. b k)
  by (simp only: setsum-product setsum-Sigma [rule-format]
                 finite-atLeastLessThan)

 have (λn. (∑ k=0..<n. norm (a k)) * (∑ k=0..<n. norm (b k)))
      ----> (∑ k. norm (a k)) * (∑ k. norm (b k))
  using a b by (intro LIMSEQ-mult summable-sumr-LIMSEQ-suminf)
 hence (λn. setsum ?f (?S1 n)) ----> (∑ k. norm (a k)) * (∑ k. norm (b k))
  by (simp only: setsum-product setsum-Sigma [rule-format]
                 finite-atLeastLessThan)
 hence convergent (λn. setsum ?f (?S1 n))
  by (rule convergentI)
 hence Cauchy: Cauchy (λn. setsum ?f (?S1 n))
  by (rule convergent-Cauchy)
 have Zseq (λn. setsum ?f (?S1 n − ?S2 n))
 proof (rule ZseqI, simp only: norm-setsum-f)
```

```
    fix r :: real
    assume r: 0 < r
    from CauchyD [OF Cauchy r] obtain N
    where ∀m≥N. ∀n≥N. norm (setsum ?f (?S1 m) − setsum ?f (?S1 n)) < r
..
    hence ⋀m n. ⟦N ≤ n; n ≤ m⟧ ⟹ norm (setsum ?f (?S1 m − ?S1 n)) < r
      by (simp only: setsum-diff finite-S1 S1-mono)
    hence N: ⋀m n. ⟦N ≤ n; n ≤ m⟧ ⟹ setsum ?f (?S1 m − ?S1 n) < r
      by (simp only: norm-setsum-f)
    show ∃N. ∀n≥N. setsum ?f (?S1 n − ?S2 n) < r
    proof (intro exI allI impI)
      fix n assume 2 * N ≤ n
      hence n: N ≤ n div 2 by simp
      have setsum ?f (?S1 n − ?S2 n) ≤ setsum ?f (?S1 n − ?S1 (n div 2))
        by (intro setsum-mono2 finite-Diff finite-S1 f-nonneg
                  Diff-mono subset-refl S1-le-S2)
      also have ... < r
        using n div-le-dividend by (rule N)
      finally show setsum ?f (?S1 n − ?S2 n) < r .
    qed
  qed
  hence Zseq (λn. setsum ?g (?S1 n − ?S2 n))
    apply (rule Zseq-le [rule-format])
    apply (simp only: norm-setsum-f)
    apply (rule order-trans [OF norm-setsum setsum-mono])
    apply (auto simp add: norm-mult-ineq)
    done
  hence 2: (λn. setsum ?g (?S1 n) − setsum ?g (?S2 n)) ----> 0
    by (simp only: LIMSEQ-Zseq-iff setsum-diff finite-S1 S2-le-S1 diff-0-right)

  with 1 have (λn. setsum ?g (?S2 n)) ----> (∑k. a k) * (∑k. b k)
    by (rule LIMSEQ-diff-approach-zero2)
  thus ?thesis by (simp only: sums-def setsum-triangle-reindex)
qed

lemma Cauchy-product:
  fixes a b :: nat ⇒ 'a::{real-normed-algebra,banach}
  assumes a: summable (λk. norm (a k))
  assumes b: summable (λk. norm (b k))
  shows (∑k. a k) * (∑k. b k) = (∑k. ∑i=0..k. a i * b (k − i))
using a b
by (rule Cauchy-product-sums [THEN sums-unique])

end
```

19 Lim: Limits and Continuity

theory *Lim*

imports *SEQ*
begin

Standard Definitions

definition
LIM :: [*'a*::*real-normed-vector* => *'b*::*real-normed-vector*, *'a*, *'b*] => *bool*
(((-)/ –– (-)/ –––> (-)) [*60*, *0*, *60*] *60*) **where**
f –– *a* –––> *L* =
(∀ *r* > *0*. ∃ *s* > *0*. ∀ *x*. *x* ≠ *a* & *norm* (*x* − *a*) < *s*
–––> *norm* (*f x* − *L*) < *r*)

definition
isCont :: [*'a*::*real-normed-vector* => *'b*::*real-normed-vector*, *'a*] => *bool* **where**
isCont f a = (*f* –– *a* –––> (*f a*))

definition
isUCont :: [*'a*::*real-normed-vector* => *'b*::*real-normed-vector*] => *bool* **where**
isUCont f = (∀ *r*>*0*. ∃ *s*>*0*. ∀ *x y*. *norm* (*x* − *y*) < *s* ⟶ *norm* (*f x* − *f y*) < *r*)

19.1 Limits of Functions

19.1.1 Purely standard proofs

lemma *LIM-eq*:
f –– *a* –––> *L* =
(∀ *r*>*0*.∃ *s*>*0*.∀ *x*. *x* ≠ *a* & *norm* (*x*−*a*) < *s* –––> *norm* (*f x* − *L*) < *r*)
by (*simp add*: *LIM-def diff-minus*)

lemma *LIM-I*:
(!!*r*. *0*<*r* ==> ∃ *s*>*0*.∀ *x*. *x* ≠ *a* & *norm* (*x*−*a*) < *s* –––> *norm* (*f x* − *L*) < *r*)
==> *f* –– *a* –––> *L*
by (*simp add*: *LIM-eq*)

lemma *LIM-D*:
[| *f* –– *a* –––> *L*; *0*<*r* |]
==> ∃ *s*>*0*.∀ *x*. *x* ≠ *a* & *norm* (*x*−*a*) < *s* –––> *norm* (*f x* − *L*) < *r*
by (*simp add*: *LIM-eq*)

lemma *LIM-offset*: *f* –– *a* –––> *L* ⟹ (λ*x*. *f* (*x* + *k*)) –– *a* − *k* –––> *L*
apply (*rule LIM-I*)
apply (*drule-tac r*=*r* **in** *LIM-D*, *safe*)
apply (*rule-tac x*=*s* **in** *exI*, *safe*)
apply (*drule-tac x*=*x* + *k* **in** *spec*)
apply (*simp add*: *compare-rls*)
done

lemma *LIM-offset-zero*: *f* –– *a* –––> *L* ⟹ (λ*h*. *f* (*a* + *h*)) –– *0* –––> *L*
by (*drule-tac k*=*a* **in** *LIM-offset*, *simp add*: *add-commute*)

lemma *LIM-offset-zero-cancel*: $(\lambda h.\ f\ (a + h)) -- 0 --> L \Longrightarrow f -- a --> L$
by (*drule-tac k=− a* **in** *LIM-offset*, *simp*)

lemma *LIM-const* [*simp*]: $(\%x.\ k) -- x --> k$
by (*simp add*: *LIM-def*)

lemma *LIM-add*:
fixes *f g* :: *'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*
assumes *f*: $f -- a --> L$ **and** *g*: $g -- a --> M$
shows $(\%x.\ f\ x + g(x)) -- a --> (L + M)$
proof (*rule LIM-I*)
fix *r* :: *real*
assume *r*: $0 < r$
from *LIM-D* [*OF f half-gt-zero* [*OF r*]]
obtain *fs*
where *fs*: $0 < fs$
and *fs-lt*: $\forall x.\ x \neq a\ \&\ norm\ (x-a) < fs --> norm\ (f\ x - L) < r/2$
by *blast*
from *LIM-D* [*OF g half-gt-zero* [*OF r*]]
obtain *gs*
where *gs*: $0 < gs$
and *gs-lt*: $\forall x.\ x \neq a\ \&\ norm\ (x-a) < gs --> norm\ (g\ x - M) < r/2$
by *blast*
show $\exists s>0.\forall x.\ x \neq a \wedge norm\ (x-a) < s \longrightarrow norm\ (f\ x + g\ x - (L + M)) < r$
proof (*intro exI conjI strip*)
show $0 < min\ fs\ gs$ **by** (*simp add*: *fs gs*)
fix *x* :: *'a*
assume $x \neq a \wedge norm\ (x-a) < min\ fs\ gs$
hence $x \neq a \wedge norm\ (x-a) < fs \wedge norm\ (x-a) < gs$ **by** *simp*
with *fs-lt gs-lt*
have $norm\ (f\ x - L) < r/2$ **and** $norm\ (g\ x - M) < r/2$ **by** *blast+*
hence $norm\ (f\ x - L) + norm\ (g\ x - M) < r$ **by** *arith*
thus $norm\ (f\ x + g\ x - (L + M)) < r$
by (*blast intro*: *norm-diff-triangle-ineq order-le-less-trans*)
qed
qed

lemma *LIM-add-zero*:
$[\![f -- a --> 0;\ g -- a --> 0]\!] \Longrightarrow (\lambda x.\ f\ x + g\ x) -- a --> 0$
by (*drule* (*1*) *LIM-add*, *simp*)

lemma *minus-diff-minus*:
fixes *a b* :: *'a*::*ab-group-add*
shows $(-\ a) - (-\ b) = -\ (a - b)$
by *simp*

lemma *LIM-minus*: $f -- a --> L ==> (\%x.\ -f(x)) -- a --> -L$

by (*simp only*: *LIM-eq minus-diff-minus norm-minus-cancel*)

lemma *LIM-add-minus*:
[| f –– x –––> l; g –– x –––> m |] ==> (%x. $f(x) + -g(x)$) –– x –––> ($l + -m$)
by (*intro LIM-add LIM-minus*)

lemma *LIM-diff*:
[| f –– x –––> l; g –– x –––> m |] ==> (%x. $f(x) - g(x)$) –– x –––> $l-m$
by (*simp only*: *diff-minus LIM-add LIM-minus*)

lemma *LIM-zero*: f –– a –––> l ⟹ (λx. $f\ x - l$) –– a –––> 0
by (*simp add*: *LIM-def*)

lemma *LIM-zero-cancel*: (λx. $f\ x - l$) –– a –––> 0 ⟹ f –– a –––> l
by (*simp add*: *LIM-def*)

lemma *LIM-zero-iff*: (λx. $f\ x - l$) –– a –––> 0 = f –– a –––> l
by (*simp add*: *LIM-def*)

lemma *LIM-imp-LIM*:
assumes *f*: f –– a –––> l
assumes *le*: $\bigwedge x.\ x \neq a \Longrightarrow norm\ (g\ x - m) \leq norm\ (f\ x - l)$
shows g –– a –––> m
apply (*rule LIM-I*, *drule LIM-D* [*OF f*], *safe*)
apply (*rule-tac x*=*s* **in** *exI*, *safe*)
apply (*drule-tac x*=*x* **in** *spec*, *safe*)
apply (*erule* (*1*) *order-le-less-trans* [*OF le*])
done

lemma *LIM-norm*: f –– a –––> l ⟹ (λx. $norm\ (f\ x)$) –– a –––> $norm\ l$
by (*erule LIM-imp-LIM*, *simp add*: *norm-triangle-ineq3*)

lemma *LIM-norm-zero*: f –– a –––> 0 ⟹ (λx. $norm\ (f\ x)$) –– a –––> 0
by (*drule LIM-norm*, *simp*)

lemma *LIM-norm-zero-cancel*: (λx. $norm\ (f\ x)$) –– a –––> 0 ⟹ f –– a –––> 0
by (*erule LIM-imp-LIM*, *simp*)

lemma *LIM-norm-zero-iff*: (λx. $norm\ (f\ x)$) –– a –––> 0 = f –– a –––> 0
by (*rule iffI* [*OF LIM-norm-zero-cancel LIM-norm-zero*])

lemma *LIM-rabs*: f –– a –––> (l::*real*) ⟹ (λx. $|f\ x|$) –– a –––> $|l|$
by (*fold real-norm-def*, *rule LIM-norm*)

lemma *LIM-rabs-zero*: f –– a –––> (*0*::*real*) ⟹ (λx. $|f\ x|$) –– a –––> 0
by (*fold real-norm-def*, *rule LIM-norm-zero*)

lemma *LIM-rabs-zero-cancel*: $(\lambda x.\ |f\ x|) \dashrightarrow a \longrightarrow (0::real) \Longrightarrow f \dashrightarrow a \longrightarrow 0$
by (*fold real-norm-def*, *rule LIM-norm-zero-cancel*)

lemma *LIM-rabs-zero-iff*: $(\lambda x.\ |f\ x|) \dashrightarrow a \longrightarrow (0::real) = f \dashrightarrow a \longrightarrow 0$
by (*fold real-norm-def*, *rule LIM-norm-zero-iff*)

lemma *LIM-const-not-eq*:
fixes $a :: 'a::real\text{-}normed\text{-}algebra\text{-}1$
shows $k \neq L \Longrightarrow \neg\ (\lambda x.\ k) \dashrightarrow a \longrightarrow L$
apply (*simp add*: *LIM-eq*)
apply (*rule-tac x=norm* $(k - L)$ **in** *exI*, *simp*, *safe*)
apply (*rule-tac x=a + of-real* $(s/2)$ **in** *exI*, *simp add*: *norm-of-real*)
done

lemmas *LIM-not-zero* = *LIM-const-not-eq* [**where** $L = 0$]

lemma *LIM-const-eq*:
fixes $a :: 'a::real\text{-}normed\text{-}algebra\text{-}1$
shows $(\lambda x.\ k) \dashrightarrow a \longrightarrow L \Longrightarrow k = L$
apply (*rule ccontr*)
apply (*blast dest*: *LIM-const-not-eq*)
done

lemma *LIM-unique*:
fixes $a :: 'a::real\text{-}normed\text{-}algebra\text{-}1$
shows $[\![f \dashrightarrow a \longrightarrow L;\ f \dashrightarrow a \longrightarrow M]\!] \Longrightarrow L = M$
apply (*drule* (*1*) *LIM-diff*)
apply (*auto dest*!: *LIM-const-eq*)
done

lemma *LIM-ident* [*simp*]: $(\lambda x.\ x) \dashrightarrow a \longrightarrow a$
by (*auto simp add*: *LIM-def*)

Limits are equal for functions equal except at limit point

lemma *LIM-equal*:
$[|\ \forall x.\ x \neq a \longrightarrow (f\ x = g\ x)\ |] ==> (f \dashrightarrow a \longrightarrow l) = (g \dashrightarrow a \longrightarrow l)$
by (*simp add*: *LIM-def*)

lemma *LIM-cong*:
$[\![a = b;\ \bigwedge x.\ x \neq b \Longrightarrow f\ x = g\ x;\ l = m]\!]$
$\Longrightarrow ((\lambda x.\ f\ x) \dashrightarrow a \longrightarrow l) = ((\lambda x.\ g\ x) \dashrightarrow b \longrightarrow m)$
by (*simp add*: *LIM-def*)

lemma *LIM-equal2*:
assumes *1*: $0 < R$
assumes *2*: $\bigwedge x.\ [\![x \neq a;\ norm\ (x - a) < R]\!] \Longrightarrow f\ x = g\ x$
shows $g \dashrightarrow a \longrightarrow l \Longrightarrow f \dashrightarrow a \longrightarrow l$
apply (*unfold LIM-def*, *safe*)

apply (*drule-tac x=r* **in** *spec*, *safe*)
apply (*rule-tac x=min s R* **in** *exI*, *safe*)
apply (*simp add*: *1*)
apply (*simp add*: *2*)
done

Two uses in Hyperreal/Transcendental.ML

lemma *LIM-trans*:
[| (%*x*. *f*(*x*) + −*g*(*x*)) −− *a* −−> *0*; *g* −− *a* −−> *l* |] ==> *f* −− *a* −−> *l*
apply (*drule LIM-add*, *assumption*)
apply (*auto simp add*: *add-assoc*)
done

lemma *LIM-compose*:
assumes *g*: *g* −− *l* −−> *g l*
assumes *f*: *f* −− *a* −−> *l*
shows (λ*x*. *g* (*f x*)) −− *a* −−> *g l*
proof (*rule LIM-I*)
fix *r*::*real* **assume** *r*: *0* < *r*
obtain *s* **where** *s*: *0* < *s*
and *less-r*: ⋀*y*. ⟦*y* ≠ *l*; *norm* (*y* − *l*) < *s*⟧ ⟹ *norm* (*g y* − *g l*) < *r*
using *LIM-D* [*OF g r*] **by** *fast*
obtain *t* **where** *t*: *0* < *t*
and *less-s*: ⋀*x*. ⟦*x* ≠ *a*; *norm* (*x* − *a*) < *t*⟧ ⟹ *norm* (*f x* − *l*) < *s*
using *LIM-D* [*OF f s*] **by** *fast*

show ∃ *t*>*0*. ∀ *x*. *x* ≠ *a* ∧ *norm* (*x* − *a*) < *t* ⟶ *norm* (*g* (*f x*) − *g l*) < *r*
proof (*rule exI*, *safe*)
show *0* < *t* **using** *t* **.**
next
fix *x* **assume** *x* ≠ *a* **and** *norm* (*x* − *a*) < *t*
hence *norm* (*f x* − *l*) < *s* **by** (*rule less-s*)
thus *norm* (*g* (*f x*) − *g l*) < *r*
using *r less-r* **by** (*case-tac f x* = *l*, *simp-all*)
qed
qed

lemma *LIM-compose2*:
assumes *f*: *f* −− *a* −−> *b*
assumes *g*: *g* −− *b* −−> *c*
assumes *inj*: ∃ *d*>*0*. ∀ *x*. *x* ≠ *a* ∧ *norm* (*x* − *a*) < *d* ⟶ *f x* ≠ *b*
shows (λ*x*. *g* (*f x*)) −− *a* −−> *c*
proof (*rule LIM-I*)
fix *r* :: *real*
assume *r*: *0* < *r*
obtain *s* **where** *s*: *0* < *s*
and *less-r*: ⋀*y*. ⟦*y* ≠ *b*; *norm* (*y* − *b*) < *s*⟧ ⟹ *norm* (*g y* − *c*) < *r*
using *LIM-D* [*OF g r*] **by** *fast*
obtain *t* **where** *t*: *0* < *t*

```
    and less-s: ⋀x. ⟦x ≠ a; norm (x − a) < t⟧ ⟹ norm (f x − b) < s
    using LIM-D [OF f s] by fast
  obtain d where d: 0 < d
    and neq-b: ⋀x. ⟦x ≠ a; norm (x − a) < d⟧ ⟹ f x ≠ b
    using inj by fast

  show ∃t>0. ∀x. x ≠ a ∧ norm (x − a) < t ⟶ norm (g (f x) − c) < r
  proof (safe intro!: exI)
    show 0 < min d t using d t by simp
  next
    fix x
    assume x ≠ a and norm (x − a) < min d t
    hence f x ≠ b and norm (f x − b) < s
      using neq-b less-s by simp-all
    thus norm (g (f x) − c) < r
      by (rule less-r)
  qed
qed

lemma LIM-o: ⟦g −− l −−> g l; f −− a −−> l⟧ ⟹ (g ∘ f) −− a −−> g l
unfolding o-def by (rule LIM-compose)

lemma real-LIM-sandwich-zero:
  fixes f g :: 'a::real-normed-vector ⇒ real
  assumes f: f −− a −−> 0
  assumes 1: ⋀x. x ≠ a ⟹ 0 ≤ g x
  assumes 2: ⋀x. x ≠ a ⟹ g x ≤ f x
  shows g −− a −−> 0
proof (rule LIM-imp-LIM [OF f])
  fix x assume x: x ≠ a
  have norm (g x − 0) = g x by (simp add: 1 x)
  also have g x ≤ f x by (rule 2 [OF x])
  also have f x ≤ |f x| by (rule abs-ge-self)
  also have |f x| = norm (f x − 0) by simp
  finally show norm (g x − 0) ≤ norm (f x − 0) .
qed
```

Bounded Linear Operators

```
lemma (in bounded-linear) cont: f −− a −−> f a
proof (rule LIM-I)
  fix r::real assume r: 0 < r
  obtain K where K: 0 < K and norm-le: ⋀x. norm (f x) ≤ norm x * K
    using pos-bounded by fast
  show ∃s>0. ∀x. x ≠ a ∧ norm (x − a) < s ⟶ norm (f x − f a) < r
  proof (rule exI, safe)
    from r K show 0 < r / K by (rule divide-pos-pos)
  next
    fix x assume x: norm (x − a) < r / K
    have norm (f x − f a) = norm (f (x − a)) by (simp only: diff)
```

```
    also have ... ≤ norm (x − a) * K by (rule norm-le)
    also from K x have ... < r by (simp only: pos-less-divide-eq)
    finally show norm (f x − f a) < r .
  qed
qed

lemma (in bounded-linear) LIM:
  g −− a −−> l ⟹ (λx. f (g x)) −− a −−> f l
by (rule LIM-compose [OF cont])

lemma (in bounded-linear) LIM-zero:
  g −− a −−> 0 ⟹ (λx. f (g x)) −− a −−> 0
by (drule LIM, simp only: zero)
```

Bounded Bilinear Operators

```
lemma (in bounded-bilinear) LIM-prod-zero:
  assumes f: f −− a −−> 0
  assumes g: g −− a −−> 0
  shows (λx. f x ** g x) −− a −−> 0
proof (rule LIM-I)
  fix r::real assume r: 0 < r
  obtain K where K: 0 < K
    and norm-le: ⋀x y. norm (x ** y) ≤ norm x * norm y * K
    using pos-bounded by fast
  from K have K': 0 < inverse K
    by (rule positive-imp-inverse-positive)
  obtain s where s: 0 < s
    and norm-f: ⋀x. ⟦x ≠ a; norm (x − a) < s⟧ ⟹ norm (f x) < r
    using LIM-D [OF f r] by auto
  obtain t where t: 0 < t
    and norm-g: ⋀x. ⟦x ≠ a; norm (x − a) < t⟧ ⟹ norm (g x) < inverse K
    using LIM-D [OF g K'] by auto
  show ∃s>0. ∀x. x ≠ a ∧ norm (x − a) < s ⟶ norm (f x ** g x − 0) < r
  proof (rule exI, safe)
    from s t show 0 < min s t by simp
  next
    fix x assume x: x ≠ a
    assume norm (x − a) < min s t
    hence xs: norm (x − a) < s and xt: norm (x − a) < t by simp-all
    from x xs have 1: norm (f x) < r by (rule norm-f)
    from x xt have 2: norm (g x) < inverse K by (rule norm-g)
    have norm (f x ** g x) ≤ norm (f x) * norm (g x) * K by (rule norm-le)
    also from 1 2 K have ... < r * inverse K * K
      by (intro mult-strict-right-mono mult-strict-mono' norm-ge-zero)
    also from K have r * inverse K * K = r by simp
    finally show norm (f x ** g x − 0) < r by simp
  qed
qed
```

lemma (**in** *bounded-bilinear*) *LIM-left-zero*:
  f –– a ––> $0 \Longrightarrow (\lambda x.\ f\ x$ ** $c)$ –– a ––> 0
by (*rule bounded-linear.LIM-zero* [*OF bounded-linear-left*])

lemma (**in** *bounded-bilinear*) *LIM-right-zero*:
  f –– a ––> $0 \Longrightarrow (\lambda x.\ c$ ** $f\ x)$ –– a ––> 0
by (*rule bounded-linear.LIM-zero* [*OF bounded-linear-right*])

lemma (**in** *bounded-bilinear*) *LIM*:
  $[\![f$ –– a ––> $L;\ g$ –– a ––> $M]\!] \Longrightarrow (\lambda x.\ f\ x$ ** $g\ x)$ –– a ––> L ** M
apply (*drule LIM-zero*)
apply (*drule LIM-zero*)
apply (*rule LIM-zero-cancel*)
apply (*subst prod-diff-prod*)
apply (*rule LIM-add-zero*)
apply (*rule LIM-add-zero*)
apply (*erule* (*1*) *LIM-prod-zero*)
apply (*erule LIM-left-zero*)
apply (*erule LIM-right-zero*)
done

lemmas *LIM-mult* = *mult.LIM*

lemmas *LIM-mult-zero* = *mult.LIM-prod-zero*

lemmas *LIM-mult-left-zero* = *mult.LIM-left-zero*

lemmas *LIM-mult-right-zero* = *mult.LIM-right-zero*

lemmas *LIM-scaleR* = *scaleR.LIM*

lemmas *LIM-of-real* = *of-real.LIM*

lemma *LIM-power*:
  fixes f :: $'a$::*real-normed-vector* ⇒ $'b$::{*recpower*,*real-normed-algebra*}
  assumes f: f –– a ––> l
  shows $(\lambda x.\ f\ x\ \hat{}\ n)$ –– a ––> $l\ \hat{}\ n$
by (*induct n*, *simp*, *simp add*: *power-Suc LIM-mult f*)

19.1.2 Derived theorems about *LIM*

lemma *LIM-inverse-lemma*:
  fixes x :: $'a$::*real-normed-div-algebra*
  assumes r: $0 < r$
  assumes x: $norm\ (x - 1) < min\ (1/2)\ (r/2)$
  shows $norm\ (inverse\ x - 1) < r$
proof −
  from r **have** *r2*: $0 < r/2$ **by** *simp*
  from x **have** *0*: $x \neq 0$ **by** *clarsimp*

from *x* **have** *x'*: *norm (1 − x) < min (1/2) (r/2)*
by (*simp only*: *norm-minus-commute*)
hence *less1*: *norm (1 − x) < r/2* **by** *simp*
have *norm (1::'a) − norm x ≤ norm (1 − x)* **by** (*rule norm-triangle-ineq2*)
also from *x'* **have** *norm (1 − x) < 1/2* **by** *simp*
finally have *1/2 < norm x* **by** *simp*
hence *inverse (norm x) < inverse (1/2)*
by (*rule less-imp-inverse-less*, *simp*)
hence *less2*: *norm (inverse x) < 2*
by (*simp add*: *nonzero-norm-inverse 0*)
from *less1 less2 r2 norm-ge-zero*
have *norm (1 − x) * norm (inverse x) < (r/2) * 2*
by (*rule mult-strict-mono*)
thus *norm (inverse x − 1) < r*
by (*simp only*: *norm-mult* [*symmetric*] *left-diff-distrib*, *simp add*: *0*)
qed

lemma *LIM-inverse-fun*:
assumes *a*: *a ≠ (0::'a::real-normed-div-algebra)*
shows *inverse −− a −−> inverse a*
proof (*rule LIM-equal2*)
from *a* **show** *0 < norm a* **by** *simp*
next
fix *x* **assume** *norm (x − a) < norm a*
hence *x ≠ 0* **by** *auto*
with *a* **show** *inverse x = inverse (inverse a * x) * inverse a*
by (*simp add*: *nonzero-inverse-mult-distrib*
nonzero-imp-inverse-nonzero
nonzero-inverse-inverse-eq mult-assoc)
next
have *1*: *inverse −− 1 −−> inverse (1::'a)*
apply (*rule LIM-I*)
apply (*rule-tac x=min (1/2) (r/2)* **in** *exI*)
apply (*simp add*: *LIM-inverse-lemma*)
done
have *(λx. inverse a * x) −− a −−> inverse a * a*
by (*intro LIM-mult LIM-ident LIM-const*)
hence *(λx. inverse a * x) −− a −−> 1*
by (*simp add*: *a*)
with *1* **have** *(λx. inverse (inverse a * x)) −− a −−> inverse 1*
by (*rule LIM-compose*)
hence *(λx. inverse (inverse a * x)) −− a −−> 1*
by *simp*
hence *(λx. inverse (inverse a * x) * inverse a) −− a −−> 1 * inverse a*
by (*intro LIM-mult LIM-const*)
thus *(λx. inverse (inverse a * x) * inverse a) −− a −−> inverse a*
by *simp*
qed

lemma *LIM-inverse*:
fixes $L :: 'a$::*real-normed-div-algebra*
shows $[\![f \dashrightarrow a \longrightarrow L;\ L \neq 0]\!] \Longrightarrow (\lambda x.\ inverse\ (f\ x)) \dashrightarrow a \longrightarrow inverse\ L$
by (*rule LIM-inverse-fun* [*THEN LIM-compose*])

19.2 Continuity

19.2.1 Purely standard proofs

lemma *LIM-isCont-iff*: $(f \dashrightarrow a \longrightarrow f\ a) = ((\lambda h.\ f\ (a + h)) \dashrightarrow 0 \longrightarrow f\ a)$
by (*rule iffI* [*OF LIM-offset-zero LIM-offset-zero-cancel*])

lemma *isCont-iff*: $isCont\ f\ x = (\lambda h.\ f\ (x + h)) \dashrightarrow 0 \longrightarrow f\ x$
by (*simp add*: *isCont-def LIM-isCont-iff*)

lemma *isCont-ident* [*simp*]: $isCont\ (\lambda x.\ x)\ a$
unfolding *isCont-def* **by** (*rule LIM-ident*)

lemma *isCont-const* [*simp*]: $isCont\ (\lambda x.\ k)\ a$
unfolding *isCont-def* **by** (*rule LIM-const*)

lemma *isCont-norm*: $isCont\ f\ a \Longrightarrow isCont\ (\lambda x.\ norm\ (f\ x))\ a$
unfolding *isCont-def* **by** (*rule LIM-norm*)

lemma *isCont-rabs*: $isCont\ f\ a \Longrightarrow isCont\ (\lambda x.\ |f\ x :: real|)\ a$
unfolding *isCont-def* **by** (*rule LIM-rabs*)

lemma *isCont-add*: $[\![isCont\ f\ a;\ isCont\ g\ a]\!] \Longrightarrow isCont\ (\lambda x.\ f\ x + g\ x)\ a$
unfolding *isCont-def* **by** (*rule LIM-add*)

lemma *isCont-minus*: $isCont\ f\ a \Longrightarrow isCont\ (\lambda x.\ -\ f\ x)\ a$
unfolding *isCont-def* **by** (*rule LIM-minus*)

lemma *isCont-diff*: $[\![isCont\ f\ a;\ isCont\ g\ a]\!] \Longrightarrow isCont\ (\lambda x.\ f\ x - g\ x)\ a$
unfolding *isCont-def* **by** (*rule LIM-diff*)

lemma *isCont-mult*:
fixes $f\ g :: 'a$::*real-normed-vector* $\Rightarrow 'b$::*real-normed-algebra*
shows $[\![isCont\ f\ a;\ isCont\ g\ a]\!] \Longrightarrow isCont\ (\lambda x.\ f\ x * g\ x)\ a$
unfolding *isCont-def* **by** (*rule LIM-mult*)

lemma *isCont-inverse*:
fixes $f :: 'a$::*real-normed-vector* $\Rightarrow 'b$::*real-normed-div-algebra*
shows $[\![isCont\ f\ a;\ f\ a \neq 0]\!] \Longrightarrow isCont\ (\lambda x.\ inverse\ (f\ x))\ a$
unfolding *isCont-def* **by** (*rule LIM-inverse*)

lemma *isCont-LIM-compose*:
$[\![isCont\ g\ l;\ f \dashrightarrow a \longrightarrow l]\!] \Longrightarrow (\lambda x.\ g\ (f\ x)) \dashrightarrow a \longrightarrow g\ l$
unfolding *isCont-def* **by** (*rule LIM-compose*)

lemma *isCont-LIM-compose2*:
assumes *f* [*unfolded isCont-def*]: *isCont f a*
assumes *g*: *g –– f a ––> l*
assumes *inj*: ∃ *d*>*0*. ∀ *x*. *x* ≠ *a* ∧ *norm* (*x* − *a*) < *d* ⟶ *f x* ≠ *f a*
shows (λ*x*. *g* (*f x*)) –– *a* ––> *l*
by (*rule LIM-compose2* [*OF f g inj*])

lemma *isCont-o2*: ⟦*isCont f a*; *isCont g* (*f a*)⟧ ⟹ *isCont* (λ*x*. *g* (*f x*)) *a*
unfolding *isCont-def* **by** (*rule LIM-compose*)

lemma *isCont-o*: ⟦*isCont f a*; *isCont g* (*f a*)⟧ ⟹ *isCont* (*g o f*) *a*
unfolding *o-def* **by** (*rule isCont-o2*)

lemma (**in** *bounded-linear*) *isCont*: *isCont f a*
unfolding *isCont-def* **by** (*rule cont*)

lemma (**in** *bounded-bilinear*) *isCont*:
⟦*isCont f a*; *isCont g a*⟧ ⟹ *isCont* (λ*x*. *f x* ** *g x*) *a*
unfolding *isCont-def* **by** (*rule LIM*)

lemmas *isCont-scaleR* = *scaleR.isCont*

lemma *isCont-of-real*:
isCont f a ⟹ *isCont* (λ*x*. *of-real* (*f x*)) *a*
unfolding *isCont-def* **by** (*rule LIM-of-real*)

lemma *isCont-power*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *'b*::{*recpower*,*real-normed-algebra*}
shows *isCont f a* ⟹ *isCont* (λ*x*. *f x* ^ *n*) *a*
unfolding *isCont-def* **by** (*rule LIM-power*)

lemma *isCont-abs* [*simp*]: *isCont abs* (*a*::*real*)
by (*rule isCont-rabs* [*OF isCont-ident*])

19.3 Uniform Continuity

lemma *isUCont-isCont*: *isUCont f* ==> *isCont f x*
by (*simp add*: *isUCont-def isCont-def LIM-def*, *force*)

lemma *isUCont-Cauchy*:
⟦*isUCont f*; *Cauchy X*⟧ ⟹ *Cauchy* (λ*n*. *f* (*X n*))
unfolding *isUCont-def*
apply (*rule CauchyI*)
apply (*drule-tac x*=*e* **in** *spec*, *safe*)
apply (*drule-tac e*=*s* **in** *CauchyD*, *safe*)
apply (*rule-tac x*=*M* **in** *exI*, *simp*)
done

lemma (**in** *bounded-linear*) *isUCont*: *isUCont f*

unfolding *isUCont-def*
proof (*intro allI impI*)
  fix *r*::*real* **assume** *r*: $0 < r$
  obtain *K* **where** *K*: $0 < K$ **and** *norm-le*: $\bigwedge x.\ norm\ (f\ x) \leq norm\ x * K$
    using *pos-bounded* **by** *fast*
  show $\exists\, s{>}0.\ \forall\, x\ y.\ norm\ (x - y) < s \longrightarrow norm\ (f\ x - f\ y) < r$
  proof (*rule exI*, *safe*)
    from *r K* **show** $0 < r\ /\ K$ **by** (*rule divide-pos-pos*)
  next
    fix *x y* :: *'a*
    assume *xy*: $norm\ (x - y) < r\ /\ K$
    have $norm\ (f\ x - f\ y) = norm\ (f\ (x - y))$ **by** (*simp only*: *diff*)
    also have $\ldots \leq norm\ (x - y) * K$ **by** (*rule norm-le*)
    also from *K xy* **have** $\ldots < r$ **by** (*simp only*: *pos-less-divide-eq*)
    finally show $norm\ (f\ x - f\ y) < r$ **.**
  qed
qed

lemma (**in** *bounded-linear*) *Cauchy*: *Cauchy X* $\Longrightarrow$ *Cauchy* $(\lambda n.\ f\ (X\ n))$
by (*rule isUCont* [*THEN isUCont-Cauchy*])

19.4 Relation of LIM and LIMSEQ

lemma *LIMSEQ-SEQ-conv1*:
  fixes *a* :: *'a*::*real-normed-vector*
  assumes *X*: X *–– a ––> L*
  shows $\forall\, S.\ (\forall\, n.\ S\ n \neq a) \wedge S$ *––––> a* $\longrightarrow (\lambda n.\ X\ (S\ n))$ *––––> L*
proof (*safe intro*!: *LIMSEQ-I*)
  fix *S* :: *nat* $\Rightarrow$ *'a*
  fix *r* :: *real*
  assume *rgz*: $0 < r$
  assume *as*: $\forall\, n.\ S\ n \neq a$
  assume *S*: *S ––––> a*
  from *LIM-D* [*OF X rgz*] **obtain** *s*
    where *sgz*: $0 < s$
    and *aux*: $\bigwedge x.\ [\![x \neq a;\ norm\ (x - a) < s]\!] \Longrightarrow norm\ (X\ x - L) < r$
    by *fast*
  from *LIMSEQ-D* [*OF S sgz*]
  obtain *no* **where** $\forall\, n{\geq}no.\ norm\ (S\ n - a) < s$ **by** *blast*
  hence $\forall\, n{\geq}no.\ norm\ (X\ (S\ n) - L) < r$ **by** (*simp add*: *aux as*)
  thus $\exists\, no.\ \forall\, n{\geq}no.\ norm\ (X\ (S\ n) - L) < r$ **..**
qed

lemma *LIMSEQ-SEQ-conv2*:
  fixes *a* :: *real*
  assumes $\forall\, S.\ (\forall\, n.\ S\ n \neq a) \wedge S$ *––––> a* $\longrightarrow (\lambda n.\ X\ (S\ n))$ *––––> L*
  shows *X –– a ––> L*
proof (*rule ccontr*)
  assume $\neg$ *(X –– a ––> L)*

hence $\neg\ (\forall r > 0.\ \exists s > 0.\ \forall x.\ x \neq a\ \&\ norm\ (x - a) < s \longrightarrow norm\ (X\ x - L) < r)$ **by** (*unfold LIM-def*)
hence $\exists r > 0.\ \forall s > 0.\ \exists x.\ \neg(x \neq a \wedge |x - a| < s \longrightarrow norm\ (X\ x - L) < r)$ **by** *simp*
hence $\exists r > 0.\ \forall s > 0.\ \exists x.\ (x \neq a \wedge |x - a| < s \wedge norm\ (X\ x - L) \geq r)$ **by** (*simp add*: *linorder-not-less*)
then obtain r **where** *rdef*: $r > 0 \wedge (\forall s > 0.\ \exists x.\ (x \neq a \wedge |x - a| < s \wedge norm\ (X\ x - L) \geq r))$ **by** *auto*

let $?F = \lambda n{::}nat.\ SOME\ x.\ x{\neq}a \wedge |x - a| < inverse\ (real\ (Suc\ n)) \wedge norm\ (X\ x - L) \geq r$
have $\bigwedge n.\ \exists x.\ x{\neq}a \wedge |x - a| < inverse\ (real\ (Suc\ n)) \wedge norm\ (X\ x - L) \geq r$
using *rdef* **by** *simp*
hence *F*: $\bigwedge n.\ ?F\ n \neq a \wedge |?F\ n - a| < inverse\ (real\ (Suc\ n)) \wedge norm\ (X\ (?F\ n) - L) \geq r$
by (*rule someI-ex*)
hence *F1*: $\bigwedge n.\ ?F\ n \neq a$
and *F2*: $\bigwedge n.\ |?F\ n - a| < inverse\ (real\ (Suc\ n))$
and *F3*: $\bigwedge n.\ norm\ (X\ (?F\ n) - L) \geq r$
by *fast+*

have $?F \longrightarrow a$
proof (*rule LIMSEQ-I*, *unfold real-norm-def*)
fix *e*::*real*
assume $0 < e$

then have $\exists no.\ inverse\ (real\ (Suc\ no)) < e$ **by** (*rule reals-Archimedean*)
then obtain m **where** *nodef*: $inverse\ (real\ (Suc\ m)) < e$ **by** *auto*
show $\exists no.\ \forall n.\ no \leq n \longrightarrow |?F\ n - a| < e$
proof (*intro exI allI impI*)
fix n
assume *mlen*: $m \leq n$
have $|?F\ n - a| < inverse\ (real\ (Suc\ n))$
by (*rule F2*)
also have $inverse\ (real\ (Suc\ n)) \leq inverse\ (real\ (Suc\ m))$
using *mlen* **by** *auto*
also from *nodef* **have**
$inverse\ (real\ (Suc\ m)) < e$.
finally show $|?F\ n - a| < e$.
qed
qed

moreover have $\forall n.\ ?F\ n \neq a$
by (*rule allI*) (*rule F1*)

moreover from *prems* **have** $\forall S.\ (\forall n.\ S\ n \neq a) \wedge S \longrightarrow a \longrightarrow (\lambda n.\ X\ (S\ n)) \longrightarrow L$ **by** *simp*
ultimately have $(\lambda n.\ X\ (?F\ n)) \longrightarrow L$ **by** *simp*

moreover have ¬ ((λn. X (?F n)) ----> L)
proof −
{
fix no::nat
obtain n **where** n = no + 1 **by** simp
then have nolen: no ≤ n **by** simp

have norm (X (?F n) − L) ≥ r
by (rule F3)
with nolen **have** ∃n. no ≤ n ∧ norm (X (?F n) − L) ≥ r **by** fast
}
then have (∀no. ∃n. no ≤ n ∧ norm (X (?F n) − L) ≥ r) **by** simp
with rdef **have** ∃e>0. (∀no. ∃n. no ≤ n ∧ norm (X (?F n) − L) ≥ e) **by** auto
thus ?thesis **by** (unfold LIMSEQ-def, auto simp add: linorder-not-less)
qed
ultimately show False **by** simp
qed

lemma LIMSEQ-SEQ-conv:
(∀S. (∀n. S n ≠ a) ∧ S ----> (a::real) ⟶ (λn. X (S n)) ----> L) =
(X -- a --> L)
proof
assume ∀S. (∀n. S n ≠ a) ∧ S ----> a ⟶ (λn. X (S n)) ----> L
thus X -- a --> L **by** (rule LIMSEQ-SEQ-conv2)
next
assume (X -- a --> L)
thus ∀S. (∀n. S n ≠ a) ∧ S ----> a ⟶ (λn. X (S n)) ----> L **by** (rule LIMSEQ-SEQ-conv1)
qed

end

20 Deriv: Differentiation

theory Deriv
imports Lim Univ-Poly
begin

Standard Definitions

definition
deriv :: ['a::real-normed-field ⇒ 'a, 'a, 'a] ⇒ bool
— Differentiation: D is derivative of function f at x
((DERIV (-)/ (-)/ :> (-)) [1000, 1000, 60] 60) **where**
DERIV f x :> D = ((%h. (f(x + h) − f x) / h) -- 0 --> D)

definition
differentiable :: ['a::real-normed-field ⇒ 'a, 'a] ⇒ bool

(**infixl** *differentiable 60*) **where**
f differentiable x = (∃ *D*. *DERIV f x :> D*)

consts
Bolzano-bisect :: [*real*∗*real*=>*bool*, *real*, *real*, *nat*] => (*real*∗*real*)
primrec
Bolzano-bisect P a b 0 = (*a*,*b*)
Bolzano-bisect P a b (*Suc n*) =
(*let* (*x*,*y*) = *Bolzano-bisect P a b n*
in if P(*x*, (*x*+*y*)/*2*) *then* ((*x*+*y*)/*2*, *y*)
else (*x*, (*x*+*y*)/*2*))

20.1 Derivatives

lemma *DERIV-iff*: (*DERIV f x :> D*) = ((%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−> *D*)
by (*simp add*: *deriv-def*)

lemma *DERIV-D*: *DERIV f x :> D* ==> (%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−> *D*
by (*simp add*: *deriv-def*)

lemma *DERIV-const* [*simp*]: *DERIV* (λ*x*. *k*) *x :> 0*
by (*simp add*: *deriv-def*)

lemma *DERIV-ident* [*simp*]: *DERIV* (λ*x*. *x*) *x :> 1*
by (*simp add*: *deriv-def cong*: *LIM-cong*)

lemma *add-diff-add*:
fixes *a b c d* :: *'a*::*ab-group-add*
shows (*a* + *c*) − (*b* + *d*) = (*a* − *b*) + (*c* − *d*)
by *simp*

lemma *DERIV-add*:
⟦*DERIV f x :> D*; *DERIV g x :> E*⟧ ⟹ *DERIV* (λ*x*. *f x* + *g x*) *x :> D* + *E*
by (*simp only*: *deriv-def add-diff-add add-divide-distrib LIM-add*)

lemma *DERIV-minus*:
DERIV f x :> D ⟹ *DERIV* (λ*x*. − *f x*) *x :>* − *D*
by (*simp only*: *deriv-def minus-diff-minus divide-minus-left LIM-minus*)

lemma *DERIV-diff*:
⟦*DERIV f x :> D*; *DERIV g x :> E*⟧ ⟹ *DERIV* (λ*x*. *f x* − *g x*) *x :> D* − *E*
by (*simp only*: *diff-def DERIV-add DERIV-minus*)

lemma *DERIV-add-minus*:
⟦*DERIV f x :> D*; *DERIV g x :> E*⟧ ⟹ *DERIV* (λ*x*. *f x* + − *g x*) *x :> D* + − *E*

by (*simp only*: *DERIV-add DERIV-minus*)

lemma *DERIV-isCont*: *DERIV f x :> D* $\Longrightarrow$ *isCont f x*
proof (*unfold isCont-iff*)
  assume *DERIV f x :> D*
  hence (λ*h*. (*f*(*x*+*h*) − *f*(*x*)) / *h*) −− *0* −−> *D*
    by (*rule DERIV-D*)
  hence (λ*h*. (*f*(*x*+*h*) − *f*(*x*)) / *h* ∗ *h*) −− *0* −−> *D* ∗ *0*
    by (*intro LIM-mult LIM-ident*)
  hence (λ*h*. (*f*(*x*+*h*) − *f*(*x*)) ∗ (*h* / *h*)) −− *0* −−> *0*
    by *simp*
  hence (λ*h*. *f*(*x*+*h*) − *f*(*x*)) −− *0* −−> *0*
    by (*simp cong*: *LIM-cong*)
  thus (λ*h*. *f*(*x*+*h*)) −− *0* −−> *f*(*x*)
    by (*simp add*: *LIM-def*)
qed

lemma *DERIV-mult-lemma*:
  fixes *a b c d* :: *'a*::*real-field*
  shows (*a* ∗ *b* − *c* ∗ *d*) / *h* = *a* ∗ ((*b* − *d*) / *h*) + ((*a* − *c*) / *h*) ∗ *d*
by (*simp add*: *diff-minus add-divide-distrib* [*symmetric*] *ring-distribs*)

lemma *DERIV-mult'*:
  assumes *f*: *DERIV f x :> D*
  assumes *g*: *DERIV g x :> E*
  shows *DERIV* (λ*x*. *f x* ∗ *g x*) *x* :> *f x* ∗ *E* + *D* ∗ *g x*
proof (*unfold deriv-def*)
  from *f* **have** *isCont f x*
    by (*rule DERIV-isCont*)
  hence (λ*h*. *f*(*x*+*h*)) −− *0* −−> *f x*
    by (*simp only*: *isCont-iff*)
  hence (λ*h*. *f*(*x*+*h*) ∗ ((*g*(*x*+*h*) − *g x*) / *h*) +
            ((*f*(*x*+*h*) − *f x*) / *h*) ∗ *g x*)
          −− *0* −−> *f x* ∗ *E* + *D* ∗ *g x*
    by (*intro LIM-add LIM-mult LIM-const DERIV-D f g*)
  thus (λ*h*. (*f*(*x*+*h*) ∗ *g*(*x*+*h*) − *f x* ∗ *g x*) / *h*)
        −− *0* −−> *f x* ∗ *E* + *D* ∗ *g x*
    by (*simp only*: *DERIV-mult-lemma*)
qed

lemma *DERIV-mult*:
    [| *DERIV f x :> Da*; *DERIV g x :> Db* |]
     ==> *DERIV* (%*x*. *f x* ∗ *g x*) *x* :> (*Da* ∗ *g*(*x*)) + (*Db* ∗ *f*(*x*))
by (*drule* (*1*) *DERIV-mult'*, *simp only*: *mult-commute add-commute*)

lemma *DERIV-unique*:
    [| *DERIV f x :> D*; *DERIV f x :> E* |] ==> *D* = *E*
apply (*simp add*: *deriv-def*)
apply (*blast intro*: *LIM-unique*)

done

Differentiation of finite sum

lemma *DERIV-sumr* [*rule-format* (*no-asm*)]:
(∀ *r*. *m* ≤ *r* & *r* < (*m* + *n*) −−> *DERIV* (%*x*. *f r x*) *x* :> (*f' r x*))
−−> *DERIV* (%*x*. ∑ *n*=*m*..<*n*::*nat*. *f n x* :: *real*) *x* :> (∑ *r*=*m*..<*n*. *f' r x*)
apply (*induct n*)
apply (*auto intro*: *DERIV-add*)
done

Alternative definition for differentiability

lemma *DERIV-LIM-iff*:
((%*h*. (*f*(*a* + *h*) − *f*(*a*)) / *h*) −− 0 −−> *D*) =
((%*x*. (*f*(*x*)−*f*(*a*)) / (*x*−*a*)) −− *a* −−> *D*)
apply (*rule iffI*)
apply (*drule-tac k*=− *a* **in** *LIM-offset*)
apply (*simp add*: *diff-minus*)
apply (*drule-tac k*=*a* **in** *LIM-offset*)
apply (*simp add*: *add-commute*)
done

lemma *DERIV-iff2*: (*DERIV f x* :> *D*) = ((%*z*. (*f*(*z*) − *f*(*x*)) / (*z*−*x*)) −− *x* −−> *D*)
by (*simp add*: *deriv-def diff-minus* [*symmetric*] *DERIV-LIM-iff*)

lemma *inverse-diff-inverse*:
⟦(*a*::'*a*::*division-ring*) ≠ 0; *b* ≠ 0⟧
⟹ *inverse a* − *inverse b* = − (*inverse a* ∗ (*a* − *b*) ∗ *inverse b*)
by (*simp add*: *ring-simps*)

lemma *DERIV-inverse-lemma*:
⟦*a* ≠ 0; *b* ≠ (0::'*a*::*real-normed-field*)⟧
⟹ (*inverse a* − *inverse b*) / *h*
= − (*inverse a* ∗ ((*a* − *b*) / *h*) ∗ *inverse b*)
by (*simp add*: *inverse-diff-inverse*)

lemma *DERIV-inverse'*:
assumes *der*: *DERIV f x* :> *D*
assumes *neq*: *f x* ≠ 0
shows *DERIV* (λ*x*. *inverse* (*f x*)) *x* :> − (*inverse* (*f x*) ∗ *D* ∗ *inverse* (*f x*))
(**is** *DERIV* - - :> *?E*)
proof (*unfold DERIV-iff2*)
from *der* **have** *lim-f*: *f* −− *x* −−> *f x*
by (*rule DERIV-isCont* [*unfolded isCont-def*])

from *neq* **have** 0 < *norm* (*f x*) **by** *simp*
with *LIM-D* [*OF lim-f*] **obtain** *s*
where *s*: 0 < *s*

```
    and less-fx: ⋀z. ⟦z ≠ x; norm (z − x) < s⟧
                 ⟹ norm (f z − f x) < norm (f x)
    by fast

  show (λz. (inverse (f z) − inverse (f x)) / (z − x)) −− x −−> ?E
  proof (rule LIM-equal2 [OF s])
    fix z
    assume z ≠ x norm (z − x) < s
    hence norm (f z − f x) < norm (f x) by (rule less-fx)
    hence f z ≠ 0 by auto
    thus (inverse (f z) − inverse (f x)) / (z − x) =
          − (inverse (f z) * ((f z − f x) / (z − x)) * inverse (f x))
      using neq by (rule DERIV-inverse-lemma)
  next
    from der have (λz. (f z − f x) / (z − x)) −− x −−> D
      by (unfold DERIV-iff2)
    thus (λz. − (inverse (f z) * ((f z − f x) / (z − x)) * inverse (f x)))
          −− x −−> ?E
      by (intro LIM-mult LIM-inverse LIM-minus LIM-const lim-f neq)
  qed
qed

lemma DERIV-divide:
  ⟦DERIV f x :> D; DERIV g x :> E; g x ≠ 0⟧
   ⟹ DERIV (λx. f x / g x) x :> (D * g x − f x * E) / (g x * g x)
apply (subgoal-tac f x * − (inverse (g x) * E * inverse (g x)) +
        D * inverse (g x) = (D * g x − f x * E) / (g x * g x))
apply (erule subst)
apply (unfold divide-inverse)
apply (erule DERIV-mult')
apply (erule (1) DERIV-inverse')
apply (simp add: ring-distribs nonzero-inverse-mult-distrib)
apply (simp add: mult-ac)
done

lemma DERIV-power-Suc:
  fixes f :: 'a ⇒ 'a::{real-normed-field,recpower}
  assumes f: DERIV f x :> D
  shows DERIV (λx. f x ^ Suc n) x :> (1 + of-nat n) * (D * f x ^ n)
proof (induct n)
case 0
  show ?case by (simp add: power-Suc f)
case (Suc k)
  from DERIV-mult' [OF f Suc] show ?case
    apply (simp only: of-nat-Suc ring-distribs mult-1-left)
    apply (simp only: power-Suc right-distrib mult-ac add-ac)
    done
qed
```

lemma *DERIV-power*:
fixes $f :: 'a \Rightarrow 'a::\{real\text{-}normed\text{-}field,recpower\}$
assumes f: *DERIV f x :> D*
shows *DERIV* $(\lambda x.\ f\ x\ \hat{}\ n)\ x$ *:> of-nat n* $*$ $(D * f\ x\ \hat{}\ (n - Suc\ 0))$
by (*cases n*, *simp*, *simp add*: *DERIV-power-Suc f*)

lemma *CARAT-DERIV*:
(*DERIV f x :> l*) =
(∃ *g*. (∀ *z*. *f z* − *f x* = *g z* $*$ (*z*−*x*)) & *isCont g x* & *g x* = *l*)
(**is** *?lhs* = *?rhs*)
proof
assume *der*: *DERIV f x :> l*
show ∃ *g*. (∀ *z*. *f z* − *f x* = *g z* $*$ (*z*−*x*)) ∧ *isCont g x* ∧ *g x* = *l*
proof (*intro exI conjI*)
let *?g* = (%*z*. *if z* = *x then l else* (*f z* − *f x*) / (*z*−*x*))
show ∀ *z*. *f z* − *f x* = *?g z* $*$ (*z*−*x*) **by** *simp*
show *isCont ?g x* **using** *der*
by (*simp add*: *isCont-iff DERIV-iff diff-minus*
cong: *LIM-equal* [*rule-format*])
show *?g x* = *l* **by** *simp*
qed
next
assume *?rhs*
then obtain *g* **where**
(∀ *z*. *f z* − *f x* = *g z* $*$ (*z*−*x*)) **and** *isCont g x* **and** *g x* = *l* **by** *blast*
thus (*DERIV f x :> l*)
by (*auto simp add*: *isCont-iff DERIV-iff cong*: *LIM-cong*)
qed

lemma *DERIV-chain'*:
assumes *f*: *DERIV f x :> D*
assumes *g*: *DERIV g* (*f x*) *:> E*
shows *DERIV* $(\lambda x.\ g\ (f\ x))\ x$ *:> E* $*$ *D*
proof (*unfold DERIV-iff2*)
obtain *d* **where** *d*: ∀ *y*. *g y* − *g* (*f x*) = *d y* $*$ (*y* − *f x*)
and *cont-d*: *isCont d* (*f x*) **and** *dfx*: *d* (*f x*) = *E*
using *CARAT-DERIV* [*THEN iffD1*, *OF g*] **by** *fast*
from *f* **have** *f* −− *x* −−> *f x*
by (*rule DERIV-isCont* [*unfolded isCont-def*])
with *cont-d* **have** $(\lambda z.\ d\ (f\ z))$ −− *x* −−> *d* (*f x*)
by (*rule isCont-LIM-compose*)
hence $(\lambda z.\ d\ (f\ z) * ((f\ z - f\ x) / (z - x)))$
−− *x* −−> *d* (*f x*) $*$ *D*
by (*rule LIM-mult* [*OF - f* [*unfolded DERIV-iff2*]])

thus $(\lambda z.\ (g\ (f\ z) - g\ (f\ x))\ /\ (z - x)) -- x --> E * D$
by (*simp add*: *d dfx real-scaleR-def*)
qed

lemma *DERIV-cmult*:
$DERIV\ f\ x :> D ==> DERIV\ (\%x.\ c * f\ x)\ x :> c*D$
by (*drule DERIV-mult'* [*OF DERIV-const*], *simp*)

lemma *DERIV-chain*: $[|\ DERIV\ f\ (g\ x) :> Da;\ DERIV\ g\ x :> Db\ |] ==> DERIV\ (f\ o\ g)\ x :> Da * Db$
by (*drule* (*1*) *DERIV-chain'*, *simp add*: *o-def real-scaleR-def mult-commute*)

lemma *DERIV-chain2*: $[|\ DERIV\ f\ (g\ x) :> Da;\ DERIV\ g\ x :> Db\ |] ==> DERIV\ (\%x.\ f\ (g\ x))\ x :> Da * Db$
by (*auto dest*: *DERIV-chain simp add*: *o-def*)

lemma *DERIV-cmult-Id* [*simp*]: $DERIV\ (op * c)\ x :> c$
by (*cut-tac c = c* **and** *x = x* **in** *DERIV-ident* [*THEN DERIV-cmult*], *simp*)

lemma *DERIV-pow*: $DERIV\ (\%x.\ x\ \hat{}\ n)\ x :> real\ n * (x\ \hat{}\ (n - Suc\ 0))$
apply (*cut-tac DERIV-power* [*OF DERIV-ident*])
apply (*simp add*: *real-scaleR-def real-of-nat-def*)
done

Power of -1

lemma *DERIV-inverse*:
fixes $x :: 'a::\{real\text{-}normed\text{-}field,recpower\}$
shows $x \neq 0 ==> DERIV\ (\%x.\ inverse(x))\ x :> (-(inverse\ x\ \hat{}\ Suc\ (Suc\ 0)))$
by (*drule DERIV-inverse'* [*OF DERIV-ident*]) (*simp add*: *power-Suc*)

Derivative of inverse

lemma *DERIV-inverse-fun*:
fixes $x :: 'a::\{real\text{-}normed\text{-}field,recpower\}$
shows $[|\ DERIV\ f\ x :> d;\ f(x) \neq 0\ |]$
$==> DERIV\ (\%x.\ inverse(f\ x))\ x :> (-\ (d * inverse(f(x)\ \hat{}\ Suc\ (Suc\ 0))))$
by (*drule* (*1*) *DERIV-inverse'*) (*simp add*: *mult-ac power-Suc nonzero-inverse-mult-distrib*)

Derivative of quotient

lemma *DERIV-quotient*:
fixes $x :: 'a::\{real\text{-}normed\text{-}field,recpower\}$
shows $[|\ DERIV\ f\ x :> d;\ DERIV\ g\ x :> e;\ g(x) \neq 0\ |]$
$==> DERIV\ (\%y.\ f(y)\ /\ (g\ y))\ x :> (d*g(x) - (e*f(x)))\ /\ (g(x)\ \hat{}\ Suc\ (Suc\ 0))$
by (*drule* (*2*) *DERIV-divide*) (*simp add*: *mult-commute power-Suc*)

20.2 Differentiability predicate

lemma *differentiableD*: *f differentiable x ==> ∃ D. DERIV f x :> D*
by (*simp add*: *differentiable-def*)

lemma *differentiableI*: *DERIV f x :> D ==> f differentiable x*
by (*force simp add*: *differentiable-def*)

lemma *differentiable-const*: *(λz. a) differentiable x*
apply (*unfold differentiable-def*)
apply (*rule-tac x=0* **in** *exI*)
apply *simp*
done

lemma *differentiable-sum*:
assumes *f differentiable x*
and *g differentiable x*
shows *(λx. f x + g x) differentiable x*
proof −
from *prems* **have** *∃ D. DERIV f x :> D* **by** (*unfold differentiable-def*)
then obtain *df* **where** *DERIV f x :> df* **..**
moreover from *prems* **have** *∃ D. DERIV g x :> D* **by** (*unfold differentiable-def*)
then obtain *dg* **where** *DERIV g x :> dg* **..**
ultimately have *DERIV (λx. f x + g x) x :> df + dg* **by** (*rule DERIV-add*)
hence *∃ D. DERIV (λx. f x + g x) x :> D* **by** *auto*
thus *?thesis* **by** (*fold differentiable-def*)
qed

lemma *differentiable-diff*:
assumes *f differentiable x*
and *g differentiable x*
shows *(λx. f x − g x) differentiable x*
proof −
from *prems* **have** *f differentiable x* **by** *simp*
moreover
from *prems* **have** *∃ D. DERIV g x :> D* **by** (*unfold differentiable-def*)
then obtain *dg* **where** *DERIV g x :> dg* **..**
then have *DERIV (λx. − g x) x :> −dg* **by** (*rule DERIV-minus*)
hence *∃ D. DERIV (λx. − g x) x :> D* **by** *auto*
hence *(λx. − g x) differentiable x* **by** (*fold differentiable-def*)
ultimately
show *?thesis*
by (*auto simp*: *diff-def dest*: *differentiable-sum*)
qed

lemma *differentiable-mult*:
assumes *f differentiable x*
and *g differentiable x*
shows *(λx. f x * g x) differentiable x*
proof −

from *prems* **have** $\exists D.\ DERIV\ f\ x\ :>\ D$ **by** (*unfold differentiable-def*)
then obtain *df* **where** *DERIV f x :> df* **..**
moreover from *prems* **have** $\exists D.\ DERIV\ g\ x\ :>\ D$ **by** (*unfold differentiable-def*)
then obtain *dg* **where** *DERIV g x :> dg* **..**
ultimately have *DERIV* $(\lambda x.\ f\ x\ *\ g\ x)\ x\ :>\ df\ *\ g\ x\ +\ dg\ *\ f\ x$ **by** (*simp add*: *DERIV-mult*)
hence $\exists D.\ DERIV\ (\lambda x.\ f\ x\ *\ g\ x)\ x\ :>\ D$ **by** *auto*
thus *?thesis* **by** (*fold differentiable-def*)
qed

20.3 Nested Intervals and Bisection

Lemmas about nested intervals and proof by bisection (cf.Harrison). All considerably tidied by lcp.

lemma *lemma-f-mono-add* [*rule-format* (*no-asm*)]: $(\forall n.\ (f::nat=>real)\ n \leq f\ (Suc\ n))\ --> f\ m \leq f(m + no)$
apply (*induct no*)
apply (*auto intro*: *order-trans*)
done

lemma *f-inc-g-dec-Beq-f*: [| $\forall n.\ f(n) \leq f(Suc\ n)$;
$\forall n.\ g(Suc\ n) \leq g(n)$;
$\forall n.\ f(n) \leq g(n)$ |]
==> *Bseq* $(f :: nat \Rightarrow real)$
apply (*rule-tac k = f 0* **and** *K = g 0* **in** *BseqI2*, *rule allI*)
apply (*induct-tac n*)
apply (*auto intro*: *order-trans*)
apply (*rule-tac y = g* (*Suc na*) **in** *order-trans*)
apply (*induct-tac* [*2*] *na*)
apply (*auto intro*: *order-trans*)
done

lemma *f-inc-g-dec-Beq-g*: [| $\forall n.\ f(n) \leq f(Suc\ n)$;
$\forall n.\ g(Suc\ n) \leq g(n)$;
$\forall n.\ f(n) \leq g(n)$ |]
==> *Bseq* $(g :: nat \Rightarrow real)$
apply (*subst Bseq-minus-iff* [*symmetric*])
apply (*rule-tac g = %x. - (f x)* **in** *f-inc-g-dec-Beq-f*)
apply *auto*
done

lemma *f-inc-imp-le-lim*:
fixes $f :: nat \Rightarrow real$
shows $[\![\forall n.\ f\ n \leq f\ (Suc\ n);\ convergent\ f]\!] \Longrightarrow f\ n \leq lim\ f$
apply (*rule linorder-not-less* [*THEN iffD1*])
apply (*auto simp add*: *convergent-LIMSEQ-iff LIMSEQ-iff monoseq-Suc*)
apply (*drule real-less-sum-gt-zero*)
apply (*drule-tac x = f n + - lim f* **in** *spec*, *safe*)
apply (*drule-tac P = %na. no≤na --> ?Q na* **and** *x = no + n* **in** *spec*, *auto*)

apply (*subgoal-tac lim f* $\leq$ *f* (*no* + *n*))
apply (*drule-tac no=no* **and** *m=n* **in** *lemma-f-mono-add*)
apply (*auto simp add*: *add-commute*)
apply (*induct-tac no*)
apply *simp*
apply (*auto intro*: *order-trans simp add*: *diff-minus abs-if*)
done

lemma *lim-uminus*: *convergent g* ==> *lim* (%*x*. − *g x*) = − (*lim g*)
apply (*rule LIMSEQ-minus* [*THEN limI*])
apply (*simp add*: *convergent-LIMSEQ-iff*)
done

lemma *g-dec-imp-lim-le*:
fixes *g* :: *nat* ⇒ *real*
shows ⟦∀ *n*. *g* (*Suc n*) ≤ *g*(*n*); *convergent g*⟧ ⟹ *lim g* ≤ *g n*
apply (*subgoal-tac* − (*g n*) ≤ − (*lim g*))
apply (*cut-tac* [*2*] *f* = %*x*. − (*g x*) **in** *f-inc-imp-le-lim*)
apply (*auto simp add*: *lim-uminus convergent-minus-iff* [*symmetric*])
done

lemma *lemma-nest*: [| ∀ *n*. *f*(*n*) ≤ *f*(*Suc n*);
∀ *n*. *g*(*Suc n*) ≤ *g*(*n*);
∀ *n*. *f*(*n*) ≤ *g*(*n*) |]
==> ∃ *l m* :: *real*. *l* ≤ *m* & ((∀ *n*. *f*(*n*) ≤ *l*) & *f* −−−−> *l*) &
((∀ *n*. *m* ≤ *g*(*n*)) & *g* −−−−> *m*)
apply (*subgoal-tac monoseq f* & *monoseq g*)
prefer *2* **apply** (*force simp add*: *LIMSEQ-iff monoseq-Suc*)
apply (*subgoal-tac Bseq f* & *Bseq g*)
prefer *2* **apply** (*blast intro*: *f-inc-g-dec-Beq-f f-inc-g-dec-Beq-g*)
apply (*auto dest*!: *Bseq-monoseq-convergent simp add*: *convergent-LIMSEQ-iff*)
apply (*rule-tac x* = *lim f* **in** *exI*)
apply (*rule-tac x* = *lim g* **in** *exI*)
apply (*auto intro*: *LIMSEQ-le*)
apply (*auto simp add*: *f-inc-imp-le-lim g-dec-imp-lim-le convergent-LIMSEQ-iff*)
done

lemma *lemma-nest-unique*: [| ∀ *n*. *f*(*n*) ≤ *f*(*Suc n*);
∀ *n*. *g*(*Suc n*) ≤ *g*(*n*);
∀ *n*. *f*(*n*) ≤ *g*(*n*);
(%*n*. *f*(*n*) − *g*(*n*)) −−−−> *0* |]
==> ∃ *l*::*real*. ((∀ *n*. *f*(*n*) ≤ *l*) & *f* −−−−> *l*) &
((∀ *n*. *l* ≤ *g*(*n*)) & *g* −−−−> *l*)
apply (*drule lemma-nest*, *auto*)
apply (*subgoal-tac l* = *m*)
apply (*drule-tac* [*2*] *X* = *f* **in** *LIMSEQ-diff*)
apply (*auto intro*: *LIMSEQ-unique*)
done

The universal quantifiers below are required for the declaration of *Bolzano-nest-unique*

below.

lemma *Bolzano-bisect-le*:
$a \leq b \Longrightarrow \forall n.\ fst\ (Bolzano\text{-}bisect\ P\ a\ b\ n) \leq snd\ (Bolzano\text{-}bisect\ P\ a\ b\ n)$
apply (*rule allI*)
apply (*induct-tac n*)
apply (*auto simp add*: *Let-def split-def*)
done

lemma *Bolzano-bisect-fst-le-Suc*: $a \leq b \Longrightarrow$
$\forall n.\ fst(Bolzano\text{-}bisect\ P\ a\ b\ n) \leq fst\ (Bolzano\text{-}bisect\ P\ a\ b\ (Suc\ n))$
apply (*rule allI*)
apply (*induct-tac n*)
apply (*auto simp add*: *Bolzano-bisect-le Let-def split-def*)
done

lemma *Bolzano-bisect-Suc-le-snd*: $a \leq b \Longrightarrow$
$\forall n.\ snd(Bolzano\text{-}bisect\ P\ a\ b\ (Suc\ n)) \leq snd\ (Bolzano\text{-}bisect\ P\ a\ b\ n)$
apply (*rule allI*)
apply (*induct-tac n*)
apply (*auto simp add*: *Bolzano-bisect-le Let-def split-def*)
done

lemma *eq-divide-2-times-iff*: $((x::real) = y\ /\ (2 * z)) = (2 * x = y/z)$
apply (*auto*)
apply (*drule-tac f* $= \%u.\ (1/2) * u$ **in** *arg-cong*)
apply (*simp*)
done

lemma *Bolzano-bisect-diff*:
$a \leq b \Longrightarrow$
$snd(Bolzano\text{-}bisect\ P\ a\ b\ n) - fst(Bolzano\text{-}bisect\ P\ a\ b\ n) =$
$(b-a)\ /\ (2\ \hat{}\ n)$
apply (*induct n*)
apply (*auto simp add*: *eq-divide-2-times-iff add-divide-distrib Let-def split-def*)
done

lemmas *Bolzano-nest-unique* =
lemma-nest-unique
[*OF Bolzano-bisect-fst-le-Suc Bolzano-bisect-Suc-le-snd Bolzano-bisect-le*]

lemma *not-P-Bolzano-bisect*:
assumes *P*: $!!a\ b\ c.\ [|\ P(a,b);\ P(b,c);\ a \leq b;\ b \leq c|] \Longrightarrow P(a,c)$
and *notP*: $\sim P(a,b)$
and *le*: $a \leq b$
shows $\sim P(fst(Bolzano\text{-}bisect\ P\ a\ b\ n),\ snd(Bolzano\text{-}bisect\ P\ a\ b\ n))$
proof (*induct n*)
case *0* **show** *?case* **using** *notP* **by** *simp*
next

case (*Suc n*)
thus *?case*
by (*auto simp del*: *surjective-pairing* [*symmetric*]
simp add: *Let-def split-def Bolzano-bisect-le* [*OF le*]
P [*of fst* (*Bolzano-bisect P a b n*) - *snd* (*Bolzano-bisect P a b n*)])
qed

lemma *not-P-Bolzano-bisect'*:
[| ∀ *a b c*. *P*(*a*,*b*) & *P*(*b*,*c*) & *a* ≤ *b* & *b* ≤ *c* −−> *P*(*a*,*c*);
~ *P*(*a*,*b*); *a* ≤ *b* |] ==>
∀ *n*. ~ *P*(*fst*(*Bolzano-bisect P a b n*), *snd*(*Bolzano-bisect P a b n*))
by (*blast elim*!: *not-P-Bolzano-bisect* [*THEN* [*2*] *rev-notE*])

lemma *lemma-BOLZANO*:
[| ∀ *a b c*. *P*(*a*,*b*) & *P*(*b*,*c*) & *a* ≤ *b* & *b* ≤ *c* −−> *P*(*a*,*c*);
∀ *x*. ∃ *d*::*real*. *0* < *d* &
(∀ *a b*. *a* ≤ *x* & *x* ≤ *b* & (*b*−*a*) < *d* −−> *P*(*a*,*b*));
a ≤ *b* |]
==> *P*(*a*,*b*)
apply (*rule Bolzano-nest-unique* [**where** *P1*=*P*, *THEN exE*], *assumption*+)
apply (*rule LIMSEQ-minus-cancel*)
apply (*simp* (*no-asm-simp*) *add*: *Bolzano-bisect-diff LIMSEQ-divide-realpow-zero*)
apply (*rule ccontr*)
apply (*drule not-P-Bolzano-bisect'*, *assumption*+)
apply (*rename-tac l*)
apply (*drule-tac x* = *l* **in** *spec*, *clarify*)
apply (*simp add*: *LIMSEQ-def*)
apply (*drule-tac P* = %*r*. *0*<*r* −−> *?Q r* **and** *x* = *d*/*2* **in** *spec*)
apply (*drule-tac P* = %*r*. *0*<*r* −−> *?Q r* **and** *x* = *d*/*2* **in** *spec*)
apply (*drule real-less-half-sum*, *auto*)
apply (*drule-tac x* = *fst* (*Bolzano-bisect P a b* (*no* + *noa*)) **in** *spec*)
apply (*drule-tac x* = *snd* (*Bolzano-bisect P a b* (*no* + *noa*)) **in** *spec*)
apply *safe*
apply (*simp-all* (*no-asm-simp*))
apply (*rule-tac y* = *abs* (*fst* (*Bolzano-bisect P a b* (*no* + *noa*)) − *l*) + *abs* (*snd* (*Bolzano-bisect P a b* (*no* + *noa*)) − *l*) **in** *order-le-less-trans*)
apply (*simp* (*no-asm-simp*) *add*: *abs-if*)
apply (*rule real-sum-of-halves* [*THEN subst*])
apply (*rule add-strict-mono*)
apply (*simp-all add*: *diff-minus* [*symmetric*])
done

lemma *lemma-BOLZANO2*: ((∀ *a b c*. (*a* ≤ *b* & *b* ≤ *c* & *P*(*a*,*b*) & *P*(*b*,*c*)) −−> *P*(*a*,*c*)) &
(∀ *x*. ∃ *d*::*real*. *0* < *d* &

```
            (∀ a b. a ≤ x & x ≤ b & (b−a) < d −−> P(a,b))))
     −−> (∀ a b. a ≤ b −−> P(a,b))
apply clarify
apply (blast intro: lemma-BOLZANO)
done
```

20.4 Intermediate Value Theorem

Prove Contrapositive by Bisection

```
lemma IVT: [| f(a::real) ≤ (y::real); y ≤ f(b);
        a ≤ b;
        (∀ x. a ≤ x & x ≤ b −−> isCont f x) |]
     ==> ∃ x. a ≤ x & x ≤ b & f(x) = y
apply (rule contrapos-pp, assumption)
apply (cut-tac P = % (u,v) . a ≤ u & u ≤ v & v ≤ b −−> ~ (f (u) ≤ y & y ≤
f (v)) in lemma-BOLZANO2)
apply safe
apply simp-all
apply (simp add: isCont-iff LIM-def)
apply (rule ccontr)
apply (subgoal-tac a ≤ x & x ≤ b)
 prefer 2
 apply simp
 apply (drule-tac P = %d. 0<d −−> ?P d and x = 1 in spec, arith)
apply (drule-tac x = x in spec)+
apply simp
apply (drule-tac P = %r. ?P r −−> (∃ s>0. ?Q r s)  and x = |y − f x| in spec)
apply safe
apply simp
apply (drule-tac x = s in spec, clarify)
apply (cut-tac x = f x and y = y in linorder-less-linear, safe)
apply (drule-tac x = ba−x in spec)
apply (simp-all add: abs-if)
apply (drule-tac x = aa−x in spec)
apply (case-tac x ≤ aa, simp-all)
done

lemma IVT2: [| f(b::real) ≤ (y::real); y ≤ f(a);
        a ≤ b;
        (∀ x. a ≤ x & x ≤ b −−> isCont f x)
     |] ==> ∃ x. a ≤ x & x ≤ b & f(x) = y
apply (subgoal-tac − f a ≤ −y & −y ≤ − f b, clarify)
apply (drule IVT [where f = %x. − f x], assumption)
apply (auto intro: isCont-minus)
done


lemma IVT-objl: (f(a::real) ≤ (y::real) & y ≤ f(b) & a ≤ b &
     (∀ x. a ≤ x & x ≤ b −−> isCont f x))
```

$--> (\exists x.\ a \leq x\ \&\ x \leq b\ \&\ f(x) = y)$
apply (*blast intro*: *IVT*)
done

lemma *IVT2-objl*: $(f(b::real) \leq (y::real)\ \&\ y \leq f(a)\ \&\ a \leq b\ \&$
$(\forall x.\ a \leq x\ \&\ x \leq b\ --> isCont\ f\ x))$
$--> (\exists x.\ a \leq x\ \&\ x \leq b\ \&\ f(x) = y)$
apply (*blast intro*: *IVT2*)
done

By bisection, function continuous on closed interval is bounded above

lemma *isCont-bounded*:
$[|\ a \leq b; \forall x.\ a \leq x\ \&\ x \leq b\ --> isCont\ f\ x\ |]$
$==> \exists M::real.\ \forall x::real.\ a \leq x\ \&\ x \leq b\ --> f(x) \leq M$
apply (*cut-tac P* = % (*u*,*v*) . $a \leq u\ \&\ u \leq v\ \&\ v \leq b\ -->\ (\exists M.\ \forall x.\ u \leq x\ \&$
$x \leq v\ -->\ f\ x \leq M)$ **in** *lemma-BOLZANO2*)
apply *safe*
apply *simp-all*
apply (*rename-tac x xa ya M Ma*)
apply (*cut-tac x* = *M* **and** *y* = *Ma* **in** *linorder-linear*, *safe*)
apply (*rule-tac x* = *Ma* **in** *exI*, *clarify*)
apply (*cut-tac x* = *xb* **and** *y* = *xa* **in** *linorder-linear*, *force*)
apply (*rule-tac x* = *M* **in** *exI*, *clarify*)
apply (*cut-tac x* = *xb* **and** *y* = *xa* **in** *linorder-linear*, *force*)
apply (*case-tac* $a \leq x\ \&\ x \leq b$)
apply (*rule-tac* [*2*] *x* = *1* **in** *exI*)
prefer *2* **apply** *force*
apply (*simp add*: *LIM-def isCont-iff*)
apply (*drule-tac x* = *x* **in** *spec*, *auto*)
apply (*erule-tac V* = $\forall M.\ \exists x.\ a \leq x\ \&\ x \leq b\ \&\ \sim f\ x \leq M$ **in** *thin-rl*)
apply (*drule-tac x* = *1* **in** *spec*, *auto*)
apply (*rule-tac x* = *s* **in** *exI*, *clarify*)
apply (*rule-tac x* = $|f\ x| + 1$ **in** *exI*, *clarify*)
apply (*drule-tac x* = *xa*−*x* **in** *spec*)
apply (*auto simp add*: *abs-ge-self*)
done

Refine the above to existence of least upper bound

lemma *lemma-reals-complete*: $((\exists x.\ x \in S)\ \&\ (\exists y.\ isUb\ UNIV\ S\ (y::real))) -->$
$(\exists t.\ isLub\ UNIV\ S\ t)$
by (*blast intro*: *reals-complete*)

lemma *isCont-has-Ub*: $[|\ a \leq b; \forall x.\ a \leq x\ \&\ x \leq b\ --> isCont\ f\ x\ |]$
$==> \exists M::real.\ (\forall x::real.\ a \leq x\ \&\ x \leq b\ --> f(x) \leq M)\ \&$
$(\forall N.\ N < M\ --> (\exists x.\ a \leq x\ \&\ x \leq b\ \&\ N < f(x)))$
apply (*cut-tac S* = *Collect* (%*y*. $\exists x.\ a \leq x\ \&\ x \leq b\ \&\ y = f\ x$)
in *lemma-reals-complete*)
apply *auto*
apply (*drule isCont-bounded*, *assumption*)

apply (*auto simp add*: *isUb-def leastP-def isLub-def setge-def setle-def*)
apply (*rule exI*, *auto*)
apply (*auto dest*!: *spec simp add*: *linorder-not-less*)
done

Now show that it attains its upper bound

lemma *isCont-eq-Ub*:
  assumes *le*: $a \leq b$
    and *con*: $\forall x$::*real*. $a \leq x$ & $x \leq b$ --> *isCont f x*
  shows $\exists M$::*real*. $(\forall x.\ a \leq x$ & $x \leq b$ --> $f(x) \leq M)$ &
      $(\exists x.\ a \leq x$ & $x \leq b$ & $f(x) = M)$
proof −
  from *isCont-has-Ub* [*OF le con*]
  obtain *M* **where** *M1*: $\forall x.\ a \leq x \wedge x \leq b \longrightarrow f\,x \leq M$
      and *M2*: !!*N*. *N*<*M* ==> $\exists x.\ a \leq x \wedge x \leq b \wedge N < f\,x$ **by** *blast*
  show *?thesis*
  proof (*intro exI*, *intro conjI*)
    show $\forall x.\ a \leq x \wedge x \leq b \longrightarrow f\,x \leq M$ **by** (*rule M1*)
    show $\exists x.\ a \leq x \wedge x \leq b \wedge f\,x = M$
    proof (*rule ccontr*)
      assume $\neg\ (\exists x.\ a \leq x \wedge x \leq b \wedge f\,x = M)$
      with *M1* **have** *M3*: $\forall x.\ a \leq x$ & $x \leq b$ --> $f\,x < M$
        by (*fastsimp simp add*: *linorder-not-le* [*symmetric*])
      hence $\forall x.\ a \leq x$ & $x \leq b$ --> *isCont* (%*x*. *inverse* $(M - f\,x))\ x$
        by (*auto simp add*: *isCont-inverse isCont-diff con*)
      from *isCont-bounded* [*OF le this*]
      obtain *k* **where** *k*: !!*x*. $a \leq x$ & $x \leq b$ --> *inverse* $(M - f\,x) \leq k$ **by** *auto*
      have *Minv*: !!*x*. $a \leq x$ & $x \leq b$ --> $0 <$ *inverse* $(M - f\ (x))$
        by (*simp add*: *M3 compare-rls*)
      have !!*x*. $a \leq x$ & $x \leq b$ --> *inverse* $(M - f\,x) < k$+*1* **using** *k*
        by (*auto intro*: *order-le-less-trans* [*of* - *k*])
      with *Minv*
      have !!*x*. $a \leq x$ & $x \leq b$ --> *inverse*(*k*+*1*) < *inverse*(*inverse*$(M - f\,x)$)
        by (*intro strip less-imp-inverse-less*, *simp-all*)
      hence *invlt*: !!*x*. $a \leq x$ & $x \leq b$ --> *inverse*(*k*+*1*) < $M - f\,x$
        by *simp*
      have M − *inverse* (*k*+*1*) < M **using** *k* [*of a*] *Minv* [*of a*] *le*
        by (*simp*, *arith*)
      from *M2* [*OF this*]
      obtain *x* **where** *ax*: $a \leq x$ & $x \leq b$ & M − *inverse*(*k*+*1*) < $f\,x$ **..**
      thus *False* **using** *invlt* [*of x*] **by** *force*
    qed
  qed
qed

Same theorem for lower bound

lemma *isCont-eq-Lb*: [| $a \leq b$; $\forall x.\ a \leq x$ & $x \leq b$ --> *isCont f x* |]
    ==> $\exists M$::*real*. $(\forall x$::*real*. $a \leq x$ & $x \leq b$ --> $M \leq f(x))$ &
      $(\exists x.\ a \leq x$ & $x \leq b$ & $f(x) = M)$

apply (*subgoal-tac* $\forall x.\ a \leq x\ \&\ x \leq b \longrightarrow isCont\ (\%x.\ -\ (f\ x))\ x$)
prefer *2* **apply** (*blast intro*: *isCont-minus*)
apply (*drule-tac* $f = (\%x.\ -\ (f\ x))$ **in** *isCont-eq-Ub*)
apply *safe*
apply *auto*
done

Another version.

lemma *isCont-Lb-Ub*: $[|a \leq b; \forall x.\ a \leq x\ \&\ x \leq b \longrightarrow isCont\ f\ x\ |]$
$\Longrightarrow \exists L\ M{::}real.\ (\forall x{::}real.\ a \leq x\ \&\ x \leq b \longrightarrow L \leq f(x)\ \&\ f(x) \leq M)\ \&$
$(\forall y.\ L \leq y\ \&\ y \leq M \longrightarrow (\exists x.\ a \leq x\ \&\ x \leq b\ \&\ (f(x) = y)))$
apply (*frule isCont-eq-Lb*)
apply (*frule-tac* [*2*] *isCont-eq-Ub*)
apply (*assumption+*, *safe*)
apply (*rule-tac* $x = f\ x$ **in** *exI*)
apply (*rule-tac* $x = f\ xa$ **in** *exI*, *simp*, *safe*)
apply (*cut-tac* $x = x$ **and** $y = xa$ **in** *linorder-linear*, *safe*)
apply (*cut-tac* $f = f$ **and** $a = x$ **and** $b = xa$ **and** $y = y$ **in** *IVT-objl*)
apply (*cut-tac* [*2*] $f = f$ **and** $a = xa$ **and** $b = x$ **and** $y = y$ **in** *IVT2-objl*, *safe*)
apply (*rule-tac* [*2*] $x = xb$ **in** *exI*)
apply (*rule-tac* [*4*] $x = xb$ **in** *exI*, *simp-all*)
done

If $(0{::}'a) < f'\ x$ then x is Locally Strictly Increasing At The Right

lemma *DERIV-left-inc*:
fixes $f :: real \Rightarrow real$
assumes *der*: *DERIV f x :> l*
and *l*: $0 < l$
shows $\exists d > 0.\ \forall h > 0.\ h < d \longrightarrow f(x) < f(x + h)$
proof −
from *l der* [*THEN DERIV-D*, *THEN LIM-D* [**where** $r = l$]]
have $\exists s > 0.\ (\forall z.\ z \neq 0 \wedge |z| < s \longrightarrow |(f(x+z) - f\ x)\ /\ z - l| < l)$
by (*simp add*: *diff-minus*)
then obtain *s*
where *s*: $0 < s$
and *all*: $!!z.\ z \neq 0 \wedge |z| < s \longrightarrow |(f(x+z) - f\ x)\ /\ z - l| < l$
by *auto*
thus *?thesis*
proof (*intro exI conjI strip*)
show $0<s$ **using** *s* **.**
fix *h*::*real*
assume $0 < h\ h < s$
with *all* [*of h*] **show** $f\ x < f\ (x+h)$
proof (*simp add*: *abs-if pos-less-divide-eq diff-minus* [*symmetric*]
split add: *split-if-asm*)
assume $\sim\ (f\ (x+h) - f\ x)\ /\ h < l$ **and** *h*: $0 < h$
with *l*
have $0 < (f\ (x+h) - f\ x)\ /\ h$ **by** *arith*
thus $f\ x < f\ (x+h)$

```
  by (simp add: pos-less-divide-eq h)
   qed
  qed
qed

lemma DERIV-left-dec:
  fixes f :: real => real
  assumes der: DERIV f x :> l
     and l:   l < 0
  shows ∃ d > 0. ∀ h > 0. h < d --> f(x) < f(x-h)
proof -
  from l der [THEN DERIV-D, THEN LIM-D [where r = -l]]
  have ∃ s > 0. (∀ z. z ≠ 0 ∧ |z| < s ⟶ |(f(x+z) - f x) / z - l| < -l)
    by (simp add: diff-minus)
  then obtain s
       where s:   0 < s
         and all: !!z. z ≠ 0 ∧ |z| < s ⟶ |(f(x+z) - f x) / z - l| < -l
    by auto
  thus ?thesis
  proof (intro exI conjI strip)
    show 0<s using s .
    fix h::real
    assume 0 < h h < s
    with all [of -h] show f x < f (x-h)
    proof (simp add: abs-if pos-less-divide-eq diff-minus [symmetric]
    split add: split-if-asm)
      assume  -((f (x-h) - f x) / h) < l and h: 0 < h
      with l
      have 0 < (f (x-h) - f x) / h by arith
      thus f x < f (x-h)
  by (simp add: pos-less-divide-eq h)
   qed
  qed
qed

lemma DERIV-local-max:
  fixes f :: real => real
  assumes der: DERIV f x :> l
     and d:   0 < d
     and le:  ∀ y. |x-y| < d --> f(y) ≤ f(x)
  shows l = 0
proof (cases rule: linorder-cases [of l 0])
  case equal thus ?thesis .
next
  case less
  from DERIV-left-dec [OF der less]
  obtain d' where d': 0 < d'
            and lt: ∀ h > 0. h < d' ⟶ f x < f (x-h) by blast
  from real-lbound-gt-zero [OF d d']
```

obtain e **where** $0 < e \wedge e < d \wedge e < d'$ **..**
with *lt le* [*THEN spec* [**where** $x{=}x{-}e$]]
show *?thesis* **by** (*auto simp add*: *abs-if*)
next
case *greater*
from *DERIV-left-inc* [*OF der greater*]
obtain d' **where** d': $0 < d'$
and *lt*: $\forall h > 0.\ h < d' \longrightarrow f\ x < f\ (x + h)$ **by** *blast*
from *real-lbound-gt-zero* [*OF d d'*]
obtain e **where** $0 < e \wedge e < d \wedge e < d'$ **..**
with *lt le* [*THEN spec* [**where** $x{=}x{+}e$]]
show *?thesis* **by** (*auto simp add*: *abs-if*)
qed

Similar theorem for a local minimum

lemma *DERIV-local-min*:
fixes f :: *real* => *real*
shows [| *DERIV f x* :> l; $0 < d$; $\forall y.\ |x{-}y| < d$ --> $f(x) \leq f(y)$ |] ==> $l = 0$
by (*drule DERIV-minus* [*THEN DERIV-local-max*], *auto*)

In particular, if a function is locally flat

lemma *DERIV-local-const*:
fixes f :: *real* => *real*
shows [| *DERIV f x* :> l; $0 < d$; $\forall y.\ |x{-}y| < d$ --> $f(x) = f(y)$ |] ==> $l = 0$
by (*auto dest*!: *DERIV-local-max*)

Lemma about introducing open ball in open interval

lemma *lemma-interval-lt*:
[| $a < x$; $x < b$ |]
==> $\exists d{::}real.\ 0 < d\ \&\ (\forall y.\ |x{-}y| < d$ --> $a < y\ \&\ y < b)$
apply (*simp add*: *abs-less-iff*)
apply (*insert linorder-linear* [*of* $x{-}a$ $b{-}x$], *safe*)
apply (*rule-tac* $x = x{-}a$ **in** *exI*)
apply (*rule-tac* [*2*] $x = b{-}x$ **in** *exI*, *auto*)
done

lemma *lemma-interval*: [| $a < x$; $x < b$ |] ==>
$\exists d{::}real.\ 0 < d\ \&\ (\forall y.\ |x{-}y| < d$ --> $a \leq y\ \&\ y \leq b)$
apply (*drule lemma-interval-lt*, *auto*)
apply (*auto intro*!: *exI*)
done

Rolle's Theorem. If f is defined and continuous on the closed interval $[a,b]$ and differentiable on the open interval (a,b), and $f\ a = f\ b$, then there exists $x0 \in (a,b)$ such that $f'\ x0 = (0::'a)$

theorem *Rolle*:

assumes *lt*: $a < b$
and *eq*: $f(a) = f(b)$
and *con*: $\forall x.\ a \leq x\ \&\ x \leq b \dashrightarrow isCont\ f\ x$
and *dif* [*rule-format*]: $\forall x.\ a < x\ \&\ x < b \dashrightarrow f\ differentiable\ x$
shows $\exists z{::}real.\ a < z\ \&\ z < b\ \&\ DERIV\ f\ z :> 0$
proof −
have *le*: $a \leq b$ **using** *lt* **by** *simp*
from *isCont-eq-Ub* [*OF le con*]
obtain *x* **where** *x-max*: $\forall z.\ a \leq z \wedge z \leq b \longrightarrow f\ z \leq f\ x$
and *alex*: $a \leq x$ **and** *xleb*: $x \leq b$
by *blast*
from *isCont-eq-Lb* [*OF le con*]
obtain x' **where** x'*-min*: $\forall z.\ a \leq z \wedge z \leq b \longrightarrow f\ x' \leq f\ z$
and $alex'$: $a \leq x'$ **and** $x'leb$: $x' \leq b$
by *blast*
show *?thesis*
proof *cases*
assume *axb*: $a < x\ \&\ x < b$
— f attains its maximum within the interval
hence *ax*: $a<x$ **and** *xb*: $x<b$ **by** *auto*
from *lemma-interval* [*OF ax xb*]
obtain *d* **where** *d*: $0<d$ **and** *bound*: $\forall y.\ |x-y| < d \longrightarrow a \leq y \wedge y \leq b$
by *blast*
hence $bound'$: $\forall y.\ |x-y| < d \longrightarrow f\ y \leq f\ x$ **using** *x-max*
by *blast*
from *differentiableD* [*OF dif* [*OF axb*]]
obtain *l* **where** *der*: $DERIV\ f\ x :> l$ **..**
have *l=0* **by** (*rule DERIV-local-max* [*OF der d bound'*])
— the derivative at a local maximum is zero
thus *?thesis* **using** *ax xb der* **by** *auto*
next
assume *notaxb*: $\sim (a < x\ \&\ x < b)$
hence *xeqab*: $x=a \mid x=b$ **using** *alex xleb* **by** *arith*
hence *fb-eq-fx*: $f\ b = f\ x$ **by** (*auto simp add*: *eq*)
show *?thesis*
proof *cases*
assume $ax'b$: $a < x'\ \&\ x' < b$
— f attains its minimum within the interval
hence ax': $a<x'$ **and** $x'b$: $x'<b$ **by** *auto*
from *lemma-interval* [*OF ax' x'b*]
obtain *d* **where** *d*: $0<d$ **and** *bound*: $\forall y.\ |x'-y| < d \longrightarrow a \leq y \wedge y \leq b$
by *blast*
hence $bound'$: $\forall y.\ |x'-y| < d \longrightarrow f\ x' \leq f\ y$ **using** x'*-min*
by *blast*
from *differentiableD* [*OF dif* [*OF ax'b*]]
obtain *l* **where** *der*: $DERIV\ f\ x' :> l$ **..**
have *l=0* **by** (*rule DERIV-local-min* [*OF der d bound'*])
— the derivative at a local minimum is zero
thus *?thesis* **using** *ax' x'b der* **by** *auto*

```
    next
      assume notax'b: ~ (a < x' & x' < b)
        — f is constant througout the interval
      hence x'eqab: x'=a | x'=b using alex' x'leb by arith
      hence fb-eq-fx': f b = f x' by (auto simp add: eq)
      from dense [OF lt]
      obtain r where ar: a < r and rb: r < b by blast
      from lemma-interval [OF ar rb]
      obtain d where d: 0<d and bound: ∀ y. |r-y| < d ⟶ a ≤ y ∧ y ≤ b
  by blast
      have eq-fb: ∀ z. a ≤ z --> z ≤ b --> f z = f b
      proof (clarify)
        fix z::real
        assume az: a ≤ z and zb: z ≤ b
        show f z = f b
        proof (rule order-antisym)
          show f z ≤ f b by (simp add: fb-eq-fx x-max az zb)
          show f b ≤ f z by (simp add: fb-eq-fx' x'-min az zb)
        qed
      qed
      have bound': ∀ y. |r-y| < d ⟶ f r = f y
      proof (intro strip)
        fix y::real
        assume lt: |r-y| < d
        hence f y = f b by (simp add: eq-fb bound)
        thus f r = f y by (simp add: eq-fb ar rb order-less-imp-le)
      qed
      from differentiableD [OF dif [OF conjI [OF ar rb]]]
      obtain l where der: DERIV f r :> l ..
      have l=0 by (rule DERIV-local-const [OF der d bound'])
        — the derivative of a constant function is zero
      thus ?thesis using ar rb der by auto
    qed
  qed
qed
```

20.5 Mean Value Theorem

```
lemma lemma-MVT:
    f a - (f b - f a)/(b-a) * a = f b - (f b - f a)/(b-a) * (b::real)
proof cases
  assume a=b thus ?thesis by simp
next
  assume a≠b
  hence ba: b-a ≠ 0 by arith
  show ?thesis
    by (rule real-mult-left-cancel [OF ba, THEN iffD1],
        simp add: right-diff-distrib,
        simp add: left-diff-distrib)
```

qed

theorem *MVT*:
assumes *lt*: $a < b$
and *con*: $\forall x.\ a \le x\ \&\ x \le b \dashrightarrow isCont\ f\ x$
and *dif* [*rule-format*]: $\forall x.\ a < x\ \&\ x < b \dashrightarrow f\ differentiable\ x$
shows $\exists l\ z{::}real.\ a < z\ \&\ z < b\ \&\ DERIV\ f\ z :> l\ \&$
$(f(b) - f(a) = (b-a) * l)$
proof −
let *?F* = $\%x.\ f\ x - ((f\ b - f\ a)\ /\ (b-a)) * x$
have *contF*: $\forall x.\ a \le x \wedge x \le b \longrightarrow isCont\ ?F\ x$ **using** *con*
by (*fast intro*: *isCont-diff isCont-const isCont-mult isCont-ident*)
have *difF*: $\forall x.\ a < x \wedge x < b \longrightarrow ?F\ differentiable\ x$
proof (*clarify*)
fix *x*::*real*
assume *ax*: $a < x$ **and** *xb*: $x < b$
from *differentiableD* [*OF dif* [*OF conjI* [*OF ax xb*]]]
obtain *l* **where** *der*: $DERIV\ f\ x :> l$ **..**
show *?F differentiable x*
by (*rule differentiableI* [**where** $D = l - (f\ b - f\ a)/(b-a)$],
blast intro: *DERIV-diff DERIV-cmult-Id der*)
qed
from *Rolle* [**where** *f* = *?F*, *OF lt lemma-MVT contF difF*]
obtain *z* **where** *az*: $a < z$ **and** *zb*: $z < b$ **and** *der*: $DERIV\ ?F\ z :> 0$
by *blast*
have $DERIV\ (\%x.\ ((f\ b - f\ a)/(b-a)) * x)\ z :> (f\ b - f\ a)/(b-a)$
by (*rule DERIV-cmult-Id*)
hence *derF*: $DERIV\ (\lambda x.\ ?F\ x + (f\ b - f\ a)\ /\ (b - a) * x)\ z$
$:> 0 + (f\ b - f\ a)\ /\ (b - a)$
by (*rule DERIV-add* [*OF der*])
show *?thesis*
proof (*intro exI conjI*)
show $a < z$ **using** *az* **.**
show $z < b$ **using** *zb* **.**
show $f\ b - f\ a = (b - a) * ((f\ b - f\ a)/(b-a))$ **by** (*simp*)
show $DERIV\ f\ z :> ((f\ b - f\ a)/(b-a))$ **using** *derF* **by** *simp*
qed
qed

A function is constant if its derivative is 0 over an interval.

lemma *DERIV-isconst-end*:
fixes *f* :: *real* => *real*
shows [| $a < b$;
$\forall x.\ a \le x\ \&\ x \le b \dashrightarrow isCont\ f\ x$;
$\forall x.\ a < x\ \&\ x < b \dashrightarrow DERIV\ f\ x :> 0$ |]
==> $f\ b = f\ a$
apply (*drule MVT*, *assumption*)
apply (*blast intro*: *differentiableI*)
apply (*auto dest*!: *DERIV-unique simp add*: *diff-eq-eq*)

done

lemma *DERIV-isconst1*:
fixes f :: *real* => *real*
shows [| $a < b$;
$\forall x.\ a \leq x \ \&\ x \leq b$ --> *isCont* $f\ x$;
$\forall x.\ a < x \ \&\ x < b$ --> *DERIV* $f\ x$:> 0 |]
==> $\forall x.\ a \leq x \ \&\ x \leq b$ --> $f\ x = f\ a$
apply *safe*
apply (*drule-tac* $x = a$ **in** *order-le-imp-less-or-eq*, *safe*)
apply (*drule-tac* $b = x$ **in** *DERIV-isconst-end*, *auto*)
done

lemma *DERIV-isconst2*:
fixes f :: *real* => *real*
shows [| $a < b$;
$\forall x.\ a \leq x \ \&\ x \leq b$ --> *isCont* $f\ x$;
$\forall x.\ a < x \ \&\ x < b$ --> *DERIV* $f\ x$:> 0;
$a \leq x$; $x \leq b$ |]
==> $f\ x = f\ a$
apply (*blast dest*: *DERIV-isconst1*)
done

lemma *DERIV-isconst-all*:
fixes f :: *real* => *real*
shows $\forall x.$ *DERIV* $f\ x$:> 0 ==> $f(x) = f(y)$
apply (*rule linorder-cases* [*of* x y])
apply (*blast intro*: *sym DERIV-isCont DERIV-isconst-end*)+
done

lemma *DERIV-const-ratio-const*:
fixes f :: *real* => *real*
shows [|$a \neq b$; $\forall x.$ *DERIV* $f\ x$:> k |] ==> $(f(b) - f(a)) = (b-a) * k$
apply (*rule linorder-cases* [*of* a b], *auto*)
apply (*drule-tac* [!] $f = f$ **in** *MVT*)
apply (*auto dest*: *DERIV-isCont DERIV-unique simp add*: *differentiable-def*)
apply (*auto dest*: *DERIV-unique simp add*: *ring-distribs diff-minus*)
done

lemma *DERIV-const-ratio-const2*:
fixes f :: *real* => *real*
shows [|$a \neq b$; $\forall x.$ *DERIV* $f\ x$:> k |] ==> $(f(b) - f(a))/(b-a) = k$
apply (*rule-tac* $c1 = b-a$ **in** *real-mult-right-cancel* [*THEN iffD1*])
apply (*auto dest*!: *DERIV-const-ratio-const simp add*: *mult-assoc*)
done

lemma *real-average-minus-first* [*simp*]: $((a + b)\ /2 - a) = (b-a)/(2::real)$
by (*simp*)

lemma *real-average-minus-second* [*simp*]: $((b + a)/2 - a) = (b-a)/(2::real)$
by (*simp*)

Gallileo's "trick": average velocity = av. of end velocities

lemma *DERIV-const-average*:
  fixes $v :: real \Rightarrow real$
  assumes *neq*: $a \neq (b::real)$
    and *der*: $\forall x.\ DERIV\ v\ x :> k$
  shows $v\ ((a + b)/2) = (v\ a + v\ b)/2$
proof (*cases rule*: *linorder-cases* [*of a b*])
  case *equal* **with** *neq* **show** *?thesis* **by** *simp*
next
  case *less*
  have $(v\ b - v\ a)\ /\ (b - a) = k$
    by (*rule DERIV-const-ratio-const2* [*OF neq der*])
  hence $(b-a) * ((v\ b - v\ a)\ /\ (b-a)) = (b-a) * k$ **by** *simp*
  moreover have $(v\ ((a + b)\ /\ 2) - v\ a)\ /\ ((a + b)\ /\ 2 - a) = k$
    by (*rule DERIV-const-ratio-const2* [*OF - der*], *simp add*: *neq*)
  ultimately show *?thesis* **using** *neq* **by** *force*
next
  case *greater*
  have $(v\ b - v\ a)\ /\ (b - a) = k$
    by (*rule DERIV-const-ratio-const2* [*OF neq der*])
  hence $(b-a) * ((v\ b - v\ a)\ /\ (b-a)) = (b-a) * k$ **by** *simp*
  moreover have $(v\ ((b + a)\ /\ 2) - v\ a)\ /\ ((b + a)\ /\ 2 - a) = k$
    by (*rule DERIV-const-ratio-const2* [*OF - der*], *simp add*: *neq*)
  ultimately show *?thesis* **using** *neq* **by** (*force simp add*: *add-commute*)
qed

Dull lemma: an continuous injection on an interval must have a strict maximum at an end point, not in the middle.

lemma *lemma-isCont-inj*:
  fixes $f :: real \Rightarrow real$
  assumes *d*: $0 < d$
    and *inj* [*rule-format*]: $\forall z.\ |z-x| \leq d \longrightarrow g(f\ z) = z$
    and *cont*: $\forall z.\ |z-x| \leq d \longrightarrow isCont\ f\ z$
  shows $\exists z.\ |z-x| \leq d\ \&\ f\ x < f\ z$
proof (*rule ccontr*)
  assume $\sim (\exists z.\ |z-x| \leq d\ \&\ f\ x < f\ z)$
  hence *all* [*rule-format*]: $\forall z.\ |z - x| \leq d \longrightarrow f\ z \leq f\ x$ **by** *auto*
  show *False*
  proof (*cases rule*: *linorder-le-cases* [*of* $f(x-d)$ $f(x+d)$])
    case *le*
    from *d cont all* [*of* $x+d$]
    have *flef*: $f(x+d) \leq f\ x$
     and *xlex*: $x - d \leq x$
     and *cont'*: $\forall z.\ x - d \leq z \wedge z \leq x \longrightarrow isCont\ f\ z$
      by (*auto simp add*: *abs-if*)
    from *IVT* [*OF le flef xlex cont'*]

```
    obtain x' where x−d ≤ x' x' ≤ x f x' = f(x+d) by blast
    moreover
    hence g(f x') = g (f(x+d)) by simp
    ultimately show False using d inj [of x'] inj [of x+d]
      by (simp add: abs-le-iff)
  next
    case ge
    from d cont all [of x−d]
    have flef: f(x−d) ≤ f x
     and xlex: x ≤ x+d
     and cont': ∀ z. x ≤ z ∧ z ≤ x+d ⟶ isCont f z
       by (auto simp add: abs-if)
    from IVT2 [OF ge flef xlex cont']
    obtain x' where x ≤ x' x' ≤ x+d f x' = f(x−d) by blast
    moreover
    hence g(f x') = g (f(x−d)) by simp
    ultimately show False using d inj [of x'] inj [of x−d]
      by (simp add: abs-le-iff)
  qed
qed
```

Similar version for lower bound.

```
lemma lemma-isCont-inj2:
  fixes f g :: real ⇒ real
  shows [| 0 < d; ∀ z. |z−x| ≤ d −−> g(f z) = z;
        ∀ z. |z−x| ≤ d −−> isCont f z |]
      ==> ∃ z. |z−x| ≤ d & f z < f x
apply (insert lemma-isCont-inj
         [where f = %x. − f x and g = %y. g(−y) and x = x and d = d])
apply (simp add: isCont-minus linorder-not-le)
done
```

Show there's an interval surrounding $f\ x$ in $f[[x - d, x + d]]$.

```
lemma isCont-inj-range:
  fixes f :: real ⇒ real
  assumes d: 0 < d
      and inj: ∀ z. |z−x| ≤ d −−> g(f z) = z
      and cont: ∀ z. |z−x| ≤ d −−> isCont f z
  shows ∃ e>0. ∀ y. |y − f x| ≤ e −−> (∃ z. |z−x| ≤ d & f z = y)
proof −
  have x−d ≤ x+d ∀ z. x−d ≤ z ∧ z ≤ x+d ⟶ isCont f z using cont d
    by (auto simp add: abs-le-iff)
  from isCont-Lb-Ub [OF this]
  obtain L M
  where all1 [rule-format]: ∀ z. x−d ≤ z ∧ z ≤ x+d ⟶ L ≤ f z ∧ f z ≤ M
    and all2 [rule-format]:
          ∀ y. L ≤ y ∧ y ≤ M ⟶ (∃ z. x−d ≤ z ∧ z ≤ x+d ∧ f z = y)
    by auto
  with d have L ≤ f x & f x ≤ M by simp
```

```
  moreover have L ≠ f x
  proof −
    from lemma-isCont-inj2 [OF d inj cont]
    obtain u where |u − x| ≤ d f u < f x  by auto
    thus ?thesis using all1 [of u] by arith
  qed
  moreover have f x ≠ M
  proof −
    from lemma-isCont-inj [OF d inj cont]
    obtain u where |u − x| ≤ d f x < f u  by auto
    thus ?thesis using all1 [of u] by arith
  qed
  ultimately have L < f x & f x < M by arith
  hence 0 < f x − L 0 < M − f x by arith+
  from real-lbound-gt-zero [OF this]
  obtain e where e: 0 < e e < f x − L e < M − f x by auto
  thus ?thesis
  proof (intro exI conjI)
    show 0<e using e(1) .
    show ∀ y. |y − f x| ≤ e ⟶ (∃ z. |z − x| ≤ d ∧ f z = y)
    proof (intro strip)
      fix y::real
      assume |y − f x| ≤ e
      with e have L ≤ y ∧ y ≤ M by arith
      from all2 [OF this]
      obtain z where x − d ≤ z z ≤ x + d f z = y by blast
      thus ∃ z. |z − x| ≤ d ∧ f z = y
        by (force simp add: abs-le-iff)
    qed
  qed
qed
```

Continuity of inverse function

```
lemma isCont-inverse-function:
  fixes f g :: real ⇒ real
  assumes d: 0 < d
      and inj: ∀ z. |z−x| ≤ d −−> g(f z) = z
      and cont: ∀ z. |z−x| ≤ d −−> isCont f z
  shows isCont g (f x)
proof (simp add: isCont-iff LIM-eq)
  show ∀ r. 0 < r ⟶
        (∃ s>0. ∀ z. z≠0 ∧ |z| < s ⟶ |g(f x + z) − g(f x)| < r)
  proof (intro strip)
    fix r::real
    assume r: 0<r
    from real-lbound-gt-zero [OF r d]
    obtain e where e: 0 < e and e-lt: e < r ∧ e < d by blast
    with inj cont
    have e-simps: ∀ z. |z−x| ≤ e −−> g (f z) = z
```

$\forall z.\ |z-x| \leq e \dashrightarrow isCont\ f\ z$ **by** *auto*
from *isCont-inj-range* [*OF e this*]
obtain e' **where** e': $0 < e'$
and *all*: $\forall y.\ |y - f\ x| \leq e' \longrightarrow (\exists z.\ |z - x| \leq e \wedge f\ z = y)$
by *blast*
show $\exists s{>}0.\ \forall z.\ z{\neq}0 \wedge |z| < s \longrightarrow |g(f\ x + z) - g(f\ x)| < r$
proof (*intro exI conjI*)
show $0{<}e'$ **using** e' **.**
show $\forall z.\ z \neq 0 \wedge |z| < e' \longrightarrow |g\ (f\ x + z) - g\ (f\ x)| < r$
proof (*intro strip*)
fix *z*::*real*
assume z: $z \neq 0 \wedge |z| < e'$
with *e e-lt e-simps all* [*rule-format*, *of f x* + *z*]
show $|g\ (f\ x + z) - g\ (f\ x)| < r$ **by** *force*
qed
qed
qed
qed

Derivative of inverse function

lemma *DERIV-inverse-function*:
fixes $f\ g$ :: $real \Rightarrow real$
assumes *der*: $DERIV\ f\ (g\ x) :> D$
assumes *neq*: $D \neq 0$
assumes a: $a < x$ **and** b: $x < b$
assumes *inj*: $\forall y.\ a < y \wedge y < b \longrightarrow f\ (g\ y) = y$
assumes *cont*: $isCont\ g\ x$
shows $DERIV\ g\ x :> inverse\ D$
unfolding *DERIV-iff2*
proof (*rule LIM-equal2*)
show $0 < min\ (x - a)\ (b - x)$
using a b **by** *simp*
next
fix y
assume $norm\ (y - x) < min\ (x - a)\ (b - x)$
hence $a < y$ **and** $y < b$
by (*simp-all add*: *abs-less-iff*)
thus $(g\ y - g\ x)\ /\ (y - x) =$
$inverse\ ((f\ (g\ y) - x)\ /\ (g\ y - g\ x))$
by (*simp add*: *inj*)
next
have $(\lambda z.\ (f\ z - f\ (g\ x))\ /\ (z - g\ x)) -- g\ x \dashrightarrow D$
by (*rule der* [*unfolded DERIV-iff2*])
hence *1*: $(\lambda z.\ (f\ z - x)\ /\ (z - g\ x)) -- g\ x \dashrightarrow D$
using *inj a b* **by** *simp*
have *2*: $\exists d{>}0.\ \forall y.\ y \neq x \wedge norm\ (y - x) < d \longrightarrow g\ y \neq g\ x$
proof (*safe intro*!: *exI*)
show $0 < min\ (x - a)\ (b - x)$
using a b **by** *simp*

```
  next
    fix y
    assume norm (y − x) < min (x − a) (b − x)
    hence y: a < y y < b
      by (simp-all add: abs-less-iff)
    assume g y = g x
    hence f (g y) = f (g x) by simp
    hence y = x using inj y a b by simp
    also assume y ≠ x
    finally show False by simp
  qed
  have (λy. (f (g y) − x) / (g y − g x)) −− x −−> D
    using cont 1 2 by (rule isCont-LIM-compose2)
  thus (λy. inverse ((f (g y) − x) / (g y − g x)))
       −− x −−> inverse D
    using neq by (rule LIM-inverse)
qed

theorem GMVT:
  fixes a b :: real
  assumes alb: a < b
  and fc: ∀x. a ≤ x ∧ x ≤ b ⟶ isCont f x
  and fd: ∀x. a < x ∧ x < b ⟶ f differentiable x
  and gc: ∀x. a ≤ x ∧ x ≤ b ⟶ isCont g x
  and gd: ∀x. a < x ∧ x < b ⟶ g differentiable x
  shows ∃g'c f'c c. DERIV g c :> g'c ∧ DERIV f c :> f'c ∧ a < c ∧ c < b ∧
((f b − f a) ∗ g'c) = ((g b − g a) ∗ f'c)
proof −
  let ?h = λx. (f b − f a)∗(g x) − (g b − g a)∗(f x)
  from prems have a < b by simp
  moreover have ∀x. a ≤ x ∧ x ≤ b ⟶ isCont ?h x
  proof −
    have ∀x. a <= x ∧ x <= b ⟶ isCont (λx. f b − f a) x by simp
    with gc have ∀x. a <= x ∧ x <= b ⟶ isCont (λx. (f b − f a) ∗ g x) x
      by (auto intro: isCont-mult)
    moreover
    have ∀x. a <= x ∧ x <= b ⟶ isCont (λx. g b − g a) x by simp
    with fc have ∀x. a <= x ∧ x <= b ⟶ isCont (λx. (g b − g a) ∗ f x) x
      by (auto intro: isCont-mult)
    ultimately show ?thesis
      by (fastsimp intro: isCont-diff)
  qed
  moreover
  have ∀x. a < x ∧ x < b ⟶ ?h differentiable x
  proof −
    have ∀x. a < x ∧ x < b ⟶ (λx. f b − f a) differentiable x by (simp add:
differentiable-const)
    with gd have ∀x. a < x ∧ x < b ⟶ (λx. (f b − f a) ∗ g x) differentiable x
by (simp add: differentiable-mult)
```

moreover
have $\forall x.\ a < x \wedge x < b \longrightarrow (\lambda x.\ g\ b - g\ a)$ *differentiable* x **by** (*simp add*: *differentiable-const*)
with *fd* **have** $\forall x.\ a < x \wedge x < b \longrightarrow (\lambda x.\ (g\ b - g\ a) * f\ x)$ *differentiable* x **by** (*simp add*: *differentiable-mult*)
ultimately show *?thesis* **by** (*simp add*: *differentiable-diff*)
qed
ultimately have $\exists l\ z.\ a < z \wedge z < b \wedge DERIV\ ?h\ z :> l \wedge ?h\ b - ?h\ a = (b - a) * l$ **by** (*rule MVT*)
then obtain l **where** *ldef*: $\exists z.\ a < z \wedge z < b \wedge DERIV\ ?h\ z :> l \wedge ?h\ b - ?h\ a = (b - a) * l$ **..**
then obtain c **where** *cdef*: $a < c \wedge c < b \wedge DERIV\ ?h\ c :> l \wedge ?h\ b - ?h\ a = (b - a) * l$ **..**

from *cdef* **have** *cint*: $a < c \wedge c < b$ **by** *auto*
with *gd* **have** *g differentiable c* **by** *simp*
hence $\exists D.\ DERIV\ g\ c :> D$ **by** (*rule differentiableD*)
then obtain $g'c$ **where** $g'cdef$: $DERIV\ g\ c :> g'c$ **..**

from *cdef* **have** $a < c \wedge c < b$ **by** *auto*
with *fd* **have** *f differentiable c* **by** *simp*
hence $\exists D.\ DERIV\ f\ c :> D$ **by** (*rule differentiableD*)
then obtain $f'c$ **where** $f'cdef$: $DERIV\ f\ c :> f'c$ **..**

from *cdef* **have** $DERIV\ ?h\ c :> l$ **by** *auto*
moreover
{
have $DERIV\ (\lambda x.\ (f\ b - f\ a) * g\ x)\ c :> g'c * (f\ b - f\ a)$
apply (*insert DERIV-const* [**where** $k=f\ b - f\ a$])
apply (*drule meta-spec* [*of* - c])
apply (*drule DERIV-mult* [*OF* - $g'cdef$])
by *simp*
moreover have $DERIV\ (\lambda x.\ (g\ b - g\ a) * f\ x)\ c :> f'c * (g\ b - g\ a)$
apply (*insert DERIV-const* [**where** $k=g\ b - g\ a$])
apply (*drule meta-spec* [*of* - c])
apply (*drule DERIV-mult* [*OF* - $f'cdef$])
by *simp*
ultimately have $DERIV\ ?h\ c :> g'c * (f\ b - f\ a) - f'c * (g\ b - g\ a)$
by (*simp add*: *DERIV-diff*)
}
ultimately have *leq*: $l = g'c * (f\ b - f\ a) - f'c * (g\ b - g\ a)$ **by** (*rule DERIV-unique*)

{
from *cdef* **have** $?h\ b - ?h\ a = (b - a) * l$ **by** *auto*
also with *leq* **have** $\ldots = (b - a) * (g'c * (f\ b - f\ a) - f'c * (g\ b - g\ a))$ **by** *simp*
finally have $?h\ b - ?h\ a = (b - a) * (g'c * (f\ b - f\ a) - f'c * (g\ b - g\ a))$ **by** *simp*

```
  }
  moreover
  {
    have ?h b − ?h a =
        ((f b)∗(g b) − (f a)∗(g b) − (g b)∗(f b) + (g a)∗(f b)) −
         ((f b)∗(g a) − (f a)∗(g a) − (g b)∗(f a) + (g a)∗(f a))
      by (simp add: mult-ac add-ac right-diff-distrib)
    hence ?h b − ?h a = 0 by auto
  }
  ultimately have (b − a) ∗ (g′c ∗ (f b − f a) − f′c ∗ (g b − g a)) = 0 by auto
  with alb have g′c ∗ (f b − f a) − f′c ∗ (g b − g a) = 0 by simp
  hence g′c ∗ (f b − f a) = f′c ∗ (g b − g a) by simp
  hence (f b − f a) ∗ g′c = (g b − g a) ∗ f′c by (simp add: mult-ac)

  with g′cdef f′cdef cint show ?thesis by auto
qed

lemma lemma-DERIV-subst: [| DERIV f x :> D; D = E |] ==> DERIV f x :>
E
by auto
```

20.6 Derivatives of univariate polynomials

```
primrec pderiv-aux :: nat => real list => real list where
   pderiv-aux-Nil:  pderiv-aux n [] = []
|  pderiv-aux-Cons: pderiv-aux n (h#t) =
                    (real n ∗ h)#(pderiv-aux (Suc n) t)

definition
  pderiv :: real list => real list where
  pderiv p = (if p = [] then [] else pderiv-aux 1 (tl p))
```

The derivative

```
lemma pderiv-Nil: pderiv [] = []

apply (simp add: pderiv-def)
done
declare pderiv-Nil [simp]

lemma pderiv-singleton: pderiv [c] = []
by (simp add: pderiv-def)
declare pderiv-singleton [simp]

lemma pderiv-Cons: pderiv (h#t) = pderiv-aux 1 t
by (simp add: pderiv-def)

lemma DERIV-cmult2: DERIV f x :> D ==> DERIV (%x. (f x) ∗ c :: real) x
:> D ∗ c
by (simp add: DERIV-cmult mult-commute [of - c])
```

lemma *DERIV-pow2*: *DERIV* (%*x*. *x* ^ *Suc n*) *x* :> *real* (*Suc n*) * (*x* ^ *n*)
by (*rule lemma-DERIV-subst*, *rule DERIV-pow*, *simp*)
declare *DERIV-pow2* [*simp*] *DERIV-pow* [*simp*]

lemma *lemma-DERIV-poly1*: ∀ *n*. *DERIV* (%*x*. (*x* ^ (*Suc n*) * *poly p x*)) *x* :>
    x ^ *n* * *poly* (*pderiv-aux* (*Suc n*) *p*) *x*
apply (*induct p*)
apply (*auto intro*!: *DERIV-add DERIV-cmult2*
      simp add: *pderiv-def right-distrib real-mult-assoc* [*symmetric*]
      simp del: *realpow-Suc*)
apply (*subst mult-commute*)
apply (*simp del*: *realpow-Suc*)
apply (*simp add*: *mult-commute realpow-Suc* [*symmetric*] *del*: *realpow-Suc*)
done

lemma *lemma-DERIV-poly*: *DERIV* (%*x*. (*x* ^ (*Suc n*) * *poly p x*)) *x* :>
    x ^ *n* * *poly* (*pderiv-aux* (*Suc n*) *p*) *x*
by (*simp add*: *lemma-DERIV-poly1 del*: *realpow-Suc*)

lemma *DERIV-add-const*: *DERIV f x* :> *D* ==> *DERIV* (%*x*. *a* + *f x* :: *real*)
x :> *D*
by (*rule lemma-DERIV-subst*, *rule DERIV-add*, *auto*)

lemma *poly-DERIV*[*simp*]: *DERIV* (%*x*. *poly p x*) *x* :> *poly* (*pderiv p*) *x*
apply (*induct p*)
apply (*auto simp add*: *pderiv-Cons*)
apply (*rule DERIV-add-const*)
apply (*rule lemma-DERIV-subst*)
apply (*rule lemma-DERIV-poly* [**where** *n*=*0*, *simplified*], *simp*)
done

Consequences of the derivative theorem above

lemma *poly-differentiable*[*simp*]: (%*x*. *poly p x*) *differentiable* (*x*::*real*)
apply (*simp add*: *differentiable-def*)
apply (*blast intro*: *poly-DERIV*)
done

lemma *poly-isCont*[*simp*]: *isCont* (%*x*. *poly p x*) (*x*::*real*)
by (*rule poly-DERIV* [*THEN DERIV-isCont*])

lemma *poly-IVT-pos*: [| *a* < *b*; *poly p* (*a*::*real*) < *0*; *0* < *poly p b* |]
    ==> ∃ *x*. *a* < *x* & *x* < *b* & (*poly p x* = *0*)
apply (*cut-tac f* = %*x*. *poly p x* **and** *a* = *a* **and** *b* = *b* **and** *y* = *0* **in** *IVT-objl*)
apply (*auto simp add*: *order-le-less*)
done

lemma *poly-IVT-neg*: [| (*a*::*real*) < *b*; *0* < *poly p a*; *poly p b* < *0* |]
    ==> ∃ *x*. *a* < *x* & *x* < *b* & (*poly p x* = *0*)

apply (*insert poly-IVT-pos* [**where** $p = -- p$])
apply (*simp add*: *poly-minus neg-less-0-iff-less*)
done

lemma *poly-MVT*: $a < b ==>$
$\exists x.\ a < x\ \&\ x < b\ \&\ (poly\ p\ b - poly\ p\ a = (b - a) * poly\ (pderiv\ p)\ x)$
apply (*drule-tac* $f = poly\ p$ **in** *MVT*, *auto*)
apply (*rule-tac* $x = z$ **in** *exI*)
apply (*auto simp add*: *real-mult-left-cancel poly-DERIV* [*THEN DERIV-unique*])
done

Lemmas for Derivatives

lemma *lemma-poly-pderiv-aux-add*: $\forall p2\ n.\ poly\ (pderiv\text{-}aux\ n\ (p1 +++ p2))\ x =$
$poly\ (pderiv\text{-}aux\ n\ p1 +++ pderiv\text{-}aux\ n\ p2)\ x$
apply (*induct p1*, *simp*, *clarify*)
apply (*case-tac p2*)
apply (*auto simp add*: *right-distrib*)
done

lemma *poly-pderiv-aux-add*: $poly\ (pderiv\text{-}aux\ n\ (p1 +++ p2))\ x =$
$poly\ (pderiv\text{-}aux\ n\ p1 +++ pderiv\text{-}aux\ n\ p2)\ x$
apply (*simp add*: *lemma-poly-pderiv-aux-add*)
done

lemma *lemma-poly-pderiv-aux-cmult*: $\forall n.\ poly\ (pderiv\text{-}aux\ n\ (c\ \%*\ p))\ x = poly\ (c\ \%*\ pderiv\text{-}aux\ n\ p)\ x$
apply (*induct p*)
apply (*auto simp add*: *poly-cmult mult-ac*)
done

lemma *poly-pderiv-aux-cmult*: $poly\ (pderiv\text{-}aux\ n\ (c\ \%*\ p))\ x = poly\ (c\ \%*\ pderiv\text{-}aux\ n\ p)\ x$
by (*simp add*: *lemma-poly-pderiv-aux-cmult*)

lemma *poly-pderiv-aux-minus*:
$poly\ (pderiv\text{-}aux\ n\ (-- p))\ x = poly\ (-- pderiv\text{-}aux\ n\ p)\ x$
apply (*simp add*: *poly-minus-def poly-pderiv-aux-cmult*)
done

lemma *lemma-poly-pderiv-aux-mult1*: $\forall n.\ poly\ (pderiv\text{-}aux\ (Suc\ n)\ p)\ x = poly\ ((pderiv\text{-}aux\ n\ p) +++ p)\ x$
apply (*induct p*)
apply (*auto simp add*: *real-of-nat-Suc left-distrib*)
done

lemma *lemma-poly-pderiv-aux-mult*: $poly\ (pderiv\text{-}aux\ (Suc\ n)\ p)\ x = poly\ ((pderiv\text{-}aux\ n\ p) +++ p)\ x$
by (*simp add*: *lemma-poly-pderiv-aux-mult1*)

lemma *lemma-poly-pderiv-add*: $\forall q.\ poly\ (pderiv\ (p\ +{+}{+}\ q))\ x = poly\ (pderiv\ p\ +{+}{+}\ pderiv\ q)\ x$
apply (*induct p*, *simp*, *clarify*)
apply (*case-tac q*)
apply (*auto simp add*: *poly-pderiv-aux-add poly-add pderiv-def*)
done

lemma *poly-pderiv-add*: *poly* (*pderiv* (p +++ q)) x = *poly* (*pderiv p* +++ *pderiv* q) x
by (*simp add*: *lemma-poly-pderiv-add*)

lemma *poly-pderiv-cmult*: *poly* (*pderiv* (c %* p)) x = *poly* (c %* (*pderiv p*)) x
apply (*induct p*)
apply (*auto simp add*: *poly-pderiv-aux-cmult poly-cmult pderiv-def*)
done

lemma *poly-pderiv-minus*: *poly* (*pderiv* (−−p)) x = *poly* (−−(*pderiv p*)) x
by (*simp add*: *poly-minus-def poly-pderiv-cmult*)

lemma *lemma-poly-mult-pderiv*:
  poly (*pderiv* (h#t)) x = *poly* ((*0* # (*pderiv t*)) +++ t) x
apply (*simp add*: *pderiv-def*)
apply (*induct t*)
apply (*auto simp add*: *poly-add lemma-poly-pderiv-aux-mult*)
done

lemma *poly-pderiv-mult*: $\forall q.\ poly\ (pderiv\ (p\ *{*}{*}\ q))\ x =$
    poly (p *** (*pderiv q*) +++ q *** (*pderiv p*)) x
apply (*induct p*)
apply (*auto simp add*: *poly-add poly-cmult poly-pderiv-cmult poly-pderiv-add poly-mult*)
apply (*rule lemma-poly-mult-pderiv* [*THEN ssubst*])
apply (*rule lemma-poly-mult-pderiv* [*THEN ssubst*])
apply (*rule poly-add* [*THEN ssubst*])
apply (*rule poly-add* [*THEN ssubst*])
apply (*simp* (*no-asm-simp*) *add*: *poly-mult right-distrib add-ac mult-ac*)
done

lemma *poly-pderiv-exp*: *poly* (*pderiv* (p %^ (*Suc n*))) x =
    poly ((*real* (*Suc n*)) %* (p %^ n) *** *pderiv p*) x
apply (*induct n*)
apply (*auto simp add*: *poly-add poly-pderiv-cmult poly-cmult poly-pderiv-mult*
        real-of-nat-zero poly-mult real-of-nat-Suc
        right-distrib left-distrib mult-ac)
done

lemma *poly-pderiv-exp-prime*: *poly* (*pderiv* ([−a, *1*] %^ (*Suc n*))) x =
    poly (*real* (*Suc n*) %* ([−a, *1*] %^ n)) x
apply (*simp add*: *poly-pderiv-exp poly-mult del*: *pexp-Suc*)
apply (*simp add*: *poly-cmult pderiv-def*)

done

lemma *real-mult-zero-disj-iff*[*simp*]: $(x * y = 0) = (x = (0::real) \mid y = 0)$
by *simp*

lemma *pderiv-aux-iszero* [*rule-format*, *simp*]:
$\forall n.$ *list-all* $(\%c.\ c = 0)$ (*pderiv-aux* (*Suc n*) *p*) = *list-all* $(\%c.\ c = 0)$ *p*
by (*induct p*, *auto*)

lemma *pderiv-aux-iszero-num*: (*number-of n* :: *nat*) $\neq$ *0*
==> (*list-all* $(\%c.\ c = 0)$ (*pderiv-aux* (*number-of n*) *p*) =
list-all $(\%c.\ c = 0)$ *p*)
unfolding *neq0-conv*
apply (*rule-tac n1* = *number-of n* **and** *m1* = *0* **in** *less-imp-Suc-add* [*THEN exE*], *force*)
apply (*rule-tac n1* = *0* + *x* **in** *pderiv-aux-iszero* [*THEN subst*])
apply (*simp* (*no-asm-simp*) *del*: *pderiv-aux-iszero*)
done

instance *real*:: *idom-char-0*
apply (*intro-classes*)
done

instance *real*:: *recpower-idom-char-0*
apply (*intro-classes*)
done

lemma *pderiv-iszero* [*rule-format*]:
poly (*pderiv p*) = *poly* [] --> $(\exists h.$ *poly p* = *poly* [*h*])
apply (*simp add*: *poly-zero*)
apply (*induct p*, *force*)
apply (*simp add*: *pderiv-Cons pderiv-aux-iszero-num del*: *poly-Cons*)
apply (*auto simp add*: *poly-zero* [*symmetric*])
done

lemma *pderiv-zero-obj*: *poly p* = *poly* [] --> (*poly* (*pderiv p*) = *poly* [])
apply (*simp add*: *poly-zero*)
apply (*induct p*, *force*)
apply (*simp add*: *pderiv-Cons pderiv-aux-iszero-num del*: *poly-Cons*)
done

lemma *pderiv-zero*: *poly p* = *poly* [] ==> (*poly* (*pderiv p*) = *poly* [])
by (*blast elim*: *pderiv-zero-obj* [*THEN impE*])
declare *pderiv-zero* [*simp*]

lemma *poly-pderiv-welldef*: *poly p* = *poly q* ==> (*poly* (*pderiv p*) = *poly* (*pderiv q*))
apply (*cut-tac p* = *p* +++ --*q* **in** *pderiv-zero-obj*)

apply (*simp add*: *fun-eq poly-add poly-minus poly-pderiv-add poly-pderiv-minus del*: *pderiv-zero*)
done

lemma *lemma-order-pderiv* [*rule-format*]:
∀ *p q a*. *0 < n* &
poly (*pderiv p*) ≠ *poly* [] &
poly p = *poly* ([− *a*, *1*] %^ *n* *** *q*) & ~ [− *a*, *1*] *divides q*
−−> *n* = *Suc* (*order a* (*pderiv p*))
apply (*induct n*, *safe*)
apply (*rule order-unique-lemma*, *rule conjI*, *assumption*)
apply (*subgoal-tac* ∀ *r*. *r divides* (*pderiv p*) = *r divides* (*pderiv* ([−*a*, *1*] %^ *Suc n* *** *q*)))
apply (*drule-tac* [*2*] *poly-pderiv-welldef*)
prefer *2* **apply** (*simp add*: *divides-def del*: *pmult-Cons pexp-Suc*)
apply (*simp del*: *pmult-Cons pexp-Suc*)
apply (*rule conjI*)
apply (*simp add*: *divides-def fun-eq del*: *pmult-Cons pexp-Suc*)
apply (*rule-tac x* = [−*a*, *1*] *** (*pderiv q*) +++ *real* (*Suc n*) %* *q* **in** *exI*)
apply (*simp add*: *poly-pderiv-mult poly-pderiv-exp-prime poly-add poly-mult poly-cmult right-distrib mult-ac del*: *pmult-Cons pexp-Suc*)
apply (*simp add*: *poly-mult right-distrib left-distrib mult-ac del*: *pmult-Cons*)
apply (*erule-tac V* = ∀ *r*. *r divides pderiv p* = *r divides pderiv* ([− *a*, *1*] %^ *Suc n* *** *q*) **in** *thin-rl*)
apply (*unfold divides-def*)
apply (*simp* (*no-asm*) *add*: *poly-pderiv-mult poly-pderiv-exp-prime fun-eq poly-add poly-mult del*: *pmult-Cons pexp-Suc*)
apply (*rule contrapos-np*, *assumption*)
apply (*rotate-tac 3*, *erule contrapos-np*)
apply (*simp del*: *pmult-Cons pexp-Suc*, *safe*)
apply (*rule-tac x* = *inverse* (*real* (*Suc n*)) %* (*qa* +++ −− (*pderiv q*)) **in** *exI*)
apply (*subgoal-tac poly* ([−*a*, *1*] %^ *n* *** *q*) = *poly* ([−*a*, *1*] %^ *n* *** ([−*a*, *1*] *** (*inverse* (*real* (*Suc n*)) %* (*qa* +++ −− (*pderiv q*))))))
apply (*drule poly-mult-left-cancel* [*THEN iffD1*], *simp*)
apply (*simp add*: *fun-eq poly-mult poly-add poly-cmult poly-minus del*: *pmult-Cons mult-cancel-left*, *safe*)
apply (*rule-tac c1* = *real* (*Suc n*) **in** *real-mult-left-cancel* [*THEN iffD1*])
apply (*simp* (*no-asm*))
apply (*subgoal-tac real* (*Suc n*) ∗ (*poly* ([− *a*, *1*] %^ *n*) *xa* ∗ *poly q xa*) =
(*poly qa xa* + − *poly* (*pderiv q*) *xa*) ∗
(*poly* ([− *a*, *1*] %^ *n*) *xa* ∗
((− *a* + *xa*) ∗ (*inverse* (*real* (*Suc n*)) ∗ *real* (*Suc n*)))))
apply (*simp only*: *mult-ac*)
apply (*rotate-tac 2*)
apply (*drule-tac x* = *xa* **in** *spec*)
apply (*simp add*: *left-distrib mult-ac del*: *pmult-Cons*)
done

lemma *order-pderiv*: [| *poly* (*pderiv p*) ≠ *poly* []; *order a p* ≠ *0* |]

```
    ==> (order a p = Suc (order a (pderiv p)))
apply (case-tac poly p = poly [])
apply (auto dest: pderiv-zero)
apply (drule-tac a = a and p = p in order-decomp)
using neq0-conv
apply (blast intro: lemma-order-pderiv)
done
```

Now justify the standard squarefree decomposition, i.e. f / gcd(f,f').

```
lemma poly-squarefree-decomp-order: [| poly (pderiv p) ≠ poly [];
        poly p = poly (q *** d);
        poly (pderiv p) = poly (e *** d);
        poly d = poly (r *** p +++ s *** pderiv p)
    |] ==> order a q = (if order a p = 0 then 0 else 1)
apply (subgoal-tac order a p = order a q + order a d)
apply (rule-tac [2] s = order a (q *** d) in trans)
prefer 2 apply (blast intro: order-poly)
apply (rule-tac [2] order-mult)
 prefer 2 apply force
apply (case-tac order a p = 0, simp)
apply (subgoal-tac order a (pderiv p) = order a e + order a d)
apply (rule-tac [2] s = order a (e *** d) in trans)
prefer 2 apply (blast intro: order-poly)
apply (rule-tac [2] order-mult)
 prefer 2 apply force
apply (case-tac poly p = poly [])
apply (drule-tac p = p in pderiv-zero, simp)
apply (drule order-pderiv, assumption)
apply (subgoal-tac order a (pderiv p) ≤ order a d)
apply (subgoal-tac [2]  ([−a, 1] %^ (order a (pderiv p))) divides d)
 prefer 2 apply (simp add: poly-entire order-divides)
apply (subgoal-tac [2]  ([−a, 1] %^ (order a (pderiv p))) divides p & ([−a, 1] %^
(order a (pderiv p))) divides (pderiv p) )
 prefer 3 apply (simp add: order-divides)
 prefer 2 apply (simp add: divides-def del: pexp-Suc pmult-Cons, safe)
apply (rule-tac x = r *** qa +++ s *** qaa in exI)
apply (simp add: fun-eq poly-add poly-mult left-distrib right-distrib mult-ac del:
pexp-Suc pmult-Cons, auto)
done


lemma poly-squarefree-decomp-order2: [| poly (pderiv p) ≠ poly [];
        poly p = poly (q *** d);
        poly (pderiv p) = poly (e *** d);
        poly d = poly (r *** p +++ s *** pderiv p)
    |] ==> ∀ a. order a q = (if order a p = 0 then 0 else 1)
apply (blast intro: poly-squarefree-decomp-order)
done
```

lemma *order-pderiv2*: [| *poly* (*pderiv p*) ≠ *poly* []; *order a p* ≠ *0* |]
    ==> (*order a* (*pderiv p*) = *n*) = (*order a p* = *Suc n*)
apply (*auto dest*: *order-pderiv*)
done

lemma *rsquarefree-roots*:
  rsquarefree p = (∀ *a*. ~(*poly p a* = *0* & *poly* (*pderiv p*) *a* = *0*))
apply (*simp add*: *rsquarefree-def*)
apply (*case-tac poly p* = *poly* [], *simp*, *simp*)
apply (*case-tac poly* (*pderiv p*) = *poly* [])
apply *simp*
apply (*drule pderiv-iszero*, *clarify*)
apply (*subgoal-tac* ∀ *a*. *order a p* = *order a* [*h*])
apply (*simp add*: *fun-eq*)
apply (*rule allI*)
apply (*cut-tac p* = [*h*] **and** *a* = *a* **in** *order-root*)
apply (*simp add*: *fun-eq*)
apply (*blast intro*: *order-poly*)
apply (*auto simp add*: *order-root order-pderiv2*)
apply (*erule-tac x*=*a* **in** *allE*, *simp*)
done

lemma *poly-squarefree-decomp*: [| *poly* (*pderiv p*) ≠ *poly* [];
        poly p = *poly* (*q* *** *d*);
        poly (*pderiv p*) = *poly* (*e* *** *d*);
        poly d = *poly* (*r* *** *p* +++ *s* *** *pderiv p*)
      |] ==> *rsquarefree q* & (∀ *a*. (*poly q a* = *0*) = (*poly p a* = *0*))
apply (*frule poly-squarefree-decomp-order2*, *assumption*+)
apply (*case-tac poly p* = *poly* [])
apply (*blast dest*: *pderiv-zero*)
apply (*simp* (*no-asm*) *add*: *rsquarefree-def order-root del*: *pmult-Cons*)
apply (*simp add*: *poly-entire del*: *pmult-Cons*)
done

end

21 NthRoot: Nth Roots of Real Numbers

theory *NthRoot*
imports ~~/*src*/*HOL*/*Library*/*Parity* ../*Hyperreal*/*Deriv*
begin

21.1 Existence of Nth Root

Existence follows from the Intermediate Value Theorem

lemma *realpow-pos-nth*:
  assumes *n*: *0* < *n*

```
  assumes a: 0 < a
  shows ∃r>0. r ^ n = (a::real)
proof −
  have ∃r≥0. r ≤ (max 1 a) ∧ r ^ n = a
  proof (rule IVT)
    show 0 ^ n ≤ a using n a by (simp add: power-0-left)
    show 0 ≤ max 1 a by simp
    from n have n1: 1 ≤ n by simp
    have a ≤ max 1 a ^ 1 by simp
    also have max 1 a ^ 1 ≤ max 1 a ^ n
      using n1 by (rule power-increasing, simp)
    finally show a ≤ max 1 a ^ n .
    show ∀r. 0 ≤ r ∧ r ≤ max 1 a ⟶ isCont (λx. x ^ n) r
      by (simp add: isCont-power)
  qed
  then obtain r where r: 0 ≤ r ∧ r ^ n = a by fast
  with n a have r ≠ 0 by (auto simp add: power-0-left)
  with r have 0 < r ∧ r ^ n = a by simp
  thus ?thesis ..
qed


lemma realpow-pos-nth2: (0::real) < a ⟹ ∃r>0. r ^ Suc n = a
by (blast intro: realpow-pos-nth)
```

Uniqueness of nth positive root

```
lemma realpow-pos-nth-unique:
  ⟦0 < n; 0 < a⟧ ⟹ ∃!r. 0 < r ∧ r ^ n = (a::real)
apply (auto intro!: realpow-pos-nth)
apply (rule-tac n=n in power-eq-imp-eq-base, simp-all)
done
```

21.2 Nth Root

We define roots of negative reals such that *root n (− x) = − root n x*. This allows us to omit side conditions from many theorems.

```
definition
  root :: [nat, real] ⇒ real where
  root n x = (if 0 < x then (THE u. 0 < u ∧ u ^ n = x) else
              if x < 0 then − (THE u. 0 < u ∧ u ^ n = − x) else 0)

lemma real-root-zero [simp]: root n 0 = 0
unfolding root-def by simp

lemma real-root-minus: 0 < n ⟹ root n (− x) = − root n x
unfolding root-def by simp

lemma real-root-gt-zero: ⟦0 < n; 0 < x⟧ ⟹ 0 < root n x
```

apply (*simp add*: *root-def*)
apply (*drule* (*1*) *realpow-pos-nth-unique*)
apply (*erule theI'* [*THEN conjunct1*])
done

lemma *real-root-pow-pos*:
  ⟦*0* < *n*; *0* < *x*⟧ ⟹ *root n x* ^ *n* = *x*
apply (*simp add*: *root-def*)
apply (*drule* (*1*) *realpow-pos-nth-unique*)
apply (*erule theI'* [*THEN conjunct2*])
done

lemma *real-root-pow-pos2* [*simp*]:
  ⟦*0* < *n*; *0* ≤ *x*⟧ ⟹ *root n x* ^ *n* = *x*
by (*auto simp add*: *order-le-less real-root-pow-pos*)

lemma *odd-real-root-pow*: *odd n* ⟹ *root n x* ^ *n* = *x*
apply (*rule-tac x*=*0* **and** *y*=*x* **in** *linorder-le-cases*)
apply (*erule* (*1*) *real-root-pow-pos2* [*OF odd-pos*])
apply (*subgoal-tac root n* (− *x*) ^ *n* = − *x*)
apply (*simp add*: *real-root-minus odd-pos*)
apply (*simp add*: *odd-pos*)
done

lemma *real-root-ge-zero*: ⟦*0* < *n*; *0* ≤ *x*⟧ ⟹ *0* ≤ *root n x*
by (*auto simp add*: *order-le-less real-root-gt-zero*)

lemma *real-root-power-cancel*: ⟦*0* < *n*; *0* ≤ *x*⟧ ⟹ *root n* (*x* ^ *n*) = *x*
apply (*subgoal-tac* *0* ≤ *x* ^ *n*)
apply (*subgoal-tac* *0* ≤ *root n* (*x* ^ *n*))
apply (*subgoal-tac root n* (*x* ^ *n*) ^ *n* = *x* ^ *n*)
apply (*erule* (*3*) *power-eq-imp-eq-base*)
apply (*erule* (*1*) *real-root-pow-pos2*)
apply (*erule* (*1*) *real-root-ge-zero*)
apply (*erule zero-le-power*)
done

lemma *odd-real-root-power-cancel*: *odd n* ⟹ *root n* (*x* ^ *n*) = *x*
apply (*rule-tac x*=*0* **and** *y*=*x* **in** *linorder-le-cases*)
apply (*erule* (*1*) *real-root-power-cancel* [*OF odd-pos*])
apply (*subgoal-tac root n* ((− *x*) ^ *n*) = − *x*)
apply (*simp add*: *real-root-minus odd-pos*)
apply (*erule real-root-power-cancel* [*OF odd-pos*], *simp*)
done

lemma *real-root-pos-unique*:
  ⟦*0* < *n*; *0* ≤ *y*; *y* ^ *n* = *x*⟧ ⟹ *root n x* = *y*
by (*erule subst*, *rule real-root-power-cancel*)

lemma *odd-real-root-unique*:
⟦*odd n*; *y ^ n = x*⟧ ⟹ *root n x = y*
by (*erule subst*, *rule odd-real-root-power-cancel*)

lemma *real-root-one* [*simp*]: *0 < n* ⟹ *root n 1 = 1*
by (*simp add*: *real-root-pos-unique*)

Root function is strictly monotonic, hence injective

lemma *real-root-less-mono-lemma*:
⟦*0 < n*; *0 ≤ x*; *x < y*⟧ ⟹ *root n x < root n y*
apply (*subgoal-tac 0 ≤ y*)
apply (*subgoal-tac root n x ^ n < root n y ^ n*)
apply (*erule power-less-imp-less-base*)
apply (*erule* (*1*) *real-root-ge-zero*)
apply *simp*
apply *simp*
done

lemma *real-root-less-mono*: ⟦*0 < n*; *x < y*⟧ ⟹ *root n x < root n y*
apply (*cases 0 ≤ x*)
apply (*erule* (*2*) *real-root-less-mono-lemma*)
apply (*cases 0 ≤ y*)
apply (*rule-tac y=0* **in** *order-less-le-trans*)
apply (*subgoal-tac 0 < root n* (*− x*))
apply (*simp add*: *real-root-minus*)
apply (*simp add*: *real-root-gt-zero*)
apply (*simp add*: *real-root-ge-zero*)
apply (*subgoal-tac root n* (*− y*) *< root n* (*− x*))
apply (*simp add*: *real-root-minus*)
apply (*simp add*: *real-root-less-mono-lemma*)
done

lemma *real-root-le-mono*: ⟦*0 < n*; *x ≤ y*⟧ ⟹ *root n x ≤ root n y*
by (*auto simp add*: *order-le-less real-root-less-mono*)

lemma *real-root-less-iff* [*simp*]:
0 < n ⟹ (*root n x < root n y*) = (*x < y*)
apply (*cases x < y*)
apply (*simp add*: *real-root-less-mono*)
apply (*simp add*: *linorder-not-less real-root-le-mono*)
done

lemma *real-root-le-iff* [*simp*]:
0 < n ⟹ (*root n x ≤ root n y*) = (*x ≤ y*)
apply (*cases x ≤ y*)
apply (*simp add*: *real-root-le-mono*)
apply (*simp add*: *linorder-not-le real-root-less-mono*)
done

lemma *real-root-eq-iff* [*simp*]:
  $0 < n \Longrightarrow (root\ n\ x = root\ n\ y) = (x = y)$
by (*simp add*: *order-eq-iff*)

lemmas *real-root-gt-0-iff* [*simp*] = *real-root-less-iff* [**where** *x=0*, *simplified*]
lemmas *real-root-lt-0-iff* [*simp*] = *real-root-less-iff* [**where** *y=0*, *simplified*]
lemmas *real-root-ge-0-iff* [*simp*] = *real-root-le-iff* [**where** *x=0*, *simplified*]
lemmas *real-root-le-0-iff* [*simp*] = *real-root-le-iff* [**where** *y=0*, *simplified*]
lemmas *real-root-eq-0-iff* [*simp*] = *real-root-eq-iff* [**where** *y=0*, *simplified*]

lemma *real-root-gt-1-iff* [*simp*]: $0 < n \Longrightarrow (1 < root\ n\ y) = (1 < y)$
by (*insert real-root-less-iff* [**where** *x=1*], *simp*)

lemma *real-root-lt-1-iff* [*simp*]: $0 < n \Longrightarrow (root\ n\ x < 1) = (x < 1)$
by (*insert real-root-less-iff* [**where** *y=1*], *simp*)

lemma *real-root-ge-1-iff* [*simp*]: $0 < n \Longrightarrow (1 \leq root\ n\ y) = (1 \leq y)$
by (*insert real-root-le-iff* [**where** *x=1*], *simp*)

lemma *real-root-le-1-iff* [*simp*]: $0 < n \Longrightarrow (root\ n\ x \leq 1) = (x \leq 1)$
by (*insert real-root-le-iff* [**where** *y=1*], *simp*)

lemma *real-root-eq-1-iff* [*simp*]: $0 < n \Longrightarrow (root\ n\ x = 1) = (x = 1)$
by (*insert real-root-eq-iff* [**where** *y=1*], *simp*)

Roots of roots

lemma *real-root-Suc-0* [*simp*]: $root\ (Suc\ 0)\ x = x$
by (*simp add*: *odd-real-root-unique*)

lemma *real-root-pos-mult-exp*:
  $[\![0 < m;\ 0 < n;\ 0 < x]\!] \Longrightarrow root\ (m * n)\ x = root\ m\ (root\ n\ x)$
by (*rule real-root-pos-unique*, *simp-all add*: *power-mult*)

lemma *real-root-mult-exp*:
  $[\![0 < m;\ 0 < n]\!] \Longrightarrow root\ (m * n)\ x = root\ m\ (root\ n\ x)$
apply (*rule linorder-cases* [**where** *x=x* **and** *y=0*])
apply (*subgoal-tac root* $(m * n)\ (-x) = root\ m\ (root\ n\ (-x))$)
apply (*simp add*: *real-root-minus*)
apply (*simp-all add*: *real-root-pos-mult-exp*)
done

lemma *real-root-commute*:
  $[\![0 < m;\ 0 < n]\!] \Longrightarrow root\ m\ (root\ n\ x) = root\ n\ (root\ m\ x)$
by (*simp add*: *real-root-mult-exp* [*symmetric*] *mult-commute*)

Monotonicity in first argument

lemma *real-root-strict-decreasing*:
  $[\![0 < n;\ n < N;\ 1 < x]\!] \Longrightarrow root\ N\ x < root\ n\ x$
apply (*subgoal-tac root n* $(root\ N\ x)\ \hat{}\ n < root\ N\ (root\ n\ x)\ \hat{}\ N$, *simp*)

apply (*simp add*: *real-root-commute power-strict-increasing*
del: *real-root-pow-pos2*)
done

lemma *real-root-strict-increasing*:
⟦*0* < *n*; *n* < *N*; *0* < *x*; *x* < *1*⟧ ⟹ *root n x* < *root N x*
apply (*subgoal-tac root N* (*root n x*) ^ *N* < *root n* (*root N x*) ^ *n*, *simp*)
apply (*simp add*: *real-root-commute power-strict-decreasing*
del: *real-root-pow-pos2*)
done

lemma *real-root-decreasing*:
⟦*0* < *n*; *n* < *N*; *1* ≤ *x*⟧ ⟹ *root N x* ≤ *root n x*
by (*auto simp add*: *order-le-less real-root-strict-decreasing*)

lemma *real-root-increasing*:
⟦*0* < *n*; *n* < *N*; *0* ≤ *x*; *x* ≤ *1*⟧ ⟹ *root n x* ≤ *root N x*
by (*auto simp add*: *order-le-less real-root-strict-increasing*)

Roots of multiplication and division

lemma *real-root-mult-lemma*:
⟦*0* < *n*; *0* ≤ *x*; *0* ≤ *y*⟧ ⟹ *root n* (*x* ∗ *y*) = *root n x* ∗ *root n y*
by (*simp add*: *real-root-pos-unique mult-nonneg-nonneg power-mult-distrib*)

lemma *real-root-inverse-lemma*:
⟦*0* < *n*; *0* ≤ *x*⟧ ⟹ *root n* (*inverse x*) = *inverse* (*root n x*)
by (*simp add*: *real-root-pos-unique power-inverse* [*symmetric*])

lemma *real-root-mult*:
assumes *n*: *0* < *n*
shows *root n* (*x* ∗ *y*) = *root n x* ∗ *root n y*
proof (*rule linorder-le-cases*, *rule-tac* [!] *linorder-le-cases*)
assume *0* ≤ *x* **and** *0* ≤ *y*
thus *?thesis* **by** (*rule real-root-mult-lemma* [*OF n*])
next
assume *0* ≤ *x* **and** *y* ≤ *0*
hence *0* ≤ *x* **and** *0* ≤ − *y* **by** *simp-all*
hence *root n* (*x* ∗ − *y*) = *root n x* ∗ *root n* (− *y*)
by (*rule real-root-mult-lemma* [*OF n*])
thus *?thesis* **by** (*simp add*: *real-root-minus* [*OF n*])
next
assume *x* ≤ *0* **and** *0* ≤ *y*
hence *0* ≤ − *x* **and** *0* ≤ *y* **by** *simp-all*
hence *root n* (− *x* ∗ *y*) = *root n* (− *x*) ∗ *root n y*
by (*rule real-root-mult-lemma* [*OF n*])
thus *?thesis* **by** (*simp add*: *real-root-minus* [*OF n*])
next
assume *x* ≤ *0* **and** *y* ≤ *0*
hence *0* ≤ − *x* **and** *0* ≤ − *y* **by** *simp-all*

hence *root n (− x ∗ − y) = root n (− x) ∗ root n (− y)*
by (*rule real-root-mult-lemma* [*OF n*])
thus *?thesis* **by** (*simp add*: *real-root-minus* [*OF n*])
qed

lemma *real-root-inverse*:
assumes *n*: *0 < n*
shows *root n (inverse x) = inverse (root n x)*
proof (*rule linorder-le-cases*)
assume *0 ≤ x*
thus *?thesis* **by** (*rule real-root-inverse-lemma* [*OF n*])
next
assume *x ≤ 0*
hence *0 ≤ − x* **by** *simp*
hence *root n (inverse (− x)) = inverse (root n (− x))*
by (*rule real-root-inverse-lemma* [*OF n*])
thus *?thesis* **by** (*simp add*: *real-root-minus* [*OF n*])
qed

lemma *real-root-divide*:
0 < n ⟹ root n (x / y) = root n x / root n y
by (*simp add*: *divide-inverse real-root-mult real-root-inverse*)

lemma *real-root-power*:
0 < n ⟹ root n (x ^ k) = root n x ^ k
by (*induct k*, *simp-all add*: *real-root-mult*)

lemma *real-root-abs*: *0 < n ⟹ root n |x| = |root n x|*
by (*simp add*: *abs-if real-root-minus*)

Continuity and derivatives

lemma *isCont-root-pos*:
assumes *n*: *0 < n*
assumes *x*: *0 < x*
shows *isCont (root n) x*
proof −
have *isCont (root n) (root n x ^ n)*
proof (*rule isCont-inverse-function* [**where** *f*=*λa. a ^ n*])
show *0 < root n x* **using** *n x* **by** *simp*
show *∀ z. |z − root n x| ≤ root n x ⟶ root n (z ^ n) = z*
by (*simp add*: *abs-le-iff real-root-power-cancel n*)
show *∀ z. |z − root n x| ≤ root n x ⟶ isCont (λa. a ^ n) z*
by (*simp add*: *isCont-power*)
qed
thus *?thesis* **using** *n x* **by** *simp*
qed

lemma *isCont-root-neg*:
⟦0 < n; *x < 0⟧ ⟹ isCont (root n) x*

apply (*subgoal-tac isCont* (λ*x*. − *root n* (− *x*)) *x*)
apply (*simp add*: *real-root-minus*)
apply (*rule isCont-o2* [*OF isCont-minus* [*OF isCont-ident*]])
apply (*simp add*: *isCont-minus isCont-root-pos*)
done

lemma *isCont-root-zero*:
0 < *n* ⟹ *isCont* (*root n*) *0*
unfolding *isCont-def*
apply (*rule LIM-I*)
apply (*rule-tac x*=*r* ^ *n* **in** *exI*, *safe*)
apply (*simp*)
apply (*simp add*: *real-root-abs* [*symmetric*])
apply (*rule-tac n*=*n* **in** *power-less-imp-less-base*, *simp-all*)
done

lemma *isCont-real-root*: *0* < *n* ⟹ *isCont* (*root n*) *x*
apply (*rule-tac x*=*x* **and** *y*=*0* **in** *linorder-cases*)
apply (*simp-all add*: *isCont-root-pos isCont-root-neg isCont-root-zero*)
done

lemma *DERIV-real-root*:
assumes *n*: *0* < *n*
assumes *x*: *0* < *x*
shows *DERIV* (*root n*) *x* :> *inverse* (*real n* ∗ *root n x* ^ (*n* − *Suc 0*))
proof (*rule DERIV-inverse-function*)
show *0* < *x* **using** *x* **.**
show *x* < *x* + *1* **by** *simp*
show ∀ *y*. *0* < *y* ∧ *y* < *x* + *1* ⟶ *root n y* ^ *n* = *y*
using *n* **by** *simp*
show *DERIV* (λ*x*. *x* ^ *n*) (*root n x*) :> *real n* ∗ *root n x* ^ (*n* − *Suc 0*)
by (*rule DERIV-pow*)
show *real n* ∗ *root n x* ^ (*n* − *Suc 0*) ≠ *0*
using *n x* **by** *simp*
show *isCont* (*root n*) *x*
using *n* **by** (*rule isCont-real-root*)
qed

lemma *DERIV-odd-real-root*:
assumes *n*: *odd n*
assumes *x*: *x* ≠ *0*
shows *DERIV* (*root n*) *x* :> *inverse* (*real n* ∗ *root n x* ^ (*n* − *Suc 0*))
proof (*rule DERIV-inverse-function*)
show *x* − *1* < *x* **by** *simp*
show *x* < *x* + *1* **by** *simp*
show ∀ *y*. *x* − *1* < *y* ∧ *y* < *x* + *1* ⟶ *root n y* ^ *n* = *y*
using *n* **by** (*simp add*: *odd-real-root-pow*)
show *DERIV* (λ*x*. *x* ^ *n*) (*root n x*) :> *real n* ∗ *root n x* ^ (*n* − *Suc 0*)
by (*rule DERIV-pow*)

show *real n ∗ root n x ^ (n − Suc 0) ≠ 0*
using *odd-pos* [*OF n*] *x* **by** *simp*
show *isCont* (*root n*) *x*
using *odd-pos* [*OF n*] **by** (*rule isCont-real-root*)
qed

21.3 Square Root

definition
sqrt :: *real* ⇒ *real* **where**
sqrt = *root 2*

lemma *pos2*: *0* < (*2*::*nat*) **by** *simp*

lemma *real-sqrt-unique*: ⟦y^2 = *x*; *0* ≤ *y*⟧ ⟹ *sqrt x* = *y*
unfolding *sqrt-def* **by** (*rule real-root-pos-unique* [*OF pos2*])

lemma *real-sqrt-abs* [*simp*]: *sqrt* (x^2) = |*x*|
apply (*rule real-sqrt-unique*)
apply (*rule power2-abs*)
apply (*rule abs-ge-zero*)
done

lemma *real-sqrt-pow2* [*simp*]: *0* ≤ *x* ⟹ (*sqrt x*)^2 = *x*
unfolding *sqrt-def* **by** (*rule real-root-pow-pos2* [*OF pos2*])

lemma *real-sqrt-pow2-iff* [*simp*]: ((*sqrt x*)^2 = *x*) = (*0* ≤ *x*)
apply (*rule iffI*)
apply (*erule subst*)
apply (*rule zero-le-power2*)
apply (*erule real-sqrt-pow2*)
done

lemma *real-sqrt-zero* [*simp*]: *sqrt 0* = *0*
unfolding *sqrt-def* **by** (*rule real-root-zero*)

lemma *real-sqrt-one* [*simp*]: *sqrt 1* = *1*
unfolding *sqrt-def* **by** (*rule real-root-one* [*OF pos2*])

lemma *real-sqrt-minus*: *sqrt* (− *x*) = − *sqrt x*
unfolding *sqrt-def* **by** (*rule real-root-minus* [*OF pos2*])

lemma *real-sqrt-mult*: *sqrt* (*x ∗ y*) = *sqrt x ∗ sqrt y*
unfolding *sqrt-def* **by** (*rule real-root-mult* [*OF pos2*])

lemma *real-sqrt-inverse*: *sqrt* (*inverse x*) = *inverse* (*sqrt x*)
unfolding *sqrt-def* **by** (*rule real-root-inverse* [*OF pos2*])

lemma *real-sqrt-divide*: *sqrt* (*x / y*) = *sqrt x / sqrt y*

unfolding *sqrt-def* **by** (*rule real-root-divide* [*OF pos2*])

lemma *real-sqrt-power*: *sqrt* (*x* ^ *k*) = *sqrt x* ^ *k*
unfolding *sqrt-def* **by** (*rule real-root-power* [*OF pos2*])

lemma *real-sqrt-gt-zero*: *0* < *x* ⟹ *0* < *sqrt x*
unfolding *sqrt-def* **by** (*rule real-root-gt-zero* [*OF pos2*])

lemma *real-sqrt-ge-zero*: *0* ≤ *x* ⟹ *0* ≤ *sqrt x*
unfolding *sqrt-def* **by** (*rule real-root-ge-zero* [*OF pos2*])

lemma *real-sqrt-less-mono*: *x* < *y* ⟹ *sqrt x* < *sqrt y*
unfolding *sqrt-def* **by** (*rule real-root-less-mono* [*OF pos2*])

lemma *real-sqrt-le-mono*: *x* ≤ *y* ⟹ *sqrt x* ≤ *sqrt y*
unfolding *sqrt-def* **by** (*rule real-root-le-mono* [*OF pos2*])

lemma *real-sqrt-less-iff* [*simp*]: (*sqrt x* < *sqrt y*) = (*x* < *y*)
unfolding *sqrt-def* **by** (*rule real-root-less-iff* [*OF pos2*])

lemma *real-sqrt-le-iff* [*simp*]: (*sqrt x* ≤ *sqrt y*) = (*x* ≤ *y*)
unfolding *sqrt-def* **by** (*rule real-root-le-iff* [*OF pos2*])

lemma *real-sqrt-eq-iff* [*simp*]: (*sqrt x* = *sqrt y*) = (*x* = *y*)
unfolding *sqrt-def* **by** (*rule real-root-eq-iff* [*OF pos2*])

lemmas *real-sqrt-gt-0-iff* [*simp*] = *real-sqrt-less-iff* [**where** *x=0*, *simplified*]
lemmas *real-sqrt-lt-0-iff* [*simp*] = *real-sqrt-less-iff* [**where** *y=0*, *simplified*]
lemmas *real-sqrt-ge-0-iff* [*simp*] = *real-sqrt-le-iff* [**where** *x=0*, *simplified*]
lemmas *real-sqrt-le-0-iff* [*simp*] = *real-sqrt-le-iff* [**where** *y=0*, *simplified*]
lemmas *real-sqrt-eq-0-iff* [*simp*] = *real-sqrt-eq-iff* [**where** *y=0*, *simplified*]

lemmas *real-sqrt-gt-1-iff* [*simp*] = *real-sqrt-less-iff* [**where** *x=1*, *simplified*]
lemmas *real-sqrt-lt-1-iff* [*simp*] = *real-sqrt-less-iff* [**where** *y=1*, *simplified*]
lemmas *real-sqrt-ge-1-iff* [*simp*] = *real-sqrt-le-iff* [**where** *x=1*, *simplified*]
lemmas *real-sqrt-le-1-iff* [*simp*] = *real-sqrt-le-iff* [**where** *y=1*, *simplified*]
lemmas *real-sqrt-eq-1-iff* [*simp*] = *real-sqrt-eq-iff* [**where** *y=1*, *simplified*]

lemma *isCont-real-sqrt*: *isCont sqrt x*
unfolding *sqrt-def* **by** (*rule isCont-real-root* [*OF pos2*])

lemma *DERIV-real-sqrt*:
  0 < *x* ⟹ *DERIV sqrt x* :> *inverse* (*sqrt x*) / *2*
unfolding *sqrt-def* **by** (*rule DERIV-real-root* [*OF pos2*, *simplified*])

lemma *not-real-square-gt-zero* [*simp*]: (~ (*0*::*real*) < *x*∗*x*) = (*x* = *0*)
apply *auto*
apply (*cut-tac x* = *x* **and** *y* = *0* **in** *linorder-less-linear*)
apply (*simp add*: *zero-less-mult-iff*)

done

lemma *real-sqrt-abs2* [*simp*]: $sqrt(x*x) = |x|$
apply (*subst power2-eq-square* [*symmetric*])
apply (*rule real-sqrt-abs*)
done

lemma *real-sqrt-pow2-gt-zero*: $0 < x \Longrightarrow 0 < (sqrt\ x)^2$
by *simp*

lemma *real-sqrt-not-eq-zero*: $0 < x \Longrightarrow sqrt\ x \neq 0$
by *simp*

lemma *real-inv-sqrt-pow2*: $0 < x \Longrightarrow inverse\ (sqrt(x))\ \hat{}\ 2 = inverse\ x$
by (*simp add*: *power-inverse* [*symmetric*])

lemma *real-sqrt-eq-zero-cancel*: $[|\ 0 \leq x;\ sqrt(x) = 0|] \Longrightarrow x = 0$
by *simp*

lemma *real-sqrt-ge-one*: $1 \leq x \Longrightarrow 1 \leq sqrt\ x$
by *simp*

lemma *real-sqrt-two-gt-zero* [*simp*]: $0 < sqrt\ 2$
by *simp*

lemma *real-sqrt-two-ge-zero* [*simp*]: $0 \leq sqrt\ 2$
by *simp*

lemma *real-sqrt-two-gt-one* [*simp*]: $1 < sqrt\ 2$
by *simp*

lemma *sqrt-divide-self-eq*:
  assumes *nneg*: $0 \leq x$
  shows $sqrt\ x\ /\ x = inverse\ (sqrt\ x)$
proof *cases*
  assume $x=0$ **thus** *?thesis* **by** *simp*
next
  assume *nz*: $x\neq 0$
  hence *pos*: $0<x$ **using** *nneg* **by** *arith*
  show *?thesis*
  proof (*rule right-inverse-eq* [*THEN iffD1*, *THEN sym*])
    show $sqrt\ x\ /\ x \neq 0$ **by** (*simp add*: *divide-inverse nneg nz*)
    show $inverse\ (sqrt\ x)\ /\ (sqrt\ x\ /\ x) = 1$
      by (*simp add*: *divide-inverse mult-assoc* [*symmetric*]
                power2-eq-square [*symmetric*] *real-inv-sqrt-pow2 pos nz*)
  qed
qed

lemma *real-divide-square-eq* [*simp*]: $(((r::real) * a)\ /\ (r * r)) = a\ /\ r$

apply (*simp add*: *divide-inverse*)
apply (*case-tac r=0*)
apply (*auto simp add*: *mult-ac*)
done

lemma *lemma-real-divide-sqrt-less*: *0 < u ==> u / sqrt 2 < u*
by (*simp add*: *divide-less-eq mult-compare-simps*)

lemma *four-x-squared*:
  fixes *x*::*real*
  shows $4 * x^2 = (2 * x)^2$
by (*simp add*: *power2-eq-square*)

21.4 Square Root of Sum of Squares

lemma *real-sqrt-mult-self-sum-ge-zero* [*simp*]: $0 \leq sqrt(x*x + y*y)$
by (*rule real-sqrt-ge-zero* [*OF sum-squares-ge-zero*])

lemma *real-sqrt-sum-squares-ge-zero* [*simp*]: $0 \leq sqrt\ (x^2 + y^2)$
by *simp*

declare *real-sqrt-sum-squares-ge-zero* [*THEN abs-of-nonneg*, *simp*]

lemma *real-sqrt-sum-squares-mult-ge-zero* [*simp*]:
    $0 \leq sqrt\ ((x^2 + y^2)*(xa^2 + ya^2))$
by (*auto intro*!: *real-sqrt-ge-zero simp add*: *zero-le-mult-iff*)

lemma *real-sqrt-sum-squares-mult-squared-eq* [*simp*]:
    $sqrt\ ((x^2 + y^2) * (xa^2 + ya^2))\ \hat{}\ 2 = (x^2 + y^2) * (xa^2 + ya^2)$
by (*auto simp add*: *zero-le-mult-iff*)

lemma *real-sqrt-sum-squares-eq-cancel*: $sqrt\ (x^2 + y^2) = x \Longrightarrow y = 0$
by (*drule-tac f* = $\%x.\ x^2$ **in** *arg-cong*, *simp*)

lemma *real-sqrt-sum-squares-eq-cancel2*: $sqrt\ (x^2 + y^2) = y \Longrightarrow x = 0$
by (*drule-tac f* = $\%x.\ x^2$ **in** *arg-cong*, *simp*)

lemma *real-sqrt-sum-squares-ge1* [*simp*]: $x \leq sqrt\ (x^2 + y^2)$
by (*rule power2-le-imp-le*, *simp-all*)

lemma *real-sqrt-sum-squares-ge2* [*simp*]: $y \leq sqrt\ (x^2 + y^2)$
by (*rule power2-le-imp-le*, *simp-all*)

lemma *real-sqrt-ge-abs1* [*simp*]: $|x| \leq sqrt\ (x^2 + y^2)$
by (*rule power2-le-imp-le*, *simp-all*)

lemma *real-sqrt-ge-abs2* [*simp*]: $|y| \leq sqrt\ (x^2 + y^2)$
by (*rule power2-le-imp-le*, *simp-all*)

lemma *le-real-sqrt-sumsq* [*simp*]: $x \leq sqrt\ (x * x + y * y)$
by (*simp add*: *power2-eq-square* [*symmetric*])

lemma *power2-sum*:
  fixes $x\ y :: 'a$::{*number-ring*,*recpower*}
  shows $(x + y)^2 = x^2 + y^2 + 2 * x * y$
by (*simp add*: *ring-distribs power2-eq-square*)

lemma *power2-diff*:
  fixes $x\ y :: 'a$::{*number-ring*,*recpower*}
  shows $(x - y)^2 = x^2 + y^2 - 2 * x * y$
by (*simp add*: *ring-distribs power2-eq-square*)

lemma *real-sqrt-sum-squares-triangle-ineq*:
  $sqrt\ ((a + c)^2 + (b + d)^2) \leq sqrt\ (a^2 + b^2) + sqrt\ (c^2 + d^2)$
apply (*rule power2-le-imp-le*, *simp*)
apply (*simp add*: *power2-sum*)
apply (*simp only*: *mult-assoc right-distrib* [*symmetric*])
apply (*rule mult-left-mono*)
apply (*rule power2-le-imp-le*)
apply (*simp add*: *power2-sum power-mult-distrib*)
apply (*simp add*: *ring-distribs*)
apply (*subgoal-tac* $0 \leq b^2 * c^2 + a^2 * d^2 - 2 * (a * c) * (b * d)$, *simp*)
apply (*rule-tac* $b=(a * d - b * c)^2$ **in** *ord-le-eq-trans*)
apply (*rule zero-le-power2*)
apply (*simp add*: *power2-diff power-mult-distrib*)
apply (*simp add*: *mult-nonneg-nonneg*)
apply *simp*
apply (*simp add*: *add-increasing*)
done

lemma *real-sqrt-sum-squares-less*:
  $[\![|x| < u\ /\ sqrt\ 2; |y| < u\ /\ sqrt\ 2]\!] \Longrightarrow sqrt\ (x^2 + y^2) < u$
apply (*rule power2-less-imp-less*, *simp*)
apply (*drule power-strict-mono* [*OF - abs-ge-zero pos2*])
apply (*drule power-strict-mono* [*OF - abs-ge-zero pos2*])
apply (*simp add*: *power-divide*)
apply (*drule order-le-less-trans* [*OF abs-ge-zero*])
apply (*simp add*: *zero-less-divide-iff*)
done

Needed for the infinitely close relation over the nonstandard complex numbers

lemma *lemma-sqrt-hcomplex-capprox*:
  $[|\ 0 < u; x < u/2; y < u/2; 0 \leq x; 0 \leq y\ |] ==> sqrt\ (x^2 + y^2) < u$
apply (*rule-tac* $y = u/sqrt\ 2$ **in** *order-le-less-trans*)
apply (*erule-tac* [2] *lemma-real-divide-sqrt-less*)
apply (*rule power2-le-imp-le*)
apply (*auto simp add*: *real-0-le-divide-iff power-divide*)

apply (*rule-tac* $t = u^2$ **in** *real-sum-of-halves* [*THEN subst*])
apply (*rule add-mono*)
apply (*auto simp add*: *four-x-squared simp del*: *realpow-Suc intro*: *power-mono*)
done

Legacy theorem names:

lemmas *real-root-pos2* = *real-root-power-cancel*
lemmas *real-root-pos-pos* = *real-root-gt-zero* [*THEN order-less-imp-le*]
lemmas *real-root-pos-pos-le* = *real-root-ge-zero*
lemmas *real-sqrt-mult-distrib* = *real-sqrt-mult*
lemmas *real-sqrt-mult-distrib2* = *real-sqrt-mult*
lemmas *real-sqrt-eq-zero-cancel-iff* = *real-sqrt-eq-0-iff*

lemma *real-root-pos*: $0 < x \Longrightarrow root\ (Suc\ n)\ (x\ \hat{}\ (Suc\ n)) = x$
by (*rule real-root-power-cancel* [*OF zero-less-Suc order-less-imp-le*])

end

22 Transcendental: Power Series, Transcendental Functions etc.

theory *Transcendental*
imports *Fact Series Deriv NthRoot*
begin

22.1 Properties of Power Series

lemma *lemma-realpow-diff*:
  fixes $y :: 'a$::*recpower*
  shows $p \leq n \Longrightarrow y\ \hat{}\ (Suc\ n - p) = (y\ \hat{}\ (n - p)) * y$
proof −
  assume $p \leq n$
  hence $Suc\ n - p = Suc\ (n - p)$ **by** (*rule Suc-diff-le*)
  thus *?thesis* **by** (*simp add*: *power-Suc power-commutes*)
qed

lemma *lemma-realpow-diff-sumr*:
  fixes $y :: 'a$::{*recpower*,*comm-semiring-0*} **shows**
    $(\sum p{=}0..{<}Suc\ n.\ (x\ \hat{}\ p) * y\ \hat{}\ (Suc\ n - p)) =$
     $y * (\sum p{=}0..{<}Suc\ n.\ (x\ \hat{}\ p) * y\ \hat{}\ (n - p))$
by (*auto simp add*: *setsum-right-distrib lemma-realpow-diff mult-ac*
  simp del: *setsum-op-ivl-Suc cong*: *strong-setsum-cong*)

lemma *lemma-realpow-diff-sumr2*:
  fixes $y :: 'a$::{*recpower*,*comm-ring*} **shows**
    $x\ \hat{}\ (Suc\ n) - y\ \hat{}\ (Suc\ n) =$

```
    (x − y) * (∑p=0..<Suc n. (x ^ p) * y ^ (n − p))
apply (induct n, simp add: power-Suc)
apply (simp add: power-Suc del: setsum-op-ivl-Suc)
apply (subst setsum-op-ivl-Suc)
apply (subst lemma-realpow-diff-sumr)
apply (simp add: right-distrib del: setsum-op-ivl-Suc)
apply (subst mult-left-commute [where a=x − y])
apply (erule subst)
apply (simp add: power-Suc ring-simps)
done

lemma lemma-realpow-rev-sumr:
    (∑p=0..<Suc n. (x ^ p) * (y ^ (n − p))) =
     (∑p=0..<Suc n. (x ^ (n − p)) * (y ^ p))
apply (rule setsum-reindex-cong [where f=λi. n − i])
apply (rule inj-onI, simp)
apply auto
apply (rule-tac x=n − x in image-eqI, simp, simp)
done
```

Power series has a 'circle' of convergence, i.e. if it sums for x, then it sums absolutely for z with $|z| < |x|$.

```
lemma powser-insidea:
  fixes x z :: 'a::{real-normed-field,banach,recpower}
  assumes 1: summable (λn. f n * x ^ n)
  assumes 2: norm z < norm x
  shows summable (λn. norm (f n * z ^ n))
proof −
  from 2 have x-neq-0: x ≠ 0 by clarsimp
  from 1 have (λn. f n * x ^ n) −−−−> 0
    by (rule summable-LIMSEQ-zero)
  hence convergent (λn. f n * x ^ n)
    by (rule convergentI)
  hence Cauchy (λn. f n * x ^ n)
    by (simp add: Cauchy-convergent-iff)
  hence Bseq (λn. f n * x ^ n)
    by (rule Cauchy-Bseq)
  then obtain K where 3: 0 < K and 4: ∀n. norm (f n * x ^ n) ≤ K
    by (simp add: Bseq-def, safe)
  have ∃N. ∀n≥N. norm (norm (f n * z ^ n)) ≤
                 K * norm (z ^ n) * inverse (norm (x ^ n))
  proof (intro exI allI impI)
    fix n::nat assume 0 ≤ n
    have norm (norm (f n * z ^ n)) * norm (x ^ n) =
          norm (f n * x ^ n) * norm (z ^ n)
      by (simp add: norm-mult abs-mult)
    also have ... ≤ K * norm (z ^ n)
      by (simp only: mult-right-mono 4 norm-ge-zero)
    also have ... = K * norm (z ^ n) * (inverse (norm (x ^ n)) * norm (x ^ n))
```

```
    by (simp add: x-neq-0)
  also have ... = K * norm (z ^ n) * inverse (norm (x ^ n)) * norm (x ^ n)
    by (simp only: mult-assoc)
  finally show norm (norm (f n * z ^ n)) ≤
               K * norm (z ^ n) * inverse (norm (x ^ n))
    by (simp add: mult-le-cancel-right x-neq-0)
 qed
 moreover have summable (λn. K * norm (z ^ n) * inverse (norm (x ^ n)))
 proof -
  from 2 have norm (norm (z * inverse x)) < 1
    using x-neq-0
    by (simp add: nonzero-norm-divide divide-inverse [symmetric])
  hence summable (λn. norm (z * inverse x) ^ n)
    by (rule summable-geometric)
  hence summable (λn. K * norm (z * inverse x) ^ n)
    by (rule summable-mult)
  thus summable (λn. K * norm (z ^ n) * inverse (norm (x ^ n)))
    using x-neq-0
    by (simp add: norm-mult nonzero-norm-inverse power-mult-distrib
                  power-inverse norm-power mult-assoc)
 qed
 ultimately show summable (λn. norm (f n * z ^ n))
  by (rule summable-comparison-test)
qed

lemma powser-inside:
 fixes f :: nat ⇒ 'a::{real-normed-field,banach,recpower} shows
   [| summable (%n. f(n) * (x ^ n)); norm z < norm x |]
    ==> summable (%n. f(n) * (z ^ n))
by (rule powser-insidea [THEN summable-norm-cancel])
```

22.2 Term-by-Term Differentiability of Power Series

```
definition
 diffs :: (nat => 'a::ring-1) => nat => 'a where
 diffs c = (%n. of-nat (Suc n) * c(Suc n))
```

Lemma about distributing negation over it

```
lemma diffs-minus: diffs (%n. - c n) = (%n. - diffs c n)
by (simp add: diffs-def)
```

Show that we can shift the terms down one

```
lemma lemma-diffs:
    (∑ n=0..<n. (diffs c)(n) * (x ^ n)) =
     (∑ n=0..<n. of-nat n * c(n) * (x ^ (n - Suc 0))) +
     (of-nat n * c(n) * x ^ (n - Suc 0))
apply (induct n)
apply (auto simp add: mult-assoc add-assoc [symmetric] diffs-def)
done
```

lemma *lemma-diffs2*:
(∑ n=0..<n. *of-nat* n * c(n) * (x ^ (n − *Suc* 0))) =
(∑ n=0..<n. (*diffs* c)(n) * (x ^ n)) −
(*of-nat* n * c(n) * x ^ (n − *Suc* 0))
by (*auto simp add*: *lemma-diffs*)

lemma *diffs-equiv*:
summable (%n. (*diffs* c)(n) * (x ^ n)) ==>
(%n. *of-nat* n * c(n) * (x ^ (n − *Suc* 0))) *sums*
(∑ n. (*diffs* c)(n) * (x ^ n))
apply (*subgoal-tac* (%n. *of-nat* n * c (n) * (x ^ (n − *Suc* 0))) −−−−> 0)
apply (*rule-tac* [2] *LIMSEQ-imp-Suc*)
apply (*drule summable-sums*)
apply (*auto simp add*: *sums-def*)
apply (*drule-tac X*=(λn. ∑ n = 0..<n. *diffs* c n * x ^ n) **in** *LIMSEQ-diff*)
apply (*auto simp add*: *lemma-diffs2* [*symmetric*] *diffs-def* [*symmetric*])
apply (*simp add*: *diffs-def summable-LIMSEQ-zero*)
done

lemma *lemma-termdiff1*:
fixes z :: 'a :: {*recpower*,*comm-ring*} **shows**
(∑ p=0..<m. (((z + h) ^ (m − p)) * (z ^ p)) − (z ^ m)) =
(∑ p=0..<m. (z ^ p) * (((z + h) ^ (m − p)) − (z ^ (m − p))))
by (*auto simp add*: *right-distrib diff-minus power-add* [*symmetric*] *mult-ac*
cong: *strong-setsum-cong*)

lemma *less-add-one*: m < n ==> (∃ d. n = m + d + *Suc* 0)
by (*simp add*: *less-iff-Suc-add*)

lemma *sumdiff*: a + b − (c + d) = a − c + b − (d::*real*)
by *arith*

lemma *sumr-diff-mult-const2*:
setsum f {0..<n} − *of-nat* n * (r::'a::*ring-1*) = (∑ i = 0..<n. f i − r)
by (*simp add*: *setsum-subtractf*)

lemma *lemma-termdiff2*:
fixes h :: 'a :: {*recpower*,*field*}
assumes h: h ≠ 0 **shows**
((z + h) ^ n − z ^ n) / h − *of-nat* n * z ^ (n − *Suc* 0) =
h * (∑ p=0..< n − *Suc* 0. ∑ q=0..< n − *Suc* 0 − p.
(z + h) ^ q * z ^ (n − 2 − q)) (**is** *?lhs* = *?rhs*)
apply (*subgoal-tac* h * *?lhs* = h * *?rhs*, *simp add*: h)
apply (*simp add*: *right-diff-distrib diff-divide-distrib* h)
apply (*simp add*: *mult-assoc* [*symmetric*])
apply (*cases* n, *simp*)
apply (*simp add*: *lemma-realpow-diff-sumr2* h

```
            right-diff-distrib [symmetric] mult-assoc
          del: realpow-Suc setsum-op-ivl-Suc of-nat-Suc)
apply (subst lemma-realpow-rev-sumr)
apply (subst sumr-diff-mult-const2)
apply simp
apply (simp only: lemma-termdiff1 setsum-right-distrib)
apply (rule setsum-cong [OF refl])
apply (simp add: diff-minus [symmetric] less-iff-Suc-add)
apply (clarify)
apply (simp add: setsum-right-distrib lemma-realpow-diff-sumr2 mult-ac
          del: setsum-op-ivl-Suc realpow-Suc)
apply (subst mult-assoc [symmetric], subst power-add [symmetric])
apply (simp add: mult-ac)
done

lemma real-setsum-nat-ivl-bounded2:
  fixes K :: 'a::ordered-semidom
  assumes f: ⋀p::nat. p < n ⟹ f p ≤ K
  assumes K: 0 ≤ K
  shows setsum f {0..<n-k} ≤ of-nat n * K
apply (rule order-trans [OF setsum-mono])
apply (rule f, simp)
apply (simp add: mult-right-mono K)
done

lemma lemma-termdiff3:
  fixes h z :: 'a::{real-normed-field,recpower}
  assumes 1: h ≠ 0
  assumes 2: norm z ≤ K
  assumes 3: norm (z + h) ≤ K
  shows norm (((z + h) ^ n - z ^ n) / h - of-nat n * z ^ (n - Suc 0))
        ≤ of-nat n * of-nat (n - Suc 0) * K ^ (n - 2) * norm h
proof -
  have norm (((z + h) ^ n - z ^ n) / h - of-nat n * z ^ (n - Suc 0)) =
        norm (∑p = 0..<n - Suc 0. ∑q = 0..<n - Suc 0 - p.
          (z + h) ^ q * z ^ (n - 2 - q)) * norm h
    apply (subst lemma-termdiff2 [OF 1])
    apply (subst norm-mult)
    apply (rule mult-commute)
    done
  also have ... ≤ of-nat n * (of-nat (n - Suc 0) * K ^ (n - 2)) * norm h
  proof (rule mult-right-mono [OF - norm-ge-zero])
    from norm-ge-zero 2 have K: 0 ≤ K by (rule order-trans)
    have le-Kn: ⋀i j n. i + j = n ⟹ norm ((z + h) ^ i * z ^ j) ≤ K ^ n
      apply (erule subst)
      apply (simp only: norm-mult norm-power power-add)
      apply (intro mult-mono power-mono 2 3 norm-ge-zero zero-le-power K)
      done
    show norm (∑p = 0..<n - Suc 0. ∑q = 0..<n - Suc 0 - p.
```

```
          (z + h) ^ q * z ^ (n - 2 - q))
        ≤ of-nat n * (of-nat (n - Suc 0) * K ^ (n - 2))
    apply (intro
      order-trans [OF norm-setsum]
      real-setsum-nat-ivl-bounded2
      mult-nonneg-nonneg
      zero-le-imp-of-nat
      zero-le-power K)
    apply (rule le-Kn, simp)
    done
 qed
 also have ... = of-nat n * of-nat (n - Suc 0) * K ^ (n - 2) * norm h
   by (simp only: mult-assoc)
 finally show ?thesis .
qed

lemma lemma-termdiff4:
 fixes f :: 'a::{real-normed-field,recpower} ⇒
            'b::real-normed-vector
 assumes k: 0 < (k::real)
 assumes le: ⋀h. [[h ≠ 0; norm h < k]] ⟹ norm (f h) ≤ K * norm h
 shows f -- 0 --> 0
proof (simp add: LIM-def, safe)
 fix r::real assume r: 0 < r
 have zero-le-K: 0 ≤ K
   apply (cut-tac k)
   apply (cut-tac h=of-real (k/2) in le, simp)
   apply (simp del: of-real-divide)
   apply (drule order-trans [OF norm-ge-zero])
   apply (simp add: zero-le-mult-iff)
   done
 show ∃s. 0 < s ∧ (∀x. x ≠ 0 ∧ norm x < s ⟶ norm (f x) < r)
 proof (cases)
   assume K = 0
   with k r le have 0 < k ∧ (∀x. x ≠ 0 ∧ norm x < k ⟶ norm (f x) < r)
     by simp
   thus ∃s. 0 < s ∧ (∀x. x ≠ 0 ∧ norm x < s ⟶ norm (f x) < r) ..
 next
   assume K-neq-zero: K ≠ 0
   with zero-le-K have K: 0 < K by simp
   show ∃s. 0 < s ∧ (∀x. x ≠ 0 ∧ norm x < s ⟶ norm (f x) < r)
   proof (rule exI, safe)
     from k r K show 0 < min k (r * inverse K / 2)
       by (simp add: mult-pos-pos positive-imp-inverse-positive)
   next
     fix x::'a
     assume x1: x ≠ 0 and x2: norm x < min k (r * inverse K / 2)
     from x2 have x3: norm x < k and x4: norm x < r * inverse K / 2
       by simp-all
```

```
    from x1 x3 le have norm (f x) ≤ K * norm x by simp
    also from x4 K have K * norm x < K * (r * inverse K / 2)
      by (rule mult-strict-left-mono)
    also have ... = r / 2
      using K-neq-zero by simp
    also have r / 2 < r
      using r by simp
    finally show norm (f x) < r .
  qed
 qed
qed

lemma lemma-termdiff5:
 fixes g :: 'a::{recpower,real-normed-field} ⇒
            nat ⇒ 'b::banach
 assumes k: 0 < (k::real)
 assumes f: summable f
 assumes le: ⋀h n. ⟦h ≠ 0; norm h < k⟧ ⟹ norm (g h n) ≤ f n * norm h
 shows (λh. suminf (g h)) -- 0 --> 0
proof (rule lemma-termdiff4 [OF k])
 fix h::'a assume h ≠ 0 and norm h < k
 hence A: ∀ n. norm (g h n) ≤ f n * norm h
   by (simp add: le)
 hence ∃ N. ∀ n≥N. norm (norm (g h n)) ≤ f n * norm h
   by simp
 moreover from f have B: summable (λn. f n * norm h)
   by (rule summable-mult2)
 ultimately have C: summable (λn. norm (g h n))
   by (rule summable-comparison-test)
 hence norm (suminf (g h)) ≤ (∑ n. norm (g h n))
   by (rule summable-norm)
 also from A C B have (∑ n. norm (g h n)) ≤ (∑ n. f n * norm h)
   by (rule summable-le)
 also from f have (∑ n. f n * norm h) = suminf f * norm h
   by (rule suminf-mult2 [symmetric])
 finally show norm (suminf (g h)) ≤ suminf f * norm h .
qed
```

FIXME: Long proofs

```
lemma termdiffs-aux:
 fixes x :: 'a::{recpower,real-normed-field,banach}
 assumes 1: summable (λn. diffs (diffs c) n * K ^ n)
 assumes 2: norm x < norm K
 shows (λh. ∑ n. c n * (((x + h) ^ n - x ^ n) / h
          - of-nat n * x ^ (n - Suc 0))) -- 0 --> 0
proof -
 from dense [OF 2]
 obtain r where r1: norm x < r and r2: r < norm K by fast
 from norm-ge-zero r1 have r: 0 < r
```

```
    by (rule order-le-less-trans)
  hence r-neq-0: r ≠ 0 by simp
  show ?thesis
  proof (rule lemma-termdiff5)
    show 0 < r − norm x using r1 by simp
  next
    from r r2 have norm (of-real r::'a) < norm K
      by simp
    with 1 have summable (λn. norm (diffs (diffs c) n * (of-real r ^ n)))
      by (rule powser-insidea)
    hence summable (λn. diffs (diffs (λn. norm (c n))) n * r ^ n)
      using r
      by (simp add: diffs-def norm-mult norm-power del: of-nat-Suc)
    hence summable (λn. of-nat n * diffs (λn. norm (c n)) n * r ^ (n − Suc 0))
      by (rule diffs-equiv [THEN sums-summable])
    also have (λn. of-nat n * diffs (λn. norm (c n)) n * r ^ (n − Suc 0))
      = (λn. diffs (%m. of-nat (m − Suc 0) * norm (c m) * inverse r) n * (r ^
n))
      apply (rule ext)
      apply (simp add: diffs-def)
      apply (case-tac n, simp-all add: r-neq-0)
      done
    finally have summable
      (λn. of-nat n * (of-nat (n − Suc 0) * norm (c n) * inverse r) * r ^ (n −
Suc 0))
      by (rule diffs-equiv [THEN sums-summable])
    also have
      (λn. of-nat n * (of-nat (n − Suc 0) * norm (c n) * inverse r) *
          r ^ (n − Suc 0)) =
       (λn. norm (c n) * of-nat n * of-nat (n − Suc 0) * r ^ (n − 2))
      apply (rule ext)
      apply (case-tac n, simp)
      apply (case-tac nat, simp)
      apply (simp add: r-neq-0)
      done
    finally show
      summable (λn. norm (c n) * of-nat n * of-nat (n − Suc 0) * r ^ (n − 2)) .
  next
    fix h::'a and n::nat
    assume h: h ≠ 0
    assume norm h < r − norm x
    hence norm x + norm h < r by simp
    with norm-triangle-ineq have xh: norm (x + h) < r
      by (rule order-le-less-trans)
    show norm (c n * (((x + h) ^ n − x ^ n) / h − of-nat n * x ^ (n − Suc 0)))
        ≤ norm (c n) * of-nat n * of-nat (n − Suc 0) * r ^ (n − 2) * norm h
      apply (simp only: norm-mult mult-assoc)
      apply (rule mult-left-mono [OF - norm-ge-zero])
      apply (simp (no-asm) add: mult-assoc [symmetric])
```

```
    apply (rule lemma-termdiff3)
    apply (rule h)
    apply (rule r1 [THEN order-less-imp-le])
    apply (rule xh [THEN order-less-imp-le])
    done
  qed
qed

lemma termdiffs:
  fixes K x :: 'a::{recpower,real-normed-field,banach}
  assumes 1: summable (λn. c n * K ^ n)
  assumes 2: summable (λn. (diffs c) n * K ^ n)
  assumes 3: summable (λn. (diffs (diffs c)) n * K ^ n)
  assumes 4: norm x < norm K
  shows DERIV (λx. ∑ n. c n * x ^ n) x :> (∑ n. (diffs c) n * x ^ n)
proof (simp add: deriv-def, rule LIM-zero-cancel)
  show (λh. (suminf (λn. c n * (x + h) ^ n) - suminf (λn. c n * x ^ n)) / h
          - suminf (λn. diffs c n * x ^ n)) -- 0 --> 0
  proof (rule LIM-equal2)
    show 0 < norm K - norm x by (simp add: less-diff-eq 4)
  next
    fix h :: 'a
    assume h ≠ 0
    assume norm (h - 0) < norm K - norm x
    hence norm x + norm h < norm K by simp
    hence 5: norm (x + h) < norm K
      by (rule norm-triangle-ineq [THEN order-le-less-trans])
    have A: summable (λn. c n * x ^ n)
      by (rule powser-inside [OF 1 4])
    have B: summable (λn. c n * (x + h) ^ n)
      by (rule powser-inside [OF 1 5])
    have C: summable (λn. diffs c n * x ^ n)
      by (rule powser-inside [OF 2 4])
    show ((∑ n. c n * (x + h) ^ n) - (∑ n. c n * x ^ n)) / h
          - (∑ n. diffs c n * x ^ n) =
        (∑ n. c n * (((x + h) ^ n - x ^ n) / h - of-nat n * x ^ (n - Suc 0)))
      apply (subst sums-unique [OF diffs-equiv [OF C]])
      apply (subst suminf-diff [OF B A])
      apply (subst suminf-divide [symmetric])
      apply (rule summable-diff [OF B A])
      apply (subst suminf-diff)
      apply (rule summable-divide)
      apply (rule summable-diff [OF B A])
      apply (rule sums-summable [OF diffs-equiv [OF C]])
      apply (rule-tac f=suminf in arg-cong)
      apply (rule ext)
      apply (simp add: ring-simps)
      done
  next
```

```
    show (λh. ∑n. c n * (((x + h) ^ n − x ^ n) / h −
            of-nat n * x ^ (n − Suc 0))) −− 0 −−> 0
      by (rule termdiffs-aux [OF 3 4])
  qed
qed
```

22.3 Exponential Function

```
definition
  exp :: 'a ⇒ 'a::{recpower,real-normed-field,banach} where
  exp x = (∑n. x ^ n /R real (fact n))

definition
  sin :: real => real where
  sin x = (∑n. (if even(n) then 0 else
          (−1 ^ ((n − Suc 0) div 2))/(real (fact n))) * x ^ n)

definition
  cos :: real => real where
  cos x = (∑n. (if even(n) then (−1 ^ (n div 2))/(real (fact n))
                           else 0) * x ^ n)

lemma summable-exp-generic:
  fixes x :: 'a::{real-normed-algebra-1,recpower,banach}
  defines S-def: S ≡ λn. x ^ n /R real (fact n)
  shows summable S
proof −
  have S-Suc: ⋀n. S (Suc n) = (x * S n) /R real (Suc n)
    unfolding S-def by (simp add: power-Suc del: mult-Suc)
  obtain r :: real where r0: 0 < r and r1: r < 1
    using dense [OF zero-less-one] by fast
  obtain N :: nat where N: norm x < real N * r
    using reals-Archimedean3 [OF r0] by fast
  from r1 show ?thesis
  proof (rule ratio-test [rule-format])
    fix n :: nat
    assume n: N ≤ n
    have norm x ≤ real N * r
      using N by (rule order-less-imp-le)
    also have real N * r ≤ real (Suc n) * r
      using r0 n by (simp add: mult-right-mono)
    finally have norm x * norm (S n) ≤ real (Suc n) * r * norm (S n)
      using norm-ge-zero by (rule mult-right-mono)
    hence norm (x * S n) ≤ real (Suc n) * r * norm (S n)
      by (rule order-trans [OF norm-mult-ineq])
    hence norm (x * S n) / real (Suc n) ≤ r * norm (S n)
      by (simp add: pos-divide-le-eq mult-ac)
    thus norm (S (Suc n)) ≤ r * norm (S n)
      by (simp add: S-Suc norm-scaleR inverse-eq-divide)
```

```
  qed
qed

lemma summable-norm-exp:
  fixes x :: 'a::{real-normed-algebra-1,recpower,banach}
  shows summable (λn. norm (x ^ n /R real (fact n)))
proof (rule summable-norm-comparison-test [OF exI, rule-format])
  show summable (λn. norm x ^ n /R real (fact n))
    by (rule summable-exp-generic)
next
  fix n show norm (x ^ n /R real (fact n)) ≤ norm x ^ n /R real (fact n)
    by (simp add: norm-scaleR norm-power-ineq)
qed

lemma summable-exp: summable (%n. inverse (real (fact n)) * x ^ n)
by (insert summable-exp-generic [where x=x], simp)

lemma summable-sin:
     summable (%n.
          (if even n then 0
          else -1 ^ ((n - Suc 0) div 2)/(real (fact n))) *
               x ^ n)
apply (rule-tac g = (%n. inverse (real (fact n)) * |x| ^ n) in summable-comparison-test)
apply (rule-tac [2] summable-exp)
apply (rule-tac x = 0 in exI)
apply (auto simp add: divide-inverse abs-mult power-abs [symmetric] zero-le-mult-iff)
done

lemma summable-cos:
      summable (%n.
          (if even n then
          -1 ^ (n div 2)/(real (fact n)) else 0) * x ^ n)
apply (rule-tac g = (%n. inverse (real (fact n)) * |x| ^ n) in summable-comparison-test)
apply (rule-tac [2] summable-exp)
apply (rule-tac x = 0 in exI)
apply (auto simp add: divide-inverse abs-mult power-abs [symmetric] zero-le-mult-iff)
done

lemma lemma-STAR-sin:
     (if even n then 0
       else -1 ^ ((n - Suc 0) div 2)/(real (fact n))) * 0 ^ n = 0
by (induct n, auto)

lemma lemma-STAR-cos:
     0 < n -->
      -1 ^ (n div 2)/(real (fact n)) * 0 ^ n = 0
by (induct n, auto)

lemma lemma-STAR-cos1:
```

```
    0 < n −−>
     (−1) ^ (n div 2)/(real (fact n)) * 0 ^ n = 0
by (induct n, auto)

lemma lemma-STAR-cos2:
  (∑ n=1..<n. if even n then −1 ^ (n div 2)/(real (fact n)) *  0 ^ n
                        else 0) = 0
apply (induct n)
apply (case-tac [2] n, auto)
done

lemma exp-converges: (λn. x ^ n /R real (fact n)) sums exp x
unfolding exp-def by (rule summable-exp-generic [THEN summable-sums])

lemma sin-converges:
     (%n. (if even n then 0
          else −1 ^ ((n − Suc 0) div 2)/(real (fact n))) *
               x ^ n) sums sin(x)
unfolding sin-def by (rule summable-sin [THEN summable-sums])

lemma cos-converges:
     (%n. (if even n then
          −1 ^ (n div 2)/(real (fact n))
          else 0) * x ^ n) sums cos(x)
unfolding cos-def by (rule summable-cos [THEN summable-sums])
```

22.4 Formal Derivatives of Exp, Sin, and Cos Series

```
lemma exp-fdiffs:
     diffs (%n. inverse(real (fact n))) = (%n. inverse(real (fact n)))
by (simp add: diffs-def mult-assoc [symmetric] real-of-nat-def of-nat-mult
        del: mult-Suc of-nat-Suc)

lemma diffs-of-real: diffs (λn. of-real (f n)) = (λn. of-real (diffs f n))
by (simp add: diffs-def)

lemma sin-fdiffs:
     diffs(%n. if even n then 0
          else −1 ^ ((n − Suc 0) div 2)/(real (fact n)))
      = (%n. if even n then
                −1 ^ (n div 2)/(real (fact n))
             else 0)
by (auto intro!: ext
        simp add: diffs-def divide-inverse real-of-nat-def of-nat-mult
        simp del: mult-Suc of-nat-Suc)

lemma sin-fdiffs2:
     diffs(%n. if even n then 0
          else −1 ^ ((n − Suc 0) div 2)/(real (fact n))) n
```

```
      = (if even n then
              −1 ^ (n div 2)/(real (fact n))
            else 0)
by (simp only: sin-fdiffs)

lemma cos-fdiffs:
     diffs(%n. if even n then
              −1 ^ (n div 2)/(real (fact n)) else 0)
      = (%n. − (if even n then 0
         else −1 ^ ((n − Suc 0)div 2)/(real (fact n))))
by (auto intro!: ext
          simp add: diffs-def divide-inverse odd-Suc-mult-two-ex real-of-nat-def
of-nat-mult
        simp del: mult-Suc of-nat-Suc)


lemma cos-fdiffs2:
     diffs(%n. if even n then
              −1 ^ (n div 2)/(real (fact n)) else 0) n
      = − (if even n then 0
         else −1 ^ ((n − Suc 0)div 2)/(real (fact n)))
by (simp only: cos-fdiffs)
```

Now at last we can get the derivatives of exp, sin and cos

```
lemma lemma-sin-minus:
    − sin x = (∑ n. − ((if even n then 0
                else −1 ^ ((n − Suc 0) div 2)/(real (fact n))) ∗ x ^ n))
by (auto intro!: sums-unique sums-minus sin-converges)

lemma lemma-exp-ext: exp = (λx. ∑ n. x ^ n /R real (fact n))
by (auto intro!: ext simp add: exp-def)

lemma DERIV-exp [simp]: DERIV exp x :> exp(x)
apply (simp add: exp-def)
apply (subst lemma-exp-ext)
apply (subgoal-tac DERIV (λu. ∑ n. of-real (inverse (real (fact n))) ∗ u ^ n) x
:> (∑ n. diffs (λn. of-real (inverse (real (fact n)))) n ∗ x ^ n))
apply (rule-tac [2] K = of-real (1 + norm x) in termdiffs)
apply (simp-all only: diffs-of-real scaleR-conv-of-real exp-fdiffs)
apply (rule exp-converges [THEN sums-summable, unfolded scaleR-conv-of-real])+
apply (simp del: of-real-add)
done

lemma lemma-sin-ext:
    sin = (%x. ∑ n.
              (if even n then 0
                  else −1 ^ ((n − Suc 0) div 2)/(real (fact n))) ∗
              x ^ n)
by (auto intro!: ext simp add: sin-def)
```

lemma *lemma-cos-ext*:
  $cos = (\%x.\ \sum n.$
    $(if\ even\ n\ then\ -1\ \hat{}\ (n\ div\ 2)/(real\ (fact\ n))\ else\ 0) *$
    $x\ \hat{}\ n)$
by (*auto intro*!: *ext simp add*: *cos-def*)

lemma *DERIV-sin* [*simp*]: *DERIV sin x :> cos*(*x*)
apply (*simp add*: *cos-def*)
apply (*subst lemma-sin-ext*)
apply (*auto simp add*: *sin-fdiffs2* [*symmetric*])
apply (*rule-tac K = 1 + |x|* **in** *termdiffs*)
apply (*auto intro*: *sin-converges cos-converges sums-summable intro*!: *sums-minus* [*THEN sums-summable*] *simp add*: *cos-fdiffs sin-fdiffs*)
done

lemma *DERIV-cos* [*simp*]: *DERIV cos x :> −sin*(*x*)
apply (*subst lemma-cos-ext*)
apply (*auto simp add*: *lemma-sin-minus cos-fdiffs2* [*symmetric*] *minus-mult-left*)
apply (*rule-tac K = 1 + |x|* **in** *termdiffs*)
apply (*auto intro*: *sin-converges cos-converges sums-summable intro*!: *sums-minus* [*THEN sums-summable*] *simp add*: *cos-fdiffs sin-fdiffs diffs-minus*)
done

lemma *isCont-exp* [*simp*]: *isCont exp x*
by (*rule DERIV-exp* [*THEN DERIV-isCont*])

lemma *isCont-sin* [*simp*]: *isCont sin x*
by (*rule DERIV-sin* [*THEN DERIV-isCont*])

lemma *isCont-cos* [*simp*]: *isCont cos x*
by (*rule DERIV-cos* [*THEN DERIV-isCont*])

22.5 Properties of the Exponential Function

lemma *powser-zero*:
  fixes *f* :: *nat* ⇒ *'a*::{*real-normed-algebra-1*,*recpower*}
  shows $(\sum n.\ f\ n * 0\ \hat{}\ n) = f\ 0$
proof −
  have $(\sum n = 0..<1.\ f\ n * 0\ \hat{}\ n) = (\sum n.\ f\ n * 0\ \hat{}\ n)$
    by (*rule sums-unique* [*OF series-zero*], *simp add*: *power-0-left*)
  thus *?thesis* **by** *simp*
qed

lemma *exp-zero* [*simp*]: *exp 0 = 1*
unfolding *exp-def* **by** (*simp add*: *scaleR-conv-of-real powser-zero*)

lemma *setsum-head2*:
  $m \leq n \Longrightarrow setsum\ f\ \{m..n\} = f\ m + setsum\ f\ \{Suc\ m..n\}$

by (*simp add*: *setsum-head atLeastSucAtMost-greaterThanAtMost*)

lemma *setsum-cl-ivl-Suc2*:
$(\sum i{=}m..Suc\ n.\ f\ i) = (if\ Suc\ n < m\ then\ 0\ else\ f\ m + (\sum i{=}m..n.\ f\ (Suc\ i)))$
by (*simp add*: *setsum-head2 setsum-shift-bounds-cl-Suc-ivl*
del: *setsum-cl-ivl-Suc*)

lemma *exp-series-add*:
fixes *x y* :: *'a*::{*real-field*,*recpower*}
defines *S-def*: $S \equiv \lambda x\ n.\ x\ \hat{}\ n\ /_R\ real\ (fact\ n)$
shows $S\ (x + y)\ n = (\sum i{=}0..n.\ S\ x\ i * S\ y\ (n - i))$
proof (*induct n*)
case *0*
show *?case*
unfolding *S-def* **by** *simp*
next
case (*Suc n*)
have *S-Suc*: $\bigwedge x\ n.\ S\ x\ (Suc\ n) = (x * S\ x\ n)\ /_R\ real\ (Suc\ n)$
unfolding *S-def* **by** (*simp add*: *power-Suc del*: *mult-Suc*)
hence *times-S*: $\bigwedge x\ n.\ x * S\ x\ n = real\ (Suc\ n) *_R S\ x\ (Suc\ n)$
by *simp*

have $real\ (Suc\ n) *_R S\ (x + y)\ (Suc\ n) = (x + y) * S\ (x + y)\ n$
by (*simp only*: *times-S*)
also have $\ldots = (x + y) * (\sum i{=}0..n.\ S\ x\ i * S\ y\ (n{-}i))$
by (*simp only*: *Suc*)
also have $\ldots = x * (\sum i{=}0..n.\ S\ x\ i * S\ y\ (n{-}i))$
$+\ y * (\sum i{=}0..n.\ S\ x\ i * S\ y\ (n{-}i))$
by (*rule left-distrib*)
also have $\ldots = (\sum i{=}0..n.\ (x * S\ x\ i) * S\ y\ (n{-}i))$
$+ (\sum i{=}0..n.\ S\ x\ i * (y * S\ y\ (n{-}i)))$
by (*simp only*: *setsum-right-distrib mult-ac*)
also have $\ldots = (\sum i{=}0..n.\ real\ (Suc\ i) *_R (S\ x\ (Suc\ i) * S\ y\ (n{-}i)))$
$+ (\sum i{=}0..n.\ real\ (Suc\ n{-}i) *_R (S\ x\ i * S\ y\ (Suc\ n{-}i)))$
by (*simp add*: *times-S Suc-diff-le*)
also have $(\sum i{=}0..n.\ real\ (Suc\ i) *_R (S\ x\ (Suc\ i) * S\ y\ (n{-}i))) =$
$(\sum i{=}0..Suc\ n.\ real\ i *_R (S\ x\ i * S\ y\ (Suc\ n{-}i)))$
by (*subst setsum-cl-ivl-Suc2*, *simp*)
also have $(\sum i{=}0..n.\ real\ (Suc\ n{-}i) *_R (S\ x\ i * S\ y\ (Suc\ n{-}i))) =$
$(\sum i{=}0..Suc\ n.\ real\ (Suc\ n{-}i) *_R (S\ x\ i * S\ y\ (Suc\ n{-}i)))$
by (*subst setsum-cl-ivl-Suc*, *simp*)
also have $(\sum i{=}0..Suc\ n.\ real\ i *_R (S\ x\ i * S\ y\ (Suc\ n{-}i))) +$
$(\sum i{=}0..Suc\ n.\ real\ (Suc\ n{-}i) *_R (S\ x\ i * S\ y\ (Suc\ n{-}i))) =$
$(\sum i{=}0..Suc\ n.\ real\ (Suc\ n) *_R (S\ x\ i * S\ y\ (Suc\ n{-}i)))$
by (*simp only*: *setsum-addf* [*symmetric*] *scaleR-left-distrib* [*symmetric*]
real-of-nat-add [*symmetric*], *simp*)
also have $\ldots = real\ (Suc\ n) *_R (\sum i{=}0..Suc\ n.\ S\ x\ i * S\ y\ (Suc\ n{-}i))$
by (*simp only*: *scaleR-right.setsum*)
finally show

```
    S (x + y) (Suc n) = (∑ i=0..Suc n. S x i * S y (Suc n - i))
    by (simp add: scaleR-cancel-left del: setsum-cl-ivl-Suc)
qed

lemma exp-add: exp (x + y) = exp x * exp y
unfolding exp-def
by (simp only: Cauchy-product summable-norm-exp exp-series-add)

lemma exp-of-real: exp (of-real x) = of-real (exp x)
unfolding exp-def
apply (subst of-real.suminf)
apply (rule summable-exp-generic)
apply (simp add: scaleR-conv-of-real)
done

lemma exp-ge-add-one-self-aux: 0 ≤ (x::real) ==> (1 + x) ≤ exp(x)
apply (drule order-le-imp-less-or-eq, auto)
apply (simp add: exp-def)
apply (rule real-le-trans)
apply (rule-tac [2] n = 2 and f = (%n. inverse (real (fact n)) * x ^ n) in
series-pos-le)
apply (auto intro: summable-exp simp add: numeral-2-eq-2 zero-le-mult-iff)
done

lemma exp-gt-one [simp]: 0 < (x::real) ==> 1 < exp x
apply (rule order-less-le-trans)
apply (rule-tac [2] exp-ge-add-one-self-aux, auto)
done

lemma DERIV-exp-add-const: DERIV (%x. exp (x + y)) x :> exp(x + y)
proof -
  have DERIV (exp ∘ (λx. x + y)) x :> exp (x + y) * (1+0)
   by (fast intro: DERIV-chain DERIV-add DERIV-exp DERIV-ident DERIV-const)

  thus ?thesis by (simp add: o-def)
qed

lemma DERIV-exp-minus [simp]: DERIV (%x. exp (-x)) x :> - exp(-x)
proof -
  have DERIV (exp ∘ uminus) x :> exp (- x) * - 1
    by (fast intro: DERIV-chain DERIV-minus DERIV-exp DERIV-ident)
  thus ?thesis by (simp add: o-def)
qed

lemma DERIV-exp-exp-zero [simp]: DERIV (%x. exp (x + y) * exp (- x)) x :>
0
proof -
  have DERIV (λx. exp (x + y) * exp (- x)) x
      :> exp (x + y) * exp (- x) + - exp (- x) * exp (x + y)
```

by (*fast intro*: *DERIV-exp-add-const DERIV-exp-minus DERIV-mult*)
thus *?thesis* **by** (*simp add*: *mult-commute*)
qed

lemma *exp-add-mult-minus* [*simp*]: *exp*(*x* + *y*)∗*exp*(−*x*) = *exp*(*y*::*real*)
proof −
have ∀ *x*. *DERIV* (%*x*. *exp* (*x* + *y*) ∗ *exp* (− *x*)) *x* :> *0* **by** *simp*
hence *exp* (*x* + *y*) ∗ *exp* (− *x*) = *exp* (*0* + *y*) ∗ *exp* (− *0*)
by (*rule DERIV-isconst-all*)
thus *?thesis* **by** *simp*
qed

lemma *exp-mult-minus* [*simp*]: *exp x* ∗ *exp*(−*x*) = *1*
by (*simp add*: *exp-add* [*symmetric*])

lemma *exp-mult-minus2* [*simp*]: *exp*(−*x*)∗*exp*(*x*) = *1*
by (*simp add*: *mult-commute*)

lemma *exp-minus*: *exp*(−*x*) = *inverse*(*exp*(*x*))
by (*auto intro*: *inverse-unique* [*symmetric*])

Proof: because every exponential can be seen as a square.

lemma *exp-ge-zero* [*simp*]: *0* ≤ *exp* (*x*::*real*)
apply (*rule-tac t* = *x* **in** *real-sum-of-halves* [*THEN subst*])
apply (*subst exp-add*, *auto*)
done

lemma *exp-not-eq-zero* [*simp*]: *exp x* ≠ *0*
apply (*cut-tac x* = *x* **in** *exp-mult-minus2*)
apply (*auto simp del*: *exp-mult-minus2*)
done

lemma *exp-gt-zero* [*simp*]: *0* < *exp* (*x*::*real*)
by (*simp add*: *order-less-le*)

lemma *inv-exp-gt-zero* [*simp*]: *0* < *inverse*(*exp x*::*real*)
by (*auto intro*: *positive-imp-inverse-positive*)

lemma *abs-exp-cancel* [*simp*]: |*exp x*::*real*| = *exp x*
by *auto*

lemma *exp-real-of-nat-mult*: *exp*(*real n* ∗ *x*) = *exp*(*x*) ^ *n*
apply (*induct n*)
apply (*auto simp add*: *real-of-nat-Suc right-distrib exp-add mult-commute*)
done

lemma *exp-diff*: *exp*(*x* − *y*) = *exp*(*x*)/(*exp y*)
apply (*simp add*: *diff-minus divide-inverse*)

apply (*simp* (*no-asm*) *add*: *exp-add exp-minus*)
done

lemma *exp-less-mono*:
  fixes *x y* :: *real*
  assumes *xy*: *x* < *y* **shows** *exp x* < *exp y*
proof −
  from *xy* **have** *1* < *exp* (*y* + − *x*)
    by (*rule real-less-sum-gt-zero* [*THEN exp-gt-one*])
  hence *exp x* ∗ *inverse* (*exp x*) < *exp y* ∗ *inverse* (*exp x*)
    by (*auto simp add*: *exp-add exp-minus*)
  thus *?thesis*
    by (*simp add*: *divide-inverse* [*symmetric*] *pos-less-divide-eq*
          del: *divide-self-if*)
qed

lemma *exp-less-cancel*: *exp* (*x*::*real*) < *exp y* ==> *x* < *y*
apply (*simp add*: *linorder-not-le* [*symmetric*])
apply (*auto simp add*: *order-le-less exp-less-mono*)
done

lemma *exp-less-cancel-iff* [*iff*]: (*exp*(*x*::*real*) < *exp*(*y*)) = (*x* < *y*)
by (*auto intro*: *exp-less-mono exp-less-cancel*)

lemma *exp-le-cancel-iff* [*iff*]: (*exp*(*x*::*real*) ≤ *exp*(*y*)) = (*x* ≤ *y*)
by (*auto simp add*: *linorder-not-less* [*symmetric*])

lemma *exp-inj-iff* [*iff*]: (*exp* (*x*::*real*) = *exp y*) = (*x* = *y*)
by (*simp add*: *order-eq-iff*)

lemma *lemma-exp-total*: *1* ≤ *y* ==> ∃ *x*. *0* ≤ *x* & *x* ≤ *y* − *1* & *exp*(*x*::*real*) = *y*
apply (*rule IVT*)
apply (*auto intro*: *isCont-exp simp add*: *le-diff-eq*)
apply (*subgoal-tac 1* + (*y* − *1*) ≤ *exp* (*y* − *1*))
apply *simp*
apply (*rule exp-ge-add-one-self-aux*, *simp*)
done

lemma *exp-total*: *0* < (*y*::*real*) ==> ∃ *x*. *exp x* = *y*
apply (*rule-tac x* = *1* **and** *y* = *y* **in** *linorder-cases*)
apply (*drule order-less-imp-le* [*THEN lemma-exp-total*])
apply (*rule-tac* [*2*] *x* = *0* **in** *exI*)
apply (*frule-tac* [*3*] *real-inverse-gt-one*)
apply (*drule-tac* [*4*] *order-less-imp-le* [*THEN lemma-exp-total*], *auto*)
apply (*rule-tac x* = −*x* **in** *exI*)
apply (*simp add*: *exp-minus*)
done

22.6 Properties of the Logarithmic Function

definition
ln :: *real* => *real* **where**
ln x = (*THE u*. *exp u* = *x*)

lemma *ln-exp* [*simp*]: *ln* (*exp x*) = *x*
by (*simp add*: *ln-def*)

lemma *exp-ln* [*simp*]: *0* < *x* ⟹ *exp* (*ln x*) = *x*
by (*auto dest*: *exp-total*)

lemma *exp-ln-iff* [*simp*]: (*exp* (*ln x*) = *x*) = (*0* < *x*)
apply (*auto dest*: *exp-total*)
apply (*erule subst*, *simp*)
done

lemma *ln-mult*: [| *0* < *x*; *0* < *y* |] ==> *ln*(*x* * *y*) = *ln*(*x*) + *ln*(*y*)
apply (*rule exp-inj-iff* [*THEN iffD1*])
apply (*simp add*: *exp-add exp-ln mult-pos-pos*)
done

lemma *ln-inj-iff*[*simp*]: [| *0* < *x*; *0* < *y* |] ==> (*ln x* = *ln y*) = (*x* = *y*)
apply (*simp only*: *exp-ln-iff* [*symmetric*])
apply (*erule subst*)+
apply *simp*
done

lemma *ln-one*[*simp*]: *ln 1* = *0*
by (*rule exp-inj-iff* [*THEN iffD1*], *auto*)

lemma *ln-inverse*: *0* < *x* ==> *ln*(*inverse x*) = − *ln x*
apply (*rule-tac a1* = *ln x* **in** *add-left-cancel* [*THEN iffD1*])
apply (*auto simp add*: *positive-imp-inverse-positive ln-mult* [*symmetric*])
done

lemma *ln-div*:
[|*0* < *x*; *0* < *y*|] ==> *ln*(*x*/*y*) = *ln x* − *ln y*
apply (*simp add*: *divide-inverse*)
apply (*auto simp add*: *positive-imp-inverse-positive ln-mult ln-inverse*)
done

lemma *ln-less-cancel-iff*[*simp*]: [| *0* < *x*; *0* < *y*|] ==> (*ln x* < *ln y*) = (*x* < *y*)
apply (*simp only*: *exp-ln-iff* [*symmetric*])
apply (*erule subst*)+
apply *simp*
done

lemma *ln-le-cancel-iff*[*simp*]: [| *0* < *x*; *0* < *y*|] ==> (*ln x* ≤ *ln y*) = (*x* ≤ *y*)
by (*auto simp add*: *linorder-not-less* [*symmetric*])

lemma *ln-realpow*: $0 < x \Longrightarrow ln(x \hat{} n) = real\ n * ln(x)$
by (*auto dest*!: *exp-total simp add*: *exp-real-of-nat-mult* [*symmetric*])

lemma *ln-add-one-self-le-self* [*simp*]: $0 \leq x \Longrightarrow ln(1 + x) \leq x$
apply (*rule ln-exp* [*THEN subst*])
apply (*rule ln-le-cancel-iff* [*THEN iffD2*])
apply (*auto simp add*: *exp-ge-add-one-self-aux*)
done

lemma *ln-less-self* [*simp*]: $0 < x \Longrightarrow ln\ x < x$
apply (*rule order-less-le-trans*)
apply (*rule-tac* [2] *ln-add-one-self-le-self*)
apply (*rule ln-less-cancel-iff* [*THEN iffD2*], *auto*)
done

lemma *ln-ge-zero* [*simp*]:
  assumes *x*: $1 \leq x$ **shows** $0 \leq ln\ x$
proof −
  have $0 < x$ **using** *x* **by** *arith*
  hence $exp\ 0 \leq exp\ (ln\ x)$
    by (*simp add*: *x*)
  thus *?thesis* **by** (*simp only*: *exp-le-cancel-iff*)
qed

lemma *ln-ge-zero-imp-ge-one*:
  assumes *ln*: $0 \leq ln\ x$
      and *x*: $0 < x$
  shows $1 \leq x$
proof −
  from *ln* **have** $ln\ 1 \leq ln\ x$ **by** *simp*
  thus *?thesis* **by** (*simp add*: *x del*: *ln-one*)
qed

lemma *ln-ge-zero-iff* [*simp*]: $0 < x \Longrightarrow (0 \leq ln\ x) = (1 \leq x)$
by (*blast intro*: *ln-ge-zero ln-ge-zero-imp-ge-one*)

lemma *ln-less-zero-iff* [*simp*]: $0 < x \Longrightarrow (ln\ x < 0) = (x < 1)$
by (*insert ln-ge-zero-iff* [*of x*], *arith*)

lemma *ln-gt-zero*:
  assumes *x*: $1 < x$ **shows** $0 < ln\ x$
proof −
  have $0 < x$ **using** *x* **by** *arith*
  hence $exp\ 0 < exp\ (ln\ x)$ **by** (*simp add*: *x*)
  thus *?thesis* **by** (*simp only*: *exp-less-cancel-iff*)
qed

lemma *ln-gt-zero-imp-gt-one*:

assumes *ln*: *0 < ln x*
and *x*: *0 < x*
shows *1 < x*
proof −
from *ln* **have** *ln 1 < ln x* **by** *simp*
thus *?thesis* **by** (*simp add*: *x del*: *ln-one*)
qed

lemma *ln-gt-zero-iff* [*simp*]: *0 < x ==> (0 < ln x) = (1 < x)*
by (*blast intro*: *ln-gt-zero ln-gt-zero-imp-gt-one*)

lemma *ln-eq-zero-iff* [*simp*]: *0 < x ==> (ln x = 0) = (x = 1)*
by (*insert ln-less-zero-iff* [*of x*] *ln-gt-zero-iff* [*of x*], *arith*)

lemma *ln-less-zero*: [| *0 < x*; *x < 1* |] *==> ln x < 0*
by *simp*

lemma *exp-ln-eq*: *exp u = x ==> ln x = u*
by *auto*

lemma *isCont-ln*: *0 < x ⟹ isCont ln x*
apply (*subgoal-tac isCont ln* (*exp* (*ln x*)), *simp*)
apply (*rule isCont-inverse-function* [**where** *f=exp*], *simp-all*)
done

lemma *DERIV-ln*: *0 < x ⟹ DERIV ln x :> inverse x*
apply (*rule DERIV-inverse-function* [**where** *f=exp* **and** *a=0* **and** *b=x+1*])
apply (*erule lemma-DERIV-subst* [*OF DERIV-exp exp-ln*])
apply (*simp-all add*: *abs-if isCont-ln*)
done

22.7 Basic Properties of the Trigonometric Functions

lemma *sin-zero* [*simp*]: *sin 0 = 0*
unfolding *sin-def* **by** (*simp add*: *powser-zero*)

lemma *cos-zero* [*simp*]: *cos 0 = 1*
unfolding *cos-def* **by** (*simp add*: *powser-zero*)

lemma *DERIV-sin-sin-mult* [*simp*]:
DERIV (%*x*. *sin*(*x*)∗*sin*(*x*)) *x :> cos*(*x*) ∗ *sin*(*x*) + *cos*(*x*) ∗ *sin*(*x*)
by (*rule DERIV-mult*, *auto*)

lemma *DERIV-sin-sin-mult2* [*simp*]:
DERIV (%*x*. *sin*(*x*)∗*sin*(*x*)) *x :> 2* ∗ *cos*(*x*) ∗ *sin*(*x*)
apply (*cut-tac x = x* **in** *DERIV-sin-sin-mult*)
apply (*auto simp add*: *mult-assoc*)
done

lemma *DERIV-sin-realpow2* [*simp*]:
  DERIV (%*x*. (*sin x*)^2) *x* :> *cos*(*x*) * *sin*(*x*) + *cos*(*x*) * *sin*(*x*)
by (*auto simp add*: *numeral-2-eq-2 real-mult-assoc* [*symmetric*])

lemma *DERIV-sin-realpow2a* [*simp*]:
  DERIV (%*x*. (*sin x*)^2) *x* :> *2* * *cos*(*x*) * *sin*(*x*)
by (*auto simp add*: *numeral-2-eq-2*)

lemma *DERIV-cos-cos-mult* [*simp*]:
  DERIV (%*x*. *cos*(*x*)**cos*(*x*)) *x* :> −*sin*(*x*) * *cos*(*x*) + −*sin*(*x*) * *cos*(*x*)
by (*rule DERIV-mult*, *auto*)

lemma *DERIV-cos-cos-mult2* [*simp*]:
  DERIV (%*x*. *cos*(*x*)**cos*(*x*)) *x* :> −*2* * *cos*(*x*) * *sin*(*x*)
apply (*cut-tac x* = *x* **in** *DERIV-cos-cos-mult*)
apply (*auto simp add*: *mult-ac*)
done

lemma *DERIV-cos-realpow2* [*simp*]:
  DERIV (%*x*. (*cos x*)^2) *x* :> −*sin*(*x*) * *cos*(*x*) + −*sin*(*x*) * *cos*(*x*)
by (*auto simp add*: *numeral-2-eq-2 real-mult-assoc* [*symmetric*])

lemma *DERIV-cos-realpow2a* [*simp*]:
  DERIV (%*x*. (*cos x*)^2) *x* :> −*2* * *cos*(*x*) * *sin*(*x*)
by (*auto simp add*: *numeral-2-eq-2*)

lemma *lemma-DERIV-subst*: [| *DERIV f x* :> *D*; *D* = *E* |] ==> *DERIV f x* :> *E*
by *auto*

lemma *DERIV-cos-realpow2b*: *DERIV* (%*x*. (*cos x*)^2) *x* :> −(*2* * *cos*(*x*) * *sin*(*x*))
apply (*rule lemma-DERIV-subst*)
apply (*rule DERIV-cos-realpow2a*, *auto*)
done

lemma *DERIV-cos-cos-mult3* [*simp*]:
  DERIV (%*x*. *cos*(*x*)**cos*(*x*)) *x* :> −(*2* * *cos*(*x*) * *sin*(*x*))
apply (*rule lemma-DERIV-subst*)
apply (*rule DERIV-cos-cos-mult2*, *auto*)
done

lemma *DERIV-sin-circle-all*:
  ∀ *x*. *DERIV* (%*x*. (*sin x*)^2 + (*cos x*)^2) *x* :>
      (*2***cos*(*x*)***sin*(*x*) − *2***cos*(*x*)***sin*(*x*))
apply (*simp only*: *diff-minus*, *safe*)
apply (*rule DERIV-add*)
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *DERIV-sin-circle-all-zero* [*simp*]:
  $\forall x.\ DERIV\ (\%x.\ (sin\ x)^2 + (cos\ x)^2)\ x :> 0$
by (*cut-tac DERIV-sin-circle-all*, *auto*)

lemma *sin-cos-squared-add* [*simp*]: $((sin\ x)^2) + ((cos\ x)^2) = 1$
apply (*cut-tac* $x = x$ **and** $y = 0$ **in** *DERIV-sin-circle-all-zero* [*THEN DERIV-isconst-all*])
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *sin-cos-squared-add2* [*simp*]: $((cos\ x)^2) + ((sin\ x)^2) = 1$
apply (*subst add-commute*)
apply (*simp* (*no-asm*) *del*: *realpow-Suc*)
done

lemma *sin-cos-squared-add3* [*simp*]: $cos\ x * cos\ x + sin\ x * sin\ x = 1$
apply (*cut-tac* $x = x$ **in** *sin-cos-squared-add2*)
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *sin-squared-eq*: $(sin\ x)^2 = 1 - (cos\ x)^2$
apply (*rule-tac a1* $= (cos\ x)^2$ **in** *add-right-cancel* [*THEN iffD1*])
apply (*simp del*: *realpow-Suc*)
done

lemma *cos-squared-eq*: $(cos\ x)^2 = 1 - (sin\ x)^2$
apply (*rule-tac a1* $= (sin\ x)^2$ **in** *add-right-cancel* [*THEN iffD1*])
apply (*simp del*: *realpow-Suc*)
done

lemma *real-gt-one-ge-zero-add-less*: $[|\ 1 < x;\ 0 \leq y\ |] ==> 1 < x + (y::real)$
by *arith*

lemma *abs-sin-le-one* [*simp*]: $|sin\ x| \leq 1$
by (*rule power2-le-imp-le*, *simp-all add*: *sin-squared-eq*)

lemma *sin-ge-minus-one* [*simp*]: $-1 \leq sin\ x$
apply (*insert abs-sin-le-one* [*of x*])
apply (*simp add*: *abs-le-iff del*: *abs-sin-le-one*)
done

lemma *sin-le-one* [*simp*]: $sin\ x \leq 1$
apply (*insert abs-sin-le-one* [*of x*])
apply (*simp add*: *abs-le-iff del*: *abs-sin-le-one*)
done

lemma *abs-cos-le-one* [*simp*]: $|cos\ x| \leq 1$
by (*rule power2-le-imp-le*, *simp-all add*: *cos-squared-eq*)

lemma *cos-ge-minus-one* [*simp*]: $-1 \leq cos\ x$
apply (*insert abs-cos-le-one* [*of x*])
apply (*simp add*: *abs-le-iff del*: *abs-cos-le-one*)
done

lemma *cos-le-one* [*simp*]: $cos\ x \leq 1$
apply (*insert abs-cos-le-one* [*of x*])
apply (*simp add*: *abs-le-iff del*: *abs-cos-le-one*)
done

lemma *DERIV-fun-pow*: *DERIV g x :> m ==>*
DERIV (%*x*. (*g x*) ^ *n*) *x :> real n* * (*g x*) ^ (*n* − *1*) * *m*
apply (*rule lemma-DERIV-subst*)
apply (*rule-tac f* = (%*x*. *x* ^ *n*) **in** *DERIV-chain2*)
apply (*rule DERIV-pow*, *auto*)
done

lemma *DERIV-fun-exp*:
DERIV g x :> m ==> DERIV (%*x*. *exp*(*g x*)) *x :> exp*(*g x*) * *m*
apply (*rule lemma-DERIV-subst*)
apply (*rule-tac f* = *exp* **in** *DERIV-chain2*)
apply (*rule DERIV-exp*, *auto*)
done

lemma *DERIV-fun-sin*:
DERIV g x :> m ==> DERIV (%*x*. *sin*(*g x*)) *x :> cos*(*g x*) * *m*
apply (*rule lemma-DERIV-subst*)
apply (*rule-tac f* = *sin* **in** *DERIV-chain2*)
apply (*rule DERIV-sin*, *auto*)
done

lemma *DERIV-fun-cos*:
DERIV g x :> m ==> DERIV (%*x*. *cos*(*g x*)) *x :> −sin*(*g x*) * *m*
apply (*rule lemma-DERIV-subst*)
apply (*rule-tac f* = *cos* **in** *DERIV-chain2*)
apply (*rule DERIV-cos*, *auto*)
done

lemmas *DERIV-intros* = *DERIV-ident DERIV-const DERIV-cos DERIV-cmult*
DERIV-sin DERIV-exp DERIV-inverse DERIV-pow
DERIV-add DERIV-diff DERIV-mult DERIV-minus
DERIV-inverse-fun DERIV-quotient DERIV-fun-pow
DERIV-fun-exp DERIV-fun-sin DERIV-fun-cos

lemma *lemma-DERIV-sin-cos-add*:
∀ *x*.
DERIV (%*x*. (*sin* (*x* + *y*) − (*sin x* * *cos y* + *cos x* * *sin y*)) ^ *2* +
(*cos* (*x* + *y*) − (*cos x* * *cos y* − *sin x* * *sin y*)) ^ *2*) *x :> 0*

apply (*safe*, *rule lemma-DERIV-subst*)
apply (*best intro*!: *DERIV-intros intro*: *DERIV-chain2*)
— replaces the old *DERIV-tac*
apply (*auto simp add*: *diff-minus left-distrib right-distrib mult-ac add-ac*)
done

lemma *sin-cos-add* [*simp*]:
(*sin* (*x* + *y*) − (*sin x* ∗ *cos y* + *cos x* ∗ *sin y*)) ^ *2* +
(*cos* (*x* + *y*) − (*cos x* ∗ *cos y* − *sin x* ∗ *sin y*)) ^ *2* = *0*
apply (*cut-tac y* = *0* **and** *x* = *x* **and** *y7* = *y*
in *lemma-DERIV-sin-cos-add* [*THEN DERIV-isconst-all*])
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *sin-add*: *sin* (*x* + *y*) = *sin x* ∗ *cos y* + *cos x* ∗ *sin y*
apply (*cut-tac x* = *x* **and** *y* = *y* **in** *sin-cos-add*)
apply (*simp del*: *sin-cos-add*)
done

lemma *cos-add*: *cos* (*x* + *y*) = *cos x* ∗ *cos y* − *sin x* ∗ *sin y*
apply (*cut-tac x* = *x* **and** *y* = *y* **in** *sin-cos-add*)
apply (*simp del*: *sin-cos-add*)
done

lemma *lemma-DERIV-sin-cos-minus*:
∀ *x*. *DERIV* (%*x*. (*sin*(−*x*) + (*sin x*)) ^ *2* + (*cos*(−*x*) − (*cos x*)) ^ *2*) *x* :> *0*
apply (*safe*, *rule lemma-DERIV-subst*)
apply (*best intro*!: *DERIV-intros intro*: *DERIV-chain2*)
apply (*auto simp add*: *diff-minus left-distrib right-distrib mult-ac add-ac*)
done

lemma *sin-cos-minus* [*simp*]:
(*sin*(−*x*) + (*sin x*)) ^ *2* + (*cos*(−*x*) − (*cos x*)) ^ *2* = *0*
apply (*cut-tac y* = *0* **and** *x* = *x*
in *lemma-DERIV-sin-cos-minus* [*THEN DERIV-isconst-all*])
apply *simp*
done

lemma *sin-minus* [*simp*]: *sin* (−*x*) = −*sin*(*x*)
apply (*cut-tac x* = *x* **in** *sin-cos-minus*)
apply (*simp del*: *sin-cos-minus*)
done

lemma *cos-minus* [*simp*]: *cos* (−*x*) = *cos*(*x*)
apply (*cut-tac x* = *x* **in** *sin-cos-minus*)
apply (*simp del*: *sin-cos-minus*)
done

lemma *sin-diff*: *sin* (*x* − *y*) = *sin x* ∗ *cos y* − *cos x* ∗ *sin y*

by (*simp add*: *diff-minus sin-add*)

lemma *sin-diff2*: *sin* (*x* − *y*) = *cos y* ∗ *sin x* − *sin y* ∗ *cos x*
by (*simp add*: *sin-diff mult-commute*)

lemma *cos-diff*: *cos* (*x* − *y*) = *cos x* ∗ *cos y* + *sin x* ∗ *sin y*
by (*simp add*: *diff-minus cos-add*)

lemma *cos-diff2*: *cos* (*x* − *y*) = *cos y* ∗ *cos x* + *sin y* ∗ *sin x*
by (*simp add*: *cos-diff mult-commute*)

lemma *sin-double* [*simp*]: *sin*(2 ∗ *x*) = 2∗ *sin x* ∗ *cos x*
by (*cut-tac x* = *x* **and** *y* = *x* **in** *sin-add*, *auto*)

lemma *cos-double*: *cos*(2∗ *x*) = ((*cos x*)^2) − ((*sin x*)^2)
apply (*cut-tac x* = *x* **and** *y* = *x* **in** *cos-add*)
apply (*simp add*: *power2-eq-square*)
done

22.8 The Constant Pi

definition
pi :: *real* **where**
pi = 2 ∗ (*THE x*. 0 ≤ (*x*::*real*) & *x* ≤ 2 & *cos x* = 0)

Show that there's a least positive *x* with *cos x* = 0; hence define pi.

lemma *sin-paired*:
(%*n*. −1 ^ *n* /(*real* (*fact* (2 ∗ *n* + 1))) ∗ *x* ^ (2 ∗ *n* + 1))
sums sin x
proof −
have (λ*n*. ∑ *k* = *n* ∗ 2..<*n* ∗ 2 + 2.
(*if even k then* 0
else −1 ^ ((*k* − *Suc* 0) *div* 2) / *real* (*fact k*)) ∗
x ^ *k*)
sums sin x
unfolding *sin-def*
by (*rule sin-converges* [*THEN sums-summable*, *THEN sums-group*], *simp*)
thus *?thesis* **by** (*simp add*: *mult-ac*)
qed

lemma *sin-gt-zero*: [|0 < *x*; *x* < 2 |] ==> 0 < *sin x*
apply (*subgoal-tac*
(λ*n*. ∑ *k* = *n* ∗ 2..<*n* ∗ 2 + 2.
−1 ^ *k* / *real* (*fact* (2 ∗ *k* + 1)) ∗ *x* ^ (2 ∗ *k* + 1))
sums (∑ *n*. −1 ^ *n* / *real* (*fact* (2 ∗ *n* + 1)) ∗ *x* ^ (2 ∗ *n* + 1)))
prefer 2
apply (*rule sin-paired* [*THEN sums-summable*, *THEN sums-group*], *simp*)
apply (*rotate-tac* 2)

apply (*drule sin-paired* [*THEN sums-unique*, *THEN ssubst*])
apply (*auto simp del*: *fact-Suc realpow-Suc*)
apply (*frule sums-unique*)
apply (*auto simp del*: *fact-Suc realpow-Suc*)
apply (*rule-tac n1 = 0* **in** *series-pos-less* [*THEN* [*2*] *order-le-less-trans*])
apply (*auto simp del*: *fact-Suc realpow-Suc*)
apply (*erule sums-summable*)
apply (*case-tac m=0*)
apply (*simp* (*no-asm-simp*))
apply (*subgoal-tac 6 * (x * (x * x) / real* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc* (*Suc 0*)))))))
*< 6 * x*)
apply (*simp only*: *mult-less-cancel-left*, *simp*)
apply (*simp* (*no-asm-simp*) *add*: *numeral-2-eq-2* [*symmetric*] *mult-assoc* [*symmetric*])
apply (*subgoal-tac x*x < 2*3*, *simp*)
apply (*rule mult-strict-mono*)
apply (*auto simp add*: *real-0-less-add-iff real-of-nat-Suc simp del*: *fact-Suc*)
apply (*subst fact-Suc*)
apply (*subst fact-Suc*)
apply (*subst fact-Suc*)
apply (*subst fact-Suc*)
apply (*subst real-of-nat-mult*)
apply (*subst real-of-nat-mult*)
apply (*subst real-of-nat-mult*)
apply (*subst real-of-nat-mult*)
apply (*simp* (*no-asm*) *add*: *divide-inverse del*: *fact-Suc*)
apply (*auto simp add*: *mult-assoc* [*symmetric*] *simp del*: *fact-Suc*)
apply (*rule-tac c=real* (*Suc* (*Suc* (*4*m*))) **in** *mult-less-imp-less-right*)
apply (*auto simp add*: *mult-assoc simp del*: *fact-Suc*)
apply (*rule-tac c=real* (*Suc* (*Suc* (*Suc* (*4*m*)))) **in** *mult-less-imp-less-right*)
apply (*auto simp add*: *mult-assoc mult-less-cancel-left simp del*: *fact-Suc*)
apply (*subgoal-tac x * (x * x ^ (4*m)) = (x ^ (4*m)) * (x * x)*)
apply (*erule ssubst*)+
apply (*auto simp del*: *fact-Suc*)
apply (*subgoal-tac 0 < x ^ (4 * m)*)
prefer *2* **apply** (*simp only*: *zero-less-power*)
apply (*simp* (*no-asm-simp*) *add*: *mult-less-cancel-left*)
apply (*rule mult-strict-mono*)
apply (*simp-all* (*no-asm-simp*))
done

lemma *sin-gt-zero1*: [|*0 < x*; *x < 2* |] ==> *0 < sin x*
by (*auto intro*: *sin-gt-zero*)

lemma *cos-double-less-one*: [| *0 < x*; *x < 2* |] ==> *cos* (*2 * x*) *< 1*
apply (*cut-tac x = x* **in** *sin-gt-zero1*)
apply (*auto simp add*: *cos-squared-eq cos-double*)
done

lemma *cos-paired*:

```
    (%n. -1 ^ n /(real (fact (2 * n))) * x ^ (2 * n)) sums cos x
proof -
  have (λn. ∑k = n * 2..<n * 2 + 2.
          (if even k then -1 ^ (k div 2) / real (fact k) else 0) *
          x ^ k)
       sums cos x
    unfolding cos-def
    by (rule cos-converges [THEN sums-summable, THEN sums-group], simp)
  thus ?thesis by (simp add: mult-ac)
qed

lemma fact-lemma: real (n::nat) * 4 = real (4 * n)
by simp

lemma cos-two-less-zero [simp]: cos (2) < 0
apply (cut-tac x = 2 in cos-paired)
apply (drule sums-minus)
apply (rule neg-less-iff-less [THEN iffD1])
apply (frule sums-unique, auto)
apply (rule-tac y =
 ∑n=0..< Suc(Suc(Suc 0)). - (-1 ^ n / (real(fact (2*n))) * 2 ^ (2*n))
      in order-less-trans)
apply (simp (no-asm) add: fact-num-eq-if realpow-num-eq-if del: fact-Suc realpow-Suc)
apply (simp (no-asm) add: mult-assoc del: setsum-op-ivl-Suc)
apply (rule sumr-pos-lt-pair)
apply (erule sums-summable, safe)
apply (simp (no-asm) add: divide-inverse real-0-less-add-iff mult-assoc [symmetric]

            del: fact-Suc)
apply (rule real-mult-inverse-cancel2)
apply (rule real-of-nat-fact-gt-zero)+
apply (simp (no-asm) add: mult-assoc [symmetric] del: fact-Suc)
apply (subst fact-lemma)
apply (subst fact-Suc [of Suc (Suc (Suc (Suc (Suc (Suc (4 * d))))))])
apply (simp only: real-of-nat-mult)
apply (rule mult-strict-mono, force)
  apply (rule-tac [3] real-of-nat-fact-ge-zero)
 prefer 2 apply force
apply (rule real-of-nat-less-iff [THEN iffD2])
apply (rule fact-less-mono, auto)
done

lemmas cos-two-neq-zero [simp] = cos-two-less-zero [THEN less-imp-neq]
lemmas cos-two-le-zero [simp] = cos-two-less-zero [THEN order-less-imp-le]

lemma cos-is-zero: EX! x. 0 ≤ x & x ≤ 2 & cos x = 0
apply (subgoal-tac ∃x. 0 ≤ x & x ≤ 2 & cos x = 0)
apply (rule-tac [2] IVT2)
apply (auto intro: DERIV-isCont DERIV-cos)
```

apply (*cut-tac* $x = xa$ **and** $y = y$ **in** *linorder-less-linear*)
apply (*rule ccontr*)
apply (*subgoal-tac* $(\forall x.\ cos\ differentiable\ x)\ \&\ (\forall x.\ isCont\ cos\ x)$)
apply (*auto intro*: *DERIV-cos DERIV-isCont simp add*: *differentiable-def*)
apply (*drule-tac* $f = cos$ **in** *Rolle*)
apply (*drule-tac* [*5*] $f = cos$ **in** *Rolle*)
apply (*auto dest*!: *DERIV-cos* [*THEN DERIV-unique*] *simp add*: *differentiable-def*)
apply (*drule-tac* $y1 = xa$ **in** *order-le-less-trans* [*THEN sin-gt-zero*])
apply (*assumption*, *rule-tac* $y{=}y$ **in** *order-less-le-trans*, *simp-all*)
apply (*drule-tac* $y1 = y$ **in** *order-le-less-trans* [*THEN sin-gt-zero*], *assumption*, *simp-all*)
done

lemma *pi-half*: $pi/2 = (THE\ x.\ 0 \leq x\ \&\ x \leq 2\ \&\ cos\ x = 0)$
by (*simp add*: *pi-def*)

lemma *cos-pi-half* [*simp*]: $cos\ (pi\ /\ 2) = 0$
by (*simp add*: *pi-half cos-is-zero* [*THEN theI'*])

lemma *pi-half-gt-zero* [*simp*]: $0 < pi\ /\ 2$
apply (*rule order-le-neq-trans*)
apply (*simp add*: *pi-half cos-is-zero* [*THEN theI'*])
apply (*rule notI*, *drule arg-cong* [**where** $f{=}cos$], *simp*)
done

lemmas *pi-half-neq-zero* [*simp*] = *pi-half-gt-zero* [*THEN less-imp-neq*, *symmetric*]
lemmas *pi-half-ge-zero* [*simp*] = *pi-half-gt-zero* [*THEN order-less-imp-le*]

lemma *pi-half-less-two* [*simp*]: $pi\ /\ 2 < 2$
apply (*rule order-le-neq-trans*)
apply (*simp add*: *pi-half cos-is-zero* [*THEN theI'*])
apply (*rule notI*, *drule arg-cong* [**where** $f{=}cos$], *simp*)
done

lemmas *pi-half-neq-two* [*simp*] = *pi-half-less-two* [*THEN less-imp-neq*]
lemmas *pi-half-le-two* [*simp*] = *pi-half-less-two* [*THEN order-less-imp-le*]

lemma *pi-gt-zero* [*simp*]: $0 < pi$
by (*insert pi-half-gt-zero*, *simp*)

lemma *pi-ge-zero* [*simp*]: $0 \leq pi$
by (*rule pi-gt-zero* [*THEN order-less-imp-le*])

lemma *pi-neq-zero* [*simp*]: $pi \neq 0$
by (*rule pi-gt-zero* [*THEN less-imp-neq*, *THEN not-sym*])

lemma *pi-not-less-zero* [*simp*]: $\neg\ pi < 0$
by (*simp add*: *linorder-not-less*)

lemma *minus-pi-half-less-zero* [*simp*]: $-(pi/2) < 0$
by *auto*

lemma *sin-pi-half* [*simp*]: $sin(pi/2) = 1$
apply (*cut-tac* $x = pi/2$ **in** *sin-cos-squared-add2*)
apply (*cut-tac sin-gt-zero* [*OF pi-half-gt-zero pi-half-less-two*])
apply (*simp add*: *power2-eq-square*)
done

lemma *cos-pi* [*simp*]: $cos\ pi = -1$
by (*cut-tac* $x = pi/2$ **and** $y = pi/2$ **in** *cos-add*, *simp*)

lemma *sin-pi* [*simp*]: $sin\ pi = 0$
by (*cut-tac* $x = pi/2$ **and** $y = pi/2$ **in** *sin-add*, *simp*)

lemma *sin-cos-eq*: $sin\ x = cos\ (pi/2 - x)$
by (*simp add*: *diff-minus cos-add*)
declare *sin-cos-eq* [*symmetric*, *simp*]

lemma *minus-sin-cos-eq*: $-sin\ x = cos\ (x + pi/2)$
by (*simp add*: *cos-add*)
declare *minus-sin-cos-eq* [*symmetric*, *simp*]

lemma *cos-sin-eq*: $cos\ x = sin\ (pi/2 - x)$
by (*simp add*: *diff-minus sin-add*)
declare *cos-sin-eq* [*symmetric*, *simp*]

lemma *sin-periodic-pi* [*simp*]: $sin\ (x + pi) = -\ sin\ x$
by (*simp add*: *sin-add*)

lemma *sin-periodic-pi2* [*simp*]: $sin\ (pi + x) = -\ sin\ x$
by (*simp add*: *sin-add*)

lemma *cos-periodic-pi* [*simp*]: $cos\ (x + pi) = -\ cos\ x$
by (*simp add*: *cos-add*)

lemma *sin-periodic* [*simp*]: $sin\ (x + 2*pi) = sin\ x$
by (*simp add*: *sin-add cos-double*)

lemma *cos-periodic* [*simp*]: $cos\ (x + 2*pi) = cos\ x$
by (*simp add*: *cos-add cos-double*)

lemma *cos-npi* [*simp*]: $cos\ (real\ n * pi) = -1\ \hat{}\ n$
apply (*induct n*)
apply (*auto simp add*: *real-of-nat-Suc left-distrib*)
done

lemma *cos-npi2* [*simp*]: $cos\ (pi * real\ n) = -1\ \hat{}\ n$
proof −

have *cos* (*pi* ∗ *real n*) = *cos* (*real n* ∗ *pi*) **by** (*simp only*: *mult-commute*)
also have ... = −*1* ^ *n* **by** (*rule cos-npi*)
finally show *?thesis* **.**
qed

lemma *sin-npi* [*simp*]: *sin* (*real* (*n*::*nat*) ∗ *pi*) = *0*
apply (*induct n*)
apply (*auto simp add*: *real-of-nat-Suc left-distrib*)
done

lemma *sin-npi2* [*simp*]: *sin* (*pi* ∗ *real* (*n*::*nat*)) = *0*
by (*simp add*: *mult-commute* [*of pi*])

lemma *cos-two-pi* [*simp*]: *cos* (*2* ∗ *pi*) = *1*
by (*simp add*: *cos-double*)

lemma *sin-two-pi* [*simp*]: *sin* (*2* ∗ *pi*) = *0*
by *simp*

lemma *sin-gt-zero2*: [| *0* < *x*; *x* < *pi*/*2* |] ==> *0* < *sin x*
apply (*rule sin-gt-zero*, *assumption*)
apply (*rule order-less-trans*, *assumption*)
apply (*rule pi-half-less-two*)
done

lemma *sin-less-zero*:
assumes *lb*: − *pi*/*2* < *x* **and** *x* < *0* **shows** *sin x* < *0*
proof −
have *0* < *sin* (− *x*) **using** *prems* **by** (*simp only*: *sin-gt-zero2*)
thus *?thesis* **by** *simp*
qed

lemma *pi-less-4*: *pi* < *4*
by (*cut-tac pi-half-less-two*, *auto*)

lemma *cos-gt-zero*: [| *0* < *x*; *x* < *pi*/*2* |] ==> *0* < *cos x*
apply (*cut-tac pi-less-4*)
apply (*cut-tac f* = *cos* **and** *a* = *0* **and** *b* = *x* **and** *y* = *0* **in** *IVT2-objl*, *safe*, *simp-all*)
apply (*cut-tac cos-is-zero*, *safe*)
apply (*rename-tac y z*)
apply (*drule-tac x* = *y* **in** *spec*)
apply (*drule-tac x* = *pi*/*2* **in** *spec*, *simp*)
done

lemma *cos-gt-zero-pi*: [| −(*pi*/*2*) < *x*; *x* < *pi*/*2* |] ==> *0* < *cos x*
apply (*rule-tac x* = *x* **and** *y* = *0* **in** *linorder-cases*)
apply (*rule cos-minus* [*THEN subst*])
apply (*rule cos-gt-zero*)

apply (*auto intro*: *cos-gt-zero*)
done

lemma *cos-ge-zero*: [| $-(pi/2) \leq x$; $x \leq pi/2$ |] ==> $0 \leq cos\ x$
apply (*auto simp add*: *order-le-less cos-gt-zero-pi*)
apply (*subgoal-tac* $x = pi/2$, *auto*)
done

lemma *sin-gt-zero-pi*: [| $0 < x$; $x < pi$ |] ==> $0 < sin\ x$
apply (*subst sin-cos-eq*)
apply (*rotate-tac 1*)
apply (*drule real-sum-of-halves* [*THEN ssubst*])
apply (*auto intro*!: *cos-gt-zero-pi simp del*: *sin-cos-eq* [*symmetric*])
done

lemma *sin-ge-zero*: [| $0 \leq x$; $x \leq pi$ |] ==> $0 \leq sin\ x$
by (*auto simp add*: *order-le-less sin-gt-zero-pi*)

lemma *cos-total*: [| $-1 \leq y$; $y \leq 1$ |] ==> *EX*! $x.\ 0 \leq x$ & $x \leq pi$ & ($cos\ x = y$)
apply (*subgoal-tac* $\exists x.\ 0 \leq x$ & $x \leq pi$ & $cos\ x = y$)
apply (*rule-tac* [*2*] *IVT2*)
apply (*auto intro*: *order-less-imp-le DERIV-isCont DERIV-cos*)
apply (*cut-tac* $x = xa$ **and** $y = y$ **in** *linorder-less-linear*)
apply (*rule ccontr*, *auto*)
apply (*drule-tac* $f = cos$ **in** *Rolle*)
apply (*drule-tac* [*5*] $f = cos$ **in** *Rolle*)
apply (*auto intro*: *order-less-imp-le DERIV-isCont DERIV-cos*
dest!: *DERIV-cos* [*THEN DERIV-unique*]
simp add: *differentiable-def*)
apply (*auto dest*: *sin-gt-zero-pi* [*OF order-le-less-trans order-less-le-trans*])
done

lemma *sin-total*:
[| $-1 \leq y$; $y \leq 1$ |] ==> *EX*! $x.\ -(pi/2) \leq x$ & $x \leq pi/2$ & ($sin\ x = y$)
apply (*rule ccontr*)
apply (*subgoal-tac* $\forall x.\ (-\ (pi/2) \leq x$ & $x \leq pi/2$ & $(sin\ x = y)) = (0 \leq (x + pi/2)$ & $(x + pi/2) \leq pi$ & $(cos\ (x + pi/2) = -y)))$
apply (*erule contrapos-np*)
apply (*simp del*: *minus-sin-cos-eq* [*symmetric*])
apply (*cut-tac* $y{=}{-}y$ **in** *cos-total*, *simp*) **apply** *simp*
apply (*erule ex1E*)
apply (*rule-tac* $a = x - (pi/2)$ **in** *ex1I*)
apply (*simp* (*no-asm*) *add*: *add-assoc*)
apply (*rotate-tac 3*)
apply (*drule-tac* $x = xa + pi/2$ **in** *spec*, *safe*, *simp-all*)
done

lemma *reals-Archimedean4*:

```
    [| 0 < y; 0 ≤ x |] ==> ∃n. real n * y ≤ x & x < real (Suc n) * y
apply (auto dest!: reals-Archimedean3)
apply (drule-tac x = x in spec, clarify)
apply (subgoal-tac x < real(LEAST m::nat. x < real m * y) * y)
 prefer 2 apply (erule LeastI)
apply (case-tac LEAST m::nat. x < real m * y, simp)
apply (subgoal-tac ~ x < real nat * y)
 prefer 2 apply (rule not-less-Least, simp, force)
done


lemma cos-zero-lemma:
    [| 0 ≤ x; cos x = 0 |] ==>
     ∃n::nat. ~even n & x = real n * (pi/2)
apply (drule pi-gt-zero [THEN reals-Archimedean4], safe)
apply (subgoal-tac 0 ≤ x - real n * pi &
                   (x - real n * pi) ≤ pi & (cos (x - real n * pi) = 0) )
apply (auto simp add: compare-rls)
  prefer 3 apply (simp add: cos-diff)
 prefer 2 apply (simp add: real-of-nat-Suc left-distrib)
apply (simp add: cos-diff)
apply (subgoal-tac EX! x. 0 ≤ x & x ≤ pi & cos x = 0)
apply (rule-tac [2] cos-total, safe)
apply (drule-tac x = x - real n * pi in spec)
apply (drule-tac x = pi/2 in spec)
apply (simp add: cos-diff)
apply (rule-tac x = Suc (2 * n) in exI)
apply (simp add: real-of-nat-Suc left-distrib, auto)
done

lemma sin-zero-lemma:
    [| 0 ≤ x; sin x = 0 |] ==>
     ∃n::nat. even n & x = real n * (pi/2)
apply (subgoal-tac ∃n::nat. ~ even n & x + pi/2 = real n * (pi/2) )
 apply (clarify, rule-tac x = n - 1 in exI)
 apply (force simp add: odd-Suc-mult-two-ex real-of-nat-Suc left-distrib)
apply (rule cos-zero-lemma)
apply (simp-all add: add-increasing)
done


lemma cos-zero-iff:
    (cos x = 0) =
     ((∃n::nat. ~even n & (x = real n * (pi/2))) |
      (∃n::nat. ~even n & (x = -(real n * (pi/2)))))
apply (rule iffI)
apply (cut-tac linorder-linear [of 0 x], safe)
apply (drule cos-zero-lemma, assumption+)
apply (cut-tac x=-x in cos-zero-lemma, simp, simp)
```

apply (*force simp add*: *minus-equation-iff* [*of x*])
apply (*auto simp only*: *odd-Suc-mult-two-ex real-of-nat-Suc left-distrib*)
apply (*auto simp add*: *cos-add*)
done

lemma *sin-zero-iff*:
(*sin x = 0*) =
((∃ *n*::*nat*. *even n* & (*x = real n * (pi/2)*)) |
(∃ *n*::*nat*. *even n* & (*x = −(real n * (pi/2))*)))
apply (*rule iffI*)
apply (*cut-tac linorder-linear* [*of 0 x*], *safe*)
apply (*drule sin-zero-lemma*, *assumption*+)
apply (*cut-tac x=−x* **in** *sin-zero-lemma*, *simp*, *simp*, *safe*)
apply (*force simp add*: *minus-equation-iff* [*of x*])
apply (*auto simp add*: *even-mult-two-ex*)
done

22.9 Tangent

definition
tan :: *real* => *real* **where**
tan x = (*sin x*)/(*cos x*)

lemma *tan-zero* [*simp*]: *tan 0 = 0*
by (*simp add*: *tan-def*)

lemma *tan-pi* [*simp*]: *tan pi = 0*
by (*simp add*: *tan-def*)

lemma *tan-npi* [*simp*]: *tan* (*real* (*n*::*nat*) * *pi*) = *0*
by (*simp add*: *tan-def*)

lemma *tan-minus* [*simp*]: *tan* (*−x*) = *− tan x*
by (*simp add*: *tan-def minus-mult-left*)

lemma *tan-periodic* [*simp*]: *tan* (*x + 2*pi*) = *tan x*
by (*simp add*: *tan-def*)

lemma *lemma-tan-add1*:
[| *cos x ≠ 0*; *cos y ≠ 0* |]
==> *1 − tan*(*x*)*tan*(*y*) = *cos* (*x + y*)/(*cos x * cos y*)
apply (*simp add*: *tan-def divide-inverse*)
apply (*auto simp del*: *inverse-mult-distrib*
simp add: *inverse-mult-distrib* [*symmetric*] *mult-ac*)
apply (*rule-tac c1 = cos x * cos y* **in** *real-mult-right-cancel* [*THEN subst*])
apply (*auto simp del*: *inverse-mult-distrib*
simp add: *mult-assoc left-diff-distrib cos-add*)
done

lemma *add-tan-eq*:
[| *cos x* ≠ *0*; *cos y* ≠ *0* |]
==> *tan x* + *tan y* = *sin*(*x* + *y*)/(*cos x* ∗ *cos y*)
apply (*simp add*: *tan-def*)
apply (*rule-tac c1* = *cos x* ∗ *cos y* **in** *real-mult-right-cancel* [*THEN subst*])
apply (*auto simp add*: *mult-assoc left-distrib*)
apply (*simp add*: *sin-add*)
done

lemma *tan-add*:
[| *cos x* ≠ *0*; *cos y* ≠ *0*; *cos* (*x* + *y*) ≠ *0* |]
==> *tan*(*x* + *y*) = (*tan*(*x*) + *tan*(*y*))/(*1* − *tan*(*x*) ∗ *tan*(*y*))
apply (*simp* (*no-asm-simp*) *add*: *add-tan-eq lemma-tan-add1*)
apply (*simp add*: *tan-def*)
done

lemma *tan-double*:
[| *cos x* ≠ *0*; *cos* (*2* ∗ *x*) ≠ *0* |]
==> *tan* (*2* ∗ *x*) = (*2* ∗ *tan x*)/(*1* − (*tan*(*x*) ^ *2*))
apply (*insert tan-add* [*of x x*])
apply (*simp add*: *mult-2* [*symmetric*])
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *tan-gt-zero*: [| *0* < *x*; *x* < *pi*/*2* |] ==> *0* < *tan x*
by (*simp add*: *tan-def zero-less-divide-iff sin-gt-zero2 cos-gt-zero-pi*)

lemma *tan-less-zero*:
assumes *lb*: − *pi*/*2* < *x* **and** *x* < *0* **shows** *tan x* < *0*
proof −
have *0* < *tan* (− *x*) **using** *prems* **by** (*simp only*: *tan-gt-zero*)
thus *?thesis* **by** *simp*
qed

lemma *lemma-DERIV-tan*:
cos x ≠ *0* ==> *DERIV* (%*x*. *sin*(*x*)/*cos*(*x*)) *x* :> *inverse*((*cos x*)^2)
apply (*rule lemma-DERIV-subst*)
apply (*best intro*!: *DERIV-intros intro*: *DERIV-chain2*)
apply (*auto simp add*: *divide-inverse numeral-2-eq-2*)
done

lemma *DERIV-tan* [*simp*]: *cos x* ≠ *0* ==> *DERIV tan x* :> *inverse*((*cos x*)^2)
by (*auto dest*: *lemma-DERIV-tan simp add*: *tan-def* [*symmetric*])

lemma *isCont-tan* [*simp*]: *cos x* ≠ *0* ==> *isCont tan x*
by (*rule DERIV-tan* [*THEN DERIV-isCont*])

lemma *LIM-cos-div-sin* [*simp*]: (%*x*. *cos*(*x*)/*sin*(*x*)) −− *pi*/*2* −−> *0*

apply (*subgoal-tac* ($\lambda x.$ *cos x* $*$ *inverse* (*sin x*)) $--$ *pi* $*$ *inverse 2* $--> 0*1$)
apply (*simp add*: *divide-inverse* [*symmetric*])
apply (*rule LIM-mult*)
apply (*rule-tac* [*2*] *inverse-1* [*THEN subst*])
apply (*rule-tac* [*2*] *LIM-inverse*)
apply (*simp-all add*: *divide-inverse* [*symmetric*])
apply (*simp-all only*: *isCont-def* [*symmetric*] *cos-pi-half* [*symmetric*] *sin-pi-half* [*symmetric*])
apply (*blast intro*!: *DERIV-isCont DERIV-sin DERIV-cos*)+
done

lemma *lemma-tan-total*: *0* $<$ *y* $==>$ $\exists$ *x*. *0* $<$ *x* & *x* $<$ *pi/2* & *y* $<$ *tan x*
apply (*cut-tac LIM-cos-div-sin*)
apply (*simp only*: *LIM-def*)
apply (*drule-tac x* $=$ *inverse y* **in** *spec*, *safe*, *force*)
apply (*drule-tac ?d1.0* $=$ *s* **in** *pi-half-gt-zero* [*THEN* [*2*] *real-lbound-gt-zero*], *safe*)
apply (*rule-tac x* $=$ (*pi/2*) $-$ *e* **in** *exI*)
apply (*simp* (*no-asm-simp*))
apply (*drule-tac x* $=$ (*pi/2*) $-$ *e* **in** *spec*)
apply (*auto simp add*: *tan-def*)
apply (*rule inverse-less-iff-less* [*THEN iffD1*])
apply (*auto simp add*: *divide-inverse*)
apply (*rule real-mult-order*)
apply (*subgoal-tac* [*3*] *0* $<$ *sin e* & *0* $<$ *cos e*)
apply (*auto intro*: *cos-gt-zero sin-gt-zero2 simp add*: *mult-commute*)
done

lemma *tan-total-pos*: *0* $\leq$ *y* $==>$ $\exists$ *x*. *0* $\leq$ *x* & *x* $<$ *pi/2* & *tan x* $=$ *y*
apply (*frule order-le-imp-less-or-eq*, *safe*)
prefer *2* **apply** *force*
apply (*drule lemma-tan-total*, *safe*)
apply (*cut-tac f* $=$ *tan* **and** *a* $=$ *0* **and** *b* $=$ *x* **and** *y* $=$ *y* **in** *IVT-objl*)
apply (*auto intro*!: *DERIV-tan* [*THEN DERIV-isCont*])
apply (*drule-tac y* $=$ *xa* **in** *order-le-imp-less-or-eq*)
apply (*auto dest*: *cos-gt-zero*)
done

lemma *lemma-tan-total1*: $\exists$ *x*. $-$(*pi/2*) $<$ *x* & *x* $<$ (*pi/2*) & *tan x* $=$ *y*
apply (*cut-tac linorder-linear* [*of 0 y*], *safe*)
apply (*drule tan-total-pos*)
apply (*cut-tac* [*2*] *y*$=-$*y* **in** *tan-total-pos*, *safe*)
apply (*rule-tac* [*3*] *x* $= -x$ **in** *exI*)
apply (*auto intro*!: *exI*)
done

lemma *tan-total*: *EX*! *x*. $-$(*pi/2*) $<$ *x* & *x* $<$ (*pi/2*) & *tan x* $=$ *y*
apply (*cut-tac y* $=$ *y* **in** *lemma-tan-total1*, *auto*)
apply (*cut-tac x* $=$ *xa* **and** *y* $=$ *y* **in** *linorder-less-linear*, *auto*)
apply (*subgoal-tac* [*2*] $\exists$ *z*. *y* $<$ *z* & *z* $<$ *xa* & *DERIV tan z* $:>$ *0*)

apply (*subgoal-tac* $\exists z.\ xa < z\ \&\ z < y\ \&$ *DERIV tan z :> 0*)
apply (*rule-tac* [*4*] *Rolle*)
apply (*rule-tac* [*2*] *Rolle*)
apply (*auto intro*!: *DERIV-tan DERIV-isCont exI*
simp add: *differentiable-def*)

Now, simulate TRYALL

apply (*rule-tac* [!] *DERIV-tan asm-rl*)
apply (*auto dest*!: *DERIV-unique* [*OF - DERIV-tan*]
simp add: *cos-gt-zero-pi* [*THEN less-imp-neq*, *THEN not-sym*])
done

22.10 Inverse Trigonometric Functions

definition
arcsin :: *real* => *real* **where**
arcsin y = (*THE x.* $-(pi/2) \le x\ \&\ x \le pi/2\ \&$ *sin x* = *y*)

definition
arccos :: *real* => *real* **where**
arccos y = (*THE x.* $0 \le x\ \&\ x \le pi\ \&$ *cos x* = *y*)

definition
arctan :: *real* => *real* **where**
arctan y = (*THE x.* $-(pi/2) < x\ \&\ x < pi/2\ \&$ *tan x* = *y*)

lemma *arcsin*:
$[|\ -1 \le y;\ y \le 1\ |]$
$==>\ -(pi/2) \le$ *arcsin y* $\&$
arcsin y $\le pi/2\ \&$ *sin*(*arcsin y*) = *y*
unfolding *arcsin-def* **by** (*rule theI'* [*OF sin-total*])

lemma *arcsin-pi*:
$[|\ -1 \le y;\ y \le 1\ |]$
$==>\ -(pi/2) \le$ *arcsin y* $\&$ *arcsin y* $\le pi\ \&$ *sin*(*arcsin y*) = *y*
apply (*drule* (*1*) *arcsin*)
apply (*force intro*: *order-trans*)
done

lemma *sin-arcsin* [*simp*]: $[|\ -1 \le y;\ y \le 1\ |] ==>$ *sin*(*arcsin y*) = *y*
by (*blast dest*: *arcsin*)

lemma *arcsin-bounded*:
$[|\ -1 \le y;\ y \le 1\ |] ==> -(pi/2) \le$ *arcsin y* $\&$ *arcsin y* $\le pi/2$
by (*blast dest*: *arcsin*)

lemma *arcsin-lbound*: $[|\ -1 \le y;\ y \le 1\ |] ==> -(pi/2) \le$ *arcsin y*
by (*blast dest*: *arcsin*)

lemma *arcsin-ubound*: $[|\ -1 \le y;\ y \le 1\ |] ==>$ *arcsin y* $\le pi/2$

by (*blast dest*: *arcsin*)

lemma *arcsin-lt-bounded*:
[| $-1 < y$; $y < 1$ |] ==> $-(pi/2) < arcsin\ y$ & $arcsin\ y < pi/2$
apply (*frule order-less-imp-le*)
apply (*frule-tac* $y = y$ **in** *order-less-imp-le*)
apply (*frule arcsin-bounded*)
apply (*safe*, *simp*)
apply (*drule-tac* $y = arcsin\ y$ **in** *order-le-imp-less-or-eq*)
apply (*drule-tac* [2] $y = pi/2$ **in** *order-le-imp-less-or-eq*, *safe*)
apply (*drule-tac* [!] $f = sin$ **in** *arg-cong*, *auto*)
done

lemma *arcsin-sin*: [|$-(pi/2) \le x$; $x \le pi/2$ |] ==> $arcsin(sin\ x) = x$
apply (*unfold arcsin-def*)
apply (*rule the1-equality*)
apply (*rule sin-total*, *auto*)
done

lemma *arccos*:
[| $-1 \le y$; $y \le 1$ |]
==> $0 \le arccos\ y$ & $arccos\ y \le pi$ & $cos(arccos\ y) = y$
unfolding *arccos-def* **by** (*rule theI' [OF cos-total]*)

lemma *cos-arccos* [*simp*]: [| $-1 \le y$; $y \le 1$ |] ==> $cos(arccos\ y) = y$
by (*blast dest*: *arccos*)

lemma *arccos-bounded*: [| $-1 \le y$; $y \le 1$ |] ==> $0 \le arccos\ y$ & $arccos\ y \le pi$
by (*blast dest*: *arccos*)

lemma *arccos-lbound*: [| $-1 \le y$; $y \le 1$ |] ==> $0 \le arccos\ y$
by (*blast dest*: *arccos*)

lemma *arccos-ubound*: [| $-1 \le y$; $y \le 1$ |] ==> $arccos\ y \le pi$
by (*blast dest*: *arccos*)

lemma *arccos-lt-bounded*:
[| $-1 < y$; $y < 1$ |]
==> $0 < arccos\ y$ & $arccos\ y < pi$
apply (*frule order-less-imp-le*)
apply (*frule-tac* $y = y$ **in** *order-less-imp-le*)
apply (*frule arccos-bounded*, *auto*)
apply (*drule-tac* $y = arccos\ y$ **in** *order-le-imp-less-or-eq*)
apply (*drule-tac* [2] $y = pi$ **in** *order-le-imp-less-or-eq*, *auto*)
apply (*drule-tac* [!] $f = cos$ **in** *arg-cong*, *auto*)
done

lemma *arccos-cos*: [|$0 \le x$; $x \le pi$ |] ==> $arccos(cos\ x) = x$
apply (*simp add*: *arccos-def*)

apply (*auto intro*!: *the1-equality cos-total*)
done

lemma *arccos-cos2*: $[|x \leq 0; -pi \leq x\ |] ==> arccos(cos\ x) = -x$
apply (*simp add*: *arccos-def*)
apply (*auto intro*!: *the1-equality cos-total*)
done

lemma *cos-arcsin*: $[\![-1 \leq x; x \leq 1]\!] \Longrightarrow cos\ (arcsin\ x) = sqrt\ (1 - x^2)$
apply (*subgoal-tac* $x^2 \leq 1$)
apply (*rule power2-eq-imp-eq*)
apply (*simp add*: *cos-squared-eq*)
apply (*rule cos-ge-zero*)
apply (*erule* (*1*) *arcsin-lbound*)
apply (*erule* (*1*) *arcsin-ubound*)
apply *simp*
apply (*subgoal-tac* $|x|^2 \leq 1^2$, *simp*)
apply (*rule power-mono*, *simp*, *simp*)
done

lemma *sin-arccos*: $[\![-1 \leq x; x \leq 1]\!] \Longrightarrow sin\ (arccos\ x) = sqrt\ (1 - x^2)$
apply (*subgoal-tac* $x^2 \leq 1$)
apply (*rule power2-eq-imp-eq*)
apply (*simp add*: *sin-squared-eq*)
apply (*rule sin-ge-zero*)
apply (*erule* (*1*) *arccos-lbound*)
apply (*erule* (*1*) *arccos-ubound*)
apply *simp*
apply (*subgoal-tac* $|x|^2 \leq 1^2$, *simp*)
apply (*rule power-mono*, *simp*, *simp*)
done

lemma *arctan* [*simp*]:
$-(pi/2) < arctan\ y\ \&\ arctan\ y < pi/2\ \&\ tan\ (arctan\ y) = y$
unfolding *arctan-def* **by** (*rule theI'* [*OF tan-total*])

lemma *tan-arctan*: $tan(arctan\ y) = y$
by *auto*

lemma *arctan-bounded*: $-(pi/2) < arctan\ y\ \&\ arctan\ y < pi/2$
by (*auto simp only*: *arctan*)

lemma *arctan-lbound*: $-(pi/2) < arctan\ y$
by *auto*

lemma *arctan-ubound*: $arctan\ y < pi/2$
by (*auto simp only*: *arctan*)

lemma *arctan-tan*:

```
    [|−(pi/2) < x; x < pi/2 |] ==> arctan(tan x) = x
apply (unfold arctan-def)
apply (rule the1-equality)
apply (rule tan-total, auto)
done

lemma arctan-zero-zero [simp]: arctan 0 = 0
by (insert arctan-tan [of 0], simp)

lemma cos-arctan-not-zero [simp]: cos(arctan x) ≠ 0
apply (auto simp add: cos-zero-iff)
apply (case-tac n)
apply (case-tac [3] n)
apply (cut-tac [2] y = x in arctan-ubound)
apply (cut-tac [4] y = x in arctan-lbound)
apply (auto simp add: real-of-nat-Suc left-distrib mult-less-0-iff)
done

lemma tan-sec: cos x ≠ 0 ==> 1 + tan(x) ^ 2 = inverse(cos x) ^ 2
apply (rule power-inverse [THEN subst])
apply (rule-tac c1 = (cos x)^2 in real-mult-right-cancel [THEN iffD1])
apply (auto dest: field-power-not-zero
       simp add: power-mult-distrib left-distrib power-divide tan-def
                 mult-assoc power-inverse [symmetric]
       simp del: realpow-Suc)
done

lemma isCont-inverse-function2:
  fixes f g :: real ⇒ real shows
  ⟦a < x; x < b;
    ∀z. a ≤ z ∧ z ≤ b ⟶ g (f z) = z;
    ∀z. a ≤ z ∧ z ≤ b ⟶ isCont f z⟧
   ⟹ isCont g (f x)
apply (rule isCont-inverse-function
      [where f=f and d=min (x − a) (b − x)])
apply (simp-all add: abs-le-iff)
done

lemma isCont-arcsin: ⟦−1 < x; x < 1⟧ ⟹ isCont arcsin x
apply (subgoal-tac isCont arcsin (sin (arcsin x)), simp)
apply (rule isCont-inverse-function2 [where f=sin])
apply (erule (1) arcsin-lt-bounded [THEN conjunct1])
apply (erule (1) arcsin-lt-bounded [THEN conjunct2])
apply (fast intro: arcsin-sin, simp)
done

lemma isCont-arccos: ⟦−1 < x; x < 1⟧ ⟹ isCont arccos x
apply (subgoal-tac isCont arccos (cos (arccos x)), simp)
apply (rule isCont-inverse-function2 [where f=cos])
```

apply (*erule* (*1*) *arccos-lt-bounded* [*THEN conjunct1*])
apply (*erule* (*1*) *arccos-lt-bounded* [*THEN conjunct2*])
apply (*fast intro*: *arccos-cos*, *simp*)
done

lemma *isCont-arctan*: *isCont arctan x*
apply (*rule arctan-lbound* [*of x*, *THEN dense*, *THEN exE*], *clarify*)
apply (*rule arctan-ubound* [*of x*, *THEN dense*, *THEN exE*], *clarify*)
apply (*subgoal-tac isCont arctan* (*tan* (*arctan x*)), *simp*)
apply (*erule* (*1*) *isCont-inverse-function2* [**where** *f*=*tan*])
apply (*clarify*, *rule arctan-tan*)
apply (*erule* (*1*) *order-less-le-trans*)
apply (*erule* (*1*) *order-le-less-trans*)
apply (*clarify*, *rule isCont-tan*)
apply (*rule less-imp-neq* [*symmetric*])
apply (*rule cos-gt-zero-pi*)
apply (*erule* (*1*) *order-less-le-trans*)
apply (*erule* (*1*) *order-le-less-trans*)
done

lemma *DERIV-arcsin*:
⟦*−1 < x*; *x < 1*⟧ ⟹ *DERIV arcsin x* :> *inverse* (*sqrt* (*1 − x*^{2}))
apply (*rule DERIV-inverse-function* [**where** *f*=*sin* **and** *a*=*−1* **and** *b*=*1*])
apply (*rule lemma-DERIV-subst* [*OF DERIV-sin*])
apply (*simp add*: *cos-arcsin*)
apply (*subgoal-tac* |*x*|^2 < *1*^{2}, *simp*)
apply (*rule power-strict-mono*, *simp*, *simp*, *simp*)
apply *assumption*
apply *assumption*
apply *simp*
apply (*erule* (*1*) *isCont-arcsin*)
done

lemma *DERIV-arccos*:
⟦*−1 < x*; *x < 1*⟧ ⟹ *DERIV arccos x* :> *inverse* (*−* *sqrt* (*1 − x*^{2}))
apply (*rule DERIV-inverse-function* [**where** *f*=*cos* **and** *a*=*−1* **and** *b*=*1*])
apply (*rule lemma-DERIV-subst* [*OF DERIV-cos*])
apply (*simp add*: *sin-arccos*)
apply (*subgoal-tac* |*x*|^2 < *1*^{2}, *simp*)
apply (*rule power-strict-mono*, *simp*, *simp*, *simp*)
apply *assumption*
apply *assumption*
apply *simp*
apply (*erule* (*1*) *isCont-arccos*)
done

lemma *DERIV-arctan*: *DERIV arctan x* :> *inverse* (*1* + *x*^{2})
apply (*rule DERIV-inverse-function* [**where** *f*=*tan* **and** *a*=*x − 1* **and** *b*=*x* + *1*])

apply (*rule lemma-DERIV-subst* [*OF DERIV-tan*])
apply (*rule cos-arctan-not-zero*)
apply (*simp add*: *power-inverse tan-sec* [*symmetric*])
apply (*subgoal-tac* $0 < 1 + x^2$, *simp*)
apply (*simp add*: *add-pos-nonneg*)
apply (*simp*, *simp*, *simp*, *rule isCont-arctan*)
done

22.11 More Theorems about Sin and Cos

lemma *cos-45*: *cos* (*pi* / *4*) = *sqrt 2* / *2*
proof −
  let *?c* = *cos* (*pi* / *4*) **and** *?s* = *sin* (*pi* / *4*)
  have *nonneg*: *0* ≤ *?c*
    by (*rule cos-ge-zero*, *rule order-trans* [**where** *y=0*], *simp-all*)
  have *0* = *cos* (*pi* / *4* + *pi* / *4*)
    by *simp*
  also have *cos* (*pi* / *4* + *pi* / *4*) = $?c^2 - ?s^2$
    by (*simp only*: *cos-add power2-eq-square*)
  also have ... = *2* ∗ $?c^2$ − *1*
    by (*simp add*: *sin-squared-eq*)
  finally have $?c^2 = (sqrt\ 2\ /\ 2)^2$
    by (*simp add*: *power-divide*)
  thus *?thesis*
    using *nonneg* **by** (*rule power2-eq-imp-eq*) *simp*
qed

lemma *cos-30*: *cos* (*pi* / *6*) = *sqrt 3* / *2*
proof −
  let *?c* = *cos* (*pi* / *6*) **and** *?s* = *sin* (*pi* / *6*)
  have *pos-c*: *0* < *?c*
    by (*rule cos-gt-zero*, *simp*, *simp*)
  have *0* = *cos* (*pi* / *6* + *pi* / *6* + *pi* / *6*)
    by *simp*
  also have ... = (*?c* ∗ *?c* − *?s* ∗ *?s*) ∗ *?c* − (*?s* ∗ *?c* + *?c* ∗ *?s*) ∗ *?s*
    by (*simp only*: *cos-add sin-add*)
  also have ... = *?c* ∗ ($?c^2$ − *3* ∗ $?s^2$)
    by (*simp add*: *ring-simps power2-eq-square*)
  finally have $?c^2 = (sqrt\ 3\ /\ 2)^2$
    using *pos-c* **by** (*simp add*: *sin-squared-eq power-divide*)
  thus *?thesis*
    using *pos-c* [*THEN order-less-imp-le*]
    by (*rule power2-eq-imp-eq*) *simp*
qed

lemma *sin-45*: *sin* (*pi* / *4*) = *sqrt 2* / *2*
proof −
  have *sin* (*pi* / *4*) = *cos* (*pi* / *2* − *pi* / *4*) **by** (*rule sin-cos-eq*)
  also have *pi* / *2* − *pi* / *4* = *pi* / *4* **by** *simp*

also have *cos (pi / 4) = sqrt 2 / 2* **by** (*rule cos-45*)
finally show *?thesis* **.**
qed

lemma *sin-60*: *sin (pi / 3) = sqrt 3 / 2*
proof −
have *sin (pi / 3) = cos (pi / 2 − pi / 3)* **by** (*rule sin-cos-eq*)
also have *pi / 2 − pi / 3 = pi / 6* **by** *simp*
also have *cos (pi / 6) = sqrt 3 / 2* **by** (*rule cos-30*)
finally show *?thesis* **.**
qed

lemma *cos-60*: *cos (pi / 3) = 1 / 2*
apply (*rule power2-eq-imp-eq*)
apply (*simp add*: *cos-squared-eq sin-60 power-divide*)
apply (*rule cos-ge-zero*, *rule order-trans* [**where** *y=0*], *simp-all*)
done

lemma *sin-30*: *sin (pi / 6) = 1 / 2*
proof −
have *sin (pi / 6) = cos (pi / 2 − pi / 6)* **by** (*rule sin-cos-eq*)
also have *pi / 2 − pi / 6 = pi / 3* **by** *simp*
also have *cos (pi / 3) = 1 / 2* **by** (*rule cos-60*)
finally show *?thesis* **.**
qed

lemma *tan-30*: *tan (pi / 6) = 1 / sqrt 3*
unfolding *tan-def* **by** (*simp add*: *sin-30 cos-30*)

lemma *tan-45*: *tan (pi / 4) = 1*
unfolding *tan-def* **by** (*simp add*: *sin-45 cos-45*)

lemma *tan-60*: *tan (pi / 3) = sqrt 3*
unfolding *tan-def* **by** (*simp add*: *sin-60 cos-60*)

NEEDED??

lemma [*simp*]:
*sin (x + 1 / 2 * real (Suc m) * pi) =*
*cos (x + 1 / 2 * real (m) * pi)*
by (*simp only*: *cos-add sin-add real-of-nat-Suc left-distrib right-distrib*, *auto*)

NEEDED??

lemma [*simp*]:
*sin (x + real (Suc m) * pi / 2) =*
*cos (x + real (m) * pi / 2)*
by (*simp only*: *cos-add sin-add real-of-nat-Suc add-divide-distrib left-distrib*, *auto*)

lemma *DERIV-sin-add* [*simp*]: *DERIV (%x. sin (x + k)) xa :> cos (xa + k)*
apply (*rule lemma-DERIV-subst*)

apply (*rule-tac f = sin* **and** *g = %x. x + k* **in** *DERIV-chain2*)
apply (*best intro*!: *DERIV-intros intro*: *DERIV-chain2*)+
apply (*simp* (*no-asm*))
done

lemma *sin-cos-npi* [*simp*]: *sin* (*real* (*Suc* (*2 * n*)) * *pi* / *2*) = (*−1*) ^ *n*
proof −
have *sin* ((*real n* + *1*/*2*) * *pi*) = *cos* (*real n* * *pi*)
by (*auto simp add*: *right-distrib sin-add left-distrib mult-ac*)
thus *?thesis*
by (*simp add*: *real-of-nat-Suc left-distrib add-divide-distrib*
mult-commute [*of pi*])
qed

lemma *cos-2npi* [*simp*]: *cos* (*2 * real* (*n*::*nat*) * *pi*) = *1*
by (*simp add*: *cos-double mult-assoc power-add* [*symmetric*] *numeral-2-eq-2*)

lemma *cos-3over2-pi* [*simp*]: *cos* (*3* / *2* * *pi*) = *0*
apply (*subgoal-tac cos* (*pi* + *pi*/*2*) = *0*, *simp*)
apply (*subst cos-add*, *simp*)
done

lemma *sin-2npi* [*simp*]: *sin* (*2 * real* (*n*::*nat*) * *pi*) = *0*
by (*auto simp add*: *mult-assoc*)

lemma *sin-3over2-pi* [*simp*]: *sin* (*3* / *2* * *pi*) = − *1*
apply (*subgoal-tac sin* (*pi* + *pi*/*2*) = − *1*, *simp*)
apply (*subst sin-add*, *simp*)
done

lemma [*simp*]:
cos(*x* + *1* / *2* * *real*(*Suc m*) * *pi*) = −*sin* (*x* + *1* / *2* * *real m* * *pi*)
apply (*simp only*: *cos-add sin-add real-of-nat-Suc right-distrib left-distrib minus-mult-right*, *auto*)
done

lemma [*simp*]: *cos* (*x* + *real*(*Suc m*) * *pi* / *2*) = −*sin* (*x* + *real m* * *pi* / *2*)
by (*simp only*: *cos-add sin-add real-of-nat-Suc left-distrib add-divide-distrib*, *auto*)

lemma *cos-pi-eq-zero* [*simp*]: *cos* (*pi* * *real* (*Suc* (*2 * m*)) / *2*) = *0*
by (*simp only*: *cos-add sin-add real-of-nat-Suc left-distrib right-distrib add-divide-distrib*, *auto*)

lemma *DERIV-cos-add* [*simp*]: *DERIV* (*%x. cos* (*x* + *k*)) *xa* :> − *sin* (*xa* + *k*)
apply (*rule lemma-DERIV-subst*)
apply (*rule-tac f = cos* **and** *g = %x. x + k* **in** *DERIV-chain2*)
apply (*best intro*!: *DERIV-intros intro*: *DERIV-chain2*)+

apply (*simp* (*no-asm*))
done

lemma *sin-zero-abs-cos-one*: *sin x = 0 ==> |cos x| = 1*
by (*auto simp add*: *sin-zero-iff even-mult-two-ex*)

lemma *exp-eq-one-iff* [*simp*]: (*exp* (*x*::*real*) = *1*) = (*x = 0*)
apply *auto*
apply (*drule-tac f = ln* **in** *arg-cong*, *auto*)
done

lemma *cos-one-sin-zero*: *cos x = 1 ==> sin x = 0*
by (*cut-tac x = x* **in** *sin-cos-squared-add3*, *auto*)

22.12 Existence of Polar Coordinates

lemma *cos-x-y-le-one*: $|x / sqrt\ (x^2 + y^2)| \leq 1$
apply (*rule power2-le-imp-le* [*OF - zero-le-one*])
apply (*simp add*: *abs-divide power-divide divide-le-eq not-sum-power2-lt-zero*)
done

lemma *cos-arccos-abs*: $|y| \leq 1 \Longrightarrow cos\ (arccos\ y) = y$
by (*simp add*: *abs-le-iff*)

lemma *sin-arccos-abs*: $|y| \leq 1 \Longrightarrow sin\ (arccos\ y) = sqrt\ (1 - y^2)$
by (*simp add*: *sin-arccos abs-le-iff*)

lemmas *cos-arccos-lemma1 = cos-arccos-abs* [*OF cos-x-y-le-one*]

lemmas *sin-arccos-lemma1 = sin-arccos-abs* [*OF cos-x-y-le-one*]

lemma *polar-ex1*:
    $0 < y \Longrightarrow \exists r\ a.\ x = r * cos\ a\ \&\ y = r * sin\ a$
apply (*rule-tac x = sqrt* $(x^2 + y^2)$ **in** *exI*)
apply (*rule-tac x = arccos* $(x / sqrt\ (x^2 + y^2))$ **in** *exI*)
apply (*simp add*: *cos-arccos-lemma1*)
apply (*simp add*: *sin-arccos-lemma1*)
apply (*simp add*: *power-divide*)
apply (*simp add*: *real-sqrt-mult* [*symmetric*])
apply (*simp add*: *right-diff-distrib*)
done

lemma *polar-ex2*:
    $y < 0 \Longrightarrow \exists r\ a.\ x = r * cos\ a\ \&\ y = r * sin\ a$
apply (*insert polar-ex1* [**where** *x*=*x* **and** *y*=$-y$], *simp*, *clarify*)
apply (*rule-tac x = r* **in** *exI*)
apply (*rule-tac x =* $-a$ **in** *exI*, *simp*)
done

lemma *polar-Ex*: $\exists r\ a.\ x = r * cos\ a\ \&\ y = r * sin\ a$
apply (*rule-tac x=0* **and** *y=y* **in** *linorder-cases*)
apply (*erule polar-ex1*)
apply (*rule-tac x=x* **in** *exI*, *rule-tac x=0* **in** *exI*, *simp*)
apply (*erule polar-ex2*)
done

22.13 Theorems about Limits

lemma *isCont-inv-fun*:
fixes $f\ g :: real \Rightarrow real$
shows $[|\ 0 < d; \forall z.\ |z - x| \leq d \longrightarrow g(f(z)) = z;$
$\forall z.\ |z - x| \leq d \longrightarrow isCont\ f\ z\ |]$
$\Longrightarrow isCont\ g\ (f\ x)$
by (*rule isCont-inverse-function*)

lemma *isCont-inv-fun-inv*:
fixes $f\ g :: real \Rightarrow real$
shows $[|\ 0 < d;$
$\forall z.\ |z - x| \leq d \longrightarrow g(f(z)) = z;$
$\forall z.\ |z - x| \leq d \longrightarrow isCont\ f\ z\ |]$
$\Longrightarrow \exists e.\ 0 < e\ \&$
$(\forall y.\ 0 < |y - f(x)|\ \&\ |y - f(x)| < e \longrightarrow f(g(y)) = y)$
apply (*drule isCont-inj-range*)
prefer *2* **apply** (*assumption*, *assumption*, *auto*)
apply (*rule-tac x = e* **in** *exI*, *auto*)
apply (*rotate-tac 2*)
apply (*drule-tac x = y* **in** *spec*, *auto*)
done

Bartle/Sherbert: Introduction to Real Analysis, Theorem 4.2.9, p. 110

lemma *LIM-fun-gt-zero*:
$[|\ f \dashrightarrow c \longrightarrow (l::real);\ 0 < l\ |]$
$\Longrightarrow \exists r.\ 0 < r\ \&\ (\forall x::real.\ x \neq c\ \&\ |c - x| < r \longrightarrow 0 < f\ x)$
apply (*auto simp add*: *LIM-def*)
apply (*drule-tac x = l/2* **in** *spec*, *safe*, *force*)
apply (*rule-tac x = s* **in** *exI*)
apply (*auto simp only*: *abs-less-iff*)
done

lemma *LIM-fun-less-zero*:
$[|\ f \dashrightarrow c \longrightarrow (l::real);\ l < 0\ |]$
$\Longrightarrow \exists r.\ 0 < r\ \&\ (\forall x::real.\ x \neq c\ \&\ |c - x| < r \longrightarrow f\ x < 0)$
apply (*auto simp add*: *LIM-def*)
apply (*drule-tac x = −l/2* **in** *spec*, *safe*, *force*)
apply (*rule-tac x = s* **in** *exI*)
apply (*auto simp only*: *abs-less-iff*)
done

lemma *LIM-fun-not-zero*:
[| f −− c −−> (l::*real*); $l \neq 0$ |]
==> $\exists r.\ 0 < r$ & ($\forall x$::*real*. $x \neq c$ & $|c - x| < r$ −−> $f\ x \neq 0$)
apply (*cut-tac* $x = l$ **and** $y = 0$ **in** *linorder-less-linear*, *auto*)
apply (*drule LIM-fun-less-zero*)
apply (*drule-tac* [*3*] *LIM-fun-gt-zero*)
apply *force+*
done

end

23 Complex: Complex Numbers: Rectangular and Polar Representations

theory *Complex*
imports *../Real/Real ../Hyperreal/Transcendental*
begin

datatype *complex* = *Complex real real*

primrec
Re :: *complex* ⇒ *real*
where
Re: *Re* (*Complex x y*) = *x*

primrec
Im :: *complex* ⇒ *real*
where
Im: *Im* (*Complex x y*) = *y*

lemma *complex-surj* [*simp*]: *Complex* (*Re z*) (*Im z*) = *z*
by (*induct z*) *simp*

lemma *complex-equality* [*intro?*]: ⟦*Re x* = *Re y*; *Im x* = *Im y*⟧ ⟹ *x* = *y*
by (*induct x*, *induct y*) *simp*

lemma *expand-complex-eq*: *x* = *y* ⟷ *Re x* = *Re y* ∧ *Im x* = *Im y*
by (*induct x*, *induct y*) *simp*

lemmas *complex-Re-Im-cancel-iff* = *expand-complex-eq*

23.1 Addition and Subtraction

instantiation *complex* :: *ab-group-add*
begin

definition
complex-zero-def: $0 = Complex\ 0\ 0$

definition
complex-add-def: $x + y = Complex\ (Re\ x + Re\ y)\ (Im\ x + Im\ y)$

definition
complex-minus-def: $- x = Complex\ (- Re\ x)\ (- Im\ x)$

definition
complex-diff-def: $x - (y::complex) = x + - y$

lemma *Complex-eq-0* [*simp*]: $Complex\ a\ b = 0 \longleftrightarrow a = 0 \wedge b = 0$
by (*simp add*: *complex-zero-def*)

lemma *complex-Re-zero* [*simp*]: $Re\ 0 = 0$
by (*simp add*: *complex-zero-def*)

lemma *complex-Im-zero* [*simp*]: $Im\ 0 = 0$
by (*simp add*: *complex-zero-def*)

lemma *complex-add* [*simp*]:
$Complex\ a\ b + Complex\ c\ d = Complex\ (a + c)\ (b + d)$
by (*simp add*: *complex-add-def*)

lemma *complex-Re-add* [*simp*]: $Re\ (x + y) = Re\ x + Re\ y$
by (*simp add*: *complex-add-def*)

lemma *complex-Im-add* [*simp*]: $Im\ (x + y) = Im\ x + Im\ y$
by (*simp add*: *complex-add-def*)

lemma *complex-minus* [*simp*]:
$- (Complex\ a\ b) = Complex\ (- a)\ (- b)$
by (*simp add*: *complex-minus-def*)

lemma *complex-Re-minus* [*simp*]: $Re\ (- x) = - Re\ x$
by (*simp add*: *complex-minus-def*)

lemma *complex-Im-minus* [*simp*]: $Im\ (- x) = - Im\ x$
by (*simp add*: *complex-minus-def*)

lemma *complex-diff* [*simp*]:
$Complex\ a\ b - Complex\ c\ d = Complex\ (a - c)\ (b - d)$
by (*simp add*: *complex-diff-def*)

lemma *complex-Re-diff* [*simp*]: $Re\ (x - y) = Re\ x - Re\ y$
by (*simp add*: *complex-diff-def*)

lemma *complex-Im-diff* [*simp*]: $Im\ (x - y) = Im\ x - Im\ y$

by (*simp add*: *complex-diff-def*)

instance
by *intro-classes* (*simp-all add*: *complex-add-def complex-diff-def*)

end

23.2 Multiplication and Division

instantiation *complex* :: {*field*, *division-by-zero*}
begin

definition
complex-one-def: $1 = Complex\ 1\ 0$

definition
complex-mult-def: $x * y =$
$Complex\ (Re\ x * Re\ y - Im\ x * Im\ y)\ (Re\ x * Im\ y + Im\ x * Re\ y)$

definition
complex-inverse-def: $inverse\ x =$
$Complex\ (Re\ x\ /\ ((Re\ x)^2 + (Im\ x)^2))\ (-\ Im\ x\ /\ ((Re\ x)^2 + (Im\ x)^2))$

definition
complex-divide-def: $x\ /\ (y::complex) = x * inverse\ y$

lemma *Complex-eq-1* [*simp*]: $(Complex\ a\ b = 1) = (a = 1 \wedge b = 0)$
by (*simp add*: *complex-one-def*)

lemma *complex-Re-one* [*simp*]: $Re\ 1 = 1$
by (*simp add*: *complex-one-def*)

lemma *complex-Im-one* [*simp*]: $Im\ 1 = 0$
by (*simp add*: *complex-one-def*)

lemma *complex-mult* [*simp*]:
$Complex\ a\ b * Complex\ c\ d = Complex\ (a * c - b * d)\ (a * d + b * c)$
by (*simp add*: *complex-mult-def*)

lemma *complex-Re-mult* [*simp*]: $Re\ (x * y) = Re\ x * Re\ y - Im\ x * Im\ y$
by (*simp add*: *complex-mult-def*)

lemma *complex-Im-mult* [*simp*]: $Im\ (x * y) = Re\ x * Im\ y + Im\ x * Re\ y$
by (*simp add*: *complex-mult-def*)

lemma *complex-inverse* [*simp*]:
$inverse\ (Complex\ a\ b) = Complex\ (a\ /\ (a^2 + b^2))\ (-\ b\ /\ (a^2 + b^2))$
by (*simp add*: *complex-inverse-def*)

lemma *complex-Re-inverse*:
Re (*inverse x*) = *Re x* / ((*Re x*)^2 + (*Im x*)^2)
by (*simp add*: *complex-inverse-def*)

lemma *complex-Im-inverse*:
Im (*inverse x*) = − *Im x* / ((*Re x*)^2 + (*Im x*)^2)
by (*simp add*: *complex-inverse-def*)

instance
by *intro-classes* (*simp-all add*: *complex-mult-def*
right-distrib left-distrib right-diff-distrib left-diff-distrib
complex-inverse-def complex-divide-def
power2-eq-square add-divide-distrib [*symmetric*]
expand-complex-eq)

end

23.3 Exponentiation

instantiation *complex* :: *recpower*
begin

primrec *power-complex* **where**
complexpow-0: *z* ^ *0* = (*1*::*complex*)
| *complexpow-Suc*: *z* ^ *Suc n* = (*z*::*complex*) ∗ *z* ^ *n*

instance by *intro-classes simp-all*

end

23.4 Numerals and Arithmetic

instantiation *complex* :: *number-ring*
begin

definition *number-of-complex* **where**
complex-number-of-def: *number-of w* = (*of-int w* :: *complex*)

instance
by *intro-classes* (*simp only*: *complex-number-of-def*)

end

lemma *complex-Re-of-nat* [*simp*]: *Re* (*of-nat n*) = *of-nat n*
by (*induct n*) *simp-all*

lemma *complex-Im-of-nat* [*simp*]: *Im* (*of-nat n*) = *0*
by (*induct n*) *simp-all*

lemma *complex-Re-of-int* [*simp*]: *Re* (*of-int z*) = *of-int z*

by (*cases z rule*: *int-diff-cases*) *simp*

lemma *complex-Im-of-int* [*simp*]: *Im* (*of-int z*) = *0*
by (*cases z rule*: *int-diff-cases*) *simp*

lemma *complex-Re-number-of* [*simp*]: *Re* (*number-of v*) = *number-of v*
unfolding *number-of-eq* **by** (*rule complex-Re-of-int*)

lemma *complex-Im-number-of* [*simp*]: *Im* (*number-of v*) = *0*
unfolding *number-of-eq* **by** (*rule complex-Im-of-int*)

lemma *Complex-eq-number-of* [*simp*]:
(*Complex a b* = *number-of w*) = (*a* = *number-of w* ∧ *b* = *0*)
by (*simp add*: *expand-complex-eq*)

23.5 Scalar Multiplication

instantiation *complex* :: *real-field*
begin

definition
complex-scaleR-def: *scaleR r x* = *Complex* (*r* ∗ *Re x*) (*r* ∗ *Im x*)

lemma *complex-scaleR* [*simp*]:
scaleR r (*Complex a b*) = *Complex* (*r* ∗ *a*) (*r* ∗ *b*)
unfolding *complex-scaleR-def* **by** *simp*

lemma *complex-Re-scaleR* [*simp*]: *Re* (*scaleR r x*) = *r* ∗ *Re x*
unfolding *complex-scaleR-def* **by** *simp*

lemma *complex-Im-scaleR* [*simp*]: *Im* (*scaleR r x*) = *r* ∗ *Im x*
unfolding *complex-scaleR-def* **by** *simp*

instance
proof
fix *a b* :: *real* **and** *x y* :: *complex*
show *scaleR a* (*x* + *y*) = *scaleR a x* + *scaleR a y*
by (*simp add*: *expand-complex-eq right-distrib*)
show *scaleR* (*a* + *b*) *x* = *scaleR a x* + *scaleR b x*
by (*simp add*: *expand-complex-eq left-distrib*)
show *scaleR a* (*scaleR b x*) = *scaleR* (*a* ∗ *b*) *x*
by (*simp add*: *expand-complex-eq mult-assoc*)
show *scaleR 1 x* = *x*
by (*simp add*: *expand-complex-eq*)
show *scaleR a x* ∗ *y* = *scaleR a* (*x* ∗ *y*)
by (*simp add*: *expand-complex-eq ring-simps*)
show *x* ∗ *scaleR a y* = *scaleR a* (*x* ∗ *y*)
by (*simp add*: *expand-complex-eq ring-simps*)
qed

end

23.6 Properties of Embedding from Reals

abbreviation
complex-of-real :: *real* ⇒ *complex* **where**
complex-of-real ≡ *of-real*

lemma *complex-of-real-def*: *complex-of-real r* = *Complex r 0*
by (*simp add*: *of-real-def complex-scaleR-def*)

lemma *Re-complex-of-real* [*simp*]: *Re* (*complex-of-real z*) = *z*
by (*simp add*: *complex-of-real-def*)

lemma *Im-complex-of-real* [*simp*]: *Im* (*complex-of-real z*) = *0*
by (*simp add*: *complex-of-real-def*)

lemma *Complex-add-complex-of-real* [*simp*]:
Complex x y + *complex-of-real r* = *Complex* (*x*+*r*) *y*
by (*simp add*: *complex-of-real-def*)

lemma *complex-of-real-add-Complex* [*simp*]:
complex-of-real r + *Complex x y* = *Complex* (*r*+*x*) *y*
by (*simp add*: *complex-of-real-def*)

lemma *Complex-mult-complex-of-real*:
Complex x y ∗ *complex-of-real r* = *Complex* (*x*∗*r*) (*y*∗*r*)
by (*simp add*: *complex-of-real-def*)

lemma *complex-of-real-mult-Complex*:
complex-of-real r ∗ *Complex x y* = *Complex* (*r*∗*x*) (*r*∗*y*)
by (*simp add*: *complex-of-real-def*)

23.7 Vector Norm

instantiation *complex* :: *real-normed-field*
begin

definition
complex-norm-def: *norm z* = *sqrt* ((*Re z*)^2 + (*Im z*)^2)

abbreviation
cmod :: *complex* ⇒ *real* **where**
cmod ≡ *norm*

definition
complex-sgn-def: *sgn x* = *x* /$_R$ *cmod x*

lemmas *cmod-def* = *complex-norm-def*

lemma *complex-norm* [*simp*]: *cmod* (*Complex x y*) = *sqrt* (x^2 + y^2)
by (*simp add*: *complex-norm-def*)

instance
proof
fix *r* :: *real* **and** *x y* :: *complex*
show *0* ≤ *norm x*
by (*induct x*) *simp*
show (*norm x* = *0*) = (*x* = *0*)
by (*induct x*) *simp*
show *norm* (*x* + *y*) ≤ *norm x* + *norm y*
by (*induct x*, *induct y*)
(*simp add*: *real-sqrt-sum-squares-triangle-ineq*)
show *norm* (*scaleR r x*) = |*r*| * *norm x*
by (*induct x*)
(*simp add*: *power-mult-distrib right-distrib* [*symmetric*] *real-sqrt-mult*)
show *norm* (*x* * *y*) = *norm x* * *norm y*
by (*induct x*, *induct y*)
(*simp add*: *real-sqrt-mult* [*symmetric*] *power2-eq-square ring-simps*)
show *sgn x* = *x* /$_R$ *cmod x* **by**(*simp add*: *complex-sgn-def*)
qed

end

lemma *cmod-unit-one* [*simp*]: *cmod* (*Complex* (*cos a*) (*sin a*)) = *1*
by *simp*

lemma *cmod-complex-polar* [*simp*]:
cmod (*complex-of-real r* * *Complex* (*cos a*) (*sin a*)) = *abs r*
by (*simp add*: *norm-mult*)

lemma *complex-Re-le-cmod*: *Re x* ≤ *cmod x*
unfolding *complex-norm-def*
by (*rule real-sqrt-sum-squares-ge1*)

lemma *complex-mod-minus-le-complex-mod* [*simp*]: − *cmod x* ≤ *cmod x*
by (*rule order-trans* [*OF - norm-ge-zero*], *simp*)

lemma *complex-mod-triangle-ineq2* [*simp*]: *cmod*(*b* + *a*) − *cmod b* ≤ *cmod a*
by (*rule ord-le-eq-trans* [*OF norm-triangle-ineq2*], *simp*)

lemmas *real-sum-squared-expand* = *power2-sum* [**where** *'a*=*real*]

lemma *abs-Re-le-cmod*: |*Re x*| ≤ *cmod x*
by (*cases x*) *simp*

lemma *abs-Im-le-cmod*: |*Im x*| ≤ *cmod x*
by (*cases x*) *simp*

23.8 Completeness of the Complexes

interpretation *Re*: *bounded-linear* [*Re*]
apply (*unfold-locales*, *simp*, *simp*)
apply (*rule-tac x=1* **in** *exI*)
apply (*simp add*: *complex-norm-def*)
done

interpretation *Im*: *bounded-linear* [*Im*]
apply (*unfold-locales*, *simp*, *simp*)
apply (*rule-tac x=1* **in** *exI*)
apply (*simp add*: *complex-norm-def*)
done

lemma *LIMSEQ-Complex*:
⟦*X* ----> *a*; *Y* ----> *b*⟧ ⟹ (λ*n*. *Complex* (*X n*) (*Y n*)) ----> *Complex* *a b*
apply (*rule LIMSEQ-I*)
apply (*subgoal-tac 0 < r / sqrt 2*)
apply (*drule-tac r=r / sqrt 2* **in** *LIMSEQ-D*, *safe*)
apply (*drule-tac r=r / sqrt 2* **in** *LIMSEQ-D*, *safe*)
apply (*rename-tac M N*, *rule-tac x=max M N* **in** *exI*, *safe*)
apply (*simp add*: *real-sqrt-sum-squares-less*)
apply (*simp add*: *divide-pos-pos*)
done

instance *complex* :: *banach*
proof
fix *X* :: *nat* ⇒ *complex*
assume *X*: *Cauchy X*
from *Re.Cauchy* [*OF X*] **have** *1*: (λ*n*. *Re* (*X n*)) ----> *lim* (λ*n*. *Re* (*X n*))
by (*simp add*: *Cauchy-convergent-iff convergent-LIMSEQ-iff*)
from *Im.Cauchy* [*OF X*] **have** *2*: (λ*n*. *Im* (*X n*)) ----> *lim* (λ*n*. *Im* (*X n*))
by (*simp add*: *Cauchy-convergent-iff convergent-LIMSEQ-iff*)
have *X* ----> *Complex* (*lim* (λ*n*. *Re* (*X n*))) (*lim* (λ*n*. *Im* (*X n*)))
using *LIMSEQ-Complex* [*OF 1 2*] **by** *simp*
thus *convergent X*
by (*rule convergentI*)
qed

23.9 The Complex Number i

definition
ii :: *complex* (i) **where**
i-def: *ii* ≡ *Complex 0 1*

lemma *complex-Re-i* [*simp*]: *Re ii* = *0*
by (*simp add*: *i-def*)

lemma *complex-Im-i* [*simp*]: *Im ii* = *1*

by (*simp add*: *i-def*)

lemma *Complex-eq-i* [*simp*]: (*Complex x y* = *ii*) = (*x* = *0* ∧ *y* = *1*)
by (*simp add*: *i-def*)

lemma *complex-i-not-zero* [*simp*]: *ii* ≠ *0*
by (*simp add*: *expand-complex-eq*)

lemma *complex-i-not-one* [*simp*]: *ii* ≠ *1*
by (*simp add*: *expand-complex-eq*)

lemma *complex-i-not-number-of* [*simp*]: *ii* ≠ *number-of w*
by (*simp add*: *expand-complex-eq*)

lemma *i-mult-Complex* [*simp*]: *ii* ∗ *Complex a b* = *Complex* (− *b*) *a*
by (*simp add*: *expand-complex-eq*)

lemma *Complex-mult-i* [*simp*]: *Complex a b* ∗ *ii* = *Complex* (− *b*) *a*
by (*simp add*: *expand-complex-eq*)

lemma *i-complex-of-real* [*simp*]: *ii* ∗ *complex-of-real r* = *Complex 0 r*
by (*simp add*: *i-def complex-of-real-def*)

lemma *complex-of-real-i* [*simp*]: *complex-of-real r* ∗ *ii* = *Complex 0 r*
by (*simp add*: *i-def complex-of-real-def*)

lemma *i-squared* [*simp*]: *ii* ∗ *ii* = −*1*
by (*simp add*: *i-def*)

lemma *power2-i* [*simp*]: $ii^2 = -1$
by (*simp add*: *power2-eq-square*)

lemma *inverse-i* [*simp*]: *inverse ii* = − *ii*
by (*rule inverse-unique*, *simp*)

23.10 Complex Conjugation

definition
cnj :: *complex* ⇒ *complex* **where**
cnj z = *Complex* (*Re z*) (− *Im z*)

lemma *complex-cnj* [*simp*]: *cnj* (*Complex a b*) = *Complex a* (− *b*)
by (*simp add*: *cnj-def*)

lemma *complex-Re-cnj* [*simp*]: *Re* (*cnj x*) = *Re x*
by (*simp add*: *cnj-def*)

lemma *complex-Im-cnj* [*simp*]: *Im* (*cnj x*) = − *Im x*
by (*simp add*: *cnj-def*)

lemma *complex-cnj-cancel-iff* [*simp*]: $(cnj\ x = cnj\ y) = (x = y)$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-cnj* [*simp*]: $cnj\ (cnj\ z) = z$
by (*simp add*: *cnj-def*)

lemma *complex-cnj-zero* [*simp*]: $cnj\ 0 = 0$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-zero-iff* [*iff*]: $(cnj\ z = 0) = (z = 0)$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-add*: $cnj\ (x + y) = cnj\ x + cnj\ y$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-diff*: $cnj\ (x - y) = cnj\ x - cnj\ y$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-minus*: $cnj\ (- x) = - cnj\ x$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-one* [*simp*]: $cnj\ 1 = 1$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-mult*: $cnj\ (x * y) = cnj\ x * cnj\ y$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-inverse*: $cnj\ (inverse\ x) = inverse\ (cnj\ x)$
by (*simp add*: *complex-inverse-def*)

lemma *complex-cnj-divide*: $cnj\ (x\ /\ y) = cnj\ x\ /\ cnj\ y$
by (*simp add*: *complex-divide-def complex-cnj-mult complex-cnj-inverse*)

lemma *complex-cnj-power*: $cnj\ (x\ \hat{}\ n) = cnj\ x\ \hat{}\ n$
by (*induct n*, *simp-all add*: *complex-cnj-mult*)

lemma *complex-cnj-of-nat* [*simp*]: $cnj\ (of\text{-}nat\ n) = of\text{-}nat\ n$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-of-int* [*simp*]: $cnj\ (of\text{-}int\ z) = of\text{-}int\ z$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-number-of* [*simp*]: $cnj\ (number\text{-}of\ w) = number\text{-}of\ w$
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-scaleR*: $cnj\ (scaleR\ r\ x) = scaleR\ r\ (cnj\ x)$
by (*simp add*: *expand-complex-eq*)

lemma *complex-mod-cnj* [*simp*]: *cmod* (*cnj z*) = *cmod z*
by (*simp add*: *complex-norm-def*)

lemma *complex-cnj-complex-of-real* [*simp*]: *cnj* (*of-real x*) = *of-real x*
by (*simp add*: *expand-complex-eq*)

lemma *complex-cnj-i* [*simp*]: *cnj ii* = − *ii*
by (*simp add*: *expand-complex-eq*)

lemma *complex-add-cnj*: *z* + *cnj z* = *complex-of-real* (*2* ∗ *Re z*)
by (*simp add*: *expand-complex-eq*)

lemma *complex-diff-cnj*: *z* − *cnj z* = *complex-of-real* (*2* ∗ *Im z*) ∗ *ii*
by (*simp add*: *expand-complex-eq*)

lemma *complex-mult-cnj*: *z* ∗ *cnj z* = *complex-of-real* ((*Re z*)^2 + (*Im z*)^2)
by (*simp add*: *expand-complex-eq power2-eq-square*)

lemma *complex-mod-mult-cnj*: *cmod* (*z* ∗ *cnj z*) = (*cmod z*)^2
by (*simp add*: *norm-mult power2-eq-square*)

interpretation *cnj*: *bounded-linear* [*cnj*]
apply (*unfold-locales*)
apply (*rule complex-cnj-add*)
apply (*rule complex-cnj-scaleR*)
apply (*rule-tac x=1* **in** *exI*, *simp*)
done

23.11 The Functions *sgn* and *arg*

——— Argand ———-

definition
arg :: *complex* => *real* **where**
arg z = (*SOME a*. *Re*(*sgn z*) = *cos a* & *Im*(*sgn z*) = *sin a* & −*pi* < *a* & *a* ≤ *pi*)

lemma *sgn-eq*: *sgn z* = *z* / *complex-of-real* (*cmod z*)
by (*simp add*: *complex-sgn-def divide-inverse scaleR-conv-of-real mult-commute*)

lemma *i-mult-eq*: *ii* ∗ *ii* = *complex-of-real* (−*1*)
by (*simp add*: *i-def complex-of-real-def*)

lemma *i-mult-eq2* [*simp*]: *ii* ∗ *ii* = −(*1*::*complex*)
by (*simp add*: *i-def complex-one-def*)

lemma *complex-eq-cancel-iff2* [*simp*]:
(*Complex x y* = *complex-of-real xa*) = (*x* = *xa* & *y* = *0*)
by (*simp add*: *complex-of-real-def*)

lemma *Re-sgn* [*simp*]: *Re*(*sgn z*) = *Re*(*z*)/*cmod z*
by (*simp add*: *complex-sgn-def divide-inverse*)

lemma *Im-sgn* [*simp*]: *Im*(*sgn z*) = *Im*(*z*)/*cmod z*
by (*simp add*: *complex-sgn-def divide-inverse*)

lemma *complex-inverse-complex-split*:
inverse(*complex-of-real x* + *ii* ∗ *complex-of-real y*) =
complex-of-real(*x*/(*x* ^ 2 + *y* ^ 2)) −
ii ∗ *complex-of-real*(*y*/(*x* ^ 2 + *y* ^ 2))
by (*simp add*: *complex-of-real-def i-def diff-minus divide-inverse*)

lemma *cos-arg-i-mult-zero-pos*:
0 < *y* ==> *cos* (*arg*(*Complex* 0 *y*)) = 0
apply (*simp add*: *arg-def abs-if*)
apply (*rule-tac a* = *pi*/2 **in** *someI2*, *auto*)
apply (*rule order-less-trans* [*of* - 0], *auto*)
done

lemma *cos-arg-i-mult-zero-neg*:
y < 0 ==> *cos* (*arg*(*Complex* 0 *y*)) = 0
apply (*simp add*: *arg-def abs-if*)
apply (*rule-tac a* = − *pi*/2 **in** *someI2*, *auto*)
apply (*rule order-trans* [*of* - 0], *auto*)
done

lemma *cos-arg-i-mult-zero* [*simp*]:
y ≠ 0 ==> *cos* (*arg*(*Complex* 0 *y*)) = 0
by (*auto simp add*: *linorder-neq-iff cos-arg-i-mult-zero-pos cos-arg-i-mult-zero-neg*)

23.12 Finally! Polar Form for Complex Numbers

definition

cis :: *real* => *complex* **where**
cis a = *Complex* (*cos a*) (*sin a*)

definition

rcis :: [*real*, *real*] => *complex* **where**
rcis r a = *complex-of-real r* ∗ *cis a*

definition

expi :: *complex* => *complex* **where**
expi z = *complex-of-real*(*exp* (*Re z*)) * *cis* (*Im z*)

lemma *complex-split-polar*:
∃ *r a*. *z* = *complex-of-real r* * (*Complex* (*cos a*) (*sin a*))
apply (*induct z*)
apply (*auto simp add*: *polar-Ex complex-of-real-mult-Complex*)
done

lemma *rcis-Ex*: ∃ *r a*. *z* = *rcis r a*
apply (*induct z*)
apply (*simp add*: *rcis-def cis-def polar-Ex complex-of-real-mult-Complex*)
done

lemma *Re-rcis* [*simp*]: *Re*(*rcis r a*) = *r* * *cos a*
by (*simp add*: *rcis-def cis-def*)

lemma *Im-rcis* [*simp*]: *Im*(*rcis r a*) = *r* * *sin a*
by (*simp add*: *rcis-def cis-def*)

lemma *sin-cos-squared-add2-mult*: (*r* * *cos a*)^2 + (*r* * *sin a*)^2 = *r*^2
proof −
have (*r* * *cos a*)^2 + (*r* * *sin a*)^2 = *r*^2 * ((*cos a*)^2 + (*sin a*)^2)
by (*simp only*: *power-mult-distrib right-distrib*)
thus *?thesis* **by** *simp*
qed

lemma *complex-mod-rcis* [*simp*]: *cmod*(*rcis r a*) = *abs r*
by (*simp add*: *rcis-def cis-def sin-cos-squared-add2-mult*)

lemma *complex-mod-sqrt-Re-mult-cnj*: *cmod z* = *sqrt* (*Re* (*z* * *cnj z*))
by (*simp add*: *cmod-def power2-eq-square*)

lemma *complex-In-mult-cnj-zero* [*simp*]: *Im* (*z* * *cnj z*) = *0*
by *simp*

lemma *cis-rcis-eq*: *cis a* = *rcis 1 a*
by (*simp add*: *rcis-def*)

lemma *rcis-mult*: *rcis r1 a* * *rcis r2 b* = *rcis* (*r1**r2*) (*a* + *b*)
by (*simp add*: *rcis-def cis-def cos-add sin-add right-distrib right-diff-distrib*
complex-of-real-def)

lemma *cis-mult*: *cis a ∗ cis b = cis (a + b)*
by (*simp add*: *cis-rcis-eq rcis-mult*)

lemma *cis-zero* [*simp*]: *cis 0 = 1*
by (*simp add*: *cis-def complex-one-def*)

lemma *rcis-zero-mod* [*simp*]: *rcis 0 a = 0*
by (*simp add*: *rcis-def*)

lemma *rcis-zero-arg* [*simp*]: *rcis r 0 = complex-of-real r*
by (*simp add*: *rcis-def*)

lemma *complex-of-real-minus-one*:
complex-of-real (−(1::*real)) = −(1*::*complex)*
by (*simp add*: *complex-of-real-def complex-one-def*)

lemma *complex-i-mult-minus* [*simp*]: *ii ∗ (ii ∗ x) = − x*
by (*simp add*: *mult-assoc* [*symmetric*])

lemma *cis-real-of-nat-Suc-mult*:
cis (real (Suc n) ∗ a) = cis a ∗ cis (real n ∗ a)
by (*simp add*: *cis-def real-of-nat-Suc left-distrib cos-add sin-add right-distrib*)

lemma *DeMoivre*: *(cis a) ^ n = cis (real n ∗ a)*
apply (*induct-tac n*)
apply (*auto simp add*: *cis-real-of-nat-Suc-mult*)
done

lemma *DeMoivre2*: *(rcis r a) ^ n = rcis (r ^ n) (real n ∗ a)*
by (*simp add*: *rcis-def power-mult-distrib DeMoivre*)

lemma *cis-inverse* [*simp*]: *inverse(cis a) = cis (−a)*
by (*simp add*: *cis-def complex-inverse-complex-split diff-minus*)

lemma *rcis-inverse*: *inverse(rcis r a) = rcis (1/r) (−a)*
by (*simp add*: *divide-inverse rcis-def*)

lemma *cis-divide*: *cis a / cis b = cis (a − b)*
by (*simp add*: *complex-divide-def cis-mult real-diff-def*)

lemma *rcis-divide*: *rcis r1 a / rcis r2 b = rcis (r1/r2) (a − b)*
apply (*simp add*: *complex-divide-def*)
apply (*case-tac r2=0*, *simp*)
apply (*simp add*: *rcis-inverse rcis-mult real-diff-def*)
done

lemma *Re-cis* [*simp*]: *Re(cis a) = cos a*
by (*simp add*: *cis-def*)

lemma *Im-cis* [*simp*]: *Im*(*cis a*) = *sin a*
by (*simp add*: *cis-def*)

lemma *cos-n-Re-cis-pow-n*: *cos* (*real n* ∗ *a*) = *Re*(*cis a* ^ *n*)
by (*auto simp add*: *DeMoivre*)

lemma *sin-n-Im-cis-pow-n*: *sin* (*real n* ∗ *a*) = *Im*(*cis a* ^ *n*)
by (*auto simp add*: *DeMoivre*)

lemma *expi-add*: *expi*(*a* + *b*) = *expi*(*a*) ∗ *expi*(*b*)
by (*simp add*: *expi-def exp-add cis-mult* [*symmetric*] *mult-ac*)

lemma *expi-zero* [*simp*]: *expi* (*0*::*complex*) = *1*
by (*simp add*: *expi-def*)

lemma *complex-expi-Ex*: ∃ *a r*. *z* = *complex-of-real r* ∗ *expi a*
apply (*insert rcis-Ex* [*of z*])
apply (*auto simp add*: *expi-def rcis-def mult-assoc* [*symmetric*])
apply (*rule-tac x* = *ii* ∗ *complex-of-real a* **in** *exI*, *auto*)
done

lemma *expi-two-pi-i* [*simp*]: *expi*((*2*::*complex*) ∗ *complex-of-real pi* ∗ *ii*) = *1*
by (*simp add*: *expi-def cis-def*)

end

24 Fundamental-Theorem-Algebra: Fundamental Theorem of Algebra

theory *Fundamental-Theorem-Algebra*
imports *Univ-Poly Dense-Linear-Order Complex*
begin

25 Square root of complex numbers

definition *csqrt* :: *complex* ⇒ *complex* **where**
csqrt z = (*if Im z* = *0 then*
if 0 ≤ *Re z then Complex* (*sqrt*(*Re z*)) *0*
else Complex 0 (*sqrt*(− *Re z*))
else Complex (*sqrt*((*cmod z* + *Re z*) */2*))
((*Im z* / *abs*(*Im z*)) ∗ *sqrt*((*cmod z* − *Re z*) */2*)))

lemma *csqrt*: *csqrt z* ^ *2* = *z*
proof−
obtain *x y* **where** *xy*: *z* = *Complex x y* **by** (*cases z*, *simp-all*)
{assume *y0*: *y* = *0*

{assume *x0*: $x \geq 0$
then have *?thesis* **using** *y0 xy real-sqrt-pow2*[*OF x0*]
by (*simp add*: *csqrt-def power2-eq-square*)**}**
moreover
{assume $\neg\ x \geq 0$ **hence** *x0*: $-\ x \geq 0$ **by** *arith*
then have *?thesis* **using** *y0 xy real-sqrt-pow2*[*OF x0*]
by (*simp add*: *csqrt-def power2-eq-square*) **}**
ultimately have *?thesis* **by** *blast***}**
moreover
{assume *y0*: $y \neq 0$
{fix *x y*
let *?z* = *Complex x y*
from *abs-Re-le-cmod*[*of ?z*] **have** *tha*: *abs x* $\leq$ *cmod ?z* **by** *auto*
hence *cmod ?z* $-$ *x* $\geq$ *0 cmod ?z* $+$ *x* $\geq$ *0* **by** (*cases x* $\geq$ *0*, *arith*$+$)
hence (*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $+$ *x*) / *2* $\geq$ *0* (*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $-$ *x*) / *2* $\geq$ *0* **by** (*simp-all add*: *power2-eq-square*) **}**
note *th* = *this*
have *sq4*: $\bigwedge$*x*::*real*. *x*^*2* / *4* = (*x* / *2*) ^ *2*
by (*simp add*: *power2-eq-square*)
from *th*[*of x y*]
have *sq4'*: *sqrt* (((*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $+$ *x*)^*2* / *4*)) = (*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $+$ *x*) / *2 sqrt* (((*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $-$ *x*)^*2* / *4*)) = (*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $-$ *x*) / *2* **unfolding** *sq4* **by** *simp-all*
then have *th1*: *sqrt* ((*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $+$ *x*) $*$ (*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $+$ *x*) / *4*) $-$ *sqrt* ((*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $-$ *x*) $*$ (*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $-$ *x*) / *4*) = *x*
unfolding *power2-eq-square* **by** *simp*
have *sqrt 4* = *sqrt* (*2*^*2*) **by** *simp*
hence *sqrt4*: *sqrt 4* = *2* **by** (*simp only*: *real-sqrt-abs*)
have *th2*: *2* $*$(*y* $*$ *sqrt* ((*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $-$ *x*) $*$ (*sqrt* (*x* $*$ *x* $+$ *y* $*$ *y*) $+$ *x*) / *4*)) / |*y*| = *y*
using *iffD2*[*OF real-sqrt-pow2-iff sum-power2-ge-zero*[*of x y*]] *y0*
unfolding *power2-eq-square*
by (*simp add*: *ring-simps real-sqrt-divide sqrt4*)
from *y0 xy* **have** *?thesis* **apply** (*simp add*: *csqrt-def power2-eq-square*)
apply (*simp add*: *real-sqrt-sum-squares-mult-ge-zero*[*of x y*] *real-sqrt-pow2*[*OF th(1)*[*of x y*], *unfolded power2-eq-square*] *real-sqrt-pow2*[*OF th(2)*[*of x y*], *unfolded power2-eq-square*] *real-sqrt-mult*[*symmetric*])
using *th1 th2* **..}**
ultimately show *?thesis* **by** *blast*
qed

26 More lemmas about module of complex numbers

lemma *complex-of-real-power*: *complex-of-real x* ^ *n* = *complex-of-real* (*x*^*n*)
by (*induct n*, *auto*)

lemma *cmod-pos*: *cmod z* $\geq$ *0* **by** *simp*

lemma *complex-mod-triangle-ineq*: *cmod* $(z + w) \leq$ *cmod* $z +$ *cmod* w
using *complex-mod-triangle-ineq2*[*of z w*] **by** (*simp add*: *ring-simps*)

lemma *cmod-mult*: *cmod* $(z*w) =$ *cmod* $z *$ *cmod* w
proof−
from *rcis-Ex*[*of z*] *rcis-Ex*[*of w*]
obtain *rz az rw aw* **where** *z*: $z =$ *rcis rz az* **and** *w*: $w =$ *rcis rw aw* **by** *blast*
thus *?thesis* **by** (*simp add*: *rcis-mult abs-mult*)
qed

lemma *cmod-divide*: *cmod* $(z/w) =$ *cmod* $z /$ *cmod* w
proof−
from *rcis-Ex*[*of z*] *rcis-Ex*[*of w*]
obtain *rz az rw aw* **where** *z*: $z =$ *rcis rz az* **and** *w*: $w =$ *rcis rw aw* **by** *blast*
thus *?thesis* **by** (*simp add*: *rcis-divide*)
qed

lemma *cmod-inverse*: *cmod* (*inverse z*) = *inverse* (*cmod z*)
using *cmod-divide*[*of 1 z*] **by** (*simp add*: *inverse-eq-divide*)

lemma *cmod-uminus*: *cmod* $(- z) =$ *cmod* z
unfolding *cmod-def* **by** *simp*
lemma *cmod-abs-norm*: $|$*cmod* $w -$ *cmod* $z| \leq$ *cmod* $(w - z)$
proof−
have *ath*: $\bigwedge$(*a*::*real*) *b x*. *a* − *b* <= *x* $\Longrightarrow$ *b* − *a* <= *x* ==> *abs*(*a* − *b*) <= *x*
by *arith*
from *complex-mod-triangle-ineq2*[*of w − z z*]
have *th1*: *cmod w* − *cmod z* ≤ *cmod* (*w* − *z*) **by** *simp*
from *complex-mod-triangle-ineq2*[*of* − (*w* − *z*) *w*]
have *th2*: *cmod z* − *cmod w* ≤ *cmod* (*w* − *z*) **using** *cmod-uminus* [*of w* − *z*]
by *simp*
from *ath*[*OF th1 th2*] **show** *?thesis* **.**
qed

lemma *cmod-power*: *cmod* (*z* ^*n*) = *cmod z* ^ *n* **by** (*induct n*, *auto simp add*: *cmod-mult*)
lemma *real-down2*: (*0*::*real*) < *d1* $\Longrightarrow$ *0* < *d2* ==> *EX e*. *0* < *e* & *e* < *d1* & *e* < *d2*
apply *ferrack* **apply** *arith* **done**

lemma *cmod-complex-of-real*: *cmod* (*complex-of-real x*) = $|x|$
unfolding *cmod-def* **by** *auto*

The triangle inequality for cmod

lemma *complex-mod-triangle-sub*: *cmod w* ≤ *cmod* (*w* + *z*) + *norm z*
using *complex-mod-triangle-ineq2*[*of w* + *z* −*z*] **by** *auto*

27 Basic lemmas about complex polynomials

lemma *poly-bound-exists*:
shows $\exists m.\ m > 0 \wedge (\forall z.\ cmod\ z <= r \longrightarrow cmod\ (poly\ p\ z) \leq m)$
proof(*induct p*)
case *Nil* **thus** *?case* **by** (*rule exI*[**where** *x=1*], *simp*)
next
case (*Cons c cs*)
from *Cons.hyps* **obtain** *m* **where** *m*: $\forall z.\ cmod\ z \leq r \longrightarrow cmod\ (poly\ cs\ z) \leq m$
by *blast*
let *?k* = $1 + cmod\ c + |r * m|$
have *kp*: *?k* > *0* **using** *abs-ge-zero*[*of r*∗*m*] *cmod-pos*[*of c*] **by** *arith*
{fix *z*
assume *H*: $cmod\ z \leq r$
from *m H* **have** *th*: $cmod\ (poly\ cs\ z) \leq m$ **by** *blast*
from *H* **have** *rp*: $r \geq 0$ **using** *cmod-pos*[*of z*] **by** *arith*
have $cmod\ (poly\ (c\ \#\ cs)\ z) \leq cmod\ c + cmod\ (z* poly\ cs\ z)$
using *complex-mod-triangle-ineq*[*of c z*∗ *poly cs z*] **by** *simp*
also have $\ldots \leq cmod\ c + r*m$ **using** *mult-mono*[*OF H th rp cmod-pos*[*of poly cs z*]] **by** (*simp add*: *cmod-mult*)
also have $\ldots \leq$ *?k* **by** *simp*
finally have $cmod\ (poly\ (c\ \#\ cs)\ z) \leq$ *?k* **.}**
with *kp* **show** *?case* **by** *blast*
qed

Offsetting the variable in a polynomial gives another of same degree

lemma *poly-offset-lemma*:
shows $\exists b\ q.\ (length\ q = length\ p) \wedge (\forall x.\ poly\ (b\#q)\ (x::complex) = (a + x) * poly\ p\ x)$
proof(*induct p*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons c cs*)
from *Cons.hyps* **obtain** *b q* **where**
bq: $length\ q = length\ cs\ \forall x.\ poly\ (b\ \#\ q)\ x = (a + x) * poly\ cs\ x$
by *blast*
let *?b* = *a*∗*c*
let *?q* = (*b*+*c*)#*q*
have *lg*: *length ?q* = *length* (*c*#*cs*) **using** *bq*(*1*) **by** *simp*
{fix *x*
from *bq*(*2*)[*rule-format*, *of x*]
have *x*∗*poly* (*b* # *q*) *x* = *x*∗((*a* + *x*) ∗ *poly cs x*) **by** *simp*
hence *poly* (*?b*# *?q*) *x* = (*a* + *x*) ∗ *poly* (*c* # *cs*) *x*
by (*simp add*: *ring-simps*)**}**
with *lg* **show** *?case* **by** *blast*
qed

lemma *poly-offset*: $\exists\ q.\ length\ q = length\ p \wedge (\forall x.\ poly\ q\ (x::complex) = poly\ p\ (a + x))$
proof (*induct p*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons c cs*)
from *Cons.hyps* **obtain** *q* **where** *q*: *length q* = *length cs* $\forall x.$ *poly q x* = *poly cs* (*a* + *x*) **by** *blast*
from *poly-offset-lemma*[*of q a*] **obtain** *b p* **where**
bp: *length p* = *length q* $\forall x.$ *poly* (*b* # *p*) *x* = (*a* + *x*) ∗ *poly q x*
by *blast*
thus *?case* **using** *q bp* **by** − (*rule exI*[**where** *x*=(*c* + *b*)#*p*], *simp*)
qed

An alternative useful formulation of completeness of the reals

lemma *real-sup-exists*: **assumes** *ex*: $\exists x.\ P\ x$ **and** *bz*: $\exists z.\ \forall x.\ P\ x \longrightarrow x < z$
shows $\exists (s::real).\ \forall y.\ (\exists x.\ P\ x \wedge y < x) \longleftrightarrow y < s$
proof−
from *ex bz* **obtain** *x Y* **where** *x*: *P x* **and** *Y*: $\bigwedge x.\ P\ x \Longrightarrow x < Y$ **by** *blast*
from *ex* **have** *thx*:$\exists x.\ x \in Collect\ P$ **by** *blast*
from *bz* **have** *thY*: $\exists Y.\ isUb\ UNIV\ (Collect\ P)\ Y$
by(*auto simp add*: *isUb-def isLub-def setge-def setle-def leastP-def Ball-def order-le-less*)
from *reals-complete*[*OF thx thY*] **obtain** *L* **where** *L*: *isLub UNIV* (*Collect P*) *L*
by *blast*
from *Y*[*OF x*] **have** *xY*: *x* < *Y* **.**
from *L* **have** *L'*: $\forall x.\ P\ x \longrightarrow x \leq L$ **by** (*auto simp add*: *isUb-def isLub-def setge-def setle-def leastP-def Ball-def*)
from *Y* **have** *Y'*: $\forall x.\ P\ x \longrightarrow x \leq Y$
apply (*clarsimp*, *atomize* (*full*)) **by** *auto*
from *L Y'* **have** *L* ≤ *Y* **by** (*auto simp add*: *isUb-def isLub-def setge-def setle-def leastP-def Ball-def*)
{fix *y*
{fix *z* **assume** *z*: *P z y* < *z*
from *L' z* **have** *y* < *L* **by** *auto* **}**
moreover
{assume *yL*: *y* < *L* $\forall z.$ *P z* $\longrightarrow \neg$ *y* < *z*
hence *nox*: $\forall z.$ *P z* $\longrightarrow$ *y* ≥ *z* **by** *auto*
from *nox L* **have** *y* ≥ *L* **by** (*auto simp add*: *isUb-def isLub-def setge-def setle-def leastP-def Ball-def*)
with *yL*(*1*) **have** *False* **by** *arith***}**
ultimately have $(\exists x.\ P\ x \wedge y < x) \longleftrightarrow y < L$ **by** *blast***}**
thus *?thesis* **by** *blast*
qed

28 Some theorems about Sequences

Given a binary function $f :: nat \Rightarrow 'a \Rightarrow 'a$, its values are uniquely determined by a function g

lemma *num-Axiom*: $EX!\ g.\ g\ 0 = e \wedge (\forall n.\ g\ (Suc\ n) = f\ n\ (g\ n))$
unfolding *Ex1-def*
apply (*rule-tac x=nat-rec e f* **in** *exI*)
apply (*rule conjI*)+
apply (*rule def-nat-rec-0*, *simp*)
apply (*rule allI*, *rule def-nat-rec-Suc*, *simp*)
apply (*rule allI*, *rule impI*, *rule ext*)
apply (*erule conjE*)
apply (*induct-tac x*)
apply (*simp add*: *nat-rec-0*)
apply (*erule-tac x=n* **in** *allE*)
apply (*simp*)
done

An equivalent formulation of monotony – Not used here, but might be useful

lemma *mono-Suc*: $mono\ f = (\forall n.\ (f\ n :: 'a :: order) \leq f\ (Suc\ n))$
unfolding *mono-def*
proof *auto*
fix $A\ B :: nat$
assume H: $\forall n.\ f\ n \leq f\ (Suc\ n)\ A \leq B$
hence $\exists k.\ B = A + k$ **apply** $-$ **apply** (*thin-tac* $\forall n.\ f\ n \leq f\ (Suc\ n)$)
by *presburger*
then obtain k **where** k: $B = A + k$ **by** *blast*
{fix $a\ k$
have $f\ a \leq f\ (a + k)$
proof (*induct k*)
case *0* **thus** *?case* **by** *simp*
next
case (*Suc k*)
from *Suc.hyps H(1)*[*rule-format*, *of a + k*] **show** *?case* **by** *simp*
qed}
with k **show** $f\ A \leq f\ B$ **by** *blast*
qed

for any sequence, there is a mootonic subsequence

lemma *seq-monosub*: $\exists f.\ subseq\ f \wedge monoseq\ (\lambda\ n.\ (s\ (f\ n)))$
proof−
{assume H: $\forall n.\ \exists p > n.\ \forall\ m \geq p.\ s\ m \leq s\ p$
let *?P* $= \lambda\ p\ n.\ p > n \wedge (\forall m \geq p.\ s\ m \leq s\ p)$
from *num-Axiom*[*of SOME p. ?P p 0 λp n. SOME p. ?P p n*]
obtain f **where** f: $f\ 0 = (SOME\ p.\ ?P\ p\ 0)\ \forall n.\ f\ (Suc\ n) = (SOME\ p.\ ?P\ p\ (f\ n))$ **by** *blast*
have *?P* $(f\ 0)\ 0$ **unfolding** $f(1)$ *some-eq-ex*[*of* $\lambda p.$ *?P p 0*]
using H **apply** $-$

```
    apply (erule allE[where x=0], erule exE, rule-tac x=p in exI)
    unfolding order-le-less by blast
  hence f0: f 0 > 0 ∀ m ≥ f 0. s m ≤ s (f 0) by blast+
  {fix n
   have ?P (f (Suc n)) (f n)
     unfolding f(2)[rule-format, of n] some-eq-ex[of λp. ?P p (f n)]
     using H apply −
    apply (erule allE[where x=f n], erule exE, rule-tac x=p in exI)
    unfolding order-le-less by blast
  hence f (Suc n) > f n ∀ m ≥ f (Suc n). s m ≤ s (f (Suc n)) by blast+}
note fSuc = this
  {fix p q assume pq: p ≥ f q
   have s p ≤ s(f(q))  using f0(2)[rule-format, of p] pq fSuc
     by (cases q, simp-all) }
  note pqth = this
  {fix q
   have f (Suc q) > f q apply (induct q)
     using f0(1) fSuc(1)[of 0] apply simp by (rule fSuc(1))}
  note fss = this
  from fss have th1: subseq f unfolding subseq-Suc-iff ..
  {fix a b
   have f a ≤ f (a + b)
   proof(induct b)
     case 0 thus ?case by simp
   next
     case (Suc b)
     from fSuc(1)[of a + b] Suc.hyps show ?case by simp
   qed}
  note fmon0 = this
  have monoseq (λn. s (f n))
  proof−
    {fix n
     have s (f n) ≥ s (f (Suc n))
     proof(cases n)
       case 0
       assume n0: n = 0
       from fSuc(1)[of 0] have th0: f 0 ≤ f (Suc 0) by simp
       from f0(2)[rule-format, OF th0] show ?thesis  using n0 by simp
     next
       case (Suc m)
       assume m: n = Suc m
       from fSuc(1)[of n] m have th0: f (Suc m) ≤ f (Suc (Suc m)) by simp
       from m fSuc(2)[rule-format, OF th0] show ?thesis by simp
     qed}
    thus monoseq (λn. s (f n)) unfolding monoseq-Suc by blast
  qed
  with th1 have ?thesis by blast}
moreover
{fix N assume N: ∀ p >N. ∃ m≥p. s m > s p
```

```
  {fix p assume p: p ≥ Suc N
   hence pN: p > N by arith with N obtain m where m: m ≥ p s m > s p
by blast
   have m ≠ p using m(2) by auto
   with m have ∃m>p. s p < s m by − (rule exI[where x=m], auto)}
  note th0 = this
  let ?P = λm x. m > x ∧ s x < s m
  from num-Axiom[of SOME x. ?P x (Suc N) λm x. SOME y. ?P y x]
  obtain f where f: f 0 = (SOME x. ?P x (Suc N))
   ∀n. f (Suc n) = (SOME m. ?P m (f n)) by blast
  have ?P (f 0) (Suc N)  unfolding f(1) some-eq-ex[of λp. ?P p (Suc N)]
   using N apply −
   apply (erule allE[where x=Suc N], clarsimp)
   apply (rule-tac x=m in exI)
   apply auto
   apply (subgoal-tac Suc N ≠ m)
   apply simp
   apply (rule ccontr, simp)
   done
  hence f0: f 0 > Suc N s (Suc N) < s (f 0) by blast+
  {fix n
   have f n > N ∧ ?P (f (Suc n)) (f n)
    unfolding f(2)[rule-format, of n] some-eq-ex[of λp. ?P p (f n)]
   proof (induct n)
    case 0 thus ?case
     using f0 N apply auto
     apply (erule allE[where x=f 0], clarsimp)
     apply (rule-tac x=m in exI, simp)
     by (subgoal-tac f 0 ≠ m, auto)
   next
    case (Suc n)
    from Suc.hyps have Nfn: N < f n by blast
    from Suc.hyps obtain m where m: m > f n s (f n) < s m by blast
    with Nfn have mN: m > N by arith
     note key = Suc.hyps[unfolded some-eq-ex[of λp. ?P p (f n), symmetric]
f(2)[rule-format, of n, symmetric]]

    from key have th0: f (Suc n) > N by simp
    from N[rule-format, OF th0]
    obtain m' where m': m' ≥ f (Suc n) s (f (Suc n)) < s m' by blast
    have m' ≠ f (Suc (n)) apply (rule ccontr) using m'(2) by auto
    hence m' > f (Suc n) using m'(1) by simp
    with key m'(2) show ?case by auto
   qed}
  note fSuc = this
  {fix n
   have f n ≥ Suc N ∧ f(Suc n) > f n ∧ s(f n) < s(f(Suc n)) using fSuc[of
n] by auto
   hence f n ≥ Suc N f(Suc n) > f n s(f n) < s(f(Suc n)) by blast+}
```

```
    note thf = this
    have sqf: subseq f unfolding subseq-Suc-iff using thf by simp
    have monoseq (λn. s (f n)) unfolding monoseq-Suc using thf
      apply −
      apply (rule disjI1)
      apply auto
      apply (rule order-less-imp-le)
      apply blast
      done
    then have ?thesis using sqf by blast}
  ultimately show ?thesis unfolding linorder-not-less[symmetric] by blast
qed

lemma seq-suble: assumes sf: subseq f shows n ≤ f n
proof(induct n)
  case 0 thus ?case by simp
next
  case (Suc n)
  from sf[unfolded subseq-Suc-iff, rule-format, of n] Suc.hyps
  have n < f (Suc n) by arith
  thus ?case by arith
qed
```

29 Fundamental theorem of algebra

```
lemma unimodular-reduce-norm:
  assumes md: cmod z = 1
  shows cmod (z + 1) < 1 ∨ cmod (z − 1) < 1 ∨ cmod (z + ii) < 1 ∨ cmod (z
− ii) < 1
proof−
  obtain x y where z: z = Complex x y by (cases z, auto)
  from md z have xy: x^2 + y^2 = 1 by (simp add: cmod-def)
  {assume C: cmod (z + 1) ≥ 1 cmod (z − 1) ≥ 1 cmod (z + ii) ≥ 1 cmod (z
− ii) ≥ 1
    from C z xy have 2∗x ≤ 1 2∗x ≥ −1 2∗y ≤ 1 2∗y ≥ −1
      by (simp-all add: cmod-def power2-eq-square ring-simps)
    hence abs (2∗x) ≤ 1 abs (2∗y) ≤ 1 by simp-all
    hence (abs (2 ∗ x))^2 <= 1^2 (abs (2 ∗ y)) ^2 <= 1^2
      by − (rule power-mono, simp, simp)+
    hence th0: 4∗x^2 ≤ 1 4∗y^2 ≤ 1
      by (simp-all add: power2-abs power-mult-distrib)
    from add-mono[OF th0] xy have False by simp }
  thus ?thesis unfolding linorder-not-le[symmetric] by blast
qed
```

Hence we can always reduce modulus of $1 + b\ z\hat{}n$ if nonzero

```
lemma reduce-poly-simple:
 assumes b: b ≠ 0 and n: n≠0
  shows ∃z. cmod (1 + b ∗ z^n) < 1
```

using *n*
proof(*induct n rule*: *nat-less-induct*)
fix *n*
assume *IH*: $\forall m{<}n.\ m \neq 0 \longrightarrow (\exists z.\ cmod\ (1 + b * z\ \hat{}\ m) < 1)$ **and** *n*: $n \neq 0$
let *?P* = $\lambda z\ n.\ cmod\ (1 + b * z\ \hat{}\ n) < 1$
{assume *e*: *even n*
hence $\exists m.\ n = 2{*}m$ **by** *presburger*
then obtain *m* **where** *m*: $n = 2{*}m$ **by** *blast*
from *n m* **have** $m{\neq}0\ m < n$ **by** *presburger+*
with *IH*[*rule-format*, *of m*] **obtain** *z* **where** *z*: *?P z m* **by** *blast*
from *z* **have** *?P* (*csqrt z*) *n* **by** (*simp add*: *m power-mult csqrt*)
hence $\exists z.$ *?P z n* **..}**
moreover
{assume *o*: *odd n*
from *b* **have** *b'*: $b\hat{}2 \neq 0$ **unfolding** *power2-eq-square* **by** *simp*
have *Im* (*inverse b*) * (*Im* (*inverse b*) * |*Im b* * *Im b* + *Re b* * *Re b*|) +
Re (*inverse b*) * (*Re* (*inverse b*) * |*Im b* * *Im b* + *Re b* * *Re b*|) =
((*Re* (*inverse b*))$\hat{}2$ + (*Im* (*inverse b*))$\hat{}2$) * |*Im b* * *Im b* + *Re b* * *Re b*| **by** *algebra*
also have ... = *cmod* (*inverse b*) $\hat{}\ 2$ * *cmod b* $\hat{}\ 2$
apply (*simp add*: *cmod-def*) **using** *realpow-two-le-add-order*[*of Re b Im b*]
by (*simp add*: *power2-eq-square*)
finally
have *th0*: *Im* (*inverse b*) * (*Im* (*inverse b*) * |*Im b* * *Im b* + *Re b* * *Re b*|) +
Re (*inverse b*) * (*Re* (*inverse b*) * |*Im b* * *Im b* + *Re b* * *Re b*|) =
1
apply (*simp add*: *power2-eq-square cmod-mult*[*symmetric*] *cmod-inverse*[*symmetric*])
using *right-inverse*[*OF b'*]
by (*simp add*: *power2-eq-square*[*symmetric*] *power-inverse*[*symmetric*] *ring-simps*)
have *th0*: *cmod* (*complex-of-real* (*cmod b*) / *b*) = *1*
apply (*simp add*: *complex-Re-mult cmod-def power2-eq-square Re-complex-of-real Im-complex-of-real divide-inverse ring-simps*)
by (*simp add*: *real-sqrt-mult*[*symmetric*] *th0*)
from *o* **have** $\exists m.\ n = Suc\ (2{*}m)$ **by** *presburger+*
then obtain *m* **where** *m*: $n = Suc\ (2{*}m)$ **by** *blast*
from *unimodular-reduce-norm*[*OF th0*] *o*
have $\exists v.\ cmod\ (complex\text{-}of\text{-}real\ (cmod\ b) / b + v\hat{}n) < 1$
apply (*cases cmod* (*complex-of-real* (*cmod b*) / *b* + *1*) < *1*, *rule-tac x=1* **in** *exI*, *simp*)
apply (*cases cmod* (*complex-of-real* (*cmod b*) / *b* − *1*) < *1*, *rule-tac x=−1* **in** *exI*, *simp add*: *diff-def*)
apply (*cases cmod* (*complex-of-real* (*cmod b*) / *b* + *ii*) < *1*)
apply (*cases even m*, *rule-tac x=ii* **in** *exI*, *simp add*: *m power-mult*)
apply (*rule-tac x=− ii* **in** *exI*, *simp add*: *m power-mult*)
apply (*cases even m*, *rule-tac x=− ii* **in** *exI*, *simp add*: *m power-mult diff-def*)
apply (*rule-tac x=ii* **in** *exI*, *simp add*: *m power-mult diff-def*)
done
then obtain *v* **where** *v*: *cmod* (*complex-of-real* (*cmod b*) / *b* + *v*$\hat{}$*n*) < *1* **by** *blast*

```
    let ?w = v / complex-of-real (root n (cmod b))
    from odd-real-root-pow[OF o, of cmod b]
    have th1: ?w ^ n = v^n / complex-of-real (cmod b)
      by (simp add: power-divide complex-of-real-power)
     have th2:cmod (complex-of-real (cmod b) / b) = 1 using b by (simp add:
cmod-divide)
    hence th3: cmod (complex-of-real (cmod b) / b) ≥ 0 by simp
    have th4: cmod (complex-of-real (cmod b) / b) *
   cmod (1 + b * (v ^ n / complex-of-real (cmod b)))
   < cmod (complex-of-real (cmod b) / b) * 1
      apply (simp only: cmod-mult[symmetric] right-distrib)
      using b v by (simp add: th2)

    from mult-less-imp-less-left[OF th4 th3]
    have ?P ?w n unfolding th1 .
    hence ∃ z. ?P z n .. }
  ultimately show ∃ z. ?P z n by blast
qed
```

Bolzano-Weierstrass type property for closed disc in complex plane.

```
lemma metric-bound-lemma: cmod (x − y) <= |Re x − Re y| + |Im x − Im y|
  using real-sqrt-sum-squares-triangle-ineq[of Re x − Re y 0 0 Im x − Im y ]
  unfolding cmod-def by simp

lemma bolzano-weierstrass-complex-disc:
  assumes r: ∀ n. cmod (s n) ≤ r
  shows ∃ f z. subseq f ∧ (∀ e >0. ∃ N. ∀ n ≥ N. cmod (s (f n) − z) < e)
proof−
  from seq-monosub[of Re o s]
  obtain f g where f: subseq f monoseq (λn. Re (s (f n)))
    unfolding o-def by blast
  from seq-monosub[of Im o s o f]
  obtain g where g: subseq g monoseq (λn. Im (s(f(g n)))) unfolding o-def by
blast
  let ?h = f o g
  from r[rule-format, of 0] have rp: r ≥ 0 using cmod-pos[of s 0] by arith
  have th:∀ n. r + 1 ≥ | Re (s n)|
  proof
    fix n
   from abs-Re-le-cmod[of s n] r[rule-format, of n]  show |Re (s n)| ≤ r + 1 by
arith
  qed
  have conv1: convergent (λn. Re (s ( f n)))
    apply (rule Bseq-monoseq-convergent)
    apply (simp add: Bseq-def)
    apply (rule exI[where x= r + 1])
    using th rp apply simp
    using f(2) .
  have th:∀ n. r + 1 ≥ | Im (s n)|
```

proof
fix *n*
from *abs-Im-le-cmod*[*of s n*] *r*[*rule-format*, *of n*] **show** $|Im\ (s\ n)| \leq r + 1$ **by** *arith*
qed

have *conv2*: *convergent* ($\lambda n.\ Im\ (s\ (f\ (g\ n))))$
apply (*rule Bseq-monoseq-convergent*)
apply (*simp add*: *Bseq-def*)
apply (*rule exI*[**where** $x = r + 1$])
using *th rp* **apply** *simp*
using $g(2)$ **.**

from *conv1*[*unfolded convergent-def*] **obtain** x **where** *LIMSEQ* ($\lambda n.\ Re\ (s\ (f\ n)))\ x$
by *blast*
hence x: $\forall r{>}0.\ \exists n0.\ \forall n{\geq}n0.\ |\ Re\ (s\ (f\ n)) - x\ | < r$
unfolding *LIMSEQ-def real-norm-def* **.**

from *conv2*[*unfolded convergent-def*] **obtain** y **where** *LIMSEQ* ($\lambda n.\ Im\ (s\ (f\ (g\ n))))\ y$
by *blast*
hence y: $\forall r{>}0.\ \exists n0.\ \forall n{\geq}n0.\ |\ Im\ (s\ (f\ (g\ n)))\ - y\ | < r$
unfolding *LIMSEQ-def real-norm-def* **.**
let *?w* = *Complex x y*
from $f(1)$ $g(1)$ **have** *hs*: *subseq ?h* **unfolding** *subseq-def* **by** *auto*
{fix *e* **assume** *ep*: $e > (0{::}real)$
hence *e2*: $e/2 > 0$ **by** *simp*
from *x*[*rule-format*, *OF e2*] *y*[*rule-format*, *OF e2*]
obtain *N1 N2* **where** *N1*: $\forall n{\geq}N1.\ |Re\ (s\ (f\ n)) - x| < e\ /\ 2$ **and** *N2*: $\forall n{\geq}N2.\ |Im\ (s\ (f\ (g\ n))) - y| < e\ /\ 2$ **by** *blast*
{fix *n* **assume** *nN12*: $n \geq N1 + N2$
hence *nN1*: $g\ n \geq N1$ **and** *nN2*: $n \geq N2$ **using** *seq-suble*[*OF* $g(1)$, *of n*] **by** *arith+*
from *add-strict-mono*[*OF N1*[*rule-format*, *OF nN1*] *N2*[*rule-format*, *OF nN2*]]
have *cmod* $(s\ (?h\ n) - ?w) < e$
using *metric-bound-lemma*[*of s* $(f\ (g\ n))$ *?w*] **by** *simp* **}**
hence $\exists N.\ \forall n{\geq}N.$ *cmod* $(s\ (?h\ n) - ?w) < e$ **by** *blast* **}**
with *hs* **show** *?thesis* **by** *blast*
qed

Polynomial is continuous.

lemma *poly-cont*:
assumes *ep*: $e > 0$
shows $\exists d > 0.\ \forall w.\ 0 < cmod\ (w - z) \wedge cmod\ (w - z) < d \longrightarrow cmod\ (poly\ p\ w - poly\ p\ z) < e$
proof−
from *poly-offset*[*of p z*] **obtain** q **where** q: *length* q = *length* p $\bigwedge x.$ *poly* $q\ x$ =

```
poly p (z + x) by blast
  {fix w
    note q(2)[of w − z, simplified]}
  note th = this
  show ?thesis unfolding th[symmetric]
  proof(induct q)
    case Nil thus ?case  using ep by auto
  next
    case (Cons c cs)
    from poly-bound-exists[of 1 cs]
    obtain m where m: m > 0 ⋀z. cmod z ≤ 1 ⟹ cmod (poly cs z) ≤ m by
blast
    from ep m(1) have em0: e/m > 0 by (simp add: field-simps)
    have one0: 1 > (0::real)  by arith
    from real-lbound-gt-zero[OF one0 em0]
    obtain d where d: d >0 d < 1 d < e / m by blast
    from d(1,3) m(1) have dm: d∗m > 0 d∗m < e
      by (simp-all add: field-simps real-mult-order)
    show ?case
      proof(rule ex-forward[OF real-lbound-gt-zero[OF one0 em0]], clarsimp simp
add: cmod-mult)
        fix d w
        assume H: d > 0 d < 1 d < e/m w≠z cmod (w−z) < d
        hence d1: cmod (w−z) ≤ 1 d ≥ 0 by simp-all
        from H(3) m(1) have dme: d∗m < e by (simp add: field-simps)
        from H have th: cmod (w−z) ≤ d by simp
        from mult-mono[OF th m(2)[OF d1(1)] d1(2) cmod-pos] dme
        show cmod (w − z) ∗ cmod (poly cs (w − z)) < e by simp
      qed
    qed
qed
```

Hence a polynomial attains minimum on a closed disc in the complex plane.

```
lemma poly-minimum-modulus-disc:
  ∃z. ∀w. cmod w ≤ r ⟶ cmod (poly p z) ≤ cmod (poly p w)
proof−
  {assume ¬ r ≥ 0 hence ?thesis unfolding linorder-not-le
     apply −
     apply (rule exI[where x=0])
     apply auto
     apply (subgoal-tac cmod w < 0)
     apply simp
     apply arith
     done }
  moreover
  {assume rp: r ≥ 0
    from rp have cmod 0 ≤ r ∧ cmod (poly p 0) = − (− cmod (poly p 0)) by
simp
    hence mth1: ∃x z. cmod z ≤ r ∧ cmod (poly p z) = − x  by blast
```

{fix *x z*
assume *H*: *cmod z* ≤ *r cmod* (*poly p z*) = − *x* ¬*x* < *1*
hence − *x* < *0* **by** *arith*
with *H*(*2*) *cmod-pos*[*of poly p z*] **have** *False* **by** *simp* **}**
then have *mth2*: ∃ *z*. ∀ *x*. (∃ *z*. *cmod z* ≤ *r* ∧ *cmod* (*poly p z*) = − *x*) ⟶ *x* < *z* **by** *blast*
from *real-sup-exists*[*OF mth1 mth2*] **obtain** *s* **where**
s: ∀ *y*. (∃ *x*. (∃ *z*. *cmod z* ≤ *r* ∧ *cmod* (*poly p z*) = − *x*) ∧ *y* < *x*) ⟷(*y* < *s*) **by** *blast*
let *?m* = −*s*
{fix *y*
from *s*[*rule-format*, *of* −*y*] **have**
(∃ *z x*. *cmod z* ≤ *r* ∧ −(− *cmod* (*poly p z*)) < *y*) ⟷ *?m* < *y*
unfolding *minus-less-iff*[*of y*] *equation-minus-iff* **by** *blast* **}**
note *s1* = *this*[*unfolded minus-minus*]
from *s1*[*of ?m*] **have** *s1m*: ⋀*z x*. *cmod z* ≤ *r* ⟹ *cmod* (*poly p z*) ≥ *?m*
by *auto*
{fix *n*::*nat*
from *s1*[*rule-format*, *of ?m* + *1*/*real* (*Suc n*)]
have ∃ *z*. *cmod z* ≤ *r* ∧ *cmod* (*poly p z*) < − *s* + *1* / *real* (*Suc n*)
by *simp***}**
hence *th*: ∀ *n*. ∃ *z*. *cmod z* ≤ *r* ∧ *cmod* (*poly p z*) < − *s* + *1* / *real* (*Suc n*) **..**
from *choice*[*OF th*] **obtain** *g* **where**
g: ∀ *n*. *cmod* (*g n*) ≤ *r* ∀ *n*. *cmod* (*poly p* (*g n*)) <*?m*+*1* /*real*(*Suc n*)
by *blast*
from *bolzano-weierstrass-complex-disc*[*OF g*(*1*)]
obtain *f z* **where** *fz*: *subseq f* ∀ *e*>*0*. ∃ *N*. ∀ *n*≥*N*. *cmod* (*g* (*f n*) − *z*) < *e*
by *blast*
{fix *w*
assume *wr*: *cmod w* ≤ *r*
let *?e* = |*cmod* (*poly p z*) − *?m*|
{assume *e*: *?e* > *0*
hence *e2*: *?e*/*2* > *0* **by** *simp*
from *poly-cont*[*OF e2*, *of z p*] **obtain** *d* **where**
d: *d*>*0* ∀ *w*. *0*<*cmod* (*w* − *z*)∧ *cmod*(*w* − *z*) < *d* ⟶ *cmod*(*poly p w* − *poly p z*) < *?e*/*2* **by** *blast*
{fix *w* **assume** *w*: *cmod* (*w* − *z*) < *d*
have *cmod*(*poly p w* − *poly p z*) < *?e* / *2*
using *d*(*2*)[*rule-format*, *of w*] *w e* **by** (*cases w*=*z*, *simp-all*)**}**
note *th1* = *this*

from *fz*(*2*)[*rule-format*, *OF d*(*1*)] **obtain** *N1* **where**
N1: ∀ *n*≥*N1*. *cmod* (*g* (*f n*) − *z*) < *d* **by** *blast*
from *reals-Archimedean2*[*of 2*/*?e*] **obtain** *N2*::*nat* **where**
N2: *2*/*?e* < *real N2* **by** *blast*
have *th2*: *cmod*(*poly p* (*g*(*f*(*N1* + *N2*))) − *poly p z*) < *?e*/*2*
using *N1*[*rule-format*, *of N1* + *N2*] *th1* **by** *simp*
{fix *a b e2 m* :: *real*
have *a* < *e2* ⟹ *abs*(*b* − *m*) < *e2* ⟹ *2* ∗ *e2* <= *abs*(*b* − *m*) + *a*

```
        ==> False by arith}
      note th0 = this
      have ath:
        ⋀m x e. m <= x ⟹ x < m + e ==> abs(x − m::real) < e by arith
      from s1m[OF g(1)[rule-format]]
      have th31: ?m ≤ cmod(poly p (g (f (N1 + N2)))) .
      from seq-suble[OF fz(1), of N1+N2]
      have th00: real (Suc (N1+N2)) ≤ real (Suc (f (N1+N2))) by simp
      have th000: 0 ≤ (1::real) (1::real) ≤ 1 real (Suc (N1+N2)) > 0
        using N2 by auto
      from frac-le[OF th000 th00] have th00: ?m +1 / real (Suc (f (N1 + N2)))
≤ ?m + 1 / real (Suc (N1 + N2)) by simp
      from g(2)[rule-format, of f (N1 + N2)]
      have th01:cmod (poly p (g (f (N1 + N2)))) < − s + 1 / real (Suc (f (N1
+ N2))) .
      from order-less-le-trans[OF th01 th00]
       have th32: cmod(poly p (g (f (N1 + N2)))) < ?m + (1/ real(Suc (N1 +
N2))) .
      from N2 have 2/?e < real (Suc (N1 + N2)) by arith
      with e2 less-imp-inverse-less[of 2/?e real (Suc (N1 + N2))]
      have ?e/2 > 1/ real (Suc (N1 + N2)) by (simp add: inverse-eq-divide)
      with ath[OF th31 th32]
      have thc1:|cmod(poly p (g (f (N1 + N2)))) − ?m|< ?e/2 by arith
      have ath2: ⋀(a::real) b c m. |a − b| <= c ==> |b − m| <= |a − m| + c
        by arith
      have th22: |cmod (poly p (g (f (N1 + N2)))) − cmod (poly p z)|
≤ cmod (poly p (g (f (N1 + N2)))) − poly p z)
        by (simp add: cmod-abs-norm)
      from ath2[OF th22, of ?m]
       have thc2: 2*(?e/2) ≤ |cmod(poly p (g (f (N1 + N2)))) − ?m| + cmod
(poly p (g (f (N1 + N2)))) − poly p z) by simp
      from th0[OF th2 thc1 thc2] have False .}
      hence ?e = 0 by auto
      then have cmod (poly p z) = ?m by simp
      with s1m[OF wr]
      have cmod (poly p z) ≤ cmod (poly p w) by simp }
    hence ?thesis by blast}
  ultimately show ?thesis by blast
qed

lemma (rcis (sqrt (abs r)) (a/2)) ^ 2 = rcis (abs r) a
  unfolding power2-eq-square
  apply (simp add: rcis-mult)
  apply (simp add: power2-eq-square[symmetric])
  done

lemma cispi: cis pi = −1
  unfolding cis-def
  by simp
```

lemma (*rcis* (*sqrt* (*abs r*)) ((*pi* + *a*)/*2*)) ^ *2* = *rcis* (− *abs r*) *a*
unfolding *power2-eq-square*
apply (*simp add*: *rcis-mult add-divide-distrib*)
apply (*simp add*: *power2-eq-square*[*symmetric*] *rcis-def cispi cis-mult*[*symmetric*])
done

Nonzero polynomial in z goes to infinity as z does.

instance *complex*::*idom-char-0* **by** (*intro-classes*)
instance *complex* :: *recpower-idom-char-0* **by** *intro-classes*

lemma *poly-infinity*:
assumes *ex*: *list-ex* (λ*c*. *c* ≠ *0*) *p*
shows ∃ *r*. ∀ *z*. *r* ≤ *cmod z* ⟶ *d* ≤ *cmod* (*poly* (*a*#*p*) *z*)
using *ex*
proof(*induct p arbitrary*: *a d*)
case (*Cons c cs a d*)
{assume *H*: *list-ex* (λ*c*. *c*≠*0*) *cs*
with *Cons.hyps* **obtain** *r* **where** *r*: ∀ *z*. *r* ≤ *cmod z* ⟶ *d* + *cmod a* ≤ *cmod* (*poly* (*c* # *cs*) *z*) **by** *blast*
let *?r* = *1* + |*r*|
{fix *z* **assume** *h*: *1* + |*r*| ≤ *cmod z*
have *r0*: *r* ≤ *cmod z* **using** *h* **by** *arith*
from *r*[*rule-format*, *OF r0*]
have *th0*: *d* + *cmod a* ≤ *1* ∗ *cmod*(*poly* (*c*#*cs*) *z*) **by** *arith*
from *h* **have** *z1*: *cmod z* ≥ *1* **by** *arith*
from *order-trans*[*OF th0 mult-right-mono*[*OF z1 cmod-pos*[*of poly* (*c*#*cs*) *z*]]]
have *th1*: *d* ≤ *cmod*(*z* ∗ *poly* (*c*#*cs*) *z*) − *cmod a*
unfolding *cmod-mult* **by** (*simp add*: *ring-simps*)
from *complex-mod-triangle-sub*[*of z* ∗ *poly* (*c*#*cs*) *z a*]
have *th2*: *cmod*(*z* ∗ *poly* (*c*#*cs*) *z*) − *cmod a* ≤ *cmod* (*poly* (*a*#*c*#*cs*) *z*)
by (*simp add*: *diff-le-eq ring-simps*)
from *th1 th2* **have** *d* ≤ *cmod* (*poly* (*a*#*c*#*cs*) *z*) **by** *arith***}**
hence *?case* **by** *blast***}**
moreover
{assume *cs0*: ¬ (*list-ex* (λ*c*. *c* ≠ *0*) *cs*)
with *Cons.prems* **have** *c0*: *c* ≠ *0* **by** *simp*
from *cs0* **have** *cs0'*: *list-all* (λ*c*. *c* = *0*) *cs*
by (*auto simp add*: *list-all-iff list-ex-iff*)
{fix *z*
assume *h*: (|*d*| + *cmod a*) / *cmod c* ≤ *cmod z*
from *c0* **have** *cmod c* > *0* **by** *simp*
from *h c0* **have** *th0*: |*d*| + *cmod a* ≤ *cmod* (*z*∗*c*)
by (*simp add*: *field-simps cmod-mult*)
have *ath*: ⋀*mzh mazh ma*. *mzh* <= *mazh* + *ma* ==> *abs*(*d*) + *ma* <= *mzh* ==> *d* <= *mazh* **by** *arith*
from *complex-mod-triangle-sub*[*of z*∗*c a*]
have *th1*: *cmod* (*z* ∗ *c*) ≤ *cmod* (*a* + *z* ∗ *c*) + *cmod a*
by (*simp add*: *ring-simps*)

from *ath*[*OF th1 th0*] **have** $d \leq$ *cmod* (*poly* (*a* # *c* # *cs*) *z*)
using *poly-0*[*OF cs0*'] **by** *simp*}
then have *?case* **by** *blast*}
ultimately show *?case* **by** *blast*
qed *simp*

Hence polynomial's modulus attains its minimum somewhere.

lemma *poly-minimum-modulus*:
$\exists z. \forall w.$ *cmod* (*poly p z*) $\leq$ *cmod* (*poly p w*)
proof(*induct p*)
case (*Cons c cs*)
{**assume** *cs0*: *list-ex* ($\lambda c.\ c \neq 0$) *cs*
from *poly-infinity*[*OF cs0*, *of cmod* (*poly* (*c*#*cs*) *0*) *c*]
obtain *r* **where** *r*: $\bigwedge z.\ r \leq$ *cmod z* $\Longrightarrow$ *cmod* (*poly* (*c* # *cs*) *0*) $\leq$ *cmod* (*poly* (*c* # *cs*) *z*) **by** *blast*
have *ath*: $\bigwedge z\ r.\ r \leq$ *cmod z* $\vee$ *cmod z* $\leq |r|$ **by** *arith*
from *poly-minimum-modulus-disc*[*of* $|r|$ *c*#*cs*]
obtain *v* **where** *v*: $\bigwedge w.$ *cmod w* $\leq |r| \Longrightarrow$ *cmod* (*poly* (*c* # *cs*) *v*) $\leq$ *cmod* (*poly* (*c* # *cs*) *w*) **by** *blast*
{**fix** *z* **assume** *z*: $r \leq$ *cmod z*
from *v*[*of 0*] *r*[*OF z*]
have *cmod* (*poly* (*c* # *cs*) *v*) $\leq$ *cmod* (*poly* (*c* # *cs*) *z*)
by *simp* }
note *v0* = *this*
from *v0 v ath*[*of r*] **have** *?case* **by** *blast*}
moreover
{**assume** *cs0*: $\neg$ (*list-ex* ($\lambda c.\ c \neq 0$) *cs*)
hence *th*:*list-all* ($\lambda c.\ c = 0$) *cs* **by** (*simp add*: *list-all-iff list-ex-iff*)
from *poly-0*[*OF th*] *Cons.hyps* **have** *?case* **by** *simp*}
ultimately show *?case* **by** *blast*
qed *simp*

Constant function (non-syntactic characterization).

definition *constant f* = ($\forall x\ y.\ f\ x = f\ y$)

lemma *nonconstant-length*: $\neg$ (*constant* (*poly p*)) $\Longrightarrow$ *length p* $\geq$ *2*
unfolding *constant-def*
apply (*induct p*, *auto*)
apply (*unfold not-less*[*symmetric*])
apply *simp*
apply (*rule ccontr*)
apply *auto*
done

lemma *poly-replicate-append*:
poly ((*replicate n 0*)@*p*) (*x*::'*a*::{*recpower*, *comm-ring*}) = *x*^*n* * *poly p x*
by(*induct n*, *auto simp add*: *power-Suc ring-simps*)

Decomposition of polynomial, skipping zero coefficients after the first.

lemma *poly-decompose-lemma*:
assumes *nz*: $\neg(\forall z.\ z{\neq}0 \longrightarrow poly\ p\ z = (0::'a::\{recpower,idom\}))$
shows $\exists k\ a\ q.\ a{\neq}0 \wedge Suc\ (length\ q + k) = length\ p \wedge$
$(\forall z.\ poly\ p\ z = z\hat{}k * poly\ (a\#q)\ z)$
using *nz*
proof(*induct p*)
case *Nil* **thus** *?case* **by** *simp*
next
case (*Cons c cs*)
{assume *c0*: *c* = *0*

from *Cons.hyps Cons.prems c0* **have** *?case* **apply** *auto*
apply (*rule-tac x=k+1* **in** *exI*)
apply (*rule-tac x=a* **in** *exI*, *clarsimp*)
apply (*rule-tac x=q* **in** *exI*)
by (*auto simp add*: *power-Suc*)**}**
moreover
{assume *c0*: *c≠0*
hence *?case* **apply**−
apply (*rule exI*[**where** *x=0*])
apply (*rule exI*[**where** *x=c*], *clarsimp*)
apply (*rule exI*[**where** *x=cs*])
apply *auto*
done}
ultimately show *?case* **by** *blast*
qed

lemma *poly-decompose*:
assumes *nc*: ~*constant*(*poly p*)
shows $\exists k\ a\ q.\ a{\neq}(0::'a::\{recpower,idom\}) \wedge k{\neq}0 \wedge$
length q + *k* + *1* = *length p* $\wedge$
$(\forall z.\ poly\ p\ z = poly\ p\ 0 + z\hat{}k * poly\ (a\#q)\ z)$
using *nc*
proof(*induct p*)
case *Nil* **thus** *?case* **by** (*simp add*: *constant-def*)
next
case (*Cons c cs*)
{assume *C*:$\forall z.\ z \neq 0 \longrightarrow poly\ cs\ z = 0$
{fix *x y*
from *C* **have** *poly* (*c*#*cs*) *x* = *poly* (*c*#*cs*) *y* **by** (*cases x=0*, *auto*)**}**
with *Cons.prems* **have** *False* **by** (*auto simp add*: *constant-def*)**}**
hence *th*: ¬ $(\forall z.\ z \neq 0 \longrightarrow poly\ cs\ z = 0)$ **..**
from *poly-decompose-lemma*[*OF th*]
show *?case*
apply *clarsimp*
apply (*rule-tac x=k+1* **in** *exI*)
apply (*rule-tac x=a* **in** *exI*)
apply *simp*
apply (*rule-tac x=q* **in** *exI*)

apply (*auto simp add*: *power-Suc*)
done
qed

Fundamental theorem of algebral

lemma *fundamental-theorem-of-algebra*:
assumes *nc*: ~*constant*(*poly p*)
shows ∃ *z*::*complex*. *poly p z* = *0*
using *nc*
proof(*induct n*≡ *length p arbitrary*: *p rule*: *nat-less-induct*)
fix *n* **fix** *p* :: *complex list*
let *?p* = *poly p*
assume *H*: ∀ *m*<*n*. ∀ *p*. ¬ *constant* (*poly p*) ⟶ *m* = *length p* ⟶ (∃ (*z*::*complex*). *poly p z* = *0*) **and** *nc*: ¬ *constant ?p* **and** *n*: *n* = *length p*
let *?ths* = ∃ *z*. *?p z* = *0*

from *nonconstant-length*[*OF nc*] **have** *n2*: *n*≥ *2* **by** (*simp add*: *n*)
from *poly-minimum-modulus* **obtain** *c* **where**
c: ∀ *w*. *cmod* (*?p c*) ≤ *cmod* (*?p w*) **by** *blast*
{assume *pc*: *?p c* = *0* **hence** *?ths* **by** *blast***}**
moreover
{assume *pc0*: *?p c* ≠ *0*
from *poly-offset*[*of p c*] **obtain** *q* **where**
q: *length q* = *length p* ∀ *x*. *poly q x* = *?p* (*c*+*x*) **by** *blast*
{assume *h*: *constant* (*poly q*)
from *q*(*2*) **have** *th*: ∀ *x*. *poly q* (*x* − *c*) = *?p x* **by** *auto*
{fix *x y*
from *th* **have** *?p x* = *poly q* (*x* − *c*) **by** *auto*
also have ... = *poly q* (*y* − *c*)
using *h* **unfolding** *constant-def* **by** *blast*
also have ... = *?p y* **using** *th* **by** *auto*
finally have *?p x* = *?p y* **.}**
with *nc* **have** *False* **unfolding** *constant-def* **by** *blast* **}**
hence *qnc*: ¬ *constant* (*poly q*) **by** *blast*
from *q*(*2*) **have** *pqc0*: *?p c* = *poly q 0* **by** *simp*
from *c pqc0* **have** *cq0*: ∀ *w*. *cmod* (*poly q 0*) ≤ *cmod* (*?p w*) **by** *simp*
let *?a0* = *poly q 0*
from *pc0 pqc0* **have** *a00*: *?a0* ≠ *0* **by** *simp*
from *a00*
have *qr*: ∀ *z*. *poly q z* = *poly* (*map* (*op* ∗ (*inverse ?a0*)) *q*) *z* ∗ *?a0*
by (*simp add*: *poly-cmult-map*)
let *?r* = *map* (*op* ∗ (*inverse ?a0*)) *q*
have *lgqr*: *length q* = *length ?r* **by** *simp*
{assume *h*: ⋀ *x y*. *poly ?r x* = *poly ?r y*
{fix *x y*
from *qr*[*rule-format*, *of x*]
have *poly q x* = *poly ?r x* ∗ *?a0* **by** *auto*
also have ... = *poly ?r y* ∗ *?a0* **using** *h* **by** *simp*
also have ... = *poly q y* **using** *qr*[*rule-format*, *of y*] **by** *simp*

```
    finally have poly q x = poly q y .}
  with qnc have False unfolding constant-def by blast}
hence rnc: ¬ constant (poly ?r) unfolding constant-def by blast
from qr[rule-format, of 0] a00 have r01: poly ?r 0 = 1 by auto
{fix w
  have cmod (poly ?r w) < 1 ⟷ cmod (poly q w / ?a0) < 1
    using qr[rule-format, of w] a00 by simp
  also have ... ⟷ cmod (poly q w) < cmod ?a0
    using a00 unfolding cmod-divide by (simp add: field-simps)
  finally have cmod (poly ?r w) < 1 ⟷ cmod (poly q w) < cmod ?a0 .}
note mrmq-eq = this
from poly-decompose[OF rnc] obtain k a s where
  kas: a≠0 k≠0 length s + k + 1 = length ?r
  ∀ z. poly ?r z = poly ?r 0 + z^k* poly (a#s) z by blast
{assume k + 1 = n
  with kas(3) lgqr[symmetric] q(1) n[symmetric] have s0:s=[] by auto
  {fix w
    have cmod (poly ?r w) = cmod (1 + a * w ^ k)
      using kas(4)[rule-format, of w] s0 r01 by (simp add: ring-simps)}
  note hth = this [symmetric]
    from reduce-poly-simple[OF kas(1,2)]
  have ∃ w. cmod (poly ?r w) < 1 unfolding hth by blast}
moreover
{assume kn: k+1 ≠ n
  from kn kas(3) q(1) n[symmetric] have k1n: k + 1 < n by simp
  have th01: ¬ constant (poly (1#((replicate (k − 1) 0)@[a])))
    unfolding constant-def poly-Nil poly-Cons poly-replicate-append
    using kas(1) apply simp
    by (rule exI[where x=0], rule exI[where x=1], simp)
  from kas(2) have th02: k+1 = length (1#((replicate (k − 1) 0)@[a]))
    by simp
  from H[rule-format, OF k1n th01 th02]
  obtain w where w: 1 + w^k * a = 0
    unfolding poly-Nil poly-Cons poly-replicate-append
    using kas(2) by (auto simp add: power-Suc[symmetric, of - k − Suc 0]
      mult-assoc[of - - a, symmetric])
  from poly-bound-exists[of cmod w s] obtain m where
    m: m > 0 ∀ z. cmod z ≤ cmod w ⟶ cmod (poly s z) ≤ m by blast
  have w0: w≠0 using kas(2) w by (auto simp add: power-0-left)
  from w have (1 + w ^ k * a) − 1 = 0 − 1 by simp
  then have wm1: w^k * a = − 1 by simp
  have inv0: 0 < inverse (cmod w ^ (k + 1) * m)
    using cmod-pos[of w] w0 m(1)
      by (simp add: inverse-eq-divide zero-less-mult-iff)
  with real-down2[OF zero-less-one] obtain t where
    t: t > 0 t < 1 t < inverse (cmod w ^ (k + 1) * m) by blast
  let ?ct = complex-of-real t
  let ?w = ?ct * w
  have 1 + ?w^k * (a + ?w * poly s ?w) = 1 + ?ct^k * (w^k * a) + ?w^k *
```

*?w * poly s ?w* **using** *kas(1)* **by** (*simp add*: *ring-simps power-mult-distrib*)
also have ... = *complex-of-real (1 − t^k) + ?w^k * ?w * poly s ?w*
unfolding *wm1* **by** (*simp*)
finally have *cmod (1 + ?w^k * (a + ?w * poly s ?w)) = cmod (complex-of-real (1 − t^k) + ?w^k * ?w * poly s ?w)*
apply −
apply (*rule cong*[*OF refl*[*of cmod*]])
apply *assumption*
done
with *complex-mod-triangle-ineq*[*of complex-of-real (1 − t^k) ?w^k * ?w * poly s ?w*]
have *th11*: *cmod (1 + ?w^k * (a + ?w * poly s ?w)) ≤ |1 − t^k| + cmod (?w^k * ?w * poly s ?w)* **unfolding** *cmod-complex-of-real* **by** *simp*
have *ath*: ⋀*x* (*t*::*real*). *0≤ x ⟹ x < t ⟹ t≤1 ⟹ |1 − t| + x < 1* **by** *arith*
have *t *cmod w ≤ 1 * cmod w* **apply** (*rule mult-mono*) **using** *t*(*1*,*2*) **by** *auto*
then have *tw*: *cmod ?w ≤ cmod w* **using** *t*(*1*) **by** (*simp add*: *cmod-mult*)
from *t inv0* **have** *t* (cmod w ^ (k + 1) * m) < 1*
by (*simp add*: *inverse-eq-divide field-simps*)
with *zero-less-power*[*OF t*(*1*), *of k*]
have *th30*: *t^k * (t* (cmod w ^ (k + 1) * m)) < t^k * 1*
apply − **apply** (*rule mult-strict-left-mono*) **by** *simp-all*
have *cmod (?w^k * ?w * poly s ?w) = t^k * (t* (cmod w ^ (k+1) * cmod (poly s ?w)))* **using** *w0 t*(*1*)
by (*simp add*: *ring-simps power-mult-distrib cmod-complex-of-real cmod-power cmod-mult*)
then have *cmod (?w^k * ?w * poly s ?w) ≤ t^k * (t* (cmod w ^ (k + 1) * m))*
using *t*(*1*,*2*) *m*(*2*)[*rule-format*, *OF tw*] *w0*
apply (*simp only*:)
apply *auto*
apply (*rule mult-mono*, *simp-all add*: *cmod-pos*)+
apply (*simp add*: *zero-le-mult-iff zero-le-power*)
done
with *th30* **have** *th120*: *cmod (?w^k * ?w * poly s ?w) < t^k* **by** *simp*
from *power-strict-mono*[*OF t*(*2*), *of k*] *t*(*1*) *kas*(*2*) **have** *th121*: *t^k ≤ 1*
by *auto*
from *ath*[*OF cmod-pos*[*of ?w^k * ?w * poly s ?w*] *th120 th121*]
have *th12*: *|1 − t^k| + cmod (?w^k * ?w * poly s ?w) < 1* .
from *th11 th12*
have *cmod (1 + ?w^k * (a + ?w * poly s ?w)) < 1* **by** *arith*
then have *cmod (poly ?r ?w) < 1*
unfolding *kas*(*4*)[*rule-format*, *of ?w*] *r01* **by** *simp*
then have ∃ *w*. *cmod (poly ?r w) < 1* **by** *blast*}
ultimately have *cr0-contr*: ∃ *w*. *cmod (poly ?r w) < 1* **by** *blast*
from *cr0-contr cq0 q*(*2*)
have *?ths* **unfolding** *mrmq-eq not-less*[*symmetric*] **by** *auto*}
ultimately show *?ths* **by** *blast*

qed

Alternative version with a syntactic notion of constant polynomial.

lemma *fundamental-theorem-of-algebra-alt*:
assumes *nc*: $\sim(\exists\, a\ l.\ a \neq 0 \wedge \mathit{list\text{-}all}(\lambda b.\ b = 0)\ l \wedge p = a \# l)$
shows $\exists\, z.\ \mathit{poly}\ p\ z = (0::\mathit{complex})$
using *nc*
proof(*induct p*)
case (*Cons c cs*)
{assume *c=0* **hence** *?case* **by** *auto*}
moreover
{assume *c0*: *c≠0*
{assume *nc*: *constant* (*poly* (*c#cs*))
from *nc*[*unfolded constant-def*, *rule-format*, *of 0*]
have $\forall\, w.\ w \neq 0 \longrightarrow \mathit{poly}\ \mathit{cs}\ w = 0$ **by** *auto*
hence *list-all* ($\lambda c.\ c=0$) *cs*
proof(*induct cs*)
case (*Cons d ds*)
{assume *d=0* **hence** *?case* **using** *Cons.prems Cons.hyps* **by** *simp*}
moreover
{assume *d0*: *d≠0*
from *poly-bound-exists*[*of 1 ds*] **obtain** *m* **where**
m: $m > 0\ \forall\, z.\ \forall\, z.\ \mathit{cmod}\ z \leq 1 \longrightarrow \mathit{cmod}\ (\mathit{poly}\ \mathit{ds}\ z) \leq m$ **by** *blast*
have *dm*: *cmod d / m > 0* **using** *d0 m(1)* **by** (*simp add*: *field-simps*)
from *real-down2*[*OF dm zero-less-one*] **obtain** *x* **where**
x: $x > 0\ x < \mathit{cmod}\ d\ /\ m\ x < 1$ **by** *blast*
let *?x = complex-of-real x*
from *x* **have** *cx*: *?x ≠ 0 cmod ?x ≤ 1* **by** *simp-all*
from *Cons.prems*[*rule-format*, *OF cx(1)*]
have *cth*: *cmod* (*?x∗poly ds ?x*) = *cmod d* **by** (*simp add*: *eq-diff-eq*[*symmetric*])
from *m(2)*[*rule-format*, *OF cx(2)*] *x(1)*
have *th0*: *cmod* (*?x∗poly ds ?x*) ≤ *x∗m*
by (*simp add*: *cmod-mult*)
from *x(2) m(1)* **have** *x∗m < cmod d* **by** (*simp add*: *field-simps*)
with *th0* **have** *cmod* (*?x∗poly ds ?x*) ≠ *cmod d* **by** *auto*
with *cth* **have** *?case* **by** *blast*}
ultimately show *?case* **by** *blast*
qed *simp*}
then have *nc*: ¬ *constant* (*poly* (*c#cs*)) **using** *Cons.prems c0*
by *blast*
from *fundamental-theorem-of-algebra*[*OF nc*] **have** *?case* **.}**
ultimately show *?case* **by** *blast*
qed *simp*

30 Nullstellenstatz, degrees and divisibility of polynomials

lemma *nullstellensatz-lemma*:

```
  fixes p :: complex list
  assumes ∀x. poly p x = 0 ⟶ poly q x = 0
  and degree p = n and n ≠ 0
  shows p divides (pexp q n)
using prems
proof(induct n arbitrary: p q rule: nat-less-induct)
  fix n::nat fix p q :: complex list
  assume IH: ∀m<n. ∀p q.
                (∀x. poly p x = (0::complex) ⟶ poly q x = 0) ⟶
                degree p = m ⟶ m ≠ 0 ⟶ p divides (q %^ m)
    and pq0: ∀x. poly p x = 0 ⟶ poly q x = 0
    and dpn: degree p = n and n0: n ≠ 0
  let ?ths = p divides (q %^ n)
  {fix a assume a: poly p a = 0
    {assume p0: poly p = poly []
      hence ?ths unfolding divides-def  using pq0 n0
        apply − apply (rule exI[where x=[]], rule ext)
        by (auto simp add: poly-mult poly-exp)}
    moreover
    {assume p0: poly p ≠ poly []
      and oa: order  a p ≠ 0
      from p0 have pne: p ≠ [] by auto
      let ?op = order a p
      from p0 have ap: ([− a, 1] %^ ?op) divides p
        ¬ pexp [− a, 1] (Suc ?op) divides p using order by blast+
      note oop = order-degree[OF p0, unfolded dpn]
      {assume q0: q = []
        hence ?ths using n0 unfolding divides-def
          apply simp
          apply (rule exI[where x=[]], rule ext)
          by (simp add: divides-def poly-exp poly-mult)}
      moreover
      {assume q0: q≠[]
        from pq0[rule-format, OF a, unfolded poly-linear-divides] q0
        obtain r where r: q = pmult [− a, 1] r by blast
        from ap[unfolded divides-def] obtain s where
          s: poly p = poly (pmult (pexp [− a, 1] ?op) s) by blast
        have s0: poly s ≠ poly []
          using s p0 by (simp add: poly-entire)
        hence pns0: poly (pnormalize s) ≠ poly [] and sne: s≠[] by auto
        {assume ds0: degree s = 0
          from ds0 pns0 have ∃k. pnormalize s = [k] unfolding degree-def
            by (cases pnormalize s, auto)
          then obtain k where kpn: pnormalize s = [k] by blast
          from pns0[unfolded poly-zero] kpn have k: k ≠0 poly s = poly [k]
            using poly-normalize[of s] by simp-all
          let ?w = pmult (pmult [1/k] (pexp [−a,1] (n − ?op))) (pexp r n)
          from k r s oop have poly (pexp q n) = poly (pmult p ?w)
                by − (rule ext, simp add: poly-mult poly-exp poly-cmult poly-add
```

power-add[*symmetric*] *ring-simps power-mult-distrib*[*symmetric*])
hence *?ths* **unfolding** *divides-def* **by** *blast*}
moreover
{**assume** *ds0*: *degree s* ≠ *0*
from *ds0 s0 dpn degree-unique*[*OF s*, *unfolded linear-pow-mul-degree*] *oa*
have *dsn*: *degree s* < *n* **by** *auto*
{**fix** *x* **assume** *h*: *poly s x* = *0*
{**assume** *xa*: *x* = *a*
from *h*[*unfolded xa poly-linear-divides*] *sne* **obtain** *u* **where**
u: *s* = *pmult* [− *a*, *1*] *u* **by** *blast*
have *poly p* = *poly* (*pmult* (*pexp* [− *a*, *1*] (*Suc ?op*)) *u*)
unfolding *s u*
apply (*rule ext*)
by (*simp add*: *ring-simps power-mult-distrib*[*symmetric*] *poly-mult poly-cmult poly-add poly-exp*)
with *ap*(*2*)[*unfolded divides-def*] **have** *False* **by** *blast*}
note *xa* = *this*
from *h s* **have** *poly p x* = *0* **by** (*simp add*: *poly-mult*)
with *pq0* **have** *poly q x* = *0* **by** *blast*
with *r xa* **have** *poly r x* = *0*
by (*auto simp add*: *poly-mult poly-add poly-cmult eq-diff-eq*[*symmetric*])}
note *impth* = *this*
from *IH*[*rule-format*, *OF dsn*, *of s r*] *impth ds0*
have *s divides* (*pexp r* (*degree s*)) **by** *blast*
then obtain *u* **where** *u*: *poly* (*pexp r* (*degree s*)) = *poly* (*pmult s u*)
unfolding *divides-def* **by** *blast*
hence *u'*: ⋀*x*. *poly s x* ∗ *poly u x* = *poly r x* ^ *degree s*
by (*simp add*: *poly-mult*[*symmetric*] *poly-exp*[*symmetric*])
let *?w* = *pmult* (*pmult u* (*pexp* [−*a*,*1*] (*n* − *?op*))) (*pexp r* (*n* − *degree s*))
from *u' s r oop*[*of a*] *dsn* **have** *poly* (*pexp q n*) = *poly* (*pmult p ?w*)
apply − **apply** (*rule ext*)
apply (*simp only*: *power-mult-distrib power-add*[*symmetric*] *poly-add poly-mult poly-exp poly-cmult ring-simps*)

apply (*simp add*: *power-mult-distrib power-add*[*symmetric*] *poly-add poly-mult poly-exp poly-cmult mult-assoc*[*symmetric*])
done
hence *?ths* **unfolding** *divides-def* **by** *blast*}
ultimately have *?ths* **by** *blast* }
ultimately have *?ths* **by** *blast*}
ultimately have *?ths* **using** *a order-root* **by** *blast*}
moreover
{**assume** *exa*: ¬ (∃ *a*. *poly p a* = *0*)
from *fundamental-theorem-of-algebra-alt*[*of p*] *exa* **obtain** *c cs* **where**
ccs: *c*≠*0 list-all* (λ*c*. *c* = *0*) *cs p* = *c*#*cs* **by** *blast*

from *poly-0*[*OF ccs*(*2*)] *ccs*(*3*)
have *pp*: ⋀*x*. *poly p x* = *c* **by** *simp*

```
    let ?w = pmult [1/c] (pexp q n)
    from pp ccs(1)
    have poly (pexp q n) = poly (pmult p ?w)
      apply − apply (rule ext)
      unfolding poly-mult-assoc[symmetric] by (simp add: poly-mult)
    hence ?ths unfolding divides-def by blast}
  ultimately show ?ths by blast
qed

lemma nullstellensatz-univariate:
  (∀ x. poly p x = (0::complex) ⟶ poly q x = 0) ⟷
    p divides (q %^ (degree p)) ∨ (poly p = poly [] ∧ poly q = poly [])
proof−
  {assume pe: poly p = poly []
   hence eq: (∀ x. poly p x = (0::complex) ⟶ poly q x = 0) ⟷ poly q = poly []
     apply auto
     by (rule ext, simp)
    {assume p divides (pexp q (degree p))
     then obtain r where r: poly (pexp q (degree p)) = poly (pmult p r)
       unfolding divides-def by blast
     from cong[OF r refl] pe degree-unique[OF pe]
     have False by (simp add: poly-mult degree-def)}
   with eq pe have ?thesis by blast}
  moreover
  {assume pe: poly p ≠ poly []
   have p0: poly [0] = poly [] by (rule ext, simp)
   {assume dp: degree p = 0
    then obtain k where pnormalize p = [k] using pe poly-normalize[of p]
      unfolding degree-def by (cases pnormalize p, auto)
    hence k: pnormalize p = [k] poly p = poly [k] k≠0
      using pe poly-normalize[of p] by (auto simp add: p0)
    hence th1: ∀ x. poly p x ≠ 0 by simp
    from k(2,3) dp have poly (pexp q (degree p)) = poly (pmult p [1/k])
      by − (rule ext, simp add: poly-mult poly-exp)
    hence th2: p divides (pexp q (degree p)) unfolding divides-def by blast
    from th1 th2 pe have ?thesis by blast}
   moreover
   {assume dp: degree p ≠ 0
    then obtain n where n: degree p = Suc n  by (cases degree p, auto)
    {assume p divides (pexp q (Suc n))
     then obtain u where u: poly (pexp q (Suc n)) = poly (pmult p u)
       unfolding divides-def by blast
     hence u' :⋀x. poly (pexp q (Suc n)) x = poly (pmult p u) x by simp-all
     {fix x assume h: poly p x = 0 poly q x ≠ 0
      hence poly (pexp q (Suc n)) x ≠ 0 by (simp only: poly-exp) simp
      hence False using u' h(1) by (simp only: poly-mult poly-exp) simp}}
     with n nullstellensatz-lemma[of p q degree p] dp
     have ?thesis by auto}
   ultimately have ?thesis by blast}
```

ultimately show *?thesis* **by** *blast*
qed

Useful lemma

lemma (**in** *idom-char-0*) *constant-degree*: *constant* (*poly p*) ⟷ *degree p* = *0* (**is** *?lhs* = *?rhs*)
proof
assume *l*: *?lhs*
from *l*[*unfolded constant-def*, *rule-format*, *of - zero*]
have *th*: *poly p* = *poly* [*poly p 0*] **apply** − **by** (*rule ext*, *simp*)
from *degree-unique*[*OF th*] **show** *?rhs* **by** (*simp add*: *degree-def*)
next
assume *r*: *?rhs*
from *r* **have** *pnormalize p* = [] ∨ (∃ *k*. *pnormalize p* = [*k*])
unfolding *degree-def* **by** (*cases pnormalize p*, *auto*)
then show *?lhs* **unfolding** *constant-def poly-normalize*[*of p*, *symmetric*]
by (*auto simp del*: *poly-normalize*)
qed

lemma *divides-degree-lemma*: **assumes** *dpn*: *degree* (*p*::*complex list*) = *n*
shows *n* ≤ *degree* (*p* *** *q*) ∨ *poly* (*p* *** *q*) = *poly* []
using *dpn*
proof(*induct n arbitrary*: *p q*)
case *0* **thus** *?case* **by** *simp*
next
case (*Suc n p q*)
from *Suc.prems fundamental-theorem-of-algebra*[*of p*] *constant-degree*[*of p*]
obtain *a* **where** *a*: *poly p a* = *0* **by** *auto*
then obtain *r* **where** *r*: *p* = *pmult* [−*a*, *1*] *r* **unfolding** *poly-linear-divides*
using *Suc.prems* **by** (*auto simp add*: *degree-def*)
{assume *h*: *poly* (*pmult r q*) = *poly* []
hence *poly* (*pmult p q*) = *poly* [] **using** *r*
apply − **apply** (*rule ext*) **by** (*auto simp add*: *poly-entire poly-mult poly-add poly-cmult*) **hence** *?case* **by** *blast***}**
moreover
{assume *h*: *poly* (*pmult r q*) ≠ *poly* []
hence *r0*: *poly r* ≠ *poly* [] **and** *q0*: *poly q* ≠ *poly* []
by (*auto simp add*: *poly-entire*)
have *eq*: *poly* (*pmult p q*) = *poly* (*pmult* [−*a*, *1*] (*pmult r q*))
apply − **apply** (*rule ext*)
by (*simp add*: *r poly-mult poly-add poly-cmult ring-simps*)
from *linear-mul-degree*[*OF h*, *of* − *a*]
have *dqe*: *degree* (*pmult p q*) = *degree* (*pmult r q*) + *1*
unfolding *degree-unique*[*OF eq*] **.**
from *linear-mul-degree*[*OF r0*, *of* − *a*, *unfolded r*[*symmetric*]] *r Suc.prems*
have *dr*: *degree r* = *n* **by** *auto*
from *Suc.hyps*[*OF dr*, *of q*] **have** *Suc n* ≤ *degree* (*pmult p q*)

unfolding *dqe* **using** *h* **by** (*auto simp del*: *poly.simps*)
hence *?case* **by** *blast*}
ultimately show *?case* **by** *blast*
qed

lemma *divides-degree*: **assumes** *pq*: *p divides* (*q*:: *complex list*)
shows *degree p* ≤ *degree q* ∨ *poly q* = *poly* []
using *pq divides-degree-lemma*[*OF refl*, *of p*]
apply (*auto simp add*: *divides-def poly-entire*)
apply *atomize*
apply (*erule-tac x*=*qa* **in** *allE*, *auto*)
apply (*subgoal-tac degree q* = *degree* (*p* *** *qa*), *simp*)
apply (*rule degree-unique*, *simp*)
done

lemma *mpoly-base-conv*:
(*0*::*complex*) ≡ *poly* [] *x c* ≡ *poly* [*c*] *x x* ≡ *poly* [*0*,*1*] *x* **by** *simp-all*

lemma *mpoly-norm-conv*:
poly [*0*] (*x*::*complex*) ≡ *poly* [] *x poly* [*poly* [] *y*] *x* ≡ *poly* [] *x* **by** *simp-all*

lemma *mpoly-sub-conv*:
poly p (*x*::*complex*) − *poly q x* ≡ *poly p x* + −*1* ∗ *poly q x*
by (*simp add*: *diff-def*)

lemma *poly-pad-rule*: *poly p x* = *0* ==> *poly* (*0*#*p*) *x* = (*0*::*complex*) **by** *simp*

lemma *poly-cancel-eq-conv*: *p* = (*0*::*complex*) ⟹ *a* ≠ *0* ⟹ (*q* = *0*) ≡ (*a* ∗ *q* − *b* ∗ *p* = *0*) **apply** (*atomize* (*full*)) **by** *auto*

lemma *resolve-eq-raw*: *poly* [] *x* ≡ *0 poly* [*c*] *x* ≡ (*c*::*complex*) **by** *auto*
lemma *resolve-eq-then*: (*P* ⟹ (*Q* ≡ *Q1*)) ⟹ (¬*P* ⟹ (*Q* ≡ *Q2*))
⟹ *Q* ≡ *P* ∧ *Q1* ∨ ¬*P*∧ *Q2* **apply** (*atomize* (*full*)) **by** *blast*
lemma *expand-ex-beta-conv*: *list-ex P* [*c*] ≡ *P c* **by** *simp*

lemma *poly-divides-pad-rule*:
fixes *p q* :: *complex list*
assumes *pq*: *p divides q*
shows *p divides* ((*0*::*complex*)#*q*)
proof−
from *pq* **obtain** *r* **where** *r*: *poly q* = *poly* (*p* *** *r*) **unfolding** *divides-def* **by** *blast*
hence *poly* (*0*#*q*) = *poly* (*p* *** ([*0*,*1*] *** *r*))
by − (*rule ext*, *simp add*: *poly-mult poly-cmult poly-add*)
thus *?thesis* **unfolding** *divides-def* **by** *blast*
qed

lemma *poly-divides-pad-const-rule*:
fixes *p q* :: *complex list*
assumes *pq*: *p divides q*
shows *p divides* (*a %* q*)
proof−
from *pq* **obtain** *r* **where** *r*: *poly q* = *poly* (*p *** r*) **unfolding** *divides-def* **by** *blast*
hence *poly* (*a %* q*) = *poly* (*p **** (*a %* r*))
by − (*rule ext*, *simp add*: *poly-mult poly-cmult poly-add*)
thus *?thesis* **unfolding** *divides-def* **by** *blast*
qed

lemma *poly-divides-conv0*:
fixes *p* :: *complex list*
assumes *lgpq*: *length q* < *length p* **and** *lq*:*last p* ≠ *0*
shows *p divides q* ≡ (¬ (*list-ex* (λ*c*. *c* ≠ *0*) *q*)) (**is** *?lhs* ≡ *?rhs*)
proof−
{assume *r*: *?rhs*
hence *eq*: *poly q* = *poly* [] **unfolding** *poly-zero*
by (*simp add*: *list-all-iff list-ex-iff*)
hence *poly q* = *poly* (*p *** []*) **by** − (*rule ext*, *simp add*: *poly-mult*)
hence *?lhs* **unfolding** *divides-def* **by** *blast***}**
moreover
{assume *l*: *?lhs*
have *ath*: ⋀*lq lp dq*::*nat*. *lq* < *lp* ==> *lq* ≠ *0* ⟹ *dq* <= *lq* − *1* ==> *dq* < *lp* − *1*
by *arith*
{assume *q0*: *length q* = *0*
hence *q* = [] **by** *simp*
hence *?rhs* **by** *simp***}**
moreover
{assume *lgq0*: *length q* ≠ *0*
from *pnormalize-length*[*of q*] **have** *dql*: *degree q* ≤ *length q* − *1*
unfolding *degree-def* **by** *simp*
from *ath*[*OF lgpq lgq0 dql*, *unfolded pnormal-degree*[*OF lq*, *symmetric*]] *divides-degree*[*OF l*] **have** *poly q* = *poly* [] **by** *auto*
hence *?rhs* **unfolding** *poly-zero* **by** (*simp add*: *list-all-iff list-ex-iff*)**}**
ultimately have *?rhs* **by** *blast* **}**
ultimately show *?lhs* ≡ *?rhs* **by** − (*atomize* (*full*), *blast*)
qed

lemma *poly-divides-conv1*:
assumes *a0*: *a*≠ (*0*::*complex*) **and** *pp'*: (*p*::*complex list*) *divides p'*
and *qrp'*: ⋀*x*. *a* * *poly q x* − *poly p' x* ≡ *poly r x*
shows *p divides q* ≡ *p divides* (*r*::*complex list*) (**is** *?lhs* ≡ *?rhs*)
proof−
{
from *pp'* **obtain** *t* **where** *t*: *poly p'* = *poly* (*p *** t*)

```
    unfolding divides-def by blast
  {assume l: ?lhs
    then obtain u where u: poly q = poly (p *** u) unfolding divides-def by
blast
     have poly r = poly (p *** ((a %* u) +++ (-- t)))
      using u qrp' t
      by - (rule ext,
        simp add: poly-add poly-mult poly-cmult poly-minus ring-simps)
     then have ?rhs unfolding divides-def by blast}
  moreover
  {assume r: ?rhs
    then obtain u where u: poly r = poly (p *** u) unfolding divides-def by
blast
    from u t qrp' a0 have poly q = poly (p *** ((1/a) %* (u +++ t)))
       by - (rule ext, atomize (full), simp add: poly-mult poly-add poly-cmult
field-simps)
    hence ?lhs unfolding divides-def by blast}
  ultimately have ?lhs = ?rhs by blast }
thus ?lhs ≡ ?rhs by - (atomize(full), blast)
qed

lemma basic-cqe-conv1:
  (∃ x. poly p x = 0 ∧ poly [] x ≠ 0) ≡ False
  (∃ x. poly [] x ≠ 0) ≡ False
  (∃ x. poly [c] x ≠ 0) ≡ c≠0
  (∃ x. poly [] x = 0) ≡ True
  (∃ x. poly [c] x = 0) ≡ c = 0 by simp-all

lemma basic-cqe-conv2:
  assumes l:last (a#b#p) ≠ 0
  shows (∃ x. poly (a#b#p) x = (0::complex)) ≡ True
proof-
  {fix h t
    assume h: h≠0 list-all (λc. c=(0::complex)) t  a#b#p = h#t
    hence list-all (λc. c= 0) (b#p) by simp
    moreover have last (b#p) ∈ set (b#p) by simp
    ultimately have last (b#p) = 0 by (simp add: list-all-iff)
    with l have False by simp}
  hence th: ¬ (∃ h t. h≠0 ∧ list-all (λc. c=0) t ∧ a#b#p = h#t)
    by blast
  from fundamental-theorem-of-algebra-alt[OF th]
  show (∃ x. poly (a#b#p) x = (0::complex)) ≡ True by auto
qed

lemma  basic-cqe-conv-2b: (∃ x. poly p x ≠ (0::complex)) ≡ (list-ex (λc. c ≠ 0)
p)
proof-
  have ¬ (list-ex (λc. c ≠ 0) p) ⟷ poly p = poly []
    by (simp add: poly-zero list-all-iff list-ex-iff)
```

also have ... $\longleftrightarrow$ $(\neg\ (\exists\, x.\ poly\ p\ x \neq 0))$ **by** (*auto intro*: *ext*)
finally show $(\exists\, x.\ poly\ p\ x \neq (0::complex)) \equiv (list\text{-}ex\ (\lambda c.\ c \neq 0)\ p)$
by − (*atomize* (*full*), *blast*)
qed

lemma *basic-cqe-conv3*:
fixes *p q* :: *complex list*
assumes *l*: *last* $(a\#p) \neq 0$
shows $(\exists\, x.\ poly\ (a\#p)\ x = 0 \wedge poly\ q\ x \neq 0) \equiv \neg\ ((a\#p)\ divides\ (q\ \%\hat{}\ (length\ p)))$
proof−
note *np* = *pnormalize-eq*[*OF l*]
{assume *poly* $(a\#p)$ = *poly* [] **hence** *False* **using** *l*
unfolding *poly-zero* **apply** (*auto simp add*: *list-all-iff del*: *last.simps*)
apply (*cases p*, *simp-all*) **done}**
then have *p0*: *poly* $(a\#p) \neq$ *poly* [] **by** *blast*
from *np* **have** *dp*:*degree* $(a\#p)$ = *length p* **by** (*simp add*: *degree-def*)
from *nullstellensatz-univariate*[*of a*#*p q*] *p0 dp*
show $(\exists\, x.\ poly\ (a\#p)\ x = 0 \wedge poly\ q\ x \neq 0) \equiv \neg\ ((a\#p)\ divides\ (q\ \%\hat{}\ (length\ p)))$
by − (*atomize* (*full*), *auto*)
qed

lemma *basic-cqe-conv4*:
fixes *p q* :: *complex list*
assumes *h*: $\bigwedge x.\ poly\ (q\ \%\hat{}\ n)\ x \equiv poly\ r\ x$
shows *p divides* $(q\ \%\hat{}\ n) \equiv p\ divides\ r$
proof−
from *h* **have** *poly* $(q\ \%\hat{}\ n)$ = *poly r* **by** (*auto intro*: *ext*)
thus *p divides* $(q\ \%\hat{}\ n) \equiv p\ divides\ r$ **unfolding** *divides-def* **by** *simp*
qed

lemma *pmult-Cons-Cons*: $((a::complex)\#b\#p)\ ***\ q = (a\ \%*q)\ +++\ (0\#((b\#p)\ ***\ q))$
by *simp*

lemma *elim-neg-conv*: $-\ z \equiv (-1) * (z::complex)$ **by** *simp*
lemma *eqT-intr*: *PROP P* $\Longrightarrow$ (*True* $\Longrightarrow$ *PROP P*) *PROP P* $\Longrightarrow$ *True* **by** *blast*+
lemma *negate-negate-rule*: *Trueprop P* $\equiv \neg\ P \equiv$ *False* **by** (*atomize* (*full*), *auto*)
lemma *last-simps*: *last* $[x] = x$ *last* $(x\#y\#ys) =$ *last* $(y\#ys)$ **by** *simp-all*
lemma *length-simps*: *length* [] = *0 length* $(x\#y\#xs) =$ *length xs* + *2 length* $[x] =$ *1* **by** *simp-all*

lemma *complex-entire*: $(z::complex) \neq 0 \wedge w \neq 0 \equiv z*w \neq 0$ **by** *simp*
lemma *resolve-eq-ne*: $(P \equiv True) \equiv (\neg P \equiv False)$ $(P \equiv False) \equiv (\neg P \equiv True)$
by (*atomize* (*full*)) *simp-all*
lemma *cqe-conv1*: *poly* [] $x = 0 \longleftrightarrow$ *True* **by** *simp*
lemma *cqe-conv2*: $(p \Longrightarrow (q \equiv r)) \equiv ((p \wedge q) \equiv (p \wedge r))$ (**is** *?l* ≡ *?r*)

proof
assume $p \Longrightarrow q \equiv r$ **thus** $p \wedge q \equiv p \wedge r$ **apply** $-$ **apply** (*atomize* (*full*)) **by** *blast*
next
assume $p \wedge q \equiv p \wedge r$ p
thus $q \equiv r$ **apply** $-$ **apply** (*atomize* (*full*)) **apply** *blast* **done**
qed
lemma *poly-const-conv*: *poly* [c] (x::*complex*) $= y \longleftrightarrow c = y$ **by** *simp*

end

31 Order-Relation: Orders as Relations

theory *Order-Relation*
imports *ATP-Linkup Hilbert-Choice*
begin

This prelude could be moved to theory Relation:

definition *irrefl* $r \equiv \forall x.\ (x,x) \notin r$

definition *total-on* A $r \equiv \forall x{\in}A. \forall y{\in}A.\ x{\neq}y \longrightarrow (x,y){\in}r \vee (y,x){\in}r$

abbreviation *total* $\equiv$ *total-on UNIV*

lemma *total-on-empty*[*simp*]: *total-on* {} r
by(*simp add*:*total-on-def*)

lemma *refl-on-converse*[*simp*]: *refl* A ($r\hat{}{-}1$) $=$ *refl* A r
by(*auto simp add*:*refl-def*)

lemma *total-on-converse*[*simp*]: *total-on* A ($r\hat{}{-}1$) $=$ *total-on* A r
by (*auto simp*: *total-on-def*)

lemma *irrefl-diff-Id*[*simp*]: *irrefl*($r{-}Id$)
by(*simp add*:*irrefl-def*)

declare [[*simp-depth-limit* $= 2$]]
lemma *trans-diff-Id*: *trans* $r \Longrightarrow$ *antisym* $r \Longrightarrow$ *trans* ($r{-}Id$)
by(*simp add*: *antisym-def trans-def*) *blast*
declare [[*simp-depth-limit* $= 50$]]

lemma *total-on-diff-Id*[*simp*]: *total-on* A ($r{-}Id$) $=$ *total-on* A r
by(*simp add*: *total-on-def*)

31.1 Orders on a set

definition *preorder-on* A $r \equiv$ *refl* A $r \wedge$ *trans* r

definition *partial-order-on A r* ≡ *preorder-on A r* ∧ *antisym r*

definition *linear-order-on A r* ≡ *partial-order-on A r* ∧ *total-on A r*

definition *strict-linear-order-on A r* ≡ *trans r* ∧ *irrefl r* ∧ *total-on A r*

definition *well-order-on A r* ≡ *linear-order-on A r* ∧ *wf*(*r* − *Id*)

lemmas *order-on-defs* =
preorder-on-def partial-order-on-def linear-order-on-def
strict-linear-order-on-def well-order-on-def

lemma *preorder-on-empty*[*simp*]: *preorder-on* {} {}
by(*simp add*:*preorder-on-def trans-def*)

lemma *partial-order-on-empty*[*simp*]: *partial-order-on* {} {}
by(*simp add*:*partial-order-on-def*)

lemma *lnear-order-on-empty*[*simp*]: *linear-order-on* {} {}
by(*simp add*:*linear-order-on-def*)

lemma *well-order-on-empty*[*simp*]: *well-order-on* {} {}
by(*simp add*:*well-order-on-def*)

lemma *preorder-on-converse*[*simp*]: *preorder-on A* (*r*^−*1*) = *preorder-on A r*
by (*simp add*:*preorder-on-def*)

lemma *partial-order-on-converse*[*simp*]:
partial-order-on A (*r*^−*1*) = *partial-order-on A r*
by (*simp add*: *partial-order-on-def*)

lemma *linear-order-on-converse*[*simp*]:
linear-order-on A (*r*^−*1*) = *linear-order-on A r*
by (*simp add*: *linear-order-on-def*)

lemma *strict-linear-order-on-diff-Id*:
linear-order-on A r ⟹ *strict-linear-order-on A* (*r*−*Id*)
by(*simp add*: *order-on-defs trans-diff-Id*)

31.2 Orders on the field

abbreviation *Refl r* ≡ *refl* (*Field r*) *r*

abbreviation *Preorder r* ≡ *preorder-on* (*Field r*) *r*

abbreviation *Partial-order r* ≡ *partial-order-on* (*Field r*) *r*

abbreviation *Total r* ≡ *total-on* (*Field r*) *r*

abbreviation *Linear-order r* ≡ *linear-order-on* (*Field r*) *r*

abbreviation *Well-order r* ≡ *well-order-on* (*Field r*) *r*

lemma *subset-Image-Image-iff*:
⟦ *Preorder r*; *A* ⊆ *Field r*; *B* ⊆ *Field r*⟧ ⟹
r `` *A* ⊆ *r* `` *B* ⟷ (∀ *a*∈*A*.∃ *b*∈*B*. (*b*,*a*):*r*)
apply(*auto simp add*: *subset-eq preorder-on-def refl-def Image-def*)
apply *metis*
by(*metis trans-def*)

lemma *subset-Image1-Image1-iff*:
⟦ *Preorder r*; *a* : *Field r*; *b* : *Field r*⟧ ⟹ *r* `` {*a*} ⊆ *r* `` {*b*} ⟷ (*b*,*a*):*r*
by(*simp add*:*subset-Image-Image-iff*)

lemma *Refl-antisym-eq-Image1-Image1-iff*:
⟦*Refl r*; *antisym r*; *a*:*Field r*; *b*:*Field r*⟧ ⟹ *r* `` {*a*} = *r* `` {*b*} ⟷ *a*=*b*
by(*simp add*: *expand-set-eq antisym-def refl-def*) *metis*

lemma *Partial-order-eq-Image1-Image1-iff*:
⟦*Partial-order r*; *a*:*Field r*; *b*:*Field r*⟧ ⟹ *r* `` {*a*} = *r* `` {*b*} ⟷ *a*=*b*
by(*auto simp*:*order-on-defs Refl-antisym-eq-Image1-Image1-iff*)

31.3 Orders on a type

abbreviation *strict-linear-order* ≡ *strict-linear-order-on UNIV*

abbreviation *linear-order* ≡ *linear-order-on UNIV*

abbreviation *well-order r* ≡ *well-order-on UNIV*

end

32 Zorn: Zorn's Lemma

theory *Zorn*
imports *Order-Relation*
begin

definition *chain-subset* :: *'a set set* ⇒ *bool* (*chain*$_{\subseteq}$) **where**
chain$_{\subseteq}$ *C* ≡ ∀ *A*∈*C*.∀ *B*∈*C*. *A* ⊆ *B* ∨ *B* ⊆ *A*

The lemma and section numbers refer to an unpublished article [**?**].

definition
chain :: *'a set set => 'a set set set* **where**
chain S = $\{F.\ F \subseteq S\ \&\ chain_{\subseteq}\ F\}$

definition
super :: *['a set set,'a set set] => 'a set set set* **where**
super S c = $\{d.\ d \in chain\ S\ \&\ c \subset d\}$

definition
maxchain :: *'a set set => 'a set set set* **where**
maxchain S = $\{c.\ c \in chain\ S\ \&\ super\ S\ c = \{\}\}$

definition
succ :: *['a set set,'a set set] => 'a set set* **where**
succ S c =
(*if* $c \notin chain\ S \mid c \in maxchain\ S$
then c else SOME $c'.\ c' \in super\ S\ c$)

inductive-set
TFin :: *'a set set => 'a set set set*
for *S* :: *'a set set*
where
succI: $x \in TFin\ S ==> succ\ S\ x \in TFin\ S$
| *Pow-UnionI*: $Y \in Pow(TFin\ S) ==> Union(Y) \in TFin\ S$

32.1 Mathematical Preamble

lemma *Union-lemma0*:
$(\forall x \in C.\ x \subseteq A \mid B \subseteq x) ==> Union(C) \subseteq A \mid B \subseteq Union(C)$
by *blast*

This is theorem *increasingD2* of ZF/Zorn.thy

lemma *Abrial-axiom1*: $x \subseteq succ\ S\ x$
apply (*auto simp add*: *succ-def super-def maxchain-def*)
apply (*rule contrapos-np*, *assumption*)
apply (*rule-tac Q=*$\lambda S.\ xa \in S$ **in** *someI2*, *blast+*)
done

lemmas *TFin-UnionI* = *TFin.Pow-UnionI* [*OF PowI*]

lemma *TFin-induct*:
assumes *H*: $n \in TFin\ S$
and *I*: $!!x.\ x \in TFin\ S ==> P\ x ==> P\ (succ\ S\ x)$
$!!Y.\ Y \subseteq TFin\ S ==> Ball\ Y\ P ==> P(Union\ Y)$
shows *P n* **using** *H*
apply (*induct rule*: *TFin.induct* [**where** *P=P*])
apply (*blast intro*: *I*)+
done

lemma *succ-trans*: $x \subseteq y \Longrightarrow x \subseteq succ\ S\ y$
apply (*erule subset-trans*)
apply (*rule Abrial-axiom1*)
done

Lemma 1 of section 3.1

lemma *TFin-linear-lemma1*:
$[|\ n \in TFin\ S;\ \ m \in TFin\ S;$
$\forall x \in TFin\ S.\ x \subseteq m \longrightarrow x = m \mid succ\ S\ x \subseteq m$
$|] \Longrightarrow n \subseteq m \mid succ\ S\ m \subseteq n$
apply (*erule TFin-induct*)
apply (*erule-tac* [*2*] *Union-lemma0*)
apply (*blast del*: *subsetI intro*: *succ-trans*)
done

Lemma 2 of section 3.2

lemma *TFin-linear-lemma2*:
$m \in TFin\ S \Longrightarrow \forall n \in TFin\ S.\ n \subseteq m \longrightarrow n{=}m \mid succ\ S\ n \subseteq m$
apply (*erule TFin-induct*)
apply (*rule impI* [*THEN ballI*])

case split using *TFin-linear-lemma1*

apply (*rule-tac n1* $= n$ **and** *m1* $= x$ **in** *TFin-linear-lemma1* [*THEN disjE*],
assumption+)
apply (*drule-tac x* $= n$ **in** *bspec*, *assumption*)
apply (*blast del*: *subsetI intro*: *succ-trans*, *blast*)

second induction step

apply (*rule impI* [*THEN ballI*])
apply (*rule Union-lemma0* [*THEN disjE*])
apply (*rule-tac* [*3*] *disjI2*)
prefer *2* **apply** *blast*
apply (*rule ballI*)
apply (*rule-tac n1* $= n$ **and** *m1* $= x$ **in** *TFin-linear-lemma1* [*THEN disjE*],
assumption+, *auto*)
apply (*blast intro*!: *Abrial-axiom1* [*THEN subsetD*])
done

Re-ordering the premises of Lemma 2

lemma *TFin-subsetD*:
$[|\ n \subseteq m;\ \ m \in TFin\ S;\ \ n \in TFin\ S\ |] \Longrightarrow n{=}m \mid succ\ S\ n \subseteq m$
by (*rule TFin-linear-lemma2* [*rule-format*])

Consequences from section 3.3 – Property 3.2, the ordering is total

lemma *TFin-subset-linear*: $[|\ m \in TFin\ S;\ \ n \in TFin\ S|] \Longrightarrow n \subseteq m \mid m \subseteq n$
apply (*rule disjE*)
apply (*rule TFin-linear-lemma1* [*OF - -TFin-linear-lemma2*])
apply (*assumption*+, *erule disjI2*)

apply (*blast del*: *subsetI*
intro: *subsetI Abrial-axiom1* [*THEN subset-trans*])
done

Lemma 3 of section 3.3

lemma *eq-succ-upper*: [| $n \in TFin\ S$; $m \in TFin\ S$; $m = succ\ S\ m$ |] ==> $n \subseteq m$
apply (*erule TFin-induct*)
apply (*drule TFin-subsetD*)
apply (*assumption+*, *force*, *blast*)
done

Property 3.3 of section 3.3

lemma *equal-succ-Union*: $m \in TFin\ S$ ==> $(m = succ\ S\ m) = (m = Union(TFin\ S))$
apply (*rule iffI*)
apply (*rule Union-upper* [*THEN equalityI*])
apply *assumption*
apply (*rule eq-succ-upper* [*THEN Union-least*], *assumption+*)
apply (*erule ssubst*)
apply (*rule Abrial-axiom1* [*THEN equalityI*])
apply (*blast del*: *subsetI intro*: *subsetI TFin-UnionI TFin.succI*)
done

32.2 Hausdorff's Theorem: Every Set Contains a Maximal Chain.

NB: We assume the partial ordering is $\subseteq$, the subset relation!

lemma *empty-set-mem-chain*: $(\{\} :: 'a\ set\ set) \in chain\ S$
by (*unfold chain-def chain-subset-def*) *auto*

lemma *super-subset-chain*: $super\ S\ c \subseteq chain\ S$
by (*unfold super-def*) *blast*

lemma *maxchain-subset-chain*: $maxchain\ S \subseteq chain\ S$
by (*unfold maxchain-def*) *blast*

lemma *mem-super-Ex*: $c \in chain\ S - maxchain\ S$ ==> $EX\ d.\ d \in super\ S\ c$
by (*unfold super-def maxchain-def*) *auto*

lemma *select-super*:
$c \in chain\ S - maxchain\ S$ ==> $(\epsilon\ c'.\ c' : super\ S\ c) : super\ S\ c$
apply (*erule mem-super-Ex* [*THEN exE*])
apply (*rule someI2* [**where** $Q = \%X.\ X : super\ S\ c$], *auto*)
done

lemma *select-not-equals*:
$c \in chain\ S - maxchain\ S$ ==> $(\epsilon\ c'.\ c' : super\ S\ c) \neq c$

apply (*rule notI*)
apply (*drule select-super*)
apply (*simp add*: *super-def less-le*)
done

lemma *succI3*: $c \in chain\ S - maxchain\ S \Longrightarrow succ\ S\ c = (\epsilon\, c'.\ c'$: *super* $S\ c)$
by (*unfold succ-def*) (*blast intro*!: *if-not-P*)

lemma *succ-not-equals*: $c \in chain\ S - maxchain\ S \Longrightarrow succ\ S\ c \neq c$
apply (*frule succI3*)
apply (*simp* (*no-asm-simp*))
apply (*rule select-not-equals*, *assumption*)
done

lemma *TFin-chain-lemma4*: $c \in TFin\ S \Longrightarrow (c :: 'a\ set\ set)$: *chain S*
apply (*erule TFin-induct*)
apply (*simp add*: *succ-def select-super* [*THEN super-subset-chain*[*THEN subsetD*]])
apply (*unfold chain-def chain-subset-def*)
apply (*rule CollectI*, *safe*)
apply (*drule bspec*, *assumption*)
apply (*rule-tac* [*2*] *m1* = *Xa* **and** *n1* = *X* **in** *TFin-subset-linear* [*THEN disjE*], *best*+)
done

theorem *Hausdorff*: $\exists\, c.\ (c :: 'a\ set\ set)$: *maxchain S*
apply (*rule-tac x* = *Union* (*TFin S*) **in** *exI*)
apply (*rule classical*)
apply (*subgoal-tac succ S* (*Union* (*TFin S*)) = *Union* (*TFin S*))
prefer *2*
apply (*blast intro*!: *TFin-UnionI equal-succ-Union* [*THEN iffD2*, *symmetric*])
apply (*cut-tac subset-refl* [*THEN TFin-UnionI*, *THEN TFin-chain-lemma4*])
apply (*drule DiffI* [*THEN succ-not-equals*], *blast*+)
done

32.3 Zorn's Lemma: If All Chains Have Upper Bounds Then There Is a Maximal Element

lemma *chain-extend*:
$[|\ c \in chain\ S;\ z \in S;\ \forall x \in c.\ x \subseteq (z :: 'a\ set)\ |] \Longrightarrow \{z\}\ Un\ c \in chain\ S$
by (*unfold chain-def chain-subset-def*) *blast*

lemma *chain-Union-upper*: $[|\ c \in chain\ S;\ x \in c\ |] \Longrightarrow x \subseteq Union(c)$
by *auto*

lemma *chain-ball-Union-upper*: $c \in chain\ S \Longrightarrow \forall x \in c.\ x \subseteq Union(c)$
by *auto*

lemma *maxchain-Zorn*:

[| $c \in$ *maxchain* S; $u \in S$; *Union*$(c) \subseteq u$ |] ==> *Union*$(c) = u$
apply (*rule ccontr*)
apply (*simp add*: *maxchain-def*)
apply (*erule conjE*)
apply (*subgoal-tac* $(\{u\}$ *Un* $c) \in$ *super* S c)
apply *simp*
apply (*unfold super-def less-le*)
apply (*blast intro*: *chain-extend dest*: *chain-Union-upper*)
done

theorem *Zorn-Lemma*:
$\forall c \in$ *chain* S. *Union*(c): S ==> $\exists y \in S. \forall z \in S. y \subseteq z$ --> $y = z$
apply (*cut-tac Hausdorff maxchain-subset-chain*)
apply (*erule exE*)
apply (*drule subsetD*, *assumption*)
apply (*drule bspec*, *assumption*)
apply (*rule-tac* x = *Union*(c) **in** *bexI*)
apply (*rule ballI*, *rule impI*)
apply (*blast dest*!: *maxchain-Zorn*, *assumption*)
done

32.4 Alternative version of Zorn's Lemma

lemma *Zorn-Lemma2*:
$\forall c \in$ *chain* S. $\exists y \in S. \forall x \in c. x \subseteq y$
==> $\exists y \in S. \forall x \in S.$ $(y :: 'a$ *set*$) \subseteq x$ --> $y = x$
apply (*cut-tac Hausdorff maxchain-subset-chain*)
apply (*erule exE*)
apply (*drule subsetD*, *assumption*)
apply (*drule bspec*, *assumption*, *erule bexE*)
apply (*rule-tac* x = y **in** *bexI*)
prefer *2* **apply** *assumption*
apply *clarify*
apply (*rule ccontr*)
apply (*frule-tac* z = x **in** *chain-extend*)
apply (*assumption*, *blast*)
apply (*unfold maxchain-def super-def less-le*)
apply (*blast elim*!: *equalityCE*)
done

Various other lemmas

lemma *chainD*: [| $c \in$ *chain* S; $x \in c$; $y \in c$ |] ==> $x \subseteq y \mid y \subseteq x$
by (*unfold chain-def chain-subset-def*) *blast*

lemma *chainD2*: !!$(c :: 'a$ *set set*$)$. $c \in$ *chain* S ==> $c \subseteq S$
by (*unfold chain-def*) *blast*

definition *Chain* :: $('a*'a)set \Rightarrow 'a\ set\ set$ **where**
$Chain\ r \equiv \{A.\ \forall a{\in}A.\forall b{\in}A.\ (a,b) : r \vee (b,a) \in r\}$

lemma *mono-Chain*: $r \subseteq s \Longrightarrow Chain\ r \subseteq Chain\ s$
unfolding *Chain-def* **by** *blast*

Zorn's lemma for partial orders:

lemma *Zorns-po-lemma*:
assumes *po*: *Partial-order r* **and** *u*: $\forall C{\in}Chain\ r.\ \exists u{\in}Field\ r.\ \forall a{\in}C.\ (a,u){:}r$
shows $\exists m{\in}Field\ r.\ \forall a{\in}Field\ r.\ (m,a){:}r \longrightarrow a{=}m$
proof−
have *Preorder r* **using** *po* **by**(*simp add*:*partial-order-on-def*)
— Mirror r in the set of subsets below (wrt r) elements of A
let *?B* = $\%x.\ r\hat{}{-}1\ ``\ \{x\}$ **let** *?S* = *?B* ' *Field r*
have $\forall C \in chain\ ?S.\ EX\ U{:}?S.\ ALL\ A{:}C.\ A{\subseteq}U$
proof (*auto simp*:*chain-def chain-subset-def*)
fix *C* **assume** *1*: $C \subseteq ?S$ **and** *2*: $\forall A{\in}C.\forall B{\in}C.\ A{\subseteq}B \mid B{\subseteq}A$
let *?A* = $\{x{\in}Field\ r.\ \exists M{\in}C.\ M = ?B\ x\}$
have $C = ?B\ `\ ?A$ **using** *1* **by**(*auto simp*: *image-def*)
have *?A*∈*Chain r*
proof (*simp add*:*Chain-def*, *intro allI impI*, *elim conjE*)
fix *a b*
assume $a \in Field\ r\ ?B\ a \in C\ b \in Field\ r\ ?B\ b \in C$
hence $?B\ a \subseteq ?B\ b \vee ?B\ b \subseteq ?B\ a$ **using** *2* **by** *auto*
thus $(a,\ b) \in r \vee (b,\ a) \in r$ **using** ‹*Preorder r*› ‹*a*:*Field r*› ‹*b*:*Field r*›
by(*simp add*:*subset-Image1-Image1-iff*)
qed
then obtain *u* **where** *uA*: *u*:*Field r* $\forall a{\in}?A.\ (a,u) : r$ **using** *u* **by** *auto*
have $\forall A{\in}C.\ A \subseteq r\hat{}{-}1\ ``\ \{u\}$ (**is** *?P u*)
proof *auto*
fix *a B* **assume** *aB*: *B*:*C a*:*B*
with *1* **obtain** *x* **where** *x*:*Field r* $B = r\hat{}{-}1\ ``\ \{x\}$ **by** *auto*
thus $(a,u) : r$ **using** *uA aB* ‹*Preorder r*›
by (*auto simp add*: *preorder-on-def refl-def*) (*metis transD*)
qed
thus *EX u*:*Field r*. *?P u* **using** ‹*u*:*Field r*› **by** *blast*
qed
from *Zorn-Lemma2*[*OF this*]
obtain *m B* **where** *m*:*Field r* $B = r\hat{}{-}1\ ``\ \{m\}$
$\forall x{\in}Field\ r.\ B \subseteq r\hat{}{-}1\ ``\ \{x\} \longrightarrow B = r\hat{}{-}1\ ``\ \{x\}$
by *auto*
hence $\forall a{\in}Field\ r.\ (m,\ a) \in r \longrightarrow a = m$ **using** *po* ‹*Preorder r*› ‹*m*:*Field r*›
by(*auto simp*:*subset-Image1-Image1-iff Partial-order-eq-Image1-Image1-iff*)
thus *?thesis* **using** ‹*m*:*Field r*› **by** *blast*
qed

definition *init-seg-of* :: $(('a*'a)set * ('a*'a)set)set$ **where**
$init\text{-}seg\text{-}of == \{(r,s).\ r \subseteq s \wedge (\forall a\ b\ c.\ (a,b){:}s \wedge (b,c){:}r \longrightarrow (a,b){:}r)\}$

abbreviation *initialSegmentOf* :: $('a*'a)set \Rightarrow ('a*'a)set \Rightarrow bool$
(**infix** *initial'-segment'-of 55*) **where**
r initial-segment-of s == (*r*,*s*):*init-seg-of*

lemma *refl-init-seg-of*[*simp*]: *r initial-segment-of r*
by(*simp add*:*init-seg-of-def*)

lemma *trans-init-seg-of*:
r initial-segment-of s $\Longrightarrow$ *s initial-segment-of t* $\Longrightarrow$ *r initial-segment-of t*
by(*simp* (*no-asm-use*) *add*: *init-seg-of-def*)
(*metis Domain-iff UnCI Un-absorb2 subset-trans*)

lemma *antisym-init-seg-of*:
r initial-segment-of s $\Longrightarrow$ *s initial-segment-of r* $\Longrightarrow$ *r*=*s*
by(*auto simp*:*init-seg-of-def*)

lemma *Chain-init-seg-of-Union*:
R $\in$ *Chain init-seg-of* $\Longrightarrow$ *r*$\in$*R* $\Longrightarrow$ *r initial-segment-of* $\bigcup R$
by(*auto simp add*:*init-seg-of-def Chain-def Ball-def*) *blast*

lemma *chain-subset-trans-Union*:
$chain_{\subseteq}$ *R* $\Longrightarrow$ $\forall$ *r*$\in$*R*. *trans r* $\Longrightarrow$ *trans*($\bigcup R$)
apply(*auto simp add*:*chain-subset-def*)
apply(*simp* (*no-asm-use*) *add*:*trans-def*)
apply (*metis subsetD*)
done

lemma *chain-subset-antisym-Union*:
$chain_{\subseteq}$ *R* $\Longrightarrow$ $\forall$ *r*$\in$*R*. *antisym r* $\Longrightarrow$ *antisym*($\bigcup R$)
apply(*auto simp add*:*chain-subset-def antisym-def*)
apply (*metis subsetD*)
done

lemma *chain-subset-Total-Union*:
assumes $chain_{\subseteq}$ *R* $\forall$ *r*$\in$*R*. *Total r*
shows *Total* ($\bigcup R$)
proof (*simp add*: *total-on-def Ball-def*, *auto del*:*disjCI*)
fix *r s a b* **assume** *A*: *r*:*R s*:*R a*:*Field r b*:*Field s a*$\neq$*b*
from ‹$chain_{\subseteq}$ *R*› ‹*r*:*R*› ‹*s*:*R*› **have** *r*$\subseteq$*s* $\vee$ *s*$\subseteq$*r*
by(*simp add*:*chain-subset-def*)
thus ($\exists$ *r*$\in$*R*. (*a*,*b*) $\in$ *r*) $\vee$ ($\exists$ *r*$\in$*R*. (*b*,*a*) $\in$ *r*)
proof
assume *r*$\subseteq$*s* **hence** (*a*,*b*):*s* $\vee$ (*b*,*a*):*s* **using** *assms*(*2*) *A*
by(*simp add*:*total-on-def*)(*metis mono-Field subsetD*)
thus *?thesis* **using** ‹*s*:*R*› **by** *blast*
next
assume *s*$\subseteq$*r* **hence** (*a*,*b*):*r* $\vee$ (*b*,*a*):*r* **using** *assms*(*2*) *A*
by(*simp add*:*total-on-def*)(*metis mono-Field subsetD*)

thus *?thesis* **using** ‹*r*:*R*› **by** *blast*
qed
qed

lemma *wf-Union-wf-init-segs*:
assumes $R \in$ *Chain init-seg-of* **and** $\forall\, r{\in}R.$ *wf r* **shows** *wf*$(\bigcup R)$
proof(*simp add*:*wf-iff-no-infinite-down-chain*, *rule ccontr*, *auto*)
fix *f* **assume** *1*: $\forall\, i.\ \exists\, r{\in}R.\ (f(Suc\ i), f\ i) \in r$
then obtain *r* **where** *r*:*R* **and** $(f(Suc\ 0), f\ 0) : r$ **by** *auto*
{ fix *i* **have** $(f(Suc\ i), f\ i) \in r$
proof(*induct i*)
case *0* **show** *?case* **by** *fact*
next
case (*Suc i*)
moreover obtain *s* **where** $s{\in}R$ **and** $(f(Suc(Suc\ i)), f(Suc\ i)) \in s$
using *1* **by** *auto*
moreover hence *s initial-segment-of r* $\vee$ *r initial-segment-of s*
using *assms*(*1*) ‹*r*:*R*› **by**(*simp add*: *Chain-def*)
ultimately show *?case* **by**(*simp add*:*init-seg-of-def*) *blast*
qed
}
thus *False* **using** *assms*(*2*) ‹*r*:*R*›
by(*simp add*:*wf-iff-no-infinite-down-chain*) *blast*
qed

lemma *Chain-inits-DiffI*:
$R \in$ *Chain init-seg-of* $\Longrightarrow \{r - s\ |r.\ r \in R\} \in$ *Chain init-seg-of*
apply(*auto simp*:*Chain-def init-seg-of-def*)
apply (*metis subsetD*)
apply (*metis subsetD*)
done

theorem *well-ordering*: $\exists\, r{::}('a{*}'a)set.$ *Well-order r* $\wedge$ *Field r* $=$ *UNIV*
proof−
— The initial segment relation on well-orders:
let *?WO* $= \{r{::}('a{*}'a)set.$ *Well-order r*$\}$
def *I* $\equiv$ *init-seg-of* $\cap$ *?WO* $\times$ *?WO*
have *I-init*: *I* $\subseteq$ *init-seg-of* **by**(*auto simp*:*I-def*)
hence *subch*: $!!R.$ *R* : *Chain I* $\Longrightarrow$ *chain*$_{\subseteq}$ *R*
by(*auto simp*:*init-seg-of-def chain-subset-def Chain-def*)
have *Chain-wo*: $!!R\ r.$ *R* $\in$ *Chain I* $\Longrightarrow$ *r* $\in$ *R* $\Longrightarrow$ *Well-order r*
by(*simp add*:*Chain-def I-def*) *blast*
have *FI*: *Field I* $=$ *?WO* **by**(*auto simp add*:*I-def init-seg-of-def Field-def*)
hence *0*: *Partial-order I*
by(*auto simp*: *partial-order-on-def preorder-on-def antisym-def antisym-init-seg-of refl-def trans-def I-def elim*!: *trans-init-seg-of*)
— I-chains have upper bounds in ?WO wrt I: their Union
{ fix *R* **assume** *R* $\in$ *Chain I*
hence *Ris*: *R* $\in$ *Chain init-seg-of* **using** *mono-Chain*[*OF I-init*] **by** *blast*

have *subch*: $chain_{\subseteq}\ R$ **using** ‹R : *Chain I*› *I-init*
by(*auto simp*:*init-seg-of-def chain-subset-def Chain-def*)
have $\forall r{\in}R.$ *Refl r* $\forall r{\in}R.$ *trans r* $\forall r{\in}R.$ *antisym r* $\forall r{\in}R.$ *Total r*
$\forall r{\in}R.$ *wf*($r{-}Id$)
using *Chain-wo*[*OF* ‹$R \in$ *Chain I*›] **by**(*simp-all add*:*order-on-defs*)
have *Refl* ($\bigcup R$) **using** ‹$\forall r{\in}R.$ *Refl r*› **by**(*auto simp*:*refl-def*)
moreover have *trans* ($\bigcup R$)
by(*rule chain-subset-trans-Union*[*OF subch* ‹$\forall r{\in}R.$ *trans r*›])
moreover have *antisym*($\bigcup R$)
by(*rule chain-subset-antisym-Union*[*OF subch* ‹$\forall r{\in}R.$ *antisym r*›])
moreover have *Total* ($\bigcup R$)
by(*rule chain-subset-Total-Union*[*OF subch* ‹$\forall r{\in}R.$ *Total r*›])
moreover have *wf*(($\bigcup R){-}Id$)
proof−
have $(\bigcup R){-}Id = \bigcup\{r{-}Id|r.\ r \in R\}$ **by** *blast*
with ‹$\forall r{\in}R.$ *wf*($r{-}Id$)› *wf-Union-wf-init-segs*[*OF Chain-inits-DiffI*[*OF Ris*]]
show *?thesis* **by** (*simp* (*no-asm-simp*)) *blast*
qed
ultimately have *Well-order* ($\bigcup R$) **by**(*simp add*:*order-on-defs*)
moreover have $\forall r \in R.$ *r initial-segment-of* $\bigcup R$ **using** *Ris*
by(*simp add*: *Chain-init-seg-of-Union*)
ultimately have $\bigcup R$: *?WO* $\wedge$ ($\forall r{\in}R.$ (r,$\bigcup R$) : I)
using *mono-Chain*[*OF I-init*] ‹$R \in$ *Chain I*›
by(*simp* (*no-asm*) *add*:*I-def del*:*Field-Union*)(*metis Chain-wo subsetD*)
}
hence *1*: $\forall R \in$ *Chain I*. $\exists u{\in}$*Field I*. $\forall r{\in}R.$ (r,u) : I **by** (*subst FI*) *blast*
— Zorn's Lemma yields a maximal well-order m:
then obtain m::($'a{*}'a$)*set* **where** *Well-order m* **and**
max: $\forall r.$ *Well-order r* $\wedge$ (m,r):$I \longrightarrow r{=}m$
using *Zorns-po-lemma*[*OF 0 1*] **by** (*auto simp*:*FI*)
— Now show by contradiction that m covers the whole type:
{ **fix** x::$'a$ **assume** $x \notin$ *Field m*
— We assume that x is not covered and extend m at the top with x
have $m \neq \{\}$
proof
assume $m{=}\{\}$
moreover have *Well-order* $\{(x,x)\}$
by(*simp add*:*order-on-defs refl-def trans-def antisym-def total-on-def Field-def Domain-def Range-def*)
ultimately show *False* **using** *max*
by (*auto simp*:*I-def init-seg-of-def simp del*:*Field-insert*)
qed
hence *Field m* $\neq \{\}$ **by**(*auto simp*:*Field-def*)
moreover have *wf*($m{-}Id$) **using** ‹*Well-order m*›
by(*simp add*:*well-order-on-def*)
— The extension of m by x:
let *?s* = $\{(a,x)|a.\ a$: *Field m*$\}$ **let** *?m* = *insert* (x,x) *m Un ?s*
have *Fm*: *Field ?m* = *insert x* (*Field m*)
apply(*simp add*:*Field-insert Field-Un*)

```
    unfolding Field-def by auto
  have Refl m trans m antisym m Total m wf(m−Id)
    using ‹Well-order m› by(simp-all add:order-on-defs)
— We show that the extension is a well-order
  have Refl ?m using ‹Refl m› Fm by(auto simp:refl-def)
  moreover have trans ?m using ‹trans m› ‹x ∉ Field m›
    unfolding trans-def Field-def Domain-def Range-def by blast
  moreover have antisym ?m using ‹antisym m› ‹x ∉ Field m›
    unfolding antisym-def Field-def Domain-def Range-def by blast
  moreover have Total ?m using ‹Total m› Fm by(auto simp: total-on-def)
  moreover have wf(?m−Id)
  proof−
    have wf ?s using ‹x ∉ Field m›
      by(auto simp add:wf-eq-minimal Field-def Domain-def Range-def) metis
    thus ?thesis using ‹wf(m−Id)› ‹x ∉ Field m›
      wf-subset[OF ‹wf ?s› Diff-subset]
      by (fastsimp intro!: wf-Un simp add: Un-Diff Field-def)
  qed
  ultimately have Well-order ?m by(simp add:order-on-defs)
— We show that the extension is above m
  moreover hence (m,?m) : I using ‹Well-order m› ‹x ∉ Field m›
    by(fastsimp simp:I-def init-seg-of-def Field-def Domain-def Range-def)
  ultimately
— This contradicts maximality of m:
  have False using max ‹x ∉ Field m› unfolding Field-def by blast
 }
 hence Field m = UNIV by auto
 moreover with ‹Well-order m› have Well-order m by simp
 ultimately show ?thesis by blast
qed

corollary well-order-on: ∃ r::('a*'a)set. well-order-on A r
proof −
 obtain r::('a*'a)set where wo: Well-order r and univ: Field r = UNIV
   using well-ordering[where 'a = 'a] by blast
 let ?r = {(x,y). x:A & y:A & (x,y):r}
 have 1: Field ?r = A using wo univ
   by(fastsimp simp: Field-def Domain-def Range-def order-on-defs refl-def)
 have Refl r trans r antisym r Total r wf(r−Id)
   using ‹Well-order r› by(simp-all add:order-on-defs)
 have Refl ?r using ‹Refl r› by(auto simp:refl-def 1 univ)
 moreover have trans ?r using ‹trans r›
   unfolding trans-def by blast
 moreover have antisym ?r using ‹antisym r›
   unfolding antisym-def by blast
 moreover have Total ?r using ‹Total r› by(simp add:total-on-def 1 univ)
 moreover have wf(?r − Id) by(rule wf-subset[OF ‹wf(r−Id)›]) blast
 ultimately have Well-order ?r by(simp add:order-on-defs)
 with 1 show ?thesis by metis
```

qed

end

33 Filter: Filters and Ultrafilters

theory *Filter*
imports ~~/*src*/*HOL*/*Library*/*Zorn* ~~/*src*/*HOL*/*Library*/*Infinite-Set*
begin

33.1 Definitions and basic properties

33.1.1 Filters

locale *filter* =
  fixes F :: $'a$ *set set*
  assumes *UNIV* [*iff*]: $UNIV \in F$
  assumes *empty* [*iff*]: $\{\} \notin F$
  assumes *Int*: $[\![u \in F; v \in F]\!] \Longrightarrow u \cap v \in F$
  assumes *subset*: $[\![u \in F; u \subseteq v]\!] \Longrightarrow v \in F$

lemma (**in** *filter*) *memD*: $A \in F \Longrightarrow - A \notin F$
proof
  assume $A \in F$ **and** $- A \in F$
  hence $A \cap (- A) \in F$ **by** (*rule Int*)
  thus *False* **by** *simp*
qed

lemma (**in** *filter*) *not-memI*: $- A \in F \Longrightarrow A \notin F$
by (*drule memD*, *simp*)

lemma (**in** *filter*) *Int-iff*: $(x \cap y \in F) = (x \in F \wedge y \in F)$
by (*auto elim*: *subset intro*: *Int*)

33.1.2 Ultrafilters

locale *ultrafilter* = *filter* +
  assumes *ultra*: $A \in F \vee - A \in F$

lemma (**in** *ultrafilter*) *memI*: $- A \notin F \Longrightarrow A \in F$
by (*cut-tac ultra* [*of A*], *simp*)

lemma (**in** *ultrafilter*) *not-memD*: $A \notin F \Longrightarrow - A \in F$
by (*rule memI*, *simp*)

lemma (**in** *ultrafilter*) *not-mem-iff*: $(A \notin F) = (- A \in F)$
by (*rule iffI* [*OF not-memD not-memI*])

lemma (**in** *ultrafilter*) *Compl-iff*: $(-\ A \in F) = (A \notin F)$
by (*rule iffI* [*OF not-memI not-memD*])

lemma (**in** *ultrafilter*) *Un-iff*: $(x \cup y \in F) = (x \in F \vee y \in F)$
apply (*rule iffI*)
apply (*erule contrapos-pp*)
apply (*simp add*: *Int-iff not-mem-iff*)
apply (*auto elim*: *subset*)
done

33.1.3 Free Ultrafilters

locale *freeultrafilter* = *ultrafilter* +
assumes *infinite*: $A \in F \Longrightarrow$ *infinite* A

lemma (**in** *freeultrafilter*) *finite*: *finite* $A \Longrightarrow A \notin F$
by (*erule contrapos-pn*, *erule infinite*)

lemma (**in** *freeultrafilter*) *singleton*: $\{x\} \notin F$
by (*rule finite*, *simp*)

lemma (**in** *freeultrafilter*) *insert-iff* [*simp*]: $(\mathit{insert}\ x\ A \in F) = (A \in F)$
apply (*subst insert-is-Un*)
apply (*subst Un-iff*)
apply (*simp add*: *singleton*)
done

lemma (**in** *freeultrafilter*) *filter*: *filter* F **by** *unfold-locales*

lemma (**in** *freeultrafilter*) *ultrafilter*: *ultrafilter* F
by *unfold-locales*

33.2 Collect properties

lemma (**in** *filter*) *Collect-ex*:
$(\{n.\ \exists x.\ P\ n\ x\} \in F) = (\exists X.\ \{n.\ P\ n\ (X\ n)\} \in F)$
proof
assume $\{n.\ \exists x.\ P\ n\ x\} \in F$
hence $\{n.\ P\ n\ (\mathit{SOME}\ x.\ P\ n\ x)\} \in F$
by (*auto elim*: *someI subset*)
thus $\exists X.\ \{n.\ P\ n\ (X\ n)\} \in F$ **by** *fast*
next
show $\exists X.\ \{n.\ P\ n\ (X\ n)\} \in F \Longrightarrow \{n.\ \exists x.\ P\ n\ x\} \in F$
by (*auto elim*: *subset*)
qed

lemma (**in** *filter*) *Collect-conj*:
$(\{n.\ P\ n \wedge Q\ n\} \in F) = (\{n.\ P\ n\} \in F \wedge \{n.\ Q\ n\} \in F)$
by (*subst Collect-conj-eq*, *rule Int-iff*)

lemma (**in** *ultrafilter*) *Collect-not*:
$(\{n.\ \neg\ P\ n\} \in F) = (\{n.\ P\ n\} \notin F)$
by (*subst Collect-neg-eq*, *rule Compl-iff*)

lemma (**in** *ultrafilter*) *Collect-disj*:
$(\{n.\ P\ n \vee Q\ n\} \in F) = (\{n.\ P\ n\} \in F \vee \{n.\ Q\ n\} \in F)$
by (*subst Collect-disj-eq*, *rule Un-iff*)

lemma (**in** *ultrafilter*) *Collect-all*:
$(\{n.\ \forall x.\ P\ n\ x\} \in F) = (\forall X.\ \{n.\ P\ n\ (X\ n)\} \in F)$
apply (*rule Not-eq-iff* [*THEN iffD1*])
apply (*simp add*: *Collect-not* [*symmetric*])
apply (*rule Collect-ex*)
done

33.3 Maximal filter = Ultrafilter

A filter F is an ultrafilter iff it is a maximal filter, i.e. whenever G is a filter and $F \subseteq G$ then $F = G$

Lemmas that shows existence of an extension to what was assumed to be a maximal filter. Will be used to derive contradiction in proof of property of ultrafilter.

lemma *extend-lemma1*: $UNIV \in F \Longrightarrow A \in \{X.\ \exists f{\in}F.\ A \cap f \subseteq X\}$
by *blast*

lemma *extend-lemma2*: $F \subseteq \{X.\ \exists f{\in}F.\ A \cap f \subseteq X\}$
by *blast*

lemma (**in** *filter*) *extend-filter*:
assumes *A*: $-\ A \notin F$
shows *filter* $\{X.\ \exists f{\in}F.\ A \cap f \subseteq X\}$ (**is** *filter ?X*)
proof (*rule filter.intro*)
  show $UNIV \in ?X$ **by** *blast*
next
  show $\{\} \notin ?X$
  proof (*clarify*)
    fix *f* **assume** *f*: $f \in F$ **and** *Af*: $A \cap f \subseteq \{\}$
    from *Af* **have** *fA*: $f \subseteq -\ A$ **by** *blast*
    from *f fA* **have** $-\ A \in F$ **by** (*rule subset*)
    with *A* **show** *False* **by** *simp*
  qed
next
  fix *u* **and** *v*
  assume *u*: $u \in ?X$ **and** *v*: $v \in ?X$
  from *u* **obtain** *f* **where** *f*: $f \in F$ **and** *Af*: $A \cap f \subseteq u$ **by** *blast*
  from *v* **obtain** *g* **where** *g*: $g \in F$ **and** *Ag*: $A \cap g \subseteq v$ **by** *blast*
  from *f g* **have** *fg*: $f \cap g \in F$ **by** (*rule Int*)
  from *Af Ag* **have** *Afg*: $A \cap (f \cap g) \subseteq u \cap v$ **by** *blast*

from *fg Afg* **show** $u \cap v \in ?X$ **by** *blast*
next
fix u **and** v
assume *uv*: $u \subseteq v$ **and** u: $u \in ?X$
from u **obtain** f **where** f: $f \in F$ **and** *Afu*: $A \cap f \subseteq u$ **by** *blast*
from *Afu uv* **have** *Afv*: $A \cap f \subseteq v$ **by** *blast*
from f *Afv* **have** $\exists f{\in}F.\ A \cap f \subseteq v$ **by** *blast*
thus $v \in ?X$ **by** *simp*
qed

lemma (**in** *filter*) *max-filter-ultrafilter*:
assumes *max*: $\bigwedge G.\ [\![filter\ G;\ F \subseteq G]\!] \Longrightarrow F = G$
shows *ultrafilter-axioms F*
proof (*rule ultrafilter-axioms.intro*)
fix A **show** $A \in F \vee -A \in F$
proof (*rule disjCI*)
let $?X = \{X.\ \exists f{\in}F.\ A \cap f \subseteq X\}$
assume *AF*: $-A \notin F$
from *AF* **have** X: *filter* $?X$ **by** (*rule extend-filter*)
from *UNIV* **have** *AX*: $A \in ?X$ **by** (*rule extend-lemma1*)
have *FX*: $F \subseteq ?X$ **by** (*rule extend-lemma2*)
from X *FX* **have** $F = ?X$ **by** (*rule max*)
with *AX* **show** $A \in F$ **by** *simp*
qed
qed

lemma (**in** *ultrafilter*) *max-filter*:
assumes G: *filter* G **and** *sub*: $F \subseteq G$ **shows** $F = G$
proof
show $F \subseteq G$ **using** *sub* **.**
show $G \subseteq F$
proof
fix A **assume** A: $A \in G$
from G A **have** $-A \notin G$ **by** (*rule filter.memD*)
with *sub* **have** B: $-A \notin F$ **by** *blast*
thus $A \in F$ **by** (*rule memI*)
qed
qed

33.4 Ultrafilter Theorem

A locale makes proof of ultrafilter Theorem more modular

locale (**open**) *UFT* =
fixes *frechet* :: *'a set set*
and *superfrechet* :: *'a set set set*

assumes *infinite-UNIV*: *infinite* (*UNIV* :: *'a set*)

defines *frechet-def*: *frechet* ≡ {*A*. *finite* (− *A*)}

and *superfrechet-def*: $superfrechet \equiv \{G.\ filter\ G \wedge frechet \subseteq G\}$

lemma (**in** *UFT*) *superfrechetI*:
$[\![filter\ G; frechet \subseteq G]\!] \Longrightarrow G \in superfrechet$
by (*simp add*: *superfrechet-def*)

lemma (**in** *UFT*) *superfrechetD1*:
$G \in superfrechet \Longrightarrow filter\ G$
by (*simp add*: *superfrechet-def*)

lemma (**in** *UFT*) *superfrechetD2*:
$G \in superfrechet \Longrightarrow frechet \subseteq G$
by (*simp add*: *superfrechet-def*)

A few properties of free filters

lemma *filter-cofinite*:
assumes *inf*: *infinite* (*UNIV* :: *'a set*)
shows *filter* {*A*:: *'a set*. *finite* (− *A*)} (**is** *filter ?F*)
proof (*rule filter.intro*)
show $UNIV \in ?F$ **by** *simp*
next
show $\{\} \notin ?F$ **using** *inf* **by** *simp*
next
fix *u v* **assume** $u \in ?F$ **and** $v \in ?F$
thus $u \cap v \in ?F$ **by** *simp*
next
fix *u v* **assume** *uv*: $u \subseteq v$ **and** *u*: $u \in ?F$
from *uv* **have** *vu*: $- v \subseteq - u$ **by** *simp*
from *u* **show** $v \in ?F$
by (*simp add*: *finite-subset* [*OF vu*])
qed

We prove: 1. Existence of maximal filter i.e. ultrafilter; 2. Freeness property i.e ultrafilter is free. Use a locale to prove various lemmas and then export main result: The ultrafilter Theorem

lemma (**in** *UFT*) *filter-frechet*: *filter frechet*
by (*unfold frechet-def*, *rule filter-cofinite* [*OF infinite-UNIV*])

lemma (**in** *UFT*) *frechet-in-superfrechet*: $frechet \in superfrechet$
by (*rule superfrechetI* [*OF filter-frechet subset-refl*])

lemma (**in** *UFT*) *lemma-mem-chain-filter*:
$[\![c \in chain\ superfrechet; x \in c]\!] \Longrightarrow filter\ x$
by (*unfold chain-def superfrechet-def*, *blast*)

33.4.1 Unions of chains of superfrechets

In this section we prove that superfrechet is closed with respect to unions of non-empty chains. We must show 1) Union of a chain is a filter, 2) Union

of a chain contains frechet.

Number 2 is trivial, but 1 requires us to prove all the filter rules.

lemma (**in** *UFT*) *Union-chain-UNIV*:
$[\![c \in \mathit{chain\ superfrechet};\ c \neq \{\}]\!] \Longrightarrow \mathit{UNIV} \in \bigcup c$
proof −
assume *1*: $c \in \mathit{chain\ superfrechet}$ **and** *2*: $c \neq \{\}$
from *2* **obtain** x **where** *3*: $x \in c$ **by** *blast*
from *1 3* **have** *filter x* **by** (*rule lemma-mem-chain-filter*)
hence $\mathit{UNIV} \in x$ **by** (*rule filter.UNIV*)
with *3* **show** $\mathit{UNIV} \in \bigcup c$ **by** *blast*
qed

lemma (**in** *UFT*) *Union-chain-empty*:
$c \in \mathit{chain\ superfrechet} \Longrightarrow \{\} \notin \bigcup c$
proof
assume *1*: $c \in \mathit{chain\ superfrechet}$ **and** *2*: $\{\} \in \bigcup c$
from *2* **obtain** x **where** *3*: $x \in c$ **and** *4*: $\{\} \in x$ **..**
from *1 3* **have** *filter x* **by** (*rule lemma-mem-chain-filter*)
hence $\{\} \notin x$ **by** (*rule filter.empty*)
with *4* **show** *False* **by** *simp*
qed

lemma (**in** *UFT*) *Union-chain-Int*:
$[\![c \in \mathit{chain\ superfrechet};\ u \in \bigcup c;\ v \in \bigcup c]\!] \Longrightarrow u \cap v \in \bigcup c$
proof −
assume *c*: $c \in \mathit{chain\ superfrechet}$
assume $u \in \bigcup c$
then obtain x **where** *ux*: $u \in x$ **and** *xc*: $x \in c$ **..**
assume $v \in \bigcup c$
then obtain y **where** *vy*: $v \in y$ **and** *yc*: $y \in c$ **..**
from *c xc yc* **have** $x \subseteq y \vee y \subseteq x$ **by** (*rule chainD*)
with *xc yc* **have** *xyc*: $x \cup y \in c$
by (*auto simp add*: *Un-absorb1 Un-absorb2*)
with *c* **have** *fxy*: *filter* $(x \cup y)$ **by** (*rule lemma-mem-chain-filter*)
from *ux* **have** *uxy*: $u \in x \cup y$ **by** *simp*
from *vy* **have** *vxy*: $v \in x \cup y$ **by** *simp*
from *fxy uxy vxy* **have** $u \cap v \in x \cup y$ **by** (*rule filter.Int*)
with *xyc* **show** $u \cap v \in \bigcup c$ **..**
qed

lemma (**in** *UFT*) *Union-chain-subset*:
$[\![c \in \mathit{chain\ superfrechet};\ u \in \bigcup c;\ u \subseteq v]\!] \Longrightarrow v \in \bigcup c$
proof −
assume *c*: $c \in \mathit{chain\ superfrechet}$
and *u*: $u \in \bigcup c$ **and** *uv*: $u \subseteq v$
from *u* **obtain** x **where** *ux*: $u \in x$ **and** *xc*: $x \in c$ **..**
from *c xc* **have** *fx*: *filter x* **by** (*rule lemma-mem-chain-filter*)
from *fx ux uv* **have** *vx*: $v \in x$ **by** (*rule filter.subset*)
with *xc* **show** $v \in \bigcup c$ **..**

qed

lemma (**in** *UFT*) *Union-chain-filter*:
assumes *chain*: $c \in$ *chain superfrechet* **and** *nonempty*: $c \neq \{\}$
shows *filter* $(\bigcup c)$
proof (*rule filter.intro*)
show $UNIV \in \bigcup c$ **using** *chain nonempty* **by** (*rule Union-chain-UNIV*)
next
show $\{\} \notin \bigcup c$ **using** *chain* **by** (*rule Union-chain-empty*)
next
fix u v **assume** $u \in \bigcup c$ **and** $v \in \bigcup c$
with *chain* **show** $u \cap v \in \bigcup c$ **by** (*rule Union-chain-Int*)
next
fix u v **assume** $u \in \bigcup c$ **and** $u \subseteq v$
with *chain* **show** $v \in \bigcup c$ **by** (*rule Union-chain-subset*)
qed

lemma (**in** *UFT*) *lemma-mem-chain-frechet-subset*:
$[\![c \in$ *chain superfrechet*; $x \in c]\!] \Longrightarrow$ *frechet* $\subseteq x$
by (*unfold superfrechet-def chain-def*, *blast*)

lemma (**in** *UFT*) *Union-chain-superfrechet*:
$[\![c \neq \{\}$; $c \in$ *chain superfrechet*$]\!] \Longrightarrow \bigcup c \in$ *superfrechet*
proof (*rule superfrechetI*)
assume *1*: $c \in$ *chain superfrechet* **and** *2*: $c \neq \{\}$
thus *filter* $(\bigcup c)$ **by** (*rule Union-chain-filter*)
from *2* **obtain** x **where** *3*: $x \in c$ **by** *blast*
from *1 3* **have** *frechet* $\subseteq x$ **by** (*rule lemma-mem-chain-frechet-subset*)
also from *3* **have** $x \subseteq \bigcup c$ **by** *blast*
finally show *frechet* $\subseteq \bigcup c$ **.**
qed

33.4.2 Existence of free ultrafilter

lemma (**in** *UFT*) *max-cofinite-filter-Ex*:
$\exists\, U{\in}$*superfrechet*. $\forall\, G{\in}$*superfrechet*. $U \subseteq G \longrightarrow U = G$
proof (*rule Zorn-Lemma2* [*rule-format*])
fix c **assume** *c*: $c \in$ *chain superfrechet*
show $\exists\, U{\in}$*superfrechet*. $\forall\, G{\in}c$. $G \subseteq U$ (**is** *?U*)
proof (*cases*)
assume $c = \{\}$
with *frechet-in-superfrechet* **show** *?U* **by** *blast*
next
assume *A*: $c \neq \{\}$
from *A c* **have** $\bigcup c \in$ *superfrechet*
by (*rule Union-chain-superfrechet*)
thus *?U* **by** *blast*
qed
qed

lemma (**in** *UFT*) *mem-superfrechet-all-infinite*:
 $[\![U \in \mathit{superfrechet};\ A \in U]\!] \Longrightarrow \mathit{infinite}\ A$
proof
 assume *U*: $U \in \mathit{superfrechet}$ **and** *A*: $A \in U$ **and** *fin*: $\mathit{finite}\ A$
 from *U* **have** *fil*: $\mathit{filter}\ U$ **and** *fre*: $\mathit{frechet} \subseteq U$
  by (*simp-all add*: *superfrechet-def*)
 from *fin* **have** $-A \in \mathit{frechet}$ **by** (*simp add*: *frechet-def*)
 with *fre* **have** *cA*: $-A \in U$ **by** (*rule subsetD*)
 from *fil A cA* **have** $A \cap -A \in U$ **by** (*rule filter.Int*)
 with *fil* **show** *False* **by** (*simp add*: *filter.empty*)
qed

There exists a free ultrafilter on any infinite set

lemma (**in** *UFT*) *freeultrafilter-ex*:
 $\exists U::'a\ \mathit{set}\ \mathit{set}.\ \mathit{freeultrafilter}\ U$
proof −
 from *max-cofinite-filter-Ex* **obtain** *U*
  where *U*: $U \in \mathit{superfrechet}$
   and *max* [*rule-format*]: $\forall G{\in}\mathit{superfrechet}.\ U \subseteq G \longrightarrow U = G$ **..**
 from *U* **have** *fil*: $\mathit{filter}\ U$ **by** (*rule superfrechetD1*)
 from *U* **have** *fre*: $\mathit{frechet} \subseteq U$ **by** (*rule superfrechetD2*)
 have *ultra*: *ultrafilter-axioms U*
 proof (*rule filter.max-filter-ultrafilter* [*OF fil*])
  fix *G* **assume** *G*: $\mathit{filter}\ G$ **and** *UG*: $U \subseteq G$
  from *fre UG* **have** $\mathit{frechet} \subseteq G$ **by** *simp*
  with *G* **have** $G \in \mathit{superfrechet}$ **by** (*rule superfrechetI*)
  from *this UG* **show** $U = G$ **by** (*rule max*)
 qed
 have *free*: *freeultrafilter-axioms U*
 proof (*rule freeultrafilter-axioms.intro*)
  fix *A* **assume** $A \in U$
  with *U* **show** $\mathit{infinite}\ A$ **by** (*rule mem-superfrechet-all-infinite*)
 qed
 show *?thesis*
 proof
  from *fil ultra free* **show** *freeultrafilter U*
   by (*rule freeultrafilter.intro* [*OF ultrafilter.intro*])

 qed
qed

lemmas *freeultrafilter-Ex* = *UFT.freeultrafilter-ex*

hide (**open**) *const filter*

end

34 StarDef: Construction of Star Types Using Ultrafilters

theory *StarDef*
imports *Filter*
uses (*transfer.ML*)
begin

34.1 A Free Ultrafilter over the Naturals

definition
FreeUltrafilterNat :: *nat set set* ($\mathcal{U}$) **where**
$\mathcal{U}$ = (*SOME U*. *freeultrafilter U*)

lemma *freeultrafilter-FreeUltrafilterNat*: *freeultrafilter* $\mathcal{U}$
apply (*unfold FreeUltrafilterNat-def*)
apply (*rule someI-ex* [**where** *P*=*freeultrafilter*])
apply (*rule freeultrafilter-Ex*)
apply (*rule nat-infinite*)
done

interpretation *FreeUltrafilterNat*: *freeultrafilter* [*FreeUltrafilterNat*]
by (*rule freeultrafilter-FreeUltrafilterNat*)

This rule takes the place of the old ultra tactic

lemma *ultra*:
⟦{*n*. *P n*} ∈ $\mathcal{U}$; {*n*. *P n* ⟶ *Q n*} ∈ $\mathcal{U}$⟧ ⟹ {*n*. *Q n*} ∈ $\mathcal{U}$
by (*simp add*: *Collect-imp-eq*
FreeUltrafilterNat.Un-iff FreeUltrafilterNat.Compl-iff)

34.2 Definition of *star* type constructor

definition
starrel :: ((*nat* ⇒ *'a*) × (*nat* ⇒ *'a*)) *set* **where**
starrel = {(*X*,*Y*). {*n*. *X n* = *Y n*} ∈ $\mathcal{U}$}

typedef *'a star* = (*UNIV* :: (*nat* ⇒ *'a*) *set*) // *starrel*
by (*auto intro*: *quotientI*)

definition
star-n :: (*nat* ⇒ *'a*) ⇒ *'a star* **where**
star-n X = *Abs-star* (*starrel* `` {*X*})

theorem *star-cases* [*case-names star-n*, *cases type*: *star*]:
(⋀*X*. *x* = *star-n X* ⟹ *P*) ⟹ *P*
by (*cases x*, *unfold star-n-def star-def*, *erule quotientE*, *fast*)

lemma *all-star-eq*: (∀ *x*. *P x*) = (∀ *X*. *P* (*star-n X*))
by (*auto*, *rule-tac x*=*x* **in** *star-cases*, *simp*)

lemma *ex-star-eq*: $(\exists\, x.\ P\ x) = (\exists\, X.\ P\ (\mathit{star\text{-}n}\ X))$
by (*auto*, *rule-tac x=x* **in** *star-cases*, *auto*)

Proving that *starrel* is an equivalence relation

lemma *starrel-iff* [*iff*]: $((X,Y) \in \mathit{starrel}) = (\{n.\ X\ n = Y\ n\} \in \mathcal{U})$
by (*simp add*: *starrel-def*)

lemma *equiv-starrel*: *equiv UNIV starrel*
proof (*rule equiv.intro*)
  show *reflexive starrel* **by** (*simp add*: *refl-def*)
  show *sym starrel* **by** (*simp add*: *sym-def eq-commute*)
  show *trans starrel* **by** (*auto intro*: *transI elim*!: *ultra*)
qed

lemmas *equiv-starrel-iff* =
  eq-equiv-class-iff [*OF equiv-starrel UNIV-I UNIV-I*]

lemma *starrel-in-star*: *starrel*``{*x*} ∈ *star*
by (*simp add*: *star-def quotientI*)

lemma *star-n-eq-iff*: $(\mathit{star\text{-}n}\ X = \mathit{star\text{-}n}\ Y) = (\{n.\ X\ n = Y\ n\} \in \mathcal{U})$
by (*simp add*: *star-n-def Abs-star-inject starrel-in-star equiv-starrel-iff*)

34.3 Transfer principle

This introduction rule starts each transfer proof.

lemma *transfer-start*:
  $P \equiv \{n.\ Q\} \in \mathcal{U} \Longrightarrow \mathit{Trueprop}\ P \equiv \mathit{Trueprop}\ Q$
by (*subgoal-tac* $P \equiv Q$, *simp*, *simp add*: *atomize-eq*)

Initialize transfer tactic.

use *transfer.ML*
setup *Transfer.setup*

Transfer introduction rules.

lemma *transfer-ex* [*transfer-intro*]:
  $[\![\bigwedge X.\ p\ (\mathit{star\text{-}n}\ X) \equiv \{n.\ P\ n\ (X\ n)\} \in \mathcal{U}]\!]$
    $\Longrightarrow \exists\, x{::}'a\ \mathit{star}.\ p\ x \equiv \{n.\ \exists\, x.\ P\ n\ x\} \in \mathcal{U}$
by (*simp only*: *ex-star-eq FreeUltrafilterNat.Collect-ex*)

lemma *transfer-all* [*transfer-intro*]:
  $[\![\bigwedge X.\ p\ (\mathit{star\text{-}n}\ X) \equiv \{n.\ P\ n\ (X\ n)\} \in \mathcal{U}]\!]$
    $\Longrightarrow \forall\, x{::}'a\ \mathit{star}.\ p\ x \equiv \{n.\ \forall\, x.\ P\ n\ x\} \in \mathcal{U}$
by (*simp only*: *all-star-eq FreeUltrafilterNat.Collect-all*)

lemma *transfer-not* [*transfer-intro*]:
  $[\![p \equiv \{n.\ P\ n\} \in \mathcal{U}]\!] \Longrightarrow \neg\ p \equiv \{n.\ \neg\ P\ n\} \in \mathcal{U}$

by (*simp only*: *FreeUltrafilterNat.Collect-not*)

lemma *transfer-conj* [*transfer-intro*]:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ q \equiv \{n.\ Q\ n\} \in \mathcal{U}]\!]$
$\Longrightarrow p \wedge q \equiv \{n.\ P\ n \wedge Q\ n\} \in \mathcal{U}$
by (*simp only*: *FreeUltrafilterNat.Collect-conj*)

lemma *transfer-disj* [*transfer-intro*]:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ q \equiv \{n.\ Q\ n\} \in \mathcal{U}]\!]$
$\Longrightarrow p \vee q \equiv \{n.\ P\ n \vee Q\ n\} \in \mathcal{U}$
by (*simp only*: *FreeUltrafilterNat.Collect-disj*)

lemma *transfer-imp* [*transfer-intro*]:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ q \equiv \{n.\ Q\ n\} \in \mathcal{U}]\!]$
$\Longrightarrow p \longrightarrow q \equiv \{n.\ P\ n \longrightarrow Q\ n\} \in \mathcal{U}$
by (*simp only*: *imp-conv-disj transfer-disj transfer-not*)

lemma *transfer-iff* [*transfer-intro*]:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ q \equiv \{n.\ Q\ n\} \in \mathcal{U}]\!]$
$\Longrightarrow p = q \equiv \{n.\ P\ n = Q\ n\} \in \mathcal{U}$
by (*simp only*: *iff-conv-conj-imp transfer-conj transfer-imp*)

lemma *transfer-if-bool* [*transfer-intro*]:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ x \equiv \{n.\ X\ n\} \in \mathcal{U};\ y \equiv \{n.\ Y\ n\} \in \mathcal{U}]\!]$
$\Longrightarrow$ (*if p then x else y*) $\equiv \{n.$ *if P n then X n else Y n*$\} \in \mathcal{U}$
by (*simp only*: *if-bool-eq-conj transfer-conj transfer-imp transfer-not*)

lemma *transfer-eq* [*transfer-intro*]:
$[\![x \equiv$ *star-n X*; $y \equiv$ *star-n Y*$]\!] \Longrightarrow x = y \equiv \{n.\ X\ n = Y\ n\} \in \mathcal{U}$
by (*simp only*: *star-n-eq-iff*)

lemma *transfer-if* [*transfer-intro*]:
$[\![p \equiv \{n.\ P\ n\} \in \mathcal{U};\ x \equiv$ *star-n X*; $y \equiv$ *star-n Y*$]\!]$
$\Longrightarrow$ (*if p then x else y*) $\equiv$ *star-n* $(\lambda n.$ *if P n then X n else Y n*$)$
apply (*rule eq-reflection*)
apply (*auto simp add*: *star-n-eq-iff transfer-not elim*!: *ultra*)
done

lemma *transfer-fun-eq* [*transfer-intro*]:
$[\![\bigwedge X.\ f\ ($*star-n X*$) = g\ ($*star-n X*$)$
$\equiv \{n.\ F\ n\ (X\ n) = G\ n\ (X\ n)\} \in \mathcal{U}]\!]$
$\Longrightarrow f = g \equiv \{n.\ F\ n = G\ n\} \in \mathcal{U}$
by (*simp only*: *expand-fun-eq transfer-all*)

lemma *transfer-star-n* [*transfer-intro*]: *star-n X* $\equiv$ *star-n* $(\lambda n.\ X\ n)$
by (*rule reflexive*)

lemma *transfer-bool* [*transfer-intro*]: $p \equiv \{n.\ p\} \in \mathcal{U}$
by (*simp add*: *atomize-eq*)

34.4 Standard elements

definition
star-of :: *'a* ⇒ *'a star* **where**
star-of x == *star-n* (λ*n*. *x*)

definition
Standard :: *'a star set* **where**
Standard = *range star-of*

Transfer tactic should remove occurrences of *star-of*

setup ⟪ *Transfer.add-const StarDef.star-of* ⟫

declare *star-of-def* [*transfer-intro*]

lemma *star-of-inject*: (*star-of x* = *star-of y*) = (*x* = *y*)
by (*transfer*, *rule refl*)

lemma *Standard-star-of* [*simp*]: *star-of x* ∈ *Standard*
by (*simp add*: *Standard-def*)

34.5 Internal functions

definition
Ifun :: (*'a* ⇒ *'b*) *star* ⇒ *'a star* ⇒ *'b star* (- ⋆ - [*300*,*301*] *300*) **where**
Ifun f ≡ λ*x*. *Abs-star*
(⋃ *F*∈*Rep-star f*. ⋃ *X*∈*Rep-star x*. *starrel*``{λ*n*. *F n* (*X n*)})

lemma *Ifun-congruent2*:
congruent2 starrel starrel (λ*F X*. *starrel*``{λ*n*. *F n* (*X n*)})
by (*auto simp add*: *congruent2-def equiv-starrel-iff elim*!: *ultra*)

lemma *Ifun-star-n*: *star-n F* ⋆ *star-n X* = *star-n* (λ*n*. *F n* (*X n*))
by (*simp add*: *Ifun-def star-n-def Abs-star-inverse starrel-in-star*
UN-equiv-class2 [*OF equiv-starrel equiv-starrel Ifun-congruent2*])

Transfer tactic should remove occurrences of *Ifun*

setup ⟪ *Transfer.add-const StarDef.Ifun* ⟫

lemma *transfer-Ifun* [*transfer-intro*]:
⟦*f* ≡ *star-n F*; *x* ≡ *star-n X*⟧ ⟹ *f* ⋆ *x* ≡ *star-n* (λ*n*. *F n* (*X n*))
by (*simp only*: *Ifun-star-n*)

lemma *Ifun-star-of* [*simp*]: *star-of f* ⋆ *star-of x* = *star-of* (*f x*)
by (*transfer*, *rule refl*)

lemma *Standard-Ifun* [*simp*]:
⟦*f* ∈ *Standard*; *x* ∈ *Standard*⟧ ⟹ *f* ⋆ *x* ∈ *Standard*
by (*auto simp add*: *Standard-def*)

Nonstandard extensions of functions

definition
starfun :: $('a \Rightarrow 'b) \Rightarrow ('a\ star \Rightarrow 'b\ star)$ $(*f* - [80]\ 80)$ **where**
starfun f == $\lambda x.$ *star-of f* $\star$ *x*

definition
starfun2 :: $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('a\ star \Rightarrow 'b\ star \Rightarrow 'c\ star)$
$(*f2* - [80]\ 80)$ **where**
starfun2 f == $\lambda x\ y.$ *star-of f* $\star$ *x* $\star$ *y*

declare *starfun-def* [*transfer-unfold*]
declare *starfun2-def* [*transfer-unfold*]

lemma *starfun-star-n*: $(*f* f)\ (star\text{-}n\ X) = star\text{-}n\ (\lambda n.\ f\ (X\ n))$
by (*simp only*: *starfun-def star-of-def Ifun-star-n*)

lemma *starfun2-star-n*:
$(*f2* f)\ (star\text{-}n\ X)\ (star\text{-}n\ Y) = star\text{-}n\ (\lambda n.\ f\ (X\ n)\ (Y\ n))$
by (*simp only*: *starfun2-def star-of-def Ifun-star-n*)

lemma *starfun-star-of* [*simp*]: $(*f* f)\ (star\text{-}of\ x) = star\text{-}of\ (f\ x)$
by (*transfer*, *rule refl*)

lemma *starfun2-star-of* [*simp*]: $(*f2* f)\ (star\text{-}of\ x) = *f* f\ x$
by (*transfer*, *rule refl*)

lemma *Standard-starfun* [*simp*]: $x \in Standard \Longrightarrow starfun\ f\ x \in Standard$
by (*simp add*: *starfun-def*)

lemma *Standard-starfun2* [*simp*]:
$[\![x \in Standard;\ y \in Standard]\!] \Longrightarrow starfun2\ f\ x\ y \in Standard$
by (*simp add*: *starfun2-def*)

lemma *Standard-starfun-iff*:
assumes *inj*: $\bigwedge x\ y.\ f\ x = f\ y \Longrightarrow x = y$
shows $(starfun\ f\ x \in Standard) = (x \in Standard)$
proof
assume $x \in Standard$
thus $starfun\ f\ x \in Standard$ **by** *simp*
next
have *inj'*: $\bigwedge x\ y.\ starfun\ f\ x = starfun\ f\ y \Longrightarrow x = y$
using *inj* **by** *transfer*
assume $starfun\ f\ x \in Standard$
then obtain *b* **where** *b*: $starfun\ f\ x = star\text{-}of\ b$
unfolding *Standard-def* **..**
hence $\exists x.\ starfun\ f\ x = star\text{-}of\ b$ **..**
hence $\exists a.\ f\ a = b$ **by** *transfer*
then obtain *a* **where** $f\ a = b$ **..**
hence $starfun\ f\ (star\text{-}of\ a) = star\text{-}of\ b$ **by** *transfer*

with *b* **have** *starfun f x = starfun f (star-of a)* **by** *simp*
hence *x = star-of a* **by** (*rule inj'*)
thus *x ∈ Standard*
unfolding *Standard-def* **by** *auto*
qed

lemma *Standard-starfun2-iff*:
assumes *inj*: ⋀*a b a' b'*. *f a b = f a' b' ⟹ a = a' ∧ b = b'*
shows (*starfun2 f x y ∈ Standard*) = (*x ∈ Standard ∧ y ∈ Standard*)
proof
assume *x ∈ Standard ∧ y ∈ Standard*
thus *starfun2 f x y ∈ Standard* **by** *simp*
next
have *inj'*: ⋀*x y z w*. *starfun2 f x y = starfun2 f z w ⟹ x = z ∧ y = w*
using *inj* **by** *transfer*
assume *starfun2 f x y ∈ Standard*
then obtain *c* **where** *c*: *starfun2 f x y = star-of c*
unfolding *Standard-def* **..**
hence ∃ *x y*. *starfun2 f x y = star-of c* **by** *auto*
hence ∃ *a b*. *f a b = c* **by** *transfer*
then obtain *a b* **where** *f a b = c* **by** *auto*
hence *starfun2 f (star-of a) (star-of b) = star-of c*
by *transfer*
with *c* **have** *starfun2 f x y = starfun2 f (star-of a) (star-of b)*
by *simp*
hence *x = star-of a ∧ y = star-of b*
by (*rule inj'*)
thus *x ∈ Standard ∧ y ∈ Standard*
unfolding *Standard-def* **by** *auto*
qed

34.6 Internal predicates

definition
unstar :: *bool star ⇒ bool* **where**
unstar b = (*b = star-of True*)

lemma *unstar-star-n*: *unstar* (*star-n P*) = ({*n*. *P n*} ∈ 𝒰)
by (*simp add*: *unstar-def star-of-def star-n-eq-iff*)

lemma *unstar-star-of* [*simp*]: *unstar* (*star-of p*) = *p*
by (*simp add*: *unstar-def star-of-inject*)

Transfer tactic should remove occurrences of *unstar*

setup ⟪ *Transfer.add-const StarDef.unstar* ⟫

lemma *transfer-unstar* [*transfer-intro*]:
p ≡ star-n P ⟹ unstar p ≡ {*n*. *P n*} ∈ 𝒰
by (*simp only*: *unstar-star-n*)

definition
starP :: (*'a* ⇒ *bool*) ⇒ *'a star* ⇒ *bool* (∗*p*∗ - [*80*] *80*) **where**
∗*p*∗ *P* = (λ*x*. *unstar* (*star-of P* ⋆ *x*))

definition
starP2 :: (*'a* ⇒ *'b* ⇒ *bool*) ⇒ *'a star* ⇒ *'b star* ⇒ *bool* (∗*p2*∗ - [*80*] *80*) **where**
∗*p2*∗ *P* = (λ*x y*. *unstar* (*star-of P* ⋆ *x* ⋆ *y*))

declare *starP-def* [*transfer-unfold*]
declare *starP2-def* [*transfer-unfold*]

lemma *starP-star-n*: (∗*p*∗ *P*) (*star-n X*) = ({*n*. *P* (*X n*)} ∈ *U*)
by (*simp only*: *starP-def star-of-def Ifun-star-n unstar-star-n*)

lemma *starP2-star-n*:
(∗*p2*∗ *P*) (*star-n X*) (*star-n Y*) = ({*n*. *P* (*X n*) (*Y n*)} ∈ *U*)
by (*simp only*: *starP2-def star-of-def Ifun-star-n unstar-star-n*)

lemma *starP-star-of* [*simp*]: (∗*p*∗ *P*) (*star-of x*) = *P x*
by (*transfer*, *rule refl*)

lemma *starP2-star-of* [*simp*]: (∗*p2*∗ *P*) (*star-of x*) = ∗*p*∗ *P x*
by (*transfer*, *rule refl*)

34.7 Internal sets

definition
Iset :: *'a set star* ⇒ *'a star set* **where**
Iset A = {*x*. (∗*p2*∗ *op* ∈) *x A*}

lemma *Iset-star-n*:
(*star-n X* ∈ *Iset* (*star-n A*)) = ({*n*. *X n* ∈ *A n*} ∈ *U*)
by (*simp add*: *Iset-def starP2-star-n*)

Transfer tactic should remove occurrences of *Iset*

setup ⟪ *Transfer.add-const StarDef.Iset* ⟫

lemma *transfer-mem* [*transfer-intro*]:
⟦*x* ≡ *star-n X*; *a* ≡ *Iset* (*star-n A*)⟧
⟹ *x* ∈ *a* ≡ {*n*. *X n* ∈ *A n*} ∈ *U*
by (*simp only*: *Iset-star-n*)

lemma *transfer-Collect* [*transfer-intro*]:
⟦⋀*X*. *p* (*star-n X*) ≡ {*n*. *P n* (*X n*)} ∈ *U*⟧
⟹ *Collect p* ≡ *Iset* (*star-n* (λ*n*. *Collect* (*P n*)))
by (*simp add*: *atomize-eq expand-set-eq all-star-eq Iset-star-n*)

lemma *transfer-set-eq* [*transfer-intro*]:

$[\![a \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ A);\ b \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ B)]\!]$
$\Longrightarrow a = b \equiv \{n.\ A\ n = B\ n\} \in \mathcal{U}$
by (*simp only*: *expand-set-eq transfer-all transfer-iff transfer-mem*)

lemma *transfer-ball* [*transfer-intro*]:
$[\![a \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ A);\ \bigwedge X.\ p\ (\mathit{star\text{-}n}\ X) \equiv \{n.\ P\ n\ (X\ n)\} \in \mathcal{U}]\!]$
$\Longrightarrow \forall x{\in}a.\ p\ x \equiv \{n.\ \forall x{\in}A\ n.\ P\ n\ x\} \in \mathcal{U}$
by (*simp only*: *Ball-def transfer-all transfer-imp transfer-mem*)

lemma *transfer-bex* [*transfer-intro*]:
$[\![a \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ A);\ \bigwedge X.\ p\ (\mathit{star\text{-}n}\ X) \equiv \{n.\ P\ n\ (X\ n)\} \in \mathcal{U}]\!]$
$\Longrightarrow \exists x{\in}a.\ p\ x \equiv \{n.\ \exists x{\in}A\ n.\ P\ n\ x\} \in \mathcal{U}$
by (*simp only*: *Bex-def transfer-ex transfer-conj transfer-mem*)

lemma *transfer-Iset* [*transfer-intro*]:
$[\![a \equiv \mathit{star\text{-}n}\ A]\!] \Longrightarrow \mathit{Iset}\ a \equiv \mathit{Iset}\ (\mathit{star\text{-}n}\ (\lambda n.\ A\ n))$
by *simp*

Nonstandard extensions of sets.

definition
$\mathit{starset} :: {'a}\ \mathit{set} \Rightarrow {'a}\ \mathit{star\ set}\ (*s* \text{-}\ [80]\ 80)$ **where**
$\mathit{starset}\ A = \mathit{Iset}\ (\mathit{star\text{-}of}\ A)$

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

lemma *starset-mem*: $(\mathit{star\text{-}of}\ x \in *s*\ A) = (x \in A)$
by (*transfer*, *rule refl*)

lemma *starset-UNIV*: $*s*\ (\mathit{UNIV}::{'a}\ \mathit{set}) = (\mathit{UNIV}::{'a}\ \mathit{star\ set})$
by (*transfer UNIV-def*, *rule refl*)

lemma *starset-empty*: $*s*\ \{\} = \{\}$
by (*transfer empty-def*, *rule refl*)

lemma *starset-insert*: $*s*\ (\mathit{insert}\ x\ A) = \mathit{insert}\ (\mathit{star\text{-}of}\ x)\ (*s*\ A)$
by (*transfer insert-def Un-def*, *rule refl*)

lemma *starset-Un*: $*s*\ (A \cup B) = *s*\ A \cup *s*\ B$
by (*transfer Un-def*, *rule refl*)

lemma *starset-Int*: $*s*\ (A \cap B) = *s*\ A \cap *s*\ B$
by (*transfer Int-def*, *rule refl*)

lemma *starset-Compl*: $*s*\ -A = -(*s*\ A)$
by (*transfer Compl-eq*, *rule refl*)

lemma *starset-diff*: $*s*\ (A - B) = *s*\ A - *s*\ B$
by (*transfer set-diff-eq*, *rule refl*)

lemma *starset-image*: $*s*\ (f\ `\ A) = (\ *f*\ f)\ `\ (\ *s*\ A)$
by (*transfer image-def*, *rule refl*)

lemma *starset-vimage*: $*s*\ (f\ -`\ A) = (\ *f*\ f)\ -`\ (\ *s*\ A)$
by (*transfer vimage-def*, *rule refl*)

lemma *starset-subset*: $(\ *s*\ A \subseteq *s*\ B) = (A \subseteq B)$
by (*transfer subset-eq*, *rule refl*)

lemma *starset-eq*: $(\ *s*\ A = *s*\ B) = (A = B)$
by (*transfer*, *rule refl*)

lemmas *starset-simps* [*simp*] =
starset-mem *starset-UNIV*
starset-empty *starset-insert*
starset-Un *starset-Int*
starset-Compl *starset-diff*
starset-image *starset-vimage*
starset-subset *starset-eq*

34.8 Syntactic classes

instantiation *star* :: (*zero*) *zero*
begin

definition
star-zero-def: $0 \equiv$ *star-of 0*

instance ..

end

instantiation *star* :: (*one*) *one*
begin

definition
star-one-def: $1 \equiv$ *star-of 1*

instance ..

end

instantiation *star* :: (*plus*) *plus*
begin

definition
star-add-def: $(op\ +) \equiv *f2*\ (op\ +)$

instance ..

end

instantiation *star* :: (*times*) *times*
begin

definition
star-mult-def: $(op\ *) \equiv *f2*\ (op\ *)$

instance ..

end

instantiation *star* :: (*uminus*) *uminus*
begin

definition
star-minus-def: $uminus \equiv *f*\ uminus$

instance ..

end

instantiation *star* :: (*minus*) *minus*
begin

definition
star-diff-def: $(op\ -) \equiv *f2*\ (op\ -)$

instance ..

end

instantiation *star* :: (*abs*) *abs*
begin

definition
star-abs-def: $abs \equiv *f*\ abs$

instance ..

end

instantiation *star* :: (*sgn*) *sgn*
begin

definition
star-sgn-def: $sgn \equiv *f*\ sgn$

instance ..

end

instantiation *star* :: (*inverse*) *inverse*
begin

definition
star-divide-def: $(op\ /) \equiv *f2*\ (op\ /)$

definition
star-inverse-def: $inverse \equiv *f*\ inverse$

instance ..

end

instantiation *star* :: (*number*) *number*
begin

definition
star-number-def: $number\text{-}of\ b \equiv star\text{-}of\ (number\text{-}of\ b)$

instance ..

end

instantiation *star* :: (*Divides.div*) *Divides.div*
begin

definition
star-div-def: $(op\ div) \equiv *f2*\ (op\ div)$

definition
star-mod-def: $(op\ mod) \equiv *f2*\ (op\ mod)$

instance ..

end

instantiation *star* :: (*power*) *power*
begin

definition
star-power-def: $(op\ \hat{}) \equiv \lambda x\ n.\ (\ *f*\ (\lambda x.\ x\ \hat{}\ n))\ x$

instance ..

end

instantiation *star* :: (*ord*) *ord*
begin

definition
star-le-def: $(op \leq) \equiv *p2* (op \leq)$

definition
star-less-def: $(op <) \equiv *p2* (op <)$

instance ..

end

lemmas *star-class-defs* [*transfer-unfold*] =
star-zero-def *star-one-def* *star-number-def*
star-add-def *star-diff-def* *star-minus-def*
star-mult-def *star-divide-def* *star-inverse-def*
star-le-def *star-less-def* *star-abs-def* *star-sgn-def*
star-div-def *star-mod-def* *star-power-def*

Class operations preserve standard elements

lemma *Standard-zero*: $0 \in Standard$
by (*simp add*: *star-zero-def*)

lemma *Standard-one*: $1 \in Standard$
by (*simp add*: *star-one-def*)

lemma *Standard-number-of*: $number\text{-}of\ b \in Standard$
by (*simp add*: *star-number-def*)

lemma *Standard-add*: $[\![x \in Standard; y \in Standard]\!] \Longrightarrow x + y \in Standard$
by (*simp add*: *star-add-def*)

lemma *Standard-diff*: $[\![x \in Standard; y \in Standard]\!] \Longrightarrow x - y \in Standard$
by (*simp add*: *star-diff-def*)

lemma *Standard-minus*: $x \in Standard \Longrightarrow - x \in Standard$
by (*simp add*: *star-minus-def*)

lemma *Standard-mult*: $[\![x \in Standard; y \in Standard]\!] \Longrightarrow x * y \in Standard$
by (*simp add*: *star-mult-def*)

lemma *Standard-divide*: $[\![x \in Standard; y \in Standard]\!] \Longrightarrow x\ /\ y \in Standard$
by (*simp add*: *star-divide-def*)

lemma *Standard-inverse*: $x \in Standard \Longrightarrow inverse\ x \in Standard$
by (*simp add*: *star-inverse-def*)

lemma *Standard-abs*: $x \in Standard \Longrightarrow abs\ x \in Standard$
by (*simp add*: *star-abs-def*)

lemma *Standard-div*: $[\![x \in Standard;\ y \in Standard]\!] \Longrightarrow x\ div\ y \in Standard$
by (*simp add*: *star-div-def*)

lemma *Standard-mod*: $[\![x \in Standard;\ y \in Standard]\!] \Longrightarrow x\ mod\ y \in Standard$
by (*simp add*: *star-mod-def*)

lemma *Standard-power*: $x \in Standard \Longrightarrow x\ \hat{}\ n \in Standard$
by (*simp add*: *star-power-def*)

lemmas *Standard-simps* [*simp*] =
Standard-zero *Standard-one* *Standard-number-of*
Standard-add *Standard-diff* *Standard-minus*
Standard-mult *Standard-divide* *Standard-inverse*
Standard-abs *Standard-div* *Standard-mod*
Standard-power

star-of preserves class operations

lemma *star-of-add*: $star\text{-}of\ (x + y) = star\text{-}of\ x + star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-diff*: $star\text{-}of\ (x - y) = star\text{-}of\ x - star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-minus*: $star\text{-}of\ (-x) = -\ star\text{-}of\ x$
by *transfer* (*rule refl*)

lemma *star-of-mult*: $star\text{-}of\ (x * y) = star\text{-}of\ x * star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-divide*: $star\text{-}of\ (x\ /\ y) = star\text{-}of\ x\ /\ star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-inverse*: $star\text{-}of\ (inverse\ x) = inverse\ (star\text{-}of\ x)$
by *transfer* (*rule refl*)

lemma *star-of-div*: $star\text{-}of\ (x\ div\ y) = star\text{-}of\ x\ div\ star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-mod*: $star\text{-}of\ (x\ mod\ y) = star\text{-}of\ x\ mod\ star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-power*: $star\text{-}of\ (x\ \hat{}\ n) = star\text{-}of\ x\ \hat{}\ n$
by *transfer* (*rule refl*)

lemma *star-of-abs*: $star\text{-}of\ (abs\ x) = abs\ (star\text{-}of\ x)$
by *transfer* (*rule refl*)

star-of preserves numerals

lemma *star-of-zero*: *star-of 0 = 0*
by *transfer* (*rule refl*)

lemma *star-of-one*: *star-of 1 = 1*
by *transfer* (*rule refl*)

lemma *star-of-number-of*: *star-of* (*number-of x*) = *number-of x*
by *transfer* (*rule refl*)

star-of preserves orderings

lemma *star-of-less*: (*star-of x* < *star-of y*) = (*x* < *y*)
by *transfer* (*rule refl*)

lemma *star-of-le*: (*star-of x* ≤ *star-of y*) = (*x* ≤ *y*)
by *transfer* (*rule refl*)

lemma *star-of-eq*: (*star-of x* = *star-of y*) = (*x* = *y*)
by *transfer* (*rule refl*)

As above, for 0

lemmas *star-of-0-less* = *star-of-less* [*of 0*, *simplified star-of-zero*]
lemmas *star-of-0-le* = *star-of-le* [*of 0*, *simplified star-of-zero*]
lemmas *star-of-0-eq* = *star-of-eq* [*of 0*, *simplified star-of-zero*]

lemmas *star-of-less-0* = *star-of-less* [*of - 0*, *simplified star-of-zero*]
lemmas *star-of-le-0* = *star-of-le* [*of - 0*, *simplified star-of-zero*]
lemmas *star-of-eq-0* = *star-of-eq* [*of - 0*, *simplified star-of-zero*]

As above, for 1

lemmas *star-of-1-less* = *star-of-less* [*of 1*, *simplified star-of-one*]
lemmas *star-of-1-le* = *star-of-le* [*of 1*, *simplified star-of-one*]
lemmas *star-of-1-eq* = *star-of-eq* [*of 1*, *simplified star-of-one*]

lemmas *star-of-less-1* = *star-of-less* [*of - 1*, *simplified star-of-one*]
lemmas *star-of-le-1* = *star-of-le* [*of - 1*, *simplified star-of-one*]
lemmas *star-of-eq-1* = *star-of-eq* [*of - 1*, *simplified star-of-one*]

As above, for numerals

lemmas *star-of-number-less* =
star-of-less [*of number-of w*, *standard*, *simplified star-of-number-of*]
lemmas *star-of-number-le* =
star-of-le [*of number-of w*, *standard*, *simplified star-of-number-of*]
lemmas *star-of-number-eq* =
star-of-eq [*of number-of w*, *standard*, *simplified star-of-number-of*]

lemmas *star-of-less-number* =
star-of-less [*of - number-of w*, *standard*, *simplified star-of-number-of*]

lemmas *star-of-le-number* =
star-of-le [*of* - *number-of w*, *standard*, *simplified star-of-number-of*]
lemmas *star-of-eq-number* =
star-of-eq [*of* - *number-of w*, *standard*, *simplified star-of-number-of*]

lemmas *star-of-simps* [*simp*] =
star-of-add *star-of-diff* *star-of-minus*
star-of-mult *star-of-divide* *star-of-inverse*
star-of-div *star-of-mod*
star-of-power *star-of-abs*
star-of-zero *star-of-one* *star-of-number-of*
star-of-less *star-of-le* *star-of-eq*
star-of-0-less *star-of-0-le* *star-of-0-eq*
star-of-less-0 *star-of-le-0* *star-of-eq-0*
star-of-1-less *star-of-1-le* *star-of-1-eq*
star-of-less-1 *star-of-le-1* *star-of-eq-1*
star-of-number-less *star-of-number-le* *star-of-number-eq*
star-of-less-number *star-of-le-number* *star-of-eq-number*

34.9 Ordering and lattice classes

instance *star* :: (*order*) *order*
apply (*intro-classes*)
apply (*transfer*, *rule order-less-le*)
apply (*transfer*, *rule order-refl*)
apply (*transfer*, *erule* (*1*) *order-trans*)
apply (*transfer*, *erule* (*1*) *order-antisym*)
done

instantiation *star* :: (*lower-semilattice*) *lower-semilattice*
begin

definition
star-inf-def [*transfer-unfold*]: *inf* ≡ *∗f2∗ inf*

instance
by *default* (*transfer star-inf-def*, *auto*)+

end

instantiation *star* :: (*upper-semilattice*) *upper-semilattice*
begin

definition
star-sup-def [*transfer-unfold*]: *sup* ≡ *∗f2∗ sup*

instance
by *default* (*transfer star-sup-def*, *auto*)+

end

instance *star* :: (*lattice*) *lattice* **..**

instance *star* :: (*distrib-lattice*) *distrib-lattice*
by *default* (*transfer*, *auto simp add*: *sup-inf-distrib1*)

lemma *Standard-inf* [*simp*]:
⟦*x* ∈ *Standard*; *y* ∈ *Standard*⟧ ⟹ *inf x y* ∈ *Standard*
by (*simp add*: *star-inf-def*)

lemma *Standard-sup* [*simp*]:
⟦*x* ∈ *Standard*; *y* ∈ *Standard*⟧ ⟹ *sup x y* ∈ *Standard*
by (*simp add*: *star-sup-def*)

lemma *star-of-inf* [*simp*]: *star-of* (*inf x y*) = *inf* (*star-of x*) (*star-of y*)
by *transfer* (*rule refl*)

lemma *star-of-sup* [*simp*]: *star-of* (*sup x y*) = *sup* (*star-of x*) (*star-of y*)
by *transfer* (*rule refl*)

instance *star* :: (*linorder*) *linorder*
by (*intro-classes*, *transfer*, *rule linorder-linear*)

lemma *star-max-def* [*transfer-unfold*]: *max* = ∗*f2*∗ *max*
apply (*rule ext*, *rule ext*)
apply (*unfold max-def*, *transfer*, *fold max-def*)
apply (*rule refl*)
done

lemma *star-min-def* [*transfer-unfold*]: *min* = ∗*f2*∗ *min*
apply (*rule ext*, *rule ext*)
apply (*unfold min-def*, *transfer*, *fold min-def*)
apply (*rule refl*)
done

lemma *Standard-max* [*simp*]:
⟦*x* ∈ *Standard*; *y* ∈ *Standard*⟧ ⟹ *max x y* ∈ *Standard*
by (*simp add*: *star-max-def*)

lemma *Standard-min* [*simp*]:
⟦*x* ∈ *Standard*; *y* ∈ *Standard*⟧ ⟹ *min x y* ∈ *Standard*
by (*simp add*: *star-min-def*)

lemma *star-of-max* [*simp*]: *star-of* (*max x y*) = *max* (*star-of x*) (*star-of y*)
by *transfer* (*rule refl*)

lemma *star-of-min* [*simp*]: *star-of* (*min x y*) = *min* (*star-of x*) (*star-of y*)
by *transfer* (*rule refl*)

34.10 Ordered group classes

instance *star* :: (*semigroup-add*) *semigroup-add*
by (*intro-classes*, *transfer*, *rule add-assoc*)

instance *star* :: (*ab-semigroup-add*) *ab-semigroup-add*
by (*intro-classes*, *transfer*, *rule add-commute*)

instance *star* :: (*semigroup-mult*) *semigroup-mult*
by (*intro-classes*, *transfer*, *rule mult-assoc*)

instance *star* :: (*ab-semigroup-mult*) *ab-semigroup-mult*
by (*intro-classes*, *transfer*, *rule mult-commute*)

instance *star* :: (*comm-monoid-add*) *comm-monoid-add*
by (*intro-classes*, *transfer*, *rule comm-monoid-add-class.zero-plus.add-0*)

instance *star* :: (*monoid-mult*) *monoid-mult*
apply (*intro-classes*)
apply (*transfer*, *rule mult-1-left*)
apply (*transfer*, *rule mult-1-right*)
done

instance *star* :: (*comm-monoid-mult*) *comm-monoid-mult*
by (*intro-classes*, *transfer*, *rule mult-1*)

instance *star* :: (*cancel-semigroup-add*) *cancel-semigroup-add*
apply (*intro-classes*)
apply (*transfer*, *erule add-left-imp-eq*)
apply (*transfer*, *erule add-right-imp-eq*)
done

instance *star* :: (*cancel-ab-semigroup-add*) *cancel-ab-semigroup-add*
by (*intro-classes*, *transfer*, *rule add-imp-eq*)

instance *star* :: (*ab-group-add*) *ab-group-add*
apply (*intro-classes*)
apply (*transfer*, *rule left-minus*)
apply (*transfer*, *rule diff-minus*)
done

instance *star* :: (*pordered-ab-semigroup-add*) *pordered-ab-semigroup-add*
by (*intro-classes*, *transfer*, *rule add-left-mono*)

instance *star* :: (*pordered-cancel-ab-semigroup-add*) *pordered-cancel-ab-semigroup-add*
..

instance *star* :: (*pordered-ab-semigroup-add-imp-le*) *pordered-ab-semigroup-add-imp-le*
by (*intro-classes*, *transfer*, *rule add-le-imp-le-left*)

instance *star* :: (*pordered-comm-monoid-add*) *pordered-comm-monoid-add* **..**
instance *star* :: (*pordered-ab-group-add*) *pordered-ab-group-add* **..**

instance *star* :: (*pordered-ab-group-add-abs*) *pordered-ab-group-add-abs*
by *intro-classes* (*transfer*,
simp add: *abs-ge-self abs-leI abs-triangle-ineq*)+

instance *star* :: (*ordered-cancel-ab-semigroup-add*) *ordered-cancel-ab-semigroup-add*
..
instance *star* :: (*lordered-ab-group-add-meet*) *lordered-ab-group-add-meet* **..**
instance *star* :: (*lordered-ab-group-add-meet*) *lordered-ab-group-add-meet* **..**
instance *star* :: (*lordered-ab-group-add*) *lordered-ab-group-add* **..**

instance *star* :: (*lordered-ab-group-add-abs*) *lordered-ab-group-add-abs*
by (*intro-classes*, *transfer*, *rule abs-lattice*)

34.11 Ring and field classes

instance *star* :: (*semiring*) *semiring*
apply (*intro-classes*)
apply (*transfer*, *rule left-distrib*)
apply (*transfer*, *rule right-distrib*)
done

instance *star* :: (*semiring-0*) *semiring-0*
by *intro-classes* (*transfer*, *simp*)+

instance *star* :: (*semiring-0-cancel*) *semiring-0-cancel* **..**

instance *star* :: (*comm-semiring*) *comm-semiring*
by (*intro-classes*, *transfer*, *rule left-distrib*)

instance *star* :: (*comm-semiring-0*) *comm-semiring-0* **..**
instance *star* :: (*comm-semiring-0-cancel*) *comm-semiring-0-cancel* **..**

instance *star* :: (*zero-neq-one*) *zero-neq-one*
by (*intro-classes*, *transfer*, *rule zero-neq-one*)

instance *star* :: (*semiring-1*) *semiring-1* **..**
instance *star* :: (*comm-semiring-1*) *comm-semiring-1* **..**

instance *star* :: (*no-zero-divisors*) *no-zero-divisors*
by (*intro-classes*, *transfer*, *rule no-zero-divisors*)

instance *star* :: (*semiring-1-cancel*) *semiring-1-cancel* **..**
instance *star* :: (*comm-semiring-1-cancel*) *comm-semiring-1-cancel* **..**
instance *star* :: (*ring*) *ring* **..**
instance *star* :: (*comm-ring*) *comm-ring* **..**
instance *star* :: (*ring-1*) *ring-1* **..**

instance *star* :: (*comm-ring-1*) *comm-ring-1* **..**
instance *star* :: (*ring-no-zero-divisors*) *ring-no-zero-divisors* **..**
instance *star* :: (*ring-1-no-zero-divisors*) *ring-1-no-zero-divisors* **..**
instance *star* :: (*idom*) *idom* **..**

instance *star* :: (*division-ring*) *division-ring*
apply (*intro-classes*)
apply (*transfer*, *erule left-inverse*)
apply (*transfer*, *erule right-inverse*)
done

instance *star* :: (*field*) *field*
apply (*intro-classes*)
apply (*transfer*, *erule left-inverse*)
apply (*transfer*, *rule divide-inverse*)
done

instance *star* :: (*division-by-zero*) *division-by-zero*
by (*intro-classes*, *transfer*, *rule inverse-zero*)

instance *star* :: (*pordered-semiring*) *pordered-semiring*
apply (*intro-classes*)
apply (*transfer*, *erule* (*1*) *mult-left-mono*)
apply (*transfer*, *erule* (*1*) *mult-right-mono*)
done

instance *star* :: (*pordered-cancel-semiring*) *pordered-cancel-semiring* **..**

instance *star* :: (*ordered-semiring-strict*) *ordered-semiring-strict*
apply (*intro-classes*)
apply (*transfer*, *erule* (*1*) *mult-strict-left-mono*)
apply (*transfer*, *erule* (*1*) *mult-strict-right-mono*)
done

instance *star* :: (*pordered-comm-semiring*) *pordered-comm-semiring*
by (*intro-classes*, *transfer*, *rule mult-mono1-class.less-eq-less-times-zero.mult-mono1*)

instance *star* :: (*pordered-cancel-comm-semiring*) *pordered-cancel-comm-semiring*
..

instance *star* :: (*ordered-comm-semiring-strict*) *ordered-comm-semiring-strict*
by (*intro-classes*, *transfer*, *rule ordered-comm-semiring-strict-class.plus-less-eq-less-zero-times.mult-strict-left-

instance *star* :: (*pordered-ring*) *pordered-ring* **..**
instance *star* :: (*pordered-ring-abs*) *pordered-ring-abs*
by *intro-classes* (*transfer*, *rule abs-eq-mult*)
instance *star* :: (*lordered-ring*) *lordered-ring* **..**

instance *star* :: (*abs-if*) *abs-if*

by (*intro-classes*, *transfer*, *rule abs-if*)

instance *star* :: (*sgn-if*) *sgn-if*
by (*intro-classes*, *transfer*, *rule sgn-if*)

instance *star* :: (*ordered-ring-strict*) *ordered-ring-strict* **..**
instance *star* :: (*pordered-comm-ring*) *pordered-comm-ring* **..**

instance *star* :: (*ordered-semidom*) *ordered-semidom*
by (*intro-classes*, *transfer*, *rule zero-less-one*)

instance *star* :: (*ordered-idom*) *ordered-idom* **..**
instance *star* :: (*ordered-field*) *ordered-field* **..**

34.12 Power classes

Proving the class axiom *power-Suc* for type *'a star* is a little tricky, because it quantifies over values of type *nat*. The transfer principle does not handle quantification over non-star types in general, but we can work around this by fixing an arbitrary *nat* value, and then applying the transfer principle.

instance *star* :: (*recpower*) *recpower*
proof
show $\bigwedge a::'a\ star.\ a\ \hat{}\ 0 = 1$
by *transfer* (*rule power-0*)
next
fix *n* **show** $\bigwedge a::'a\ star.\ a\ \hat{}\ Suc\ n = a * a\ \hat{}\ n$
by *transfer* (*rule power-Suc*)
qed

34.13 Number classes

lemma *star-of-nat-def* [*transfer-unfold*]: *of-nat n* = *star-of* (*of-nat n*)
by (*induct n*, *simp-all*)

lemma *Standard-of-nat* [*simp*]: *of-nat n* ∈ *Standard*
by (*simp add*: *star-of-nat-def*)

lemma *star-of-of-nat* [*simp*]: *star-of* (*of-nat n*) = *of-nat n*
by *transfer* (*rule refl*)

lemma *star-of-int-def* [*transfer-unfold*]: *of-int z* = *star-of* (*of-int z*)
by (*rule-tac z*=*z* **in** *int-diff-cases*, *simp*)

lemma *Standard-of-int* [*simp*]: *of-int z* ∈ *Standard*
by (*simp add*: *star-of-int-def*)

lemma *star-of-of-int* [*simp*]: *star-of* (*of-int z*) = *of-int z*
by *transfer* (*rule refl*)

instance *star* :: (*semiring-char-0*) *semiring-char-0*
by *intro-classes* (*simp only*: *star-of-nat-def star-of-eq of-nat-eq-iff*)

instance *star* :: (*ring-char-0*) *ring-char-0* **..**

instance *star* :: (*number-ring*) *number-ring*
by (*intro-classes*, *simp only*: *star-number-def star-of-int-def number-of-eq*)

34.14 Finite class

lemma *starset-finite*: *finite* $A \Longrightarrow *s* A = star\text{-}of\ `\ A$
by (*erule finite-induct*, *simp-all*)

instance *star* :: (*finite*) *finite*
apply (*intro-classes*)
apply (*subst starset-UNIV* [*symmetric*])
apply (*subst starset-finite* [*OF finite*])
apply (*rule finite-imageI* [*OF finite*])
done

end

35 HyperNat: Hypernatural numbers

theory *HyperNat*
imports *StarDef*
begin

types *hypnat* = *nat star*

abbreviation
  hypnat-of-nat :: *nat* => *nat star* **where**
  hypnat-of-nat == *star-of*

definition
  hSuc :: *hypnat* => *hypnat* **where**
  hSuc-def [*transfer-unfold*]: *hSuc* = $*f*$ *Suc*

35.1 Properties Transferred from Naturals

lemma *hSuc-not-zero* [*iff*]: $\bigwedge m.\ hSuc\ m \neq 0$
by *transfer* (*rule Suc-not-Zero*)

lemma *zero-not-hSuc* [*iff*]: $\bigwedge m.\ 0 \neq hSuc\ m$
by *transfer* (*rule Zero-not-Suc*)

lemma *hSuc-hSuc-eq* [*iff*]: $\bigwedge m\ n.\ (hSuc\ m = hSuc\ n) = (m = n)$

by *transfer* (*rule Suc-Suc-eq*)

lemma *zero-less-hSuc* [*iff*]: $\bigwedge n.\ 0 < hSuc\ n$
by *transfer* (*rule zero-less-Suc*)

lemma *hypnat-minus-zero* [*simp*]: $!!z.\ z - z = (0::hypnat)$
by *transfer* (*rule diff-self-eq-0*)

lemma *hypnat-diff-0-eq-0* [*simp*]: $!!n.\ (0::hypnat) - n = 0$
by *transfer* (*rule diff-0-eq-0*)

lemma *hypnat-add-is-0* [*iff*]: $!!m\ n.\ (m+n = (0::hypnat)) = (m=0\ \&\ n=0)$
by *transfer* (*rule add-is-0*)

lemma *hypnat-diff-diff-left*: $!!i\ j\ k.\ (i::hypnat) - j - k = i - (j+k)$
by *transfer* (*rule diff-diff-left*)

lemma *hypnat-diff-commute*: $!!i\ j\ k.\ (i::hypnat) - j - k = i-k-j$
by *transfer* (*rule diff-commute*)

lemma *hypnat-diff-add-inverse* [*simp*]: $!!m\ n.\ ((n::hypnat) + m) - n = m$
by *transfer* (*rule diff-add-inverse*)

lemma *hypnat-diff-add-inverse2* [*simp*]: $!!m\ n.\ ((m::hypnat) + n) - n = m$
by *transfer* (*rule diff-add-inverse2*)

lemma *hypnat-diff-cancel* [*simp*]: $!!k\ m\ n.\ ((k::hypnat) + m) - (k+n) = m - n$
by *transfer* (*rule diff-cancel*)

lemma *hypnat-diff-cancel2* [*simp*]: $!!k\ m\ n.\ ((m::hypnat) + k) - (n+k) = m - n$
by *transfer* (*rule diff-cancel2*)

lemma *hypnat-diff-add-0* [*simp*]: $!!m\ n.\ (n::hypnat) - (n+m) = (0::hypnat)$
by *transfer* (*rule diff-add-0*)

lemma *hypnat-diff-mult-distrib*: $!!k\ m\ n.\ ((m::hypnat) - n) * k = (m * k) - (n * k)$
by *transfer* (*rule diff-mult-distrib*)

lemma *hypnat-diff-mult-distrib2*: $!!k\ m\ n.\ (k::hypnat) * (m - n) = (k * m) - (k * n)$
by *transfer* (*rule diff-mult-distrib2*)

lemma *hypnat-le-zero-cancel* [*iff*]: $!!n.\ (n \leq (0::hypnat)) = (n = 0)$
by *transfer* (*rule le-0-eq*)

lemma *hypnat-mult-is-0* [*simp*]: $!!m\ n.\ (m*n = (0::hypnat)) = (m=0 \mid n=0)$
by *transfer* (*rule mult-is-0*)

lemma *hypnat-diff-is-0-eq* [*simp*]: !!*m n*. ((*m*::*hypnat*) − *n* = *0*) = (*m* ≤ *n*)
by *transfer* (*rule diff-is-0-eq*)

lemma *hypnat-not-less0* [*iff*]: !!*n*. ~ *n* < (*0*::*hypnat*)
by *transfer* (*rule not-less0*)

lemma *hypnat-less-one* [*iff*]:
  !!*n*. (*n* < (*1*::*hypnat*)) = (*n*=*0*)
by *transfer* (*rule less-one*)

lemma *hypnat-add-diff-inverse*: !!*m n*. ~ *m*<*n* ==> *n*+(*m*−*n*) = (*m*::*hypnat*)
by *transfer* (*rule add-diff-inverse*)

lemma *hypnat-le-add-diff-inverse* [*simp*]: !!*m n*. *n* ≤ *m* ==> *n*+(*m*−*n*) = (*m*::*hypnat*)
by *transfer* (*rule le-add-diff-inverse*)

lemma *hypnat-le-add-diff-inverse2* [*simp*]: !!*m n*. *n*≤*m* ==> (*m*−*n*)+*n* = (*m*::*hypnat*)
by *transfer* (*rule le-add-diff-inverse2*)

declare *hypnat-le-add-diff-inverse2* [*OF order-less-imp-le*]

lemma *hypnat-le0* [*iff*]: !!*n*. (*0*::*hypnat*) ≤ *n*
by *transfer* (*rule le0*)

lemma *hypnat-le-add1* [*simp*]: !!*x n*. (*x*::*hypnat*) ≤ *x* + *n*
by *transfer* (*rule le-add1*)

lemma *hypnat-add-self-le* [*simp*]: !!*x n*. (*x*::*hypnat*) ≤ *n* + *x*
by *transfer* (*rule le-add2*)

lemma *hypnat-add-one-self-less* [*simp*]: (*x*::*hypnat*) < *x* + (*1*::*hypnat*)
by (*insert add-strict-left-mono* [*OF zero-less-one*], *auto*)

lemma *hypnat-neq0-conv* [*iff*]: !!*n*. (*n* ≠ *0*) = (*0* < (*n*::*hypnat*))
by *transfer* (*rule neq0-conv*)

lemma *hypnat-gt-zero-iff*: ((*0*::*hypnat*) < *n*) = ((*1*::*hypnat*) ≤ *n*)
by (*auto simp add*: *linorder-not-less* [*symmetric*])

lemma *hypnat-gt-zero-iff2*: (*0* < *n*) = (∃ *m*. *n* = *m* + (*1*::*hypnat*))
apply *safe*
 apply (*rule-tac x* = *n* − (*1*::*hypnat*) **in** *exI*)
 apply (*simp add*: *hypnat-gt-zero-iff*)
apply (*insert add-le-less-mono* [*OF - zero-less-one*, *of 0*], *auto*)
done

lemma *hypnat-add-self-not-less*: ~ (*x* + *y* < (*x*::*hypnat*))
by (*simp add*: *linorder-not-le* [*symmetric*] *add-commute* [*of x*])

lemma *hypnat-diff-split*:
  $P(a - b::hypnat) = ((a<b \longrightarrow P\ 0) \ \&\ (ALL\ d.\ a = b + d \longrightarrow P\ d))$
  — elimination of $-$ on *hypnat*
proof (*cases a<b rule*: *case-split*)
 case *True*
  thus *?thesis*
   by (*auto simp add*: *hypnat-add-self-not-less order-less-imp-le*
                hypnat-diff-is-0-eq [*THEN iffD2*])
next
 case *False*
  thus *?thesis*
   by (*auto simp add*: *linorder-not-less dest*: *order-le-less-trans*)
qed

35.2 Properties of the set of embedded natural numbers

lemma *of-nat-eq-star-of* [*simp*]: *of-nat* = *star-of*
proof
 fix *n* :: *nat*
 show *of-nat n* = *star-of n* **by** *transfer simp*
qed

lemma *Nats-eq-Standard*: (*Nats* :: *nat star set*) = *Standard*
by (*auto simp add*: *Nats-def Standard-def*)

lemma *hypnat-of-nat-mem-Nats* [*simp*]: *hypnat-of-nat n* ∈ *Nats*
by (*simp add*: *Nats-eq-Standard*)

lemma *hypnat-of-nat-one* [*simp*]: *hypnat-of-nat* (*Suc 0*) = (*1*::*hypnat*)
by *transfer simp*

lemma *hypnat-of-nat-Suc* [*simp*]:
  hypnat-of-nat (*Suc n*) = *hypnat-of-nat n* + (*1*::*hypnat*)
by *transfer simp*

lemma *of-nat-eq-add* [*rule-format*]:
  ∀ *d*::*hypnat*. *of-nat m* = *of-nat n* + *d* −−> *d* ∈ *range of-nat*
apply (*induct n*)
apply (*auto simp add*: *add-assoc*)
apply (*case-tac x*)
apply (*auto simp add*: *add-commute* [*of 1*])
done

lemma *Nats-diff* [*simp*]: [|*a* ∈ *Nats*; *b* ∈ *Nats*|] ==> (*a*−*b* :: *hypnat*) ∈ *Nats*
by (*simp add*: *Nats-eq-Standard*)

35.3 Infinite Hypernatural Numbers − *HNatInfinite*

definition

HNatInfinite :: *hypnat set* **where**
HNatInfinite = $\{n.\ n \notin Nats\}$

lemma *Nats-not-HNatInfinite-iff*: $(x \in Nats) = (x \notin HNatInfinite)$
by (*simp add*: *HNatInfinite-def*)

lemma *HNatInfinite-not-Nats-iff*: $(x \in HNatInfinite) = (x \notin Nats)$
by (*simp add*: *HNatInfinite-def*)

lemma *star-of-neq-HNatInfinite*: $N \in HNatInfinite \Longrightarrow star\text{-}of\ n \neq N$
by (*auto simp add*: *HNatInfinite-def Nats-eq-Standard*)

lemma *star-of-Suc-lessI*:
$\bigwedge N.\ [\![star\text{-}of\ n < N;\ star\text{-}of\ (Suc\ n) \neq N]\!] \Longrightarrow star\text{-}of\ (Suc\ n) < N$
by *transfer* (*rule Suc-lessI*)

lemma *star-of-less-HNatInfinite*:
assumes *N*: $N \in HNatInfinite$
shows $star\text{-}of\ n < N$
proof (*induct n*)
case *0*
from *N* **have** $star\text{-}of\ 0 \neq N$ **by** (*rule star-of-neq-HNatInfinite*)
thus $star\text{-}of\ 0 < N$ **by** *simp*
next
case (*Suc n*)
from *N* **have** $star\text{-}of\ (Suc\ n) \neq N$ **by** (*rule star-of-neq-HNatInfinite*)
with *Suc* **show** $star\text{-}of\ (Suc\ n) < N$ **by** (*rule star-of-Suc-lessI*)
qed

lemma *star-of-le-HNatInfinite*: $N \in HNatInfinite \Longrightarrow star\text{-}of\ n \leq N$
by (*rule star-of-less-HNatInfinite* [*THEN order-less-imp-le*])

35.3.1 Closure Rules

lemma *Nats-less-HNatInfinite*: $[\![x \in Nats;\ y \in HNatInfinite]\!] \Longrightarrow x < y$
by (*auto simp add*: *Nats-def star-of-less-HNatInfinite*)

lemma *Nats-le-HNatInfinite*: $[\![x \in Nats;\ y \in HNatInfinite]\!] \Longrightarrow x \leq y$
by (*rule Nats-less-HNatInfinite* [*THEN order-less-imp-le*])

lemma *zero-less-HNatInfinite*: $x \in HNatInfinite \Longrightarrow 0 < x$
by (*simp add*: *Nats-less-HNatInfinite*)

lemma *one-less-HNatInfinite*: $x \in HNatInfinite \Longrightarrow 1 < x$
by (*simp add*: *Nats-less-HNatInfinite*)

lemma *one-le-HNatInfinite*: $x \in HNatInfinite \Longrightarrow 1 \leq x$
by (*simp add*: *Nats-le-HNatInfinite*)

lemma *zero-not-mem-HNatInfinite* [*simp*]: $0 \notin HNatInfinite$
by (*simp add*: *HNatInfinite-def*)

lemma *Nats-downward-closed*:
$[\![x \in Nats; (y::hypnat) \leq x]\!] \Longrightarrow y \in Nats$
apply (*simp only*: *linorder-not-less* [*symmetric*])
apply (*erule contrapos-np*)
apply (*drule HNatInfinite-not-Nats-iff* [*THEN iffD2*])
apply (*erule* (*1*) *Nats-less-HNatInfinite*)
done

lemma *HNatInfinite-upward-closed*:
$[\![x \in HNatInfinite; x \leq y]\!] \Longrightarrow y \in HNatInfinite$
apply (*simp only*: *HNatInfinite-not-Nats-iff*)
apply (*erule contrapos-nn*)
apply (*erule* (*1*) *Nats-downward-closed*)
done

lemma *HNatInfinite-add*: $x \in HNatInfinite \Longrightarrow x + y \in HNatInfinite$
apply (*erule HNatInfinite-upward-closed*)
apply (*rule hypnat-le-add1*)
done

lemma *HNatInfinite-add-one*: $x \in HNatInfinite \Longrightarrow x + 1 \in HNatInfinite$
by (*rule HNatInfinite-add*)

lemma *HNatInfinite-diff*:
$[\![x \in HNatInfinite; y \in Nats]\!] \Longrightarrow x - y \in HNatInfinite$
apply (*frule* (*1*) *Nats-le-HNatInfinite*)
apply (*simp only*: *HNatInfinite-not-Nats-iff*)
apply (*erule contrapos-nn*)
apply (*drule* (*1*) *Nats-add*, *simp*)
done

lemma *HNatInfinite-is-Suc*: $x \in HNatInfinite ==> \exists y.\ x = y + (1::hypnat)$
apply (*rule-tac* $x = x - (1::hypnat)$ **in** *exI*)
apply (*simp add*: *Nats-le-HNatInfinite*)
done

35.4 Existence of an infinite hypernatural number

definition

whn :: *hypnat* **where**
hypnat-omega-def: $whn = star\text{-}n\ (\%n::nat.\ n)$

lemma *hypnat-of-nat-neq-whn*: $hypnat\text{-}of\text{-}nat\ n \neq whn$
by (*simp add*: *hypnat-omega-def star-of-def star-n-eq-iff*)

lemma *whn-neq-hypnat-of-nat*: $whn \neq hypnat\text{-}of\text{-}nat\ n$
by (*simp add*: *hypnat-omega-def star-of-def star-n-eq-iff*)

lemma *whn-not-Nats* [*simp*]: $whn \notin Nats$
by (*simp add*: *Nats-def image-def whn-neq-hypnat-of-nat*)

lemma *HNatInfinite-whn* [*simp*]: $whn \in HNatInfinite$
by (*simp add*: *HNatInfinite-def*)

lemma *lemma-unbounded-set* [*simp*]: $\{n{::}nat.\ m < n\} \in FreeUltrafilterNat$
apply (*insert finite-atMost* [*of m*])
apply (*simp add*: *atMost-def*)
apply (*drule FreeUltrafilterNat.finite*)
apply (*drule FreeUltrafilterNat.not-memD*)
apply (*simp add*: *Collect-neg-eq* [*symmetric*] *linorder-not-le*)
done

lemma *Compl-Collect-le*: $-\{n{::}nat.\ N \leq n\} = \{n.\ n < N\}$
by (*simp add*: *Collect-neg-eq* [*symmetric*] *linorder-not-le*)

lemma *hypnat-of-nat-eq*:
$hypnat\text{-}of\text{-}nat\ m = star\text{-}n\ (\%n{::}nat.\ m)$
by (*simp add*: *star-of-def*)

lemma *SHNat-eq*: $Nats = \{n.\ \exists N.\ n = hypnat\text{-}of\text{-}nat\ N\}$
by (*simp add*: *Nats-def image-def*)

lemma *Nats-less-whn*: $n \in Nats \Longrightarrow n < whn$
by (*simp add*: *Nats-less-HNatInfinite*)

lemma *Nats-le-whn*: $n \in Nats \Longrightarrow n \leq whn$
by (*simp add*: *Nats-le-HNatInfinite*)

lemma *hypnat-of-nat-less-whn* [*simp*]: $hypnat\text{-}of\text{-}nat\ n < whn$
by (*simp add*: *Nats-less-whn*)

lemma *hypnat-of-nat-le-whn* [*simp*]: $hypnat\text{-}of\text{-}nat\ n \leq whn$
by (*simp add*: *Nats-le-whn*)

lemma *hypnat-zero-less-hypnat-omega* [*simp*]: $0 < whn$
by (*simp add*: *Nats-less-whn*)

lemma *hypnat-one-less-hypnat-omega* [*simp*]: $1 < whn$
by (*simp add*: *Nats-less-whn*)

35.4.1 Alternative characterization of the set of infinite hypernaturals

$HNatInfinite = \{N.\ \forall n{\in}\mathbb{N}.\ n < N\}$

lemma *HNatInfinite-FreeUltrafilterNat-lemma*:
∀ *N*::*nat*. {*n*. *f n* ≠ *N*} ∈ *FreeUltrafilterNat*
==> {*n*. *N* < *f n*} ∈ *FreeUltrafilterNat*
apply (*induct-tac N*)
apply (*drule-tac x* = *0* **in** *spec*, *simp*)
apply (*drule-tac x* = *Suc n* **in** *spec*)
apply (*elim ultra*, *auto*)
done

lemma *HNatInfinite-iff*: *HNatInfinite* = {*N*. ∀ *n* ∈ *Nats*. *n* < *N*}
apply (*safe intro*!: *Nats-less-HNatInfinite*)
apply (*auto simp add*: *HNatInfinite-def*)
done

35.4.2 Alternative Characterization of *HNatInfinite* using Free Ultrafilter

lemma *HNatInfinite-FreeUltrafilterNat*:
star-n X ∈ *HNatInfinite* ==> ∀ *u*. {*n*. *u* < *X n*}: *FreeUltrafilterNat*
apply (*auto simp add*: *HNatInfinite-iff SHNat-eq*)
apply (*drule-tac x*=*star-of u* **in** *spec*, *simp*)
apply (*simp add*: *star-of-def star-less-def starP2-star-n*)
done

lemma *FreeUltrafilterNat-HNatInfinite*:
∀ *u*. {*n*. *u* < *X n*}: *FreeUltrafilterNat* ==> *star-n X* ∈ *HNatInfinite*
by (*auto simp add*: *star-less-def starP2-star-n HNatInfinite-iff SHNat-eq hypnat-of-nat-eq*)

lemma *HNatInfinite-FreeUltrafilterNat-iff*:
(*star-n X* ∈ *HNatInfinite*) = (∀ *u*. {*n*. *u* < *X n*}: *FreeUltrafilterNat*)
by (*rule iffI* [*OF HNatInfinite-FreeUltrafilterNat*
FreeUltrafilterNat-HNatInfinite])

35.5 Embedding of the Hypernaturals into other types

definition
of-hypnat :: *hypnat* ⇒ *'a*::*semiring-1-cancel star* **where**
of-hypnat-def [*transfer-unfold*]: *of-hypnat* = ∗*f*∗ *of-nat*

lemma *of-hypnat-0* [*simp*]: *of-hypnat 0* = *0*
by *transfer* (*rule of-nat-0*)

lemma *of-hypnat-1* [*simp*]: *of-hypnat 1* = *1*
by *transfer* (*rule of-nat-1*)

lemma *of-hypnat-hSuc*: ⋀*m*. *of-hypnat* (*hSuc m*) = *1* + *of-hypnat m*
by *transfer* (*rule of-nat-Suc*)

lemma *of-hypnat-add* [*simp*]:
⋀*m n*. *of-hypnat* (*m* + *n*) = *of-hypnat m* + *of-hypnat n*

by *transfer* (*rule of-nat-add*)

lemma *of-hypnat-mult* [*simp*]:
$\bigwedge m\ n.$ *of-hypnat* $(m * n) =$ *of-hypnat* $m *$ *of-hypnat* n
by *transfer* (*rule of-nat-mult*)

lemma *of-hypnat-less-iff* [*simp*]:
$\bigwedge m\ n.$ (*of-hypnat* $m <$ (*of-hypnat* n::'*a*::*ordered-semidom star*)) $= (m < n)$
by *transfer* (*rule of-nat-less-iff*)

lemma *of-hypnat-0-less-iff* [*simp*]:
$\bigwedge n.$ $(0 <$ (*of-hypnat* n::'*a*::*ordered-semidom star*)) $= (0 < n)$
by *transfer* (*rule of-nat-0-less-iff*)

lemma *of-hypnat-less-0-iff* [*simp*]:
$\bigwedge m.$ $\neg$ (*of-hypnat* m::'*a*::*ordered-semidom star*) < 0
by *transfer* (*rule of-nat-less-0-iff*)

lemma *of-hypnat-le-iff* [*simp*]:
$\bigwedge m\ n.$ (*of-hypnat* $m \leq$ (*of-hypnat* n::'*a*::*ordered-semidom star*)) $= (m \leq n)$
by *transfer* (*rule of-nat-le-iff*)

lemma *of-hypnat-0-le-iff* [*simp*]:
$\bigwedge n.$ $0 \leq$ (*of-hypnat* n::'*a*::*ordered-semidom star*)
by *transfer* (*rule of-nat-0-le-iff*)

lemma *of-hypnat-le-0-iff* [*simp*]:
$\bigwedge m.$ ((*of-hypnat* m::'*a*::*ordered-semidom star*) $\leq 0) = (m = 0)$
by *transfer* (*rule of-nat-le-0-iff*)

lemma *of-hypnat-eq-iff* [*simp*]:
$\bigwedge m\ n.$ (*of-hypnat* $m =$ (*of-hypnat* n::'*a*::*ordered-semidom star*)) $= (m = n)$
by *transfer* (*rule of-nat-eq-iff*)

lemma *of-hypnat-eq-0-iff* [*simp*]:
$\bigwedge m.$ ((*of-hypnat* m::'*a*::*ordered-semidom star*) $= 0) = (m = 0)$
by *transfer* (*rule of-nat-eq-0-iff*)

lemma *HNatInfinite-of-hypnat-gt-zero*:
$N \in$ *HNatInfinite* $\Longrightarrow$ (0::'*a*::*ordered-semidom star*) $<$ *of-hypnat* N
by (*rule ccontr*, *simp add*: *linorder-not-less*)

end

36 HyperDef: Construction of Hyperreals Using Ultrafilters

theory *HyperDef*
imports *HyperNat ../Real/Real*
uses (*hypreal-arith.ML*)
begin

types *hypreal* = *real star*

abbreviation
hypreal-of-real :: *real* => *real star* **where**
hypreal-of-real == *star-of*

abbreviation
hypreal-of-hypnat :: *hypnat* ⇒ *hypreal* **where**
hypreal-of-hypnat ≡ *of-hypnat*

definition
omega :: *hypreal* **where**
— an infinite number = [<*1*,*2*,*3*,...>]
omega = *star-n* (λ*n*. *real* (*Suc n*))

definition
epsilon :: *hypreal* **where**
— an infinitesimal number = [<*1*,*1*/*2*,*1*/*3*,...>]
epsilon = *star-n* (λ*n*. *inverse* (*real* (*Suc n*)))

notation (*xsymbols*)
omega (ω) **and**
epsilon (ε)

notation (*HTML* **output**)
omega (ω) **and**
epsilon (ε)

36.1 Real vector class instances

instantiation *star* :: (*scaleR*) *scaleR*
begin

definition
star-scaleR-def [*transfer-unfold*]: *scaleR r* ≡ ∗*f*∗ (*scaleR r*)

instance ..

end

lemma *Standard-scaleR* [*simp*]: *x* ∈ *Standard* ⟹ *scaleR r x* ∈ *Standard*

by (*simp add*: *star-scaleR-def*)

lemma *star-of-scaleR* [*simp*]: *star-of* (*scaleR r x*) = *scaleR r* (*star-of x*)
by *transfer* (*rule refl*)

instance *star* :: (*real-vector*) *real-vector*
proof
  fix *a b* :: *real*
  show $\bigwedge x\ y{::}'a\ star.\ scaleR\ a\ (x + y) = scaleR\ a\ x + scaleR\ a\ y$
    by *transfer* (*rule scaleR-right-distrib*)
  show $\bigwedge x{::}'a\ star.\ scaleR\ (a + b)\ x = scaleR\ a\ x + scaleR\ b\ x$
    by *transfer* (*rule scaleR-left-distrib*)
  show $\bigwedge x{::}'a\ star.\ scaleR\ a\ (scaleR\ b\ x) = scaleR\ (a * b)\ x$
    by *transfer* (*rule scaleR-scaleR*)
  show $\bigwedge x{::}'a\ star.\ scaleR\ 1\ x = x$
    by *transfer* (*rule scaleR-one*)
qed

instance *star* :: (*real-algebra*) *real-algebra*
proof
  fix *a* :: *real*
  show $\bigwedge x\ y{::}'a\ star.\ scaleR\ a\ x * y = scaleR\ a\ (x * y)$
    by *transfer* (*rule mult-scaleR-left*)
  show $\bigwedge x\ y{::}'a\ star.\ x * scaleR\ a\ y = scaleR\ a\ (x * y)$
    by *transfer* (*rule mult-scaleR-right*)
qed

instance *star* :: (*real-algebra-1*) *real-algebra-1* **..**

instance *star* :: (*real-div-algebra*) *real-div-algebra* **..**

instance *star* :: (*real-field*) *real-field* **..**

lemma *star-of-real-def* [*transfer-unfold*]: *of-real r* = *star-of* (*of-real r*)
by (*unfold of-real-def*, *transfer*, *rule refl*)

lemma *Standard-of-real* [*simp*]: *of-real r* ∈ *Standard*
by (*simp add*: *star-of-real-def*)

lemma *star-of-of-real* [*simp*]: *star-of* (*of-real r*) = *of-real r*
by *transfer* (*rule refl*)

lemma *of-real-eq-star-of* [*simp*]: *of-real* = *star-of*
proof
  fix *r* :: *real*
  show *of-real r* = *star-of r*
    by *transfer simp*
qed

lemma *Reals-eq-Standard*: (*Reals* :: *hypreal set*) = *Standard*
by (*simp add*: *Reals-def Standard-def*)

36.2 Injection from *hypreal*

definition
of-hypreal :: *hypreal* ⇒ *'a*::*real-algebra-1 star* **where**
of-hypreal = *f* *of-real*

declare *of-hypreal-def* [*transfer-unfold*]

lemma *Standard-of-hypreal* [*simp*]:
r ∈ *Standard* ⟹ *of-hypreal r* ∈ *Standard*
by (*simp add*: *of-hypreal-def*)

lemma *of-hypreal-0* [*simp*]: *of-hypreal 0* = *0*
by *transfer* (*rule of-real-0*)

lemma *of-hypreal-1* [*simp*]: *of-hypreal 1* = *1*
by *transfer* (*rule of-real-1*)

lemma *of-hypreal-add* [*simp*]:
⋀*x y*. *of-hypreal* (*x* + *y*) = *of-hypreal x* + *of-hypreal y*
by *transfer* (*rule of-real-add*)

lemma *of-hypreal-minus* [*simp*]: ⋀*x*. *of-hypreal* (− *x*) = − *of-hypreal x*
by *transfer* (*rule of-real-minus*)

lemma *of-hypreal-diff* [*simp*]:
⋀*x y*. *of-hypreal* (*x* − *y*) = *of-hypreal x* − *of-hypreal y*
by *transfer* (*rule of-real-diff*)

lemma *of-hypreal-mult* [*simp*]:
⋀*x y*. *of-hypreal* (*x* * *y*) = *of-hypreal x* * *of-hypreal y*
by *transfer* (*rule of-real-mult*)

lemma *of-hypreal-inverse* [*simp*]:
⋀*x*. *of-hypreal* (*inverse x*) =
inverse (*of-hypreal x* :: *'a*::{*real-div-algebra*,*division-by-zero*} *star*)
by *transfer* (*rule of-real-inverse*)

lemma *of-hypreal-divide* [*simp*]:
⋀*x y*. *of-hypreal* (*x* / *y*) =
(*of-hypreal x* / *of-hypreal y* :: *'a*::{*real-field*,*division-by-zero*} *star*)
by *transfer* (*rule of-real-divide*)

lemma *of-hypreal-eq-iff* [*simp*]:
⋀*x y*. (*of-hypreal x* = *of-hypreal y*) = (*x* = *y*)
by *transfer* (*rule of-real-eq-iff*)

lemma *of-hypreal-eq-0-iff* [*simp*]:
$\bigwedge x.\ (\mathit{of\text{-}hypreal}\ x = 0) = (x = 0)$
by *transfer* (*rule of-real-eq-0-iff*)

36.3 Properties of *starrel*

lemma *lemma-starrel-refl* [*simp*]: $x \in \mathit{starrel}\ ``\ \{x\}$
by (*simp add*: *starrel-def*)

lemma *starrel-in-hypreal* [*simp*]: $\mathit{starrel}``\{x\}:\mathit{star}$
by (*simp add*: *star-def starrel-def quotient-def*, *blast*)

declare *Abs-star-inject* [*simp*] *Abs-star-inverse* [*simp*]
declare *equiv-starrel* [*THEN eq-equiv-class-iff*, *simp*]

36.4 *hypreal-of-real*: the Injection from *real* to *hypreal*

lemma *inj-star-of*: *inj star-of*
by (*rule inj-onI*, *simp*)

lemma *mem-Rep-star-iff*: $(X \in \mathit{Rep\text{-}star}\ x) = (x = \mathit{star\text{-}n}\ X)$
by (*cases x*, *simp add*: *star-n-def*)

lemma *Rep-star-star-n-iff* [*simp*]:
$(X \in \mathit{Rep\text{-}star}\ (\mathit{star\text{-}n}\ Y)) = (\{n.\ Y\ n = X\ n\} \in \mathcal{U})$
by (*simp add*: *star-n-def*)

lemma *Rep-star-star-n*: $X \in \mathit{Rep\text{-}star}\ (\mathit{star\text{-}n}\ X)$
by *simp*

36.5 Properties of *star-n*

lemma *star-n-add*:
$\mathit{star\text{-}n}\ X + \mathit{star\text{-}n}\ Y = \mathit{star\text{-}n}\ (\%n.\ X\ n + Y\ n)$
by (*simp only*: *star-add-def starfun2-star-n*)

lemma *star-n-minus*:
$-\ \mathit{star\text{-}n}\ X = \mathit{star\text{-}n}\ (\%n.\ -(X\ n))$
by (*simp only*: *star-minus-def starfun-star-n*)

lemma *star-n-diff*:
$\mathit{star\text{-}n}\ X - \mathit{star\text{-}n}\ Y = \mathit{star\text{-}n}\ (\%n.\ X\ n - Y\ n)$
by (*simp only*: *star-diff-def starfun2-star-n*)

lemma *star-n-mult*:
$\mathit{star\text{-}n}\ X * \mathit{star\text{-}n}\ Y = \mathit{star\text{-}n}\ (\%n.\ X\ n * Y\ n)$
by (*simp only*: *star-mult-def starfun2-star-n*)

lemma *star-n-inverse*:

inverse (*star-n X*) = *star-n* (%*n*. *inverse*(*X n*))
by (*simp only*: *star-inverse-def starfun-star-n*)

lemma *star-n-le*:
star-n X ≤ *star-n Y* =
({*n*. *X n* ≤ *Y n*} ∈ *FreeUltrafilterNat*)
by (*simp only*: *star-le-def starP2-star-n*)

lemma *star-n-less*:
star-n X < *star-n Y* = ({*n*. *X n* < *Y n*} ∈ *FreeUltrafilterNat*)
by (*simp only*: *star-less-def starP2-star-n*)

lemma *star-n-zero-num*: *0* = *star-n* (%*n*. *0*)
by (*simp only*: *star-zero-def star-of-def*)

lemma *star-n-one-num*: *1* = *star-n* (%*n*. *1*)
by (*simp only*: *star-one-def star-of-def*)

lemma *star-n-abs*:
abs (*star-n X*) = *star-n* (%*n*. *abs* (*X n*))
by (*simp only*: *star-abs-def starfun-star-n*)

36.6 Misc Others

lemma *hypreal-not-refl2*: !!(*x*::*hypreal*). *x* < *y* ==> *x* ≠ *y*
by (*auto*)

lemma *hypreal-eq-minus-iff*: ((*x*::*hypreal*) = *y*) = (*x* + − *y* = *0*)
by *auto*

lemma *hypreal-mult-left-cancel*: (*c*::*hypreal*) ≠ *0* ==> (*c*∗*a*=*c*∗*b*) = (*a*=*b*)
by *auto*

lemma *hypreal-mult-right-cancel*: (*c*::*hypreal*) ≠ *0* ==> (*a*∗*c*=*b*∗*c*) = (*a*=*b*)
by *auto*

lemma *hypreal-omega-gt-zero* [*simp*]: *0* < *omega*
by (*simp add*: *omega-def star-n-zero-num star-n-less*)

36.7 Existence of Infinite Hyperreal Number

Existence of infinite number not corresponding to any real number. Use assumption that member $\mathcal{U}$ is not finite.

A few lemmas first

lemma *lemma-omega-empty-singleton-disj*: {*n*::*nat*. *x* = *real n*} = {} |
(∃ *y*. {*n*::*nat*. *x* = *real n*} = {*y*})
by *force*

lemma *lemma-finite-omega-set*: *finite* {*n*::*nat*. *x* = *real n*}
by (*cut-tac x* = *x* **in** *lemma-omega-empty-singleton-disj*, *auto*)

lemma *not-ex-hypreal-of-real-eq-omega*:
~ (∃ *x*. *hypreal-of-real x* = *omega*)
apply (*simp add*: *omega-def*)
apply (*simp add*: *star-of-def star-n-eq-iff*)
apply (*auto simp add*: *real-of-nat-Suc diff-eq-eq* [*symmetric*]
lemma-finite-omega-set [*THEN FreeUltrafilterNat.finite*])
done

lemma *hypreal-of-real-not-eq-omega*: *hypreal-of-real x* ≠ *omega*
by (*insert not-ex-hypreal-of-real-eq-omega*, *auto*)

Existence of infinitesimal number also not corresponding to any real number

lemma *lemma-epsilon-empty-singleton-disj*:
{*n*::*nat*. *x* = *inverse*(*real*(*Suc n*))} = {} |
(∃ *y*. {*n*::*nat*. *x* = *inverse*(*real*(*Suc n*))} = {*y*})
by *auto*

lemma *lemma-finite-epsilon-set*: *finite* {*n*. *x* = *inverse*(*real*(*Suc n*))}
by (*cut-tac x* = *x* **in** *lemma-epsilon-empty-singleton-disj*, *auto*)

lemma *not-ex-hypreal-of-real-eq-epsilon*: ~ (∃ *x*. *hypreal-of-real x* = *epsilon*)
by (*auto simp add*: *epsilon-def star-of-def star-n-eq-iff*
lemma-finite-epsilon-set [*THEN FreeUltrafilterNat.finite*])

lemma *hypreal-of-real-not-eq-epsilon*: *hypreal-of-real x* ≠ *epsilon*
by (*insert not-ex-hypreal-of-real-eq-epsilon*, *auto*)

lemma *hypreal-epsilon-not-zero*: *epsilon* ≠ *0*
by (*simp add*: *epsilon-def star-zero-def star-of-def star-n-eq-iff*
del: *star-of-zero*)

lemma *hypreal-epsilon-inverse-omega*: *epsilon* = *inverse*(*omega*)
by (*simp add*: *epsilon-def omega-def star-n-inverse*)

lemma *hypreal-epsilon-gt-zero*: *0* < *epsilon*
by (*simp add*: *hypreal-epsilon-inverse-omega*)

36.8 Absolute Value Function for the Hyperreals

lemma *hrabs-add-less*:
[| *abs x* < *r*; *abs y* < *s* |] ==> *abs*(*x*+*y*) < *r* + (*s*::*hypreal*)
by (*simp add*: *abs-if split*: *split-if-asm*)

lemma *hrabs-less-gt-zero*: *abs x* < *r* ==> (*0*::*hypreal*) < *r*
by (*blast intro*!: *order-le-less-trans abs-ge-zero*)

lemma *hrabs-disj*: *abs x* = (*x*::'*a*::*abs-if*) | *abs x* = −*x*
by (*simp add*: *abs-if*)

lemma *hrabs-add-lemma-disj*: (*y*::*hypreal*) + − *x* + (*y* + − *z*) = *abs* (*x* + − *z*)
==> *y* = *z* | *x* = *y*
by (*simp add*: *abs-if split add*: *split-if-asm*)

36.9 Embedding the Naturals into the Hyperreals

abbreviation
hypreal-of-nat :: *nat* => *hypreal* **where**
hypreal-of-nat == *of-nat*

lemma *SNat-eq*: *Nats* = {*n*. ∃ *N*. *n* = *hypreal-of-nat N*}
by (*simp add*: *Nats-def image-def*)

lemma *hypreal-of-nat-eq*:
hypreal-of-nat (*n*::*nat*) = *hypreal-of-real* (*real n*)
by (*simp add*: *real-of-nat-def*)

lemma *hypreal-of-nat*:
hypreal-of-nat m = *star-n* (%*n*. *real m*)
apply (*fold star-of-def*)
apply (*simp add*: *real-of-nat-def*)
done

use *hypreal-arith.ML*
declaration ⟪ *K hypreal-arith-setup* ⟫

36.10 Exponentials on the Hyperreals

lemma *hpowr-0* [*simp*]: *r* ^ *0* = (*1*::*hypreal*)
by (*rule power-0*)

lemma *hpowr-Suc* [*simp*]: *r* ^ (*Suc n*) = (*r*::*hypreal*) ∗ (*r* ^ *n*)
by (*rule power-Suc*)

lemma *hrealpow-two*: (*r*::*hypreal*) ^ *Suc* (*Suc 0*) = *r* ∗ *r*
by *simp*

lemma *hrealpow-two-le* [*simp*]: (*0*::*hypreal*) ≤ *r* ^ *Suc* (*Suc 0*)
by (*auto simp add*: *zero-le-mult-iff*)

lemma *hrealpow-two-le-add-order* [*simp*]:
(*0*::*hypreal*) ≤ *u* ^ *Suc* (*Suc 0*) + *v* ^ *Suc* (*Suc 0*)
by (*simp only*: *hrealpow-two-le add-nonneg-nonneg*)

lemma *hrealpow-two-le-add-order2* [*simp*]:
(*0*::*hypreal*) ≤ *u* ^ *Suc* (*Suc 0*) + *v* ^ *Suc* (*Suc 0*) + *w* ^ *Suc* (*Suc 0*)
by (*simp only*: *hrealpow-two-le add-nonneg-nonneg*)

lemma *hypreal-add-nonneg-eq-0-iff*:
[| *0* ≤ *x*; *0* ≤ *y* |] ==> (*x*+*y* = *0*) = (*x* = *0* & *y* = (*0*::*hypreal*))
by *arith*

FIXME: DELETE THESE

lemma *hypreal-three-squares-add-zero-iff*:
(*x*∗*x* + *y*∗*y* + *z*∗*z* = *0*) = (*x* = *0* & *y* = *0* & *z* = (*0*::*hypreal*))
apply (*simp only*: *zero-le-square add-nonneg-nonneg hypreal-add-nonneg-eq-0-iff*, *auto*)
done

lemma *hrealpow-three-squares-add-zero-iff* [*simp*]:
(*x* ^ *Suc* (*Suc 0*) + *y* ^ *Suc* (*Suc 0*) + *z* ^ *Suc* (*Suc 0*) = (*0*::*hypreal*)) =
(*x* = *0* & *y* = *0* & *z* = *0*)
by (*simp only*: *hypreal-three-squares-add-zero-iff hrealpow-two*)

lemma *hrabs-hrealpow-two* [*simp*]:
abs(*x* ^ *Suc* (*Suc 0*)) = (*x*::*hypreal*) ^ *Suc* (*Suc 0*)
by (*simp add*: *abs-mult*)

lemma *two-hrealpow-ge-one* [*simp*]: (*1*::*hypreal*) ≤ *2* ^ *n*
by (*insert power-increasing* [*of 0 n 2*::*hypreal*], *simp*)

lemma *two-hrealpow-gt* [*simp*]: *hypreal-of-nat n* < *2* ^ *n*
apply (*induct-tac n*)
apply (*auto simp add*: *left-distrib*)
apply (*cut-tac n* = *n* **in** *two-hrealpow-ge-one*, *arith*)
done

lemma *hrealpow*:
star-n X ^ *m* = *star-n* (%*n*. (*X n*::*real*) ^ *m*)
apply (*induct-tac m*)
apply (*auto simp add*: *star-n-one-num star-n-mult power-0*)
done

lemma *hrealpow-sum-square-expand*:
(*x* + (*y*::*hypreal*)) ^ *Suc* (*Suc 0*) =
x ^ *Suc* (*Suc 0*) + *y* ^ *Suc* (*Suc 0*) + (*hypreal-of-nat* (*Suc* (*Suc 0*)))∗*x*∗*y*
by (*simp add*: *right-distrib left-distrib*)

lemma *power-hypreal-of-real-number-of*:
(*number-of v* :: *hypreal*) ^ *n* = *hypreal-of-real* ((*number-of v*) ^ *n*)
by *simp*
declare *power-hypreal-of-real-number-of* [*of* - *number-of w*, *standard*, *simp*]

36.11 Powers with Hypernatural Exponents

definition

pow :: ['*a*::*power star*, *nat star*] ⇒ '*a star* (**infixr** *pow* 80) **where**
hyperpow-def [*transfer-unfold*]:
R pow N = (∗*f2*∗ *op* ^) *R N*

lemma *Standard-hyperpow* [*simp*]:
⟦*r* ∈ *Standard*; *n* ∈ *Standard*⟧ ⟹ *r pow n* ∈ *Standard*
unfolding *hyperpow-def* **by** *simp*

lemma *hyperpow*: *star-n X pow star-n Y* = *star-n* (%*n*. *X n* ^ *Y n*)
by (*simp add*: *hyperpow-def starfun2-star-n*)

lemma *hyperpow-zero* [*simp*]:
⋀*n*. (*0*::'*a*::{*recpower*,*semiring-0*} *star*) *pow* (*n* + (*1*::*hypnat*)) = *0*
by *transfer simp*

lemma *hyperpow-not-zero*:
⋀*r n*. *r* ≠ (*0*::'*a*::{*recpower*,*field*} *star*) ==> *r pow n* ≠ *0*
by *transfer* (*rule field-power-not-zero*)

lemma *hyperpow-inverse*:
⋀*r n*. *r* ≠ (*0*::'*a*::{*recpower*,*division-by-zero*,*field*} *star*)
⟹ *inverse* (*r pow n*) = (*inverse r*) *pow n*
by *transfer* (*rule power-inverse*)

lemma *hyperpow-hrabs*:
⋀*r n*. *abs* (*r*::'*a*::{*recpower*,*ordered-idom*} *star*) *pow n* = *abs* (*r pow n*)
by *transfer* (*rule power-abs* [*symmetric*])

lemma *hyperpow-add*:
⋀*r n m*. (*r*::'*a*::*recpower star*) *pow* (*n* + *m*) = (*r pow n*) ∗ (*r pow m*)
by *transfer* (*rule power-add*)

lemma *hyperpow-one* [*simp*]:
⋀*r*. (*r*::'*a*::*recpower star*) *pow* (*1*::*hypnat*) = *r*
by *transfer* (*rule power-one-right*)

lemma *hyperpow-two*:
⋀*r*. (*r*::'*a*::*recpower star*) *pow* ((*1*::*hypnat*) + (*1*::*hypnat*)) = *r* ∗ *r*
by *transfer* (*simp add*: *power-Suc*)

lemma *hyperpow-gt-zero*:
$\bigwedge r\ n.\ (0::'a::\{recpower,ordered\text{-}semidom\}\ star) < r \Longrightarrow 0 < r\ pow\ n$
by *transfer* (*rule zero-less-power*)

lemma *hyperpow-ge-zero*:
$\bigwedge r\ n.\ (0::'a::\{recpower,ordered\text{-}semidom\}\ star) \leq r \Longrightarrow 0 \leq r\ pow\ n$
by *transfer* (*rule zero-le-power*)

lemma *hyperpow-le*:
$\bigwedge x\ y\ n.\ [\![(0::'a::\{recpower,ordered\text{-}semidom\}\ star) < x;\ x \leq y]\!]$
$\Longrightarrow x\ pow\ n \leq y\ pow\ n$
by *transfer* (*rule power-mono* [*OF - order-less-imp-le*])

lemma *hyperpow-eq-one* [*simp*]:
$\bigwedge n.\ 1\ pow\ n = (1::'a::recpower\ star)$
by *transfer* (*rule power-one*)

lemma *hrabs-hyperpow-minus-one* [*simp*]:
$\bigwedge n.\ abs(-1\ pow\ n) = (1::'a::\{number\text{-}ring,recpower,ordered\text{-}idom\}\ star)$
by *transfer* (*rule abs-power-minus-one*)

lemma *hyperpow-mult*:
$\bigwedge r\ s\ n.\ (r * s::'a::\{comm\text{-}monoid\text{-}mult,recpower\}\ star)\ pow\ n$
$= (r\ pow\ n) * (s\ pow\ n)$
by *transfer* (*rule power-mult-distrib*)

lemma *hyperpow-two-le* [*simp*]:
$(0::'a::\{recpower,ordered\text{-}ring\text{-}strict\}\ star) \leq r\ pow\ (1 + 1)$
by (*auto simp add*: *hyperpow-two zero-le-mult-iff*)

lemma *hrabs-hyperpow-two* [*simp*]:
$abs(x\ pow\ (1 + 1)) =$
$(x::'a::\{recpower,ordered\text{-}ring\text{-}strict\}\ star)\ pow\ (1 + 1)$
by (*simp only*: *abs-of-nonneg hyperpow-two-le*)

lemma *hyperpow-two-hrabs* [*simp*]:
$abs(x::'a::\{recpower,ordered\text{-}idom\}\ star)\ pow\ (1 + 1) = x\ pow\ (1 + 1)$
by (*simp add*: *hyperpow-hrabs*)

The precondition could be weakened to $(0::'a) \leq x$

lemma *hypreal-mult-less-mono*:
[| $u<v$; $x<y$; $(0::hypreal) < v$; $0 < x$ |] ==> $u*x < v* y$
by (*simp add*: *Ring-and-Field.mult-strict-mono order-less-imp-le*)

lemma *hyperpow-two-gt-one*:
$\bigwedge r::'a::\{recpower,ordered\text{-}semidom\}\ star.\ 1 < r \Longrightarrow 1 < r\ pow\ (1 + 1)$
by *transfer* (*simp add*: *power-gt1*)

lemma *hyperpow-two-ge-one*:

$\bigwedge r::'a::\{recpower,ordered\text{-}semidom\}\ star.\ 1 \leq r \Longrightarrow 1 \leq r\ pow\ (1 + 1)$
by *transfer* (*simp add*: *one-le-power*)

lemma *two-hyperpow-ge-one* [*simp*]: (*1*::*hypreal*) ≤ *2 pow n*
apply (*rule-tac y = 1 pow n* **in** *order-trans*)
apply (*rule-tac* [*2*] *hyperpow-le*, *auto*)
done

lemma *hyperpow-minus-one2* [*simp*]:
!!*n*. *−1 pow* ((*1 + 1*)∗*n*) = (*1*::*hypreal*)
by *transfer* (*subst power-mult*, *simp*)

lemma *hyperpow-less-le*:
!!*r n N*. [|(*0*::*hypreal*) ≤ *r*; *r* ≤ *1*; *n* < *N*|] ==> *r pow N* ≤ *r pow n*
by *transfer* (*rule power-decreasing* [*OF order-less-imp-le*])

lemma *hyperpow-SHNat-le*:
[| *0* ≤ *r*; *r* ≤ (*1*::*hypreal*); *N* ∈ *HNatInfinite* |]
==> *ALL n*: *Nats*. *r pow N* ≤ *r pow n*
by (*auto intro*!: *hyperpow-less-le simp add*: *HNatInfinite-iff*)

lemma *hyperpow-realpow*:
(*hypreal-of-real r*) *pow* (*hypnat-of-nat n*) = *hypreal-of-real* (*r* ^ *n*)
by *transfer* (*rule refl*)

lemma *hyperpow-SReal* [*simp*]:
(*hypreal-of-real r*) *pow* (*hypnat-of-nat n*) ∈ *Reals*
by (*simp add*: *Reals-eq-Standard*)

lemma *hyperpow-zero-HNatInfinite* [*simp*]:
N ∈ *HNatInfinite* ==> (*0*::*hypreal*) *pow N* = *0*
by (*drule HNatInfinite-is-Suc*, *auto*)

lemma *hyperpow-le-le*:
[| (*0*::*hypreal*) ≤ *r*; *r* ≤ *1*; *n* ≤ *N* |] ==> *r pow N* ≤ *r pow n*
apply (*drule order-le-less* [*of n*, *THEN iffD1*])
apply (*auto intro*: *hyperpow-less-le*)
done

lemma *hyperpow-Suc-le-self2*:
[| (*0*::*hypreal*) ≤ *r*; *r* < *1* |] ==> *r pow* (*n* + (*1*::*hypnat*)) ≤ *r*
apply (*drule-tac n* = (*1*::*hypnat*) **in** *hyperpow-le-le*)
apply *auto*
done

lemma *hyperpow-hypnat-of-nat*: $\bigwedge x.\ x\ pow\ hypnat\text{-}of\text{-}nat\ n = x\ \hat{}\ n$
by *transfer* (*rule refl*)

lemma *of-hypreal-hyperpow*:

$\bigwedge x\ n.$ *of-hypreal* $(x\ pow\ n) =$
$(of\text{-}hypreal\ x{::}'a{::}\{real\text{-}algebra\text{-}1,recpower\}\ star)\ pow\ n$
by *transfer* (*rule of-real-power*)

end

37 NSA: Infinite Numbers, Infinitesimals, Infinitely Close Relation

theory *NSA*
imports *HyperDef ../Real/RComplete*
begin

definition
hnorm :: $'a{::}norm\ star \Rightarrow real\ star$ **where**
$hnorm = *f* norm$

definition
Infinitesimal :: $('a{::}real\text{-}normed\text{-}vector)\ star\ set$ **where**
$Infinitesimal = \{x.\ \forall r \in Reals.\ 0 < r \longrightarrow hnorm\ x < r\}$

definition
HFinite :: $('a{::}real\text{-}normed\text{-}vector)\ star\ set$ **where**
$HFinite = \{x.\ \exists r \in Reals.\ hnorm\ x < r\}$

definition
HInfinite :: $('a{::}real\text{-}normed\text{-}vector)\ star\ set$ **where**
$HInfinite = \{x.\ \forall r \in Reals.\ r < hnorm\ x\}$

definition
approx :: $['a{::}real\text{-}normed\text{-}vector\ star, 'a\ star] \Rightarrow bool$ (**infixl** @= *50*) **where**
— the 'infinitely close' relation
$(x\ @=\ y) = ((x - y) \in Infinitesimal)$

definition
st :: *hypreal* => *hypreal* **where**
— the standard part of a hyperreal
$st = (\%x.\ @r.\ x \in HFinite\ \&\ r \in Reals\ \&\ r\ @=\ x)$

definition
monad :: $'a{::}real\text{-}normed\text{-}vector\ star \Rightarrow 'a\ star\ set$ **where**
$monad\ x = \{y.\ x\ @=\ y\}$

definition
galaxy :: $'a{::}real\text{-}normed\text{-}vector\ star \Rightarrow 'a\ star\ set$ **where**
$galaxy\ x = \{y.\ (x + -y) \in HFinite\}$

notation (*xsymbols*)
approx (**infixl** $\approx$ *50*)

notation (*HTML* **output**)
approx (**infixl** $\approx$ *50*)

lemma *SReal-def*: *Reals* == $\{x.\ \exists r.\ x = hypreal\text{-}of\text{-}real\ r\}$
by (*simp add*: *Reals-def image-def*)

37.1 Nonstandard Extension of the Norm Function

definition
scaleHR :: *real star* $\Rightarrow$ *'a star* $\Rightarrow$ *'a*::*real-normed-vector star* **where**
scaleHR = *starfun2 scaleR*

declare *hnorm-def* [*transfer-unfold*]
declare *scaleHR-def* [*transfer-unfold*]

lemma *Standard-hnorm* [*simp*]: $x \in Standard \Longrightarrow hnorm\ x \in Standard$
by (*simp add*: *hnorm-def*)

lemma *star-of-norm* [*simp*]: *star-of* (*norm x*) = *hnorm* (*star-of x*)
by *transfer* (*rule refl*)

lemma *hnorm-ge-zero* [*simp*]:
$\bigwedge x$::*'a*::*real-normed-vector star*. $0 \le hnorm\ x$
by *transfer* (*rule norm-ge-zero*)

lemma *hnorm-eq-zero* [*simp*]:
$\bigwedge x$::*'a*::*real-normed-vector star*. (*hnorm x* = *0*) = (*x* = *0*)
by *transfer* (*rule norm-eq-zero*)

lemma *hnorm-triangle-ineq*:
$\bigwedge x\ y$::*'a*::*real-normed-vector star*. *hnorm* (*x* + *y*) $\le$ *hnorm x* + *hnorm y*
by *transfer* (*rule norm-triangle-ineq*)

lemma *hnorm-triangle-ineq3*:
$\bigwedge x\ y$::*'a*::*real-normed-vector star*. $|hnorm\ x - hnorm\ y| \le hnorm\ (x - y)$
by *transfer* (*rule norm-triangle-ineq3*)

lemma *hnorm-scaleR*:
$\bigwedge x$::*'a*::*real-normed-vector star*.
hnorm (*a* $*_R$ *x*) = |*star-of a*| * *hnorm x*
by *transfer* (*rule norm-scaleR*)

lemma *hnorm-scaleHR*:
$\bigwedge a\ (x$::*'a*::*real-normed-vector star*).
hnorm (*scaleHR a x*) = |*a*| * *hnorm x*
by *transfer* (*rule norm-scaleR*)

lemma *hnorm-mult-ineq*:
$\bigwedge x\ y::'a::$*real-normed-algebra star*$.\ hnorm\ (x * y) \leq hnorm\ x * hnorm\ y$
by *transfer* (*rule norm-mult-ineq*)

lemma *hnorm-mult*:
$\bigwedge x\ y::'a::$*real-normed-div-algebra star*$.\ hnorm\ (x * y) = hnorm\ x * hnorm\ y$
by *transfer* (*rule norm-mult*)

lemma *hnorm-hyperpow*:
$\bigwedge(x::'a::\{$*real-normed-div-algebra*,*recpower*$\}$ *star*$)\ n.$
$hnorm\ (x\ pow\ n) = hnorm\ x\ pow\ n$
by *transfer* (*rule norm-power*)

lemma *hnorm-one* [*simp*]:
$hnorm\ (1::'a::$*real-normed-div-algebra star*$) = 1$
by *transfer* (*rule norm-one*)

lemma *hnorm-zero* [*simp*]:
$hnorm\ (0::'a::$*real-normed-vector star*$) = 0$
by *transfer* (*rule norm-zero*)

lemma *zero-less-hnorm-iff* [*simp*]:
$\bigwedge x::'a::$*real-normed-vector star*$.\ (0 < hnorm\ x) = (x \neq 0)$
by *transfer* (*rule zero-less-norm-iff*)

lemma *hnorm-minus-cancel* [*simp*]:
$\bigwedge x::'a::$*real-normed-vector star*$.\ hnorm\ (-x) = hnorm\ x$
by *transfer* (*rule norm-minus-cancel*)

lemma *hnorm-minus-commute*:
$\bigwedge a\ b::'a::$*real-normed-vector star*$.\ hnorm\ (a - b) = hnorm\ (b - a)$
by *transfer* (*rule norm-minus-commute*)

lemma *hnorm-triangle-ineq2*:
$\bigwedge a\ b::'a::$*real-normed-vector star*$.\ hnorm\ a - hnorm\ b \leq hnorm\ (a - b)$
by *transfer* (*rule norm-triangle-ineq2*)

lemma *hnorm-triangle-ineq4*:
$\bigwedge a\ b::'a::$*real-normed-vector star*$.\ hnorm\ (a - b) \leq hnorm\ a + hnorm\ b$
by *transfer* (*rule norm-triangle-ineq4*)

lemma *abs-hnorm-cancel* [*simp*]:
$\bigwedge a::'a::$*real-normed-vector star*$.\ |hnorm\ a| = hnorm\ a$
by *transfer* (*rule abs-norm-cancel*)

lemma *hnorm-of-hypreal* [*simp*]:
$\bigwedge r.\ hnorm\ (of\text{-}hypreal\ r::'a::$*real-normed-algebra-1 star*$) = |r|$
by *transfer* (*rule norm-of-real*)

lemma *nonzero-hnorm-inverse*:
$\bigwedge a::'a::$*real-normed-div-algebra star*$.$
$a \neq 0 \Longrightarrow hnorm\ (inverse\ a) = inverse\ (hnorm\ a)$
by *transfer* (*rule nonzero-norm-inverse*)

lemma *hnorm-inverse*:
$\bigwedge a::'a::\{$*real-normed-div-algebra*,*division-by-zero*$\}$ *star*$.$
$hnorm\ (inverse\ a) = inverse\ (hnorm\ a)$
by *transfer* (*rule norm-inverse*)

lemma *hnorm-divide*:
$\bigwedge a\ b::'a::\{$*real-normed-field*,*division-by-zero*$\}$ *star*$.$
$hnorm\ (a\ /\ b) = hnorm\ a\ /\ hnorm\ b$
by *transfer* (*rule norm-divide*)

lemma *hypreal-hnorm-def* [*simp*]:
$\bigwedge r::$*hypreal*$.\ hnorm\ r \equiv |r|$
by *transfer* (*rule real-norm-def*)

lemma *hnorm-add-less*:
$\bigwedge (x::'a::$*real-normed-vector star*$)\ y\ r\ s.$
$[\![hnorm\ x < r;\ hnorm\ y < s]\!] \Longrightarrow hnorm\ (x + y) < r + s$
by *transfer* (*rule norm-add-less*)

lemma *hnorm-mult-less*:
$\bigwedge (x::'a::$*real-normed-algebra star*$)\ y\ r\ s.$
$[\![hnorm\ x < r;\ hnorm\ y < s]\!] \Longrightarrow hnorm\ (x * y) < r * s$
by *transfer* (*rule norm-mult-less*)

lemma *hnorm-scaleHR-less*:
$[\![|x| < r;\ hnorm\ y < s]\!] \Longrightarrow hnorm\ (scaleHR\ x\ y) < r * s$
apply (*simp only*: *hnorm-scaleHR*)
apply (*simp add*: *mult-strict-mono'*)
done

37.2 Closure Laws for the Standard Reals

lemma *Reals-minus-iff* [*simp*]: $(-x \in Reals) = (x \in Reals)$
apply *auto*
apply (*drule Reals-minus*, *auto*)
done

lemma *Reals-add-cancel*: $[\![x + y \in Reals;\ y \in Reals]\!] \Longrightarrow x \in Reals$
by (*drule* (*1*) *Reals-diff*, *simp*)

lemma *SReal-hrabs*: $(x::hypreal) \in Reals \Longrightarrow abs\ x \in Reals$
by (*simp add*: *Reals-eq-Standard*)

lemma *SReal-hypreal-of-real* [*simp*]: *hypreal-of-real x* ∈ *Reals*
by (*simp add*: *Reals-eq-Standard*)

lemma *SReal-divide-number-of*: *r* ∈ *Reals* ==> *r*/(*number-of w*::*hypreal*) ∈ *Reals*
by *simp*

epsilon is not in Reals because it is an infinitesimal

lemma *SReal-epsilon-not-mem*: *epsilon* ∉ *Reals*
apply (*simp add*: *SReal-def*)
apply (*auto simp add*: *hypreal-of-real-not-eq-epsilon* [*THEN not-sym*])
done

lemma *SReal-omega-not-mem*: *omega* ∉ *Reals*
apply (*simp add*: *SReal-def*)
apply (*auto simp add*: *hypreal-of-real-not-eq-omega* [*THEN not-sym*])
done

lemma *SReal-UNIV-real*: {*x*. *hypreal-of-real x* ∈ *Reals*} = (*UNIV*::*real set*)
by *simp*

lemma *SReal-iff*: (*x* ∈ *Reals*) = (∃ *y*. *x* = *hypreal-of-real y*)
by (*simp add*: *SReal-def*)

lemma *hypreal-of-real-image*: *hypreal-of-real* ‘(*UNIV*::*real set*) = *Reals*
by (*simp add*: *Reals-eq-Standard Standard-def*)

lemma *inv-hypreal-of-real-image*: *inv hypreal-of-real* ‘ *Reals* = *UNIV*
apply (*auto simp add*: *SReal-def*)
apply (*rule inj-star-of* [*THEN inv-f-f*, *THEN subst*], *blast*)
done

lemma *SReal-hypreal-of-real-image*:
[| ∃ *x*. *x*: *P*; *P* ⊆ *Reals* |] ==> ∃ *Q*. *P* = *hypreal-of-real* ‘ *Q*
by (*simp add*: *SReal-def image-def*, *blast*)

lemma *SReal-dense*:
[| (*x*::*hypreal*) ∈ *Reals*; *y* ∈ *Reals*; *x*<*y* |] ==> ∃ *r* ∈ *Reals*. *x*<*r* & *r*<*y*
apply (*auto simp add*: *SReal-def*)
apply (*drule dense*, *auto*)
done

Completeness of Reals, but both lemmas are unused.

lemma *SReal-sup-lemma*:
P ⊆ *Reals* ==> ((∃ *x* ∈ *P*. *y* < *x*) =
(∃ *X*. *hypreal-of-real X* ∈ *P* & *y* < *hypreal-of-real X*))
by (*blast dest*!: *SReal-iff* [*THEN iffD1*])

lemma *SReal-sup-lemma2*:
[| *P* ⊆ *Reals*; ∃ *x*. *x* ∈ *P*; ∃ *y* ∈ *Reals*. ∀ *x* ∈ *P*. *x* < *y* |]

$==>$ $(\exists X.\ X \in \{w.\ hypreal\text{-}of\text{-}real\ w \in P\})$ &
$(\exists Y.\ \forall X \in \{w.\ hypreal\text{-}of\text{-}real\ w \in P\}.\ X < Y)$
apply (*rule conjI*)
apply (*fast dest*!: *SReal-iff* [*THEN iffD1*])
apply (*auto*, *frule subsetD*, *assumption*)
apply (*drule SReal-iff* [*THEN iffD1*])
apply (*auto*, *rule-tac x = ya* **in** *exI*, *auto*)
done

37.3 Set of Finite Elements is a Subring of the Extended Reals

lemma *HFinite-add*: $[|x \in HFinite;\ y \in HFinite|] ==> (x+y) \in HFinite$
apply (*simp add*: *HFinite-def*)
apply (*blast intro*!: *Reals-add hnorm-add-less*)
done

lemma *HFinite-mult*:
fixes *x y* :: *'a*::*real-normed-algebra star*
shows $[|x \in HFinite;\ y \in HFinite|] ==> x*y \in HFinite$
apply (*simp add*: *HFinite-def*)
apply (*blast intro*!: *Reals-mult hnorm-mult-less*)
done

lemma *HFinite-scaleHR*:
$[|x \in HFinite;\ y \in HFinite|] ==> scaleHR\ x\ y \in HFinite$
apply (*simp add*: *HFinite-def*)
apply (*blast intro*!: *Reals-mult hnorm-scaleHR-less*)
done

lemma *HFinite-minus-iff*: $(-x \in HFinite) = (x \in HFinite)$
by (*simp add*: *HFinite-def*)

lemma *HFinite-star-of* [*simp*]: *star-of x* $\in$ *HFinite*
apply (*simp add*: *HFinite-def*)
apply (*rule-tac x=star-of* (*norm x*) + *1* **in** *bexI*)
apply (*transfer*, *simp*)
apply (*blast intro*: *Reals-add SReal-hypreal-of-real Reals-1*)
done

lemma *SReal-subset-HFinite*: (*Reals*::*hypreal set*) $\subseteq$ *HFinite*
by (*auto simp add*: *SReal-def*)

lemma *HFiniteD*: $x \in HFinite ==> \exists t \in Reals.\ hnorm\ x < t$
by (*simp add*: *HFinite-def*)

lemma *HFinite-hrabs-iff* [*iff*]: (*abs* (*x*::*hypreal*) $\in$ *HFinite*) = ($x \in HFinite$)
by (*simp add*: *HFinite-def*)

lemma *HFinite-hnorm-iff* [*iff*]:
(*hnorm* (*x*::*hypreal*) ∈ *HFinite*) = (*x* ∈ *HFinite*)
by (*simp add*: *HFinite-def*)

lemma *HFinite-number-of* [*simp*]: *number-of w* ∈ *HFinite*
unfolding *star-number-def* **by** (*rule HFinite-star-of*)

lemma *HFinite-0* [*simp*]: *0* ∈ *HFinite*
unfolding *star-zero-def* **by** (*rule HFinite-star-of*)

lemma *HFinite-1* [*simp*]: *1* ∈ *HFinite*
unfolding *star-one-def* **by** (*rule HFinite-star-of*)

lemma *hrealpow-HFinite*:
fixes *x* :: *'a*::{*real-normed-algebra,recpower*} *star*
shows *x* ∈ *HFinite* ==> *x* ^ *n* ∈ *HFinite*
apply (*induct-tac n*)
apply (*auto simp add*: *power-Suc intro*: *HFinite-mult*)
done

lemma *HFinite-bounded*:
[|(*x*::*hypreal*) ∈ *HFinite*; *y* ≤ *x*; *0* ≤ *y* |] ==> *y* ∈ *HFinite*
apply (*case-tac x* ≤ *0*)
apply (*drule-tac y* = *x* **in** *order-trans*)
apply (*drule-tac* [*2*] *order-antisym*)
apply (*auto simp add*: *linorder-not-le*)
apply (*auto intro*: *order-le-less-trans simp add*: *abs-if HFinite-def*)
done

37.4 Set of Infinitesimals is a Subring of the Hyperreals

lemma *InfinitesimalI*:
(⋀*r*. ⟦*r* ∈ ℝ; *0* < *r*⟧ ⟹ *hnorm x* < *r*) ⟹ *x* ∈ *Infinitesimal*
by (*simp add*: *Infinitesimal-def*)

lemma *InfinitesimalD*:
x ∈ *Infinitesimal* ==> ∀ *r* ∈ *Reals*. *0* < *r* --> *hnorm x* < *r*
by (*simp add*: *Infinitesimal-def*)

lemma *InfinitesimalI2*:
(⋀*r*. *0* < *r* ⟹ *hnorm x* < *star-of r*) ⟹ *x* ∈ *Infinitesimal*
by (*auto simp add*: *Infinitesimal-def SReal-def*)

lemma *InfinitesimalD2*:
⟦*x* ∈ *Infinitesimal*; *0* < *r*⟧ ⟹ *hnorm x* < *star-of r*
by (*auto simp add*: *Infinitesimal-def SReal-def*)

lemma *Infinitesimal-zero* [*iff*]: *0* ∈ *Infinitesimal*
by (*simp add*: *Infinitesimal-def*)

lemma *hypreal-sum-of-halves*: *x*/(*2*::*hypreal*) + *x*/(*2*::*hypreal*) = *x*
by *auto*

lemma *Infinitesimal-add*:
[| *x* ∈ *Infinitesimal*; *y* ∈ *Infinitesimal* |] ==> (*x*+*y*) ∈ *Infinitesimal*
apply (*rule InfinitesimalI*)
apply (*rule hypreal-sum-of-halves* [*THEN subst*])
apply (*drule half-gt-zero*)
apply (*blast intro*: *hnorm-add-less SReal-divide-number-of dest*: *InfinitesimalD*)
done

lemma *Infinitesimal-minus-iff* [*simp*]: (−*x*:*Infinitesimal*) = (*x*:*Infinitesimal*)
by (*simp add*: *Infinitesimal-def*)

lemma *Infinitesimal-hnorm-iff*:
(*hnorm x* ∈ *Infinitesimal*) = (*x* ∈ *Infinitesimal*)
by (*simp add*: *Infinitesimal-def*)

lemma *Infinitesimal-hrabs-iff* [*iff*]:
(*abs* (*x*::*hypreal*) ∈ *Infinitesimal*) = (*x* ∈ *Infinitesimal*)
by (*simp add*: *abs-if*)

lemma *Infinitesimal-of-hypreal-iff* [*simp*]:
((*of-hypreal x*::'*a*::*real-normed-algebra-1 star*) ∈ *Infinitesimal*) =
(*x* ∈ *Infinitesimal*)
by (*subst Infinitesimal-hnorm-iff* [*symmetric*], *simp*)

lemma *Infinitesimal-diff*:
[| *x* ∈ *Infinitesimal*; *y* ∈ *Infinitesimal* |] ==> *x*−*y* ∈ *Infinitesimal*
by (*simp add*: *diff-def Infinitesimal-add*)

lemma *Infinitesimal-mult*:
fixes *x y* :: '*a*::*real-normed-algebra star*
shows [|*x* ∈ *Infinitesimal*; *y* ∈ *Infinitesimal*|] ==> (*x* ∗ *y*) ∈ *Infinitesimal*
apply (*rule InfinitesimalI*)
apply (*subgoal-tac hnorm* (*x* ∗ *y*) < *1* ∗ *r*, *simp only*: *mult-1*)
apply (*rule hnorm-mult-less*)
apply (*simp-all add*: *InfinitesimalD*)
done

lemma *Infinitesimal-HFinite-mult*:
fixes *x y* :: '*a*::*real-normed-algebra star*
shows [| *x* ∈ *Infinitesimal*; *y* ∈ *HFinite* |] ==> (*x* ∗ *y*) ∈ *Infinitesimal*
apply (*rule InfinitesimalI*)
apply (*drule HFiniteD*, *clarify*)
apply (*subgoal-tac* 0 < *t*)

apply (*subgoal-tac hnorm* $(x * y) < (r \;/\; t) * t$, *simp*)
apply (*subgoal-tac* $0 < r \;/\; t$)
apply (*rule hnorm-mult-less*)
apply (*simp add*: *InfinitesimalD Reals-divide*)
apply *assumption*
apply (*simp only*: *divide-pos-pos*)
apply (*erule order-le-less-trans* [*OF hnorm-ge-zero*])
done

lemma *Infinitesimal-HFinite-scaleHR*:
[| $x \in$ *Infinitesimal*; $y \in$ *HFinite* |] ==> *scaleHR x y* $\in$ *Infinitesimal*
apply (*rule InfinitesimalI*)
apply (*drule HFiniteD*, *clarify*)
apply (*drule InfinitesimalD*)
apply (*simp add*: *hnorm-scaleHR*)
apply (*subgoal-tac* $0 < t$)
apply (*subgoal-tac* $|x| * hnorm\ y < (r \;/\; t) * t$, *simp*)
apply (*subgoal-tac* $0 < r \;/\; t$)
apply (*rule mult-strict-mono'*, *simp-all*)
apply (*simp only*: *divide-pos-pos*)
apply (*erule order-le-less-trans* [*OF hnorm-ge-zero*])
done

lemma *Infinitesimal-HFinite-mult2*:
fixes *x y* :: *'a*::*real-normed-algebra star*
shows [| $x \in$ *Infinitesimal*; $y \in$ *HFinite* |] ==> $(y * x) \in$ *Infinitesimal*
apply (*rule InfinitesimalI*)
apply (*drule HFiniteD*, *clarify*)
apply (*subgoal-tac* $0 < t$)
apply (*subgoal-tac hnorm* $(y * x) < t * (r \;/\; t)$, *simp*)
apply (*subgoal-tac* $0 < r \;/\; t$)
apply (*rule hnorm-mult-less*)
apply *assumption*
apply (*simp add*: *InfinitesimalD Reals-divide*)
apply (*simp only*: *divide-pos-pos*)
apply (*erule order-le-less-trans* [*OF hnorm-ge-zero*])
done

lemma *Infinitesimal-scaleR2*:
$x \in$ *Infinitesimal* ==> $a *_R x \in$ *Infinitesimal*
apply (*case-tac* $a = 0$, *simp*)
apply (*rule InfinitesimalI*)
apply (*drule InfinitesimalD*)
apply (*drule-tac x=r* $\;/\; |$*star-of a*$|$ **in** *bspec*)
apply (*simp add*: *Reals-eq-Standard*)
apply (*simp add*: *divide-pos-pos*)
apply (*simp add*: *hnorm-scaleR pos-less-divide-eq mult-commute*)
done

lemma *Compl-HFinite*: $- HFinite = HInfinite$
apply (*auto simp add*: *HInfinite-def HFinite-def linorder-not-less*)
apply (*rule-tac y*=$r + 1$ **in** *order-less-le-trans*, *simp*)
apply *simp*
done

lemma *HInfinite-inverse-Infinitesimal*:
fixes x :: $'a$::*real-normed-div-algebra star*
shows $x \in HInfinite \Longrightarrow inverse\ x \in Infinitesimal$
apply (*rule InfinitesimalI*)
apply (*subgoal-tac* $x \neq 0$)
apply (*rule inverse-less-imp-less*)
apply (*simp add*: *nonzero-hnorm-inverse*)
apply (*simp add*: *HInfinite-def Reals-inverse*)
apply *assumption*
apply (*clarify*, *simp add*: *Compl-HFinite* [*symmetric*])
done

lemma *HInfiniteI*: $(\bigwedge r.\ r \in \mathbb{R} \Longrightarrow r < hnorm\ x) \Longrightarrow x \in HInfinite$
by (*simp add*: *HInfinite-def*)

lemma *HInfiniteD*: $[\![x \in HInfinite;\ r \in \mathbb{R}]\!] \Longrightarrow r < hnorm\ x$
by (*simp add*: *HInfinite-def*)

lemma *HInfinite-mult*:
fixes $x\ y$ :: $'a$::*real-normed-div-algebra star*
shows $[|x \in HInfinite;\ y \in HInfinite|] \Longrightarrow (x*y) \in HInfinite$
apply (*rule HInfiniteI*, *simp only*: *hnorm-mult*)
apply (*subgoal-tac* $r * 1 < hnorm\ x * hnorm\ y$, *simp only*: *mult-1*)
apply (*case-tac* $x = 0$, *simp add*: *HInfinite-def*)
apply (*rule mult-strict-mono*)
apply (*simp-all add*: *HInfiniteD*)
done

lemma *hypreal-add-zero-less-le-mono*: $[|r < x;\ (0::hypreal) \leq y|] \Longrightarrow r < x+y$
by (*auto dest*: *add-less-le-mono*)

lemma *HInfinite-add-ge-zero*:
$[|(x::hypreal) \in HInfinite;\ 0 \leq y;\ 0 \leq x|] \Longrightarrow (x + y): HInfinite$
by (*auto intro*!: *hypreal-add-zero-less-le-mono*
simp add: *abs-if add-commute add-nonneg-nonneg HInfinite-def*)

lemma *HInfinite-add-ge-zero2*:
$[|(x::hypreal) \in HInfinite;\ 0 \leq y;\ 0 \leq x|] \Longrightarrow (y + x): HInfinite$
by (*auto intro*!: *HInfinite-add-ge-zero simp add*: *add-commute*)

lemma *HInfinite-add-gt-zero*:
$[|(x::hypreal) \in HInfinite;\ 0 < y;\ 0 < x|] \Longrightarrow (x + y): HInfinite$
by (*blast intro*: *HInfinite-add-ge-zero order-less-imp-le*)

lemma *HInfinite-minus-iff*: $(-x \in HInfinite) = (x \in HInfinite)$
by (*simp add*: *HInfinite-def*)

lemma *HInfinite-add-le-zero*:
$[|(x::hypreal) \in HInfinite; y \leq 0; x \leq 0|] ==> (x + y): HInfinite$
apply (*drule HInfinite-minus-iff* [*THEN iffD2*])
apply (*rule HInfinite-minus-iff* [*THEN iffD1*])
apply (*auto intro*: *HInfinite-add-ge-zero*)
done

lemma *HInfinite-add-lt-zero*:
$[|(x::hypreal) \in HInfinite; y < 0; x < 0|] ==> (x + y): HInfinite$
by (*blast intro*: *HInfinite-add-le-zero order-less-imp-le*)

lemma *HFinite-sum-squares*:
fixes *a b c* :: $'a::real\text{-}normed\text{-}algebra\ star$
shows $[|a: HFinite; b: HFinite; c: HFinite|]$
$==> a*a + b*b + c*c \in HFinite$
by (*auto intro*: *HFinite-mult HFinite-add*)

lemma *not-Infinitesimal-not-zero*: $x \notin Infinitesimal ==> x \neq 0$
by *auto*

lemma *not-Infinitesimal-not-zero2*: $x \in HFinite - Infinitesimal ==> x \neq 0$
by *auto*

lemma *HFinite-diff-Infinitesimal-hrabs*:
$(x::hypreal) \in HFinite - Infinitesimal ==> abs\ x \in HFinite - Infinitesimal$
by *blast*

lemma *hnorm-le-Infinitesimal*:
$[\![e \in Infinitesimal; hnorm\ x \leq e]\!] \Longrightarrow x \in Infinitesimal$
by (*auto simp add*: *Infinitesimal-def abs-less-iff*)

lemma *hnorm-less-Infinitesimal*:
$[\![e \in Infinitesimal; hnorm\ x < e]\!] \Longrightarrow x \in Infinitesimal$
by (*erule hnorm-le-Infinitesimal*, *erule order-less-imp-le*)

lemma *hrabs-le-Infinitesimal*:
$[|\ e \in Infinitesimal; abs\ (x::hypreal) \leq e\ |] ==> x \in Infinitesimal$
by (*erule hnorm-le-Infinitesimal*, *simp*)

lemma *hrabs-less-Infinitesimal*:
$[|\ e \in Infinitesimal; abs\ (x::hypreal) < e\ |] ==> x \in Infinitesimal$
by (*erule hnorm-less-Infinitesimal*, *simp*)

lemma *Infinitesimal-interval*:
$[|\ e \in Infinitesimal; e' \in Infinitesimal; e' < x\ ;\ x < e\ |]$

$==>$ $(x::hypreal) \in Infinitesimal$
by (*auto simp add*: *Infinitesimal-def abs-less-iff*)

lemma *Infinitesimal-interval2*:
$[|\ e \in Infinitesimal;\ e' \in Infinitesimal;$
$e' \leq x\ ;\ x \leq e\ |] ==> (x::hypreal) \in Infinitesimal$
by (*auto intro*: *Infinitesimal-interval simp add*: *order-le-less*)

lemma *lemma-Infinitesimal-hyperpow*:
$[|\ (x::hypreal) \in Infinitesimal;\ 0 < N\ |] ==> abs\ (x\ pow\ N) \leq abs\ x$
apply (*unfold Infinitesimal-def*)
apply (*auto intro*!: *hyperpow-Suc-le-self2*
simp add: *hyperpow-hrabs* [*symmetric*] *hypnat-gt-zero-iff2 abs-ge-zero*)
done

lemma *Infinitesimal-hyperpow*:
$[|\ (x::hypreal) \in Infinitesimal;\ 0 < N\ |] ==> x\ pow\ N \in Infinitesimal$
apply (*rule hrabs-le-Infinitesimal*)
apply (*rule-tac* [*2*] *lemma-Infinitesimal-hyperpow*, *auto*)
done

lemma *hrealpow-hyperpow-Infinitesimal-iff*:
$(x\ \hat{}\ n \in Infinitesimal) = (x\ pow\ (hypnat\text{-}of\text{-}nat\ n) \in Infinitesimal)$
by (*simp only*: *hyperpow-hypnat-of-nat*)

lemma *Infinitesimal-hrealpow*:
$[|\ (x::hypreal) \in Infinitesimal;\ 0 < n\ |] ==> x\ \hat{}\ n \in Infinitesimal$
by (*simp add*: *hrealpow-hyperpow-Infinitesimal-iff Infinitesimal-hyperpow*)

lemma *not-Infinitesimal-mult*:
fixes $x\ y :: 'a::real\text{-}normed\text{-}div\text{-}algebra\ star$
shows $[|\ x \notin Infinitesimal;\ \ y \notin Infinitesimal|] ==> (x*y) \notin Infinitesimal$
apply (*unfold Infinitesimal-def*, *clarify*, *rename-tac r s*)
apply (*simp only*: *linorder-not-less hnorm-mult*)
apply (*drule-tac* $x = r * s$ **in** *bspec*)
apply (*fast intro*: *Reals-mult*)
apply (*drule mp*, *blast intro*: *mult-pos-pos*)
apply (*drule-tac* $c = s$ **and** $d = hnorm\ y$ **and** $a = r$ **and** $b = hnorm\ x$ **in** *mult-mono*)
apply (*simp-all* (*no-asm-simp*))
done

lemma *Infinitesimal-mult-disj*:
fixes $x\ y :: 'a::real\text{-}normed\text{-}div\text{-}algebra\ star$
shows $x*y \in Infinitesimal ==> x \in Infinitesimal \mid y \in Infinitesimal$
apply (*rule ccontr*)
apply (*drule de-Morgan-disj* [*THEN iffD1*])
apply (*fast dest*: *not-Infinitesimal-mult*)

done

lemma *HFinite-Infinitesimal-not-zero*: $x \in HFinite - Infinitesimal \Longrightarrow x \neq 0$
by *blast*

lemma *HFinite-Infinitesimal-diff-mult*:
fixes $x\ y :: 'a$::*real-normed-div-algebra star*
shows $[\![x \in HFinite - Infinitesimal;$
$y \in HFinite - Infinitesimal$
$]\!] \Longrightarrow (x*y) \in HFinite - Infinitesimal$
apply *clarify*
apply (*blast dest*: *HFinite-mult not-Infinitesimal-mult*)
done

lemma *Infinitesimal-subset-HFinite*:
$Infinitesimal \subseteq HFinite$
apply (*simp add*: *Infinitesimal-def HFinite-def*, *auto*)
apply (*rule-tac* $x = 1$ **in** *bexI*, *auto*)
done

lemma *Infinitesimal-star-of-mult*:
fixes $x :: 'a$::*real-normed-algebra star*
shows $x \in Infinitesimal \Longrightarrow x * star\text{-}of\ r \in Infinitesimal$
by (*erule HFinite-star-of* [*THEN* [*2*] *Infinitesimal-HFinite-mult*])

lemma *Infinitesimal-star-of-mult2*:
fixes $x :: 'a$::*real-normed-algebra star*
shows $x \in Infinitesimal \Longrightarrow star\text{-}of\ r * x \in Infinitesimal$
by (*erule HFinite-star-of* [*THEN* [*2*] *Infinitesimal-HFinite-mult2*])

37.5 The Infinitely Close Relation

lemma *mem-infmal-iff*: $(x \in Infinitesimal) = (x\ @=\ 0)$
by (*simp add*: *Infinitesimal-def approx-def*)

lemma *approx-minus-iff*: $(x\ @=\ y) = (x - y\ @=\ 0)$
by (*simp add*: *approx-def*)

lemma *approx-minus-iff2*: $(x\ @=\ y) = (-y + x\ @=\ 0)$
by (*simp add*: *approx-def diff-minus add-commute*)

lemma *approx-refl* [*iff*]: $x\ @=\ x$
by (*simp add*: *approx-def Infinitesimal-def*)

lemma *hypreal-minus-distrib1*: $-(y + -(x::'a::ab\text{-}group\text{-}add)) = x + -y$
by (*simp add*: *add-commute*)

lemma *approx-sym*: $x\ @=\ y \Longrightarrow y\ @=\ x$
apply (*simp add*: *approx-def*)

apply (*drule Infinitesimal-minus-iff* [*THEN iffD2*])
apply *simp*
done

lemma *approx-trans*: [| *x* @= *y*; *y* @= *z* |] ==> *x* @= *z*
apply (*simp add*: *approx-def*)
apply (*drule* (*1*) *Infinitesimal-add*)
apply (*simp add*: *diff-def*)
done

lemma *approx-trans2*: [| *r* @= *x*; *s* @= *x* |] ==> *r* @= *s*
by (*blast intro*: *approx-sym approx-trans*)

lemma *approx-trans3*: [| *x* @= *r*; *x* @= *s*|] ==> *r* @= *s*
by (*blast intro*: *approx-sym approx-trans*)

lemma *number-of-approx-reorient*: (*number-of w* @= *x*) = (*x* @= *number-of w*)
by (*blast intro*: *approx-sym*)

lemma *zero-approx-reorient*: (*0* @= *x*) = (*x* @= *0*)
by (*blast intro*: *approx-sym*)

lemma *one-approx-reorient*: (*1* @= *x*) = (*x* @= *1*)
by (*blast intro*: *approx-sym*)

ML ⟪
local
(*** *re−orientation, following HOL/Integ/Bin.ML*
    We re−orient x @=y where x is 0, 1 or a numeral, unless y is as well!
 ***)

(*∗reorientation simprules using ==, for the following simproc∗*)
val meta-zero-approx-reorient = thm zero-approx-reorient RS eq-reflection;
val meta-one-approx-reorient = thm one-approx-reorient RS eq-reflection;
val meta-number-of-approx-reorient = thm number-of-approx-reorient RS eq-reflection

(*∗reorientation simplification procedure: reorients (polymorphic)
  0 = x, 1 = x, nnn = x provided x isn't 0, 1 or a numeral.∗*)
fun reorient-proc sg - (- $ t $ u) =
  case u of
      Const(@{*const-name HOL.zero*}, -) => *NONE*
    | *Const*(@{*const-name HOL.one*}, -) => *NONE*
    | *Const*(@{*const-name Int.number-of*}, -) $ - => *NONE*
    | - => *SOME* (*case t of*
              Const(@{*const-name HOL.zero*}, -) => *meta-zero-approx-reorient*
            | *Const*(@{*const-name HOL.one*}, -) => *meta-one-approx-reorient*
            | *Const*(@{*const-name Int.number-of*}, -) $ - =>
                        meta-number-of-approx-reorient);

in
val approx-reorient-simproc =
Int-Numeral-Base-Simprocs.prep-simproc
(*reorient-simproc*, [*0@=x*, *1@=x*, *number-of w @= x*], *reorient-proc*);
end;

Addsimprocs [*approx-reorient-simproc*];
⟩⟩

lemma *Infinitesimal-approx-minus*: $(x-y \in Infinitesimal) = (x \mathbin{@=} y)$
by (*simp add*: *approx-minus-iff* [*symmetric*] *mem-infmal-iff*)

lemma *approx-monad-iff*: $(x \mathbin{@=} y) = (monad(x)=monad(y))$
apply (*simp add*: *monad-def*)
apply (*auto dest*: *approx-sym elim*!: *approx-trans equalityCE*)
done

lemma *Infinitesimal-approx*:
[| $x \in Infinitesimal$; $y \in Infinitesimal$ |] ==> $x \mathbin{@=} y$
apply (*simp add*: *mem-infmal-iff*)
apply (*blast intro*: *approx-trans approx-sym*)
done

lemma *approx-add*: [| $a \mathbin{@=} b$; $c \mathbin{@=} d$ |] ==> $a+c \mathbin{@=} b+d$
proof (*unfold approx-def*)
assume *inf*: $a - b \in Infinitesimal$ $c - d \in Infinitesimal$
have $a + c - (b + d) = (a - b) + (c - d)$ **by** *simp*
also have ... $\in Infinitesimal$ **using** *inf* **by** (*rule Infinitesimal-add*)
finally show $a + c - (b + d) \in Infinitesimal$ **.**
qed

lemma *approx-minus*: $a \mathbin{@=} b$ ==> $-a \mathbin{@=} -b$
apply (*rule approx-minus-iff* [*THEN iffD2*, *THEN approx-sym*])
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*simp add*: *add-commute diff-def*)
done

lemma *approx-minus2*: $-a \mathbin{@=} -b$ ==> $a \mathbin{@=} b$
by (*auto dest*: *approx-minus*)

lemma *approx-minus-cancel* [*simp*]: $(-a \mathbin{@=} -b) = (a \mathbin{@=} b)$
by (*blast intro*: *approx-minus approx-minus2*)

lemma *approx-add-minus*: [| $a \mathbin{@=} b$; $c \mathbin{@=} d$ |] ==> $a + -c \mathbin{@=} b + -d$
by (*blast intro*!: *approx-add approx-minus*)

lemma *approx-diff*: [| $a \mathbin{@=} b$; $c \mathbin{@=} d$ |] ==> $a - c \mathbin{@=} b - d$
by (*simp only*: *diff-minus approx-add approx-minus*)

lemma *approx-mult1*:
fixes *a b c* :: *'a*::*real-normed-algebra star*
shows [| *a* @= *b*; *c*: *HFinite*|] ==> *a*∗*c* @= *b*∗*c*
by (*simp add*: *approx-def Infinitesimal-HFinite-mult*
left-diff-distrib [*symmetric*])

lemma *approx-mult2*:
fixes *a b c* :: *'a*::*real-normed-algebra star*
shows [|*a* @= *b*; *c*: *HFinite*|] ==> *c*∗*a* @= *c*∗*b*
by (*simp add*: *approx-def Infinitesimal-HFinite-mult2*
right-diff-distrib [*symmetric*])

lemma *approx-mult-subst*:
fixes *u v x y* :: *'a*::*real-normed-algebra star*
shows [|*u* @= *v*∗*x*; *x* @= *y*; *v* ∈ *HFinite*|] ==> *u* @= *v*∗*y*
by (*blast intro*: *approx-mult2 approx-trans*)

lemma *approx-mult-subst2*:
fixes *u v x y* :: *'a*::*real-normed-algebra star*
shows [| *u* @= *x*∗*v*; *x* @= *y*; *v* ∈ *HFinite* |] ==> *u* @= *y*∗*v*
by (*blast intro*: *approx-mult1 approx-trans*)

lemma *approx-mult-subst-star-of*:
fixes *u x y* :: *'a*::*real-normed-algebra star*
shows [| *u* @= *x*∗*star-of v*; *x* @= *y* |] ==> *u* @= *y*∗*star-of v*
by (*auto intro*: *approx-mult-subst2*)

lemma *approx-eq-imp*: *a* = *b* ==> *a* @= *b*
by (*simp add*: *approx-def*)

lemma *Infinitesimal-minus-approx*: *x* ∈ *Infinitesimal* ==> −*x* @= *x*
by (*blast intro*: *Infinitesimal-minus-iff* [*THEN iffD2*]
mem-infmal-iff [*THEN iffD1*] *approx-trans2*)

lemma *bex-Infinitesimal-iff*: (∃ *y* ∈ *Infinitesimal*. *x* − *z* = *y*) = (*x* @= *z*)
by (*simp add*: *approx-def*)

lemma *bex-Infinitesimal-iff2*: (∃ *y* ∈ *Infinitesimal*. *x* = *z* + *y*) = (*x* @= *z*)
by (*force simp add*: *bex-Infinitesimal-iff* [*symmetric*])

lemma *Infinitesimal-add-approx*: [| *y* ∈ *Infinitesimal*; *x* + *y* = *z* |] ==> *x* @= *z*
apply (*rule bex-Infinitesimal-iff* [*THEN iffD1*])
apply (*drule Infinitesimal-minus-iff* [*THEN iffD2*])
apply (*auto simp add*: *add-assoc* [*symmetric*])
done

lemma *Infinitesimal-add-approx-self*: *y* ∈ *Infinitesimal* ==> *x* @= *x* + *y*
apply (*rule bex-Infinitesimal-iff* [*THEN iffD1*])

apply (*drule Infinitesimal-minus-iff* [*THEN iffD2*])
apply (*auto simp add*: *add-assoc* [*symmetric*])
done

lemma *Infinitesimal-add-approx-self2*: $y \in Infinitesimal ==> x \mathbin{@=} y + x$
by (*auto dest*: *Infinitesimal-add-approx-self simp add*: *add-commute*)

lemma *Infinitesimal-add-minus-approx-self*: $y \in Infinitesimal ==> x \mathbin{@=} x + -y$
by (*blast intro*!: *Infinitesimal-add-approx-self Infinitesimal-minus-iff* [*THEN iffD2*])

lemma *Infinitesimal-add-cancel*: $[|\ y \in Infinitesimal;\ x+y \mathbin{@=} z|] ==> x \mathbin{@=} z$
apply (*drule-tac* $x = x$ **in** *Infinitesimal-add-approx-self* [*THEN approx-sym*])
apply (*erule approx-trans3* [*THEN approx-sym*], *assumption*)
done

lemma *Infinitesimal-add-right-cancel*:
$[|\ y \in Infinitesimal;\ x \mathbin{@=} z + y|] ==> x \mathbin{@=} z$
apply (*drule-tac* $x = z$ **in** *Infinitesimal-add-approx-self2* [*THEN approx-sym*])
apply (*erule approx-trans3* [*THEN approx-sym*])
apply (*simp add*: *add-commute*)
apply (*erule approx-sym*)
done

lemma *approx-add-left-cancel*: $d + b \mathbin{@=} d + c ==> b \mathbin{@=} c$
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*simp add*: *approx-minus-iff* [*symmetric*] *add-ac*)
done

lemma *approx-add-right-cancel*: $b + d \mathbin{@=} c + d ==> b \mathbin{@=} c$
apply (*rule approx-add-left-cancel*)
apply (*simp add*: *add-commute*)
done

lemma *approx-add-mono1*: $b \mathbin{@=} c ==> d + b \mathbin{@=} d + c$
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*simp add*: *approx-minus-iff* [*symmetric*] *add-ac*)
done

lemma *approx-add-mono2*: $b \mathbin{@=} c ==> b + a \mathbin{@=} c + a$
by (*simp add*: *add-commute approx-add-mono1*)

lemma *approx-add-left-iff* [*simp*]: $(a + b \mathbin{@=} a + c) = (b \mathbin{@=} c)$
by (*fast elim*: *approx-add-left-cancel approx-add-mono1*)

lemma *approx-add-right-iff* [*simp*]: $(b + a \mathbin{@=} c + a) = (b \mathbin{@=} c)$
by (*simp add*: *add-commute*)

lemma *approx-HFinite*: $[|\ x \in HFinite;\ x \mathbin{@=} y\ |] ==> y \in HFinite$
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*], *safe*)

apply (*drule Infinitesimal-subset-HFinite* [*THEN subsetD*, *THEN HFinite-minus-iff* [*THEN iffD2*]])
apply (*drule HFinite-add*)
apply (*auto simp add*: *add-assoc*)
done

lemma *approx-star-of-HFinite*: *x* @= *star-of D* ==> *x* ∈ *HFinite*
by (*rule approx-sym* [*THEN* [*2*] *approx-HFinite*], *auto*)

lemma *approx-mult-HFinite*:
fixes *a b c d* :: *'a*::*real-normed-algebra star*
shows [|*a* @= *b*; *c* @= *d*; *b*: *HFinite*; *d*: *HFinite*|] ==> *a*∗*c* @= *b*∗*d*
apply (*rule approx-trans*)
apply (*rule-tac* [*2*] *approx-mult2*)
apply (*rule approx-mult1*)
prefer *2* **apply** (*blast intro*: *approx-HFinite approx-sym*, *auto*)
done

lemma *scaleHR-left-diff-distrib*:
⋀*a b x*. *scaleHR* (*a* − *b*) *x* = *scaleHR a x* − *scaleHR b x*
by *transfer* (*rule scaleR-left-diff-distrib*)

lemma *approx-scaleR1*:
[| *a* @= *star-of b*; *c*: *HFinite*|] ==> *scaleHR a c* @= *b* ∗$_R$ *c*
apply (*unfold approx-def*)
apply (*drule* (*1*) *Infinitesimal-HFinite-scaleHR*)
apply (*simp only*: *scaleHR-left-diff-distrib*)
apply (*simp add*: *scaleHR-def star-scaleR-def* [*symmetric*])
done

lemma *approx-scaleR2*:
a @= *b* ==> *c* ∗$_R$ *a* @= *c* ∗$_R$ *b*
by (*simp add*: *approx-def Infinitesimal-scaleR2*
scaleR-right-diff-distrib [*symmetric*])

lemma *approx-scaleR-HFinite*:
[|*a* @= *star-of b*; *c* @= *d*; *d*: *HFinite*|] ==> *scaleHR a c* @= *b* ∗$_R$ *d*
apply (*rule approx-trans*)
apply (*rule-tac* [*2*] *approx-scaleR2*)
apply (*rule approx-scaleR1*)
prefer *2* **apply** (*blast intro*: *approx-HFinite approx-sym*, *auto*)
done

lemma *approx-mult-star-of*:
fixes *a c* :: *'a*::*real-normed-algebra star*
shows [|*a* @= *star-of b*; *c* @= *star-of d* |]
==> *a*∗*c* @= *star-of b*∗*star-of d*
by (*blast intro*!: *approx-mult-HFinite approx-star-of-HFinite HFinite-star-of*)

lemma *approx-SReal-mult-cancel-zero*:
[| (*a*::*hypreal*) ∈ *Reals*; *a* ≠ *0*; *a*∗*x* @= *0* |] ==> *x* @= *0*
apply (*drule Reals-inverse* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult-assoc* [*symmetric*])
done

lemma *approx-mult-SReal1*: [| (*a*::*hypreal*) ∈ *Reals*; *x* @= *0* |] ==> *x*∗*a* @= *0*
by (*auto dest*: *SReal-subset-HFinite* [*THEN subsetD*] *approx-mult1*)

lemma *approx-mult-SReal2*: [| (*a*::*hypreal*) ∈ *Reals*; *x* @= *0* |] ==> *a*∗*x* @= *0*
by (*auto dest*: *SReal-subset-HFinite* [*THEN subsetD*] *approx-mult2*)

lemma *approx-mult-SReal-zero-cancel-iff* [*simp*]:
[|(*a*::*hypreal*) ∈ *Reals*; *a* ≠ *0* |] ==> (*a*∗*x* @= *0*) = (*x* @= *0*)
by (*blast intro*: *approx-SReal-mult-cancel-zero approx-mult-SReal2*)

lemma *approx-SReal-mult-cancel*:
[| (*a*::*hypreal*) ∈ *Reals*; *a* ≠ *0*; *a*∗ *w* @= *a*∗*z* |] ==> *w* @= *z*
apply (*drule Reals-inverse* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult-assoc* [*symmetric*])
done

lemma *approx-SReal-mult-cancel-iff1* [*simp*]:
[| (*a*::*hypreal*) ∈ *Reals*; *a* ≠ *0*|] ==> (*a*∗ *w* @= *a*∗*z*) = (*w* @= *z*)
by (*auto intro*!: *approx-mult2 SReal-subset-HFinite* [*THEN subsetD*]
intro: *approx-SReal-mult-cancel*)

lemma *approx-le-bound*: [| (*z*::*hypreal*) ≤ *f*; *f* @= *g*; *g* ≤ *z* |] ==> *f* @= *z*
apply (*simp add*: *bex-Infinitesimal-iff2* [*symmetric*], *auto*)
apply (*rule-tac x* = *g*+*y*−*z* **in** *bexI*)
apply (*simp* (*no-asm*))
apply (*rule Infinitesimal-interval2*)
apply (*rule-tac* [*2*] *Infinitesimal-zero*, *auto*)
done

lemma *approx-hnorm*:
fixes *x y* :: *'a*::*real-normed-vector star*
shows *x* ≈ *y* ⟹ *hnorm x* ≈ *hnorm y*
proof (*unfold approx-def*)
assume *x* − *y* ∈ *Infinitesimal*
hence *1*: *hnorm* (*x* − *y*) ∈ *Infinitesimal*
by (*simp only*: *Infinitesimal-hnorm-iff*)
moreover have *2*: (*0*::*real star*) ∈ *Infinitesimal*
by (*rule Infinitesimal-zero*)
moreover have *3*: *0* ≤ |*hnorm x* − *hnorm y*|
by (*rule abs-ge-zero*)
moreover have *4*: |*hnorm x* − *hnorm y*| ≤ *hnorm* (*x* − *y*)
by (*rule hnorm-triangle-ineq3*)
ultimately have |*hnorm x* − *hnorm y*| ∈ *Infinitesimal*

by (*rule Infinitesimal-interval2*)
thus *hnorm x − hnorm y ∈ Infinitesimal*
by (*simp only*: *Infinitesimal-hrabs-iff*)
qed

37.6 Zero is the Only Infinitesimal that is also a Real

lemma *Infinitesimal-less-SReal*:
[| (*x*::*hypreal*) ∈ *Reals*; *y* ∈ *Infinitesimal*; *0* < *x* |] ==> *y* < *x*
apply (*simp add*: *Infinitesimal-def*)
apply (*rule abs-ge-self* [*THEN order-le-less-trans*], *auto*)
done

lemma *Infinitesimal-less-SReal2*:
(*y*::*hypreal*) ∈ *Infinitesimal* ==> ∀ *r* ∈ *Reals*. *0* < *r* −−> *y* < *r*
by (*blast intro*: *Infinitesimal-less-SReal*)

lemma *SReal-not-Infinitesimal*:
[| *0* < *y*; (*y*::*hypreal*) ∈ *Reals*|] ==> *y* ∉ *Infinitesimal*
apply (*simp add*: *Infinitesimal-def*)
apply (*auto simp add*: *abs-if*)
done

lemma *SReal-minus-not-Infinitesimal*:
[| *y* < *0*; (*y*::*hypreal*) ∈ *Reals* |] ==> *y* ∉ *Infinitesimal*
apply (*subst Infinitesimal-minus-iff* [*symmetric*])
apply (*rule SReal-not-Infinitesimal*, *auto*)
done

lemma *SReal-Int-Infinitesimal-zero*: *Reals Int Infinitesimal* = {*0*::*hypreal*}
apply *auto*
apply (*cut-tac x* = *x* **and** *y* = *0* **in** *linorder-less-linear*)
apply (*blast dest*: *SReal-not-Infinitesimal SReal-minus-not-Infinitesimal*)
done

lemma *SReal-Infinitesimal-zero*:
[| (*x*::*hypreal*) ∈ *Reals*; *x* ∈ *Infinitesimal*|] ==> *x* = *0*
by (*cut-tac SReal-Int-Infinitesimal-zero*, *blast*)

lemma *SReal-HFinite-diff-Infinitesimal*:
[| (*x*::*hypreal*) ∈ *Reals*; *x* ≠ *0* |] ==> *x* ∈ *HFinite* − *Infinitesimal*
by (*auto dest*: *SReal-Infinitesimal-zero SReal-subset-HFinite* [*THEN subsetD*])

lemma *hypreal-of-real-HFinite-diff-Infinitesimal*:
hypreal-of-real x ≠ *0* ==> *hypreal-of-real x* ∈ *HFinite* − *Infinitesimal*
by (*rule SReal-HFinite-diff-Infinitesimal*, *auto*)

lemma *star-of-Infinitesimal-iff-0* [*iff*]:
(*star-of x* ∈ *Infinitesimal*) = (*x* = *0*)

apply (*auto simp add*: *Infinitesimal-def*)
apply (*drule-tac x=hnorm* (*star-of x*) **in** *bspec*)
apply (*simp add*: *SReal-def*)
apply (*rule-tac x=norm x* **in** *exI*, *simp*)
apply *simp*
done

lemma *star-of-HFinite-diff-Infinitesimal*:
x ≠ *0* ==> *star-of x* ∈ *HFinite* − *Infinitesimal*
by *simp*

lemma *number-of-not-Infinitesimal* [*simp*]:
number-of w ≠ (*0*::*hypreal*) ==> (*number-of w* :: *hypreal*) ∉ *Infinitesimal*
by (*fast dest*: *Reals-number-of* [*THEN SReal-Infinitesimal-zero*])

lemma *one-not-Infinitesimal* [*simp*]:
(*1*::'*a*::{*real-normed-vector*,*zero-neq-one*} *star*) ∉ *Infinitesimal*
apply (*simp only*: *star-one-def star-of-Infinitesimal-iff-0*)
apply *simp*
done

lemma *approx-SReal-not-zero*:
[| (*y*::*hypreal*) ∈ *Reals*; *x* @= *y*; *y*≠ *0* |] ==> *x* ≠ *0*
apply (*cut-tac x* = *0* **and** *y* = *y* **in** *linorder-less-linear*, *simp*)
apply (*blast dest*: *approx-sym* [*THEN mem-infmal-iff* [*THEN iffD2*]] *SReal-not-Infinitesimal SReal-minus-not-Infinitesimal*)
done

lemma *HFinite-diff-Infinitesimal-approx*:
[| *x* @= *y*; *y* ∈ *HFinite* − *Infinitesimal* |]
==> *x* ∈ *HFinite* − *Infinitesimal*
apply (*auto intro*: *approx-sym* [*THEN* [*2*] *approx-HFinite*]
simp add: *mem-infmal-iff*)
apply (*drule approx-trans3*, *assumption*)
apply (*blast dest*: *approx-sym*)
done

lemma *Infinitesimal-ratio*:
fixes *x y* :: '*a*::{*real-normed-div-algebra*,*field*} *star*
shows [| *y* ≠ *0*; *y* ∈ *Infinitesimal*; *x*/*y* ∈ *HFinite* |]
==> *x* ∈ *Infinitesimal*
apply (*drule Infinitesimal-HFinite-mult2*, *assumption*)
apply (*simp add*: *divide-inverse mult-assoc*)
done

lemma *Infinitesimal-SReal-divide*:
[| (*x*::*hypreal*) ∈ *Infinitesimal*; *y* ∈ *Reals* |] ==> *x*/*y* ∈ *Infinitesimal*

apply (*simp add*: *divide-inverse*)
apply (*auto intro*!: *Infinitesimal-HFinite-mult*
dest!: *Reals-inverse* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])
done

37.7 Uniqueness: Two Infinitely Close Reals are Equal

lemma *star-of-approx-iff* [*simp*]: (*star-of x* @= *star-of y*) = (*x* = *y*)
apply *safe*
apply (*simp add*: *approx-def*)
apply (*simp only*: *star-of-diff* [*symmetric*])
apply (*simp only*: *star-of-Infinitesimal-iff-0*)
apply *simp*
done

lemma *SReal-approx-iff*:
[|(*x*::*hypreal*) ∈ *Reals*; *y* ∈ *Reals*|] ==> (*x* @= *y*) = (*x* = *y*)
apply *auto*
apply (*simp add*: *approx-def*)
apply (*drule* (*1*) *Reals-diff*)
apply (*drule* (*1*) *SReal-Infinitesimal-zero*)
apply *simp*
done

lemma *number-of-approx-iff* [*simp*]:
(*number-of v* @= (*number-of w* :: *'a*::{*number*,*real-normed-vector*} *star*)) =
(*number-of v* = (*number-of w* :: *'a*))
apply (*unfold star-number-def*)
apply (*rule star-of-approx-iff*)
done

lemma [*simp*]:
(*number-of w* @= (*0*::*'a*::{*number*,*real-normed-vector*} *star*)) =
(*number-of w* = (*0*::*'a*))
((*0*::*'a*::{*number*,*real-normed-vector*} *star*) @= *number-of w*) =
(*number-of w* = (*0*::*'a*))
(*number-of w* @= (*1*::*'b*::{*number*,*one*,*real-normed-vector*} *star*)) =
(*number-of w* = (*1*::*'b*))
((*1*::*'b*::{*number*,*one*,*real-normed-vector*} *star*) @= *number-of w*) =
(*number-of w* = (*1*::*'b*))
~ (*0* @= (*1*::*'c*::{*zero-neq-one*,*real-normed-vector*} *star*))
~ (*1* @= (*0*::*'c*::{*zero-neq-one*,*real-normed-vector*} *star*))
apply (*unfold star-number-def star-zero-def star-one-def*)
apply (*unfold star-of-approx-iff*)
by (*auto intro*: *sym*)

lemma *star-of-approx-number-of-iff* [*simp*]:
(*star-of k* @= *number-of w*) = (*k* = *number-of w*)

by (*subst star-of-approx-iff* [*symmetric*], *auto*)

lemma *star-of-approx-zero-iff* [*simp*]: (*star-of k* @= *0*) = (*k* = *0*)
by (*simp-all add*: *star-of-approx-iff* [*symmetric*])

lemma *star-of-approx-one-iff* [*simp*]: (*star-of k* @= *1*) = (*k* = *1*)
by (*simp-all add*: *star-of-approx-iff* [*symmetric*])

lemma *approx-unique-real*:
[| (*r*::*hypreal*) ∈ *Reals*; *s* ∈ *Reals*; *r* @= *x*; *s* @= *x*|] ==> *r* = *s*
by (*blast intro*: *SReal-approx-iff* [*THEN iffD1*] *approx-trans2*)

37.8 Existence of Unique Real Infinitely Close

37.8.1 Lifting of the Ub and Lub Properties

lemma *hypreal-of-real-isUb-iff*:
(*isUb* (*Reals*) (*hypreal-of-real* ' *Q*) (*hypreal-of-real Y*)) =
(*isUb* (*UNIV* :: *real set*) *Q Y*)
by (*simp add*: *isUb-def setle-def*)

lemma *hypreal-of-real-isLub1*:
isLub Reals (*hypreal-of-real* ' *Q*) (*hypreal-of-real Y*)
==> *isLub* (*UNIV* :: *real set*) *Q Y*
apply (*simp add*: *isLub-def leastP-def*)
apply (*auto intro*: *hypreal-of-real-isUb-iff* [*THEN iffD2*]
simp add: *hypreal-of-real-isUb-iff setge-def*)
done

lemma *hypreal-of-real-isLub2*:
isLub (*UNIV* :: *real set*) *Q Y*
==> *isLub Reals* (*hypreal-of-real* ' *Q*) (*hypreal-of-real Y*)
apply (*simp add*: *isLub-def leastP-def*)
apply (*auto simp add*: *hypreal-of-real-isUb-iff setge-def*)
apply (*frule-tac x2* = *x* **in** *isUbD2a* [*THEN SReal-iff* [*THEN iffD1*], *THEN exE*])
prefer *2* **apply** *assumption*
apply (*drule-tac x* = *xa* **in** *spec*)
apply (*auto simp add*: *hypreal-of-real-isUb-iff*)
done

lemma *hypreal-of-real-isLub-iff*:
(*isLub Reals* (*hypreal-of-real* ' *Q*) (*hypreal-of-real Y*)) =
(*isLub* (*UNIV* :: *real set*) *Q Y*)
by (*blast intro*: *hypreal-of-real-isLub1 hypreal-of-real-isLub2*)

lemma *lemma-isUb-hypreal-of-real*:
isUb Reals P Y ==> ∃ *Yo*. *isUb Reals P* (*hypreal-of-real Yo*)
by (*auto simp add*: *SReal-iff isUb-def*)

lemma *lemma-isLub-hypreal-of-real*:

isLub Reals P Y ==> ∃ Yo. isLub Reals P (hypreal-of-real Yo)
by (*auto simp add*: *isLub-def leastP-def isUb-def SReal-iff*)

lemma *lemma-isLub-hypreal-of-real2*:
∃ Yo. isLub Reals P (hypreal-of-real Yo) ==> ∃ Y. isLub Reals P Y
by (*auto simp add*: *isLub-def leastP-def isUb-def*)

lemma *SReal-complete*:
[| P ⊆ Reals; ∃ x. x ∈ P; ∃ Y. isUb Reals P Y |]
==> ∃ t::hypreal. isLub Reals P t
apply (*frule SReal-hypreal-of-real-image*)
apply (*auto*, *drule lemma-isUb-hypreal-of-real*)
apply (*auto intro*!: *reals-complete lemma-isLub-hypreal-of-real2*
simp add: *hypreal-of-real-isLub-iff hypreal-of-real-isUb-iff*)
done

lemma *hypreal-isLub-unique*:
[| isLub R S x; isLub R S y |] ==> x = (y::hypreal)
apply (*frule isLub-isUb*)
apply (*frule-tac x = y* **in** *isLub-isUb*)
apply (*blast intro*!: *order-antisym dest*!: *isLub-le-isUb*)
done

lemma *lemma-st-part-ub*:
(x::hypreal) ∈ HFinite ==> ∃ u. isUb Reals {s. s ∈ Reals & s < x} u
apply (*drule HFiniteD*, *safe*)
apply (*rule exI*, *rule isUbI*)
apply (*auto intro*: *setleI isUbI simp add*: *abs-less-iff*)
done

lemma *lemma-st-part-nonempty*:
(x::hypreal) ∈ HFinite ==> ∃ y. y ∈ {s. s ∈ Reals & s < x}
apply (*drule HFiniteD*, *safe*)
apply (*drule Reals-minus*)
apply (*rule-tac x = −t* **in** *exI*)
apply (*auto simp add*: *abs-less-iff*)
done

lemma *lemma-st-part-subset*: *{s. s ∈ Reals & s < x} ⊆ Reals*
by *auto*

lemma *lemma-st-part-lub*:
(x::hypreal) ∈ HFinite ==> ∃ t. isLub Reals {s. s ∈ Reals & s < x} t
by (*blast intro*!: *SReal-complete lemma-st-part-ub lemma-st-part-nonempty lemma-st-part-subset*)

lemma *lemma-hypreal-le-left-cancel*: *((t::hypreal) + r ≤ t) = (r ≤ 0)*
apply *safe*
apply (*drule-tac c = −t* **in** *add-left-mono*)

apply (*drule-tac* [*2*] $c = t$ **in** *add-left-mono*)
apply (*auto simp add*: *add-assoc* [*symmetric*])
done

lemma *lemma-st-part-le1*:
[| (x::*hypreal*) ∈ *HFinite*; *isLub Reals* {s. $s \in$ *Reals* & $s < x$} t;
$r \in$ *Reals*; $0 < r$ |] ==> $x \leq t + r$
apply (*frule isLubD1a*)
apply (*rule ccontr*, *drule linorder-not-le* [*THEN iffD2*])
apply (*drule* (*1*) *Reals-add*)
apply (*drule-tac* $y = r + t$ **in** *isLubD1* [*THEN setleD*], *auto*)
done

lemma *hypreal-setle-less-trans*:
[| S *<=* (x::*hypreal*); $x < y$ |] ==> S *<=* y
apply (*simp add*: *setle-def*)
apply (*auto dest*!: *bspec order-le-less-trans intro*: *order-less-imp-le*)
done

lemma *hypreal-gt-isUb*:
[| *isUb R S* (x::*hypreal*); $x < y$; $y \in R$ |] ==> *isUb R S* y
apply (*simp add*: *isUb-def*)
apply (*blast intro*: *hypreal-setle-less-trans*)
done

lemma *lemma-st-part-gt-ub*:
[| (x::*hypreal*) ∈ *HFinite*; $x < y$; $y \in$ *Reals* |]
==> *isUb Reals* {s. $s \in$ *Reals* & $s < x$} y
by (*auto dest*: *order-less-trans intro*: *order-less-imp-le intro*!: *isUbI setleI*)

lemma *lemma-minus-le-zero*: $t \leq t + -r$ ==> $r \leq$ (*0*::*hypreal*)
apply (*drule-tac* $c = -t$ **in** *add-left-mono*)
apply (*auto simp add*: *add-assoc* [*symmetric*])
done

lemma *lemma-st-part-le2*:
[| (x::*hypreal*) ∈ *HFinite*;
isLub Reals {s. $s \in$ *Reals* & $s < x$} t;
$r \in$ *Reals*; $0 < r$ |]
==> $t + -r \leq x$
apply (*frule isLubD1a*)
apply (*rule ccontr*, *drule linorder-not-le* [*THEN iffD1*])
apply (*drule Reals-minus*, *drule-tac* $a = t$ **in** *Reals-add*, *assumption*)
apply (*drule lemma-st-part-gt-ub*, *assumption+*)
apply (*drule isLub-le-isUb*, *assumption*)
apply (*drule lemma-minus-le-zero*)
apply (*auto dest*: *order-less-le-trans*)
done

lemma *lemma-st-part1a*:
[| (*x*::*hypreal*) ∈ *HFinite*;
isLub Reals {*s*. *s* ∈ *Reals* & *s* < *x*} *t*;
r ∈ *Reals*; *0* < *r* |]
==> *x* + −*t* ≤ *r*
apply (*subgoal-tac x* ≤ *t*+*r*)
apply (*auto intro*: *lemma-st-part-le1*)
done

lemma *lemma-st-part2a*:
[| (*x*::*hypreal*) ∈ *HFinite*;
isLub Reals {*s*. *s* ∈ *Reals* & *s* < *x*} *t*;
r ∈ *Reals*; *0* < *r* |]
==> −(*x* + −*t*) ≤ *r*
apply (*subgoal-tac* (*t* + −*r* ≤ *x*))
apply (*auto intro*: *lemma-st-part-le2*)
done

lemma *lemma-SReal-ub*:
(*x*::*hypreal*) ∈ *Reals* ==> *isUb Reals* {*s*. *s* ∈ *Reals* & *s* < *x*} *x*
by (*auto intro*: *isUbI setleI order-less-imp-le*)

lemma *lemma-SReal-lub*:
(*x*::*hypreal*) ∈ *Reals* ==> *isLub Reals* {*s*. *s* ∈ *Reals* & *s* < *x*} *x*
apply (*auto intro*!: *isLubI2 lemma-SReal-ub setgeI*)
apply (*frule isUbD2a*)
apply (*rule-tac x* = *x* **and** *y* = *y* **in** *linorder-cases*)
apply (*auto intro*!: *order-less-imp-le*)
apply (*drule SReal-dense*, *assumption*, *assumption*, *safe*)
apply (*drule-tac y* = *r* **in** *isUbD*)
apply (*auto dest*: *order-less-le-trans*)
done

lemma *lemma-st-part-not-eq1*:
[| (*x*::*hypreal*) ∈ *HFinite*;
isLub Reals {*s*. *s* ∈ *Reals* & *s* < *x*} *t*;
r ∈ *Reals*; *0* < *r* |]
==> *x* + −*t* ≠ *r*
apply *auto*
apply (*frule isLubD1a* [*THEN Reals-minus*])
apply (*drule Reals-add-cancel*, *assumption*)
apply (*drule-tac x* = *x* **in** *lemma-SReal-lub*)
apply (*drule hypreal-isLub-unique*, *assumption*, *auto*)
done

lemma *lemma-st-part-not-eq2*:
[| (*x*::*hypreal*) ∈ *HFinite*;
isLub Reals {*s*. *s* ∈ *Reals* & *s* < *x*} *t*;
r ∈ *Reals*; *0* < *r* |]

```
    ==> -(x + -t) ≠ r
apply (auto)
apply (frule isLubD1a)
apply (drule Reals-add-cancel, assumption)
apply (drule-tac a = -x in Reals-minus, simp)
apply (drule-tac x = x in lemma-SReal-lub)
apply (drule hypreal-isLub-unique, assumption, auto)
done

lemma lemma-st-part-major:
    [| (x::hypreal) ∈ HFinite;
       isLub Reals {s. s ∈ Reals & s < x} t;
       r ∈ Reals; 0 < r |]
    ==> abs (x - t) < r
apply (frule lemma-st-part1a)
apply (frule-tac [4] lemma-st-part2a, auto)
apply (drule order-le-imp-less-or-eq)+
apply (auto dest: lemma-st-part-not-eq1 lemma-st-part-not-eq2 simp add: abs-less-iff)
done

lemma lemma-st-part-major2:
    [| (x::hypreal) ∈ HFinite; isLub Reals {s. s ∈ Reals & s < x} t |]
    ==> ∀ r ∈ Reals. 0 < r --> abs (x - t) < r
by (blast dest!: lemma-st-part-major)
```

Existence of real and Standard Part Theorem

```
lemma lemma-st-part-Ex:
    (x::hypreal) ∈ HFinite
     ==> ∃ t ∈ Reals. ∀ r ∈ Reals. 0 < r --> abs (x - t) < r
apply (frule lemma-st-part-lub, safe)
apply (frule isLubD1a)
apply (blast dest: lemma-st-part-major2)
done

lemma st-part-Ex:
    (x::hypreal) ∈ HFinite ==> ∃ t ∈ Reals. x @= t
apply (simp add: approx-def Infinitesimal-def)
apply (drule lemma-st-part-Ex, auto)
done
```

There is a unique real infinitely close

```
lemma st-part-Ex1: x ∈ HFinite ==> EX! t::hypreal. t ∈ Reals & x @= t
apply (drule st-part-Ex, safe)
apply (drule-tac [2] approx-sym, drule-tac [2] approx-sym, drule-tac [2] approx-sym)
apply (auto intro!: approx-unique-real)
done
```

37.9 Finite, Infinite and Infinitesimal

lemma *HFinite-Int-HInfinite-empty* [*simp*]: *HFinite Int HInfinite* = {}
apply (*simp add*: *HFinite-def HInfinite-def*)
apply (*auto dest*: *order-less-trans*)
done

lemma *HFinite-not-HInfinite*:
  assumes *x*: *x* ∈ *HFinite* **shows** *x* ∉ *HInfinite*
proof
  assume *x'*: *x* ∈ *HInfinite*
  with *x* **have** *x* ∈ *HFinite* ∩ *HInfinite* **by** *blast*
  thus *False* **by** *auto*
qed

lemma *not-HFinite-HInfinite*: *x*∉ *HFinite* ==> *x* ∈ *HInfinite*
apply (*simp add*: *HInfinite-def HFinite-def*, *auto*)
apply (*drule-tac x* = *r* + *1* **in** *bspec*)
apply (*auto*)
done

lemma *HInfinite-HFinite-disj*: *x* ∈ *HInfinite* | *x* ∈ *HFinite*
by (*blast intro*: *not-HFinite-HInfinite*)

lemma *HInfinite-HFinite-iff*: (*x* ∈ *HInfinite*) = (*x* ∉ *HFinite*)
by (*blast dest*: *HFinite-not-HInfinite not-HFinite-HInfinite*)

lemma *HFinite-HInfinite-iff*: (*x* ∈ *HFinite*) = (*x* ∉ *HInfinite*)
by (*simp add*: *HInfinite-HFinite-iff*)

lemma *HInfinite-diff-HFinite-Infinitesimal-disj*:
    x ∉ *Infinitesimal* ==> *x* ∈ *HInfinite* | *x* ∈ *HFinite* − *Infinitesimal*
by (*fast intro*: *not-HFinite-HInfinite*)

lemma *HFinite-inverse*:
  fixes *x* :: *'a*::*real-normed-div-algebra star*
  shows [| *x* ∈ *HFinite*; *x* ∉ *Infinitesimal* |] ==> *inverse x* ∈ *HFinite*
apply (*subgoal-tac x* ≠ *0*)
apply (*cut-tac x* = *inverse x* **in** *HInfinite-HFinite-disj*)
apply (*auto dest*!: *HInfinite-inverse-Infinitesimal*
          simp add: *nonzero-inverse-inverse-eq*)
done

lemma *HFinite-inverse2*:
  fixes *x* :: *'a*::*real-normed-div-algebra star*
  shows *x* ∈ *HFinite* − *Infinitesimal* ==> *inverse x* ∈ *HFinite*
by (*blast intro*: *HFinite-inverse*)

lemma *Infinitesimal-inverse-HFinite*:
fixes x :: *'a*::*real-normed-div-algebra star*
shows $x \notin$ *Infinitesimal* ==> *inverse*$(x) \in$ *HFinite*
apply (*drule HInfinite-diff-HFinite-Infinitesimal-disj*)
apply (*blast intro*: *HFinite-inverse HInfinite-inverse-Infinitesimal Infinitesimal-subset-HFinite* [*THEN subsetD*])
done

lemma *HFinite-not-Infinitesimal-inverse*:
fixes x :: *'a*::*real-normed-div-algebra star*
shows $x \in$ *HFinite* $-$ *Infinitesimal* ==> *inverse* $x \in$ *HFinite* $-$ *Infinitesimal*
apply (*auto intro*: *Infinitesimal-inverse-HFinite*)
apply (*drule Infinitesimal-HFinite-mult2*, *assumption*)
apply (*simp add*: *not-Infinitesimal-not-zero right-inverse*)
done

lemma *approx-inverse*:
fixes x y :: *'a*::*real-normed-div-algebra star*
shows
[| x @= y; $y \in$ *HFinite* $-$ *Infinitesimal* |]
==> *inverse* x @= *inverse* y
apply (*frule HFinite-diff-Infinitesimal-approx*, *assumption*)
apply (*frule not-Infinitesimal-not-zero2*)
apply (*frule-tac* $x = x$ **in** *not-Infinitesimal-not-zero2*)
apply (*drule HFinite-inverse2*)+
apply (*drule approx-mult2*, *assumption*, *auto*)
apply (*drule-tac* $c =$ *inverse* x **in** *approx-mult1*, *assumption*)
apply (*auto intro*: *approx-sym simp add*: *mult-assoc*)
done

lemmas *star-of-approx-inverse* = *star-of-HFinite-diff-Infinitesimal* [*THEN* [*2*] *approx-inverse*]
lemmas *hypreal-of-real-approx-inverse* = *hypreal-of-real-HFinite-diff-Infinitesimal* [*THEN* [*2*] *approx-inverse*]

lemma *inverse-add-Infinitesimal-approx*:
fixes x h :: *'a*::*real-normed-div-algebra star*
shows
[| $x \in$ *HFinite* $-$ *Infinitesimal*;
$h \in$ *Infinitesimal* |] ==> *inverse*$(x + h)$ @= *inverse* x
apply (*auto intro*: *approx-inverse approx-sym Infinitesimal-add-approx-self*)
done

lemma *inverse-add-Infinitesimal-approx2*:
fixes x h :: *'a*::*real-normed-div-algebra star*
shows
[| $x \in$ *HFinite* $-$ *Infinitesimal*;
$h \in$ *Infinitesimal* |] ==> *inverse*$(h + x)$ @= *inverse* x
apply (*rule add-commute* [*THEN subst*])

apply (*blast intro*: *inverse-add-Infinitesimal-approx*)
done

lemma *inverse-add-Infinitesimal-approx-Infinitesimal*:
fixes *x h* :: *'a*::*real-normed-div-algebra star*
shows
[| *x* ∈ *HFinite* − *Infinitesimal*;
h ∈ *Infinitesimal* |] ==> *inverse*(*x* + *h*) − *inverse x* @= *h*
apply (*rule approx-trans2*)
apply (*auto intro*: *inverse-add-Infinitesimal-approx*
simp add: *mem-infmal-iff approx-minus-iff* [*symmetric*])
done

lemma *Infinitesimal-square-iff*:
fixes *x* :: *'a*::*real-normed-div-algebra star*
shows (*x* ∈ *Infinitesimal*) = (*x*∗*x* ∈ *Infinitesimal*)
apply (*auto intro*: *Infinitesimal-mult*)
apply (*rule ccontr*, *frule Infinitesimal-inverse-HFinite*)
apply (*frule not-Infinitesimal-not-zero*)
apply (*auto dest*: *Infinitesimal-HFinite-mult simp add*: *mult-assoc*)
done
declare *Infinitesimal-square-iff* [*symmetric*, *simp*]

lemma *HFinite-square-iff* [*simp*]:
fixes *x* :: *'a*::*real-normed-div-algebra star*
shows (*x*∗*x* ∈ *HFinite*) = (*x* ∈ *HFinite*)
apply (*auto intro*: *HFinite-mult*)
apply (*auto dest*: *HInfinite-mult simp add*: *HFinite-HInfinite-iff*)
done

lemma *HInfinite-square-iff* [*simp*]:
fixes *x* :: *'a*::*real-normed-div-algebra star*
shows (*x*∗*x* ∈ *HInfinite*) = (*x* ∈ *HInfinite*)
by (*auto simp add*: *HInfinite-HFinite-iff*)

lemma *approx-HFinite-mult-cancel*:
fixes *a w z* :: *'a*::*real-normed-div-algebra star*
shows [| *a*: *HFinite*−*Infinitesimal*; *a*∗ *w* @= *a*∗*z* |] ==> *w* @= *z*
apply *safe*
apply (*frule HFinite-inverse*, *assumption*)
apply (*drule not-Infinitesimal-not-zero*)
apply (*auto dest*: *approx-mult2 simp add*: *mult-assoc* [*symmetric*])
done

lemma *approx-HFinite-mult-cancel-iff1*:
fixes *a w z* :: *'a*::*real-normed-div-algebra star*
shows *a*: *HFinite*−*Infinitesimal* ==> (*a* ∗ *w* @= *a* ∗ *z*) = (*w* @= *z*)
by (*auto intro*: *approx-mult2 approx-HFinite-mult-cancel*)

lemma *HInfinite-HFinite-add-cancel*:
[| $x + y \in$ *HInfinite*; $y \in$ *HFinite* |] ==> $x \in$ *HInfinite*
apply (*rule ccontr*)
apply (*drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*auto dest*: *HFinite-add simp add*: *HInfinite-HFinite-iff*)
done

lemma *HInfinite-HFinite-add*:
[| $x \in$ *HInfinite*; $y \in$ *HFinite* |] ==> $x + y \in$ *HInfinite*
apply (*rule-tac* $y = -y$ **in** *HInfinite-HFinite-add-cancel*)
apply (*auto simp add*: *add-assoc HFinite-minus-iff*)
done

lemma *HInfinite-ge-HInfinite*:
[| (x::*hypreal*) $\in$ *HInfinite*; $x \leq y$; $0 \leq x$ |] ==> $y \in$ *HInfinite*
by (*auto intro*: *HFinite-bounded simp add*: *HInfinite-HFinite-iff*)

lemma *Infinitesimal-inverse-HInfinite*:
fixes x :: $'a$::*real-normed-div-algebra star*
shows [| $x \in$ *Infinitesimal*; $x \neq 0$ |] ==> *inverse* $x \in$ *HInfinite*
apply (*rule ccontr*, *drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*auto dest*: *Infinitesimal-HFinite-mult2*)
done

lemma *HInfinite-HFinite-not-Infinitesimal-mult*:
fixes x y :: $'a$::*real-normed-div-algebra star*
shows [| $x \in$ *HInfinite*; $y \in$ *HFinite* $-$ *Infinitesimal* |]
==> $x * y \in$ *HInfinite*
apply (*rule ccontr*, *drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*frule HFinite-Infinitesimal-not-zero*)
apply (*drule HFinite-not-Infinitesimal-inverse*)
apply (*safe*, *drule HFinite-mult*)
apply (*auto simp add*: *mult-assoc HFinite-HInfinite-iff*)
done

lemma *HInfinite-HFinite-not-Infinitesimal-mult2*:
fixes x y :: $'a$::*real-normed-div-algebra star*
shows [| $x \in$ *HInfinite*; $y \in$ *HFinite* $-$ *Infinitesimal* |]
==> $y * x \in$ *HInfinite*
apply (*rule ccontr*, *drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*frule HFinite-Infinitesimal-not-zero*)
apply (*drule HFinite-not-Infinitesimal-inverse*)
apply (*safe*, *drule-tac* x=*inverse* y **in** *HFinite-mult*)
apply *assumption*
apply (*auto simp add*: *mult-assoc* [*symmetric*] *HFinite-HInfinite-iff*)
done

lemma *HInfinite-gt-SReal*:
[| (x::*hypreal*) $\in$ *HInfinite*; $0 < x$; $y \in$ *Reals* |] ==> $y < x$

by (*auto dest*!: *bspec simp add*: *HInfinite-def abs-if order-less-imp-le*)

lemma *HInfinite-gt-zero-gt-one*:
[| (*x*::*hypreal*) ∈ *HInfinite*; *0* < *x* |] ==> *1* < *x*
by (*auto intro*: *HInfinite-gt-SReal*)

lemma *not-HInfinite-one* [*simp*]: *1* ∉ *HInfinite*
apply (*simp* (*no-asm*) *add*: *HInfinite-HFinite-iff*)
done

lemma *approx-hrabs-disj*: *abs* (*x*::*hypreal*) @= *x* | *abs x* @= −*x*
by (*cut-tac x* = *x* **in** *hrabs-disj*, *auto*)

37.10 Theorems about Monads

lemma *monad-hrabs-Un-subset*: *monad* (*abs x*) ≤ *monad*(*x*::*hypreal*) *Un monad*(−*x*)
by (*rule-tac x1* = *x* **in** *hrabs-disj* [*THEN disjE*], *auto*)

lemma *Infinitesimal-monad-eq*: *e* ∈ *Infinitesimal* ==> *monad* (*x*+*e*) = *monad x*
by (*fast intro*!: *Infinitesimal-add-approx-self* [*THEN approx-sym*] *approx-monad-iff* [*THEN iffD1*])

lemma *mem-monad-iff*: (*u* ∈ *monad x*) = (−*u* ∈ *monad* (−*x*))
by (*simp add*: *monad-def*)

lemma *Infinitesimal-monad-zero-iff*: (*x* ∈ *Infinitesimal*) = (*x* ∈ *monad 0*)
by (*auto intro*: *approx-sym simp add*: *monad-def mem-infmal-iff*)

lemma *monad-zero-minus-iff*: (*x* ∈ *monad 0*) = (−*x* ∈ *monad 0*)
apply (*simp* (*no-asm*) *add*: *Infinitesimal-monad-zero-iff* [*symmetric*])
done

lemma *monad-zero-hrabs-iff*: ((*x*::*hypreal*) ∈ *monad 0*) = (*abs x* ∈ *monad 0*)
apply (*rule-tac x1* = *x* **in** *hrabs-disj* [*THEN disjE*])
apply (*auto simp add*: *monad-zero-minus-iff* [*symmetric*])
done

lemma *mem-monad-self* [*simp*]: *x* ∈ *monad x*
by (*simp add*: *monad-def*)

37.11 Proof that $x \approx y$ implies $|x| \approx |y|$

lemma *approx-subset-monad*: *x* @= *y* ==> {*x*,*y*} ≤ *monad x*
apply (*simp* (*no-asm*))
apply (*simp add*: *approx-monad-iff*)
done

lemma *approx-subset-monad2*: *x* @= *y* ==> {*x*,*y*} ≤ *monad y*
apply (*drule approx-sym*)

apply (*fast dest*: *approx-subset-monad*)
done

lemma *mem-monad-approx*: $u \in monad\ x \Longrightarrow x\ @=\ u$
by (*simp add*: *monad-def*)

lemma *approx-mem-monad*: $x\ @=\ u \Longrightarrow u \in monad\ x$
by (*simp add*: *monad-def*)

lemma *approx-mem-monad2*: $x\ @=\ u \Longrightarrow x \in monad\ u$
apply (*simp add*: *monad-def*)
apply (*blast intro*!: *approx-sym*)
done

lemma *approx-mem-monad-zero*: $[|\ x\ @=\ y; x \in monad\ 0\ |] \Longrightarrow y \in monad\ 0$
apply (*drule mem-monad-approx*)
apply (*fast intro*: *approx-mem-monad approx-trans*)
done

lemma *Infinitesimal-approx-hrabs*:
$[|\ x\ @=\ y;\ (x::hypreal) \in Infinitesimal\ |] \Longrightarrow abs\ x\ @=\ abs\ y$
apply (*drule Infinitesimal-monad-zero-iff* [*THEN iffD1*])
apply (*blast intro*: *approx-mem-monad-zero monad-zero-hrabs-iff* [*THEN iffD1*] *mem-monad-approx approx-trans3*)
done

lemma *less-Infinitesimal-less*:
$[|\ 0 < x;\ (x::hypreal) \notin Infinitesimal;\ e :Infinitesimal\ |] \Longrightarrow e < x$
apply (*rule ccontr*)
apply (*auto intro*: *Infinitesimal-zero* [*THEN* [*2*] *Infinitesimal-interval*]
dest!: *order-le-imp-less-or-eq simp add*: *linorder-not-less*)
done

lemma *Ball-mem-monad-gt-zero*:
$[|\ 0 < (x::hypreal);\ x \notin Infinitesimal;\ u \in monad\ x\ |] \Longrightarrow 0 < u$
apply (*drule mem-monad-approx* [*THEN approx-sym*])
apply (*erule bex-Infinitesimal-iff2* [*THEN iffD2*, *THEN bexE*])
apply (*drule-tac* $e = -xa$ **in** *less-Infinitesimal-less*, *auto*)
done

lemma *Ball-mem-monad-less-zero*:
$[|\ (x::hypreal) < 0;\ x \notin Infinitesimal;\ u \in monad\ x\ |] \Longrightarrow u < 0$
apply (*drule mem-monad-approx* [*THEN approx-sym*])
apply (*erule bex-Infinitesimal-iff* [*THEN iffD2*, *THEN bexE*])
apply (*cut-tac* $x = -x$ **and** $e = xa$ **in** *less-Infinitesimal-less*, *auto*)
done

lemma *lemma-approx-gt-zero*:
$[|0 < (x::hypreal);\ x \notin Infinitesimal;\ x\ @=\ y|] \Longrightarrow 0 < y$

by (*blast dest*: *Ball-mem-monad-gt-zero approx-subset-monad*)

lemma *lemma-approx-less-zero*:
[|(*x*::*hypreal*) < *0*; *x* ∉ *Infinitesimal*; *x* @= *y*|] ==> *y* < *0*
by (*blast dest*: *Ball-mem-monad-less-zero approx-subset-monad*)

theorem *approx-hrabs*: (*x*::*hypreal*) @= *y* ==> *abs x* @= *abs y*
by (*drule approx-hnorm*, *simp*)

lemma *approx-hrabs-zero-cancel*: *abs*(*x*::*hypreal*) @= *0* ==> *x* @= *0*
apply (*cut-tac x* = *x* **in** *hrabs-disj*)
apply (*auto dest*: *approx-minus*)
done

lemma *approx-hrabs-add-Infinitesimal*:
(*e*::*hypreal*) ∈ *Infinitesimal* ==> *abs x* @= *abs*(*x*+*e*)
by (*fast intro*: *approx-hrabs Infinitesimal-add-approx-self*)

lemma *approx-hrabs-add-minus-Infinitesimal*:
(*e*::*hypreal*) ∈ *Infinitesimal* ==> *abs x* @= *abs*(*x* + −*e*)
by (*fast intro*: *approx-hrabs Infinitesimal-add-minus-approx-self*)

lemma *hrabs-add-Infinitesimal-cancel*:
[| (*e*::*hypreal*) ∈ *Infinitesimal*; *e′* ∈ *Infinitesimal*;
abs(*x*+*e*) = *abs*(*y*+*e′*)|] ==> *abs x* @= *abs y*
apply (*drule-tac x* = *x* **in** *approx-hrabs-add-Infinitesimal*)
apply (*drule-tac x* = *y* **in** *approx-hrabs-add-Infinitesimal*)
apply (*auto intro*: *approx-trans2*)
done

lemma *hrabs-add-minus-Infinitesimal-cancel*:
[| (*e*::*hypreal*) ∈ *Infinitesimal*; *e′* ∈ *Infinitesimal*;
abs(*x* + −*e*) = *abs*(*y* + −*e′*)|] ==> *abs x* @= *abs y*
apply (*drule-tac x* = *x* **in** *approx-hrabs-add-minus-Infinitesimal*)
apply (*drule-tac x* = *y* **in** *approx-hrabs-add-minus-Infinitesimal*)
apply (*auto intro*: *approx-trans2*)
done

37.12 More *HFinite* and *Infinitesimal* Theorems

lemma *Infinitesimal-add-hypreal-of-real-less*:
[| *x* < *y*; *u* ∈ *Infinitesimal* |]
==> *hypreal-of-real x* + *u* < *hypreal-of-real y*
apply (*simp add*: *Infinitesimal-def*)
apply (*drule-tac x* = *hypreal-of-real y* + −*hypreal-of-real x* **in** *bspec*, *simp*)
apply (*simp add*: *abs-less-iff*)
done

lemma *Infinitesimal-add-hrabs-hypreal-of-real-less*:

```
    [| x ∈ Infinitesimal; abs(hypreal-of-real r) < hypreal-of-real y |]
     ==> abs (hypreal-of-real r + x) < hypreal-of-real y
apply (drule-tac x = hypreal-of-real r in approx-hrabs-add-Infinitesimal)
apply (drule approx-sym [THEN bex-Infinitesimal-iff2 [THEN iffD2]])
apply (auto intro!: Infinitesimal-add-hypreal-of-real-less
          simp del: star-of-abs
          simp add: star-of-abs [symmetric])
done

lemma Infinitesimal-add-hrabs-hypreal-of-real-less2:
    [| x ∈ Infinitesimal;  abs(hypreal-of-real r) < hypreal-of-real y |]
     ==> abs (x + hypreal-of-real r) < hypreal-of-real y
apply (rule add-commute [THEN subst])
apply (erule Infinitesimal-add-hrabs-hypreal-of-real-less, assumption)
done

lemma hypreal-of-real-le-add-Infininitesimal-cancel:
    [| u ∈ Infinitesimal; v ∈ Infinitesimal;
       hypreal-of-real x + u ≤ hypreal-of-real y + v |]
     ==> hypreal-of-real x ≤ hypreal-of-real y
apply (simp add: linorder-not-less [symmetric], auto)
apply (drule-tac u = v-u in Infinitesimal-add-hypreal-of-real-less)
apply (auto simp add: Infinitesimal-diff)
done

lemma hypreal-of-real-le-add-Infininitesimal-cancel2:
    [| u ∈ Infinitesimal; v ∈ Infinitesimal;
       hypreal-of-real x + u ≤ hypreal-of-real y + v |]
     ==> x ≤ y
by (blast intro: star-of-le [THEN iffD1]
         intro!: hypreal-of-real-le-add-Infininitesimal-cancel)

lemma hypreal-of-real-less-Infinitesimal-le-zero:
   [| hypreal-of-real x < e; e ∈ Infinitesimal |] ==> hypreal-of-real x ≤ 0
apply (rule linorder-not-less [THEN iffD1], safe)
apply (drule Infinitesimal-interval)
apply (drule-tac [4] SReal-hypreal-of-real [THEN SReal-Infinitesimal-zero], auto)
done


lemma Infinitesimal-add-not-zero:
    [| h ∈ Infinitesimal; x ≠ 0 |] ==> star-of x + h ≠ 0
apply auto
apply (subgoal-tac h = - star-of x, auto intro: equals-zero-I [symmetric])
done

lemma Infinitesimal-square-cancel [simp]:
    (x::hypreal)*x + y*y ∈ Infinitesimal ==> x*x ∈ Infinitesimal
apply (rule Infinitesimal-interval2)
```

apply (*rule-tac* [*3*] *zero-le-square*, *assumption*)
apply (*auto*)
done

lemma *HFinite-square-cancel* [*simp*]:
(*x*::*hypreal*)∗*x* + *y*∗*y* ∈ *HFinite* ==> *x*∗*x* ∈ *HFinite*
apply (*rule HFinite-bounded*, *assumption*)
apply (*auto*)
done

lemma *Infinitesimal-square-cancel2* [*simp*]:
(*x*::*hypreal*)∗*x* + *y*∗*y* ∈ *Infinitesimal* ==> *y*∗*y* ∈ *Infinitesimal*
apply (*rule Infinitesimal-square-cancel*)
apply (*rule add-commute* [*THEN subst*])
apply (*simp* (*no-asm*))
done

lemma *HFinite-square-cancel2* [*simp*]:
(*x*::*hypreal*)∗*x* + *y*∗*y* ∈ *HFinite* ==> *y*∗*y* ∈ *HFinite*
apply (*rule HFinite-square-cancel*)
apply (*rule add-commute* [*THEN subst*])
apply (*simp* (*no-asm*))
done

lemma *Infinitesimal-sum-square-cancel* [*simp*]:
(*x*::*hypreal*)∗*x* + *y*∗*y* + *z*∗*z* ∈ *Infinitesimal* ==> *x*∗*x* ∈ *Infinitesimal*
apply (*rule Infinitesimal-interval2*, *assumption*)
apply (*rule-tac* [*2*] *zero-le-square*, *simp*)
apply (*insert zero-le-square* [*of y*])
apply (*insert zero-le-square* [*of z*], *simp del*:*zero-le-square*)
done

lemma *HFinite-sum-square-cancel* [*simp*]:
(*x*::*hypreal*)∗*x* + *y*∗*y* + *z*∗*z* ∈ *HFinite* ==> *x*∗*x* ∈ *HFinite*
apply (*rule HFinite-bounded*, *assumption*)
apply (*rule-tac* [*2*] *zero-le-square*)
apply (*insert zero-le-square* [*of y*])
apply (*insert zero-le-square* [*of z*], *simp del*:*zero-le-square*)
done

lemma *Infinitesimal-sum-square-cancel2* [*simp*]:
(*y*::*hypreal*)∗*y* + *x*∗*x* + *z*∗*z* ∈ *Infinitesimal* ==> *x*∗*x* ∈ *Infinitesimal*
apply (*rule Infinitesimal-sum-square-cancel*)
apply (*simp add*: *add-ac*)
done

lemma *HFinite-sum-square-cancel2* [*simp*]:
(*y*::*hypreal*)∗*y* + *x*∗*x* + *z*∗*z* ∈ *HFinite* ==> *x*∗*x* ∈ *HFinite*
apply (*rule HFinite-sum-square-cancel*)

apply (*simp add*: *add-ac*)
done

lemma *Infinitesimal-sum-square-cancel3* [*simp*]:
(*z*::*hypreal*)∗*z* + *y*∗*y* + *x*∗*x* ∈ *Infinitesimal* ==> *x*∗*x* ∈ *Infinitesimal*
apply (*rule Infinitesimal-sum-square-cancel*)
apply (*simp add*: *add-ac*)
done

lemma *HFinite-sum-square-cancel3* [*simp*]:
(*z*::*hypreal*)∗*z* + *y*∗*y* + *x*∗*x* ∈ *HFinite* ==> *x*∗*x* ∈ *HFinite*
apply (*rule HFinite-sum-square-cancel*)
apply (*simp add*: *add-ac*)
done

lemma *monad-hrabs-less*:
[| *y* ∈ *monad x*; *0* < *hypreal-of-real e* |]
==> *abs* (*y* − *x*) < *hypreal-of-real e*
apply (*drule mem-monad-approx* [*THEN approx-sym*])
apply (*drule bex-Infinitesimal-iff* [*THEN iffD2*])
apply (*auto dest*!: *InfinitesimalD*)
done

lemma *mem-monad-SReal-HFinite*:
x ∈ *monad* (*hypreal-of-real a*) ==> *x* ∈ *HFinite*
apply (*drule mem-monad-approx* [*THEN approx-sym*])
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*])
apply (*safe dest*!: *Infinitesimal-subset-HFinite* [*THEN subsetD*])
apply (*erule SReal-hypreal-of-real* [*THEN SReal-subset-HFinite* [*THEN subsetD*], *THEN HFinite-add*])
done

37.13 Theorems about Standard Part

lemma *st-approx-self*: *x* ∈ *HFinite* ==> *st x* @= *x*
apply (*simp add*: *st-def*)
apply (*frule st-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *st-SReal*: *x* ∈ *HFinite* ==> *st x* ∈ *Reals*
apply (*simp add*: *st-def*)
apply (*frule st-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *st-HFinite*: *x* ∈ *HFinite* ==> *st x* ∈ *HFinite*

by (*erule st-SReal* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])

lemma *st-unique*: ⟦$r \in \mathbb{R}$; $r \approx x$⟧ ⟹ $st\ x = r$
apply (*frule SReal-subset-HFinite* [*THEN subsetD*])
apply (*drule* (*1*) *approx-HFinite*)
apply (*unfold st-def*)
apply (*rule some-equality*)
apply (*auto intro*: *approx-unique-real*)
done

lemma *st-SReal-eq*: $x \in Reals$ ==> $st\ x = x$
apply (*erule st-unique*)
apply (*rule approx-refl*)
done

lemma *st-hypreal-of-real* [*simp*]: $st\ (hypreal\text{-}of\text{-}real\ x) = hypreal\text{-}of\text{-}real\ x$
by (*rule SReal-hypreal-of-real* [*THEN st-SReal-eq*])

lemma *st-eq-approx*: [| $x \in HFinite$; $y \in HFinite$; $st\ x = st\ y$ |] ==> x @= y
by (*auto dest*!: *st-approx-self elim*!: *approx-trans3*)

lemma *approx-st-eq*:
  assumes $x \in HFinite$ **and** $y \in HFinite$ **and** x @= y
  shows $st\ x = st\ y$
proof −
  have $st\ x$ @= x $st\ y$ @= y $st\ x \in Reals$ $st\ y \in Reals$
    by (*simp-all add*: *st-approx-self st-SReal prems*)
  with *prems* **show** *?thesis*
    by (*fast elim*: *approx-trans approx-trans2 SReal-approx-iff* [*THEN iffD1*])
qed

lemma *st-eq-approx-iff*:
    [| $x \in HFinite$; $y \in HFinite$|]
              ==> (x @= y) = ($st\ x = st\ y$)
by (*blast intro*: *approx-st-eq st-eq-approx*)

lemma *st-Infinitesimal-add-SReal*:
    [| $x \in Reals$; $e \in Infinitesimal$ |] ==> $st(x + e) = x$
apply (*erule st-unique*)
apply (*erule Infinitesimal-add-approx-self*)
done

lemma *st-Infinitesimal-add-SReal2*:
    [| $x \in Reals$; $e \in Infinitesimal$ |] ==> $st(e + x) = x$
apply (*erule st-unique*)
apply (*erule Infinitesimal-add-approx-self2*)
done

lemma *HFinite-st-Infinitesimal-add*:

$x \in HFinite \Longrightarrow \exists e \in Infinitesimal.\ x = st(x) + e$
by (*blast dest*!: *st-approx-self* [*THEN approx-sym*] *bex-Infinitesimal-iff2* [*THEN iffD2*])

lemma *st-add*: $[\![x \in HFinite;\ y \in HFinite]\!] \Longrightarrow st\ (x + y) = st\ x + st\ y$
by (*simp add*: *st-unique st-SReal st-approx-self approx-add*)

lemma *st-number-of* [*simp*]: $st\ (number\text{-}of\ w) = number\text{-}of\ w$
by (*rule Reals-number-of* [*THEN st-SReal-eq*])

lemma [*simp*]: $st\ 0 = 0\ st\ 1 = 1$
by (*simp-all add*: *st-SReal-eq*)

lemma *st-minus*: $x \in HFinite \Longrightarrow st\ (-x) = -\ st\ x$
by (*simp add*: *st-unique st-SReal st-approx-self approx-minus*)

lemma *st-diff*: $[\![x \in HFinite;\ y \in HFinite]\!] \Longrightarrow st\ (x - y) = st\ x - st\ y$
by (*simp add*: *st-unique st-SReal st-approx-self approx-diff*)

lemma *st-mult*: $[\![x \in HFinite;\ y \in HFinite]\!] \Longrightarrow st\ (x * y) = st\ x * st\ y$
by (*simp add*: *st-unique st-SReal st-approx-self approx-mult-HFinite*)

lemma *st-Infinitesimal*: $x \in Infinitesimal \Longrightarrow st\ x = 0$
by (*simp add*: *st-unique mem-infmal-iff*)

lemma *st-not-Infinitesimal*: $st(x) \neq 0 \Longrightarrow x \notin Infinitesimal$
by (*fast intro*: *st-Infinitesimal*)

lemma *st-inverse*:
$[\!|\ x \in HFinite;\ st\ x \neq 0\ |\!]$
$\Longrightarrow st(inverse\ x) = inverse\ (st\ x)$
apply (*rule-tac c1* $= st\ x$ **in** *hypreal-mult-left-cancel* [*THEN iffD1*])
apply (*auto simp add*: *st-mult* [*symmetric*] *st-not-Infinitesimal HFinite-inverse*)
apply (*subst right-inverse*, *auto*)
done

lemma *st-divide* [*simp*]:
$[\!|\ x \in HFinite;\ y \in HFinite;\ st\ y \neq 0\ |\!]$
$\Longrightarrow st(x/y) = (st\ x)\ /\ (st\ y)$
by (*simp add*: *divide-inverse st-mult st-not-Infinitesimal HFinite-inverse st-inverse*)

lemma *st-idempotent* [*simp*]: $x \in HFinite \Longrightarrow st(st(x)) = st(x)$
by (*blast intro*: *st-HFinite st-approx-self approx-st-eq*)

lemma *Infinitesimal-add-st-less*:
$[\!|\ x \in HFinite;\ y \in HFinite;\ u \in Infinitesimal;\ st\ x < st\ y\ |\!]$
$\Longrightarrow st\ x + u < st\ y$
apply (*drule st-SReal*)+

apply (*auto intro*!: *Infinitesimal-add-hypreal-of-real-less simp add*: *SReal-iff*)
done

lemma *Infinitesimal-add-st-le-cancel*:
[| $x \in$ *HFinite*; $y \in$ *HFinite*;
$u \in$ *Infinitesimal*; *st* $x \leq$ *st* $y + u$
|] ==> *st* $x \leq$ *st* y
apply (*simp add*: *linorder-not-less* [*symmetric*])
apply (*auto dest*: *Infinitesimal-add-st-less*)
done

lemma *st-le*: [| $x \in$ *HFinite*; $y \in$ *HFinite*; $x \leq y$ |] ==> $st(x) \leq st(y)$
apply (*frule HFinite-st-Infinitesimal-add*)
apply (*rotate-tac 1*)
apply (*frule HFinite-st-Infinitesimal-add*, *safe*)
apply (*rule Infinitesimal-add-st-le-cancel*)
apply (*rule-tac* [*3*] $x = ea$ **and** $y = e$ **in** *Infinitesimal-diff*)
apply (*auto simp add*: *add-assoc* [*symmetric*])
done

lemma *st-zero-le*: [| $0 \leq x$; $x \in$ *HFinite* |] ==> $0 \leq$ *st* x
apply (*subst numeral-0-eq-0* [*symmetric*])
apply (*rule st-number-of* [*THEN subst*])
apply (*rule st-le*, *auto*)
done

lemma *st-zero-ge*: [| $x \leq 0$; $x \in$ *HFinite* |] ==> *st* $x \leq 0$
apply (*subst numeral-0-eq-0* [*symmetric*])
apply (*rule st-number-of* [*THEN subst*])
apply (*rule st-le*, *auto*)
done

lemma *st-hrabs*: $x \in$ *HFinite* ==> $abs(st\ x) = st(abs\ x)$
apply (*simp add*: *linorder-not-le st-zero-le abs-if st-minus*
linorder-not-less)
apply (*auto dest*!: *st-zero-ge* [*OF order-less-imp-le*])
done

37.14 Alternative Definitions using Free Ultrafilter

37.14.1 *HFinite*

lemma *HFinite-FreeUltrafilterNat*:
star-n X $\in$ *HFinite*
==> $\exists u.$ $\{n.$ *norm* $(X\ n) < u\} \in$ *FreeUltrafilterNat*
apply (*auto simp add*: *HFinite-def SReal-def*)
apply (*rule-tac x*=*r* **in** *exI*)
apply (*simp add*: *hnorm-def star-of-def starfun-star-n*)
apply (*simp add*: *star-less-def starP2-star-n*)
done

lemma *FreeUltrafilterNat-HFinite*:
$\exists u.\ \{n.\ norm\ (X\ n) < u\} \in FreeUltrafilterNat$
$==>$ *star-n X* $\in$ *HFinite*
apply (*auto simp add*: *HFinite-def mem-Rep-star-iff*)
apply (*rule-tac x=star-of u* **in** *bexI*)
apply (*simp add*: *hnorm-def starfun-star-n star-of-def*)
apply (*simp add*: *star-less-def starP2-star-n*)
apply (*simp add*: *SReal-def*)
done

lemma *HFinite-FreeUltrafilterNat-iff*:
(*star-n X* $\in$ *HFinite*) = ($\exists u.\ \{n.\ norm\ (X\ n) < u\} \in FreeUltrafilterNat$)
by (*blast intro*!: *HFinite-FreeUltrafilterNat FreeUltrafilterNat-HFinite*)

37.14.2 *HInfinite*

lemma *lemma-Compl-eq*: $-\{n.\ u < norm\ (xa\ n)\} = \{n.\ norm\ (xa\ n) \le u\}$
by *auto*

lemma *lemma-Compl-eq2*: $-\{n.\ norm\ (xa\ n) < u\} = \{n.\ u \le norm\ (xa\ n)\}$
by *auto*

lemma *lemma-Int-eq1*:
$\{n.\ norm\ (xa\ n) \le u\}\ Int\ \{n.\ u \le norm\ (xa\ n)\}$
$= \{n.\ norm(xa\ n) = u\}$
by *auto*

lemma *lemma-FreeUltrafilterNat-one*:
$\{n.\ norm\ (xa\ n) = u\} \le \{n.\ norm\ (xa\ n) < u + (1::real)\}$
by *auto*

lemma *FreeUltrafilterNat-const-Finite*:
$\{n.\ norm\ (X\ n) = u\} \in FreeUltrafilterNat ==>$ *star-n X* $\in$ *HFinite*
apply (*rule FreeUltrafilterNat-HFinite*)
apply (*rule-tac x = u + 1* **in** *exI*)
apply (*erule ultra*, *simp*)
done

lemma *HInfinite-FreeUltrafilterNat*:
star-n X $\in$ *HInfinite* $==> \forall u.\ \{n.\ u < norm\ (X\ n)\} \in FreeUltrafilterNat$
apply (*drule HInfinite-HFinite-iff* [*THEN iffD1*])
apply (*simp add*: *HFinite-FreeUltrafilterNat-iff*)
apply (*rule allI*, *drule-tac x=u + 1* **in** *spec*)
apply (*drule FreeUltrafilterNat.not-memD*)
apply (*simp add*: *Collect-neg-eq* [*symmetric*] *linorder-not-less*)
apply (*erule ultra*, *simp*)
done

lemma *lemma-Int-HI*:
$\{n.\ norm\ (Xa\ n) < u\}\ Int\ \{n.\ X\ n = Xa\ n\} \subseteq \{n.\ norm\ (X\ n) < (u::real)\}$
by *auto*

lemma *lemma-Int-HIa*: $\{n.\ u < norm\ (X\ n)\}\ Int\ \{n.\ norm\ (X\ n) < u\} = \{\}$
by (*auto intro*: *order-less-asym*)

lemma *FreeUltrafilterNat-HInfinite*:
$\forall u.\ \{n.\ u < norm\ (X\ n)\} \in FreeUltrafilterNat \Longrightarrow star\text{-}n\ X \in HInfinite$
apply (*rule HInfinite-HFinite-iff* [*THEN iffD2*])
apply (*safe*, *drule HFinite-FreeUltrafilterNat*, *safe*)
apply (*drule-tac* $x = u$ **in** *spec*)
apply (*drule* (*1*) *FreeUltrafilterNat.Int*)
apply (*simp add*: *Collect-conj-eq* [*symmetric*])
apply (*subgoal-tac* $\forall n.\ \neg\ (norm\ (X\ n) < u \wedge u < norm\ (X\ n))$, *auto*)
done

lemma *HInfinite-FreeUltrafilterNat-iff*:
$(star\text{-}n\ X \in HInfinite) = (\forall u.\ \{n.\ u < norm\ (X\ n)\} \in FreeUltrafilterNat)$
by (*blast intro*!: *HInfinite-FreeUltrafilterNat FreeUltrafilterNat-HInfinite*)

37.14.3 *Infinitesimal*

lemma *ball-SReal-eq*: $(\forall x::hypreal \in Reals.\ P\ x) = (\forall x::real.\ P\ (star\text{-}of\ x))$
by (*unfold SReal-def*, *auto*)

lemma *Infinitesimal-FreeUltrafilterNat*:
$star\text{-}n\ X \in Infinitesimal \Longrightarrow \forall u{>}0.\ \{n.\ norm\ (X\ n) < u\} \in \mathcal{U}$
apply (*simp add*: *Infinitesimal-def ball-SReal-eq*)
apply (*simp add*: *hnorm-def starfun-star-n star-of-def*)
apply (*simp add*: *star-less-def starP2-star-n*)
done

lemma *FreeUltrafilterNat-Infinitesimal*:
$\forall u{>}0.\ \{n.\ norm\ (X\ n) < u\} \in \mathcal{U} \Longrightarrow star\text{-}n\ X \in Infinitesimal$
apply (*simp add*: *Infinitesimal-def ball-SReal-eq*)
apply (*simp add*: *hnorm-def starfun-star-n star-of-def*)
apply (*simp add*: *star-less-def starP2-star-n*)
done

lemma *Infinitesimal-FreeUltrafilterNat-iff*:
$(star\text{-}n\ X \in Infinitesimal) = (\forall u{>}0.\ \{n.\ norm\ (X\ n) < u\} \in \mathcal{U})$
by (*blast intro*!: *Infinitesimal-FreeUltrafilterNat FreeUltrafilterNat-Infinitesimal*)

lemma *lemma-Infinitesimal*:
$(\forall r.\ 0 < r \longrightarrow x < r) = (\forall n.\ x < inverse(real\ (Suc\ n)))$

apply (*auto simp add*: *real-of-nat-Suc-gt-zero*)
apply (*blast dest*!: *reals-Archimedean intro*: *order-less-trans*)
done

lemma *lemma-Infinitesimal2*:
(∀ *r* ∈ *Reals*. 0 < *r* −−> *x* < *r*) =
(∀ *n*. *x* < *inverse*(*hypreal-of-nat* (*Suc n*)))
apply *safe*
apply (*drule-tac x* = *inverse* (*hypreal-of-real* (*real* (*Suc n*))) **in** *bspec*)
apply (*simp* (*no-asm-use*))
apply (*rule real-of-nat-Suc-gt-zero* [*THEN positive-imp-inverse-positive*, *THEN star-of-less* [*THEN iffD2*], *THEN* [*2*] *impE*])
prefer *2* **apply** *assumption*
apply (*simp add*: *real-of-nat-def*)
apply (*auto dest*!: *reals-Archimedean simp add*: *SReal-iff*)
apply (*drule star-of-less* [*THEN iffD2*])
apply (*simp add*: *real-of-nat-def*)
apply (*blast intro*: *order-less-trans*)
done

lemma *Infinitesimal-hypreal-of-nat-iff*:
Infinitesimal = {*x*. ∀ *n*. *hnorm x* < *inverse* (*hypreal-of-nat* (*Suc n*))}
apply (*simp add*: *Infinitesimal-def*)
apply (*auto simp add*: *lemma-Infinitesimal2*)
done

37.15 Proof that ω is an infinite number

It will follow that epsilon is an infinitesimal number.

lemma *Suc-Un-eq*: {*n*. *n* < *Suc m*} = {*n*. *n* < *m*} *Un* {*n*. *n* = *m*}
by (*auto simp add*: *less-Suc-eq*)

lemma *finite-nat-segment*: *finite* {*n*::*nat*. *n* < *m*}
apply (*induct m*)
apply (*auto simp add*: *Suc-Un-eq*)
done

lemma *finite-real-of-nat-segment*: *finite* {*n*::*nat*. *real n* < *real* (*m*::*nat*)}
by (*auto intro*: *finite-nat-segment*)

lemma *finite-real-of-nat-less-real*: *finite* {*n*::*nat*. *real n* < *u*}
apply (*cut-tac x* = *u* **in** *reals-Archimedean2*, *safe*)
apply (*rule finite-real-of-nat-segment* [*THEN* [*2*] *finite-subset*])
apply (*auto dest*: *order-less-trans*)
done

lemma *lemma-real-le-Un-eq*:

$\{n.\ f\ n \leq u\} = \{n.\ f\ n < u\}\ Un\ \{n.\ u = (f\ n :: real)\}$
by (*auto dest*: *order-le-imp-less-or-eq simp add*: *order-less-imp-le*)

lemma *finite-real-of-nat-le-real*: *finite* $\{n{::}nat.\ real\ n \leq u\}$
by (*auto simp add*: *lemma-real-le-Un-eq lemma-finite-omega-set finite-real-of-nat-less-real*)

lemma *finite-rabs-real-of-nat-le-real*: *finite* $\{n{::}nat.\ abs(real\ n) \leq u\}$
apply (*simp* (*no-asm*) *add*: *real-of-nat-Suc-gt-zero finite-real-of-nat-le-real*)
done

lemma *rabs-real-of-nat-le-real-FreeUltrafilterNat*:
$\{n.\ abs(real\ n) \leq u\} \notin FreeUltrafilterNat$
by (*blast intro*!: *FreeUltrafilterNat.finite finite-rabs-real-of-nat-le-real*)

lemma *FreeUltrafilterNat-nat-gt-real*: $\{n.\ u < real\ n\} \in FreeUltrafilterNat$
apply (*rule ccontr*, *drule FreeUltrafilterNat.not-memD*)
apply (*subgoal-tac* $-\{n{::}nat.\ u < real\ n\} = \{n.\ real\ n \leq u\}$)
prefer *2* **apply** *force*
apply (*simp add*: *finite-real-of-nat-le-real* [*THEN FreeUltrafilterNat.finite*])
done

lemma *Compl-real-le-eq*: $-\{n{::}nat.\ real\ n \leq u\} = \{n.\ u < real\ n\}$
by (*auto dest*!: *order-le-less-trans simp add*: *linorder-not-le*)

ω is a member of *HInfinite*

lemma *FreeUltrafilterNat-omega*: $\{n.\ u < real\ n\} \in FreeUltrafilterNat$
apply (*cut-tac* $u = u$ **in** *rabs-real-of-nat-le-real-FreeUltrafilterNat*)
apply (*auto dest*: *FreeUltrafilterNat.not-memD simp add*: *Compl-real-le-eq*)
done

theorem *HInfinite-omega* [*simp*]: *omega* $\in$ *HInfinite*
apply (*simp add*: *omega-def*)
apply (*rule FreeUltrafilterNat-HInfinite*)
apply (*simp* (*no-asm*) *add*: *real-norm-def real-of-nat-Suc diff-less-eq* [*symmetric*]
FreeUltrafilterNat-omega)
done

lemma *Infinitesimal-epsilon* [*simp*]: *epsilon* $\in$ *Infinitesimal*
by (*auto intro*!: *HInfinite-inverse-Infinitesimal HInfinite-omega simp add*: *hypreal-epsilon-inverse-omega*)

lemma *HFinite-epsilon* [*simp*]: *epsilon* $\in$ *HFinite*
by (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*])

lemma *epsilon-approx-zero* [*simp*]: *epsilon* @= *0*
apply (*simp* (*no-asm*) *add*: *mem-infmal-iff* [*symmetric*])

done

lemma *real-of-nat-less-inverse-iff*:
0 < u ==> (u < inverse (real(Suc n))) = (real(Suc n) < inverse u)
apply (*simp add*: *inverse-eq-divide*)
apply (*subst pos-less-divide-eq*, *assumption*)
apply (*subst pos-less-divide-eq*)
apply (*simp add*: *real-of-nat-Suc-gt-zero*)
apply (*simp add*: *real-mult-commute*)
done

lemma *finite-inverse-real-of-posnat-gt-real*:
0 < u ==> finite {n. u < inverse(real(Suc n))}
apply (*simp* (*no-asm-simp*) *add*: *real-of-nat-less-inverse-iff*)
apply (*simp* (*no-asm-simp*) *add*: *real-of-nat-Suc less-diff-eq* [*symmetric*])
apply (*rule finite-real-of-nat-less-real*)
done

lemma *lemma-real-le-Un-eq2*:
{n. u ≤ inverse(real(Suc n))} =
{n. u < inverse(real(Suc n))} Un {n. u = inverse(real(Suc n))}
apply (*auto dest*: *order-le-imp-less-or-eq simp add*: *order-less-imp-le*)
done

lemma *real-of-nat-inverse-eq-iff*:
(u = inverse (real(Suc n))) = (real(Suc n) = inverse u)
by (*auto simp add*: *real-of-nat-Suc-gt-zero less-imp-neq* [*THEN not-sym*])

lemma *lemma-finite-omega-set2*: *finite {n::nat. u = inverse(real(Suc n))}*
apply (*simp* (*no-asm-simp*) *add*: *real-of-nat-inverse-eq-iff*)
apply (*cut-tac x = inverse u − 1* **in** *lemma-finite-omega-set*)
apply (*simp add*: *real-of-nat-Suc diff-eq-eq* [*symmetric*] *eq-commute*)
done

lemma *finite-inverse-real-of-posnat-ge-real*:
0 < u ==> finite {n. u ≤ inverse(real(Suc n))}
by (*auto simp add*: *lemma-real-le-Un-eq2 lemma-finite-omega-set2 finite-inverse-real-of-posnat-gt-real*)

lemma *inverse-real-of-posnat-ge-real-FreeUltrafilterNat*:
0 < u ==> {n. u ≤ inverse(real(Suc n))} ∉ FreeUltrafilterNat
by (*blast intro*!: *FreeUltrafilterNat.finite finite-inverse-real-of-posnat-ge-real*)

lemma *Compl-le-inverse-eq*:
− {n. u ≤ inverse(real(Suc n))} =
{n. inverse(real(Suc n)) < u}
apply (*auto dest*!: *order-le-less-trans simp add*: *linorder-not-le*)

done

lemma *FreeUltrafilterNat-inverse-real-of-posnat*:
$0 < u \Longrightarrow$
$\{n.\ inverse(real(Suc\ n)) < u\} \in FreeUltrafilterNat$
apply (*cut-tac u = u* **in** *inverse-real-of-posnat-ge-real-FreeUltrafilterNat*)
apply (*auto dest*: *FreeUltrafilterNat.not-memD simp add*: *Compl-le-inverse-eq*)
done

Example of an hypersequence (i.e. an extended standard sequence) whose term with an hypernatural suffix is an infinitesimal i.e. the whn'nth term of the hypersequence is a member of Infinitesimal

lemma *SEQ-Infinitesimal*:
$(\ *f*\ (\%n::nat.\ inverse(real(Suc\ n))))\ whn : Infinitesimal$
apply (*simp add*: *hypnat-omega-def starfun-star-n star-n-inverse*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff*)
apply (*simp add*: *real-of-nat-Suc-gt-zero FreeUltrafilterNat-inverse-real-of-posnat*)
done

Example where we get a hyperreal from a real sequence for which a particular property holds. The theorem is used in proofs about equivalence of nonstandard and standard neighbourhoods. Also used for equivalence of nonstandard ans standard definitions of pointwise limit.

lemma *real-seq-to-hypreal-Infinitesimal*:
$\forall n.\ norm(X\ n - x) < inverse(real(Suc\ n))$
$\Longrightarrow star\text{-}n\ X - star\text{-}of\ x \in Infinitesimal$
apply (*auto intro*!: *bexI dest*: *FreeUltrafilterNat-inverse-real-of-posnat FreeUltrafilterNat.Int intro*: *order-less-trans FreeUltrafilterNat.subset simp add*: *star-n-diff star-of-def Infinitesimal-FreeUltrafilterNat-iff star-n-inverse*)
done

lemma *real-seq-to-hypreal-approx*:
$\forall n.\ norm(X\ n - x) < inverse(real(Suc\ n))$
$\Longrightarrow star\text{-}n\ X\ @=\ star\text{-}of\ x$
apply (*subst approx-minus-iff*)
apply (*rule mem-infmal-iff* [*THEN subst*])
apply (*erule real-seq-to-hypreal-Infinitesimal*)
done

lemma *real-seq-to-hypreal-approx2*:
$\forall n.\ norm(x - X\ n) < inverse(real(Suc\ n))$
$\Longrightarrow star\text{-}n\ X\ @=\ star\text{-}of\ x$
apply (*rule real-seq-to-hypreal-approx*)
apply (*subst norm-minus-cancel* [*symmetric*])
apply (*simp del*: *norm-minus-cancel*)
done

lemma *real-seq-to-hypreal-Infinitesimal2*:

$\forall n.\ norm(X\ n - Y\ n) < inverse(real(Suc\ n))$
$\Longrightarrow star\text{-}n\ X - star\text{-}n\ Y \in Infinitesimal$
by (*auto intro*!: *bexI*
dest: *FreeUltrafilterNat-inverse-real-of-posnat*
FreeUltrafilterNat.Int
intro: *order-less-trans FreeUltrafilterNat.subset*
simp add: *Infinitesimal-FreeUltrafilterNat-iff star-n-diff*
star-n-inverse)

end

38 NSComplex: Nonstandard Complex Numbers

theory *NSComplex*
imports *Complex ../Hyperreal/NSA*
begin

types *hcomplex* = *complex star*

abbreviation
hcomplex-of-complex :: *complex* => *complex star* **where**
hcomplex-of-complex == *star-of*

abbreviation
hcmod :: *complex star* => *real star* **where**
hcmod == *hnorm*

definition
hRe :: *hcomplex* => *hypreal* **where**
hRe = $*f*$ *Re*

definition
hIm :: *hcomplex* => *hypreal* **where**
hIm = $*f*$ *Im*

definition
iii :: *hcomplex* **where**
iii = *star-of ii*

definition

hcnj :: *hcomplex* => *hcomplex* **where**
hcnj = ∗*f*∗ *cnj*

definition
hsgn :: *hcomplex* => *hcomplex* **where**
hsgn = ∗*f*∗ *sgn*

definition
harg :: *hcomplex* => *hypreal* **where**
harg = ∗*f*∗ *arg*

definition

hcis :: *hypreal* => *hcomplex* **where**
hcis = ∗*f*∗ *cis*

abbreviation
hcomplex-of-hypreal :: *hypreal* ⇒ *hcomplex* **where**
hcomplex-of-hypreal ≡ *of-hypreal*

definition

hrcis :: [*hypreal*, *hypreal*] => *hcomplex* **where**
hrcis = ∗*f2*∗ *rcis*

definition
hexpi :: *hcomplex* => *hcomplex* **where**
hexpi = ∗*f*∗ *expi*

definition
HComplex :: [*hypreal*,*hypreal*] => *hcomplex* **where**
HComplex = ∗*f2*∗ *Complex*

lemmas *hcomplex-defs* [*transfer-unfold*] =
hRe-def hIm-def iii-def hcnj-def hsgn-def harg-def hcis-def
hrcis-def hexpi-def HComplex-def

lemma *Standard-hRe* [*simp*]: *x* ∈ *Standard* ⟹ *hRe x* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hIm* [*simp*]: *x* ∈ *Standard* ⟹ *hIm x* ∈ *Standard*
by (*simp add*: *hcomplex-defs*)

lemma *Standard-iii* [*simp*]: *iii* ∈ *Standard*

by (*simp add*: *hcomplex-defs*)

lemma *Standard-hcnj* [*simp*]: $x \in Standard \Longrightarrow hcnj\ x \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hsgn* [*simp*]: $x \in Standard \Longrightarrow hsgn\ x \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-harg* [*simp*]: $x \in Standard \Longrightarrow harg\ x \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hcis* [*simp*]: $r \in Standard \Longrightarrow hcis\ r \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hexpi* [*simp*]: $x \in Standard \Longrightarrow hexpi\ x \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-hrcis* [*simp*]:
$[\![r \in Standard;\ s \in Standard]\!] \Longrightarrow hrcis\ r\ s \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *Standard-HComplex* [*simp*]:
$[\![r \in Standard;\ s \in Standard]\!] \Longrightarrow HComplex\ r\ s \in Standard$
by (*simp add*: *hcomplex-defs*)

lemma *hcmod-def*: $hcmod = *f* cmod$
by (*rule hnorm-def*)

38.1 Properties of Nonstandard Real and Imaginary Parts

lemma *hcomplex-hRe-hIm-cancel-iff*:
$!!w\ z.\ (w=z) = (hRe(w) = hRe(z)\ \&\ hIm(w) = hIm(z))$
by *transfer* (*rule complex-Re-Im-cancel-iff*)

lemma *hcomplex-equality* [*intro?*]:
$!!z\ w.\ hRe\ z = hRe\ w \Longrightarrow hIm\ z = hIm\ w \Longrightarrow z = w$
by *transfer* (*rule complex-equality*)

lemma *hcomplex-hRe-zero* [*simp*]: $hRe\ 0 = 0$
by *transfer* (*rule complex-Re-zero*)

lemma *hcomplex-hIm-zero* [*simp*]: $hIm\ 0 = 0$
by *transfer* (*rule complex-Im-zero*)

lemma *hcomplex-hRe-one* [*simp*]: $hRe\ 1 = 1$
by *transfer* (*rule complex-Re-one*)

lemma *hcomplex-hIm-one* [*simp*]: $hIm\ 1 = 0$
by *transfer* (*rule complex-Im-one*)

38.2 Addition for Nonstandard Complex Numbers

lemma *hRe-add*: $!!x\ y.\ hRe(x + y) = hRe(x) + hRe(y)$
by *transfer* (*rule complex-Re-add*)

lemma *hIm-add*: $!!x\ y.\ hIm(x + y) = hIm(x) + hIm(y)$
by *transfer* (*rule complex-Im-add*)

38.3 More Minus Laws

lemma *hRe-minus*: $!!z.\ hRe(-z) = -\ hRe(z)$
by *transfer* (*rule complex-Re-minus*)

lemma *hIm-minus*: $!!z.\ hIm(-z) = -\ hIm(z)$
by *transfer* (*rule complex-Im-minus*)

lemma *hcomplex-add-minus-eq-minus*:
$x + y = (0::hcomplex) ==> x = -y$
apply (*drule OrderedGroup.equals-zero-I*)
apply (*simp add*: *minus-equation-iff* [*of x y*])
done

lemma *hcomplex-i-mult-eq* [*simp*]: $iii * iii = -\ 1$
by *transfer* (*rule i-mult-eq2*)

lemma *hcomplex-i-mult-left* [*simp*]: $!!z.\ iii * (iii * z) = -z$
by *transfer* (*rule complex-i-mult-minus*)

lemma *hcomplex-i-not-zero* [*simp*]: $iii \neq 0$
by *transfer* (*rule complex-i-not-zero*)

38.4 More Multiplication Laws

lemma *hcomplex-mult-minus-one*: $-\ 1 * (z::hcomplex) = -z$
by *simp*

lemma *hcomplex-mult-minus-one-right*: $(z::hcomplex) * -\ 1 = -z$
by *simp*

lemma *hcomplex-mult-left-cancel*:
$(c::hcomplex) \neq (0::hcomplex) ==> (c*a=c*b) = (a=b)$
by *simp*

lemma *hcomplex-mult-right-cancel*:
$(c::hcomplex) \neq (0::hcomplex) ==> (a*c=b*c) = (a=b)$
by *simp*

38.5 Subraction and Division

lemma *hcomplex-diff-eq-eq* [*simp*]: $((x::hcomplex) - y = z) = (x = z + y)$

by (*rule OrderedGroup.diff-eq-eq*)

38.6 Embedding Properties for *hcomplex-of-hypreal* Map

lemma *hRe-hcomplex-of-hypreal* [*simp*]: !!z. *hRe*(*hcomplex-of-hypreal* z) = z
by *transfer* (*rule Re-complex-of-real*)

lemma *hIm-hcomplex-of-hypreal* [*simp*]: !!z. *hIm*(*hcomplex-of-hypreal* z) = *0*
by *transfer* (*rule Im-complex-of-real*)

lemma *hcomplex-of-hypreal-epsilon-not-zero* [*simp*]:
hcomplex-of-hypreal epsilon ≠ *0*
by (*simp add*: *hypreal-epsilon-not-zero*)

38.7 HComplex theorems

lemma *hRe-HComplex* [*simp*]: !!x y. *hRe* (*HComplex* x y) = x
by *transfer* (*rule Re*)

lemma *hIm-HComplex* [*simp*]: !!x y. *hIm* (*HComplex* x y) = y
by *transfer* (*rule Im*)

lemma *hcomplex-surj* [*simp*]: !!z. *HComplex* (*hRe* z) (*hIm* z) = z
by *transfer* (*rule complex-surj*)

lemma *hcomplex-induct* [*case-names rect*]:
(⋀x y. P (*HComplex* x y)) ==> P z
by (*rule hcomplex-surj* [*THEN subst*], *blast*)

38.8 Modulus (Absolute Value) of Nonstandard Complex Number

lemma *hcomplex-of-hypreal-abs*:
hcomplex-of-hypreal (*abs* x) =
hcomplex-of-hypreal(*hcmod*(*hcomplex-of-hypreal* x))
by *simp*

lemma *HComplex-inject* [*simp*]:
!!x y x' y'. *HComplex* x y = *HComplex* x' y' = (x=x' & y=y')
by *transfer* (*rule complex.inject*)

lemma *HComplex-add* [*simp*]:
!!*x1 y1 x2 y2*. *HComplex x1 y1* + *HComplex x2 y2* = *HComplex* (*x1*+*x2*) (*y1*+*y2*)
by *transfer* (*rule complex-add*)

lemma *HComplex-minus* [*simp*]: !!x y. − *HComplex* x y = *HComplex* (−x) (−y)
by *transfer* (*rule complex-minus*)

lemma *HComplex-diff* [*simp*]:

!!*x1 y1 x2 y2*. *HComplex x1 y1 − HComplex x2 y2 = HComplex* (*x1*−*x2*) (*y1*−*y2*)
by *transfer* (*rule complex-diff*)

lemma *HComplex-mult* [*simp*]:
!!*x1 y1 x2 y2*. *HComplex x1 y1 ∗ HComplex x2 y2 =*
HComplex (*x1∗x2 − y1∗y2*) (*x1∗y2 + y1∗x2*)
by *transfer* (*rule complex-mult*)

lemma *hcomplex-of-hypreal-eq*: !!*r*. *hcomplex-of-hypreal r = HComplex r 0*
by *transfer* (*rule complex-of-real-def*)

lemma *HComplex-add-hcomplex-of-hypreal* [*simp*]:
!!*x y r*. *HComplex x y + hcomplex-of-hypreal r = HComplex* (*x*+*r*) *y*
by *transfer* (*rule Complex-add-complex-of-real*)

lemma *hcomplex-of-hypreal-add-HComplex* [*simp*]:
!!*r x y*. *hcomplex-of-hypreal r + HComplex x y = HComplex* (*r*+*x*) *y*
by *transfer* (*rule complex-of-real-add-Complex*)

lemma *HComplex-mult-hcomplex-of-hypreal*:
!!*x y r*. *HComplex x y ∗ hcomplex-of-hypreal r = HComplex* (*x∗r*) (*y∗r*)
by *transfer* (*rule Complex-mult-complex-of-real*)

lemma *hcomplex-of-hypreal-mult-HComplex*:
!!*r x y*. *hcomplex-of-hypreal r ∗ HComplex x y = HComplex* (*r∗x*) (*r∗y*)
by *transfer* (*rule complex-of-real-mult-Complex*)

lemma *i-hcomplex-of-hypreal* [*simp*]:
!!*r*. *iii ∗ hcomplex-of-hypreal r = HComplex 0 r*
by *transfer* (*rule i-complex-of-real*)

lemma *hcomplex-of-hypreal-i* [*simp*]:
!!*r*. *hcomplex-of-hypreal r ∗ iii = HComplex 0 r*
by *transfer* (*rule complex-of-real-i*)

38.9 Conjugation

lemma *hcomplex-hcnj-cancel-iff* [*iff*]: !!*x y*. (*hcnj x = hcnj y*) = (*x = y*)
by *transfer* (*rule complex-cnj-cancel-iff*)

lemma *hcomplex-hcnj-hcnj* [*simp*]: !!*z*. *hcnj* (*hcnj z*) = *z*
by *transfer* (*rule complex-cnj-cnj*)

lemma *hcomplex-hcnj-hcomplex-of-hypreal* [*simp*]:
!!*x*. *hcnj* (*hcomplex-of-hypreal x*) = *hcomplex-of-hypreal x*
by *transfer* (*rule complex-cnj-complex-of-real*)

lemma *hcomplex-hmod-hcnj* [*simp*]: !!z. *hcmod* (*hcnj* z) = *hcmod* z
by *transfer* (*rule complex-mod-cnj*)

lemma *hcomplex-hcnj-minus*: !!z. *hcnj* ($-z$) = $-$ *hcnj* z
by *transfer* (*rule complex-cnj-minus*)

lemma *hcomplex-hcnj-inverse*: !!z. *hcnj*(*inverse* z) = *inverse*(*hcnj* z)
by *transfer* (*rule complex-cnj-inverse*)

lemma *hcomplex-hcnj-add*: !!w z. *hcnj*(w + z) = *hcnj*(w) + *hcnj*(z)
by *transfer* (*rule complex-cnj-add*)

lemma *hcomplex-hcnj-diff*: !!w z. *hcnj*(w − z) = *hcnj*(w) − *hcnj*(z)
by *transfer* (*rule complex-cnj-diff*)

lemma *hcomplex-hcnj-mult*: !!w z. *hcnj*(w * z) = *hcnj*(w) * *hcnj*(z)
by *transfer* (*rule complex-cnj-mult*)

lemma *hcomplex-hcnj-divide*: !!w z. *hcnj*(w / z) = (*hcnj* w)/(*hcnj* z)
by *transfer* (*rule complex-cnj-divide*)

lemma *hcnj-one* [*simp*]: *hcnj 1* = *1*
by *transfer* (*rule complex-cnj-one*)

lemma *hcomplex-hcnj-zero* [*simp*]: *hcnj 0* = *0*
by *transfer* (*rule complex-cnj-zero*)

lemma *hcomplex-hcnj-zero-iff* [*iff*]: !!z. (*hcnj* z = *0*) = (z = *0*)
by *transfer* (*rule complex-cnj-zero-iff*)

lemma *hcomplex-mult-hcnj*:
!!z. z * *hcnj* z = *hcomplex-of-hypreal* (*hRe*(z) ^ *2* + *hIm*(z) ^ *2*)
by *transfer* (*rule complex-mult-cnj*)

38.10 More Theorems about the Function *hcmod*

lemma *hcmod-hcomplex-of-hypreal-of-nat* [*simp*]:
hcmod (*hcomplex-of-hypreal*(*hypreal-of-nat* n)) = *hypreal-of-nat* n
by *simp*

lemma *hcmod-hcomplex-of-hypreal-of-hypnat* [*simp*]:
hcmod (*hcomplex-of-hypreal*(*hypreal-of-hypnat* n)) = *hypreal-of-hypnat* n
by *simp*

lemma *hcmod-mult-hcnj*: !!z. *hcmod*(z * *hcnj*(z)) = *hcmod*(z) ^ *2*
by *transfer* (*rule complex-mod-mult-cnj*)

lemma *hcmod-triangle-ineq2* [*simp*]:
!!a b. *hcmod*(b + a) − *hcmod* b ≤ *hcmod* a

by *transfer* (*rule complex-mod-triangle-ineq2*)

lemma *hcmod-diff-ineq* [*simp*]: !!*a b*. *hcmod*(*a*) − *hcmod*(*b*) ≤ *hcmod*(*a* + *b*)
by *transfer* (*rule norm-diff-ineq*)

38.11 Exponentiation

lemma *hcomplexpow-0* [*simp*]: *z* ^ *0* = (*1*::*hcomplex*)
by (*rule power-0*)

lemma *hcomplexpow-Suc* [*simp*]: *z* ^ (*Suc n*) = (*z*::*hcomplex*) ∗ (*z* ^ *n*)
by (*rule power-Suc*)

lemma *hcomplexpow-i-squared* [*simp*]: *iii* ^ *2* = −*1*
by *transfer* (*rule power2-i*)

lemma *hcomplex-of-hypreal-pow*:
!!*x*. *hcomplex-of-hypreal* (*x* ^ *n*) = (*hcomplex-of-hypreal x*) ^ *n*
by *transfer* (*rule of-real-power*)

lemma *hcomplex-hcnj-pow*: !!*z*. *hcnj*(*z* ^ *n*) = *hcnj*(*z*) ^ *n*
by *transfer* (*rule complex-cnj-power*)

lemma *hcmod-hcomplexpow*: !!*x*. *hcmod*(*x* ^ *n*) = *hcmod*(*x*) ^ *n*
by *transfer* (*rule norm-power*)

lemma *hcpow-minus*:
!!*x n*. (−*x*::*hcomplex*) *pow n* =
(*if* (∗*p*∗ *even*) *n then* (*x pow n*) *else* −(*x pow n*))
by *transfer* (*rule neg-power-if*)

lemma *hcpow-mult*:
!!*r s n*. ((*r*::*hcomplex*) ∗ *s*) *pow n* = (*r pow n*) ∗ (*s pow n*)
by *transfer* (*rule power-mult-distrib*)

lemma *hcpow-zero2* [*simp*]:
⋀*n*. *0 pow* (*hSuc n*) = (*0*::*'a*::{*recpower*,*semiring-0*} *star*)
by *transfer* (*rule power-0-Suc*)

lemma *hcpow-not-zero* [*simp*,*intro*]:
!!*r n*. *r* ≠ *0* ==> *r pow n* ≠ (*0*::*hcomplex*)
by (*rule hyperpow-not-zero*)

lemma *hcpow-zero-zero*: *r pow n* = (*0*::*hcomplex*) ==> *r* = *0*
by (*blast intro*: *ccontr dest*: *hcpow-not-zero*)

38.12 The Function *hsgn*

lemma *hsgn-zero* [*simp*]: *hsgn 0* = *0*
by *transfer* (*rule sgn-zero*)

lemma *hsgn-one* [*simp*]: *hsgn 1 = 1*
by *transfer* (*rule sgn-one*)

lemma *hsgn-minus*: !!*z*. *hsgn* (−*z*) = − *hsgn*(*z*)
by *transfer* (*rule sgn-minus*)

lemma *hsgn-eq*: !!*z*. *hsgn z = z / hcomplex-of-hypreal* (*hcmod z*)
by *transfer* (*rule sgn-eq*)

lemma *hcmod-i*: !!*x y*. *hcmod* (*HComplex x y*) = (∗*f*∗ *sqrt*) (*x ^ 2 + y ^ 2*)
by *transfer* (*rule complex-norm*)

lemma *hcomplex-eq-cancel-iff1* [*simp*]:
(*hcomplex-of-hypreal xa = HComplex x y*) = (*xa = x & y = 0*)
by (*simp add*: *hcomplex-of-hypreal-eq*)

lemma *hcomplex-eq-cancel-iff2* [*simp*]:
(*HComplex x y = hcomplex-of-hypreal xa*) = (*x = xa & y = 0*)
by (*simp add*: *hcomplex-of-hypreal-eq*)

lemma *HComplex-eq-0* [*simp*]: !!*x y*. (*HComplex x y = 0*) = (*x = 0 & y = 0*)
by *transfer* (*rule Complex-eq-0*)

lemma *HComplex-eq-1* [*simp*]: !!*x y*. (*HComplex x y = 1*) = (*x = 1 & y = 0*)
by *transfer* (*rule Complex-eq-1*)

lemma *i-eq-HComplex-0-1*: *iii = HComplex 0 1*
by *transfer* (*rule i-def* [*THEN meta-eq-to-obj-eq*])

lemma *HComplex-eq-i* [*simp*]: !!*x y*. (*HComplex x y = iii*) = (*x = 0 & y = 1*)
by *transfer* (*rule Complex-eq-i*)

lemma *hRe-hsgn* [*simp*]: !!*z*. *hRe*(*hsgn z*) = *hRe*(*z*)/*hcmod z*
by *transfer* (*rule Re-sgn*)

lemma *hIm-hsgn* [*simp*]: !!*z*. *hIm*(*hsgn z*) = *hIm*(*z*)/*hcmod z*
by *transfer* (*rule Im-sgn*)

lemma *hcomplex-inverse-complex-split*:
!!*x y*. *inverse*(*hcomplex-of-hypreal x + iii ∗ hcomplex-of-hypreal y*) =
hcomplex-of-hypreal(*x*/(*x ^ 2 + y ^ 2*)) −
iii ∗ hcomplex-of-hypreal(*y*/(*x ^ 2 + y ^ 2*))
by *transfer* (*rule complex-inverse-complex-split*)

lemma *HComplex-inverse*:
!!*x y*. *inverse* (*HComplex x y*) =
HComplex (*x*/(*x ^ 2 + y ^ 2*)) (−*y*/(*x ^ 2 + y ^ 2*))
by *transfer* (*rule complex-inverse*)

lemma *hRe-mult-i-eq*[*simp*]:
  !!*y*. *hRe* (*iii* ∗ *hcomplex-of-hypreal y*) = *0*
by *transfer simp*

lemma *hIm-mult-i-eq* [*simp*]:
  !!*y*. *hIm* (*iii* ∗ *hcomplex-of-hypreal y*) = *y*
by *transfer simp*

lemma *hcmod-mult-i* [*simp*]: !!*y*. *hcmod* (*iii* ∗ *hcomplex-of-hypreal y*) = *abs y*
by *transfer simp*

lemma *hcmod-mult-i2* [*simp*]: !!*y*. *hcmod* (*hcomplex-of-hypreal y* ∗ *iii*) = *abs y*
by *transfer simp*

lemma *cos-harg-i-mult-zero-pos*:
  !!*y*. *0* < *y* ==> (∗*f*∗ *cos*) (*harg*(*HComplex 0 y*)) = *0*
by *transfer* (*rule cos-arg-i-mult-zero-pos*)

lemma *cos-harg-i-mult-zero-neg*:
  !!*y*. *y* < *0* ==> (∗*f*∗ *cos*) (*harg*(*HComplex 0 y*)) = *0*
by *transfer* (*rule cos-arg-i-mult-zero-neg*)

lemma *cos-harg-i-mult-zero* [*simp*]:
  !!*y*. *y* ≠ *0* ==> (∗*f*∗ *cos*) (*harg*(*HComplex 0 y*)) = *0*
by *transfer* (*rule cos-arg-i-mult-zero*)

lemma *hcomplex-of-hypreal-zero-iff* [*simp*]:
  !!*y*. (*hcomplex-of-hypreal y* = *0*) = (*y* = *0*)
by *transfer* (*rule of-real-eq-0-iff*)

38.13 Polar Form for Nonstandard Complex Numbers

lemma *complex-split-polar2*:
  ∀ *n*. ∃ *r a*. (*z n*) = *complex-of-real r* ∗ (*Complex* (*cos a*) (*sin a*))
by (*blast intro*: *complex-split-polar*)

lemma *hcomplex-split-polar*:
  !!*z*. ∃ *r a*. *z* = *hcomplex-of-hypreal r* ∗ (*HComplex*((∗*f*∗ *cos*) *a*)((∗*f*∗ *sin*) *a*))
by *transfer* (*rule complex-split-polar*)

lemma *hcis-eq*:
  !!*a*. *hcis a* =
  (*hcomplex-of-hypreal*((∗*f*∗ *cos*) *a*) +
  iii ∗ *hcomplex-of-hypreal*((∗*f*∗ *sin*) *a*))

by *transfer* (*simp add*: *cis-def*)

lemma *hrcis-Ex*: !!*z*. ∃ *r a*. *z* = *hrcis r a*
by *transfer* (*rule rcis-Ex*)

lemma *hRe-hcomplex-polar* [*simp*]:
!!*r a*. *hRe* (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) =
r ∗ (∗*f*∗ *cos*) *a*
by *transfer simp*

lemma *hRe-hrcis* [*simp*]: !!*r a*. *hRe*(*hrcis r a*) = *r* ∗ (∗*f*∗ *cos*) *a*
by *transfer* (*rule Re-rcis*)

lemma *hIm-hcomplex-polar* [*simp*]:
!!*r a*. *hIm* (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) =
r ∗ (∗*f*∗ *sin*) *a*
by *transfer simp*

lemma *hIm-hrcis* [*simp*]: !!*r a*. *hIm*(*hrcis r a*) = *r* ∗ (∗*f*∗ *sin*) *a*
by *transfer* (*rule Im-rcis*)

lemma *hcmod-unit-one* [*simp*]:
!!*a*. *hcmod* (*HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) = *1*
by *transfer* (*rule cmod-unit-one*)

lemma *hcmod-complex-polar* [*simp*]:
!!*r a*. *hcmod* (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) =
abs r
by *transfer* (*rule cmod-complex-polar*)

lemma *hcmod-hrcis* [*simp*]: !!*r a*. *hcmod*(*hrcis r a*) = *abs r*
by *transfer* (*rule complex-mod-rcis*)

lemma *hcis-hrcis-eq*: !!*a*. *hcis a* = *hrcis 1 a*
by *transfer* (*rule cis-rcis-eq*)
declare *hcis-hrcis-eq* [*symmetric*, *simp*]

lemma *hrcis-mult*:
!!*a b r1 r2*. *hrcis r1 a* ∗ *hrcis r2 b* = *hrcis* (*r1*∗*r2*) (*a* + *b*)
by *transfer* (*rule rcis-mult*)

lemma *hcis-mult*: !!*a b*. *hcis a* ∗ *hcis b* = *hcis* (*a* + *b*)
by *transfer* (*rule cis-mult*)

lemma *hcis-zero* [*simp*]: *hcis 0* = *1*

by *transfer* (*rule cis-zero*)

lemma *hrcis-zero-mod* [*simp*]: !!*a*. *hrcis 0 a = 0*
by *transfer* (*rule rcis-zero-mod*)

lemma *hrcis-zero-arg* [*simp*]: !!*r*. *hrcis r 0 = hcomplex-of-hypreal r*
by *transfer* (*rule rcis-zero-arg*)

lemma *hcomplex-i-mult-minus* [*simp*]: !!*x*. *iii* * (*iii* * *x*) = − *x*
by *transfer* (*rule complex-i-mult-minus*)

lemma *hcomplex-i-mult-minus2* [*simp*]: *iii* * *iii* * *x* = − *x*
by *simp*

lemma *hcis-hypreal-of-nat-Suc-mult*:
!!*a*. *hcis* (*hypreal-of-nat* (*Suc n*) * *a*) =
hcis a * *hcis* (*hypreal-of-nat n* * *a*)
apply *transfer*
apply (*fold real-of-nat-def*)
apply (*rule cis-real-of-nat-Suc-mult*)
done

lemma *NSDeMoivre*: !!*a*. (*hcis a*) ^ *n* = *hcis* (*hypreal-of-nat n* * *a*)
apply *transfer*
apply (*fold real-of-nat-def*)
apply (*rule DeMoivre*)
done

lemma *hcis-hypreal-of-hypnat-Suc-mult*:
!! *a n*. *hcis* (*hypreal-of-hypnat* (*n* + *1*) * *a*) =
hcis a * *hcis* (*hypreal-of-hypnat n* * *a*)
by *transfer* (*fold real-of-nat-def*, *simp add*: *cis-real-of-nat-Suc-mult*)

lemma *NSDeMoivre-ext*:
!!*a n*. (*hcis a*) *pow n* = *hcis* (*hypreal-of-hypnat n* * *a*)
by *transfer* (*fold real-of-nat-def*, *rule DeMoivre*)

lemma *NSDeMoivre2*:
!!*a r*. (*hrcis r a*) ^ *n* = *hrcis* (*r* ^ *n*) (*hypreal-of-nat n* * *a*)
by *transfer* (*fold real-of-nat-def*, *rule DeMoivre2*)

lemma *DeMoivre2-ext*:
!! *a r n*. (*hrcis r a*) *pow n* = *hrcis* (*r pow n*) (*hypreal-of-hypnat n* * *a*)
by *transfer* (*fold real-of-nat-def*, *rule DeMoivre2*)

lemma *hcis-inverse* [*simp*]: !!*a*. *inverse*(*hcis a*) = *hcis* (−*a*)
by *transfer* (*rule cis-inverse*)

lemma *hrcis-inverse*: !!*a r*. *inverse*(*hrcis r a*) = *hrcis* (*inverse r*) (−*a*)

by *transfer* (*simp add*: *rcis-inverse inverse-eq-divide* [*symmetric*])

lemma *hRe-hcis* [*simp*]: !!*a*. *hRe*(*hcis a*) = (∗*f*∗ *cos*) *a*
by *transfer* (*rule Re-cis*)

lemma *hIm-hcis* [*simp*]: !!*a*. *hIm*(*hcis a*) = (∗*f*∗ *sin*) *a*
by *transfer* (*rule Im-cis*)

lemma *cos-n-hRe-hcis-pow-n*: (∗*f*∗ *cos*) (*hypreal-of-nat n* ∗ *a*) = *hRe*(*hcis a* ^ *n*)
by (*simp add*: *NSDeMoivre*)

lemma *sin-n-hIm-hcis-pow-n*: (∗*f*∗ *sin*) (*hypreal-of-nat n* ∗ *a*) = *hIm*(*hcis a* ^ *n*)
by (*simp add*: *NSDeMoivre*)

lemma *cos-n-hRe-hcis-hcpow-n*: (∗*f*∗ *cos*) (*hypreal-of-hypnat n* ∗ *a*) = *hRe*(*hcis a pow n*)
by (*simp add*: *NSDeMoivre-ext*)

lemma *sin-n-hIm-hcis-hcpow-n*: (∗*f*∗ *sin*) (*hypreal-of-hypnat n* ∗ *a*) = *hIm*(*hcis a pow n*)
by (*simp add*: *NSDeMoivre-ext*)

lemma *hexpi-add*: !!*a b*. *hexpi*(*a* + *b*) = *hexpi*(*a*) ∗ *hexpi*(*b*)
by *transfer* (*rule expi-add*)

38.14 *hcomplex-of-complex*: the Injection from type *complex* to to *hcomplex*

lemma *inj-hcomplex-of-complex*: *inj*(*hcomplex-of-complex*)

by (*rule inj-star-of*)

lemma *hcomplex-of-complex-i*: *iii* = *hcomplex-of-complex ii*
by (*rule iii-def*)

lemma *hRe-hcomplex-of-complex*:
hRe (*hcomplex-of-complex z*) = *hypreal-of-real* (*Re z*)
by *transfer* (*rule refl*)

lemma *hIm-hcomplex-of-complex*:
hIm (*hcomplex-of-complex z*) = *hypreal-of-real* (*Im z*)
by *transfer* (*rule refl*)

lemma *hcmod-hcomplex-of-complex*:
hcmod (*hcomplex-of-complex x*) = *hypreal-of-real* (*cmod x*)
by *transfer* (*rule refl*)

38.15 Numerals and Arithmetic

lemma *hcomplex-number-of-def*: (*number-of w* :: *hcomplex*) == *of-int w*
by *transfer* (*rule number-of-eq* [*THEN eq-reflection*])

lemma *hcomplex-of-hypreal-eq-hcomplex-of-complex*:
hcomplex-of-hypreal (*hypreal-of-real x*) =
hcomplex-of-complex (*complex-of-real x*)
by *transfer* (*rule refl*)

lemma *hcomplex-hypreal-number-of*:
hcomplex-of-complex (*number-of w*) = *hcomplex-of-hypreal*(*number-of w*)
by *transfer* (*rule of-real-number-of-eq* [*symmetric*])

lemma *hcomplex-number-of-hcnj* [*simp*]:
hcnj (*number-of v* :: *hcomplex*) = *number-of v*
by *transfer* (*rule complex-cnj-number-of*)

lemma *hcomplex-number-of-hcmod* [*simp*]:
hcmod(*number-of v* :: *hcomplex*) = *abs* (*number-of v* :: *hypreal*)
by *transfer* (*rule norm-number-of*)

lemma *hcomplex-number-of-hRe* [*simp*]:
hRe(*number-of v* :: *hcomplex*) = *number-of v*
by *transfer* (*rule complex-Re-number-of*)

lemma *hcomplex-number-of-hIm* [*simp*]:
hIm(*number-of v* :: *hcomplex*) = *0*
by *transfer* (*rule complex-Im-number-of*)

end

39 Star: Star-Transforms in Non-Standard Analysis

theory *Star*
imports *NSA*
begin

definition

starset-n :: (*nat* => *'a set*) => *'a star set* (∗*sn*∗ - [*80*] *80*) **where**
∗*sn*∗ *As* = *Iset* (*star-n As*)

definition
InternalSets :: *'a star set set* **where**
InternalSets = {*X*. ∃ *As*. *X* = ∗*sn*∗ *As*}

definition

is-starext :: [*'a star* => *'a star*, *'a* => *'a*] => *bool* **where**
is-starext F f = (∀ *x y*. ∃ *X* ∈ *Rep-star*(*x*). ∃ *Y* ∈ *Rep-star*(*y*).
((*y* = (*F x*)) = ({*n*. *Y n* = *f*(*X n*)} : *FreeUltrafilterNat*)))

definition

starfun-n :: (*nat* => (*'a* => *'b*)) => *'a star* => *'b star* (∗*fn*∗ - [*80*] *80*) **where**
∗*fn*∗ *F* = *Ifun* (*star-n F*)

definition
InternalFuns :: (*'a star* => *'b star*) *set* **where**
InternalFuns = {*X*. ∃ *F*. *X* = ∗*fn*∗ *F*}

lemma *no-choice*: ∀ *x*. ∃ *y*. *Q x y* ==> ∃ (*f* :: *'a* => *nat*). ∀ *x*. *Q x* (*f x*)
apply (*rule-tac x* = %*x*. *LEAST y*. *Q x y* **in** *exI*)
apply (*blast intro*: *LeastI*)
done

39.1 Properties of the Star-transform Applied to Sets of Reals

lemma *STAR-star-of-image-subset*: *star-of* ' *A* <= ∗*s*∗ *A*
by *auto*

lemma *STAR-hypreal-of-real-Int*: ∗*s*∗ *X Int Reals* = *hypreal-of-real* ' *X*
by (*auto simp add*: *SReal-def*)

lemma *STAR-star-of-Int*: ∗*s*∗ *X Int Standard* = *star-of* ' *X*
by (*auto simp add*: *Standard-def*)

lemma *lemma-not-hyprealA*: *x* ∉ *hypreal-of-real* ' *A* ==> ∀ *y* ∈ *A*. *x* ≠ *hypreal-of-real y*
by *auto*

lemma *lemma-not-starA*: *x* ∉ *star-of* ' *A* ==> ∀ *y* ∈ *A*. *x* ≠ *star-of y*
by *auto*

lemma *lemma-Compl-eq*: $-\{n.\ X\ n = xa\} = \{n.\ X\ n \neq xa\}$
by *auto*

lemma *STAR-real-seq-to-hypreal*:
$\forall n.\ (X\ n) \notin M \Longrightarrow star\text{-}n\ X \notin *s*\ M$
apply (*unfold starset-def star-of-def*)
apply (*simp add*: *Iset-star-n*)
done

lemma *STAR-singleton*: $*s*\ \{x\} = \{star\text{-}of\ x\}$
by *simp*

lemma *STAR-not-mem*: $x \notin F \Longrightarrow star\text{-}of\ x \notin *s*\ F$
by *transfer*

lemma *STAR-subset-closed*: $[|\ x : *s*\ A;\ A <= B\ |] \Longrightarrow x : *s*\ B$
by (*erule rev-subsetD*, *simp*)

Nonstandard extension of a set (defined using a constant sequence) as a special case of an internal set

lemma *starset-n-starset*: $\forall n.\ (As\ n = A) \Longrightarrow *sn*\ As = *s*\ A$
apply (*drule expand-fun-eq* [*THEN iffD2*])
apply (*simp add*: *starset-n-def starset-def star-of-def*)
done

lemma *starfun-n-starfun*: $\forall n.\ (F\ n = f) \Longrightarrow *fn*\ F = *f*\ f$
apply (*drule expand-fun-eq* [*THEN iffD2*])
apply (*simp add*: *starfun-n-def starfun-def star-of-def*)
done

lemma *hrabs-is-starext-rabs*: *is-starext abs abs*
apply (*simp add*: *is-starext-def*, *safe*)
apply (*rule-tac x=x* **in** *star-cases*)
apply (*rule-tac x=y* **in** *star-cases*)
apply (*unfold star-n-def*, *auto*)

apply (*rule bexI*, *rule-tac* [*2*] *lemma-starrel-refl*)
apply (*rule bexI*, *rule-tac* [*2*] *lemma-starrel-refl*)
apply (*fold star-n-def*)
apply (*unfold star-abs-def starfun-def star-of-def*)
apply (*simp add*: *Ifun-star-n star-n-eq-iff*)
done

Nonstandard extension of functions

lemma *starfun*:
($*f*$ *f*) (*star-n X*) = *star-n* (%*n*. *f* (*X n*))
by (*rule starfun-star-n*)

lemma *starfun-if-eq*:
!!*w*. *w* ≠ *star-of x*
==> ($*f*$ (λ*z*. *if z* = *x then a else g z*)) *w* = ($*f*$ *g*) *w*
by (*transfer*, *simp*)

lemma *starfun-mult*: !!*x*. ($*f*$ *f*) *x* * ($*f*$ *g*) *x* = ($*f*$ (%*x*. *f x* * *g x*)) *x*
by (*transfer*, *rule refl*)
declare *starfun-mult* [*symmetric*, *simp*]

lemma *starfun-add*: !!*x*. ($*f*$ *f*) *x* + ($*f*$ *g*) *x* = ($*f*$ (%*x*. *f x* + *g x*)) *x*
by (*transfer*, *rule refl*)
declare *starfun-add* [*symmetric*, *simp*]

lemma *starfun-minus*: !!*x*. − ($*f*$ *f*) *x* = ($*f*$ (%*x*. − *f x*)) *x*
by (*transfer*, *rule refl*)
declare *starfun-minus* [*symmetric*, *simp*]

lemma *starfun-add-minus*: !!*x*. ($*f*$ *f*) *x* + −($*f*$ *g*) *x* = ($*f*$ (%*x*. *f x* + −*g x*)) *x*
by (*transfer*, *rule refl*)
declare *starfun-add-minus* [*symmetric*, *simp*]

lemma *starfun-diff*: !!*x*. ($*f*$ *f*) *x* − ($*f*$ *g*) *x* = ($*f*$ (%*x*. *f x* − *g x*)) *x*
by (*transfer*, *rule refl*)
declare *starfun-diff* [*symmetric*, *simp*]

lemma *starfun-o2*: (%*x*. ($*f*$ *f*) (($*f*$ *g*) *x*)) = $*f*$ (%*x*. *f* (*g x*))
by (*transfer*, *rule refl*)

lemma *starfun-o*: ($*f*$ *f*) *o* ($*f*$ *g*) = ($*f*$ (*f o g*))
by (*transfer o-def*, *rule refl*)

NS extension of constant function

lemma *starfun-const-fun* [*simp*]: !!x. ($*f*$ (%x. k)) x = *star-of* k
by (*transfer*, *rule refl*)

the NS extension of the identity function

lemma *starfun-Id* [*simp*]: !!x. ($*f*$ (%x. x)) x = x
by (*transfer*, *rule refl*)

lemma *starfun-Idfun-approx*:
  x @= *star-of* a ==> ($*f*$ (%x. x)) x @= *star-of* a
by (*simp only*: *starfun-Id*)

The Star-function is a (nonstandard) extension of the function

lemma *is-starext-starfun*: *is-starext* ($*f*$ f) f
apply (*simp add*: *is-starext-def*, *auto*)
apply (*rule-tac* x = x **in** *star-cases*)
apply (*rule-tac* x = y **in** *star-cases*)
apply (*auto intro*!: *bexI* [*OF - Rep-star-star-n*]
        simp add: *starfun star-n-eq-iff*)
done

Any nonstandard extension is in fact the Star-function

lemma *is-starfun-starext*: *is-starext* F f ==> F = $*f*$ f
apply (*simp add*: *is-starext-def*)
apply (*rule ext*)
apply (*rule-tac* x = x **in** *star-cases*)
apply (*drule-tac* x = x **in** *spec*)
apply (*drule-tac* x = ($*f*$ f) x **in** *spec*)
apply (*auto simp add*: *starfun-star-n*)
apply (*simp add*: *star-n-eq-iff* [*symmetric*])
apply (*simp add*: *starfun-star-n* [*of* f, *symmetric*])
done

lemma *is-starext-starfun-iff*: (*is-starext* F f) = (F = $*f*$ f)
by (*blast intro*: *is-starfun-starext is-starext-starfun*)

extented function has same solution as its standard version for real arguments. i.e they are the same for all real arguments

lemma *starfun-eq*: ($*f*$ f) (*star-of* a) = *star-of* (f a)
by (*rule starfun-star-of*)

lemma *starfun-approx*: ($*f*$ f) (*star-of* a) @= *star-of* (f a)
by *simp*

lemma *starfun-lambda-cancel*:
  !!x'. ($*f*$ (%h. f (x + h))) x' = ($*f*$ f) (*star-of* x + x')

by (*transfer*, *rule refl*)

lemma *starfun-lambda-cancel2*:
($*f*$ (%*h*. *f*(*g*(*x* + *h*)))) *x*′ = ($*f*$ (*f* *o* *g*)) (*star-of x* + *x*′)
by (*unfold o-def*, *rule starfun-lambda-cancel*)

lemma *starfun-mult-HFinite-approx*:
fixes *l m* :: ′*a*::*real-normed-algebra star*
shows [| ($*f*$ *f*) *x* @= *l*; ($*f*$ *g*) *x* @= *m*;
l: *HFinite*; *m*: *HFinite*
|] ==> ($*f*$ (%*x*. *f x* ∗ *g x*)) *x* @= *l* ∗ *m*
apply (*drule* (*3*) *approx-mult-HFinite*)
apply (*auto intro*: *approx-HFinite* [*OF* - *approx-sym*])
done

lemma *starfun-add-approx*: [| ($*f*$ *f*) *x* @= *l*; ($*f*$ *g*) *x* @= *m*
|] ==> ($*f*$ (%*x*. *f x* + *g x*)) *x* @= *l* + *m*
by (*auto intro*: *approx-add*)

Examples: hrabs is nonstandard extension of rabs inverse is nonstandard extension of inverse

lemma *starfun-rabs-hrabs*: $*f*$ *abs* = *abs*
by (*simp only*: *star-abs-def*)

lemma *starfun-inverse-inverse* [*simp*]: ($*f*$ *inverse*) *x* = *inverse*(*x*)
by (*simp only*: *star-inverse-def*)

lemma *starfun-inverse*: !!*x*. *inverse* (($*f*$ *f*) *x*) = ($*f*$ (%*x*. *inverse* (*f x*))) *x*
by (*transfer*, *rule refl*)
declare *starfun-inverse* [*symmetric*, *simp*]

lemma *starfun-divide*: !!*x*. ($*f*$ *f*) *x* / ($*f*$ *g*) *x* = ($*f*$ (%*x*. *f x* / *g x*)) *x*
by (*transfer*, *rule refl*)
declare *starfun-divide* [*symmetric*, *simp*]

lemma *starfun-inverse2*: !!*x*. *inverse* (($*f*$ *f*) *x*) = ($*f*$ (%*x*. *inverse* (*f x*))) *x*
by (*transfer*, *rule refl*)

General lemma/theorem needed for proofs in elementary topology of the reals

lemma *starfun-mem-starset*:
!!*x*. ($*f*$ *f*) *x* : $*s*$ *A* ==> *x* : $*s*$ {*x*. *f x* ∈ *A*}
by (*transfer*, *simp*)

Alternative definition for hrabs with rabs function applied entrywise to equivalence class representative. This is easily proved using starfun and ns extension thm

lemma *hypreal-hrabs*:

abs (star-n X) = star-n (%n. abs (X n))
by (*simp only*: *starfun-rabs-hrabs* [*symmetric*] *starfun*)

nonstandard extension of set through nonstandard extension of rabs function i.e hrabs. A more general result should be where we replace rabs by some arbitrary function f and hrabs by its NS extenson. See second NS set extension below.

lemma *STAR-rabs-add-minus*:
∗s∗ {x. abs (x + − y) < r} =
{x. abs(x + −star-of y) < star-of r}
by (*transfer*, *rule refl*)

lemma *STAR-starfun-rabs-add-minus*:
∗s∗ {x. abs (f x + − y) < r} =
{x. abs((∗f∗ f) x + −star-of y) < star-of r}
by (*transfer*, *rule refl*)

Another characterization of Infinitesimal and one of @= relation. In this theory since *hypreal-hrabs* proved here. Maybe move both theorems??

lemma *Infinitesimal-FreeUltrafilterNat-iff2*:
(star-n X ∈ Infinitesimal) =
(∀ m. {n. norm(X n) < inverse(real(Suc m))}
∈ FreeUltrafilterNat)
by (*simp add*: *Infinitesimal-hypreal-of-nat-iff star-of-def*
hnorm-def star-of-nat-def starfun-star-n
star-n-inverse star-n-less real-of-nat-def)

lemma *HNatInfinite-inverse-Infinitesimal* [*simp*]:
n ∈ HNatInfinite ==> inverse (hypreal-of-hypnat n) ∈ Infinitesimal
apply (*cases n*)
apply (*auto simp add*: *of-hypnat-def starfun-star-n real-of-nat-def* [*symmetric*]
star-n-inverse real-norm-def
HNatInfinite-FreeUltrafilterNat-iff
Infinitesimal-FreeUltrafilterNat-iff2)
apply (*drule-tac x=Suc m* **in** *spec*)
apply (*erule ultra*, *simp*)
done

lemma *approx-FreeUltrafilterNat-iff*: *star-n X @= star-n Y =*
(∀ r>0. {n. norm (X n − Y n) < r} : FreeUltrafilterNat)
apply (*subst approx-minus-iff*)
apply (*rule mem-infmal-iff* [*THEN subst*])
apply (*simp add*: *star-n-diff*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff*)
done

lemma *approx-FreeUltrafilterNat-iff2*: *star-n X @= star-n Y =*
(∀ m. {n. norm (X n − Y n) <
inverse(real(Suc m))} : FreeUltrafilterNat)

apply (*subst approx-minus-iff*)
apply (*rule mem-infmal-iff* [*THEN subst*])
apply (*simp add*: *star-n-diff*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff2*)
done

lemma *inj-starfun*: *inj starfun*
apply (*rule inj-onI*)
apply (*rule ext*, *rule ccontr*)
apply (*drule-tac x = star-n* (%*n*. *xa*) **in** *fun-cong*)
apply (*auto simp add*: *starfun star-n-eq-iff*)
done

end

40 NatStar: Star-transforms for the Hypernaturals

theory *NatStar*
imports *Star*
begin

lemma *star-n-eq-starfun-whn*: *star-n X* = (∗*f*∗ *X*) *whn*
by (*simp add*: *hypnat-omega-def starfun-def star-of-def Ifun-star-n*)

lemma *starset-n-Un*: ∗*sn*∗ (%*n*. (*A n*) *Un* (*B n*)) = ∗*sn*∗ *A Un* ∗*sn*∗ *B*
apply (*simp add*: *starset-n-def star-n-eq-starfun-whn Un-def*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-Un*:
[| *X* ∈ *InternalSets*; *Y* ∈ *InternalSets* |]
==> (*X Un Y*) ∈ *InternalSets*
by (*auto simp add*: *InternalSets-def starset-n-Un* [*symmetric*])

lemma *starset-n-Int*:
∗*sn*∗ (%*n*. (*A n*) *Int* (*B n*)) = ∗*sn*∗ *A Int* ∗*sn*∗ *B*
apply (*simp add*: *starset-n-def star-n-eq-starfun-whn Int-def*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-Int*:
[| *X* ∈ *InternalSets*; *Y* ∈ *InternalSets* |]
==> (*X Int Y*) ∈ *InternalSets*
by (*auto simp add*: *InternalSets-def starset-n-Int* [*symmetric*])

lemma *starset-n-Compl*: ∗*sn*∗ ((%*n*. − *A n*)) = −(∗*sn*∗ *A*)

apply (*simp add*: *starset-n-def star-n-eq-starfun-whn Compl-eq*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-Compl*: $X \in InternalSets \Longrightarrow -X \in InternalSets$
by (*auto simp add*: *InternalSets-def starset-n-Compl* [*symmetric*])

lemma *starset-n-diff*: $*sn* (\%n.\ (A\ n) - (B\ n)) = *sn* A - *sn* B$
apply (*simp add*: *starset-n-def star-n-eq-starfun-whn set-diff-eq*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-diff*:
$[|\ X \in InternalSets;\ Y \in InternalSets\ |] \Longrightarrow (X - Y) \in InternalSets$
by (*auto simp add*: *InternalSets-def starset-n-diff* [*symmetric*])

lemma *NatStar-SHNat-subset*: $Nats \leq *s* (UNIV :: nat\ set)$
by *simp*

lemma *NatStar-hypreal-of-real-Int*:
$*s* X\ Int\ Nats = hypnat\text{-}of\text{-}nat\ `\ X$
by (*auto simp add*: *SHNat-eq*)

lemma *starset-starset-n-eq*: $*s* X = *sn* (\%n.\ X)$
by (*simp add*: *starset-n-starset*)

lemma *InternalSets-starset-n* [*simp*]: $(*s* X) \in InternalSets$
by (*auto simp add*: *InternalSets-def starset-starset-n-eq*)

lemma *InternalSets-UNIV-diff*:
$X \in InternalSets \Longrightarrow UNIV - X \in InternalSets$
apply (*subgoal-tac UNIV* $- X = -X$)
by (*auto intro*: *InternalSets-Compl*)

40.1 Nonstandard Extensions of Functions

Example of transfer of a property from reals to hyperreals — used for limit comparison of sequences

lemma *starfun-le-mono*:
$\forall n.\ N \leq n \longrightarrow f\ n \leq g\ n \Longrightarrow \forall n.\ hypnat\text{-}of\text{-}nat\ N \leq n \longrightarrow (*f* f)\ n \leq (*f* g)\ n$
by *transfer*

lemma *starfun-less-mono*:
$\forall n.\ N \leq n \longrightarrow f\ n < g\ n \Longrightarrow \forall n.\ hypnat\text{-}of\text{-}nat\ N \leq n \longrightarrow (*f* f)\ n < (*f* g)\ n$
by *transfer*

Nonstandard extension when we increment the argument by one

lemma *starfun-shift-one*:
  $!!N.\ (*f*\ (\%n.\ f\ (Suc\ n)))\ N = (*f*\ f)\ (N + (1::hypnat))$
by (*transfer*, *simp*)

Nonstandard extension with absolute value

lemma *starfun-abs*: $!!N.\ (*f*\ (\%n.\ abs\ (f\ n)))\ N = abs((*f*\ f)\ N)$
by (*transfer*, *rule refl*)

The hyperpow function as a nonstandard extension of realpow

lemma *starfun-pow*: $!!N.\ (*f*\ (\%n.\ r\ \hat{}\ n))\ N = (hypreal\text{-}of\text{-}real\ r)\ pow\ N$
by (*transfer*, *rule refl*)

lemma *starfun-pow2*:
  $!!N.\ (*f*\ (\%n.\ (X\ n)\ \hat{}\ m))\ N = (*f*\ X)\ N\ pow\ hypnat\text{-}of\text{-}nat\ m$
by (*transfer*, *rule refl*)

lemma *starfun-pow3*: $!!R.\ (*f*\ (\%r.\ r\ \hat{}\ n))\ R = (R)\ pow\ hypnat\text{-}of\text{-}nat\ n$
by (*transfer*, *rule refl*)

The *hypreal-of-hypnat* function as a nonstandard extension of *real-of-nat*

lemma *starfunNat-real-of-nat*: $(*f*\ real) = hypreal\text{-}of\text{-}hypnat$
by *transfer* (*simp add*: *expand-fun-eq real-of-nat-def*)

lemma *starfun-inverse-real-of-nat-eq*:
  $N \in HNatInfinite$
  $\Longrightarrow (*f*\ (\%x::nat.\ inverse(real\ x)))\ N = inverse(hypreal\text{-}of\text{-}hypnat\ N)$
apply (*rule-tac f1* $=$ *inverse* **in** *starfun-o2* [*THEN subst*])
apply (*subgoal-tac hypreal-of-hypnat N* $\sim=$ *0*)
apply (*simp-all add*: *zero-less-HNatInfinite starfunNat-real-of-nat starfun-inverse-inverse*)
done

Internal functions - some redundancy with *f* now

lemma *starfun-n*: $(*fn*\ f)\ (star\text{-}n\ X) = star\text{-}n\ (\%n.\ f\ n\ (X\ n))$
by (*simp add*: *starfun-n-def Ifun-star-n*)

Multiplication: $(*fn)\ x\ (*gn) = *(fn\ x\ gn)$

lemma *starfun-n-mult*:
  $(*fn*\ f)\ z * (*fn*\ g)\ z = (*fn*\ (\%\ i\ x.\ f\ i\ x * g\ i\ x))\ z$
apply (*cases z*)
apply (*simp add*: *starfun-n star-n-mult*)
done

Addition: $(*fn) + (*gn) = *(fn + gn)$

lemma *starfun-n-add*:
  $(*fn*\ f)\ z + (*fn*\ g)\ z = (*fn*\ (\%i\ x.\ f\ i\ x + g\ i\ x))\ z$
apply (*cases z*)

apply (*simp add*: *starfun-n star-n-add*)
done

Subtraction: $(*fn) - (*gn) = *(fn + - gn)$

lemma *starfun-n-add-minus*:
$(*fn* f) z + -(*fn* g) z = (*fn* (\%i x. f i x + -g i x)) z$
apply (*cases z*)
apply (*simp add*: *starfun-n star-n-minus star-n-add*)
done

Composition: $(*fn) o (*gn) = *(fn o gn)$

lemma *starfun-n-const-fun* [*simp*]:
$(*fn* (\%i x. k)) z = star\text{-}of k$
apply (*cases z*)
apply (*simp add*: *starfun-n star-of-def*)
done

lemma *starfun-n-minus*: $- (*fn* f) x = (*fn* (\%i x. - (f i) x)) x$
apply (*cases x*)
apply (*simp add*: *starfun-n star-n-minus*)
done

lemma *starfun-n-eq* [*simp*]:
$(*fn* f) (star\text{-}of n) = star\text{-}n (\%i. f i n)$
by (*simp add*: *starfun-n star-of-def*)

lemma *starfun-eq-iff*: $((*f* f) = (*f* g)) = (f = g)$
by (*transfer*, *rule refl*)

lemma *starfunNat-inverse-real-of-nat-Infinitesimal* [*simp*]:
$N \in HNatInfinite ==> (*f* (\%x. inverse (real x))) N \in Infinitesimal$
apply (*rule-tac f1 = inverse* **in** *starfun-o2* [*THEN subst*])
apply (*subgoal-tac hypreal-of-hypnat N ~= 0*)
apply (*simp-all add*: *zero-less-HNatInfinite starfunNat-real-of-nat*)
done

40.2 Nonstandard Characterization of Induction

lemma *hypnat-induct-obj*:
$!!n. ((*p* P) (0::hypnat) \&$
$(\forall n. (*p* P)(n) --> (*p* P)(n + 1)))$
$--> (*p* P)(n)$
by (*transfer*, *induct-tac n*, *auto*)

lemma *hypnat-induct*:
$!!n. [| (*p* P) (0::hypnat);$
$!!n. (*p* P)(n) ==> (*p* P)(n + 1)|]$
$==> (*p* P)(n)$
by (*transfer*, *induct-tac n*, *auto*)

lemma *starP2-eq-iff*: ($*p2*$ (*op* =)) = (*op* =)
by *transfer* (*rule refl*)

lemma *starP2-eq-iff2*: ($*p2*$ (%*x y*. *x* = *y*)) *X Y* = (*X* = *Y*)
by (*simp add*: *starP2-eq-iff*)

lemma *nonempty-nat-set-Least-mem*:
c ∈ (*S* :: *nat set*) ==> (*LEAST n*. *n* ∈ *S*) ∈ *S*
by (*erule LeastI*)

lemma *nonempty-set-star-has-least*:
!!*S*::*nat set star*. *Iset S* ≠ {} ==> ∃ *n* ∈ *Iset S*. ∀ *m* ∈ *Iset S*. *n* ≤ *m*
apply (*transfer empty-def*)
apply (*rule-tac x*=*LEAST n*. *n* ∈ *S* **in** *bexI*)
apply (*simp add*: *Least-le*)
apply (*rule LeastI-ex*, *auto*)
done

lemma *nonempty-InternalNatSet-has-least*:
[| (*S*::*hypnat set*) ∈ *InternalSets*; *S* ≠ {} |] ==> ∃ *n* ∈ *S*. ∀ *m* ∈ *S*. *n* ≤ *m*
apply (*clarsimp simp add*: *InternalSets-def starset-n-def*)
apply (*erule nonempty-set-star-has-least*)
done

Goldblatt page 129 Thm 11.3.2

lemma *internal-induct-lemma*:
!!*X*::*nat set star*. [| (*0*::*hypnat*) ∈ *Iset X*; ∀ *n*. *n* ∈ *Iset X* −−> *n* + *1* ∈ *Iset X* |]
==> *Iset X* = (*UNIV*:: *hypnat set*)
apply (*transfer UNIV-def*)
apply (*rule equalityI* [*OF subset-UNIV subsetI*])
apply (*induct-tac x*, *auto*)
done

lemma *internal-induct*:
[| *X* ∈ *InternalSets*; (*0*::*hypnat*) ∈ *X*; ∀ *n*. *n* ∈ *X* −−> *n* + *1* ∈ *X* |]
==> *X* = (*UNIV*:: *hypnat set*)
apply (*clarsimp simp add*: *InternalSets-def starset-n-def*)
apply (*erule* (*1*) *internal-induct-lemma*)
done

end

41 HSEQ: Sequences and Convergence (Nonstandard)

theory *HSEQ*
imports *SEQ NatStar*
begin

definition
NSLIMSEQ :: [*nat* => *'a*::*real-normed-vector*, *'a*] => *bool*
(((-)/ ----*NS*> (-)) [*60*, *60*] *60*) **where**
— Nonstandard definition of convergence of sequence
X ----*NS*> *L* = (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* ≈ *star-of L*)

definition
nslim :: (*nat* => *'a*::*real-normed-vector*) => *'a* **where**
— Nonstandard definition of limit using choice operator
nslim X = (*THE L*. *X* ----*NS*> *L*)

definition
NSconvergent :: (*nat* => *'a*::*real-normed-vector*) => *bool* **where**
— Nonstandard definition of convergence
NSconvergent X = (∃ *L*. *X* ----*NS*> *L*)

definition
NSBseq :: (*nat* => *'a*::*real-normed-vector*) => *bool* **where**
— Nonstandard definition for bounded sequence
NSBseq X = (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* : *HFinite*)

definition
NSCauchy :: (*nat* => *'a*::*real-normed-vector*) => *bool* **where**
— Nonstandard definition
NSCauchy X = (∀ *M* ∈ *HNatInfinite*. ∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *M* ≈ (∗*f*∗ *X*) *N*)

41.1 Limits of Sequences

lemma *NSLIMSEQ-iff*:
(*X* ----*NS*> *L*) = (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* ≈ *star-of L*)
by (*simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-I*:
(⋀*N*. *N* ∈ *HNatInfinite* ⟹ *starfun X N* ≈ *star-of L*) ⟹ *X* ----*NS*> *L*
by (*simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-D*:
⟦*X* ----*NS*> *L*; *N* ∈ *HNatInfinite*⟧ ⟹ *starfun X N* ≈ *star-of L*
by (*simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-const*: (%*n*. *k*) ----*NS*> *k*

by (*simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-add*:
[| X −−−−*NS*> a; Y −−−−*NS*> b |] ==> (%n. $X\ n + Y\ n$) −−−−*NS*> $a + b$
by (*auto intro*: *approx-add simp add*: *NSLIMSEQ-def starfun-add* [*symmetric*])

lemma *NSLIMSEQ-add-const*: f −−−−*NS*> a ==> (%n.($f\ n + b$)) −−−−*NS*> $a + b$
by (*simp only*: *NSLIMSEQ-add NSLIMSEQ-const*)

lemma *NSLIMSEQ-mult*:
fixes $a\ b$ :: 'a::*real-normed-algebra*
shows [| X −−−−*NS*> a; Y −−−−*NS*> b |] ==> (%n. $X\ n * Y\ n$) −−−−*NS*> $a * b$
by (*auto intro*!: *approx-mult-HFinite simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-minus*: X −−−−*NS*> a ==> (%n. $-(X\ n)$) −−−−*NS*> $-a$
by (*auto simp add*: *NSLIMSEQ-def*)

lemma *NSLIMSEQ-minus-cancel*: (%n. $-(X\ n)$) −−−−*NS*> $-a$ ==> X −−−−*NS*> a
by (*drule NSLIMSEQ-minus*, *simp*)

lemma *NSLIMSEQ-add-minus*:
[| X −−−−*NS*> a; Y −−−−*NS*> b |] ==> (%n. $X\ n + -Y\ n$) −−−−*NS*> $a + -b$
by (*simp add*: *NSLIMSEQ-add NSLIMSEQ-minus*)

lemma *NSLIMSEQ-diff*:
[| X −−−−*NS*> a; Y −−−−*NS*> b |] ==> (%n. $X\ n - Y\ n$) −−−−*NS*> $a - b$
by (*simp add*: *diff-minus NSLIMSEQ-add NSLIMSEQ-minus*)

lemma *NSLIMSEQ-diff-const*: f −−−−*NS*> a ==> (%n.($f\ n - b$)) −−−−*NS*> $a - b$
by (*simp add*: *NSLIMSEQ-diff NSLIMSEQ-const*)

lemma *NSLIMSEQ-inverse*:
fixes a :: 'a::*real-normed-div-algebra*
shows [| X −−−−*NS*> a; a ~= *0* |] ==> (%n. *inverse*($X\ n$)) −−−−*NS*> *inverse*(a)
by (*simp add*: *NSLIMSEQ-def star-of-approx-inverse*)

lemma *NSLIMSEQ-mult-inverse*:
fixes $a\ b$ :: 'a::*real-normed-field*
shows
[| X −−−−*NS*> a; Y −−−−*NS*> b; b ~= *0* |] ==> (%n. $X\ n$ / $Y\ n$)

−−−−NS> a/b
by (*simp add*: *NSLIMSEQ-mult NSLIMSEQ-inverse divide-inverse*)

lemma *starfun-hnorm*: $\bigwedge x.\ hnorm\ ((\ *f*\ f)\ x) = (\ *f*\ (\lambda x.\ norm\ (f\ x)))\ x$
by *transfer simp*

lemma *NSLIMSEQ-norm*: X *−−−−NS>* $a \Longrightarrow (\lambda n.\ norm\ (X\ n))$ *−−−−NS>* *norm a*
by (*simp add*: *NSLIMSEQ-def starfun-hnorm* [*symmetric*] *approx-hnorm*)

Uniqueness of limit

lemma *NSLIMSEQ-unique*: [| X *−−−−NS>* a; X *−−−−NS>* b |] ==> $a = b$
apply (*simp add*: *NSLIMSEQ-def*)
apply (*drule HNatInfinite-whn* [*THEN* [*2*] *bspec*])+
apply (*auto dest*: *approx-trans3*)
done

lemma *NSLIMSEQ-pow* [*rule-format*]:
fixes a :: $'a$::{*real-normed-algebra*,*recpower*}
shows (X *−−−−NS>* a) −−> (($\%n.\ (X\ n)\ \hat{}\ m$) *−−−−NS>* $a\ \hat{}\ m$)
apply (*induct m*)
apply (*auto simp add*: *power-Suc intro*: *NSLIMSEQ-mult NSLIMSEQ-const*)
done

We can now try and derive a few properties of sequences, starting with the limit comparison property for sequences.

lemma *NSLIMSEQ-le*:
[| f *−−−−NS>* l; g *−−−−NS>* m;
$\exists N.\ \forall n \geq N.\ f(n) \leq g(n)$
|] ==> $l \leq$ (m::*real*)
apply (*simp add*: *NSLIMSEQ-def*, *safe*)
apply (*drule starfun-le-mono*)
apply (*drule HNatInfinite-whn* [*THEN* [*2*] *bspec*])+
apply (*drule-tac* x = *whn* **in** *spec*)
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*])+
apply *clarify*
apply (*auto intro*: *hypreal-of-real-le-add-Infininitesimal-cancel2*)
done

lemma *NSLIMSEQ-le-const*: [| X *−−−−NS>* (r::*real*); $\forall n.\ a \leq X\ n$ |] ==> $a \leq r$
by (*erule NSLIMSEQ-le* [*OF NSLIMSEQ-const*], *auto*)

lemma *NSLIMSEQ-le-const2*: [| X *−−−−NS>* (r::*real*); $\forall n.\ X\ n \leq a$ |] ==> $r \leq a$
by (*erule NSLIMSEQ-le* [*OF - NSLIMSEQ-const*], *auto*)

Shift a convergent series by 1: By the equivalence between Cauchiness and convergence and because the successor of an infinite hypernatural is also

infinite.

lemma *NSLIMSEQ-Suc*: f ----$NS>$ l ==> (%n. $f(Suc\ n)$) ----$NS>$ l
apply (*unfold NSLIMSEQ-def*, *safe*)
apply (*drule-tac x=N + 1* **in** *bspec*)
apply (*erule HNatInfinite-add*)
apply (*simp add*: *starfun-shift-one*)
done

lemma *NSLIMSEQ-imp-Suc*: (%n. $f(Suc\ n)$) ----$NS>$ l ==> f ----$NS>$ l
apply (*unfold NSLIMSEQ-def*, *safe*)
apply (*drule-tac x=N − 1* **in** *bspec*)
apply (*erule Nats-1* [*THEN* [*2*] *HNatInfinite-diff*])
apply (*simp add*: *starfun-shift-one one-le-HNatInfinite*)
done

lemma *NSLIMSEQ-Suc-iff*: ((%n. $f(Suc\ n)$) ----$NS>$ l) = (f ----$NS>$ l)
by (*blast intro*: *NSLIMSEQ-imp-Suc NSLIMSEQ-Suc*)

41.1.1 Equivalence of *LIMSEQ* and *NSLIMSEQ*

lemma *LIMSEQ-NSLIMSEQ*:
assumes *X*: X ----> L **shows** X ----$NS>$ L
proof (*rule NSLIMSEQ-I*)
fix N **assume** N: $N \in HNatInfinite$
have *starfun X N − star-of L* $\in$ *Infinitesimal*
proof (*rule InfinitesimalI2*)
fix r::*real* **assume** r: $0 < r$
from *LIMSEQ-D* [*OF X r*]
obtain *no* **where** $\forall n \geq no.\ norm\ (X\ n - L) < r$ **..**
hence $\forall n \geq$*star-of no*. *hnorm* (*starfun X n − star-of L*) < *star-of r*
by *transfer*
thus *hnorm* (*starfun X N − star-of L*) < *star-of r*
using *N* **by** (*simp add*: *star-of-le-HNatInfinite*)
qed
thus *starfun X N* $\approx$ *star-of L*
by (*unfold approx-def*)
qed

lemma *NSLIMSEQ-LIMSEQ*:
assumes *X*: X ----$NS>$ L **shows** X ----> L
proof (*rule LIMSEQ-I*)
fix r::*real* **assume** r: $0 < r$
have $\exists no.\ \forall n \geq no.$ *hnorm* (*starfun X n − star-of L*) < *star-of r*
proof (*intro exI allI impI*)
fix n **assume** *whn* $\leq n$
with *HNatInfinite-whn* **have** $n \in HNatInfinite$
by (*rule HNatInfinite-upward-closed*)
with *X* **have** *starfun X n* $\approx$ *star-of L*

by (*rule NSLIMSEQ-D*)
hence *starfun X n − star-of L ∈ Infinitesimal*
by (*unfold approx-def*)
thus *hnorm* (*starfun X n − star-of L*) < *star-of r*
using *r* **by** (*rule InfinitesimalD2*)
qed
thus ∃ *no*. ∀ *n*≥*no*. *norm* (*X n − L*) < *r*
by *transfer*
qed

theorem *LIMSEQ-NSLIMSEQ-iff*: (*f* −−−−> *L*) = (*f* −−−−*NS*> *L*)
by (*blast intro*: *LIMSEQ-NSLIMSEQ NSLIMSEQ-LIMSEQ*)

lemma *NSLIMSEQ-finite-set*:
!!(*f*::*nat*=>*nat*). ∀ *n*. *n* ≤ *f n* ==> *finite* {*n*. *f n* ≤ *u*}
by (*rule-tac B*={..*u*} **in** *finite-subset*, *auto intro*: *order-trans*)

41.1.2 Derived theorems about *NSLIMSEQ*

We prove the NS version from the standard one, since the NS proof seems more complicated than the standard one above!

lemma *NSLIMSEQ-norm-zero*: ((λ*n*. *norm* (*X n*)) −−−−*NS*> *0*) = (*X* −−−−*NS*> *0*)
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-norm-zero*)

lemma *NSLIMSEQ-rabs-zero*: ((%*n*. |*f n*|) −−−−*NS*> *0*) = (*f* −−−−*NS*> (*0*::*real*))
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-rabs-zero*)

Generalization to other limits

lemma *NSLIMSEQ-imp-rabs*: *f* −−−−*NS*> (*l*::*real*) ==> (%*n*. |*f n*|) −−−−*NS*> |*l*|
apply (*simp add*: *NSLIMSEQ-def*)
apply (*auto intro*: *approx-hrabs*
simp add: *starfun-abs*)
done

lemma *NSLIMSEQ-inverse-zero*:
∀ *y*::*real*. ∃ *N*. ∀ *n* ≥ *N*. *y* < *f*(*n*)
==> (%*n*. *inverse*(*f n*)) −−−−*NS*> *0*
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-zero*)

lemma *NSLIMSEQ-inverse-real-of-nat*: (%*n*. *inverse*(*real*(*Suc n*))) −−−−*NS*> *0*
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-real-of-nat*)

lemma *NSLIMSEQ-inverse-real-of-nat-add*:
(%*n*. *r* + *inverse*(*real*(*Suc n*))) −−−−*NS*> *r*
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-real-of-nat-add*)

lemma *NSLIMSEQ-inverse-real-of-nat-add-minus*:
(%*n*. *r* + −*inverse*(*real*(*Suc n*))) −−−−*NS*> *r*
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-real-of-nat-add-minus*)

lemma *NSLIMSEQ-inverse-real-of-nat-add-minus-mult*:
(%*n*. *r*∗(1 + −*inverse*(*real*(*Suc n*)))) −−−−*NS*> *r*
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-real-of-nat-add-minus-mult*)

41.2 Convergence

lemma *nslimI*: *X* −−−−*NS*> *L* ==> *nslim X* = *L*
apply (*simp add*: *nslim-def*)
apply (*blast intro*: *NSLIMSEQ-unique*)
done

lemma *lim-nslim-iff*: *lim X* = *nslim X*
by (*simp add*: *lim-def nslim-def LIMSEQ-NSLIMSEQ-iff*)

lemma *NSconvergentD*: *NSconvergent X* ==> ∃ *L*. (*X* −−−−*NS*> *L*)
by (*simp add*: *NSconvergent-def*)

lemma *NSconvergentI*: (*X* −−−−*NS*> *L*) ==> *NSconvergent X*
by (*auto simp add*: *NSconvergent-def*)

lemma *convergent-NSconvergent-iff*: *convergent X* = *NSconvergent X*
by (*simp add*: *convergent-def NSconvergent-def LIMSEQ-NSLIMSEQ-iff*)

lemma *NSconvergent-NSLIMSEQ-iff*: *NSconvergent X* = (*X* −−−−*NS*> *nslim X*)
by (*auto intro*: *theI NSLIMSEQ-unique simp add*: *NSconvergent-def nslim-def*)

41.3 Bounded Monotonic Sequences

lemma *NSBseqD*: [| *NSBseq X*; *N*: *HNatInfinite* |] ==> (∗*f*∗ *X*) *N* : *HFinite*
by (*simp add*: *NSBseq-def*)

lemma *Standard-subset-HFinite*: *Standard* ⊆ *HFinite*
unfolding *Standard-def* **by** *auto*

lemma *NSBseqD2*: *NSBseq X* ⟹ (∗*f*∗ *X*) *N* ∈ *HFinite*
apply (*cases N* ∈ *HNatInfinite*)
apply (*erule* (*1*) *NSBseqD*)
apply (*rule subsetD* [*OF Standard-subset-HFinite*])
apply (*simp add*: *HNatInfinite-def Nats-eq-Standard*)
done

lemma *NSBseqI*: ∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* : *HFinite* ==> *NSBseq X*
by (*simp add*: *NSBseq-def*)

The standard definition implies the nonstandard definition

lemma *Bseq-NSBseq*: *Bseq X ==> NSBseq X*
proof (*unfold NSBseq-def*, *safe*)
assume *X*: *Bseq X*
fix *N* **assume** *N*: *N* ∈ *HNatInfinite*
from *BseqD* [*OF X*] **obtain** *K* **where** ∀ *n*. *norm* (*X n*) ≤ *K* **by** *fast*
hence ∀ *N*. *hnorm* (*starfun X N*) ≤ *star-of K* **by** *transfer*
hence *hnorm* (*starfun X N*) ≤ *star-of K* **by** *simp*
also have *star-of K* < *star-of* (*K* + *1*) **by** *simp*
finally have ∃ *x*∈*Reals*. *hnorm* (*starfun X N*) < *x* **by** (*rule bexI*, *simp*)
thus *starfun X N* ∈ *HFinite* **by** (*simp add*: *HFinite-def*)
qed

The nonstandard definition implies the standard definition

lemma *SReal-less-omega*: *r* ∈ ℝ ⟹ *r* < ω
apply (*insert HInfinite-omega*)
apply (*simp add*: *HInfinite-def*)
apply (*simp add*: *order-less-imp-le*)
done

lemma *NSBseq-Bseq*: *NSBseq X* ⟹ *Bseq X*
proof (*rule ccontr*)
let *?n* = λ*K*. *LEAST n*. *K* < *norm* (*X n*)
assume *NSBseq X*
hence *finite*: (∗*f*∗ *X*) ((∗*f*∗ *?n*) ω) ∈ *HFinite*
by (*rule NSBseqD2*)
assume ¬ *Bseq X*
hence ∀ *K*>*0*. ∃ *n*. *K* < *norm* (*X n*)
by (*simp add*: *Bseq-def linorder-not-le*)
hence ∀ *K*>*0*. *K* < *norm* (*X* (*?n K*))
by (*auto intro*: *LeastI-ex*)
hence ∀ *K*>*0*. *K* < *hnorm* ((∗*f*∗ *X*) ((∗*f*∗ *?n*) *K*))
by *transfer*
hence ω < *hnorm* ((∗*f*∗ *X*) ((∗*f*∗ *?n*) ω))
by *simp*
hence ∀ *r*∈ℝ. *r* < *hnorm* ((∗*f*∗ *X*) ((∗*f*∗ *?n*) ω))
by (*simp add*: *order-less-trans* [*OF SReal-less-omega*])
hence (∗*f*∗ *X*) ((∗*f*∗ *?n*) ω) ∈ *HInfinite*
by (*simp add*: *HInfinite-def*)
with *finite* **show** *False*
by (*simp add*: *HFinite-HInfinite-iff*)
qed

Equivalence of nonstandard and standard definitions for a bounded sequence

lemma *Bseq-NSBseq-iff*: (*Bseq X*) = (*NSBseq X*)
by (*blast intro*!: *NSBseq-Bseq Bseq-NSBseq*)

A convergent sequence is bounded: Boundedness as a necessary condition for convergence. The nonstandard version has no existential, as usual

lemma *NSconvergent-NSBseq*: *NSconvergent X ==> NSBseq X*
apply (*simp add*: *NSconvergent-def NSBseq-def NSLIMSEQ-def*)
apply (*blast intro*: *HFinite-star-of approx-sym approx-HFinite*)
done

Standard Version: easily now proved using equivalence of NS and standard definitions

lemma *convergent-Bseq*: *convergent X ==> Bseq X*
by (*simp add*: *NSconvergent-NSBseq convergent-NSconvergent-iff Bseq-NSBseq-iff*)

41.3.1 Upper Bounds and Lubs of Bounded Sequences

lemma *NSBseq-isUb*: *NSBseq X ==> ∃ U::real. isUb UNIV {x. ∃ n. X n = x} U*
by (*simp add*: *Bseq-NSBseq-iff* [*symmetric*] *Bseq-isUb*)

lemma *NSBseq-isLub*: *NSBseq X ==> ∃ U::real. isLub UNIV {x. ∃ n. X n = x} U*
by (*simp add*: *Bseq-NSBseq-iff* [*symmetric*] *Bseq-isLub*)

41.3.2 A Bounded and Monotonic Sequence Converges

The best of both worlds: Easier to prove this result as a standard theorem and then use equivalence to "transfer" it into the equivalent nonstandard form if needed!

lemma *Bmonoseq-NSLIMSEQ*: *∀ n ≥ m. X n = X m ==> ∃ L. (X ----NS> L)*
by (*auto dest*!: *Bmonoseq-LIMSEQ simp add*: *LIMSEQ-NSLIMSEQ-iff*)

lemma *NSBseq-mono-NSconvergent*:
[| *NSBseq X*; *∀ m. ∀ n ≥ m. X m ≤ X n* |] ==> *NSconvergent* (*X*::*nat*=>*real*)
by (*auto intro*: *Bseq-mono-convergent*
simp add: *convergent-NSconvergent-iff* [*symmetric*]
Bseq-NSBseq-iff [*symmetric*])

41.4 Cauchy Sequences

lemma *NSCauchyI*:
(⋀*M N*. ⟦*M ∈ HNatInfinite*; *N ∈ HNatInfinite*⟧ ⟹ *starfun X M ≈ starfun X N*)
⟹ *NSCauchy X*
by (*simp add*: *NSCauchy-def*)

lemma *NSCauchyD*:
⟦*NSCauchy X*; *M ∈ HNatInfinite*; *N ∈ HNatInfinite*⟧
⟹ *starfun X M ≈ starfun X N*
by (*simp add*: *NSCauchy-def*)

41.4.1 Equivalence Between NS and Standard

lemma *Cauchy-NSCauchy*:
assumes X: *Cauchy X* **shows** *NSCauchy X*
proof (*rule NSCauchyI*)
fix M **assume** M: $M \in$ *HNatInfinite*
fix N **assume** N: $N \in$ *HNatInfinite*
have *starfun X M − starfun X N* ∈ *Infinitesimal*
proof (*rule InfinitesimalI2*)
fix r :: *real* **assume** r: $0 < r$
from *CauchyD* [*OF X r*]
obtain k **where** ∀ $m \geq k$. ∀ $n \geq k$. *norm* $(X\ m - X\ n) < r$ **..**
hence ∀ $m \geq$ *star-of k*. ∀ $n \geq$ *star-of k*.
hnorm (*starfun X m − starfun X n*) < *star-of r*
by *transfer*
thus *hnorm* (*starfun X M − starfun X N*) < *star-of r*
using *M N* **by** (*simp add*: *star-of-le-HNatInfinite*)
qed
thus *starfun X M* ≈ *starfun X N*
by (*unfold approx-def*)
qed

lemma *NSCauchy-Cauchy*:
assumes X: *NSCauchy X* **shows** *Cauchy X*
proof (*rule CauchyI*)
fix r::*real* **assume** r: $0 < r$
have ∃ k. ∀ $m \geq k$. ∀ $n \geq k$. *hnorm* (*starfun X m − starfun X n*) < *star-of r*
proof (*intro exI allI impI*)
fix M **assume** *whn* ≤ M
with *HNatInfinite-whn* **have** M: $M \in$ *HNatInfinite*
by (*rule HNatInfinite-upward-closed*)
fix N **assume** *whn* ≤ N
with *HNatInfinite-whn* **have** N: $N \in$ *HNatInfinite*
by (*rule HNatInfinite-upward-closed*)
from *X M N* **have** *starfun X M* ≈ *starfun X N*
by (*rule NSCauchyD*)
hence *starfun X M − starfun X N* ∈ *Infinitesimal*
by (*unfold approx-def*)
thus *hnorm* (*starfun X M − starfun X N*) < *star-of r*
using r **by** (*rule InfinitesimalD2*)
qed
thus ∃ k. ∀ $m \geq k$. ∀ $n \geq k$. *norm* $(X\ m - X\ n) < r$
by *transfer*
qed

theorem *NSCauchy-Cauchy-iff*: *NSCauchy X* = *Cauchy X*
by (*blast intro*!: *NSCauchy-Cauchy Cauchy-NSCauchy*)

41.4.2 Cauchy Sequences are Bounded

A Cauchy sequence is bounded – nonstandard version

lemma *NSCauchy-NSBseq*: *NSCauchy X ==> NSBseq X*
by (*simp add*: *Cauchy-Bseq Bseq-NSBseq-iff* [*symmetric*] *NSCauchy-Cauchy-iff*)

41.4.3 Cauchy Sequences are Convergent

Equivalence of Cauchy criterion and convergence: We will prove this using our NS formulation which provides a much easier proof than using the standard definition. We do not need to use properties of subsequences such as boundedness, monotonicity etc... Compare with Harrison's corresponding proof in HOL which is much longer and more complicated. Of course, we do not have problems which he encountered with guessing the right instantiations for his 'espsilon-delta' proof(s) in this case since the NS formulations do not involve existential quantifiers.

lemma *NSconvergent-NSCauchy*: *NSconvergent X ⟹ NSCauchy X*
apply (*simp add*: *NSconvergent-def NSLIMSEQ-def NSCauchy-def*, *safe*)
apply (*auto intro*: *approx-trans2*)
done

lemma *real-NSCauchy-NSconvergent*:
  fixes *X* :: *nat* ⇒ *real*
  shows *NSCauchy X ⟹ NSconvergent X*
apply (*simp add*: *NSconvergent-def NSLIMSEQ-def*)
apply (*frule NSCauchy-NSBseq*)
apply (*simp add*: *NSBseq-def NSCauchy-def*)
apply (*drule HNatInfinite-whn* [*THEN* [*2*] *bspec*])
apply (*drule HNatInfinite-whn* [*THEN* [*2*] *bspec*])
apply (*auto dest*!: *st-part-Ex simp add*: *SReal-iff*)
apply (*blast intro*: *approx-trans3*)
done

lemma *NSCauchy-NSconvergent*:
  fixes *X* :: *nat* ⇒ *'a*::*banach*
  shows *NSCauchy X ⟹ NSconvergent X*
apply (*drule NSCauchy-Cauchy* [*THEN Cauchy-convergent*])
apply (*erule convergent-NSconvergent-iff* [*THEN iffD1*])
done

lemma *NSCauchy-NSconvergent-iff*:
  fixes *X* :: *nat* ⇒ *'a*::*banach*
  shows *NSCauchy X = NSconvergent X*
by (*fast intro*: *NSCauchy-NSconvergent NSconvergent-NSCauchy*)

41.5 Power Sequences

The sequence $x \hat{} n$ tends to 0 if $(0::'a) \leq x$ and $x < (1::'a)$. Proof will use (NS) Cauchy equivalence for convergence and also fact that bounded and monotonic sequence converges.

We now use NS criterion to bring proof of theorem through

lemma *NSLIMSEQ-realpow-zero*:
[| $0 \leq (x::real)$; $x < 1$ |] ==> $(\%n.\ x \hat{} n) ----NS> 0$
apply (*simp add*: *NSLIMSEQ-def*)
apply (*auto dest*!: *convergent-realpow simp add*: *convergent-NSconvergent-iff*)
apply (*frule NSconvergentD*)
apply (*auto simp add*: *NSLIMSEQ-def NSCauchy-NSconvergent-iff* [*symmetric*] *NSCauchy-def starfun-pow*)
apply (*frule HNatInfinite-add-one*)
apply (*drule bspec*, *assumption*)
apply (*drule bspec*, *assumption*)
apply (*drule-tac x* = *N* + (*1*::*hypnat*) **in** *bspec*, *assumption*)
apply (*simp add*: *hyperpow-add*)
apply (*drule approx-mult-subst-star-of*, *assumption*)
apply (*drule approx-trans3*, *assumption*)
apply (*auto simp del*: *star-of-mult simp add*: *star-of-mult* [*symmetric*])
done

lemma *NSLIMSEQ-rabs-realpow-zero*: $|c| < (1::real) ==> (\%n.\ |c| \hat{} n) ----NS> 0$
by (*simp add*: *LIMSEQ-rabs-realpow-zero LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

lemma *NSLIMSEQ-rabs-realpow-zero2*: $|c| < (1::real) ==> (\%n.\ c \hat{} n) ----NS> 0$
by (*simp add*: *LIMSEQ-rabs-realpow-zero2 LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

end

42 HSeries: Finite Summation and Infinite Series for Hyperreals

theory *HSeries*
imports *Series HSEQ*
begin

definition
sumhr :: (*hypnat* * *hypnat* * (*nat*=>*real*)) => *hypreal* **where**
sumhr =
(%(*M*,*N*,*f*). *starfun2* (%*m n*. *setsum f* {*m*..<*n*}) *M N*)

definition
NSsums :: [*nat*=>*real*,*real*] => *bool* (**infixr** *NSsums 80*) **where**
f NSsums s = (%*n*. *setsum f* {*0*..<*n*}) −−−−*NS*> *s*

definition
NSsummable :: (*nat*=>*real*) => *bool* **where**
NSsummable f = (∃ *s*. *f NSsums s*)

definition
NSsuminf :: (*nat*=>*real*) => *real* **where**
NSsuminf f = (*THE s*. *f NSsums s*)

lemma *sumhr-app*: *sumhr*(*M*,*N*,*f*) = (∗*f2*∗ (λ*m n*. *setsum f* {*m*..<*n*})) *M N*
by (*simp add*: *sumhr-def*)

Base case in definition of *sumr*

lemma *sumhr-zero* [*simp*]: !!*m*. *sumhr* (*m*,*0*,*f*) = *0*
unfolding *sumhr-app* **by** *transfer simp*

Recursive case in definition of *sumr*

lemma *sumhr-if*:
!!*m n*. *sumhr*(*m*,*n*+*1*,*f*) =
(*if n* + *1* ≤ *m then 0 else sumhr*(*m*,*n*,*f*) + (∗*f*∗ *f*) *n*)
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-Suc-zero* [*simp*]: !!*n*. *sumhr* (*n* + *1*, *n*, *f*) = *0*
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-eq-bounds* [*simp*]: !!*n*. *sumhr* (*n*,*n*,*f*) = *0*
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-Suc* [*simp*]: !!*m*. *sumhr* (*m*,*m* + *1*,*f*) = (∗*f*∗ *f*) *m*
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-add-lbound-zero* [*simp*]: !!*k m*. *sumhr*(*m*+*k*,*k*,*f*) = *0*
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-add*:
!!*m n*. *sumhr* (*m*,*n*,*f*) + *sumhr*(*m*,*n*,*g*) = *sumhr*(*m*,*n*,%*i*. *f i* + *g i*)
unfolding *sumhr-app* **by** *transfer* (*rule setsum-addf* [*symmetric*])

lemma *sumhr-mult*:
!!*m n*. *hypreal-of-real r* ∗ *sumhr*(*m*,*n*,*f*) = *sumhr*(*m*,*n*,%*n*. *r* ∗ *f n*)
unfolding *sumhr-app* **by** *transfer* (*rule setsum-right-distrib*)

lemma *sumhr-split-add*:
!!*n p*. *n* < *p* ==> *sumhr*(*0*,*n*,*f*) + *sumhr*(*n*,*p*,*f*) = *sumhr*(*0*,*p*,*f*)
unfolding *sumhr-app* **by** *transfer* (*simp add*: *setsum-add-nat-ivl*)

lemma *sumhr-split-diff*: $n<p ==> sumhr(0,p,f) - sumhr(0,n,f) = sumhr(n,p,f)$
by (*drule-tac* $f = f$ **in** *sumhr-split-add* [*symmetric*], *simp*)

lemma *sumhr-hrabs*: $!!m\ n.\ abs(sumhr(m,n,f)) \leq sumhr(m,n,\%i.\ abs(f\ i))$
unfolding *sumhr-app* **by** *transfer* (*rule setsum-abs*)

other general version also needed

lemma *sumhr-fun-hypnat-eq*:
$(\forall r.\ m \leq r\ \&\ r < n \dashrightarrow f\ r = g\ r) \dashrightarrow$
$sumhr(hypnat\text{-}of\text{-}nat\ m,\ hypnat\text{-}of\text{-}nat\ n,\ f) =$
$sumhr(hypnat\text{-}of\text{-}nat\ m,\ hypnat\text{-}of\text{-}nat\ n,\ g)$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-const*:
$!!n.\ sumhr(0,\ n,\ \%i.\ r) = hypreal\text{-}of\text{-}hypnat\ n * hypreal\text{-}of\text{-}real\ r$
unfolding *sumhr-app* **by** *transfer* (*simp add*: *real-of-nat-def*)

lemma *sumhr-less-bounds-zero* [*simp*]: $!!m\ n.\ n < m ==> sumhr(m,n,f) = 0$
unfolding *sumhr-app* **by** *transfer simp*

lemma *sumhr-minus*: $!!m\ n.\ sumhr(m,\ n,\ \%i.\ -f\ i) = -\ sumhr(m,\ n,\ f)$
unfolding *sumhr-app* **by** *transfer* (*rule setsum-negf*)

lemma *sumhr-shift-bounds*:
$!!m\ n.\ sumhr(m+hypnat\text{-}of\text{-}nat\ k,n+hypnat\text{-}of\text{-}nat\ k,f) =$
$sumhr(m,n,\%i.\ f(i + k))$
unfolding *sumhr-app* **by** *transfer* (*rule setsum-shift-bounds-nat-ivl*)

42.1 Nonstandard Sums

Infinite sums are obtained by summing to some infinite hypernatural (such as *whn*)

lemma *sumhr-hypreal-of-hypnat-omega*:
$sumhr(0,whn,\%i.\ 1) = hypreal\text{-}of\text{-}hypnat\ whn$
by (*simp add*: *sumhr-const*)

lemma *sumhr-hypreal-omega-minus-one*: $sumhr(0,\ whn,\ \%i.\ 1) = omega - 1$
apply (*simp add*: *sumhr-const*)

apply (*unfold star-class-defs omega-def hypnat-omega-def*
of-hypnat-def star-of-def)
apply (*simp add*: *starfun-star-n starfun2-star-n real-of-nat-def*)
done

lemma *sumhr-minus-one-realpow-zero* [*simp*]:
$!!N.\ sumhr(0,\ N + N,\ \%i.\ (-1)\ \hat{}\ (i+1)) = 0$

unfolding *sumhr-app*
by *transfer* (*simp del*: *realpow-Suc add*: *nat-mult-2* [*symmetric*])

lemma *sumhr-interval-const*:
(∀ *n*. *m* ≤ *Suc n* −−> *f n* = *r*) & *m* ≤ *na*
==> *sumhr*(*hypnat-of-nat m*,*hypnat-of-nat na*,*f*) =
(*hypreal-of-nat* (*na* − *m*) ∗ *hypreal-of-real r*)
unfolding *sumhr-app* **by** *transfer simp*

lemma *starfunNat-sumr*: !!*N*. (∗*f*∗ (%*n*. *setsum f* {*0*..<*n*})) *N* = *sumhr*(*0*,*N*,*f*)
unfolding *sumhr-app* **by** *transfer* (*rule refl*)

lemma *sumhr-hrabs-approx* [*simp*]: *sumhr*(*0*, *M*, *f*) @= *sumhr*(*0*, *N*, *f*)
==> *abs* (*sumhr*(*M*, *N*, *f*)) @= *0*
apply (*cut-tac x* = *M* **and** *y* = *N* **in** *linorder-less-linear*)
apply (*auto simp add*: *approx-refl*)
apply (*drule approx-sym* [*THEN approx-minus-iff* [*THEN iffD1*]])
apply (*auto dest*: *approx-hrabs*
simp add: *sumhr-split-diff diff-minus* [*symmetric*])
done

lemma *sums-NSsums-iff*: (*f sums l*) = (*f NSsums l*)
by (*simp add*: *sums-def NSsums-def LIMSEQ-NSLIMSEQ-iff*)

lemma *summable-NSsummable-iff*: (*summable f*) = (*NSsummable f*)
by (*simp add*: *summable-def NSsummable-def sums-NSsums-iff*)

lemma *suminf-NSsuminf-iff*: (*suminf f*) = (*NSsuminf f*)
by (*simp add*: *suminf-def NSsuminf-def sums-NSsums-iff*)

lemma *NSsums-NSsummable*: *f NSsums l* ==> *NSsummable f*
by (*simp add*: *NSsums-def NSsummable-def*, *blast*)

lemma *NSsummable-NSsums*: *NSsummable f* ==> *f NSsums* (*NSsuminf f*)
apply (*simp add*: *NSsummable-def NSsuminf-def NSsums-def*)
apply (*blast intro*: *theI NSLIMSEQ-unique*)
done

lemma *NSsums-unique*: *f NSsums s* ==> (*s* = *NSsuminf f*)
by (*simp add*: *suminf-NSsuminf-iff* [*symmetric*] *sums-NSsums-iff sums-unique*)

lemma *NSseries-zero*:
∀ *m*. *n* ≤ *Suc m* −−> *f*(*m*) = *0* ==> *f NSsums* (*setsum f* {*0*..<*n*})
by (*simp add*: *sums-NSsums-iff* [*symmetric*] *series-zero*)

lemma *NSsummable-NSCauchy*:
NSsummable f =
(∀ *M* ∈ *HNatInfinite*. ∀ *N* ∈ *HNatInfinite*. *abs* (*sumhr*(*M*,*N*,*f*)) @= *0*)

apply (*auto simp add*: *summable-NSsummable-iff* [*symmetric*]
summable-convergent-sumr-iff convergent-NSconvergent-iff
NSCauchy-NSconvergent-iff [*symmetric*] *NSCauchy-def starfunNat-sumr*)
apply (*cut-tac* $x = M$ **and** $y = N$ **in** *linorder-less-linear*)
apply (*auto simp add*: *approx-refl*)
apply (*rule approx-minus-iff* [*THEN iffD2*, *THEN approx-sym*])
apply (*rule-tac* [*2*] *approx-minus-iff* [*THEN iffD2*])
apply (*auto dest*: *approx-hrabs-zero-cancel*
simp add: *sumhr-split-diff diff-minus* [*symmetric*])
done

Terms of a convergent series tend to zero

lemma *NSsummable-NSLIMSEQ-zero*: *NSsummable f* ==> *f* −−−−*NS*> *0*
apply (*auto simp add*: *NSLIMSEQ-def NSsummable-NSCauchy*)
apply (*drule bspec*, *auto*)
apply (*drule-tac* $x = N + 1$ **in** *bspec*)
apply (*auto intro*: *HNatInfinite-add-one approx-hrabs-zero-cancel*)
done

Nonstandard comparison test

lemma *NSsummable-comparison-test*:
[| ∃ *N*. ∀ *n*. *N* ≤ *n* −−> *abs*(*f n*) ≤ *g n*; *NSsummable g* |] ==> *NSsummable f*
apply (*fold summable-NSsummable-iff*)
apply (*rule summable-comparison-test*, *simp*, *assumption*)
done

lemma *NSsummable-rabs-comparison-test*:
[| ∃ *N*. ∀ *n*. *N* ≤ *n* −−> *abs*(*f n*) ≤ *g n*; *NSsummable g* |]
==> *NSsummable* (%*k*. *abs* (*f k*))
apply (*rule NSsummable-comparison-test*)
apply (*auto*)
done

end

43 HLim: Limits and Continuity (Nonstandard)

theory *HLim*
imports *Star Lim*
begin

Nonstandard Definitions

definition
NSLIM :: [*'a*::*real-normed-vector* => *'b*::*real-normed-vector*, *'a*, *'b*] => *bool*
(((-)/ −− (-)/ −−*NS*> (-)) [*60*, *0*, *60*] *60*) **where**
f −− *a* −−*NS*> *L* =

$(\forall x.\ (x \neq \mathit{star\text{-}of}\ a\ \&\ x\ @{=}\ \mathit{star\text{-}of}\ a\ {-}{-}{>}\ (\ {*}f{*}\ f)\ x\ @{=}\ \mathit{star\text{-}of}\ L))$

definition
isNSCont :: [*'a*::*real-normed-vector* => *'b*::*real-normed-vector*, *'a*] => *bool* **where**
— NS definition dispenses with limit notions
isNSCont f a = $(\forall y.\ y\ @{=}\ \mathit{star\text{-}of}\ a\ {-}{-}{>}$
$(\ {*}f{*}\ f)\ y\ @{=}\ \mathit{star\text{-}of}\ (f\ a))$

definition
isNSUCont :: [*'a*::*real-normed-vector* => *'b*::*real-normed-vector*] => *bool* **where**
isNSUCont f = $(\forall x\ y.\ x\ @{=}\ y\ {-}{-}{>}\ (\ {*}f{*}\ f)\ x\ @{=}\ (\ {*}f{*}\ f)\ y)$

43.1 Limits of Functions

lemma *NSLIM-I*:
$(\bigwedge x.\ [\![x \neq \mathit{star\text{-}of}\ a;\ x \approx \mathit{star\text{-}of}\ a]\!] \Longrightarrow \mathit{starfun}\ f\ x \approx \mathit{star\text{-}of}\ L)$
$\Longrightarrow f\ {-}{-}\ a\ {-}{-}\mathit{NS}{>}\ L$
by (*simp add*: *NSLIM-def*)

lemma *NSLIM-D*:
$[\![f\ {-}{-}\ a\ {-}{-}\mathit{NS}{>}\ L;\ x \neq \mathit{star\text{-}of}\ a;\ x \approx \mathit{star\text{-}of}\ a]\!]$
$\Longrightarrow \mathit{starfun}\ f\ x \approx \mathit{star\text{-}of}\ L$
by (*simp add*: *NSLIM-def*)

Proving properties of limits using nonstandard definition. The properties hold for standard limits as well!

lemma *NSLIM-mult*:
fixes *l m* :: *'a*::*real-normed-algebra*
shows $[|\ f\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ l;\ g\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ m\ |]$
$\Longrightarrow (\%x.\ f(x) * g(x))\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ (l * m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-mult-HFinite*)

lemma *starfun-scaleR* [*simp*]:
starfun $(\lambda x.\ f\ x\ {*}_R\ g\ x) = (\lambda x.\ \mathit{scaleHR}\ (\mathit{starfun}\ f\ x)\ (\mathit{starfun}\ g\ x))$
by *transfer* (*rule refl*)

lemma *NSLIM-scaleR*:
$[|\ f\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ l;\ g\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ m\ |]$
$\Longrightarrow (\%x.\ f(x)\ {*}_R\ g(x))\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ (l\ {*}_R\ m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-scaleR-HFinite*)

lemma *NSLIM-add*:
$[|\ f\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ l;\ g\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ m\ |]$
$\Longrightarrow (\%x.\ f(x) + g(x))\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ (l + m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-add*)

lemma *NSLIM-const* [*simp*]: $(\%x.\ k)\ {-}{-}\ x\ {-}{-}\mathit{NS}{>}\ k$
by (*simp add*: *NSLIM-def*)

lemma *NSLIM-minus*: f $--$ a $--NS>$ L $==>$ $(\%x.\ -f(x))$ $--$ a $--NS>$ $-L$
by (*simp add*: *NSLIM-def*)

lemma *NSLIM-diff*:
$[\![f$ $--$ x $--NS>$ l; g $--$ x $--NS>$ $m]\!] \Longrightarrow (\lambda x.\ f\ x - g\ x)$ $--$ x $--NS>$ $(l - m)$
by (*simp only*: *diff-def NSLIM-add NSLIM-minus*)

lemma *NSLIM-add-minus*: $[\!|$ f $--$ x $--NS>$ l; g $--$ x $--NS>$ m $|\!]$ $==>$ $(\%x.\ f(x) + -g(x))$ $--$ x $--NS>$ $(l + -m)$
by (*simp only*: *NSLIM-add NSLIM-minus*)

lemma *NSLIM-inverse*:
fixes L :: $'a$::*real-normed-div-algebra*
shows $[\!|$ f $--$ a $--NS>$ L; $L \neq 0$ $|\!]$
$==>$ $(\%x.\ inverse(f(x)))$ $--$ a $--NS>$ $(inverse\ L)$
apply (*simp add*: *NSLIM-def*, *clarify*)
apply (*drule spec*)
apply (*auto simp add*: *star-of-approx-inverse*)
done

lemma *NSLIM-zero*:
assumes *f*: f $--$ a $--NS>$ l **shows** $(\%x.\ f(x) - l)$ $--$ a $--NS>$ 0
proof $-$
have $(\lambda x.\ f\ x - l)$ $--$ a $--NS>$ $l - l$
by (*rule NSLIM-diff* [*OF f NSLIM-const*])
thus *?thesis* **by** *simp*
qed

lemma *NSLIM-zero-cancel*: $(\%x.\ f(x) - l)$ $--$ x $--NS>$ 0 $==>$ f $--$ x $--NS>$ l
apply (*drule-tac* $g = \%x.\ l$ **and** $m = l$ **in** *NSLIM-add*)
apply (*auto simp add*: *diff-minus add-assoc*)
done

lemma *NSLIM-const-not-eq*:
fixes a :: $'a$::*real-normed-algebra-1*
shows $k \neq L \Longrightarrow \neg\ (\lambda x.\ k)$ $--$ a $--NS>$ L
apply (*simp add*: *NSLIM-def*)
apply (*rule-tac x=star-of a + of-hypreal epsilon* **in** *exI*)
apply (*simp add*: *hypreal-epsilon-not-zero approx-def*)
done

lemma *NSLIM-not-zero*:
fixes a :: $'a$::*real-normed-algebra-1*
shows $k \neq 0 \Longrightarrow \neg\ (\lambda x.\ k)$ $--$ a $--NS>$ 0
by (*rule NSLIM-const-not-eq*)

lemma *NSLIM-const-eq*:

fixes $a :: 'a$::*real-normed-algebra-1*
shows $(\lambda x.\ k) \dashrightarrow a \dashrightarrow_{NS} L \Longrightarrow k = L$
apply (*rule ccontr*)
apply (*blast dest*: *NSLIM-const-not-eq*)
done

lemma *NSLIM-unique*:
fixes $a :: 'a$::*real-normed-algebra-1*
shows $[\![f \dashrightarrow a \dashrightarrow_{NS} L; f \dashrightarrow a \dashrightarrow_{NS} M]\!] \Longrightarrow L = M$
apply (*drule* (*1*) *NSLIM-diff*)
apply (*auto dest*!: *NSLIM-const-eq*)
done

lemma *NSLIM-mult-zero*:
fixes $f\ g :: 'a$::*real-normed-vector* $\Rightarrow 'b$::*real-normed-algebra*
shows $[|\ f \dashrightarrow x \dashrightarrow_{NS} 0;\ g \dashrightarrow x \dashrightarrow_{NS} 0\ |] ==> (\%x.\ f(x)*g(x)) \dashrightarrow x \dashrightarrow_{NS} 0$
by (*drule NSLIM-mult*, *auto*)

lemma *NSLIM-self*: $(\%x.\ x) \dashrightarrow a \dashrightarrow_{NS} a$
by (*simp add*: *NSLIM-def*)

43.1.1 Equivalence of *LIM* and *NSLIM*

lemma *LIM-NSLIM*:
assumes *f*: $f \dashrightarrow a \longrightarrow L$ **shows** $f \dashrightarrow a \dashrightarrow_{NS} L$
proof (*rule NSLIM-I*)
fix x
assume *neq*: $x \neq$ *star-of a*
assume *approx*: $x \approx$ *star-of a*
have *starfun f x − star-of L* $\in$ *Infinitesimal*
proof (*rule InfinitesimalI2*)
fix r::*real* **assume** *r*: $0 < r$
from *LIM-D* [*OF f r*]
obtain s **where** *s*: $0 < s$ **and**
less-r: $\bigwedge x.\ [\![x \neq a;\ norm\ (x - a) < s]\!] \Longrightarrow norm\ (f\ x - L) < r$
by *fast*
from *less-r* **have** *less-r'*:
$\bigwedge x.\ [\![x \neq$ *star-of a*; *hnorm* $(x -$ *star-of a*$) <$ *star-of s*$]\!]$
$\Longrightarrow$ *hnorm* $($*starfun f x −* *star-of L*$) <$ *star-of r*
by *transfer*
from *approx* **have** $x -$ *star-of a* $\in$ *Infinitesimal*
by (*unfold approx-def*)
hence *hnorm* $(x -$ *star-of a*$) <$ *star-of s*
using s **by** (*rule InfinitesimalD2*)
with *neq* **show** *hnorm* $($*starfun f x −* *star-of L*$) <$ *star-of r*
by (*rule less-r'*)
qed
thus *starfun f x* $\approx$ *star-of L*

by (*unfold approx-def*)
qed

lemma *NSLIM-LIM*:
assumes *f*: f $--$ a $--NS>$ L **shows** f $--$ a $-->$ L
proof (*rule LIM-I*)
fix *r*::*real* **assume** *r*: $0 < r$
have $\exists s>0.\ \forall x.\ x \neq$ *star-of* $a \wedge$ *hnorm* $(x -$ *star-of* $a) < s$
$\longrightarrow$ *hnorm* (*starfun* $f\ x -$ *star-of* $L) <$ *star-of* r
proof (*rule exI*, *safe*)
show $0 <$ *epsilon* **by** (*rule hypreal-epsilon-gt-zero*)
next
fix x **assume** *neq*: $x \neq$ *star-of* a
assume *hnorm* $(x -$ *star-of* $a) <$ *epsilon*
with *Infinitesimal-epsilon*
have $x -$ *star-of* $a \in$ *Infinitesimal*
by (*rule hnorm-less-Infinitesimal*)
hence $x \approx$ *star-of* a
by (*unfold approx-def*)
with f *neq* **have** *starfun* $f\ x \approx$ *star-of* L
by (*rule NSLIM-D*)
hence *starfun* $f\ x -$ *star-of* $L \in$ *Infinitesimal*
by (*unfold approx-def*)
thus *hnorm* (*starfun* $f\ x -$ *star-of* $L) <$ *star-of* r
using r **by** (*rule InfinitesimalD2*)
qed
thus $\exists s>0.\ \forall x.\ x \neq a \wedge$ *norm* $(x - a) < s \longrightarrow$ *norm* $(f\ x - L) < r$
by *transfer*
qed

theorem *LIM-NSLIM-iff*: $(f$ $--$ x $-->$ $L) = (f$ $--$ x $--NS>$ $L)$
by (*blast intro*: *LIM-NSLIM NSLIM-LIM*)

43.2 Continuity

lemma *isNSContD*:
$[\![$*isNSCont* $f\ a$; $y \approx$ *star-of* $a]\!] \Longrightarrow (\ *f*\ f)\ y \approx$ *star-of* $(f\ a)$
by (*simp add*: *isNSCont-def*)

lemma *isNSCont-NSLIM*: *isNSCont* $f\ a$ ==> f $--$ a $--NS>$ $(f\ a)$
by (*simp add*: *isNSCont-def NSLIM-def*)

lemma *NSLIM-isNSCont*: f $--$ a $--NS>$ $(f\ a)$ ==> *isNSCont* $f\ a$
apply (*simp add*: *isNSCont-def NSLIM-def*, *auto*)
apply (*case-tac* $y =$ *star-of* a, *auto*)
done

NS continuity can be defined using NS Limit in similar fashion to standard def of continuity

lemma *isNSCont-NSLIM-iff*: $(isNSCont\ f\ a) = (f\ --\ a\ --NS>\ (f\ a))$
by (*blast intro*: *isNSCont-NSLIM NSLIM-isNSCont*)

Hence, NS continuity can be given in terms of standard limit

lemma *isNSCont-LIM-iff*: $(isNSCont\ f\ a) = (f\ --\ a\ -->\ (f\ a))$
by (*simp add*: *LIM-NSLIM-iff isNSCont-NSLIM-iff*)

Moreover, it's trivial now that NS continuity is equivalent to standard continuity

lemma *isNSCont-isCont-iff*: $(isNSCont\ f\ a) = (isCont\ f\ a)$
apply (*simp add*: *isCont-def*)
apply (*rule isNSCont-LIM-iff*)
done

Standard continuity ==¿ NS continuity

lemma *isCont-isNSCont*: $isCont\ f\ a \Longrightarrow isNSCont\ f\ a$
by (*erule isNSCont-isCont-iff* [*THEN iffD2*])

NS continuity ==¿ Standard continuity

lemma *isNSCont-isCont*: $isNSCont\ f\ a \Longrightarrow isCont\ f\ a$
by (*erule isNSCont-isCont-iff* [*THEN iffD1*])

Alternative definition of continuity

lemma *NSLIM-h-iff*: $(f\ --\ a\ --NS>\ L) = ((\%h.\ f(a + h))\ --\ 0\ --NS>\ L)$
apply (*simp add*: *NSLIM-def*, *auto*)
apply (*drule-tac x* = *star-of a* + *x* **in** *spec*)
apply (*drule-tac* [*2*] *x* = − *star-of a* + *x* **in** *spec*, *safe*, *simp*)
apply (*erule mem-infmal-iff* [*THEN iffD2*, *THEN Infinitesimal-add-approx-self* [*THEN approx-sym*]])
apply (*erule-tac* [*3*] *approx-minus-iff2* [*THEN iffD1*])
prefer *2* **apply** (*simp add*: *add-commute diff-def* [*symmetric*])
apply (*rule-tac x* = *x* **in** *star-cases*)
apply (*rule-tac* [*2*] *x* = *x* **in** *star-cases*)
apply (*auto simp add*: *starfun star-of-def star-n-minus star-n-add add-assoc approx-refl star-n-zero-num*)
done

lemma *NSLIM-isCont-iff*: $(f\ --\ a\ --NS>\ f\ a) = ((\%h.\ f(a + h))\ --\ 0\ --NS>\ f\ a)$
by (*rule NSLIM-h-iff*)

lemma *isNSCont-minus*: $isNSCont\ f\ a \Longrightarrow isNSCont\ (\%x.\ -\ f\ x)\ a$
by (*simp add*: *isNSCont-def*)

lemma *isNSCont-inverse*:
fixes $f :: 'a::real\text{-}normed\text{-}vector \Rightarrow 'b::real\text{-}normed\text{-}div\text{-}algebra$
shows $[|\ isNSCont\ f\ x;\ f\ x \neq 0\ |] \Longrightarrow isNSCont\ (\%x.\ inverse\ (f\ x))\ x$
by (*auto intro*: *isCont-inverse simp add*: *isNSCont-isCont-iff*)

lemma *isNSCont-const* [*simp*]: *isNSCont* (%*x*. *k*) *a*
by (*simp add*: *isNSCont-def*)

lemma *isNSCont-abs* [*simp*]: *isNSCont abs* (*a*::*real*)
apply (*simp add*: *isNSCont-def*)
apply (*auto intro*: *approx-hrabs simp add*: *starfun-rabs-hrabs*)
done

43.3 Uniform Continuity

lemma *isNSUContD*: [| *isNSUCont f*; *x* ≈ *y*|] ==> (∗*f*∗ *f*) *x* ≈ (∗*f*∗ *f*) *y*
by (*simp add*: *isNSUCont-def*)

lemma *isUCont-isNSUCont*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*
assumes *f*: *isUCont f* **shows** *isNSUCont f*
proof (*unfold isNSUCont-def*, *safe*)
fix *x y* :: *'a star*
assume *approx*: *x* ≈ *y*
have *starfun f x* − *starfun f y* ∈ *Infinitesimal*
proof (*rule InfinitesimalI2*)
fix *r*::*real* **assume** *r*: *0* < *r*
with *f* **obtain** *s* **where** *s*: *0* < *s* **and**
less-r: ⋀*x y*. *norm* (*x* − *y*) < *s* ⟹ *norm* (*f x* − *f y*) < *r*
by (*auto simp add*: *isUCont-def*)
from *less-r* **have** *less-r'*:
⋀*x y*. *hnorm* (*x* − *y*) < *star-of s*
⟹ *hnorm* (*starfun f x* − *starfun f y*) < *star-of r*
by *transfer*
from *approx* **have** *x* − *y* ∈ *Infinitesimal*
by (*unfold approx-def*)
hence *hnorm* (*x* − *y*) < *star-of s*
using *s* **by** (*rule InfinitesimalD2*)
thus *hnorm* (*starfun f x* − *starfun f y*) < *star-of r*
by (*rule less-r'*)
qed
thus *starfun f x* ≈ *starfun f y*
by (*unfold approx-def*)
qed

lemma *isNSUCont-isUCont*:
fixes *f* :: *'a*::*real-normed-vector* ⇒ *'b*::*real-normed-vector*
assumes *f*: *isNSUCont f* **shows** *isUCont f*
proof (*unfold isUCont-def*, *safe*)
fix *r*::*real* **assume** *r*: *0* < *r*
have ∃*s*>*0*. ∀*x y*. *hnorm* (*x* − *y*) < *s*
⟶ *hnorm* (*starfun f x* − *starfun f y*) < *star-of r*
proof (*rule exI*, *safe*)

```
    show 0 < epsilon by (rule hypreal-epsilon-gt-zero)
  next
    fix x y :: 'a star
    assume hnorm (x - y) < epsilon
    with Infinitesimal-epsilon
    have x - y ∈ Infinitesimal
      by (rule hnorm-less-Infinitesimal)
    hence x ≈ y
      by (unfold approx-def)
    with f have starfun f x ≈ starfun f y
      by (simp add: isNSUCont-def)
    hence starfun f x - starfun f y ∈ Infinitesimal
      by (unfold approx-def)
    thus hnorm (starfun f x - starfun f y) < star-of r
      using r by (rule InfinitesimalD2)
  qed
  thus ∃ s>0. ∀ x y. norm (x - y) < s ⟶ norm (f x - f y) < r
    by transfer
qed

end
```

44 HDeriv: Differentiation (Nonstandard)

```
theory HDeriv
imports Deriv HLim
begin
```

Nonstandard Definitions

```
definition
  nsderiv :: ['a::real-normed-field ⇒ 'a, 'a, 'a] ⇒ bool
        ((NSDERIV (-)/ (-)/ :> (-)) [1000, 1000, 60] 60) where
  NSDERIV f x :> D = (∀ h ∈ Infinitesimal - {0}.
     (( *f* f)(star-of x + h)
      - star-of (f x))/h @= star-of D)

definition
  NSdifferentiable :: ['a::real-normed-field ⇒ 'a, 'a] ⇒ bool
    (infixl NSdifferentiable 60) where
  f NSdifferentiable x = (∃ D. NSDERIV f x :> D)

definition
  increment :: [real=>real,real,hypreal] => hypreal where
  increment f x h = (@inc. f NSdifferentiable x &
          inc = ( *f* f)(hypreal-of-real x + h) - hypreal-of-real (f x))
```

44.1 Derivatives

lemma *DERIV-NS-iff*:
(*DERIV f x :> D*) = ((%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−*NS*> *D*)
by (*simp add*: *deriv-def LIM-NSLIM-iff*)

lemma *NS-DERIV-D*: *DERIV f x :> D* ==> (%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−*NS*> *D*
by (*simp add*: *deriv-def LIM-NSLIM-iff*)

lemma *hnorm-of-hypreal*:
⋀*r*. *hnorm* ((∗*f*∗ *of-real*) *r*::'*a*::*real-normed-div-algebra star*) = |*r*|
by *transfer* (*rule norm-of-real*)

lemma *Infinitesimal-of-hypreal*:
x ∈ *Infinitesimal* ⟹
((∗*f*∗ *of-real*) *x*::'*a*::*real-normed-div-algebra star*) ∈ *Infinitesimal*
apply (*rule InfinitesimalI2*)
apply (*drule* (*1*) *InfinitesimalD2*)
apply (*simp add*: *hnorm-of-hypreal*)
done

lemma *of-hypreal-eq-0-iff*:
⋀*x*. ((∗*f*∗ *of-real*) *x* = (*0*::'*a*::*real-algebra-1 star*)) = (*x* = *0*)
by *transfer* (*rule of-real-eq-0-iff*)

lemma *NSDeriv-unique*:
[| *NSDERIV f x :> D*; *NSDERIV f x :> E* |] ==> *D* = *E*
apply (*subgoal-tac* (∗*f*∗ *of-real*) *epsilon* ∈ *Infinitesimal* − {*0*::'*a star*})
apply (*simp only*: *nsderiv-def*)
apply (*drule* (*1*) *bspec*)+
apply (*drule* (*1*) *approx-trans3*)
apply *simp*
apply (*simp add*: *Infinitesimal-of-hypreal Infinitesimal-epsilon*)
apply (*simp add*: *of-hypreal-eq-0-iff hypreal-epsilon-not-zero*)
done

First NSDERIV in terms of NSLIM

first equivalence

lemma *NSDERIV-NSLIM-iff*:
(*NSDERIV f x :> D*) = ((%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−*NS*> *D*)
apply (*simp add*: *nsderiv-def NSLIM-def*, *auto*)
apply (*drule-tac x* = *xa* **in** *bspec*)
apply (*rule-tac* [*3*] *ccontr*)
apply (*drule-tac* [*3*] *x* = *h* **in** *spec*)
apply (*auto simp add*: *mem-infmal-iff starfun-lambda-cancel*)
done

second equivalence

lemma *NSDERIV-NSLIM-iff2*:
(*NSDERIV f x :> D*) = ((%*z*. (*f*(*z*) − *f*(*x*)) / (*z*−*x*)) −− *x* −−*NS*> *D*)
by (*simp add*: *NSDERIV-NSLIM-iff DERIV-LIM-iff diff-minus* [*symmetric*]
LIM-NSLIM-iff [*symmetric*])

lemma *NSDERIV-iff2*:
(*NSDERIV f x :> D*) =
(∀ *w*.
w ≠ *star-of x* & *w* ≈ *star-of x* −−>
(∗*f*∗ (%*z*. (*f z* − *f x*) / (*z*−*x*))) *w* ≈ *star-of D*)
by (*simp add*: *NSDERIV-NSLIM-iff2 NSLIM-def*)

lemma *hypreal-not-eq-minus-iff*:
(*x* ≠ *a*) = (*x* − *a* ≠ (*0*::'*a*::*ab-group-add*))
by *auto*

lemma *NSDERIVD5*:
(*NSDERIV f x :> D*) ==>
(∀ *u*. *u* ≈ *hypreal-of-real x* −−>
(∗*f*∗ (%*z*. *f z* − *f x*)) *u* ≈ *hypreal-of-real D* ∗ (*u* − *hypreal-of-real x*))
apply (*auto simp add*: *NSDERIV-iff2*)
apply (*case-tac u* = *hypreal-of-real x*, *auto*)
apply (*drule-tac x* = *u* **in** *spec*, *auto*)
apply (*drule-tac c* = *u* − *hypreal-of-real x* **and** *b* = *hypreal-of-real D* **in** *approx-mult1*)
apply (*drule-tac* [!] *hypreal-not-eq-minus-iff* [*THEN iffD1*])
apply (*subgoal-tac* [*2*] (∗*f*∗ (%*z*. *z*−*x*)) *u* ≠ (*0*::*hypreal*))
apply (*auto simp add*:
approx-minus-iff [*THEN iffD1*, *THEN mem-infmal-iff* [*THEN iffD2*]]
Infinitesimal-subset-HFinite [*THEN subsetD*])
done

lemma *NSDERIVD4*:
(*NSDERIV f x :> D*) ==>
(∀ *h* ∈ *Infinitesimal*.
((∗*f*∗ *f*)(*hypreal-of-real x* + *h*) −
hypreal-of-real (*f x*))≈ (*hypreal-of-real D*) ∗ *h*)
apply (*auto simp add*: *nsderiv-def*)
apply (*case-tac h* = (*0*::*hypreal*))
apply (*auto simp add*: *diff-minus*)
apply (*drule-tac x* = *h* **in** *bspec*)
apply (*drule-tac* [*2*] *c* = *h* **in** *approx-mult1*)
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
simp add: *diff-minus*)
done

lemma *NSDERIVD3*:

```
    (NSDERIV f x :> D) ==>
     (∀ h ∈ Infinitesimal − {0}.
            ((  *f* f)(hypreal-of-real x + h) −
              hypreal-of-real (f x))≈ (hypreal-of-real D) * h)
apply (auto simp add: nsderiv-def)
apply (rule ccontr, drule-tac x = h in bspec)
apply (drule-tac [2] c = h in approx-mult1)
apply (auto intro: Infinitesimal-subset-HFinite [THEN subsetD]
          simp add: mult-assoc diff-minus)
done
```

Differentiability implies continuity nice and simple "algebraic" proof

```
lemma NSDERIV-isNSCont: NSDERIV f x :> D ==> isNSCont f x
apply (auto simp add: nsderiv-def isNSCont-NSLIM-iff NSLIM-def)
apply (drule approx-minus-iff [THEN iffD1])
apply (drule hypreal-not-eq-minus-iff [THEN iffD1])
apply (drule-tac x = xa − star-of x in bspec)
 prefer 2 apply (simp add: add-assoc [symmetric])
apply (auto simp add: mem-infmal-iff [symmetric] add-commute)
apply (drule-tac c = xa − star-of x in approx-mult1)
apply (auto intro: Infinitesimal-subset-HFinite [THEN subsetD]
           simp add: mult-assoc nonzero-mult-divide-cancel-right)
apply (drule-tac x3=D in
          HFinite-star-of [THEN [2] Infinitesimal-HFinite-mult,
            THEN mem-infmal-iff [THEN iffD1]])
apply (auto simp add: mult-commute
           intro: approx-trans approx-minus-iff [THEN iffD2])
done
```

Differentiation rules for combinations of functions follow from clear, straightforard, algebraic manipulations

Constant function

```
lemma NSDERIV-const [simp]: (NSDERIV (%x. k) x :> 0)
by (simp add: NSDERIV-NSLIM-iff)
```

Sum of functions- proved easily

```
lemma NSDERIV-add: [| NSDERIV f x :> Da;  NSDERIV g x :> Db |]
     ==> NSDERIV (%x. f x + g x) x :> Da + Db
apply (auto simp add: NSDERIV-NSLIM-iff NSLIM-def)
apply (auto simp add: add-divide-distrib diff-divide-distrib dest!: spec)
apply (drule-tac b = star-of Da and d = star-of Db in approx-add)
apply (auto simp add: diff-def add-ac)
done
```

Product of functions - Proof is trivial but tedious and long due to rearrangement of terms

```
lemma lemma-nsderiv1:
```

fixes *a b c d* :: *'a*::*comm-ring star*
shows $(a*b) - (c*d) = (b*(a - c)) + (c*(b - d))$
by (*simp add*: *right-diff-distrib mult-ac*)

lemma *lemma-nsderiv2*:
fixes *x y z* :: *'a*::*real-normed-field star*
shows [| $(x - y) / z = star\text{-}of\ D + yb$; $z \neq 0$;
$z \in Infinitesimal$; $yb \in Infinitesimal$ |]
$==> x - y \approx 0$
apply (*simp add*: *nonzero-divide-eq-eq*)
apply (*auto intro*!: *Infinitesimal-HFinite-mult2 HFinite-add*
simp add: *mult-assoc mem-infmal-iff* [*symmetric*])
apply (*erule Infinitesimal-subset-HFinite* [*THEN subsetD*])
done

lemma *NSDERIV-mult*: [| *NSDERIV f x :> Da*; *NSDERIV g x :> Db* |]
$==>$ *NSDERIV* $(\%x.\ f\ x * g\ x)\ x :> (Da * g(x)) + (Db * f(x))$
apply (*auto simp add*: *NSDERIV-NSLIM-iff NSLIM-def*)
apply (*auto dest*!: *spec*
simp add: *starfun-lambda-cancel lemma-nsderiv1*)
apply (*simp* (*no-asm*) *add*: *add-divide-distrib diff-divide-distrib*)
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*])+
apply (*auto simp add*: *times-divide-eq-right* [*symmetric*]
simp del: *times-divide-eq*)
apply (*drule-tac D = Db* **in** *lemma-nsderiv2*, *assumption*+)
apply (*drule-tac*
approx-minus-iff [*THEN iffD2*, *THEN bex-Infinitesimal-iff2* [*THEN iffD2*]])
apply (*auto intro*!: *approx-add-mono1*
simp add: *left-distrib right-distrib mult-commute add-assoc*)
apply (*rule-tac b1 = star-of Db * star-of* $(f\ x)$
in *add-commute* [*THEN subst*])
apply (*auto intro*!: *Infinitesimal-add-approx-self2* [*THEN approx-sym*]
Infinitesimal-add Infinitesimal-mult
Infinitesimal-star-of-mult
Infinitesimal-star-of-mult2
simp add: *add-assoc* [*symmetric*])
done

Multiplying by a constant

lemma *NSDERIV-cmult*: *NSDERIV f x :> D*
$==>$ *NSDERIV* $(\%x.\ c * f\ x)\ x :> c*D$
apply (*simp only*: *times-divide-eq-right* [*symmetric*] *NSDERIV-NSLIM-iff*
minus-mult-right right-diff-distrib [*symmetric*])
apply (*erule NSLIM-const* [*THEN NSLIM-mult*])
done

Negation of function

lemma *NSDERIV-minus*: *NSDERIV f x :> D ==> NSDERIV* $(\%x.\ -(f\ x))\ x$ $:> -D$

proof (*simp add*: *NSDERIV-NSLIM-iff*)
  assume (λ*h*. (*f* (*x* + *h*) − *f x*) / *h*) −− *0* −−*NS*> *D*
  hence *deriv*: (λ*h*. − ((*f*(*x*+*h*) − *f x*) / *h*)) −− *0* −−*NS*> − *D*
    by (*rule NSLIM-minus*)
  have ∀ *h*. − ((*f* (*x* + *h*) − *f x*) / *h*) = (− *f* (*x* + *h*) + *f x*) / *h*
    by (*simp add*: *minus-divide-left diff-def*)
  with *deriv*
  show (λ*h*. (− *f* (*x* + *h*) + *f x*) / *h*) −− *0* −−*NS*> − *D* **by** *simp*
qed

Subtraction

lemma *NSDERIV-add-minus*: [| *NSDERIV f x* :> *Da*; *NSDERIV g x* :> *Db* |]
==> *NSDERIV* (%*x*. *f x* + −*g x*) *x* :> *Da* + −*Db*
by (*blast dest*: *NSDERIV-add NSDERIV-minus*)

lemma *NSDERIV-diff*:
    [| *NSDERIV f x* :> *Da*; *NSDERIV g x* :> *Db* |]
    ==> *NSDERIV* (%*x*. *f x* − *g x*) *x* :> *Da*−*Db*
apply (*simp add*: *diff-minus*)
apply (*blast intro*: *NSDERIV-add-minus*)
done

Similarly to the above, the chain rule admits an entirely straightforward derivation. Compare this with Harrison's HOL proof of the chain rule, which proved to be trickier and required an alternative characterisation of differentiability- the so-called Carathedory derivative. Our main problem is manipulation of terms.

lemma *NSDERIV-zero*:
    [| *NSDERIV g x* :> *D*;
            (∗*f*∗ *g*) (*star-of x* + *xa*) = *star-of* (*g x*);
            xa ∈ *Infinitesimal*;
            xa ≠ *0*
         |] ==> *D* = *0*
apply (*simp add*: *nsderiv-def*)
apply (*drule bspec*, *auto*)
done

lemma *NSDERIV-approx*:
    [| *NSDERIV f x* :> *D*; *h* ∈ *Infinitesimal*; *h* ≠ *0* |]
    ==> (∗*f*∗ *f*) (*star-of x* + *h*) − *star-of* (*f x*) ≈ *0*
apply (*simp add*: *nsderiv-def*)
apply (*simp add*: *mem-infmal-iff* [*symmetric*])
apply (*rule Infinitesimal-ratio*)
apply (*rule-tac* [*3*] *approx-star-of-HFinite*, *auto*)
done

lemma *NSDERIVD1*: [| *NSDERIV f (g x) :> Da*;
(*f* g) (star-of(x) + xa) ≠ star-of (g x);
(*f* g) (star-of(x) + xa) ≈ star-of (g x)
|] ==> ((*f* f) ((*f* g) (star-of(x) + xa))
− star-of (f (g x)))
/ ((*f* g) (star-of(x) + xa) − star-of (g x))
≈ star-of(Da)
by (*auto simp add*: *NSDERIV-NSLIM-iff2 NSLIM-def diff-minus* [*symmetric*])

lemma *NSDERIVD2*: [| *NSDERIV g x :> Db*; *xa* ∈ *Infinitesimal*; *xa* ≠ *0* |]
==> ((*f* g) (star-of(x) + xa) − star-of(g x)) / xa
≈ star-of(Db)
by (*auto simp add*: *NSDERIV-NSLIM-iff NSLIM-def mem-infmal-iff starfun-lambda-cancel*)

lemma *lemma-chain*: (*z*::'*a*::*real-normed-field star*) ≠ *0* ==> *x***y* = (*x***inverse*(*z*))*(*z***y*)
proof −
assume *z*: *z* ≠ *0*
have *x* * *y* = *x* * (*inverse z* * *z*) * *y* **by** (*simp add*: *z*)
thus *?thesis* **by** (*simp add*: *mult-assoc*)
qed

This proof uses both definitions of differentiability.

lemma *NSDERIV-chain*: [| *NSDERIV f (g x) :> Da*; *NSDERIV g x :> Db* |]
==> *NSDERIV* (*f o g*) *x* :> *Da* * *Db*
apply (*simp* (*no-asm-simp*) *add*: *NSDERIV-NSLIM-iff NSLIM-def*
mem-infmal-iff [*symmetric*])
apply *clarify*
apply (*frule-tac f* = *g* **in** *NSDERIV-approx*)
apply (*auto simp add*: *starfun-lambda-cancel2 starfun-o* [*symmetric*])
apply (*case-tac* (*f* g) (*star-of* (*x*) + *xa*) = *star-of* (*g x*))
apply (*drule-tac g* = *g* **in** *NSDERIV-zero*)
apply (*auto simp add*: *divide-inverse*)
apply (*rule-tac z1* = (*f* g) (*star-of* (*x*) + *xa*) − *star-of* (*g x*) **and** *y1* = *inverse xa* **in** *lemma-chain* [*THEN ssubst*])
apply (*erule hypreal-not-eq-minus-iff* [*THEN iffD1*])
apply (*rule approx-mult-star-of*)
apply (*simp-all add*: *divide-inverse* [*symmetric*])
apply (*blast intro*: *NSDERIVD1 approx-minus-iff* [*THEN iffD2*])
apply (*blast intro*: *NSDERIVD2*)
done

Differentiation of natural number powers

lemma *NSDERIV-Id* [*simp*]: *NSDERIV* (%*x*. *x*) *x* :> *1*
by (*simp add*: *NSDERIV-NSLIM-iff NSLIM-def del*: *divide-self-if*)

lemma *NSDERIV-cmult-Id* [*simp*]: *NSDERIV* (*op* * *c*) *x* :> *c*
by (*cut-tac c* = *c* **and** *x* = *x* **in** *NSDERIV-Id* [*THEN NSDERIV-cmult*], *simp*)

lemma *NSDERIV-inverse*:
fixes x :: $'a$::{*real-normed-field*,*recpower*}
shows $x \neq 0$ ==> *NSDERIV* (%x. *inverse*(x)) x :> (− (*inverse* x ^ *Suc* (*Suc* *0*)))
apply (*simp add*: *nsderiv-def*)
apply (*rule ballI*, *simp*, *clarify*)
apply (*frule* (*1*) *Infinitesimal-add-not-zero*)
apply (*simp add*: *add-commute*)

apply (*simp add*: *inverse-add nonzero-inverse-mult-distrib* [*symmetric*] *power-Suc*
nonzero-inverse-minus-eq [*symmetric*] *add-ac mult-ac diff-def*
del: *inverse-mult-distrib inverse-minus-eq*
minus-mult-left [*symmetric*] *minus-mult-right* [*symmetric*])
apply (*subst mult-commute*, *simp add*: *nonzero-mult-divide-cancel-right*)
apply (*simp* (*no-asm-simp*) *add*: *mult-assoc* [*symmetric*] *left-distrib*
del: *minus-mult-left* [*symmetric*] *minus-mult-right* [*symmetric*])
apply (*rule-tac y* = *inverse* (− (*star-of x* ∗ *star-of x*)) **in** *approx-trans*)
apply (*rule inverse-add-Infinitesimal-approx2*)
apply (*auto dest*!: *hypreal-of-real-HFinite-diff-Infinitesimal*
simp add: *inverse-minus-eq* [*symmetric*] *HFinite-minus-iff*)
apply (*rule Infinitesimal-HFinite-mult*, *auto*)
done

44.1.1 Equivalence of NS and Standard definitions

lemma *divideR-eq-divide*: $x /_R y = x / y$
by (*simp add*: *real-scaleR-def divide-inverse mult-commute*)

Now equivalence between NSDERIV and DERIV

lemma *NSDERIV-DERIV-iff*: (*NSDERIV f x* :> *D*) = (*DERIV f x* :> *D*)
by (*simp add*: *deriv-def NSDERIV-NSLIM-iff LIM-NSLIM-iff*)

lemma *NSDERIV-pow*: *NSDERIV* (%x. x ^ n) x :> *real* n ∗ (x ^ (n − *Suc* *0*))
by (*simp add*: *NSDERIV-DERIV-iff DERIV-pow*)

Derivative of inverse

lemma *NSDERIV-inverse-fun*:
fixes x :: $'a$::{*real-normed-field*,*recpower*}
shows [| *NSDERIV f x* :> d; $f(x) \neq 0$ |]
==> *NSDERIV* (%x. *inverse*(*f x*)) x :> (− (d ∗ *inverse*($f(x)$ ^ *Suc* (*Suc* *0*))))
by (*simp add*: *NSDERIV-DERIV-iff DERIV-inverse-fun del*: *realpow-Suc*)

Derivative of quotient

lemma *NSDERIV-quotient*:
fixes x :: $'a$::{*real-normed-field*,*recpower*}

shows [| *NSDERIV f x :> d*; *NSDERIV g x :> e*; $g(x) \neq 0$ |]
==> *NSDERIV* (%*y*. *f*(*y*) / (*g y*)) *x* :> (*d*∗*g*(*x*)
− (*e*∗*f*(*x*))) / (*g*(*x*) ^ *Suc* (*Suc* 0))
by (*simp add*: *NSDERIV-DERIV-iff DERIV-quotient del*: *realpow-Suc*)

lemma *CARAT-NSDERIV*: *NSDERIV f x :> l* ==>
∃ *g*. (∀ *z*. *f z* − *f x* = *g z* ∗ (*z*−*x*)) & *isNSCont g x* & *g x* = *l*
by (*auto simp add*: *NSDERIV-DERIV-iff isNSCont-isCont-iff CARAT-DERIV*
mult-commute)

lemma *hypreal-eq-minus-iff3*: (*x* = *y* + *z*) = (*x* + −*z* = (*y*::*hypreal*))
by *auto*

lemma *CARAT-DERIVD*:
assumes *all*: ∀ *z*. *f z* − *f x* = *g z* ∗ (*z*−*x*)
and *nsc*: *isNSCont g x*
shows *NSDERIV f x :> g x*
proof −
from *nsc*
have ∀ *w*. *w* ≠ *star-of x* ∧ *w* ≈ *star-of x* ⟶
(∗*f*∗ *g*) *w* ∗ (*w* − *star-of x*) / (*w* − *star-of x*) ≈
star-of (*g x*)
by (*simp add*: *isNSCont-def nonzero-mult-divide-cancel-right*)
thus *?thesis* **using** *all*
by (*simp add*: *NSDERIV-iff2 starfun-if-eq cong*: *if-cong*)
qed

44.1.2 Differentiability predicate

lemma *NSdifferentiableD*: *f NSdifferentiable x* ==> ∃ *D*. *NSDERIV f x :> D*
by (*simp add*: *NSdifferentiable-def*)

lemma *NSdifferentiableI*: *NSDERIV f x :> D* ==> *f NSdifferentiable x*
by (*force simp add*: *NSdifferentiable-def*)

44.2 (NS) Increment

lemma *incrementI*:
f NSdifferentiable x ==>
increment f x h = (∗*f*∗ *f*) (*hypreal-of-real*(*x*) + *h*) −
hypreal-of-real (*f x*)
by (*simp add*: *increment-def*)

lemma *incrementI2*: *NSDERIV f x :> D* ==>
increment f x h = (∗*f*∗ *f*) (*hypreal-of-real*(*x*) + *h*) −
hypreal-of-real (*f x*)
apply (*erule NSdifferentiableI* [*THEN incrementI*])
done

lemma *increment-thm*: [| *NSDERIV f x :> D*; *h* ∈ *Infinitesimal*; *h* ≠ *0* |]
==> ∃ *e* ∈ *Infinitesimal*. *increment f x h* = *hypreal-of-real*(*D*)∗*h* + *e*∗*h*
apply (*frule-tac h* = *h* **in** *incrementI2*, *simp add*: *nsderiv-def*)
apply (*drule bspec*, *auto*)
apply (*drule bex-Infinitesimal-iff2* [*THEN iffD2*], *clarify*)
apply (*frule-tac b1* = *hypreal-of-real* (*D*) + *y*
in *hypreal-mult-right-cancel* [*THEN iffD2*])
apply (*erule-tac* [*2*] *V* = ((∗*f*∗ *f*) (*hypreal-of-real* (*x*) + *h*) − *hypreal-of-real* (*f* *x*)) / *h* = *hypreal-of-real* (*D*) + *y* **in** *thin-rl*)
apply *assumption*
apply (*simp add*: *times-divide-eq-right* [*symmetric*])
apply (*auto simp add*: *left-distrib*)
done

lemma *increment-thm2*:
[| *NSDERIV f x :> D*; *h* ≈ *0*; *h* ≠ *0* |]
==> ∃ *e* ∈ *Infinitesimal*. *increment f x h* =
hypreal-of-real(*D*)∗*h* + *e*∗*h*
by (*blast dest*!: *mem-infmal-iff* [*THEN iffD2*] *intro*!: *increment-thm*)

lemma *increment-approx-zero*: [| *NSDERIV f x :> D*; *h* ≈ *0*; *h* ≠ *0* |]
==> *increment f x h* ≈ *0*
apply (*drule increment-thm2*,
auto intro!: *Infinitesimal-HFinite-mult2 HFinite-add simp add*: *left-distrib* [*symmetric*] *mem-infmal-iff* [*symmetric*])
apply (*erule Infinitesimal-subset-HFinite* [*THEN subsetD*])
done

end

45 HTranscendental: Nonstandard Extensions of Transcendental Functions

theory *HTranscendental*
imports *Transcendental HSeries HDeriv*
begin

definition
exphr :: *real* => *hypreal* **where**
— define exponential function using standard part
exphr x = *st*(*sumhr* (*0*, *whn*, %*n*. *inverse*(*real* (*fact n*)) ∗ (*x* ^ *n*)))

definition
sinhr :: *real* => *hypreal* **where**
sinhr x = *st*(*sumhr* (*0*, *whn*, %*n*. (*if even*(*n*) *then 0 else*
((−*1*) ^ ((*n* − *1*) *div 2*))/(*real* (*fact n*))) ∗ (*x* ^ *n*)))

definition
coshr :: *real* => *hypreal* **where**
coshr x = *st*(*sumhr* (*0*, *whn*, %*n*. (*if even*(*n*) *then*
((−*1*) ^ (*n div 2*))/(*real* (*fact n*)) *else 0*) * *x* ^ *n*))

45.1 Nonstandard Extension of Square Root Function

lemma *STAR-sqrt-zero* [*simp*]: (*f sqrt) 0 = 0*
by (*simp add*: *starfun star-n-zero-num*)

lemma *STAR-sqrt-one* [*simp*]: (*f sqrt) 1 = 1*
by (*simp add*: *starfun star-n-one-num*)

lemma *hypreal-sqrt-pow2-iff*: (((*f sqrt)*(*x*) ^ *2* = *x*) = (*0* ≤ *x*)
apply (*cases x*)
apply (*auto simp add*: *star-n-le star-n-zero-num starfun hrealpow star-n-eq-iff*
simp del: *hpowr-Suc realpow-Suc*)
done

lemma *hypreal-sqrt-gt-zero-pow2*: !!*x*. *0* < *x* ==> (*f sqrt*) (*x*) ^ *2* = *x*
by (*transfer*, *simp*)

lemma *hypreal-sqrt-pow2-gt-zero*: *0* < *x* ==> *0* < (*f sqrt*) (*x*) ^ *2*
by (*frule hypreal-sqrt-gt-zero-pow2*, *auto*)

lemma *hypreal-sqrt-not-zero*: *0* < *x* ==> (*f sqrt*) (*x*) ≠ *0*
apply (*frule hypreal-sqrt-pow2-gt-zero*)
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *hypreal-inverse-sqrt-pow2*:
0 < *x* ==> *inverse* (((*f sqrt)*(*x*)) ^ *2* = *inverse x*
apply (*cut-tac n* = *2* **and** *a* = (*f sqrt*) *x* **in** *power-inverse* [*symmetric*])
apply (*auto dest*: *hypreal-sqrt-gt-zero-pow2*)
done

lemma *hypreal-sqrt-mult-distrib*:
!!*x y*. [|*0* < *x*; *0* <*y* |] ==>
(*f sqrt)*(*x**y*) = (*f sqrt)*(*x*) * (*f sqrt)*(*y*)
apply *transfer*
apply (*auto intro*: *real-sqrt-mult-distrib*)
done

lemma *hypreal-sqrt-mult-distrib2*:
[|*0*≤*x*; *0*≤*y* |] ==>
(*f sqrt)*(*x**y*) = (*f sqrt)*(*x*) * (*f sqrt)*(*y*)
by (*auto intro*: *hypreal-sqrt-mult-distrib simp add*: *order-le-less*)

lemma *hypreal-sqrt-approx-zero* [*simp*]:
  $0 < x \Longrightarrow ((*f* sqrt)(x) \mathbin{@=} 0) = (x \mathbin{@=} 0)$
apply (*auto simp add*: *mem-infmal-iff* [*symmetric*])
apply (*rule hypreal-sqrt-gt-zero-pow2* [*THEN subst*])
apply (*auto intro*: *Infinitesimal-mult*
    dest!: *hypreal-sqrt-gt-zero-pow2* [*THEN ssubst*]
    simp add: *numeral-2-eq-2*)
done

lemma *hypreal-sqrt-approx-zero2* [*simp*]:
  $0 \leq x \Longrightarrow ((*f* sqrt)(x) \mathbin{@=} 0) = (x \mathbin{@=} 0)$
by (*auto simp add*: *order-le-less*)

lemma *hypreal-sqrt-sum-squares* [*simp*]:
  $((*f* sqrt)(x*x + y*y + z*z) \mathbin{@=} 0) = (x*x + y*y + z*z \mathbin{@=} 0)$
apply (*rule hypreal-sqrt-approx-zero2*)
apply (*rule add-nonneg-nonneg*)+
apply (*auto*)
done

lemma *hypreal-sqrt-sum-squares2* [*simp*]:
  $((*f* sqrt)(x*x + y*y) \mathbin{@=} 0) = (x*x + y*y \mathbin{@=} 0)$
apply (*rule hypreal-sqrt-approx-zero2*)
apply (*rule add-nonneg-nonneg*)
apply (*auto*)
done

lemma *hypreal-sqrt-gt-zero*: $!!x.\ 0 < x \Longrightarrow 0 < (*f* sqrt)(x)$
apply *transfer*
apply (*auto intro*: *real-sqrt-gt-zero*)
done

lemma *hypreal-sqrt-ge-zero*: $0 \leq x \Longrightarrow 0 \leq (*f* sqrt)(x)$
by (*auto intro*: *hypreal-sqrt-gt-zero simp add*: *order-le-less*)

lemma *hypreal-sqrt-hrabs* [*simp*]: $!!x.\ (*f* sqrt)(x \hat{}\ 2) = abs(x)$
by (*transfer*, *simp*)

lemma *hypreal-sqrt-hrabs2* [*simp*]: $!!x.\ (*f* sqrt)(x*x) = abs(x)$
by (*transfer*, *simp*)

lemma *hypreal-sqrt-hyperpow-hrabs* [*simp*]:
  $!!x.\ (*f* sqrt)(x\ pow\ (hypnat\text{-}of\text{-}nat\ 2)) = abs(x)$
by (*transfer*, *simp*)

lemma *star-sqrt-HFinite*: $[\![x \in HFinite;\ 0 \leq x]\!] \Longrightarrow (*f* sqrt)\ x \in HFinite$
apply (*rule HFinite-square-iff* [*THEN iffD1*])
apply (*simp only*: *hypreal-sqrt-mult-distrib2* [*symmetric*], *simp*)
done

lemma *st-hypreal-sqrt*:
[| *x* ∈ *HFinite*; *0* ≤ *x* |] ==> *st*((*f* *sqrt*) *x*) = (*f* *sqrt*)(*st x*)
apply (*rule power-inject-base* [**where** *n=1*])
apply (*auto intro*!: *st-zero-le hypreal-sqrt-ge-zero*)
apply (*rule st-mult* [*THEN subst*])
apply (*rule-tac* [*3*] *hypreal-sqrt-mult-distrib2* [*THEN subst*])
apply (*rule-tac* [*5*] *hypreal-sqrt-mult-distrib2* [*THEN subst*])
apply (*auto simp add*: *st-hrabs st-zero-le star-sqrt-HFinite*)
done

lemma *hypreal-sqrt-sum-squares-ge1* [*simp*]: !!*x y*. *x* ≤ (*f* *sqrt*)(*x* ^ *2* + *y* ^ *2*)
by *transfer* (*rule real-sqrt-sum-squares-ge1*)

lemma *HFinite-hypreal-sqrt*:
[| *0* ≤ *x*; *x* ∈ *HFinite* |] ==> (*f* *sqrt*) *x* ∈ *HFinite*
apply (*auto simp add*: *order-le-less*)
apply (*rule HFinite-square-iff* [*THEN iffD1*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2*)
done

lemma *HFinite-hypreal-sqrt-imp-HFinite*:
[| *0* ≤ *x*; (*f* *sqrt*) *x* ∈ *HFinite* |] ==> *x* ∈ *HFinite*
apply (*auto simp add*: *order-le-less*)
apply (*drule HFinite-square-iff* [*THEN iffD2*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2 del*: *HFinite-square-iff*)
done

lemma *HFinite-hypreal-sqrt-iff* [*simp*]:
0 ≤ *x* ==> ((*f* *sqrt*) *x* ∈ *HFinite*) = (*x* ∈ *HFinite*)
by (*blast intro*: *HFinite-hypreal-sqrt HFinite-hypreal-sqrt-imp-HFinite*)

lemma *HFinite-sqrt-sum-squares* [*simp*]:
((*f* *sqrt*)(*x**x + *y**y) ∈ *HFinite*) = (*x**x + *y**y ∈ *HFinite*)
apply (*rule HFinite-hypreal-sqrt-iff*)
apply (*rule add-nonneg-nonneg*)
apply (*auto*)
done

lemma *Infinitesimal-hypreal-sqrt*:
[| *0* ≤ *x*; *x* ∈ *Infinitesimal* |] ==> (*f* *sqrt*) *x* ∈ *Infinitesimal*
apply (*auto simp add*: *order-le-less*)
apply (*rule Infinitesimal-square-iff* [*THEN iffD2*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2*)
done

lemma *Infinitesimal-hypreal-sqrt-imp-Infinitesimal*:
[| $0 \leq x$; ($*f*$ *sqrt*) $x \in$ *Infinitesimal* |] ==> $x \in$ *Infinitesimal*
apply (*auto simp add*: *order-le-less*)
apply (*drule Infinitesimal-square-iff* [*THEN iffD1*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2 del*: *Infinitesimal-square-iff* [*symmetric*])
done

lemma *Infinitesimal-hypreal-sqrt-iff* [*simp*]:
$0 \leq x$ ==> (($*f*$ *sqrt*) $x \in$ *Infinitesimal*) = ($x \in$ *Infinitesimal*)
by (*blast intro*: *Infinitesimal-hypreal-sqrt-imp-Infinitesimal Infinitesimal-hypreal-sqrt*)

lemma *Infinitesimal-sqrt-sum-squares* [*simp*]:
(($*f*$ *sqrt*)($x*x$ + $y*y$) $\in$ *Infinitesimal*) = ($x*x$ + $y*y$ $\in$ *Infinitesimal*)
apply (*rule Infinitesimal-hypreal-sqrt-iff*)
apply (*rule add-nonneg-nonneg*)
apply (*auto*)
done

lemma *HInfinite-hypreal-sqrt*:
[| $0 \leq x$; $x \in$ *HInfinite* |] ==> ($*f*$ *sqrt*) $x \in$ *HInfinite*
apply (*auto simp add*: *order-le-less*)
apply (*rule HInfinite-square-iff* [*THEN iffD1*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2*)
done

lemma *HInfinite-hypreal-sqrt-imp-HInfinite*:
[| $0 \leq x$; ($*f*$ *sqrt*) $x \in$ *HInfinite* |] ==> $x \in$ *HInfinite*
apply (*auto simp add*: *order-le-less*)
apply (*drule HInfinite-square-iff* [*THEN iffD2*])
apply (*drule hypreal-sqrt-gt-zero-pow2*)
apply (*simp add*: *numeral-2-eq-2 del*: *HInfinite-square-iff*)
done

lemma *HInfinite-hypreal-sqrt-iff* [*simp*]:
$0 \leq x$ ==> (($*f*$ *sqrt*) $x \in$ *HInfinite*) = ($x \in$ *HInfinite*)
by (*blast intro*: *HInfinite-hypreal-sqrt HInfinite-hypreal-sqrt-imp-HInfinite*)

lemma *HInfinite-sqrt-sum-squares* [*simp*]:
(($*f*$ *sqrt*)($x*x$ + $y*y$) $\in$ *HInfinite*) = ($x*x$ + $y*y$ $\in$ *HInfinite*)
apply (*rule HInfinite-hypreal-sqrt-iff*)
apply (*rule add-nonneg-nonneg*)
apply (*auto*)
done

lemma *HFinite-exp* [*simp*]:
sumhr (*0*, *whn*, %*n*. *inverse* (*real* (*fact n*)) $*$ x ^ n) $\in$ *HFinite*
unfolding *sumhr-app*

apply (*simp only*: *star-zero-def starfun2-star-of*)
apply (*rule NSBseqD2*)
apply (*rule NSconvergent-NSBseq*)
apply (*rule convergent-NSconvergent-iff* [*THEN iffD1*])
apply (*rule summable-convergent-sumr-iff* [*THEN iffD1*])
apply (*rule summable-exp*)
done

lemma *exphr-zero* [*simp*]: *exphr 0 = 1*
apply (*simp add*: *exphr-def sumhr-split-add*
[*OF hypnat-one-less-hypnat-omega*, *symmetric*])
apply (*rule st-unique*, *simp*)
apply (*rule subst* [**where** *P*=$\lambda x.\ 1 \approx x$, *OF - approx-refl*])
apply (*rule rev-mp* [*OF hypnat-one-less-hypnat-omega*])
apply (*rule-tac x*=*whn* **in** *spec*)
apply (*unfold sumhr-app*, *transfer*, *simp*)
done

lemma *coshr-zero* [*simp*]: *coshr 0 = 1*
apply (*simp add*: *coshr-def sumhr-split-add*
[*OF hypnat-one-less-hypnat-omega*, *symmetric*])
apply (*rule st-unique*, *simp*)
apply (*rule subst* [**where** *P*=$\lambda x.\ 1 \approx x$, *OF - approx-refl*])
apply (*rule rev-mp* [*OF hypnat-one-less-hypnat-omega*])
apply (*rule-tac x*=*whn* **in** *spec*)
apply (*unfold sumhr-app*, *transfer*, *simp*)
done

lemma *STAR-exp-zero-approx-one* [*simp*]: ($*f*$ *exp*) (*0*::*hypreal*) @= *1*
apply (*subgoal-tac* ($*f*$ *exp*) (*0*::*hypreal*) = *1*, *simp*)
apply (*transfer*, *simp*)
done

lemma *STAR-exp-Infinitesimal*: *x* ∈ *Infinitesimal* ==> ($*f*$ *exp*) (*x*::*hypreal*)
@= *1*
apply (*case-tac x = 0*)
apply (*cut-tac* [*2*] *x = 0* **in** *DERIV-exp*)
apply (*auto simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)
apply (*drule-tac x = x* **in** *bspec*, *auto*)
apply (*drule-tac c = x* **in** *approx-mult1*)
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
simp add: *mult-assoc*)
apply (*rule approx-add-right-cancel* [**where** *d*=−*1*])
apply (*rule approx-sym* [*THEN* [*2*] *approx-trans2*])
apply (*auto simp add*: *diff-def mem-infmal-iff*)
done

lemma *STAR-exp-epsilon* [*simp*]: ($*f*$ *exp*) *epsilon* @= *1*
by (*auto intro*: *STAR-exp-Infinitesimal*)

lemma *STAR-exp-add*: !!$x\ y.\ (*f* exp)(x + y) = (*f* exp)\ x * (*f* exp)\ y$
by *transfer* (*rule exp-add*)

lemma *exphr-hypreal-of-real-exp-eq*: *exphr x = hypreal-of-real* (*exp x*)
apply (*simp add*: *exphr-def*)
apply (*rule st-unique*, *simp*)
apply (*subst starfunNat-sumr* [*symmetric*])
apply (*rule NSLIMSEQ-D* [*THEN approx-sym*])
apply (*rule LIMSEQ-NSLIMSEQ*)
apply (*subst sums-def* [*symmetric*])
apply (*cut-tac exp-converges* [**where** *x*=*x*], *simp*)
apply (*rule HNatInfinite-whn*)
done

lemma *starfun-exp-ge-add-one-self* [*simp*]: !!*x*::*hypreal*. *0* ≤ *x* ==> (*1* + *x*) ≤ (*f* exp) *x*
by *transfer* (*rule exp-ge-add-one-self-aux*)

lemma *starfun-exp-HInfinite*:
[| *x* ∈ *HInfinite*; *0* ≤ *x* |] ==> (*f* exp) (*x*::*hypreal*) ∈ *HInfinite*
apply (*frule starfun-exp-ge-add-one-self*)
apply (*rule HInfinite-ge-HInfinite*, *assumption*)
apply (*rule order-trans* [*of* - *1*+*x*], *auto*)
done

lemma *starfun-exp-minus*: !!*x*. (*f* exp) (−*x*) = *inverse*((*f* exp) *x*)
by *transfer* (*rule exp-minus*)

lemma *starfun-exp-Infinitesimal*:
[| *x* ∈ *HInfinite*; *x* ≤ *0* |] ==> (*f* exp) (*x*::*hypreal*) ∈ *Infinitesimal*
apply (*subgoal-tac* ∃ *y*. *x* = − *y*)
apply (*rule-tac* [*2*] *x* = − *x* **in** *exI*)
apply (*auto intro*!: *HInfinite-inverse-Infinitesimal starfun-exp-HInfinite*
simp add: *starfun-exp-minus HInfinite-minus-iff*)
done

lemma *starfun-exp-gt-one* [*simp*]: !!*x*::*hypreal*. *0* < *x* ==> *1* < (*f* exp) *x*
by *transfer* (*rule exp-gt-one*)

lemma *starfun-ln-exp* [*simp*]: !!*x*. (*f* ln) ((*f* exp) *x*) = *x*
by *transfer* (*rule ln-exp*)

lemma *starfun-exp-ln-iff* [*simp*]: !!*x*. ((*f* exp)((*f* ln) *x*) = *x*) = (*0* < *x*)

by *transfer* (*rule exp-ln-iff*)

lemma *starfun-exp-ln-eq*: !!*u x*. (∗*f*∗ *exp*) *u* = *x* ==> (∗*f*∗ *ln*) *x* = *u*
by *transfer* (*rule exp-ln-eq*)

lemma *starfun-ln-less-self* [*simp*]: !!*x*. *0* < *x* ==> (∗*f*∗ *ln*) *x* < *x*
by *transfer* (*rule ln-less-self*)

lemma *starfun-ln-ge-zero* [*simp*]: !!*x*. *1* ≤ *x* ==> *0* ≤ (∗*f*∗ *ln*) *x*
by *transfer* (*rule ln-ge-zero*)

lemma *starfun-ln-gt-zero* [*simp*]: !!*x* .*1* < *x* ==> *0* < (∗*f*∗ *ln*) *x*
by *transfer* (*rule ln-gt-zero*)

lemma *starfun-ln-not-eq-zero* [*simp*]: !!*x*. [| *0* < *x*; *x* ≠ *1* |] ==> (∗*f*∗ *ln*) *x* ≠ *0*
by *transfer simp*

lemma *starfun-ln-HFinite*: [| *x* ∈ *HFinite*; *1* ≤ *x* |] ==> (∗*f*∗ *ln*) *x* ∈ *HFinite*
apply (*rule HFinite-bounded*)
apply *assumption*
apply (*simp-all add*: *starfun-ln-less-self order-less-imp-le*)
done

lemma *starfun-ln-inverse*: !!*x*. *0* < *x* ==> (∗*f*∗ *ln*) (*inverse x*) = −(∗*f*∗ *ln*) *x*
by *transfer* (*rule ln-inverse*)

lemma *starfun-abs-exp-cancel*: ⋀*x*. |(∗*f*∗ *exp*) (*x*::*hypreal*)| = (∗*f*∗ *exp*) *x*
by *transfer* (*rule abs-exp-cancel*)

lemma *starfun-exp-less-mono*: ⋀*x y*::*hypreal*. *x* < *y* ⟹ (∗*f*∗ *exp*) *x* < (∗*f*∗ *exp*) *y*
by *transfer* (*rule exp-less-mono*)

lemma *starfun-exp-HFinite*: *x* ∈ *HFinite* ==> (∗*f*∗ *exp*) (*x*::*hypreal*) ∈ *HFinite*
apply (*auto simp add*: *HFinite-def*, *rename-tac u*)
apply (*rule-tac x*=(∗*f*∗ *exp*) *u* **in** *rev-bexI*)
apply (*simp add*: *Reals-eq-Standard*)
apply (*simp add*: *starfun-abs-exp-cancel*)
apply (*simp add*: *starfun-exp-less-mono*)
done

lemma *starfun-exp-add-HFinite-Infinitesimal-approx*:
[|*x* ∈ *Infinitesimal*; *z* ∈ *HFinite* |] ==> (∗*f*∗ *exp*) (*z* + *x*::*hypreal*) @= (∗*f*∗ *exp*) *z*
apply (*simp add*: *STAR-exp-add*)
apply (*frule STAR-exp-Infinitesimal*)
apply (*drule approx-mult2*)
apply (*auto intro*: *starfun-exp-HFinite*)
done

lemma *starfun-ln-HInfinite*:
[| $x \in$ *HInfinite*; $0 < x$ |] ==> ($*f*$ *ln*) $x \in$ *HInfinite*
apply (*rule ccontr*, *drule HFinite-HInfinite-iff* [*THEN iffD2*])
apply (*drule starfun-exp-HFinite*)
apply (*simp add*: *starfun-exp-ln-iff* [*THEN iffD2*] *HFinite-HInfinite-iff*)
done

lemma *starfun-exp-HInfinite-Infinitesimal-disj*:
$x \in$ *HInfinite* ==> ($*f*$ *exp*) $x \in$ *HInfinite* | ($*f*$ *exp*) (x::*hypreal*) $\in$ *Infinitesimal*
apply (*insert linorder-linear* [*of x 0*])
apply (*auto intro*: *starfun-exp-HInfinite starfun-exp-Infinitesimal*)
done

lemma *starfun-ln-HFinite-not-Infinitesimal*:
[| $x \in$ *HFinite* $-$ *Infinitesimal*; $0 < x$ |] ==> ($*f*$ *ln*) $x \in$ *HFinite*
apply (*rule ccontr*, *drule HInfinite-HFinite-iff* [*THEN iffD2*])
apply (*drule starfun-exp-HInfinite-Infinitesimal-disj*)
apply (*simp add*: *starfun-exp-ln-iff* [*symmetric*] *HInfinite-HFinite-iff*
del: *starfun-exp-ln-iff*)
done

lemma *starfun-ln-Infinitesimal-HInfinite*:
[| $x \in$ *Infinitesimal*; $0 < x$ |] ==> ($*f*$ *ln*) $x \in$ *HInfinite*
apply (*drule Infinitesimal-inverse-HInfinite*)
apply (*frule positive-imp-inverse-positive*)
apply (*drule-tac* [*2*] *starfun-ln-HInfinite*)
apply (*auto simp add*: *starfun-ln-inverse HInfinite-minus-iff*)
done

lemma *starfun-ln-less-zero*: !!x. [| $0 < x$; $x < 1$ |] ==> ($*f*$ *ln*) $x < 0$
by *transfer* (*rule ln-less-zero*)

lemma *starfun-ln-Infinitesimal-less-zero*:
[| $x \in$ *Infinitesimal*; $0 < x$ |] ==> ($*f*$ *ln*) $x < 0$
by (*auto intro*!: *starfun-ln-less-zero simp add*: *Infinitesimal-def*)

lemma *starfun-ln-HInfinite-gt-zero*:
[| $x \in$ *HInfinite*; $0 < x$ |] ==> $0 <$ ($*f*$ *ln*) x
by (*auto intro*!: *starfun-ln-gt-zero simp add*: *HInfinite-def*)

lemma *HFinite-sin* [*simp*]:
sumhr (0, *whn*, %n. (*if even*(n) *then 0 else*

```
        (−1 ^ ((n − 1) div 2))/(real (fact n))) ∗ x ^ n)
        ∈ HFinite
unfolding sumhr-app
apply (simp only: star-zero-def starfun2-star-of)
apply (rule NSBseqD2)
apply (rule NSconvergent-NSBseq)
apply (rule convergent-NSconvergent-iff [THEN iffD1])
apply (rule summable-convergent-sumr-iff [THEN iffD1])
apply (simp only: One-nat-def summable-sin)
done

lemma STAR-sin-zero [simp]: ( ∗f∗ sin) 0 = 0
by transfer (rule sin-zero)

lemma STAR-sin-Infinitesimal [simp]: x ∈ Infinitesimal ==> ( ∗f∗ sin) x @= x
apply (case-tac x = 0)
apply (cut-tac [2] x = 0 in DERIV-sin)
apply (auto simp add: NSDERIV-DERIV-iff [symmetric] nsderiv-def)
apply (drule bspec [where x = x], auto)
apply (drule approx-mult1 [where c = x])
apply (auto intro: Infinitesimal-subset-HFinite [THEN subsetD]
        simp add: mult-assoc)
done

lemma HFinite-cos [simp]:
    sumhr (0, whn, %n. (if even(n) then
        (−1 ^ (n div 2))/(real (fact n)) else
        0) ∗ x ^ n) ∈ HFinite
unfolding sumhr-app
apply (simp only: star-zero-def starfun2-star-of)
apply (rule NSBseqD2)
apply (rule NSconvergent-NSBseq)
apply (rule convergent-NSconvergent-iff [THEN iffD1])
apply (rule summable-convergent-sumr-iff [THEN iffD1])
apply (rule summable-cos)
done

lemma STAR-cos-zero [simp]: ( ∗f∗ cos) 0 = 1
by transfer (rule cos-zero)

lemma STAR-cos-Infinitesimal [simp]: x ∈ Infinitesimal ==> ( ∗f∗ cos) x @= 1
apply (case-tac x = 0)
apply (cut-tac [2] x = 0 in DERIV-cos)
apply (auto simp add: NSDERIV-DERIV-iff [symmetric] nsderiv-def)
apply (drule bspec [where x = x])
apply auto
apply (drule approx-mult1 [where c = x])
apply (auto intro: Infinitesimal-subset-HFinite [THEN subsetD]
        simp add: mult-assoc)
```

apply (*rule approx-add-right-cancel* [**where** $d = -1$])
apply (*simp add*: *diff-def*)
done

lemma *STAR-tan-zero* [*simp*]: ($*f*$ *tan*) *0* = *0*
by *transfer* (*rule tan-zero*)

lemma *STAR-tan-Infinitesimal*: $x \in$ *Infinitesimal* ==> ($*f*$ *tan*) *x* @= *x*
apply (*case-tac x* = *0*)
apply (*cut-tac* [*2*] *x* = *0* **in** *DERIV-tan*)
apply (*auto simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)
apply (*drule bspec* [**where** *x* = *x*], *auto*)
apply (*drule approx-mult1* [**where** *c* = *x*])
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
simp *add*: *mult-assoc*)
done

lemma *STAR-sin-cos-Infinitesimal-mult*:
x ∈ *Infinitesimal* ==> ($*f*$ *sin*) *x* * ($*f*$ *cos*) *x* @= *x*
apply (*insert approx-mult-HFinite* [*of* ($*f*$ *sin*) *x* - ($*f*$ *cos*) *x 1*])
apply (*simp add*: *Infinitesimal-subset-HFinite* [*THEN subsetD*])
done

lemma *HFinite-pi*: *hypreal-of-real pi* ∈ *HFinite*
by *simp*

lemma *lemma-split-hypreal-of-real*:
N ∈ *HNatInfinite*
==> *hypreal-of-real a* =
hypreal-of-hypnat N * (*inverse*(*hypreal-of-hypnat N*) * *hypreal-of-real a*)
by (*simp add*: *mult-assoc* [*symmetric*] *zero-less-HNatInfinite*)

lemma *STAR-sin-Infinitesimal-divide*:
[|*x* ∈ *Infinitesimal*; *x* ≠ *0* |] ==> ($*f*$ *sin*) *x*/*x* @= *1*
apply (*cut-tac x* = *0* **in** *DERIV-sin*)
apply (*simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)
done

lemma *lemma-sin-pi*:
n ∈ *HNatInfinite*
==> ($*f*$ *sin*) (*inverse* (*hypreal-of-hypnat n*))/(*inverse* (*hypreal-of-hypnat n*)) @= *1*
apply (*rule STAR-sin-Infinitesimal-divide*)

apply (*auto simp add*: *zero-less-HNatInfinite*)
done

lemma *STAR-sin-inverse-HNatInfinite*:
"*n* ∈ *HNatInfinite*
==> (*f* *) (*inverse* (*hypreal-of-hypnat n*)) * *hypreal-of-hypnat n* @= *1*"
apply (*frule lemma-sin-pi*)
apply (*simp add*: *divide-inverse*)
done

lemma *Infinitesimal-pi-divide-HNatInfinite*:
"*N* ∈ *HNatInfinite*
==> *hypreal-of-real pi*/(*hypreal-of-hypnat N*) ∈ *Infinitesimal*"
apply (*simp add*: *divide-inverse*)
apply (*auto intro*: *Infinitesimal-HFinite-mult2*)
done

lemma *pi-divide-HNatInfinite-not-zero* [*simp*]:
"*N* ∈ *HNatInfinite* ==> *hypreal-of-real pi*/(*hypreal-of-hypnat N*) ≠ *0*"
by (*simp add*: *zero-less-HNatInfinite*)

lemma *STAR-sin-pi-divide-HNatInfinite-approx-pi*:
"*n* ∈ *HNatInfinite*
==> (*f* *) (*hypreal-of-real pi*/(*hypreal-of-hypnat n*)) * *hypreal-of-hypnat n*
@= *hypreal-of-real pi*"
apply (*frule STAR-sin-Infinitesimal-divide*
[*OF Infinitesimal-pi-divide-HNatInfinite*
pi-divide-HNatInfinite-not-zero])
apply (*auto*)
apply (*rule approx-SReal-mult-cancel* [*of inverse* (*hypreal-of-real pi*)])
apply (*auto intro*: *Reals-inverse simp add*: *divide-inverse mult-ac*)
done

lemma *STAR-sin-pi-divide-HNatInfinite-approx-pi2*:
"*n* ∈ *HNatInfinite*
==> *hypreal-of-hypnat n* *
(*f* *) (*hypreal-of-real pi*/(*hypreal-of-hypnat n*))
@= *hypreal-of-real pi*"
apply (*rule mult-commute* [*THEN subst*])
apply (*erule STAR-sin-pi-divide-HNatInfinite-approx-pi*)
done

lemma *starfunNat-pi-divide-n-Infinitesimal*:
"*N* ∈ *HNatInfinite* ==> (*f* *) (%*x*. *pi* / *real x*)) *N* ∈ *Infinitesimal*"
by (*auto intro*!: *Infinitesimal-HFinite-mult2*
simp add: *starfun-mult* [*symmetric*] *divide-inverse*
starfun-inverse [*symmetric*] *starfunNat-real-of-nat*)

lemma *STAR-sin-pi-divide-n-approx*:
  $N \in HNatInfinite \Longrightarrow$
  $(*f* sin)((*f* (\%x.\ pi\ /\ real\ x))\ N) \mathbin{@=}$
  $hypreal\text{-}of\text{-}real\ pi/(hypreal\text{-}of\text{-}hypnat\ N)$
apply (*simp add*: *starfunNat-real-of-nat* [*symmetric*])
apply (*rule STAR-sin-Infinitesimal*)
apply (*simp add*: *divide-inverse*)
apply (*rule Infinitesimal-HFinite-mult2*)
apply (*subst starfun-inverse*)
apply (*erule starfunNat-inverse-real-of-nat-Infinitesimal*)
apply *simp*
done

lemma *NSLIMSEQ-sin-pi*: $(\%n.\ real\ n * sin\ (pi\ /\ real\ n)) ----NS> pi$
apply (*auto simp add*: *NSLIMSEQ-def starfun-mult* [*symmetric*] *starfunNat-real-of-nat*)
apply (*rule-tac f1* = *sin* **in** *starfun-o2* [*THEN subst*])
apply (*auto simp add*: *starfun-mult* [*symmetric*] *starfunNat-real-of-nat divide-inverse*)
apply (*rule-tac f1* = *inverse* **in** *starfun-o2* [*THEN subst*])
apply (*auto dest*: *STAR-sin-pi-divide-HNatInfinite-approx-pi*
    simp add: *starfunNat-real-of-nat mult-commute divide-inverse*)
done

lemma *NSLIMSEQ-cos-one*: $(\%n.\ cos\ (pi\ /\ real\ n)) ----NS> 1$
apply (*simp add*: *NSLIMSEQ-def*, *auto*)
apply (*rule-tac f1* = *cos* **in** *starfun-o2* [*THEN subst*])
apply (*rule STAR-cos-Infinitesimal*)
apply (*auto intro*!: *Infinitesimal-HFinite-mult2*
    simp add: *starfun-mult* [*symmetric*] *divide-inverse*
      starfun-inverse [*symmetric*] *starfunNat-real-of-nat*)
done

lemma *NSLIMSEQ-sin-cos-pi*:
  $(\%n.\ real\ n * sin\ (pi\ /\ real\ n) * cos\ (pi\ /\ real\ n)) ----NS> pi$
by (*insert NSLIMSEQ-mult* [*OF NSLIMSEQ-sin-pi NSLIMSEQ-cos-one*], *simp*)

A familiar approximation to *cos x* when *x* is small

lemma *STAR-cos-Infinitesimal-approx*:
  $x \in Infinitesimal \Longrightarrow (*f* cos)\ x \mathbin{@=} 1 - x\ \hat{}\ 2$
apply (*rule STAR-cos-Infinitesimal* [*THEN approx-trans*])
apply (*auto simp add*: *Infinitesimal-approx-minus* [*symmetric*]
    diff-minus add-assoc [*symmetric*] *numeral-2-eq-2*)
done

lemma *STAR-cos-Infinitesimal-approx2*:
  $x \in Infinitesimal \Longrightarrow (*f* cos)\ x \mathbin{@=} 1 - (x\ \hat{}\ 2)/2$
apply (*rule STAR-cos-Infinitesimal* [*THEN approx-trans*])
apply (*auto intro*: *Infinitesimal-SReal-divide*
    simp add: *Infinitesimal-approx-minus* [*symmetric*] *numeral-2-eq-2*)
done

end

46 NSCA: Non-Standard Complex Analysis

theory *NSCA*
imports *NSComplex ../Hyperreal/HTranscendental*
begin

abbreviation

SComplex :: *hcomplex set* **where**
SComplex ≡ *Standard*

definition
stc :: *hcomplex* => *hcomplex* **where**
— standard part map
stc x = (*SOME r*. *x* ∈ *HFinite* & *r*:*SComplex* & *r* @= *x*)

46.1 Closure Laws for SComplex, the Standard Complex Numbers

lemma *SComplex-minus-iff* [*simp*]: (−*x* ∈ *SComplex*) = (*x* ∈ *SComplex*)
by (*auto*, *drule Standard-minus*, *auto*)

lemma *SComplex-add-cancel*:
[| *x* + *y* ∈ *SComplex*; *y* ∈ *SComplex* |] ==> *x* ∈ *SComplex*
by (*drule* (*1*) *Standard-diff*, *simp*)

lemma *SReal-hcmod-hcomplex-of-complex* [*simp*]:
hcmod (*hcomplex-of-complex r*) ∈ *Reals*
by (*simp add*: *Reals-eq-Standard*)

lemma *SReal-hcmod-number-of* [*simp*]: *hcmod* (*number-of w* ::*hcomplex*) ∈ *Reals*
by (*simp add*: *Reals-eq-Standard*)

lemma *SReal-hcmod-SComplex*: *x* ∈ *SComplex* ==> *hcmod x* ∈ *Reals*
by (*simp add*: *Reals-eq-Standard*)

lemma *SComplex-divide-number-of*:
r ∈ *SComplex* ==> *r*/(*number-of w*::*hcomplex*) ∈ *SComplex*
by *simp*

lemma *SComplex-UNIV-complex*:
{*x*. *hcomplex-of-complex x* ∈ *SComplex*} = (*UNIV*::*complex set*)
by *simp*

lemma *SComplex-iff*: (*x* ∈ *SComplex*) = (∃ *y*. *x* = *hcomplex-of-complex y*)

by (*simp add*: *Standard-def image-def*)

lemma *hcomplex-of-complex-image*:
hcomplex-of-complex '(*UNIV*::*complex set*) = *SComplex*
by (*simp add*: *Standard-def*)

lemma *inv-hcomplex-of-complex-image*: *inv hcomplex-of-complex* '*SComplex* = *UNIV*
apply (*auto simp add*: *Standard-def image-def*)
apply (*rule inj-hcomplex-of-complex* [*THEN inv-f-f*, *THEN subst*], *blast*)
done

lemma *SComplex-hcomplex-of-complex-image*:
[| ∃ *x*. *x*: *P*; *P* ≤ *SComplex* |] ==> ∃ *Q*. *P* = *hcomplex-of-complex* ' *Q*
apply (*simp add*: *Standard-def*, *blast*)
done

lemma *SComplex-SReal-dense*:
[| *x* ∈ *SComplex*; *y* ∈ *SComplex*; *hcmod x* < *hcmod y*
|] ==> ∃ *r* ∈ *Reals*. *hcmod x*< *r* & *r* < *hcmod y*
apply (*auto intro*: *SReal-dense simp add*: *SReal-hcmod-SComplex*)
done

lemma *SComplex-hcmod-SReal*:
z ∈ *SComplex* ==> *hcmod z* ∈ *Reals*
by (*simp add*: *Reals-eq-Standard*)

46.2 The Finite Elements form a Subring

lemma *HFinite-hcmod-hcomplex-of-complex* [*simp*]:
hcmod (*hcomplex-of-complex r*) ∈ *HFinite*
by (*auto intro*!: *SReal-subset-HFinite* [*THEN subsetD*])

lemma *HFinite-hcmod-iff*: (*x* ∈ *HFinite*) = (*hcmod x* ∈ *HFinite*)
by (*simp add*: *HFinite-def*)

lemma *HFinite-bounded-hcmod*:
[|*x* ∈ *HFinite*; *y* ≤ *hcmod x*; *0* ≤ *y* |] ==> *y*: *HFinite*
by (*auto intro*: *HFinite-bounded simp add*: *HFinite-hcmod-iff*)

46.3 The Complex Infinitesimals form a Subring

lemma *hcomplex-sum-of-halves*: *x*/(*2*::*hcomplex*) + *x*/(*2*::*hcomplex*) = *x*
by *auto*

lemma *Infinitesimal-hcmod-iff*:
(*z* ∈ *Infinitesimal*) = (*hcmod z* ∈ *Infinitesimal*)
by (*simp add*: *Infinitesimal-def*)

lemma *HInfinite-hcmod-iff*: (*z* ∈ *HInfinite*) = (*hcmod z* ∈ *HInfinite*)
by (*simp add*: *HInfinite-def*)

lemma *HFinite-diff-Infinitesimal-hcmod*:
$x \in HFinite - Infinitesimal \Longrightarrow hcmod\ x \in HFinite - Infinitesimal$
by (*simp add*: *HFinite-hcmod-iff Infinitesimal-hcmod-iff*)

lemma *hcmod-less-Infinitesimal*:
$[|\ e \in Infinitesimal;\ hcmod\ x < hcmod\ e\ |] \Longrightarrow x \in Infinitesimal$
by (*auto elim*: *hrabs-less-Infinitesimal simp add*: *Infinitesimal-hcmod-iff*)

lemma *hcmod-le-Infinitesimal*:
$[|\ e \in Infinitesimal;\ hcmod\ x \leq hcmod\ e\ |] \Longrightarrow x \in Infinitesimal$
by (*auto elim*: *hrabs-le-Infinitesimal simp add*: *Infinitesimal-hcmod-iff*)

lemma *Infinitesimal-interval-hcmod*:
$[|\ e \in Infinitesimal;$
$e' \in Infinitesimal;$
$hcmod\ e' < hcmod\ x\ ;\ hcmod\ x < hcmod\ e$
$|] \Longrightarrow x \in Infinitesimal$
by (*auto intro*: *Infinitesimal-interval simp add*: *Infinitesimal-hcmod-iff*)

lemma *Infinitesimal-interval2-hcmod*:
$[|\ e \in Infinitesimal;$
$e' \in Infinitesimal;$
$hcmod\ e' \leq hcmod\ x\ ;\ hcmod\ x \leq hcmod\ e$
$|] \Longrightarrow x \in Infinitesimal$
by (*auto intro*: *Infinitesimal-interval2 simp add*: *Infinitesimal-hcmod-iff*)

46.4 The "Infinitely Close" Relation

lemma *approx-SComplex-mult-cancel-zero*:
$[|\ a \in SComplex;\ a \neq 0;\ a*x\ @=\ 0\ |] \Longrightarrow x\ @=\ 0$
apply (*drule Standard-inverse* [*THEN Standard-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult-assoc* [*symmetric*])
done

lemma *approx-mult-SComplex1*: $[|\ a \in SComplex;\ x\ @=\ 0\ |] \Longrightarrow x*a\ @=\ 0$
by (*auto dest*: *Standard-subset-HFinite* [*THEN subsetD*] *approx-mult1*)

lemma *approx-mult-SComplex2*: $[|\ a \in SComplex;\ x\ @=\ 0\ |] \Longrightarrow a*x\ @=\ 0$
by (*auto dest*: *Standard-subset-HFinite* [*THEN subsetD*] *approx-mult2*)

lemma *approx-mult-SComplex-zero-cancel-iff* [*simp*]:
$[|a \in SComplex;\ a \neq 0\ |] \Longrightarrow (a*x\ @=\ 0) = (x\ @=\ 0)$
by (*blast intro*: *approx-SComplex-mult-cancel-zero approx-mult-SComplex2*)

lemma *approx-SComplex-mult-cancel*:
$[|\ a \in SComplex;\ a \neq 0;\ a*\ w\ @=\ a*z\ |] \Longrightarrow w\ @=\ z$
apply (*drule Standard-inverse* [*THEN Standard-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult-assoc* [*symmetric*])

done

lemma *approx-SComplex-mult-cancel-iff1* [*simp*]:
[| $a \in SComplex$; $a \neq 0$ |] ==> $(a* w \ @= a*z) = (w \ @= z)$
by (*auto intro*!: *approx-mult2 Standard-subset-HFinite* [*THEN subsetD*]
intro: *approx-SComplex-mult-cancel*)

lemma *approx-hcmod-approx-zero*: $(x \ @= y) = (hcmod \ (y - x) \ @= 0)$
apply (*subst hnorm-minus-commute*)
apply (*simp add*: *approx-def Infinitesimal-hcmod-iff diff-minus*)
done

lemma *approx-approx-zero-iff*: $(x \ @= 0) = (hcmod \ x \ @= 0)$
by (*simp add*: *approx-hcmod-approx-zero*)

lemma *approx-minus-zero-cancel-iff* [*simp*]: $(-x \ @= 0) = (x \ @= 0)$
by (*simp add*: *approx-def*)

lemma *Infinitesimal-hcmod-add-diff*:
$u \ @= 0$ ==> $hcmod(x + u) - hcmod \ x \in Infinitesimal$
apply (*drule approx-approx-zero-iff* [*THEN iffD1*])
apply (*rule-tac* $e = hcmod \ u$ **and** $e' = - \ hcmod \ u$ **in** *Infinitesimal-interval2*)
apply (*auto simp add*: *mem-infmal-iff* [*symmetric*] *diff-def*)
apply (*rule-tac* $c1 = hcmod \ x$ **in** *add-le-cancel-left* [*THEN iffD1*])
apply (*auto simp add*: *diff-minus* [*symmetric*])
done

lemma *approx-hcmod-add-hcmod*: $u \ @= 0$ ==> $hcmod(x + u) \ @= hcmod \ x$
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*auto intro*: *Infinitesimal-hcmod-add-diff simp add*: *mem-infmal-iff* [*symmetric*]
diff-minus [*symmetric*])
done

46.5 Zero is the Only Infinitesimal Complex Number

lemma *Infinitesimal-less-SComplex*:
[| $x \in SComplex$; $y \in Infinitesimal$; $0 < hcmod \ x$ |] ==> $hcmod \ y < hcmod \ x$
by (*auto intro*: *Infinitesimal-less-SReal SComplex-hcmod-SReal simp add*: *Infinitesimal-hcmod-iff*)

lemma *SComplex-Int-Infinitesimal-zero*: $SComplex \ Int \ Infinitesimal = \{0\}$
by (*auto simp add*: *Standard-def Infinitesimal-hcmod-iff*)

lemma *SComplex-Infinitesimal-zero*:
[| $x \in SComplex$; $x \in Infinitesimal$ |] ==> $x = 0$
by (*cut-tac SComplex-Int-Infinitesimal-zero*, *blast*)

lemma *SComplex-HFinite-diff-Infinitesimal*:

[| $x \in SComplex$; $x \neq 0$ |] ==> $x \in HFinite - Infinitesimal$
by (*auto dest*: *SComplex-Infinitesimal-zero Standard-subset-HFinite* [*THEN subsetD*])

lemma *hcomplex-of-complex-HFinite-diff-Infinitesimal*:
$hcomplex\text{-}of\text{-}complex\ x \neq 0$
==> $hcomplex\text{-}of\text{-}complex\ x \in HFinite - Infinitesimal$
by (*rule SComplex-HFinite-diff-Infinitesimal*, *auto*)

lemma *number-of-not-Infinitesimal* [*simp*]:
$number\text{-}of\ w \neq (0::hcomplex) ==> (number\text{-}of\ w::hcomplex) \notin Infinitesimal$
by (*fast dest*: *Standard-number-of* [*THEN SComplex-Infinitesimal-zero*])

lemma *approx-SComplex-not-zero*:
[| $y \in SComplex$; $x\ @=\ y$; $y \neq 0$ |] ==> $x \neq 0$
by (*auto dest*: *SComplex-Infinitesimal-zero approx-sym* [*THEN mem-infmal-iff* [*THEN iffD2*]])

lemma *SComplex-approx-iff*:
[|$x \in SComplex$; $y \in SComplex$|] ==> $(x\ @=\ y) = (x = y)$
by (*auto simp add*: *Standard-def*)

lemma *number-of-Infinitesimal-iff* [*simp*]:
$((number\text{-}of\ w :: hcomplex) \in Infinitesimal) =$
$(number\text{-}of\ w = (0::hcomplex))$
apply (*rule iffI*)
apply (*fast dest*: *Standard-number-of* [*THEN SComplex-Infinitesimal-zero*])
apply (*simp* (*no-asm-simp*))
done

lemma *approx-unique-complex*:
[| $r \in SComplex$; $s \in SComplex$; $r\ @=\ x$; $s\ @=\ x$|] ==> $r = s$
by (*blast intro*: *SComplex-approx-iff* [*THEN iffD1*] *approx-trans2*)

46.6 Properties of *hRe*, *hIm* and *HComplex*

lemma *abs-hRe-le-hcmod*: $\bigwedge x.\ |hRe\ x| \leq hcmod\ x$
by *transfer* (*rule abs-Re-le-cmod*)

lemma *abs-hIm-le-hcmod*: $\bigwedge x.\ |hIm\ x| \leq hcmod\ x$
by *transfer* (*rule abs-Im-le-cmod*)

lemma *Infinitesimal-hRe*: $x \in Infinitesimal \Longrightarrow hRe\ x \in Infinitesimal$
apply (*rule InfinitesimalI2*, *simp*)
apply (*rule order-le-less-trans* [*OF abs-hRe-le-hcmod*])
apply (*erule* (*1*) *InfinitesimalD2*)
done

lemma *Infinitesimal-hIm*: $x \in Infinitesimal \Longrightarrow hIm\ x \in Infinitesimal$

apply (*rule InfinitesimalI2*, *simp*)
apply (*rule order-le-less-trans* [*OF abs-hIm-le-hcmod*])
apply (*erule* (*1*) *InfinitesimalD2*)
done

lemma *real-sqrt-lessI*: $[\![0 < u; x < u^2]\!] \Longrightarrow sqrt\ x < u$

by (*frule real-sqrt-less-mono*) *simp*

lemma *hypreal-sqrt-lessI*:
$\bigwedge x\ u.\ [\![0 < u; x < u^2]\!] \Longrightarrow (\ *f*\ sqrt)\ x < u$
by *transfer* (*rule real-sqrt-lessI*)

lemma *hypreal-sqrt-ge-zero*: $\bigwedge x.\ 0 \leq x \Longrightarrow 0 \leq (\ *f*\ sqrt)\ x$
by *transfer* (*rule real-sqrt-ge-zero*)

lemma *Infinitesimal-sqrt*:
$[\![x \in Infinitesimal; 0 \leq x]\!] \Longrightarrow (\ *f*\ sqrt)\ x \in Infinitesimal$
apply (*rule InfinitesimalI2*)
apply (*drule-tac r*=r^2 **in** *InfinitesimalD2*, *simp*)
apply (*simp add*: *hypreal-sqrt-ge-zero*)
apply (*rule hypreal-sqrt-lessI*, *simp-all*)
done

lemma *Infinitesimal-HComplex*:
$[\![x \in Infinitesimal; y \in Infinitesimal]\!] \Longrightarrow HComplex\ x\ y \in Infinitesimal$
apply (*rule Infinitesimal-hcmod-iff* [*THEN iffD2*])
apply (*simp add*: *hcmod-i*)
apply (*rule Infinitesimal-sqrt*)
apply (*rule Infinitesimal-add*)
apply (*erule Infinitesimal-hrealpow*, *simp*)
apply (*erule Infinitesimal-hrealpow*, *simp*)
apply (*rule add-nonneg-nonneg*)
apply (*rule zero-le-power2*)
apply (*rule zero-le-power2*)
done

lemma *hcomplex-Infinitesimal-iff*:
$(x \in Infinitesimal) = (hRe\ x \in Infinitesimal \wedge hIm\ x \in Infinitesimal)$
apply (*safe intro*!: *Infinitesimal-hRe Infinitesimal-hIm*)
apply (*drule* (*1*) *Infinitesimal-HComplex*, *simp*)
done

lemma *hRe-diff* [*simp*]: $\bigwedge x\ y.\ hRe\ (x - y) = hRe\ x - hRe\ y$
by *transfer* (*rule complex-Re-diff*)

lemma *hIm-diff* [*simp*]: $\bigwedge x\ y.\ hIm\ (x - y) = hIm\ x - hIm\ y$
by *transfer* (*rule complex-Im-diff*)

lemma *approx-hRe*: $x \approx y \Longrightarrow hRe\ x \approx hRe\ y$
unfolding *approx-def* **by** (*drule Infinitesimal-hRe*) *simp*

lemma *approx-hIm*: $x \approx y \Longrightarrow hIm\ x \approx hIm\ y$
unfolding *approx-def* **by** (*drule Infinitesimal-hIm*) *simp*

lemma *approx-HComplex*:
$[\![a \approx b;\ c \approx d]\!] \Longrightarrow HComplex\ a\ c \approx HComplex\ b\ d$
unfolding *approx-def* **by** (*simp add*: *Infinitesimal-HComplex*)

lemma *hcomplex-approx-iff*:
$(x \approx y) = (hRe\ x \approx hRe\ y \wedge hIm\ x \approx hIm\ y)$
unfolding *approx-def* **by** (*simp add*: *hcomplex-Infinitesimal-iff*)

lemma *HFinite-hRe*: $x \in HFinite \Longrightarrow hRe\ x \in HFinite$
apply (*auto simp add*: *HFinite-def SReal-def*)
apply (*rule-tac x*=*star-of r* **in** *exI*, *simp*)
apply (*erule order-le-less-trans* [*OF abs-hRe-le-hcmod*])
done

lemma *HFinite-hIm*: $x \in HFinite \Longrightarrow hIm\ x \in HFinite$
apply (*auto simp add*: *HFinite-def SReal-def*)
apply (*rule-tac x*=*star-of r* **in** *exI*, *simp*)
apply (*erule order-le-less-trans* [*OF abs-hIm-le-hcmod*])
done

lemma *HFinite-HComplex*:
$[\![x \in HFinite;\ y \in HFinite]\!] \Longrightarrow HComplex\ x\ y \in HFinite$
apply (*subgoal-tac HComplex x 0 + HComplex 0 y* $\in$ *HFinite*, *simp*)
apply (*rule HFinite-add*)
apply (*simp add*: *HFinite-hcmod-iff hcmod-i*)
apply (*simp add*: *HFinite-hcmod-iff hcmod-i*)
done

lemma *hcomplex-HFinite-iff*:
$(x \in HFinite) = (hRe\ x \in HFinite \wedge hIm\ x \in HFinite)$
apply (*safe intro*!: *HFinite-hRe HFinite-hIm*)
apply (*drule* (*1*) *HFinite-HComplex*, *simp*)
done

lemma *hcomplex-HInfinite-iff*:
$(x \in HInfinite) = (hRe\ x \in HInfinite \vee hIm\ x \in HInfinite)$
by (*simp add*: *HInfinite-HFinite-iff hcomplex-HFinite-iff*)

lemma *hcomplex-of-hypreal-approx-iff* [*simp*]:
$(hcomplex\text{-}of\text{-}hypreal\ x\ @{=}\ hcomplex\text{-}of\text{-}hypreal\ z) = (x\ @{=}\ z)$
by (*simp add*: *hcomplex-approx-iff*)

lemma *Standard-HComplex*:

$[\![x \in Standard;\ y \in Standard]\!] \Longrightarrow HComplex\ x\ y \in Standard$
by (*simp add*: *HComplex-def*)

lemma *stc-part-Ex*: *x*:*HFinite* ==> $\exists t \in SComplex.\ x\ @=\ t$
apply (*simp add*: *hcomplex-HFinite-iff hcomplex-approx-iff*)
apply (*rule-tac x*=*HComplex* (*st* (*hRe x*)) (*st* (*hIm x*)) **in** *bexI*)
apply (*simp add*: *st-approx-self* [*THEN approx-sym*])
apply (*simp add*: *Standard-HComplex st-SReal* [*unfolded Reals-eq-Standard*])
done

lemma *stc-part-Ex1*: *x*:*HFinite* ==> *EX*! *t*. $t \in SComplex\ \&\ \ x\ @=\ t$
apply (*drule stc-part-Ex*, *safe*)
apply (*drule-tac* [*2*] *approx-sym*, *drule-tac* [*2*] *approx-sym*, *drule-tac* [*2*] *approx-sym*)
apply (*auto intro*!: *approx-unique-complex*)
done

lemmas *hcomplex-of-complex-approx-inverse* =
hcomplex-of-complex-HFinite-diff-Infinitesimal [*THEN* [*2*] *approx-inverse*]

46.7 Theorems About Monads

lemma *monad-zero-hcmod-iff*: $(x \in monad\ 0) = (hcmod\ x{:}monad\ 0)$
by (*simp add*: *Infinitesimal-monad-zero-iff* [*symmetric*] *Infinitesimal-hcmod-iff*)

46.8 Theorems About Standard Part

lemma *stc-approx-self*: $x \in HFinite$ ==> $stc\ x\ @=\ x$
apply (*simp add*: *stc-def*)
apply (*frule stc-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *stc-SComplex*: $x \in HFinite$ ==> $stc\ x \in SComplex$
apply (*simp add*: *stc-def*)
apply (*frule stc-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *stc-HFinite*: $x \in HFinite$ ==> $stc\ x \in HFinite$
by (*erule stc-SComplex* [*THEN Standard-subset-HFinite* [*THEN subsetD*]])

lemma *stc-unique*: $[\![y \in SComplex;\ y \approx x]\!] \Longrightarrow stc\ x = y$
apply (*frule Standard-subset-HFinite* [*THEN subsetD*])
apply (*drule* (*1*) *approx-HFinite*)
apply (*unfold stc-def*)
apply (*rule some-equality*)
apply (*auto intro*: *approx-unique-complex*)

done

lemma *stc-SComplex-eq* [*simp*]: $x \in SComplex \Longrightarrow stc\ x = x$
apply (*erule stc-unique*)
apply (*rule approx-refl*)
done

lemma *stc-hcomplex-of-complex*:
$stc\ (hcomplex\text{-}of\text{-}complex\ x) = hcomplex\text{-}of\text{-}complex\ x$
by *auto*

lemma *stc-eq-approx*:
$[|\ x \in HFinite;\ y \in HFinite;\ stc\ x = stc\ y\ |] \Longrightarrow x\ @=\ y$
by (*auto dest*!: *stc-approx-self elim*!: *approx-trans3*)

lemma *approx-stc-eq*:
$[|\ x \in HFinite;\ y \in HFinite;\ x\ @=\ y\ |] \Longrightarrow stc\ x = stc\ y$
by (*blast intro*: *approx-trans approx-trans2 SComplex-approx-iff* [*THEN iffD1*]
dest: *stc-approx-self stc-SComplex*)

lemma *stc-eq-approx-iff*:
$[|\ x \in HFinite;\ y \in HFinite|] \Longrightarrow (x\ @=\ y) = (stc\ x = stc\ y)$
by (*blast intro*: *approx-stc-eq stc-eq-approx*)

lemma *stc-Infinitesimal-add-SComplex*:
$[|\ x \in SComplex;\ e \in Infinitesimal\ |] \Longrightarrow stc(x + e) = x$
apply (*erule stc-unique*)
apply (*erule Infinitesimal-add-approx-self*)
done

lemma *stc-Infinitesimal-add-SComplex2*:
$[|\ x \in SComplex;\ e \in Infinitesimal\ |] \Longrightarrow stc(e + x) = x$
apply (*erule stc-unique*)
apply (*erule Infinitesimal-add-approx-self2*)
done

lemma *HFinite-stc-Infinitesimal-add*:
$x \in HFinite \Longrightarrow \exists\, e \in Infinitesimal.\ x = stc(x) + e$
by (*blast dest*!: *stc-approx-self* [*THEN approx-sym*] *bex-Infinitesimal-iff2* [*THEN iffD2*])

lemma *stc-add*:
$[|\ x \in HFinite;\ y \in HFinite\ |] \Longrightarrow stc\ (x + y) = stc(x) + stc(y)$
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-add*)

lemma *stc-number-of* [*simp*]: $stc\ (number\text{-}of\ w) = number\text{-}of\ w$
by (*rule Standard-number-of* [*THEN stc-SComplex-eq*])

lemma *stc-zero* [*simp*]: $stc\ 0 = 0$

by *simp*

lemma *stc-one* [*simp*]: *stc 1 = 1*
by *simp*

lemma *stc-minus*: *y ∈ HFinite ==> stc(−y) = −stc(y)*
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-minus*)

lemma *stc-diff*:
[| *x ∈ HFinite*; *y ∈ HFinite* |] *==> stc (x−y) = stc(x) − stc(y)*
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-diff*)

lemma *stc-mult*:
[| *x ∈ HFinite*; *y ∈ HFinite* |]
*==> stc (x * y) = stc(x) * stc(y)*
by (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-mult-HFinite*)

lemma *stc-Infinitesimal*: *x ∈ Infinitesimal ==> stc x = 0*
by (*simp add*: *stc-unique mem-infmal-iff*)

lemma *stc-not-Infinitesimal*: *stc(x) ≠ 0 ==> x ∉ Infinitesimal*
by (*fast intro*: *stc-Infinitesimal*)

lemma *stc-inverse*:
[| *x ∈ HFinite*; *stc x ≠ 0* |]
==> stc(inverse x) = inverse (stc x)
apply (*drule stc-not-Infinitesimal*)
apply (*simp add*: *stc-unique stc-SComplex stc-approx-self approx-inverse*)
done

lemma *stc-divide* [*simp*]:
[| *x ∈ HFinite*; *y ∈ HFinite*; *stc y ≠ 0* |]
==> stc(x/y) = (stc x) / (stc y)
by (*simp add*: *divide-inverse stc-mult stc-not-Infinitesimal HFinite-inverse stc-inverse*)

lemma *stc-idempotent* [*simp*]: *x ∈ HFinite ==> stc(stc(x)) = stc(x)*
by (*blast intro*: *stc-HFinite stc-approx-self approx-stc-eq*)

lemma *HFinite-HFinite-hcomplex-of-hypreal*:
z ∈ HFinite ==> hcomplex-of-hypreal z ∈ HFinite
by (*simp add*: *hcomplex-HFinite-iff*)

lemma *SComplex-SReal-hcomplex-of-hypreal*:
x ∈ Reals ==> hcomplex-of-hypreal x ∈ SComplex
apply (*rule Standard-of-hypreal*)
apply (*simp add*: *Reals-eq-Standard*)
done

lemma *stc-hcomplex-of-hypreal*:

$z \in HFinite \Longrightarrow stc(hcomplex\text{-}of\text{-}hypreal\ z) = hcomplex\text{-}of\text{-}hypreal\ (st\ z)$
apply (*rule stc-unique*)
apply (*rule SComplex-SReal-hcomplex-of-hypreal*)
apply (*erule st-SReal*)
apply (*simp add*: *hcomplex-of-hypreal-approx-iff st-approx-self*)
done

lemma *Infinitesimal-hcnj-iff* [*simp*]:
$(hcnj\ z \in Infinitesimal) = (z \in Infinitesimal)$
by (*simp add*: *Infinitesimal-hcmod-iff*)

lemma *Infinitesimal-hcomplex-of-hypreal-epsilon* [*simp*]:
$hcomplex\text{-}of\text{-}hypreal\ epsilon \in Infinitesimal$
by (*simp add*: *Infinitesimal-hcmod-iff*)

end

47 CStar: Star-transforms in NSA, Extending Sets of Complex Numbers and Complex Functions

theory *CStar*
imports *NSCA*
begin

47.1 Properties of the *-Transform Applied to Sets of Reals

lemma *STARC-hcomplex-of-complex-Int*:
$*s*\ X\ Int\ SComplex = hcomplex\text{-}of\text{-}complex\ `\ X$
by (*auto simp add*: *Standard-def*)

lemma *lemma-not-hcomplexA*:
$x \notin hcomplex\text{-}of\text{-}complex\ `\ A \Longrightarrow \forall y \in A.\ x \neq hcomplex\text{-}of\text{-}complex\ y$
by *auto*

47.2 Theorems about Nonstandard Extensions of Functions

lemma *starfunC-hcpow*: $!!Z.\ (*f*\ (\%z.\ z\ \hat{}\ n))\ Z = Z\ pow\ hypnat\text{-}of\text{-}nat\ n$
by *transfer* (*rule refl*)

lemma *starfunCR-cmod*: $*f*\ cmod = hcmod$
by *transfer* (*rule refl*)

47.3 Internal Functions - Some Redundancy With *f* Now

lemma *starfun-Re*: $(*f*\ (\lambda x.\ Re\ (f\ x))) = (\lambda x.\ hRe\ ((*f*\ f)\ x))$
by *transfer* (*rule refl*)

lemma *starfun-Im*: $(*f* (\lambda x.\ Im\ (f\ x))) = (\lambda x.\ hIm\ ((\ *f*\ f)\ x))$
by *transfer* (*rule refl*)

lemma *starfunC-eq-Re-Im-iff*:
$((\ *f*\ f)\ x = z) = (((\ *f*\ (\%x.\ Re(f\ x)))\ x = hRe\ (z))\ \&$
$((\ *f*\ (\%x.\ Im(f\ x)))\ x = hIm\ (z)))$
by (*simp add*: *hcomplex-hRe-hIm-cancel-iff starfun-Re starfun-Im*)

lemma *starfunC-approx-Re-Im-iff*:
$((\ *f*\ f)\ x\ @=\ z) = (((\ *f*\ (\%x.\ Re(f\ x)))\ x\ @=\ hRe\ (z))\ \&$
$((\ *f*\ (\%x.\ Im(f\ x)))\ x\ @=\ hIm\ (z)))$
by (*simp add*: *hcomplex-approx-iff starfun-Re starfun-Im*)

end

48 CLim: Limits, Continuity and Differentiation for Complex Functions

theory *CLim*
imports *CStar*
begin

declare *hypreal-epsilon-not-zero* [*simp*]

lemma *lemma-complex-mult-inverse-squared* [*simp*]:
$x \neq (0::complex) \Longrightarrow (x\ *\ inverse(x)\ \hat{}\ 2) = inverse\ x$
by (*simp add*: *numeral-2-eq-2*)

Changing the quantified variable. Install earlier?

lemma *all-shift*: $(\forall x::'a::comm\text{-}ring\text{-}1.\ P\ x) = (\forall x.\ P\ (x-a))$
apply *auto*
apply (*drule-tac x=x+a* **in** *spec*)
apply (*simp add*: *diff-minus add-assoc*)
done

lemma *complex-add-minus-iff* [*simp*]: $(x + -\ a = (0::complex)) = (x=a)$
by (*simp add*: *diff-eq-eq diff-minus* [*symmetric*])

lemma *complex-add-eq-0-iff* [*iff*]: $(x+y = (0::complex)) = (y = -x)$
apply *auto*
apply (*drule sym* [*THEN diff-eq-eq* [*THEN iffD2*]], *auto*)
done

48.1 Limit of Complex to Complex Function

lemma *NSLIM-Re*: $f -- a --NS> L ==> (\%x.\ Re(f\,x)) -- a --NS> Re(L)$
by (*simp add*: *NSLIM-def starfunC-approx-Re-Im-iff*
hRe-hcomplex-of-complex)

lemma *NSLIM-Im*: $f -- a --NS> L ==> (\%x.\ Im(f\,x)) -- a --NS> Im(L)$
by (*simp add*: *NSLIM-def starfunC-approx-Re-Im-iff*
hIm-hcomplex-of-complex)

lemma *LIM-Re*: $f -- a --> L ==> (\%x.\ Re(f\,x)) -- a --> Re(L)$
by (*simp add*: *LIM-NSLIM-iff NSLIM-Re*)

lemma *LIM-Im*: $f -- a --> L ==> (\%x.\ Im(f\,x)) -- a --> Im(L)$
by (*simp add*: *LIM-NSLIM-iff NSLIM-Im*)

lemma *LIM-cnj*: $f -- a --> L ==> (\%x.\ cnj\ (f\,x)) -- a --> cnj\ L$
by (*simp add*: *LIM-def complex-cnj-diff* [*symmetric*])

lemma *LIM-cnj-iff*: $((\%x.\ cnj\ (f\,x)) -- a --> cnj\ L) = (f -- a --> L)$
by (*simp add*: *LIM-def complex-cnj-diff* [*symmetric*])

lemma *starfun-norm*: $(\ *f*\ (\lambda x.\ norm\ (f\,x))) = (\lambda x.\ hnorm\ ((\ *f*\ f)\ x))$
by *transfer* (*rule refl*)

lemma *star-of-Re* [*simp*]: *star-of* $(Re\ x) = hRe\ (star\text{-}of\ x)$
by *transfer* (*rule refl*)

lemma *star-of-Im* [*simp*]: *star-of* $(Im\ x) = hIm\ (star\text{-}of\ x)$
by *transfer* (*rule refl*)

lemma *NSCLIM-NSCRLIM-iff*:
$(f -- x --NS> L) = ((\%y.\ cmod(f\ y - L)) -- x --NS> 0)$
by (*simp add*: *NSLIM-def starfun-norm*
approx-approx-zero-iff [*symmetric*] *approx-minus-iff* [*symmetric*])

lemma *CLIM-CRLIM-iff*: $(f -- x --> L) = ((\%y.\ cmod(f\ y - L)) -- x --> 0)$
by (*simp add*: *LIM-def*)

lemma *NSCLIM-NSCRLIM-iff2*:
$(f -- x --NS> L) = ((\%y.\ cmod(f\ y - L)) -- x --NS> 0)$
by (*simp add*: *LIM-NSLIM-iff* [*symmetric*] *CLIM-CRLIM-iff*)

lemma *NSLIM-NSCRLIM-Re-Im-iff*:
$(f -- a --NS> L) = ((\%x.\ Re(f\,x)) -- a --NS> Re(L)\ \&$
$(\%x.\ Im(f\,x)) -- a --NS> Im(L))$

apply (*auto intro*: *NSLIM-Re NSLIM-Im*)
apply (*auto simp add*: *NSLIM-def starfun-Re starfun-Im*)
apply (*auto dest*!: *spec*)
apply (*simp add*: *hcomplex-approx-iff*)
done

lemma *LIM-CRLIM-Re-Im-iff*:
(*f* −− *a* −−> *L*) = ((%*x*. *Re*(*f x*)) −− *a* −−> *Re*(*L*) &
(%*x*. *Im*(*f x*)) −− *a* −−> *Im*(*L*))
by (*simp add*: *LIM-NSLIM-iff NSLIM-NSCRLIM-Re-Im-iff*)

48.2 Continuity

lemma *NSLIM-isContc-iff*:
(*f* −− *a* −−*NS*> *f a*) = ((%*h*. *f*(*a* + *h*)) −− *0* −−*NS*> *f a*)
by (*rule NSLIM-h-iff*)

48.3 Functions from Complex to Reals

lemma *isNSContCR-cmod* [*simp*]: *isNSCont cmod* (*a*)
by (*auto intro*: *approx-hnorm*
simp add: *starfunCR-cmod hcmod-hcomplex-of-complex* [*symmetric*]
isNSCont-def)

lemma *isContCR-cmod* [*simp*]: *isCont cmod* (*a*)
by (*simp add*: *isNSCont-isCont-iff* [*symmetric*])

lemma *isCont-Re*: *isCont f a* ==> *isCont* (%*x*. *Re* (*f x*)) *a*
by (*simp add*: *isCont-def LIM-Re*)

lemma *isCont-Im*: *isCont f a* ==> *isCont* (%*x*. *Im* (*f x*)) *a*
by (*simp add*: *isCont-def LIM-Im*)

48.4 Differentiation of Natural Number Powers

lemma *CDERIV-pow* [*simp*]:
DERIV (%*x*. *x* ^ *n*) *x* :> (*complex-of-real* (*real n*)) ∗ (*x* ^ (*n* − *Suc 0*))
apply (*induct-tac n*)
apply (*drule-tac* [*2*] *DERIV-ident* [*THEN DERIV-mult*])
apply (*auto simp add*: *left-distrib real-of-nat-Suc*)
apply (*case-tac n*)
apply (*auto simp add*: *mult-ac add-commute*)
done

Nonstandard version

lemma *NSCDERIV-pow*:
NSDERIV (%*x*. *x* ^ *n*) *x* :> *complex-of-real* (*real n*) ∗ (*x* ^ (*n* − *1*))
by (*simp add*: *NSDERIV-DERIV-iff*)

Can't relax the premise $x \neq (0::'a)$: it isn't continuous at zero

lemma *NSCDERIV-inverse*:
$(x::complex) \neq 0 \Longrightarrow NSDERIV\ (\%x.\ inverse(x))\ x :> (-\ (inverse\ x\ \hat{}\ 2))$
unfolding *numeral-2-eq-2*
by (*rule NSDERIV-inverse*)

lemma *CDERIV-inverse*:
$(x::complex) \neq 0 \Longrightarrow DERIV\ (\%x.\ inverse(x))\ x :> (-(inverse\ x\ \hat{}\ 2))$
unfolding *numeral-2-eq-2*
by (*rule DERIV-inverse*)

48.5 Derivative of Reciprocals (Function *inverse*)

lemma *CDERIV-inverse-fun*:
$[|\ DERIV\ f\ x :> d;\ f(x) \neq (0::complex)\ |]$
$\Longrightarrow DERIV\ (\%x.\ inverse(f\ x))\ x :> (-\ (d * inverse(f(x)\ \hat{}\ 2)))$
unfolding *numeral-2-eq-2*
by (*rule DERIV-inverse-fun*)

lemma *NSCDERIV-inverse-fun*:
$[|\ NSDERIV\ f\ x :> d;\ f(x) \neq (0::complex)\ |]$
$\Longrightarrow NSDERIV\ (\%x.\ inverse(f\ x))\ x :> (-\ (d * inverse(f(x)\ \hat{}\ 2)))$
unfolding *numeral-2-eq-2*
by (*rule NSDERIV-inverse-fun*)

48.6 Derivative of Quotient

lemma *CDERIV-quotient*:
$[|\ DERIV\ f\ x :> d;\ DERIV\ g\ x :> e;\ g(x) \neq (0::complex)\ |]$
$\Longrightarrow DERIV\ (\%y.\ f(y)\ /\ (g\ y))\ x :> (d*g(x) - (e*f(x)))\ /\ (g(x)\ \hat{}\ 2)$
unfolding *numeral-2-eq-2*
by (*rule DERIV-quotient*)

lemma *NSCDERIV-quotient*:
$[|\ NSDERIV\ f\ x :> d;\ NSDERIV\ g\ x :> e;\ g(x) \neq (0::complex)\ |]$
$\Longrightarrow NSDERIV\ (\%y.\ f(y)\ /\ (g\ y))\ x :> (d*g(x) - (e*f(x)))\ /\ (g(x)\ \hat{}\ 2)$
unfolding *numeral-2-eq-2*
by (*rule NSDERIV-quotient*)

48.7 Caratheodory Formulation of Derivative at a Point: Standard Proof

lemma *CARAT-CDERIVD*:
$(\forall z.\ f\ z - f\ x = g\ z * (z - x))\ \&\ isNSCont\ g\ x\ \&\ g\ x = l$
$\Longrightarrow NSDERIV\ f\ x :> l$
by *clarify* (*rule CARAT-DERIVD*)

end

49 Ln: Properties of ln

theory *Ln*
imports *Transcendental*
begin

lemma *exp-first-two-terms*: *exp x = 1 + x + suminf (%n.*
*inverse(real (fact (n+2))) * (x ^ (n+2)))*
proof −
have *exp x = suminf (%n. inverse(real (fact n)) * (x ^ n))*
by (*simp add*: *exp-def*)
also from *summable-exp* **have** *... = (SUM n : {0..<2}.*
*inverse(real (fact n)) * (x ^ n)) + suminf (%n.*
*inverse(real (fact (n+2))) * (x ^ (n+2))) (***is** *- = ?a + -*)
by (*rule suminf-split-initial-segment*)
also have *?a = 1 + x*
by (*simp add*: *numerals*)
finally show *?thesis* **.**
qed

lemma *exp-tail-after-first-two-terms-summable*:
*summable (%n. inverse(real (fact (n+2))) * (x ^ (n+2)))*
proof −
note *summable-exp*
thus *?thesis*
by (*frule summable-ignore-initial-segment*)
qed

lemma *aux1*: **assumes** *a*: *0 <= x* **and** *b*: *x <= 1*
shows *inverse (real (fact (n + 2))) * x ^ (n + 2) <= (x^2/2) * ((1/2)^n)*
proof (*induct n*)
show *inverse (real (fact (0 + 2))) * x ^ (0 + 2) <=*
*x ^ 2 / 2 * (1 / 2) ^ 0*
by (*simp add*: *real-of-nat-Suc power2-eq-square*)
next
fix *n*
assume *c*: *inverse (real (fact (n + 2))) * x ^ (n + 2)*
*<= x ^ 2 / 2 * (1 / 2) ^ n*
show *inverse (real (fact (Suc n + 2))) * x ^ (Suc n + 2)*
*<= x ^ 2 / 2 * (1 / 2) ^ Suc n*
proof −
have *inverse(real (fact (Suc n + 2))) <=*
*(1 / 2) *inverse (real (fact (n+2)))*
proof −
have *Suc n + 2 = Suc (n + 2)* **by** *simp*
then have *fact (Suc n + 2) = Suc (n + 2) * fact (n + 2)*
by *simp*
then have *real(fact (Suc n + 2)) = real(Suc (n + 2) * fact (n + 2))*
apply (*rule subst*)

apply (*rule refl*)
done
also have ... = *real*(*Suc* (*n* + *2*)) * *real*(*fact* (*n* + *2*))
by (*rule real-of-nat-mult*)
finally have *real* (*fact* (*Suc n* + *2*)) =
real (*Suc* (*n* + *2*)) * *real* (*fact* (*n* + *2*)) **.**
then have *inverse*(*real* (*fact* (*Suc n* + *2*))) =
inverse(*real* (*Suc* (*n* + *2*))) * *inverse*(*real* (*fact* (*n* + *2*)))
apply (*rule ssubst*)
apply (*rule inverse-mult-distrib*)
done
also have ... <= (*1*/*2*) * *inverse*(*real* (*fact* (*n* + *2*)))
apply (*rule mult-right-mono*)
apply (*subst inverse-eq-divide*)
apply *simp*
apply (*rule inv-real-of-nat-fact-ge-zero*)
done
finally show *?thesis* **.**
qed
moreover have *x* ^ (*Suc n* + *2*) <= *x* ^ (*n* + *2*)
apply (*simp add*: *mult-compare-simps*)
apply (*simp add*: *prems*)
apply (*subgoal-tac 0* <= *x* * (*x* * *x*^*n*))
apply *force*
apply (*rule mult-nonneg-nonneg*, *rule a*)+
apply (*rule zero-le-power*, *rule a*)
done
ultimately have *inverse* (*real* (*fact* (*Suc n* + *2*))) * *x* ^ (*Suc n* + *2*) <=
(*1* / *2* * *inverse* (*real* (*fact* (*n* + *2*)))) * *x* ^ (*n* + *2*)
apply (*rule mult-mono*)
apply (*rule mult-nonneg-nonneg*)
apply *simp*
apply (*subst inverse-nonnegative-iff-nonnegative*)
apply (*rule real-of-nat-fact-ge-zero*)
apply (*rule zero-le-power*)
apply (*rule a*)
done
also have ... = *1* / *2* * (*inverse* (*real* (*fact* (*n* + *2*))) * *x* ^ (*n* + *2*))
by *simp*
also have ... <= *1* / *2* * (*x* ^ *2* / *2* * (*1* / *2*) ^ *n*)
apply (*rule mult-left-mono*)
apply (*rule prems*)
apply *simp*
done
also have ... = *x* ^ *2* / *2* * (*1* / *2* * (*1* / *2*) ^ *n*)
by *auto*
also have (*1*::*real*) / *2* * (*1* / *2*) ^ *n* = (*1* / *2*) ^ (*Suc n*)
by (*rule realpow-Suc* [*THEN sym*])
finally show *?thesis* **.**

```
  qed
qed

lemma aux2: (%n. (x::real) ^ 2 / 2 * (1 / 2) ^ n) sums x^2
proof -
  have (%n. (1 / 2::real)^n) sums (1 / (1 - (1/2)))
    apply (rule geometric-sums)
    by (simp add: abs-less-iff)
  also have (1::real) / (1 - 1/2) = 2
    by simp
  finally have (%n. (1 / 2::real)^n) sums 2 .
  then have (%n. x ^ 2 / 2 * (1 / 2) ^ n) sums (x^2 / 2 * 2)
    by (rule sums-mult)
  also have x^2 / 2 * 2 = x^2
    by simp
  finally show ?thesis .
qed

lemma exp-bound: 0 <= (x::real) ==> x <= 1 ==> exp x <= 1 + x + x^2
proof -
  assume a: 0 <= x
  assume b: x <= 1
  have c: exp x = 1 + x + suminf (%n. inverse(real (fact (n+2))) *
      (x ^ (n+2)))
    by (rule exp-first-two-terms)
 moreover have suminf (%n. inverse(real (fact (n+2))) * (x ^ (n+2))) <= x^2
  proof -
    have suminf (%n. inverse(real (fact (n+2))) * (x ^ (n+2))) <=
        suminf (%n. (x^2/2) * ((1/2)^n))
      apply (rule summable-le)
      apply (auto simp only: aux1 prems)
      apply (rule exp-tail-after-first-two-terms-summable)
      by (rule sums-summable, rule aux2)
    also have ... = x^2
      by (rule sums-unique [THEN sym], rule aux2)
    finally show ?thesis .
  qed
  ultimately show ?thesis
    by auto
qed

lemma aux4: 0 <= (x::real) ==> x <= 1 ==> exp (x - x^2) <= 1 + x
proof -
  assume a: 0 <= x and b: x <= 1
  have exp (x - x^2) = exp x / exp (x^2)
    by (rule exp-diff)
  also have ... <= (1 + x + x^2) / exp (x ^2)
    apply (rule divide-right-mono)
    apply (rule exp-bound)
```

```
    apply (rule a, rule b)
    apply simp
    done
  also have ... <= (1 + x + x^2) / (1 + x^2)
    apply (rule divide-left-mono)
    apply (auto simp add: exp-ge-add-one-self-aux)
    apply (rule add-nonneg-nonneg)
    apply (insert prems, auto)
    apply (rule mult-pos-pos)
    apply auto
    apply (rule add-pos-nonneg)
    apply auto
    done
  also from a have ... <= 1 + x
    by(simp add:field-simps zero-compare-simps)
  finally show ?thesis .
qed

lemma ln-one-plus-pos-lower-bound: 0 <= x ==> x <= 1 ==>
    x - x^2 <= ln (1 + x)
proof -
  assume a: 0 <= x and b: x <= 1
  then have exp (x - x^2) <= 1 + x
    by (rule aux4)
  also have ... = exp (ln (1 + x))
  proof -
    from a have 0 < 1 + x by auto
    thus ?thesis
      by (auto simp only: exp-ln-iff [THEN sym])
  qed
  finally have exp (x - x ^ 2) <= exp (ln (1 + x)) .
  thus ?thesis by (auto simp only: exp-le-cancel-iff)
qed

lemma ln-one-minus-pos-upper-bound: 0 <= x ==> x < 1 ==> ln (1 - x) <=
- x
proof -
  assume a: 0 <= (x::real) and b: x < 1
  have (1 - x) * (1 + x + x^2) = (1 - x^3)
    by (simp add: ring-simps power2-eq-square power3-eq-cube)
  also have ... <= 1
    by (auto simp add: a)
  finally have (1 - x) * (1 + x + x ^ 2) <= 1 .
  moreover have 0 < 1 + x + x^2
    apply (rule add-pos-nonneg)
    apply (insert a, auto)
    done
  ultimately have 1 - x <= 1 / (1 + x + x^2)
    by (elim mult-imp-le-div-pos)
```

```
  also have ... <= 1 / exp x
    apply (rule divide-left-mono)
    apply (rule exp-bound, rule a)
    apply (insert prems, auto)
    apply (rule mult-pos-pos)
    apply (rule add-pos-nonneg)
    apply auto
    done
  also have ... = exp (-x)
    by (auto simp add: exp-minus real-divide-def)
  finally have 1 - x <= exp (- x) .
  also have 1 - x = exp (ln (1 - x))
  proof -
    have 0 < 1 - x
      by (insert b, auto)
    thus ?thesis
      by (auto simp only: exp-ln-iff [THEN sym])
  qed
  finally have exp (ln (1 - x)) <= exp (- x) .
  thus ?thesis by (auto simp only: exp-le-cancel-iff)
qed

lemma aux5: x < 1 ==> ln(1 - x) = - ln(1 + x / (1 - x))
proof -
  assume a: x < 1
  have ln(1 - x) = - ln(1 / (1 - x))
  proof -
    have ln(1 - x) = - (- ln (1 - x))
      by auto
    also have - ln(1 - x) = ln 1 - ln(1 - x)
      by simp
    also have ... = ln(1 / (1 - x))
      apply (rule ln-div [THEN sym])
      by (insert a, auto)
    finally show ?thesis .
  qed
  also have  1 / (1 - x) = 1 + x / (1 - x) using a by(simp add:field-simps)
  finally show ?thesis .
qed

lemma ln-one-minus-pos-lower-bound: 0 <= x ==> x <= (1 / 2) ==>
    - x - 2 * x^2 <= ln (1 - x)
proof -
  assume a: 0 <= x and b: x <= (1 / 2)
  from b have c: x < 1
    by auto
  then have ln (1 - x) = - ln (1 + x / (1 - x))
    by (rule aux5)
  also have - (x / (1 - x)) <= ...
```

```
  proof −
    have ln (1 + x / (1 − x)) <= x / (1 − x)
      apply (rule ln-add-one-self-le-self)
      apply (rule divide-nonneg-pos)
      by (insert a c, auto)
    thus ?thesis
      by auto
  qed
  also have − (x / (1 − x)) = −x / (1 − x)
    by auto
  finally have d: − x / (1 − x) <= ln (1 − x) .
  have 0 < 1 − x using prems by simp
  hence e: −x − 2 ∗ x^2 <= − x / (1 − x)
    using mult-right-le-one-le[of x∗x 2∗x] prems
    by(simp add:field-simps power2-eq-square)
  from e d show − x − 2 ∗ x^2 <= ln (1 − x)
    by (rule order-trans)
qed

lemma exp-ge-add-one-self [simp]: 1 + (x::real) <= exp x
  apply (case-tac 0 <= x)
  apply (erule exp-ge-add-one-self-aux)
  apply (case-tac x <= −1)
  apply (subgoal-tac 1 + x <= 0)
  apply (erule order-trans)
  apply simp
  apply simp
  apply (subgoal-tac 1 + x = exp(ln (1 + x)))
  apply (erule ssubst)
  apply (subst exp-le-cancel-iff)
  apply (subgoal-tac ln (1 − (− x)) <= − (− x))
  apply simp
  apply (rule ln-one-minus-pos-upper-bound)
  apply auto
done

lemma ln-add-one-self-le-self2: −1 < x ==> ln(1 + x) <= x
  apply (subgoal-tac x = ln (exp x))
  apply (erule ssubst)back
  apply (subst ln-le-cancel-iff)
  apply auto
done

lemma abs-ln-one-plus-x-minus-x-bound-nonneg:
   0 <= x ==> x <= 1 ==> abs(ln (1 + x) − x) <= x^2
proof −
  assume x: 0 <= x
  assume x <= 1
  from x have ln (1 + x) <= x
```

```
    by (rule ln-add-one-self-le-self)
  then have ln (1 + x) - x <= 0
    by simp
  then have abs(ln(1 + x) - x) = - (ln(1 + x) - x)
    by (rule abs-of-nonpos)
  also have ... = x - ln (1 + x)
    by simp
  also have ... <= x^2
  proof -
    from prems have x - x^2 <= ln (1 + x)
      by (intro ln-one-plus-pos-lower-bound)
    thus ?thesis
      by simp
  qed
  finally show ?thesis .
qed

lemma abs-ln-one-plus-x-minus-x-bound-nonpos:
    -(1 / 2) <= x ==> x <= 0 ==> abs(ln (1 + x) - x) <= 2 * x^2
proof -
  assume -(1 / 2) <= x
  assume x <= 0
  have abs(ln (1 + x) - x) = x - ln(1 - (-x))
    apply (subst abs-of-nonpos)
    apply simp
    apply (rule ln-add-one-self-le-self2)
    apply (insert prems, auto)
    done
  also have ... <= 2 * x^2
    apply (subgoal-tac - (-x) - 2 * (-x)^2 <= ln (1 - (-x)))
    apply (simp add: compare-rls)
    apply (rule ln-one-minus-pos-lower-bound)
    apply (insert prems, auto)
    done
  finally show ?thesis .
qed

lemma abs-ln-one-plus-x-minus-x-bound:
    abs x <= 1 / 2 ==> abs(ln (1 + x) - x) <= 2 * x^2
  apply (case-tac 0 <= x)
  apply (rule order-trans)
  apply (rule abs-ln-one-plus-x-minus-x-bound-nonneg)
  apply auto
  apply (rule abs-ln-one-plus-x-minus-x-bound-nonpos)
  apply auto
done

lemma DERIV-ln: 0 < x ==> DERIV ln x :> 1 / x
  apply (unfold deriv-def, unfold LIM-def, clarsimp)
```

apply (*rule exI*)
apply (*rule conjI*)
prefer *2*
apply *clarsimp*
apply (*subgoal-tac* (*ln* (*x* + *xa*) − *ln x*) / *xa* − (*1* / *x*) =
(*ln* (*1* + *xa* / *x*) − *xa* / *x*) / *xa*)
apply (*erule ssubst*)
apply (*subst abs-divide*)
apply (*rule mult-imp-div-pos-less*)
apply *force*
apply (*rule order-le-less-trans*)
apply (*rule abs-ln-one-plus-x-minus-x-bound*)
apply (*subst abs-divide*)
apply (*subst abs-of-pos*, *assumption*)
apply (*erule mult-imp-div-pos-le*)
apply (*subgoal-tac abs xa* < *min* (*x* / *2*) (*r* ∗ *x*^*2* / *2*))
apply *force*
apply *assumption*
apply (*simp add*: *power2-eq-square mult-compare-simps*)
apply (*rule mult-imp-div-pos-less*)
apply (*rule mult-pos-pos*, *assumption*, *assumption*)
apply (*subgoal-tac xa* ∗ *xa* = *abs xa* ∗ *abs xa*)
apply (*erule ssubst*)
apply (*subgoal-tac abs xa* ∗ (*abs xa* ∗ *2*) < *abs xa* ∗ (*r* ∗ (*x* ∗ *x*)))
apply (*simp only*: *mult-ac*)
apply (*rule mult-strict-left-mono*)
apply (*erule conjE*, *assumption*)
apply *force*
apply *simp*
apply (*subst ln-div* [*THEN sym*])
apply *arith*
apply (*auto simp add*: *ring-simps add-frac-eq frac-eq-eq*
add-divide-distrib power2-eq-square)
apply (*rule mult-pos-pos*, *assumption*)+
apply *assumption*
done

lemma *ln-x-over-x-mono*: *exp 1* <= *x* ==> *x* <= *y* ==> (*ln y* / *y*) <= (*ln x* / *x*)
proof −
assume *exp 1* <= *x* **and** *x* <= *y*
have *a*: *0* < *x* **and** *b*: *0* < *y*
apply (*insert prems*)
apply (*subgoal-tac 0* < *exp* (*1*::*real*))
apply *arith*
apply *auto*
apply (*subgoal-tac 0* < *exp* (*1*::*real*))
apply *arith*
apply *auto*

```
    done
  have x * ln y - x * ln x = x * (ln y - ln x)
    by (simp add: ring-simps)
  also have ... = x * ln(y / x)
    apply (subst ln-div)
    apply (rule b, rule a, rule refl)
    done
  also have y / x = (x + (y - x)) / x
    by simp
  also have ... = 1 + (y - x) / x using a prems by(simp add:field-simps)
  also have x * ln(1 + (y - x) / x) <= x * ((y - x) / x)
    apply (rule mult-left-mono)
    apply (rule ln-add-one-self-le-self)
    apply (rule divide-nonneg-pos)
    apply (insert prems a, simp-all)
    done
  also have ... = y - x using a by simp
  also have ... = (y - x) * ln (exp 1) by simp
  also have ... <= (y - x) * ln x
    apply (rule mult-left-mono)
    apply (subst ln-le-cancel-iff)
    apply force
    apply (rule a)
    apply (rule prems)
    apply (insert prems, simp)
    done
  also have ... = y * ln x - x * ln x
    by (rule left-diff-distrib)
  finally have x * ln y <= y * ln x
    by arith
  then have ln y <= (y * ln x) / x using a by(simp add:field-simps)
  also have ... = y * (ln x / x)  by simp
  finally show ?thesis using b by(simp add:field-simps)
qed

end
```

50 MacLaurin: MacLaurin Series

```
theory MacLaurin
imports Dense-Linear-Order Transcendental
begin
```

50.1 Maclaurin's Theorem with Lagrange Form of Remainder

This is a very long, messy proof even now that it's been broken down into lemmas.

lemma *Maclaurin-lemma*:
$0 < h \Longrightarrow$
$\exists B.\ f\ h = (\sum m{=}0..{<}n.\ (j\ m\ /\ real\ (fact\ m)) * (h\hat{}m)) +$
$(B * ((h\hat{}n)\ /\ real(fact\ n)))$
apply (*rule-tac x* = $(f\ h - (\sum m{=}0..{<}n.\ (j\ m\ /\ real\ (fact\ m)) * h\hat{}m)) *$
$real(fact\ n)\ /\ (h\hat{}n)$
in *exI*)
apply (*simp*)
done

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

A crude tactic to differentiate by proof.

lemmas *deriv-rulesI* =
DERIV-ident DERIV-const DERIV-cos DERIV-cmult
DERIV-sin DERIV-exp DERIV-inverse DERIV-pow
DERIV-add DERIV-diff DERIV-mult DERIV-minus
DERIV-inverse-fun DERIV-quotient DERIV-fun-pow
DERIV-fun-exp DERIV-fun-sin DERIV-fun-cos
DERIV-ident DERIV-const DERIV-cos

ML
⟪
local
exception DERIV-name;
fun get-fun-name (- $ (*Const* (*Lim.deriv*,-) $ *Abs*(-,-, *Const* (*f*,-) $ -) $ - $ -)) = *f*
| *get-fun-name* (- $ (- $ (*Const* (*Lim.deriv*,-) $ *Abs*(-,-, *Const* (*f*,-) $ -) $ - $ -)))
= *f*
| *get-fun-name* - = *raise DERIV-name*;

in

val deriv-tac =
SUBGOAL (*fn* (*prem*,*i*) =>
(*resolve-tac* @{*thms deriv-rulesI*} *i*) *ORELSE*
((*rtac* (*read-instantiate* [(*f*,*get-fun-name prem*)]
@{*thm DERIV-chain2*}) *i*) *handle DERIV-name* => *no-tac*));;

val DERIV-tac = *ALLGOALS*(*fn i* => *REPEAT*(*deriv-tac i*));

end
⟫

lemma *Maclaurin-lemma2*:
[| ∀ *m t*. *m* < *n* ∧ *0*≤*t* ∧ *t*≤*h* ⟶ *DERIV* (*diff m*) *t* :> *diff* (*Suc m*) *t*;
n = *Suc k*;
difg =
(λ*m t*. *diff m t* −
((∑ *p* = *0*..<*n* − *m*. *diff* (*m* + *p*) *0* / *real* (*fact p*) ∗ *t* ^ *p*) +
B ∗ (*t* ^ (*n* − *m*) / *real* (*fact* (*n* − *m*))))|] ==>
∀ *m t*. *m* < *n* & *0* ≤ *t* & *t* ≤ *h* −−>
DERIV (*difg m*) *t* :> *difg* (*Suc m*) *t*
apply *clarify*
apply (*rule DERIV-diff*)
apply (*simp* (*no-asm-simp*))
apply (*tactic DERIV-tac*)
apply (*tactic DERIV-tac*)
apply (*rule-tac* [*2*] *lemma-DERIV-subst*)
apply (*rule-tac* [*2*] *DERIV-quotient*)
apply (*rule-tac* [*3*] *DERIV-const*)
apply (*rule-tac* [*2*] *DERIV-pow*)
prefer *3* **apply** (*simp add*: *fact-diff-Suc*)
prefer *2* **apply** *simp*
apply (*frule-tac m* = *m* **in** *less-add-one*, *clarify*)
apply (*simp del*: *setsum-op-ivl-Suc*)
apply (*insert sumr-offset4* [*of 1*])
apply (*simp del*: *setsum-op-ivl-Suc fact-Suc realpow-Suc*)
apply (*rule lemma-DERIV-subst*)
apply (*rule DERIV-add*)
apply (*rule-tac* [*2*] *DERIV-const*)
apply (*rule DERIV-sumr*, *clarify*)
prefer *2* **apply** *simp*
apply (*simp* (*no-asm*) *add*: *divide-inverse mult-assoc del*: *fact-Suc realpow-Suc*)
apply (*rule DERIV-cmult*)
apply (*rule lemma-DERIV-subst*)
apply (*best intro*: *DERIV-chain2 intro*!: *DERIV-intros*)
apply (*subst fact-Suc*)
apply (*subst real-of-nat-mult*)
apply (*simp add*: *mult-ac*)
done

lemma *Maclaurin-lemma3*:
fixes *difg* :: *nat* => *real* => *real* **shows**
[|∀ *k t*. *k* < *Suc m* ∧ *0*≤*t* & *t*≤*h* ⟶ *DERIV* (*difg k*) *t* :> *difg* (*Suc k*) *t*;
∀ *k*<*Suc m*. *difg k 0* = *0*; *DERIV* (*difg n*) *t* :> *0*; *n* < *m*; *0* < *t*;
t < *h*|]
==> ∃ *ta*. *0* < *ta* & *ta* < *t* & *DERIV* (*difg* (*Suc n*)) *ta* :> *0*
apply (*rule Rolle*, *assumption*, *simp*)
apply (*drule-tac x* = *n* **and** *P*=%*k*. *k*<*Suc m* −−> *difg k 0* = *0* **in** *spec*)
apply (*rule DERIV-unique*)
prefer *2* **apply** *assumption*

apply *force*
apply (*metis DERIV-isCont dlo-simps*(*4*) *dlo-simps*(*9*) *less-trans-Suc nat-less-le not-less-eq real-le-trans*)
apply (*metis Suc-less-eq differentiableI dlo-simps*(*7*) *dlo-simps*(*8*) *dlo-simps*(*9*) *real-le-trans xt1*(*8*))
done

lemma *Maclaurin*:
[| *0* < *h*; *n* > *0*; *diff 0* = *f*;
∀ *m t*. *m* < *n* & *0* ≤ *t* & *t* ≤ *h* --> *DERIV* (*diff m*) *t* :> *diff* (*Suc m*) *t* |]
==> ∃ *t*. *0* < *t* &
t < *h* &
f h =
setsum (%*m*. (*diff m 0* / *real* (*fact m*)) * *h* ^ *m*) {*0*..<*n*} +
(*diff n t* / *real* (*fact n*)) * *h* ^ *n*
apply (*case-tac n* = *0*, *force*)
apply (*drule not0-implies-Suc*)
apply (*erule exE*)
apply (*frule-tac f*=*f* **and** *n*=*n* **and** *j*=%*m*. *diff m 0* **in** *Maclaurin-lemma*)
apply (*erule exE*)
apply (*subgoal-tac* ∃ *g*.
g = (%*t*. *f t* − (*setsum* (%*m*. (*diff m 0* / *real*(*fact m*)) * *t*^*m*) {*0*..<*n*} + (*B* * (*t*^*n* / *real*(*fact n*))))))
prefer *2* **apply** *blast*
apply (*erule exE*)
apply (*subgoal-tac g 0* = *0* & *g h* =*0*)
prefer *2*
apply (*simp del*: *setsum-op-ivl-Suc*)
apply (*cut-tac n* = *m* **and** *k* = *1* **in** *sumr-offset2*)
apply (*simp add*: *eq-diff-eq'* *del*: *setsum-op-ivl-Suc*)
apply (*subgoal-tac* ∃ *difg*. *difg* = (%*m t*. *diff m t* − (*setsum* (%*p*. (*diff* (*m* + *p*) *0* / *real* (*fact p*)) * (*t* ^ *p*)) {*0*..<*n*−*m*} + (*B* * ((*t* ^ (*n* − *m*)) / *real* (*fact* (*n* − *m*)))))))
prefer *2* **apply** *blast*
apply (*erule exE*)
apply (*subgoal-tac difg 0* = *g*)
prefer *2* **apply** *simp*
apply (*frule Maclaurin-lemma2*, *assumption*+)
apply (*subgoal-tac* ∀ *ma*. *ma* < *n* --> (∃ *t*. *0* < *t* & *t* < *h* & *difg* (*Suc ma*) *t* = *0*))
apply (*drule-tac x* = *m* **and** *P*=%*m*. *m*<*n* --> (∃ *t*. *?QQ m t*) **in** *spec*)
apply (*erule impE*)
apply (*simp* (*no-asm-simp*))
apply (*erule exE*)
apply (*rule-tac x* = *t* **in** *exI*)
apply (*simp del*: *realpow-Suc fact-Suc*)
apply (*subgoal-tac* ∀ *m*. *m* < *n* --> *difg m 0* = *0*)
prefer *2*
apply *clarify*

```
 apply simp
 apply (frule-tac m = ma in less-add-one, clarify)
 apply (simp del: setsum-op-ivl-Suc)
apply (insert sumr-offset4 [of 1])
apply (simp del: setsum-op-ivl-Suc fact-Suc realpow-Suc)
apply (subgoal-tac ∀m. m < n --> (∃t. 0 < t & t < h & DERIV (difg m) t
:> 0) )
apply (rule allI, rule impI)
apply (drule-tac x = ma and P=%m. m<n --> (∃t. ?QQ m t) in spec)
apply (erule impE, assumption)
apply (erule exE)
apply (rule-tac x = t in exI)

apply (erule-tac [!] V= difg = (%m t. diff m t - (setsum (%p. diff (m + p) 0 /
real (fact p) * t ^ p) {0..<n-m} + B * (t ^ (n - m) / real (fact (n - m)))))
      in thin-rl)
apply (erule-tac [!] V=g = (%t. f t - (setsum (%m. diff m 0 / real (fact m) * t
^ m) {0..<n} + B * (t ^ n / real (fact n))))
      in thin-rl)
apply (erule-tac [!] V=f h = setsum (%m. diff m 0 / real (fact m) * h ^ m)
{0..<n} + B * (h ^ n / real (fact n))
      in thin-rl)

apply (simp (no-asm-simp))
apply (rule DERIV-unique)
prefer 2 apply blast
apply force
apply (rule allI, induct-tac ma)
apply (rule impI, rule Rolle, assumption, simp, simp)
apply (metis DERIV-isCont zero-less-Suc)
apply (metis One-nat-def differentiableI dlo-simps(7))
apply safe
apply force
apply (frule Maclaurin-lemma3, assumption+, safe)
apply (rule-tac x = ta in exI, force)
done

lemma Maclaurin-objl:
  0 < h & n>0 & diff 0 = f &
  (∀m t. m < n & 0 ≤ t & t ≤ h --> DERIV (diff m) t :> diff (Suc m) t)
   --> (∃t. 0 < t & t < h &
         f h = (∑m=0..<n. diff m 0 / real (fact m) * h ^ m) +
               diff n t / real (fact n) * h ^ n)
by (blast intro: Maclaurin)


lemma Maclaurin2:
   [| 0 < h; diff 0 = f;
      ∀m t.
```

```
        m < n & 0 ≤ t & t ≤ h --> DERIV (diff m) t :> diff (Suc m) t |]
   ==> ∃t. 0 < t &
            t ≤ h &
            f h =
            (∑m=0..<n. diff m 0 / real (fact m) * h ^ m) +
            diff n t / real (fact n) * h ^ n
apply (case-tac n, auto)
apply (drule Maclaurin, auto)
done

lemma Maclaurin2-objl:
     0 < h & diff 0 = f &
      (∀m t.
         m < n & 0 ≤ t & t ≤ h --> DERIV (diff m) t :> diff (Suc m) t)
   --> (∃t. 0 < t &
            t ≤ h &
            f h =
            (∑m=0..<n. diff m 0 / real (fact m) * h ^ m) +
            diff n t / real (fact n) * h ^ n)
by (blast intro: Maclaurin2)

lemma Maclaurin-minus:
   [| h < 0; n > 0; diff 0 = f;
      ∀m t. m < n & h ≤ t & t ≤ 0 --> DERIV (diff m) t :> diff (Suc m) t |]
   ==> ∃t. h < t &
            t < 0 &
            f h =
            (∑m=0..<n. diff m 0 / real (fact m) * h ^ m) +
            diff n t / real (fact n) * h ^ n
apply (cut-tac f = %x. f (-x)
       and diff = %n x. (-1 ^ n) * diff n (-x)
       and h = -h and n = n in Maclaurin-objl)
apply (simp)
apply safe
apply (subst minus-mult-right)
apply (rule DERIV-cmult)
apply (rule lemma-DERIV-subst)
apply (rule DERIV-chain2 [where g=uminus])
apply (rule-tac [2] DERIV-minus, rule-tac [2] DERIV-ident)
prefer 2 apply force
apply force
apply (rule-tac x = -t in exI, auto)
apply (subgoal-tac (∑m = 0..<n. -1 ^ m * diff m 0 * (-h)^m / real(fact m))
=
                   (∑m = 0..<n. diff m 0 * h ^ m / real(fact m)))
apply (rule-tac [2] setsum-cong[OF refl])
apply (auto simp add: divide-inverse power-mult-distrib [symmetric])
done
```

lemma *Maclaurin-minus-objl*:
(*h* < *0* & *n* > *0* & *diff 0* = *f* &
(∀ *m t*.
m < *n* & *h* ≤ *t* & *t* ≤ *0* −−> *DERIV* (*diff m*) *t* :> *diff* (*Suc m*) *t*))
−−> (∃ *t*. *h* < *t* &
t < *0* &
f h =
(∑ *m*=*0*..<*n*. *diff m 0* / *real* (*fact m*) * *h* ^ *m*) +
diff n t / *real* (*fact n*) * *h* ^ *n*)
by (*blast intro*: *Maclaurin-minus*)

50.2 More Convenient "Bidirectional" Version.

lemma *Maclaurin-bi-le-lemma* [*rule-format*]:
n>*0* ⟶
diff 0 0 =
(∑ *m* = *0*..<*n*. *diff m 0* * *0* ^ *m* / *real* (*fact m*)) +
diff n 0 * *0* ^ *n* / *real* (*fact n*)
by (*induct n*, *auto*)

lemma *Maclaurin-bi-le*:
[| *diff 0* = *f*;
∀ *m t*. *m* < *n* & *abs t* ≤ *abs x* −−> *DERIV* (*diff m*) *t* :> *diff* (*Suc m*) *t* |]
==> ∃ *t*. *abs t* ≤ *abs x* &
f x =
(∑ *m*=*0*..<*n*. *diff m 0* / *real* (*fact m*) * *x* ^ *m*) +
diff n t / *real* (*fact n*) * *x* ^ *n*
apply (*case-tac n* = *0*, *force*)
apply (*case-tac x* = *0*)
apply (*rule-tac x* = *0* **in** *exI*)
apply (*force simp add*: *Maclaurin-bi-le-lemma*)
apply (*cut-tac x* = *x* **and** *y* = *0* **in** *linorder-less-linear*, *auto*)

Case 1, where *x* < *0*

apply (*cut-tac f* = *diff 0* **and** *diff* = *diff* **and** *h* = *x* **and** *n* = *n* **in** *Maclaurin-minus-objl*, *safe*)
apply (*simp add*: *abs-if*)
apply (*rule-tac x* = *t* **in** *exI*)
apply (*simp add*: *abs-if*)

Case 2, where *0* < *x*

apply (*cut-tac f* = *diff 0* **and** *diff* = *diff* **and** *h* = *x* **and** *n* = *n* **in** *Maclaurin-objl*, *safe*)
apply (*simp add*: *abs-if*)
apply (*rule-tac x* = *t* **in** *exI*)
apply (*simp add*: *abs-if*)
done

lemma *Maclaurin-all-lt*:

```
    [| diff 0 = f;
       ∀ m x. DERIV (diff m) x :> diff(Suc m) x;
       x ~= 0; n > 0
     |] ==> ∃ t. 0 < abs t & abs t < abs x &
            f x = (∑ m=0..<n. (diff m 0 / real (fact m)) * x ^ m) +
                  (diff n t / real (fact n)) * x ^ n
apply (rule-tac x = x and y = 0 in linorder-cases)
prefer 2 apply blast
apply (drule-tac [2] diff=diff in Maclaurin)
apply (drule-tac diff=diff in Maclaurin-minus, simp-all, safe)
apply (rule-tac [!] x = t in exI, auto)
done

lemma Maclaurin-all-lt-objl:
     diff 0 = f &
     (∀ m x. DERIV (diff m) x :> diff(Suc m) x) &
     x ~= 0 & n > 0
     --> (∃ t. 0 < abs t & abs t < abs x &
            f x = (∑ m=0..<n. (diff m 0 / real (fact m)) * x ^ m) +
                  (diff n t / real (fact n)) * x ^ n)
by (blast intro: Maclaurin-all-lt)

lemma Maclaurin-zero [rule-format]:
     x = (0::real)
     ==> n ≠ 0 -->
        (∑ m=0..<n. (diff m (0::real) / real (fact m)) * x ^ m) =
        diff 0 0
by (induct n, auto)

lemma Maclaurin-all-le: [| diff 0 = f;
       ∀ m x. DERIV (diff m) x :> diff (Suc m) x
     |] ==> ∃ t. abs t ≤ abs x &
            f x = (∑ m=0..<n. (diff m 0 / real (fact m)) * x ^ m) +
                  (diff n t / real (fact n)) * x ^ n
apply(cases n=0)
apply (force)
apply (case-tac x = 0)
apply (frule-tac diff = diff and n = n in Maclaurin-zero, assumption)
apply (drule not0-implies-Suc)
apply (rule-tac x = 0 in exI, force)
apply (frule-tac diff = diff and n = n in Maclaurin-all-lt, auto)
apply (rule-tac x = t in exI, auto)
done

lemma Maclaurin-all-le-objl: diff 0 = f &
     (∀ m x. DERIV (diff m) x :> diff (Suc m) x)
     --> (∃ t. abs t ≤ abs x &
            f x = (∑ m=0..<n. (diff m 0 / real (fact m)) * x ^ m) +
                  (diff n t / real (fact n)) * x ^ n)
```

by (*blast intro*: *Maclaurin-all-le*)

50.3 Version for Exponential Function

lemma *Maclaurin-exp-lt*: [| $x \sim= 0$; $n > 0$ |]
$==>$ ($\exists t.\ 0 < abs\ t\ \&$
$abs\ t < abs\ x\ \&$
$exp\ x = (\sum m{=}0..{<}n.\ (x\ \hat{}\ m)\ /\ real\ (fact\ m)) +$
$(exp\ t\ /\ real\ (fact\ n)) * x\ \hat{}\ n$)
by (*cut-tac diff* = $\%n.\ exp$ **and** $f = exp$ **and** $x = x$ **and** $n = n$ **in** *Maclaurin-all-lt-objl*, *auto*)

lemma *Maclaurin-exp-le*:
$\exists t.\ abs\ t \leq abs\ x\ \&$
$exp\ x = (\sum m{=}0..{<}n.\ (x\ \hat{}\ m)\ /\ real\ (fact\ m)) +$
$(exp\ t\ /\ real\ (fact\ n)) * x\ \hat{}\ n$
by (*cut-tac diff* = $\%n.\ exp$ **and** $f = exp$ **and** $x = x$ **and** $n = n$ **in** *Maclaurin-all-le-objl*, *auto*)

50.4 Version for Sine Function

lemma *MVT2*:
[| $a < b$; $\forall x.\ a \leq x\ \&\ x \leq b \dashrightarrow DERIV\ f\ x :> f'(x)$ |]
$==>$ $\exists z{::}real.\ a < z\ \&\ z < b\ \&\ (f\ b - f\ a = (b - a) * f'(z))$
apply (*drule MVT*)
apply (*blast intro*: *DERIV-isCont*)
apply (*force dest*: *order-less-imp-le simp add*: *differentiable-def*)
apply (*blast dest*: *DERIV-unique order-less-imp-le*)
done

lemma *mod-exhaust-less-4*:
$m\ mod\ 4 = 0 \mid m\ mod\ 4 = 1 \mid m\ mod\ 4 = 2 \mid m\ mod\ 4 = (3{::}nat)$
by *auto*

lemma *Suc-Suc-mult-two-diff-two* [*rule-format*, *simp*]:
$n{\neq}0 \dashrightarrow Suc\ (Suc\ (2 * n - 2)) = 2{*}n$
by (*induct n*, *auto*)

lemma *lemma-Suc-Suc-4n-diff-2* [*rule-format*, *simp*]:
$n{\neq}0 \dashrightarrow Suc\ (Suc\ (4{*}n - 2)) = 4{*}n$
by (*induct n*, *auto*)

lemma *Suc-mult-two-diff-one* [*rule-format*, *simp*]:
$n{\neq}0 \dashrightarrow Suc\ (2 * n - 1) = 2{*}n$
by (*induct n*, *auto*)

It is unclear why so many variant results are needed.

lemma *Maclaurin-sin-expansion2*:

```
    ∃ t. abs t ≤ abs x &
      sin x =
      (∑ m=0..<n. (if even m then 0
                   else (−1 ^ ((m − Suc 0) div 2)) / real (fact m)) *
                   x ^ m)
     + ((sin(t + 1/2 * real (n) *pi) / real (fact n)) * x ^ n)
apply (cut-tac f = sin and n = n and x = x
       and diff = %n x. sin (x + 1/2*real n * pi) in Maclaurin-all-lt-objl)
apply safe
apply (simp (no-asm))
apply (simp (no-asm))
apply (case-tac n, clarify, simp, simp add: lemma-STAR-sin)
apply (rule ccontr, simp)
apply (drule-tac x = x in spec, simp)
apply (erule ssubst)
apply (rule-tac x = t in exI, simp)
apply (rule setsum-cong[OF refl])
apply (auto simp add: sin-zero-iff odd-Suc-mult-two-ex)
done

lemma Maclaurin-sin-expansion:
    ∃ t. sin x =
      (∑ m=0..<n. (if even m then 0
                   else (−1 ^ ((m − Suc 0) div 2)) / real (fact m)) *
                   x ^ m)
     + ((sin(t + 1/2 * real (n) *pi) / real (fact n)) * x ^ n)
apply (insert Maclaurin-sin-expansion2 [of x n])
apply (blast intro: elim:)
done


lemma Maclaurin-sin-expansion3:
    [| n > 0; 0 < x |] ==>
      ∃ t. 0 < t & t < x &
      sin x =
      (∑ m=0..<n. (if even m then 0
                   else (−1 ^ ((m − Suc 0) div 2)) / real (fact m)) *
                   x ^ m)
     + ((sin(t + 1/2 * real(n) *pi) / real (fact n)) * x ^ n)
apply (cut-tac f = sin and n = n and h = x and diff = %n x. sin (x + 1/2*real
(n) *pi) in Maclaurin-objl)
apply safe
apply simp
apply (simp (no-asm))
apply (erule ssubst)
apply (rule-tac x = t in exI, simp)
apply (rule setsum-cong[OF refl])
apply (auto simp add: sin-zero-iff odd-Suc-mult-two-ex)
done
```

lemma *Maclaurin-sin-expansion4*:
$0 < x \Longrightarrow$
$\exists t.\ 0 < t \ \&\ t \leq x \ \&$
$sin\ x =$
$(\sum m{=}0..{<}n.\ (if\ even\ m\ then\ 0$
$else\ (-1\ \hat{}\ ((m - Suc\ 0)\ div\ 2))\ /\ real\ (fact\ m)) *$
$x\ \hat{}\ m)$
$+\ ((sin(t + 1/2 * real\ (n) * pi)\ /\ real\ (fact\ n)) * x\ \hat{}\ n)$
apply (*cut-tac f* = *sin* **and** *n* = *n* **and** *h* = *x* **and** *diff* = %*n x*. *sin* (*x* + *1/2*∗*real* (*n*) ∗*pi*) **in** *Maclaurin2-objl*)
apply *safe*
apply *simp*
apply (*simp* (*no-asm*))
apply (*erule ssubst*)
apply (*rule-tac x* = *t* **in** *exI*, *simp*)
apply (*rule setsum-cong*[*OF refl*])
apply (*auto simp add*: *sin-zero-iff odd-Suc-mult-two-ex*)
done

50.5 Maclaurin Expansion for Cosine Function

lemma *sumr-cos-zero-one* [*simp*]:
$(\sum m{=}0..{<}(Suc\ n).$
$(if\ even\ m\ then\ -1\ \hat{}\ (m\ div\ 2)/(real\ \ (fact\ m))\ else\ 0) * 0\ \hat{}\ m) = 1$
by (*induct n*, *auto*)

lemma *Maclaurin-cos-expansion*:
$\exists t.\ abs\ t \leq abs\ x \ \&$
$cos\ x =$
$(\sum m{=}0..{<}n.\ (if\ even\ m$
$then\ -1\ \hat{}\ (m\ div\ 2)/(real\ (fact\ m))$
$else\ 0) *$
$x\ \hat{}\ m)$
$+\ ((cos(t + 1/2 * real\ (n) * pi)\ /\ real\ (fact\ n)) * x\ \hat{}\ n)$
apply (*cut-tac f* = *cos* **and** *n* = *n* **and** *x* = *x* **and** *diff* = %*n x*. *cos* (*x* + *1/2*∗*real* (*n*) ∗*pi*) **in** *Maclaurin-all-lt-objl*)
apply *safe*
apply (*simp* (*no-asm*))
apply (*simp* (*no-asm*))
apply (*case-tac n*, *simp*)
apply (*simp del*: *setsum-op-ivl-Suc*)
apply (*rule ccontr*, *simp*)
apply (*drule-tac x* = *x* **in** *spec*, *simp*)
apply (*erule ssubst*)
apply (*rule-tac x* = *t* **in** *exI*, *simp*)
apply (*rule setsum-cong*[*OF refl*])
apply (*auto simp add*: *cos-zero-iff even-mult-two-ex*)
done

lemma *Maclaurin-cos-expansion2*:
[| $0 < x$; $n > 0$ |] ==>
$\exists t.\ 0 < t\ \&\ t < x\ \&$
$cos\ x =$
$(\sum m{=}0..{<}n.\ (if\ even\ m$
$then\ -1\ \hat{}\ (m\ div\ 2)/(real\ (fact\ m))$
$else\ 0) *$
$x\ \hat{}\ m)$
$+ ((cos(t + 1/2 * real\ (n) * pi)\ /\ real\ (fact\ n)) * x\ \hat{}\ n)$
apply (*cut-tac f* = *cos* **and** $n = n$ **and** $h = x$ **and** *diff* = $\%n\ x.\ cos\ (x + 1/2{*}real\ (n) * pi)$ **in** *Maclaurin-objl*)
apply *safe*
apply *simp*
apply (*simp* (*no-asm*))
apply (*erule ssubst*)
apply (*rule-tac* $x = t$ **in** *exI*, *simp*)
apply (*rule setsum-cong*[*OF refl*])
apply (*auto simp add*: *cos-zero-iff even-mult-two-ex*)
done

lemma *Maclaurin-minus-cos-expansion*:
[| $x < 0$; $n > 0$ |] ==>
$\exists t.\ x < t\ \&\ t < 0\ \&$
$cos\ x =$
$(\sum m{=}0..{<}n.\ (if\ even\ m$
$then\ -1\ \hat{}\ (m\ div\ 2)/(real\ (fact\ m))$
$else\ 0) *$
$x\ \hat{}\ m)$
$+ ((cos(t + 1/2 * real\ (n) * pi)\ /\ real\ (fact\ n)) * x\ \hat{}\ n)$
apply (*cut-tac f* = *cos* **and** $n = n$ **and** $h = x$ **and** *diff* = $\%n\ x.\ cos\ (x + 1/2{*}real\ (n) * pi)$ **in** *Maclaurin-minus-objl*)
apply *safe*
apply *simp*
apply (*simp* (*no-asm*))
apply (*erule ssubst*)
apply (*rule-tac* $x = t$ **in** *exI*, *simp*)
apply (*rule setsum-cong*[*OF refl*])
apply (*auto simp add*: *cos-zero-iff even-mult-two-ex*)
done

lemma *sin-bound-lemma*:
$[|x = y;\ abs\ u \leq (v{::}real)\ |] ==> |(x + u) - y| \leq v$
by *auto*

lemma *Maclaurin-sin-bound*:
abs(*sin x* − (∑ *m*=*0*..<*n*. (*if even m then 0 else* (−*1* ^ ((*m* − *Suc 0*) *div 2*)) / *real* (*fact m*)) ∗
x ^ *m*)) ≤ *inverse*(*real* (*fact n*)) ∗ |*x*| ^ *n*
proof −
have !! *x* (*y*::*real*). *x* ≤ *1* ⟹ *0* ≤ *y* ⟹ *x* ∗ *y* ≤ *1* ∗ *y*
by (*rule-tac mult-right-mono*,*simp-all*)
note *est* = *this*[*simplified*]
let *?diff* = λ(*n*::*nat*) *x*. *if n mod 4* = *0 then sin*(*x*) *else if n mod 4* = *1 then cos*(*x*) *else if n mod 4* = *2 then* −*sin*(*x*) *else* −*cos*(*x*)
have *diff-0*: *?diff 0* = *sin* **by** *simp*
have *DERIV-diff*: ∀ *m x*. *DERIV* (*?diff m*) *x* :> *?diff* (*Suc m*) *x*
apply (*clarify*)
apply (*subst* (*1 2 3*) *mod-Suc-eq-Suc-mod*)
apply (*cut-tac m*=*m* **in** *mod-exhaust-less-4*)
apply (*safe*, *simp-all*)
apply (*rule DERIV-minus*, *simp*)
apply (*rule lemma-DERIV-subst*, *rule DERIV-minus*, *rule DERIV-cos*, *simp*)
done
from *Maclaurin-all-le* [*OF diff-0 DERIV-diff*]
obtain *t* **where** *t1*: |*t*| ≤ |*x*| **and**
t2: *sin x* = (∑ *m* = *0*..<*n*. *?diff m 0* / *real* (*fact m*) ∗ *x* ^ *m*) +
?diff n t / *real* (*fact n*) ∗ *x* ^ *n* **by** *fast*
have *diff-m-0*:
⋀*m*. *?diff m 0* = (*if even m then 0*
else −*1* ^ ((*m* − *Suc 0*) *div 2*))
apply (*subst even-even-mod-4-iff*)
apply (*cut-tac m*=*m* **in** *mod-exhaust-less-4*)
apply (*elim disjE*, *simp-all*)
apply (*safe dest*!: *mod-eqD*, *simp-all*)
done
show *?thesis*
apply (*subst t2*)
apply (*rule sin-bound-lemma*)
apply (*rule setsum-cong*[*OF refl*])
apply (*subst diff-m-0*, *simp*)
apply (*auto intro*: *mult-right-mono* [**where** *b*=*1*, *simplified*] *mult-right-mono*
simp add: *est mult-nonneg-nonneg mult-ac divide-inverse*
power-abs [*symmetric*] *abs-mult*)
done
qed

end

51 Taylor: Taylor series

theory *Taylor*
imports *MacLaurin*

begin

We use MacLaurin and the translation of the expansion point c to 0 to prove Taylor's theorem.

lemma *taylor-up*:
assumes *INIT*: $n>0$ *diff* $0 = f$
and *DERIV*: $(\forall\ m\ t.\ m < n\ \&\ a \leq t\ \&\ t \leq b \longrightarrow DERIV\ (diff\ m)\ t :> (diff\ (Suc\ m)\ t))$
and *INTERV*: $a \leq c\ c < b$
shows $\exists\ t.\ c < t\ \&\ t < b\ \&$
$f\ b = setsum\ (\%m.\ (diff\ m\ c\ /\ real\ (fact\ m)) * (b - c)\hat{}m)\ \{0..<n\} +$
$(diff\ n\ t\ /\ real\ (fact\ n)) * (b - c)\hat{}n$
proof −
from *INTERV* **have** $0 < b-c$ **by** *arith*
moreover
from *INIT* **have** $n>0\ ((\lambda m\ x.\ diff\ m\ (x + c))\ 0) = (\lambda x.\ f\ (x + c))$ **by** *auto*
moreover
have *ALL* $m\ t.\ m < n\ \&\ 0 <= t\ \&\ t <= b - c \dashrightarrow DERIV\ (\%x.\ diff\ m\ (x + c))\ t :> diff\ (Suc\ m)\ (t + c)$
proof (*intro strip*)
fix $m\ t$
assume $m < n\ \&\ 0 <= t\ \&\ t <= b - c$
with *DERIV* **and** *INTERV* **have** $DERIV\ (diff\ m)\ (t + c) :> diff\ (Suc\ m)\ (t + c)$ **by** *auto*
moreover
from *DERIV-ident* **and** *DERIV-const* **have** $DERIV\ (\%x.\ x + c)\ t :> 1+0$ **by** (*rule DERIV-add*)
ultimately have $DERIV\ (\%x.\ diff\ m\ (x + c))\ t :> diff\ (Suc\ m)\ (t + c) * (1+0)$
by (*rule DERIV-chain2*)
thus $DERIV\ (\%x.\ diff\ m\ (x + c))\ t :> diff\ (Suc\ m)\ (t + c)$ **by** *simp*
qed
ultimately
have *EX*:$EX\ t>0.\ t < b - c\ \&$
$f\ (b - c + c) = (SUM\ m = 0..<n.\ diff\ m\ (0 + c)\ /\ real\ (fact\ m) * (b - c)\ \hat{}\ m) +$
$diff\ n\ (t + c)\ /\ real\ (fact\ n) * (b - c)\ \hat{}\ n$
by (*rule Maclaurin*)
show *?thesis*
proof −
from *EX* **obtain** x **where**
X: $0 < x\ \&\ x < b - c\ \&$
$f\ (b - c + c) = (\sum m = 0..<n.\ diff\ m\ (0 + c)\ /\ real\ (fact\ m) * (b - c)\ \hat{}\ m) +$
$diff\ n\ (x + c)\ /\ real\ (fact\ n) * (b - c)\ \hat{}\ n$ **..**
let *?H* $= x + c$
from *X* **have** $c<?H\ \&\ ?H<b \wedge f\ b = (\sum m = 0..<n.\ diff\ m\ c\ /\ real\ (fact\ m) * (b - c)\ \hat{}\ m) +$
$diff\ n\ ?H\ /\ real\ (fact\ n) * (b - c)\ \hat{}\ n$

```
    by fastsimp
   thus ?thesis by fastsimp
  qed
qed

lemma taylor-down:
  assumes INIT: n>0 diff 0 = f
  and DERIV: (∀ m t. m < n & a ≤ t & t ≤ b ⟶ DERIV (diff m) t :> (diff
(Suc m) t))
  and INTERV: a < c c ≤ b
  shows ∃ t. a < t & t < c &
    f a = setsum (% m. (diff m c / real (fact m)) * (a - c) ^m) {0..<n} +
      (diff n t / real (fact n)) * (a - c) ^n
proof -
  from INTERV have a-c < 0 by arith
  moreover
  from INIT have n>0 ((λm x. diff m (x + c)) 0) = (λx. f (x + c)) by auto
  moreover
  have ALL m t. m < n & a-c <= t & t <= 0 --> DERIV (%x. diff m (x +
c)) t :> diff (Suc m) (t + c)
  proof (rule allI impI)+
    fix m t
    assume m < n & a-c <= t & t <= 0
    with DERIV and INTERV have DERIV (diff m) (t + c) :> diff (Suc m) (t
+ c) by auto
    moreover
    from DERIV-ident and DERIV-const have DERIV (%x. x + c) t :> 1+0
by (rule DERIV-add)
    ultimately have DERIV (%x. diff m (x + c)) t :> diff (Suc m) (t + c) *
(1+0) by (rule DERIV-chain2)
    thus DERIV (%x. diff m (x + c)) t :> diff (Suc m) (t + c) by simp
  qed
  ultimately
  have EX: EX t>a - c. t < 0 &
    f (a - c + c) = (SUM m = 0..<n. diff m (0 + c) / real (fact m) * (a - c)
^ m) +
      diff n (t + c) / real (fact n) * (a - c) ^ n
    by (rule Maclaurin-minus)
  show ?thesis
  proof -
    from EX obtain x where X: a - c < x & x < 0 &
      f (a - c + c) = (SUM m = 0..<n. diff m (0 + c) / real (fact m) * (a - c)
^ m) +
        diff n (x + c) / real (fact n) * (a - c) ^ n ..
    let ?H = x + c
    from X have a<?H & ?H<c ∧ f a = (∑ m = 0..<n. diff m c / real (fact
m) * (a - c) ^ m) +
      diff n ?H / real (fact n) * (a - c) ^ n
      by fastsimp
```

thus *?thesis* **by** *fastsimp*
qed
qed

lemma *taylor*:
assumes *INIT*: $n>0$ *diff* $0 = f$
and *DERIV*: $(\forall\ m\ t.\ m < n\ \&\ a \leq t\ \&\ t \leq b \longrightarrow DERIV\ (diff\ m)\ t :> (diff\ (Suc\ m)\ t))$
and *INTERV*: $a \leq c\ \ c \leq b\ a \leq x\ x \leq b\ x \neq c$
shows $\exists\ t.\ (if\ x<c\ then\ (x < t\ \&\ t < c)\ else\ (c < t\ \&\ t < x))\ \&$
$f\ x = setsum\ (\%\ m.\ (diff\ m\ c\ /\ real\ (fact\ m)) * (x - c)\hat{}m)\ \{0..<n\} +$
$(diff\ n\ t\ /\ real\ (fact\ n)) * (x - c)\hat{}n$
proof (*cases* $x<c$)
case *True*
note *INIT*
moreover from *DERIV* **and** *INTERV*
have $\forall m\ t.\ m < n \wedge x \leq t \wedge t \leq b \longrightarrow DERIV\ (diff\ m)\ t :> diff\ (Suc\ m)\ t$
by *fastsimp*
moreover note *True*
moreover from *INTERV* **have** $c \leq b$ **by** *simp*
ultimately have *EX*: $\exists t>x.\ t < c \wedge f\ x =$
$(\sum m = 0..<n.\ diff\ m\ c\ /\ real\ (fact\ m) * (x - c)\ \hat{}\ m) +$
$diff\ n\ t\ /\ real\ (fact\ n) * (x - c)\ \hat{}\ n$
by (*rule taylor-down*)
with *True* **show** *?thesis* **by** *simp*
next
case *False*
note *INIT*
moreover from *DERIV* **and** *INTERV*
have $\forall m\ t.\ m < n \wedge a \leq t \wedge t \leq x \longrightarrow DERIV\ (diff\ m)\ t :> diff\ (Suc\ m)\ t$
by *fastsimp*
moreover from *INTERV* **have** $a \leq c$ **by** *arith*
moreover from *False* **and** *INTERV* **have** $c < x$ **by** *arith*
ultimately have *EX*: $\exists t>c.\ t < x \wedge f\ x =$
$(\sum m = 0..<n.\ diff\ m\ c\ /\ real\ (fact\ m) * (x - c)\ \hat{}\ m) +$
$diff\ n\ t\ /\ real\ (fact\ n) * (x - c)\ \hat{}\ n$
by (*rule taylor-up*)
with *False* **show** *?thesis* **by** *simp*
qed

end

52 Integration: Theory of Integration

theory *Integration*
imports *MacLaurin*
begin

We follow John Harrison in formalizing the Gauge integral.

definition
— Partitions and tagged partitions etc.

partition :: [(*real*∗*real*),*nat* => *real*] => *bool* **where**
partition = (%(*a*,*b*) *D*. *D* 0 = *a* &
(∃ *N*. (∀ *n* < *N*. *D*(*n*) < *D*(*Suc n*)) &
(∀ *n* ≥ *N*. *D*(*n*) = *b*)))

definition
psize :: (*nat* => *real*) => *nat* **where**
psize D = (*SOME N*. (∀ *n* < *N*. *D*(*n*) < *D*(*Suc n*)) &
(∀ *n* ≥ *N*. *D*(*n*) = *D*(*N*)))

definition
tpart :: [(*real*∗*real*),((*nat* => *real*)∗(*nat* =>*real*))] => *bool* **where**
tpart = (%(*a*,*b*) (*D*,*p*). *partition*(*a*,*b*) *D* &
(∀ *n*. *D*(*n*) ≤ *p*(*n*) & *p*(*n*) ≤ *D*(*Suc n*)))

— Gauges and gauge-fine divisions

definition
gauge :: [*real* => *bool*, *real* => *real*] => *bool* **where**
gauge E g = (∀ *x*. *E x* −−> 0 < *g*(*x*))

definition
fine :: [*real* => *real*, ((*nat* => *real*)∗(*nat* => *real*))] => *bool* **where**
fine = (%*g* (*D*,*p*). ∀ *n*. *n* < (*psize D*) −−> *D*(*Suc n*) − *D*(*n*) < *g*(*p n*))

— Riemann sum

definition
rsum :: [((*nat*=>*real*)∗(*nat*=>*real*)),*real*=>*real*] => *real* **where**
rsum = (%(*D*,*p*) *f*. ∑ *n*=0..<*psize*(*D*). *f*(*p n*) ∗ (*D*(*Suc n*) − *D*(*n*)))

— Gauge integrability (definite)

definition
Integral :: [(*real*∗*real*),*real*=>*real*,*real*] => *bool* **where**
Integral = (%(*a*,*b*) *f k*. ∀ *e* > 0.
(∃ *g*. *gauge*(%*x*. *a* ≤ *x* & *x* ≤ *b*) *g* &
(∀ *D p*. *tpart*(*a*,*b*) (*D*,*p*) & *fine*(*g*)(*D*,*p*) −−>
|*rsum*(*D*,*p*) *f* − *k*| < *e*)))

lemma *partition-zero* [*simp*]: *a* = *b* ==> *psize* (%*n*. *if n* = 0 *then a else b*) = 0
by (*auto simp add*: *psize-def*)

lemma *partition-one* [*simp*]: *a* < *b* ==> *psize* (%*n*. *if n* = 0 *then a else b*) = 1

apply (*simp add*: *psize-def*)
apply (*rule some-equality*, *auto*)
apply (*drule-tac x = 1* **in** *spec*, *auto*)
done

lemma *partition-single* [*simp*]:
    $a \leq b$ ==> *partition*(*a*,*b*)(%*n*. *if n = 0 then a else b*)
by (*auto simp add*: *partition-def order-le-less*)

lemma *partition-lhs*: *partition*(*a*,*b*) *D* ==> (*D*(*0*) = *a*)
by (*simp add*: *partition-def*)

lemma *partition*:
      (*partition*(*a*,*b*) *D*) =
       ((*D 0* = *a*) &
        (∀ *n* < *psize D*. *D n* < *D*(*Suc n*)) &
        (∀ *n* ≥ *psize D*. *D n* = *b*))
apply (*simp add*: *partition-def*, *auto*)
apply (*subgoal-tac* [!] *psize D* = *N*, *auto*)
apply (*simp-all* (*no-asm*) *add*: *psize-def*)
apply (*rule-tac* [!] *some-equality*, *blast*)
  prefer *2* **apply** *blast*
apply (*rule-tac* [!] *ccontr*)
apply (*simp-all add*: *linorder-neq-iff*, *safe*)
apply (*drule-tac x = Na* **in** *spec*)
apply (*rotate-tac 3*)
apply (*drule-tac x = Suc Na* **in** *spec*, *simp*)
apply (*rotate-tac 2*)
apply (*drule-tac x = N* **in** *spec*, *simp*)
apply (*drule-tac x = Na* **in** *spec*)
apply (*drule-tac x = Suc Na* **and** *P* = %*n*. *Na*≤*n* ⟶ *D n* = *D Na* **in** *spec*, *auto*)
done

lemma *partition-rhs*: *partition*(*a*,*b*) *D* ==> (*D*(*psize D*) = *b*)
by (*simp add*: *partition*)

lemma *partition-rhs2*: [|*partition*(*a*,*b*) *D*; *psize D* ≤ *n* |] ==> (*D n* = *b*)
by (*simp add*: *partition*)

lemma *lemma-partition-lt-gen* [*rule-format*]:
 partition(*a*,*b*) *D* & *m* + *Suc d* ≤ *n* & *n* ≤ (*psize D*) --> *D*(*m*) < *D*(*m* + *Suc d*)
apply (*induct d*, *auto simp add*: *partition*)
apply (*blast dest*: *Suc-le-lessD* *intro*: *less-le-trans order-less-trans*)
done

lemma *less-eq-add-Suc*: *m* < *n* ==> ∃ *d*. *n* = *m* + *Suc d*
by (*auto simp add*: *less-iff-Suc-add*)

lemma *partition-lt-gen*:
[|*partition*(*a*,*b*) *D*; *m* < *n*; *n* ≤ (*psize D*)|] ==> *D*(*m*) < *D*(*n*)
by (*auto dest*: *less-eq-add-Suc intro*: *lemma-partition-lt-gen*)

lemma *partition-lt*: *partition*(*a*,*b*) *D* ==> *n* < (*psize D*) ==> *D*(*0*) < *D*(*Suc n*)
apply (*induct n*)
apply (*auto simp add*: *partition*)
done

lemma *partition-le*: *partition*(*a*,*b*) *D* ==> *a* ≤ *b*
apply (*frule partition* [*THEN iffD1*], *safe*)
apply (*drule-tac x* = *psize D* **and** *P*=%*n*. *psize D* ≤ *n* −−> *?P n* **in** *spec*, *safe*)
apply (*case-tac psize D* = *0*)
apply (*drule-tac* [*2*] *n* = *psize D* − *1* **in** *partition-lt*, *auto*)
done

lemma *partition-gt*: [|*partition*(*a*,*b*) *D*; *n* < (*psize D*)|] ==> *D*(*n*) < *D*(*psize D*)
by (*auto intro*: *partition-lt-gen*)

lemma *partition-eq*: *partition*(*a*,*b*) *D* ==> ((*a* = *b*) = (*psize D* = *0*))
apply (*frule partition* [*THEN iffD1*], *safe*)
apply (*rotate-tac 2*)
apply (*drule-tac x* = *psize D* **in** *spec*)
apply (*rule ccontr*)
apply (*drule-tac n* = *psize D* − *1* **in** *partition-lt*)
apply *auto*
done

lemma *partition-lb*: *partition*(*a*,*b*) *D* ==> *a* ≤ *D*(*r*)
apply (*frule partition* [*THEN iffD1*], *safe*)
apply (*induct r*)
apply (*cut-tac* [*2*] *y* = *Suc r* **and** *x* = *psize D* **in** *linorder-le-less-linear*)
apply (*auto intro*: *partition-le*)
apply (*drule-tac x* = *r* **in** *spec*)
apply *arith*
done

lemma *partition-lb-lt*: [| *partition*(*a*,*b*) *D*; *psize D* ~= *0* |] ==> *a* < *D*(*Suc n*)
apply (*rule-tac t* = *a* **in** *partition-lhs* [*THEN subst*], *assumption*)
apply (*cut-tac x* = *Suc n* **and** *y* = *psize D* **in** *linorder-le-less-linear*)
apply (*frule partition* [*THEN iffD1*], *safe*)
apply (*blast intro*: *partition-lt less-le-trans*)
apply (*rotate-tac 3*)
apply (*drule-tac x* = *Suc n* **in** *spec*)
apply (*erule impE*)
apply (*erule less-imp-le*)
apply (*frule partition-rhs*)
apply (*drule partition-gt*[*of - - - 0*], *arith*)

apply (*simp* (*no-asm-simp*))
done

lemma *partition-ub*: *partition*(*a*,*b*) *D* ==> *D*(*r*) ≤ *b*
apply (*frule partition* [*THEN iffD1*])
apply (*cut-tac x* = *psize D* **and** *y* = *r* **in** *linorder-le-less-linear*, *safe*, *blast*)
apply (*subgoal-tac* ∀*x*. *D* ((*psize D*) − *x*) ≤ *b*)
apply (*rotate-tac 4*)
apply (*drule-tac x* = *psize D* − *r* **in** *spec*)
apply (*subgoal-tac psize D* − (*psize D* − *r*) = *r*)
apply *simp*
apply *arith*
apply *safe*
apply (*induct-tac x*)
apply (*simp* (*no-asm*), *blast*)
apply (*case-tac psize D* − *Suc n* = *0*)
apply (*erule-tac V* = ∀*n*. *psize D* ≤ *n* −−> *D n* = *b* **in** *thin-rl*)
apply (*simp* (*no-asm-simp*) *add*: *partition-le*)
apply (*rule order-trans*)
prefer *2* **apply** *assumption*
apply (*subgoal-tac psize D* − *n* = *Suc* (*psize D* − *Suc n*))
prefer *2* **apply** *arith*
apply (*drule-tac x* = *psize D* − *Suc n* **in** *spec*, *simp*)
done

lemma *partition-ub-lt*: [| *partition*(*a*,*b*) *D*; *n* < *psize D* |] ==> *D*(*n*) < *b*
by (*blast intro*: *partition-rhs* [*THEN subst*] *partition-gt*)

lemma *lemma-partition-append1*:
[| *partition* (*a*, *b*) *D1*; *partition* (*b*, *c*) *D2* |]
==> (∀*n* < *psize D1* + *psize D2*.
(*if n* < *psize D1 then D1 n else D2* (*n* − *psize D1*))
< (*if Suc n* < *psize D1 then D1* (*Suc n*)
else D2 (*Suc n* − *psize D1*))) &
(∀*n* ≥ *psize D1* + *psize D2*.
(*if n* < *psize D1 then D1 n else D2* (*n* − *psize D1*)) =
(*if psize D1* + *psize D2* < *psize D1 then D1* (*psize D1* + *psize D2*)
else D2 (*psize D1* + *psize D2* − *psize D1*)))
apply (*auto intro*: *partition-lt-gen*)
apply (*subgoal-tac psize D1* = *Suc n*)
apply (*auto intro*!: *partition-lt-gen simp add*: *partition-lhs partition-ub-lt*)
apply (*auto intro*!: *partition-rhs2 simp add*: *partition-rhs*
split: *nat-diff-split*)
done

lemma *lemma-psize1*:
[| *partition* (*a*, *b*) *D1*; *partition* (*b*, *c*) *D2*; *N* < *psize D1* |]
==> *D1*(*N*) < *D2* (*psize D2*)
apply (*rule-tac y* = *D1* (*psize D1*) **in** *order-less-le-trans*)

apply (*erule partition-gt*)
apply (*auto simp add*: *partition-rhs partition-le*)
done

lemma *lemma-partition-append2*:
[| *partition* (*a*, *b*) *D1*; *partition* (*b*, *c*) *D2* |]
==> *psize* (%*n*. *if n* < *psize D1 then D1 n else D2* (*n* − *psize D1*)) =
psize D1 + *psize D2*
apply (*unfold psize-def*
[*of* %*n*. *if n* < *psize D1 then D1 n else D2* (*n* − *psize D1*)])
apply (*rule some1-equality*)
prefer *2* **apply** (*blast intro*!: *lemma-partition-append1*)
apply (*rule ex1I*, *rule lemma-partition-append1*)
apply (*simp-all split*: *split-if-asm*)

The case *N* < *psize D1*

apply (*drule-tac x* = *psize D1* + *psize D2* **and** *P*=%*n*. *?P n* & *?Q n* **in** *spec*)
apply (*force dest*: *lemma-psize1*)
apply (*rule order-antisym*)

The case *psize D1* ≤ *N*: proving *N* ≤ *psize D1* + *psize D2*

apply (*drule-tac x* = *psize D1* + *psize D2* **in** *spec*)
apply (*simp add*: *partition-rhs2*)
apply (*case-tac N* − *psize D1* < *psize D2*)
prefer *2* **apply** *arith*

Proving *psize D1* + *psize D2* ≤ *N*

apply (*drule-tac x* = *psize D1* + *psize D2* **and** *P*=%*n*. *N*≤*n* −−> *?P n* **in** *spec*, *simp*)
apply (*drule-tac a* = *b* **and** *b* = *c* **in** *partition-gt*, *assumption*, *simp*)
done

lemma *tpart-eq-lhs-rhs*: [|*psize D* = *0*; *tpart*(*a*,*b*) (*D*,*p*)|] ==> *a* = *b*
by (*auto simp add*: *tpart-def partition-eq*)

lemma *tpart-partition*: *tpart*(*a*,*b*) (*D*,*p*) ==> *partition*(*a*,*b*) *D*
by (*simp add*: *tpart-def*)

lemma *partition-append*:
[| *tpart*(*a*,*b*) (*D1*,*p1*); *fine*(*g*) (*D1*,*p1*);
tpart(*b*,*c*) (*D2*,*p2*); *fine*(*g*) (*D2*,*p2*) |]
==> ∃ *D p*. *tpart*(*a*,*c*) (*D*,*p*) & *fine*(*g*) (*D*,*p*)
apply (*rule-tac x* = %*n*. *if n* < *psize D1 then D1 n else D2* (*n* − *psize D1*)
in *exI*)
apply (*rule-tac x* = %*n*. *if n* < *psize D1 then p1 n else p2* (*n* − *psize D1*)
in *exI*)
apply (*case-tac psize D1* = *0*)
apply (*auto dest*: *tpart-eq-lhs-rhs*)
prefer *2*

apply (*simp add*: *fine-def*
lemma-partition-append2 [*OF tpart-partition tpart-partition*])
— But must not expand *fine* in other subgoals
apply *auto*
apply (*subgoal-tac psize D1 = Suc n*)
prefer *2* **apply** *arith*
apply (*drule tpart-partition* [*THEN partition-rhs*])
apply (*drule tpart-partition* [*THEN partition-lhs*])
apply (*auto split*: *nat-diff-split*)
apply (*auto simp add*: *tpart-def*)
defer *1*
apply (*subgoal-tac psize D1 = Suc n*)
prefer *2* **apply** *arith*
apply (*drule partition-rhs*)
apply (*drule partition-lhs*, *auto*)
apply (*simp split*: *nat-diff-split*)
apply (*subst partition*)
apply (*subst* (*1 2*) *lemma-partition-append2*, *assumption*+)
apply (*rule conjI*)
apply (*simp add*: *partition-lhs*)
apply (*drule lemma-partition-append1*)
apply *assumption*
apply (*simp add*: *partition-rhs*)
done

We can always find a division that is fine wrt any gauge

lemma *partition-exists*:
[| $a \le b$; *gauge*(%x. $a \le x$ & $x \le b$) g |]
==> $\exists D\ p$. *tpart*(a,b) (D,p) & *fine* g (D,p)
apply (*cut-tac* P = %(u,v). $a \le u$ & $v \le b$ −−>
($\exists D\ p$. *tpart* (u,v) (D,p) & *fine* (g) (D,p))
in *lemma-BOLZANO2*)
apply *safe*
apply (*blast intro*: *order-trans*)+
apply (*auto intro*: *partition-append*)
apply (*case-tac* $a \le x$ & $x \le b$)
apply (*rule-tac* [*2*] x = *1* **in** *exI*, *auto*)
apply (*rule-tac* x = $g\ x$ **in** *exI*)
apply (*auto simp add*: *gauge-def*)
apply (*rule-tac* x = %n. *if* n = *0 then aa else ba* **in** *exI*)
apply (*rule-tac* x = %n. *if* n = *0 then x else ba* **in** *exI*)
apply (*auto simp add*: *tpart-def fine-def*)
done

Lemmas about combining gauges

lemma *gauge-min*:
[| *gauge*(E) *g1*; *gauge*(E) *g2* |]
==> *gauge*(E) (%x. *if g1*(x) < *g2*(x) *then g1*(x) *else g2*(x))
by (*simp add*: *gauge-def*)

lemma *fine-min*:
fine (%*x*. *if g1*(*x*) < *g2*(*x*) *then g1*(*x*) *else g2*(*x*)) (*D*,*p*)
==> *fine*(*g1*) (*D*,*p*) & *fine*(*g2*) (*D*,*p*)
by (*auto simp add*: *fine-def split*: *split-if-asm*)

The integral is unique if it exists

lemma *Integral-unique*:
[| *a* ≤ *b*; *Integral*(*a*,*b*) *f k1*; *Integral*(*a*,*b*) *f k2* |] ==> *k1* = *k2*
apply (*simp add*: *Integral-def*)
apply (*drule-tac x* = |*k1* − *k2*| /*2* **in** *spec*)+
apply *auto*
apply (*drule gauge-min*, *assumption*)
apply (*drule-tac g* = %*x*. *if g x* < *ga x then g x else ga x*
in *partition-exists*, *assumption*, *auto*)
apply (*drule fine-min*)
apply (*drule spec*)+
apply *auto*
apply (*subgoal-tac* |(*rsum* (*D*,*p*) *f* − *k2*) − (*rsum* (*D*,*p*) *f* − *k1*)| < |*k1* − *k2*|)
apply *arith*
apply (*drule add-strict-mono*, *assumption*)
apply (*auto simp only*: *left-distrib* [*symmetric*] *mult-2-right* [*symmetric*]
mult-less-cancel-right)
done

lemma *Integral-zero* [*simp*]: *Integral*(*a*,*a*) *f 0*
apply (*auto simp add*: *Integral-def*)
apply (*rule-tac x* = %*x*. *1* **in** *exI*)
apply (*auto dest*: *partition-eq simp add*: *gauge-def tpart-def rsum-def*)
done

lemma *sumr-partition-eq-diff-bounds* [*simp*]:
(∑ *n*=*0*..<*m*. *D* (*Suc n*) − *D n*::*real*) = *D*(*m*) − *D 0*
by (*induct m*, *auto*)

lemma *Integral-eq-diff-bounds*: *a* ≤ *b* ==> *Integral*(*a*,*b*) (%*x*. *1*) (*b* − *a*)
apply (*auto simp add*: *order-le-less rsum-def Integral-def*)
apply (*rule-tac x* = %*x*. *b* − *a* **in** *exI*)
apply (*auto simp add*: *gauge-def abs-less-iff tpart-def partition*)
done

lemma *Integral-mult-const*: *a* ≤ *b* ==> *Integral*(*a*,*b*) (%*x*. *c*) (*c*∗(*b* − *a*))
apply (*auto simp add*: *order-le-less rsum-def Integral-def*)
apply (*rule-tac x* = %*x*. *b* − *a* **in** *exI*)
apply (*auto simp add*: *setsum-right-distrib* [*symmetric*] *gauge-def abs-less-iff*
right-diff-distrib [*symmetric*] *partition tpart-def*)
done

lemma *Integral-mult*:

[| $a \leq b$; *Integral*(a,b) f k |] ==> *Integral*(a,b) (%x. $c * f\ x$) $(c * k)$
apply (*auto simp add*: *order-le-less*
dest: *Integral-unique* [*OF order-refl Integral-zero*])
apply (*auto simp add*: *rsum-def Integral-def setsum-right-distrib*[*symmetric*] *mult-assoc*)
apply (*rule-tac a2* = *c* **in** *abs-ge-zero* [*THEN order-le-imp-less-or-eq*, *THEN disjE*])
prefer *2* **apply** *force*
apply (*drule-tac x* = *e*/*abs c* **in** *spec*, *auto*)
apply (*simp add*: *zero-less-mult-iff divide-inverse*)
apply (*rule exI*, *auto*)
apply (*drule spec*)+
apply *auto*
apply (*rule-tac z1* = *inverse* (*abs c*) **in** *real-mult-less-iff1* [*THEN iffD1*])
apply (*auto simp add*: *abs-mult divide-inverse* [*symmetric*] *right-diff-distrib* [*symmetric*])
done

Fundamental theorem of calculus (Part I)

"Straddle Lemma" : Swartz and Thompson: AMM 95(7) 1988

lemma *choiceP*: $\forall x.\ P(x)$ --> $(\exists y.\ Q\ x\ y)$ ==> $\exists f.\ (\forall x.\ P(x)$ --> $Q\ x\ (f\ x))$
by (*insert bchoice* [*of Collect P Q*], *simp*)

lemma *strad1*:
$[\![\forall xa{::}real.\ xa \neq x \wedge |xa - x| < s \longrightarrow$
$|(f\ xa - f\ x)\ /\ (xa - x) - f'\ x| * 2 < e;$
$0 < e;\ a \leq x;\ x \leq b;\ 0 < s]\!]$
$\Longrightarrow \forall z.\ |z - x| < s$ -->$|f\ z - f\ x - f'\ x * (z - x)| * 2 \leq e * |z - x|$
apply *auto*
apply (*case-tac* $0 < |z - x|$)
prefer *2* **apply** (*simp add*: *zero-less-abs-iff*)
apply (*drule-tac x* = *z* **in** *spec*)
apply (*rule-tac z1* = $|$*inverse* $(z - x)|$
in *real-mult-le-cancel-iff2* [*THEN iffD1*])
apply *simp*
apply (*simp del*: *abs-inverse abs-mult add*: *abs-mult* [*symmetric*]
mult-assoc [*symmetric*])
apply (*subgoal-tac inverse* $(z - x) * (f\ z - f\ x - f'\ x * (z - x))$
$= (f\ z - f\ x)\ /\ (z - x) - f'\ x$)
apply (*simp add*: *abs-mult* [*symmetric*] *mult-ac diff-minus*)
apply (*subst mult-commute*)
apply (*simp add*: *left-distrib diff-minus*)
apply (*simp add*: *mult-assoc divide-inverse*)
apply (*simp add*: *left-distrib*)
done

lemma *lemma-straddle*:

```
    [| ∀x. a ≤ x & x ≤ b --> DERIV f x :> f'(x); 0 < e |]
     ==> ∃g. gauge(%x. a ≤ x & x ≤ b) g &
           (∀x u v. a ≤ u & u ≤ x & x ≤ v & v ≤ b & (v - u) < g(x)
             --> |(f(v) - f(u)) - (f'(x) * (v - u))| ≤ e * (v - u))
apply (simp add: gauge-def)
apply (subgoal-tac ∀x. a ≤ x & x ≤ b -->
      (∃d > 0. ∀u v. u ≤ x & x ≤ v & (v - u) < d -->
                  |(f (v) - f (u)) - (f' (x) * (v - u))| ≤ e * (v - u)))
apply (drule choiceP, auto)
apply (drule spec, auto)
apply (auto simp add: DERIV-iff2 LIM-def)
apply (drule-tac x = e/2 in spec, auto)
apply (frule strad1, assumption+)
apply (rule-tac x = s in exI, auto)
apply (rule-tac x = u and y = v in linorder-cases, auto)
apply (rule-tac y = |(f (v) - f (x)) - (f' (x) * (v - x))| +
                    |(f (x) - f (u)) - (f' (x) * (x - u))|
     in order-trans)
apply (rule abs-triangle-ineq [THEN [2] order-trans])
apply (simp add: right-diff-distrib)
apply (rule-tac t = e* (v - u) in real-sum-of-halves [THEN subst])
apply (rule add-mono)
apply (rule-tac y = (e/2) * |v - x| in order-trans)
 prefer 2 apply simp
apply (erule-tac [!] V= ∀x'. x' ~= x & |x' - x| < s --> ?P x' in thin-rl)
apply (drule-tac x = v in spec, simp add: times-divide-eq)
apply (drule-tac x = u in spec, auto)
apply (subgoal-tac |f u - f x - f' x * (u - x)| = |f x - f u - f' x * (x - u)|)
apply (rule order-trans)
apply (auto simp add: abs-le-iff)
apply (simp add: right-diff-distrib)
done

lemma FTC1: [|a ≤ b; ∀x. a ≤ x & x ≤ b --> DERIV f x :> f'(x) |]
          ==> Integral(a,b) f' (f(b) - f(a))
apply (drule order-le-imp-less-or-eq, auto)
apply (auto simp add: Integral-def)
apply (rule ccontr)
apply (subgoal-tac ∀e > 0. ∃g. gauge (%x. a ≤ x & x ≤ b) g & (∀D p. tpart (a,
b) (D, p) & fine g (D, p) --> |rsum (D, p) f' - (f b - f a)| ≤ e))
apply (rotate-tac 3)
apply (drule-tac x = e/2 in spec, auto)
apply (drule spec, auto)
apply ((drule spec)+, auto)
apply (drule-tac e = ea/ (b - a) in lemma-straddle)
apply (auto simp add: zero-less-divide-iff)
apply (rule exI)
apply (auto simp add: tpart-def rsum-def)
apply (subgoal-tac (∑ n=0..<psize D. f(D(Suc n)) - f(D n)) = f b - f a)
```

prefer *2*
apply (*cut-tac D = %n. f (D n)* **and** *m = psize D*
in *sumr-partition-eq-diff-bounds*)
apply (*simp add*: *partition-lhs partition-rhs*)
apply (*drule sym*, *simp*)
apply (*simp* (*no-asm*) *add*: *setsum-subtractf*[*symmetric*])
apply (*rule setsum-abs* [*THEN order-trans*])
apply (*subgoal-tac ea = (∑ n=0..<psize D. (ea / (b − a)) ∗ (D (Suc n) − (D n))))*
apply (*simp add*: *abs-minus-commute*)
apply (*rule-tac t = ea* **in** *ssubst*, *assumption*)
apply (*rule setsum-mono*)
apply (*rule-tac* [*2*] *setsum-right-distrib* [*THEN subst*])
apply (*auto simp add*: *partition-rhs partition-lhs partition-lb partition-ub fine-def*)
done

lemma *Integral-subst*: [| *Integral*(*a*,*b*) *f k1*; *k2=k1* |] ==> *Integral*(*a*,*b*) *f k2*
by *simp*

lemma *Integral-add*:
[| *a* ≤ *b*; *b* ≤ *c*; *Integral*(*a*,*b*) *f′ k1*; *Integral*(*b*,*c*) *f′ k2*;
∀ *x*. *a* ≤ *x* & *x* ≤ *c* −−> *DERIV f x* :> *f′ x* |]
==> *Integral*(*a*,*c*) *f′* (*k1* + *k2*)
apply (*rule FTC1* [*THEN Integral-subst*], *auto*)
apply (*frule FTC1*, *auto*)
apply (*frule-tac a = b* **in** *FTC1*, *auto*)
apply (*drule-tac x = x* **in** *spec*, *auto*)
apply (*drule-tac ?k2.0 = f b − f a* **in** *Integral-unique*)
apply (*drule-tac* [*3*] *?k2.0 = f c − f b* **in** *Integral-unique*, *auto*)
done

lemma *partition-psize-Least*:
partition(*a*,*b*) *D* ==> *psize D* = (*LEAST n*. *D*(*n*) = *b*)
apply (*auto intro*!: *Least-equality* [*symmetric*] *partition-rhs*)
apply (*auto dest*: *partition-ub-lt simp add*: *linorder-not-less* [*symmetric*])
done

lemma *lemma-partition-bounded*: *partition* (*a*, *c*) *D* ==> ~ (∃ *n*. *c* < *D*(*n*))
apply *safe*
apply (*drule-tac r = n* **in** *partition-ub*, *auto*)
done

lemma *lemma-partition-eq*:
partition (*a*, *c*) *D* ==> *D* = (%*n*. *if D n* < *c then D n else c*)
apply (*rule ext*, *auto*)
apply (*auto dest*!: *lemma-partition-bounded*)
apply (*drule-tac x = n* **in** *spec*, *auto*)

done

lemma *lemma-partition-eq2*:
  partition (*a*, *c*) *D* ==> *D* = (%*n*. *if D n* ≤ *c then D n else c*)
apply (*rule ext*, *auto*)
apply (*auto dest*!: *lemma-partition-bounded*)
apply (*drule-tac x* = *n* **in** *spec*, *auto*)
done

lemma *partition-lt-Suc*:
  [| *partition*(*a*,*b*) *D*; *n* < *psize D* |] ==> *D n* < *D* (*Suc n*)
by (*auto simp add*: *partition*)

lemma *tpart-tag-eq*: *tpart*(*a*,*c*) (*D*,*p*) ==> *p* = (%*n*. *if D n* < *c then p n else c*)
apply (*rule ext*)
apply (*auto simp add*: *tpart-def*)
apply (*drule linorder-not-less* [*THEN iffD1*])
apply (*drule-tac r* = *Suc n* **in** *partition-ub*)
apply (*drule-tac x* = *n* **in** *spec*, *auto*)
done

52.1 Lemmas for Additivity Theorem of Gauge Integral

lemma *lemma-additivity1*:
  [| *a* ≤ *D n*; *D n* < *b*; *partition*(*a*,*b*) *D* |] ==> *n* < *psize D*
by (*auto simp add*: *partition linorder-not-less* [*symmetric*])

lemma *lemma-additivity2*: [| *a* ≤ *D n*; *partition*(*a*,*D n*) *D* |] ==> *psize D* ≤ *n*
apply (*rule ccontr*, *drule not-leE*)
apply (*frule partition* [*THEN iffD1*], *safe*)
apply (*frule-tac r* = *Suc n* **in** *partition-ub*)
apply (*auto dest*!: *spec*)
done

lemma *partition-eq-bound*:
  [| *partition*(*a*,*b*) *D*; *psize D* < *m* |] ==> *D*(*m*) = *D*(*psize D*)
by (*auto simp add*: *partition*)

lemma *partition-ub2*: [| *partition*(*a*,*b*) *D*; *psize D* < *m* |] ==> *D*(*r*) ≤ *D*(*m*)
by (*simp add*: *partition partition-ub*)

lemma *tag-point-eq-partition-point*:
  [| *tpart*(*a*,*b*) (*D*,*p*); *psize D* ≤ *m* |] ==> *p*(*m*) = *D*(*m*)
apply (*simp add*: *tpart-def*, *auto*)
apply (*drule-tac x* = *m* **in** *spec*)
apply (*auto simp add*: *partition-rhs2*)
done

lemma *partition-lt-cancel*: [| *partition*(*a*,*b*) *D*; *D m* < *D n* |] ==> *m* < *n*

apply (*cut-tac less-linear* [*of n psize D*], *auto*)
apply (*cut-tac less-linear* [*of m n*])
apply (*cut-tac less-linear* [*of m psize D*])
apply (*auto dest*: *partition-gt*)
apply (*drule-tac n = m* **in** *partition-lt-gen*, *auto*)
apply (*frule partition-eq-bound*)
apply (*drule-tac* [*2*] *partition-gt*, *auto*)
apply (*metis dense-linear-order-class.dlo-simps*(*8*) *not-less partition-rhs partition-rhs2*)
apply (*metis le-less-trans dense-linear-order-class.dlo-simps*(*8*) *nat-le-linear partition-eq-bound partition-rhs2*)
done

lemma *lemma-additivity4-psize-eq*:
[| *a* ≤ *D n*; *D n* < *b*; *partition* (*a*, *b*) *D* |]
==> *psize* (%*x*. *if D x* < *D n then D*(*x*) *else D n*) = *n*
apply (*unfold psize-def*)
apply (*frule lemma-additivity1*)
apply (*assumption*, *assumption*)
apply (*rule some-equality*)
apply (*auto intro*: *partition-lt-Suc*)
apply (*drule-tac n = n* **in** *partition-lt-gen*, *assumption*)
apply (*arith*, *arith*)
apply (*cut-tac x = na* **and** *y = psize D* **in** *less-linear*)
apply (*auto dest*: *partition-lt-cancel*)
apply (*rule-tac x=N* **and** *y=n* **in** *linorder-cases*)
apply (*drule-tac x = n* **and** *P=%m. N* ≤ *m* −−> *?f m = ?g m* **in** *spec*, *simp*)
apply (*drule-tac n = n* **in** *partition-lt-gen*, *auto*)
apply (*drule-tac x = n* **in** *spec*)
apply (*simp split*: *split-if-asm*)
done

lemma *lemma-psize-left-less-psize*:
partition (*a*, *b*) *D*
==> *psize* (%*x*. *if D x* < *D n then D*(*x*) *else D n*) ≤ *psize D*
apply (*frule-tac r = n* **in** *partition-ub*)
apply (*drule-tac x = D n* **in** *order-le-imp-less-or-eq*)
apply (*auto simp add*: *lemma-partition-eq* [*symmetric*])
apply (*frule-tac r = n* **in** *partition-lb*)
apply (*drule* (*2*) *lemma-additivity4-psize-eq*)
apply (*rule ccontr*, *auto*)
apply (*frule-tac not-leE* [*THEN* [*2*] *partition-eq-bound*])
apply (*auto simp add*: *partition-rhs*)
done

lemma *lemma-psize-left-less-psize2*:
[| *partition*(*a*,*b*) *D*; *na* < *psize* (%*x*. *if D x* < *D n then D*(*x*) *else D n*) |]
==> *na* < *psize D*
by (*erule lemma-psize-left-less-psize* [*THEN* [*2*] *less-le-trans*])

lemma *lemma-additivity3*:
[| *partition*(*a*,*b*) *D*; *D na* < *D n*; *D n* < *D* (*Suc na*);
n < *psize D* |]
==> *False*
by (*metis not-less-eq partition-lt-cancel real-of-nat-less-iff*)

lemma *psize-const* [*simp*]: *psize* (%*x*. *k*) = *0*
by (*auto simp add*: *psize-def*)

lemma *lemma-additivity3a*:
[| *partition*(*a*,*b*) *D*; *D na* < *D n*; *D n* < *D* (*Suc na*);
na < *psize D* |]
==> *False*
apply (*frule-tac m* = *n* **in** *partition-lt-cancel*)
apply (*auto intro*: *lemma-additivity3*)
done

lemma *better-lemma-psize-right-eq1*:
[| *partition*(*a*,*b*) *D*; *D n* < *b* |] ==> *psize* (%*x*. *D* (*x* + *n*)) ≤ *psize D* − *n*
apply (*simp add*: *psize-def* [*of* (%*x*. *D* (*x* + *n*))])
apply (*rule-tac a* = *psize D* − *n* **in** *someI2*, *auto*)
apply (*simp add*: *partition less-diff-conv*)
apply (*simp add*: *le-diff-conv partition-rhs2 split*: *nat-diff-split*)
apply (*drule-tac x* = *psize D* − *n* **in** *spec*, *auto*)
apply (*frule partition-rhs*, *safe*)
apply (*frule partition-lt-cancel*, *assumption*)
apply (*drule partition* [*THEN iffD1*], *safe*)
apply (*subgoal-tac* ~ *D* (*psize D* − *n* + *n*) < *D* (*Suc* (*psize D* − *n* + *n*)))
apply *blast*
apply (*drule-tac x* = *Suc* (*psize D*) **and** *P*=%*n*. *?P n* ⟶ *D n* = *D* (*psize D*)
in *spec*)
apply *simp*
done

lemma *psize-le-n*: *partition* (*a*, *D n*) *D* ==> *psize D* ≤ *n*
apply (*rule ccontr*, *drule not-leE*)
apply (*frule partition-lt-Suc*, *assumption*)
apply (*frule-tac r* = *Suc n* **in** *partition-ub*, *auto*)
done

lemma *better-lemma-psize-right-eq1a*:
partition(*a*,*D n*) *D* ==> *psize* (%*x*. *D* (*x* + *n*)) ≤ *psize D* − *n*
apply (*simp add*: *psize-def* [*of* (%*x*. *D* (*x* + *n*))])
apply (*rule-tac a* = *psize D* − *n* **in** *someI2*, *auto*)
apply (*simp add*: *partition less-diff-conv*)
apply (*simp add*: *le-diff-conv*)
apply (*case-tac psize D* ≤ *n*)

apply (*force intro*: *partition-rhs2*)
apply (*simp add*: *partition linorder-not-le*)
apply (*rule ccontr*, *drule not-leE*)
apply (*frule psize-le-n*)
apply (*drule-tac* $x = psize\ D - n$ **in** *spec*, *simp*)
apply (*drule partition* [*THEN iffD1*], *safe*)
apply (*drule-tac* $x = Suc\ n$ **and** $P = \%na.\ ?s \leq na \longrightarrow D\ na = D\ n$ **in** *spec*, *auto*)
done

lemma *better-lemma-psize-right-eq*:
$partition(a,b)\ D \Longrightarrow psize\ (\%x.\ D\ (x + n)) \leq psize\ D - n$
apply (*frule-tac r1* $= n$ **in** *partition-ub* [*THEN order-le-imp-less-or-eq*])
apply (*blast intro*: *better-lemma-psize-right-eq1a better-lemma-psize-right-eq1*)
done

lemma *lemma-psize-right-eq1*:
$[|\ partition(a,b)\ D;\ D\ n < b\ |] \Longrightarrow psize\ (\%x.\ D\ (x + n)) \leq psize\ D$
apply (*simp add*: *psize-def* [*of* $(\%x.\ D\ (x + n))$])
apply (*rule-tac* $a = psize\ D - n$ **in** *someI2*, *auto*)
apply (*simp add*: *partition less-diff-conv*)
apply (*subgoal-tac* $n \leq psize\ D$)
apply (*simp add*: *partition le-diff-conv*)
apply (*rule ccontr*, *drule not-leE*)
apply (*drule-tac less-imp-le* [*THEN* [*2*] *partition-rhs2*], *assumption*, *simp*)
apply (*drule-tac* $x = psize\ D$ **in** *spec*)
apply (*simp add*: *partition*)
done

lemma *lemma-psize-right-eq1a*:
$partition(a,D\ n)\ D \Longrightarrow psize\ (\%x.\ D\ (x + n)) \leq psize\ D$
apply (*simp add*: *psize-def* [*of* $(\%x.\ D\ (x + n))$])
apply (*rule-tac* $a = psize\ D - n$ **in** *someI2*, *auto*)
apply (*simp add*: *partition less-diff-conv*)
apply (*case-tac psize* $D \leq n$)
apply (*force intro*: *partition-rhs2 simp add*: *le-diff-conv*)
apply (*simp add*: *partition le-diff-conv*)
apply (*rule ccontr*, *drule not-leE*)
apply (*drule-tac* $x = psize\ D$ **in** *spec*)
apply (*simp add*: *partition*)
done

lemma *lemma-psize-right-eq*:
$[|\ partition(a,b)\ D\ |] \Longrightarrow psize\ (\%x.\ D\ (x + n)) \leq psize\ D$
apply (*frule-tac r1* $= n$ **in** *partition-ub* [*THEN order-le-imp-less-or-eq*])
apply (*blast intro*: *lemma-psize-right-eq1a lemma-psize-right-eq1*)
done

lemma *tpart-left1*:

```
    [| a ≤ D n; tpart (a, b) (D, p) |]
     ==> tpart(a, D n) (%x. if D x < D n then D(x) else D n,
        %x. if D x < D n then p(x) else D n)
apply (frule-tac r = n in tpart-partition [THEN partition-ub])
apply (drule-tac x = D n in order-le-imp-less-or-eq)
apply (auto simp add: tpart-partition [THEN lemma-partition-eq, symmetric] tpart-tag-eq
[symmetric])
apply (frule-tac tpart-partition [THEN [3] lemma-additivity1])
apply (auto simp add: tpart-def)
apply (drule-tac [2] linorder-not-less [THEN iffD1, THEN order-le-imp-less-or-eq],
auto)
 prefer 3 apply (drule-tac x=na in spec, arith)
prefer 2 apply (blast dest: lemma-additivity3)
apply (frule (2) lemma-additivity4-psize-eq)
apply (rule partition [THEN iffD2])
apply (frule partition [THEN iffD1])
apply safe
apply (auto simp add: partition-lt-gen)
apply (drule (1) partition-lt-cancel, arith)
done

lemma fine-left1:
    [| a ≤ D n; tpart (a, b) (D, p); gauge (%x. a ≤ x & x ≤ D n) g;
       fine (%x. if x < D n then min (g x) ((D n - x)/ 2)
              else if x = D n then min (g (D n)) (ga (D n))
                   else min (ga x) ((x - D n)/ 2)) (D, p) |]
     ==> fine g
         (%x. if D x < D n then D(x) else D n,
          %x. if D x < D n then p(x) else D n)
apply (auto simp add: fine-def tpart-def gauge-def)
apply (frule-tac [!] na=na in lemma-psize-left-less-psize2)
apply (drule-tac [!] x = na in spec, auto)
apply (drule-tac [!] x = na in spec, auto)
apply (auto dest: lemma-additivity3a simp add: split-if-asm)
done

lemma tpart-right1:
    [| a ≤ D n; tpart (a, b) (D, p) |]
    ==> tpart(D n, b) (%x. D(x + n),%x. p(x + n))
apply (simp add: tpart-def partition-def, safe)
apply (rule-tac x = N - n in exI, auto)
done

lemma fine-right1:
    [| a ≤ D n; tpart (a, b) (D, p); gauge (%x. D n ≤ x & x ≤ b) ga;
       fine (%x. if x < D n then min (g x) ((D n - x)/ 2)
              else if x = D n then min (g (D n)) (ga (D n))
                   else min (ga x) ((x - D n)/ 2)) (D, p) |]
     ==> fine ga (%x. D(x + n),%x. p(x + n))
```

apply (*auto simp add*: *fine-def gauge-def*)
apply (*drule-tac x = na + n* **in** *spec*)
apply (*frule-tac n = n* **in** *tpart-partition* [*THEN better-lemma-psize-right-eq*], *auto*)
apply (*simp add*: *tpart-def*, *safe*)
apply (*subgoal-tac D n ≤ p (na + n)*)
apply (*drule-tac y = p (na + n)* **in** *order-le-imp-less-or-eq*)
apply *safe*
apply (*simp split*: *split-if-asm*, *simp*)
apply (*drule less-le-trans*, *assumption*)
apply (*rotate-tac 5*)
apply (*drule-tac x = na + n* **in** *spec*, *safe*)
apply (*rule-tac y=D (na + n)* **in** *order-trans*)
apply (*case-tac na = 0*, *auto*)
apply (*erule partition-lt-gen* [*THEN order-less-imp-le*])
apply *arith*
apply *arith*
done

lemma *rsum-add*: *rsum (D, p) (%x. f x + g x) = rsum (D, p) f + rsum(D, p) g*
by (*simp add*: *rsum-def setsum-addf left-distrib*)

Bartle/Sherbert: Theorem 10.1.5 p. 278

lemma *Integral-add-fun*:
[| *a ≤ b*; *Integral(a,b) f k1*; *Integral(a,b) g k2* |]
==> *Integral(a,b) (%x. f x + g x) (k1 + k2)*
apply (*simp add*: *Integral-def*, *auto*)
apply ((*drule-tac x = e/2* **in** *spec*)+)
apply *auto*
apply (*drule gauge-min*, *assumption*)
apply (*rule-tac x = (%x. if ga x < gaa x then ga x else gaa x)* **in** *exI*)
apply *auto*
apply (*drule fine-min*)
apply ((*drule spec*)+, *auto*)
apply (*drule-tac a = |rsum (D, p) f − k1| * 2* **and** *c = |rsum (D, p) g − k2| * 2* **in** *add-strict-mono*, *assumption*)
apply (*auto simp only*: *rsum-add left-distrib* [*symmetric*]
mult-2-right [*symmetric*] *real-mult-less-iff1*)
done

lemma *partition-lt-gen2*:
[| *partition(a,b) D*; *r < psize D* |] ==> *0 < D (Suc r) − D r*
by (*auto simp add*: *partition*)

lemma *lemma-Integral-le*:
[| ∀ *x. a ≤ x & x ≤ b −−> f x ≤ g x*;
tpart(a,b) (D,p)
|] ==> ∀ *n ≤ psize D. f (p n) ≤ g (p n)*
apply (*simp add*: *tpart-def*)

apply (*auto*, *frule partition* [*THEN iffD1*], *auto*)
apply (*drule-tac* $x = p\ n$ **in** *spec*, *auto*)
apply (*case-tac* $n = 0$, *simp*)
apply (*rule partition-lt-gen* [*THEN order-less-le-trans*, *THEN order-less-imp-le*], *auto*)
apply (*drule le-imp-less-or-eq*, *auto*)
apply (*drule-tac* [*2*] $x = psize\ D$ **in** *spec*, *auto*)
apply (*drule-tac* $r = Suc\ n$ **in** *partition-ub*)
apply (*drule-tac* $x = n$ **in** *spec*, *auto*)
done

lemma *lemma-Integral-rsum-le*:
$[\!|\ \forall x.\ a \leq x\ \&\ x \leq b\ \text{--}\!\!>\ f\ x \leq g\ x;$
$tpart(a,b)\ (D,p)$
$|\!]\ \Longrightarrow\ rsum(D,p)\ f \leq rsum(D,p)\ g$
apply (*simp add*: *rsum-def*)
apply (*auto intro*!: *setsum-mono dest*: *tpart-partition* [*THEN partition-lt-gen2*]
dest!: *lemma-Integral-le*)
done

lemma *Integral-le*:
$[\!|\ a \leq b;$
$\forall x.\ a \leq x\ \&\ x \leq b\ \text{--}\!\!>\ f(x) \leq g(x);$
$Integral(a,b)\ f\ k1;\ Integral(a,b)\ g\ k2$
$|\!]\ \Longrightarrow\ k1 \leq k2$
apply (*simp add*: *Integral-def*)
apply (*rotate-tac 2*)
apply (*drule-tac* $x = |k1 - k2|\ /2$ **in** *spec*)
apply (*drule-tac* $x = |k1 - k2|\ /2$ **in** *spec*, *auto*)
apply (*drule gauge-min*, *assumption*)
apply (*drule-tac* $g = \%x.$ *if ga x* $<$ *gaa x then ga x else gaa x*
in *partition-exists*, *assumption*, *auto*)
apply (*drule fine-min*)
apply (*drule-tac* $x = D$ **in** *spec*, *drule-tac* $x = D$ **in** *spec*)
apply (*drule-tac* $x = p$ **in** *spec*, *drule-tac* $x = p$ **in** *spec*, *auto*)
apply (*frule lemma-Integral-rsum-le*, *assumption*)
apply (*subgoal-tac* $|(rsum\ (D,p)\ f\ -\ k1)\ -\ (rsum\ (D,p)\ g\ -\ k2)| < |k1\ -\ k2|$)
apply *arith*
apply (*drule add-strict-mono*, *assumption*)
apply (*auto simp only*: *left-distrib* [*symmetric*] *mult-2-right* [*symmetric*]
real-mult-less-iff1)
done

lemma *Integral-imp-Cauchy*:
$(\exists k.\ Integral(a,b)\ f\ k) \Longrightarrow$
$(\forall e > 0.\ \exists g.\ gauge\ (\%x.\ a \leq x\ \&\ x \leq b)\ g\ \&$
$(\forall D1\ D2\ p1\ p2.$
$tpart(a,b)\ (D1,\ p1)\ \&\ fine\ g\ (D1,p1)\ \&$
$tpart(a,b)\ (D2,\ p2)\ \&\ fine\ g\ (D2,p2)\ \text{--}\!\!>$

```
                        |rsum(D1,p1) f - rsum(D2,p2) f| < e))
apply (simp add: Integral-def, auto)
apply (drule-tac x = e/2 in spec, auto)
apply (rule exI, auto)
apply (frule-tac x = D1 in spec)
apply (frule-tac x = D2 in spec)
apply ((drule spec)+, auto)
apply (erule-tac V = 0 < e in thin-rl)
apply (drule add-strict-mono, assumption)
apply (auto simp only: left-distrib [symmetric] mult-2-right [symmetric]
                  real-mult-less-iff1)
done

lemma Cauchy-iff2:
     Cauchy X =
     (∀j. (∃M. ∀m ≥ M. ∀n ≥ M. |X m - X n| < inverse(real (Suc j))))
apply (simp add: Cauchy-def, auto)
apply (drule reals-Archimedean, safe)
apply (drule-tac x = n in spec, auto)
apply (rule-tac x = M in exI, auto)
apply (drule-tac x = m in spec, simp)
apply (drule-tac x = na in spec, auto)
done

lemma partition-exists2:
     [| a ≤ b; ∀n. gauge (%x. a ≤ x & x ≤ b) (fa n) |]
     ==> ∀n. ∃D p. tpart (a, b) (D, p) & fine (fa n) (D, p)
by (blast dest: partition-exists)

lemma monotonic-anti-derivative:
  fixes f g :: real => real shows
     [| a ≤ b; ∀c. a ≤ c & c ≤ b --> f' c ≤ g' c;
        ∀x. DERIV f x :> f' x; ∀x. DERIV g x :> g' x |]
     ==> f b - f a ≤ g b - g a
apply (rule Integral-le, assumption)
apply (auto intro: FTC1)
done

end
```

53 Log: Logarithms: Standard Version

```
theory Log
imports Transcendental
begin

definition
  powr  :: [real,real] => real     (infixr powr 80) where
```

— exponentation with real exponent
*x powr a = exp(a * ln x)*

definition
log :: [real,real] => real **where**
— logarithm of *x* to base *a*
log a x = ln x / ln a

lemma *powr-one-eq-one* [*simp*]: *1 powr a = 1*
by (*simp add*: *powr-def*)

lemma *powr-zero-eq-one* [*simp*]: *x powr 0 = 1*
by (*simp add*: *powr-def*)

lemma *powr-one-gt-zero-iff* [*simp*]: *(x powr 1 = x) = (0 < x)*
by (*simp add*: *powr-def*)
declare *powr-one-gt-zero-iff* [*THEN iffD2*, *simp*]

lemma *powr-mult*:
*[| 0 < x; 0 < y |] ==> (x * y) powr a = (x powr a) * (y powr a)*
by (*simp add*: *powr-def exp-add* [*symmetric*] *ln-mult right-distrib*)

lemma *powr-gt-zero* [*simp*]: *0 < x powr a*
by (*simp add*: *powr-def*)

lemma *powr-ge-pzero* [*simp*]: *0 <= x powr y*
by (*rule order-less-imp-le*, *rule powr-gt-zero*)

lemma *powr-not-zero* [*simp*]: *x powr a ≠ 0*
by (*simp add*: *powr-def*)

lemma *powr-divide*:
[| 0 < x; 0 < y |] ==> (x / y) powr a = (x powr a)/(y powr a)
apply (*simp add*: *divide-inverse positive-imp-inverse-positive powr-mult*)
apply (*simp add*: *powr-def exp-minus* [*symmetric*] *exp-add* [*symmetric*] *ln-inverse*)
done

lemma *powr-divide2*: *x powr a / x powr b = x powr (a - b)*
apply (*simp add*: *powr-def*)
apply (*subst exp-diff* [*THEN sym*])
apply (*simp add*: *left-diff-distrib*)
done

lemma *powr-add*: *x powr (a + b) = (x powr a) * (x powr b)*
by (*simp add*: *powr-def exp-add* [*symmetric*] *left-distrib*)

lemma *powr-powr*: *(x powr a) powr b = x powr (a * b)*

by (*simp add*: *powr-def*)

lemma *powr-powr-swap*: (*x powr a*) *powr b* = (*x powr b*) *powr a*
by (*simp add*: *powr-powr real-mult-commute*)

lemma *powr-minus*: *x powr* (−*a*) = *inverse* (*x powr a*)
by (*simp add*: *powr-def exp-minus* [*symmetric*])

lemma *powr-minus-divide*: *x powr* (−*a*) = *1*/(*x powr a*)
by (*simp add*: *divide-inverse powr-minus*)

lemma *powr-less-mono*: [| *a* < *b*; *1* < *x* |] ==> *x powr a* < *x powr b*
by (*simp add*: *powr-def*)

lemma *powr-less-cancel*: [| *x powr a* < *x powr b*; *1* < *x* |] ==> *a* < *b*
by (*simp add*: *powr-def*)

lemma *powr-less-cancel-iff* [*simp*]: *1* < *x* ==> (*x powr a* < *x powr b*) = (*a* < *b*)
by (*blast intro*: *powr-less-cancel powr-less-mono*)

lemma *powr-le-cancel-iff* [*simp*]: *1* < *x* ==> (*x powr a* ≤ *x powr b*) = (*a* ≤ *b*)
by (*simp add*: *linorder-not-less* [*symmetric*])

lemma *log-ln*: *ln x* = *log* (*exp*(*1*)) *x*
by (*simp add*: *log-def*)

lemma *powr-log-cancel* [*simp*]:
[| *0* < *a*; *a* ≠ *1*; *0* < *x* |] ==> *a powr* (*log a x*) = *x*
by (*simp add*: *powr-def log-def*)

lemma *log-powr-cancel* [*simp*]: [| *0* < *a*; *a* ≠ *1* |] ==> *log a* (*a powr y*) = *y*
by (*simp add*: *log-def powr-def*)

lemma *log-mult*:
[| *0* < *a*; *a* ≠ *1*; *0* < *x*; *0* < *y* |]
==> *log a* (*x* ∗ *y*) = *log a x* + *log a y*
by (*simp add*: *log-def ln-mult divide-inverse left-distrib*)

lemma *log-eq-div-ln-mult-log*:
[| *0* < *a*; *a* ≠ *1*; *0* < *b*; *b* ≠ *1*; *0* < *x* |]
==> *log a x* = (*ln b*/*ln a*) ∗ *log b x*
by (*simp add*: *log-def divide-inverse*)

Base 10 logarithms

lemma *log-base-10-eq1*: *0* < *x* ==> *log 10 x* = (*ln* (*exp 1*) / *ln 10*) ∗ *ln x*
by (*simp add*: *log-def*)

lemma *log-base-10-eq2*: *0* < *x* ==> *log 10 x* = (*log 10* (*exp 1*)) ∗ *ln x*
by (*simp add*: *log-def*)

lemma *log-one* [*simp*]: *log a 1 = 0*
by (*simp add*: *log-def*)

lemma *log-eq-one* [*simp*]: [| *0 < a*; *a ≠ 1* |] ==> *log a a = 1*
by (*simp add*: *log-def*)

lemma *log-inverse*:
    [| *0 < a*; *a ≠ 1*; *0 < x* |] ==> *log a* (*inverse x*) = *− log a x*
apply (*rule-tac a1 = log a x* **in** *add-left-cancel* [*THEN iffD1*])
apply (*simp add*: *log-mult* [*symmetric*])
done

lemma *log-divide*:
    [|*0 < a*; *a ≠ 1*; *0 < x*; *0 < y*|] ==> *log a* (*x*/*y*) = *log a x − log a y*
by (*simp add*: *log-mult divide-inverse log-inverse*)

lemma *log-less-cancel-iff* [*simp*]:
    [| *1 < a*; *0 < x*; *0 < y* |] ==> (*log a x < log a y*) = (*x < y*)
apply *safe*
apply (*rule-tac* [*2*] *powr-less-cancel*)
apply (*drule-tac a = log a x* **in** *powr-less-mono*, *auto*)
done

lemma *log-le-cancel-iff* [*simp*]:
    [| *1 < a*; *0 < x*; *0 < y* |] ==> (*log a x ≤ log a y*) = (*x ≤ y*)
by (*simp add*: *linorder-not-less* [*symmetric*])

lemma *powr-realpow*: *0 < x* ==> *x powr* (*real n*) = *x^n*
  apply (*induct n*, *simp*)
  apply (*subgoal-tac real*(*Suc n*) = *real n + 1*)
  apply (*erule ssubst*)
  apply (*subst powr-add*, *simp*, *simp*)
done

lemma *powr-realpow2*: *0 <= x* ==> *0 < n* ==> *x^n* = (*if* (*x = 0*) *then 0*
  else x powr (*real n*))
  apply (*case-tac x = 0*, *simp*, *simp*)
  apply (*rule powr-realpow* [*THEN sym*], *simp*)
done

lemma *ln-pwr*: *0 < x* ==> *0 < y* ==> *ln*(*x powr y*) = *y ∗ ln x*
by (*unfold powr-def*, *simp*)

lemma *ln-bound*: *1 <= x* ==> *ln x <= x*
  apply (*subgoal-tac ln*(*1 + *(*x − 1*)) *<= x − 1*)
  apply *simp*
  apply (*rule ln-add-one-self-le-self*, *simp*)

done

lemma *powr-mono*: *a <= b ==> 1 <= x ==> x powr a <= x powr b*
apply (*case-tac x = 1*, *simp*)
apply (*case-tac a = b*, *simp*)
apply (*rule order-less-imp-le*)
apply (*rule powr-less-mono*, *auto*)
done

lemma *ge-one-powr-ge-zero*: *1 <= x ==> 0 <= a ==> 1 <= x powr a*
apply (*subst powr-zero-eq-one* [*THEN sym*])
apply (*rule powr-mono*, *assumption+*)
done

lemma *powr-less-mono2*: *0 < a ==> 0 < x ==> x < y ==> x powr a < y powr a*
apply (*unfold powr-def*)
apply (*rule exp-less-mono*)
apply (*rule mult-strict-left-mono*)
apply (*subst ln-less-cancel-iff*, *assumption*)
apply (*rule order-less-trans*)
prefer *2*
apply *assumption+*
done

lemma *powr-less-mono2-neg*: *a < 0 ==> 0 < x ==> x < y ==> y powr a < x powr a*
apply (*unfold powr-def*)
apply (*rule exp-less-mono*)
apply (*rule mult-strict-left-mono-neg*)
apply (*subst ln-less-cancel-iff*)
apply *assumption*
apply (*rule order-less-trans*)
prefer *2*
apply *assumption+*
done

lemma *powr-mono2*: *0 <= a ==> 0 < x ==> x <= y ==> x powr a <= y powr a*
apply (*case-tac a = 0*, *simp*)
apply (*case-tac x = y*, *simp*)
apply (*rule order-less-imp-le*)
apply (*rule powr-less-mono2*, *auto*)
done

lemma *ln-powr-bound*: *1 <= x ==> 0 < a ==> ln x <= (x powr a) / a*
apply (*rule mult-imp-le-div-pos*)
apply (*assumption*)
apply (*subst mult-commute*)

apply (*subst ln-pwr* [*THEN sym*])
apply *auto*
apply (*rule ln-bound*)
apply (*erule ge-one-powr-ge-zero*)
apply (*erule order-less-imp-le*)
done

lemma *ln-powr-bound2*: *1 < x ==> 0 < a ==> (ln x) powr a <= (a powr a) * x*
proof −
assume *1 < x* **and** *0 < a*
then have *ln x <= (x powr (1 / a)) / (1 / a)*
apply (*intro ln-powr-bound*)
apply (*erule order-less-imp-le*)
apply (*rule divide-pos-pos*)
apply *simp-all*
done
also have *... = a * (x powr (1 / a))*
by *simp*
finally have *(ln x) powr a <= (a * (x powr (1 / a))) powr a*
apply (*intro powr-mono2*)
apply (*rule order-less-imp-le*, *rule prems*)
apply (*rule ln-gt-zero*)
apply (*rule prems*)
apply *assumption*
done
also have *... = (a powr a) * ((x powr (1 / a)) powr a)*
apply (*rule powr-mult*)
apply (*rule prems*)
apply (*rule powr-gt-zero*)
done
also have *(x powr (1 / a)) powr a = x powr ((1 / a) * a)*
by (*rule powr-powr*)
also have *... = x*
apply *simp*
apply (*subgoal-tac a ~= 0*)
apply (*insert prems*, *auto*)
done
finally show *?thesis* **.**
qed

lemma *LIMSEQ-neg-powr*: *0 < s ==> (%x. (real x) powr − s) −−−−> 0*
apply (*unfold LIMSEQ-def*)
apply *clarsimp*
apply (*rule-tac x = natfloor(r powr (1 / − s)) + 1* **in** *exI*)
apply *clarify*
proof −
fix *r* **fix** *n*
assume *0 < s* **and** *0 < r* **and** *natfloor (r powr (1 / − s)) + 1 <= n*

```
  have r powr (1 / − s) < real(natfloor(r powr (1 / − s))) + 1
    by (rule real-natfloor-add-one-gt)
  also have ... = real(natfloor(r powr (1 / −s)) + 1)
    by simp
  also have ... <= real n
    apply (subst real-of-nat-le-iff)
    apply (rule prems)
    done
  finally have r powr (1 / − s) < real n.
  then have real n powr (− s) < (r powr (1 / − s)) powr − s
    apply (intro powr-less-mono2-neg)
    apply (auto simp add: prems)
    done
  also have ... = r
    by (simp add: powr-powr prems less-imp-neq [THEN not-sym])
  finally show real n powr − s < r .
 qed

end
```

54 HLog: Logarithms: Non-Standard Version

```
theory HLog
imports Log HTranscendental
begin



lemma epsilon-ge-zero [simp]: 0 ≤ epsilon
by (simp add: epsilon-def star-n-zero-num star-n-le)

lemma hpfinite-witness: epsilon : {x. 0 ≤ x & x : HFinite}
by auto


definition
  powhr :: [hypreal,hypreal] => hypreal     (infixr powhr 80) where
  x powhr a = starfun2 (op powr) x a

definition
  hlog :: [hypreal,hypreal] => hypreal where
  hlog a x = starfun2 log a x

declare powhr-def [transfer-unfold]
declare hlog-def [transfer-unfold]

lemma powhr: (star-n X) powhr (star-n Y) = star-n (%n. (X n) powr (Y n))
by (simp add: powhr-def starfun2-star-n)
```

lemma *powhr-one-eq-one* [*simp*]: !!*a*. *1 powhr a = 1*
by (*transfer*, *simp*)

lemma *powhr-mult*:
!!*a x y*. [| *0 < x*; *0 < y* |] ==> (*x * y*) *powhr a* = (*x powhr a*) * (*y powhr a*)
by (*transfer*, *rule powr-mult*)

lemma *powhr-gt-zero* [*simp*]: !!*a x*. *0 < x powhr a*
by (*transfer*, *simp*)

lemma *powhr-not-zero* [*simp*]: *x powhr a ≠ 0*
by (*rule powhr-gt-zero* [*THEN hypreal-not-refl2*, *THEN not-sym*])

lemma *powhr-divide*:
!!*a x y*. [| *0 < x*; *0 < y* |] ==> (*x / y*) *powhr a* = (*x powhr a*)/(*y powhr a*)
by (*transfer*, *rule powr-divide*)

lemma *powhr-add*: !!*a b x*. *x powhr* (*a* + *b*) = (*x powhr a*) * (*x powhr b*)
by (*transfer*, *rule powr-add*)

lemma *powhr-powhr*: !!*a b x*. (*x powhr a*) *powhr b* = *x powhr* (*a * b*)
by (*transfer*, *rule powr-powr*)

lemma *powhr-powhr-swap*: !!*a b x*. (*x powhr a*) *powhr b* = (*x powhr b*) *powhr a*
by (*transfer*, *rule powr-powr-swap*)

lemma *powhr-minus*: !!*a x*. *x powhr* (−*a*) = *inverse* (*x powhr a*)
by (*transfer*, *rule powr-minus*)

lemma *powhr-minus-divide*: *x powhr* (−*a*) = *1*/(*x powhr a*)
by (*simp add*: *divide-inverse powhr-minus*)

lemma *powhr-less-mono*: !!*a b x*. [| *a < b*; *1 < x* |] ==> *x powhr a < x powhr b*
by (*transfer*, *simp*)

lemma *powhr-less-cancel*: !!*a b x*. [| *x powhr a < x powhr b*; *1 < x* |] ==> *a < b*
by (*transfer*, *simp*)

lemma *powhr-less-cancel-iff* [*simp*]:
1 < x ==> (*x powhr a < x powhr b*) = (*a < b*)
by (*blast intro*: *powhr-less-cancel powhr-less-mono*)

lemma *powhr-le-cancel-iff* [*simp*]:
1 < x ==> (*x powhr a* ≤ *x powhr b*) = (*a* ≤ *b*)
by (*simp add*: *linorder-not-less* [*symmetric*])

lemma *hlog*:
hlog (*star-n X*) (*star-n Y*) =

star-n (%*n*. *log* (*X n*) (*Y n*))
by (*simp add*: *hlog-def starfun2-star-n*)

lemma *hlog-starfun-ln*: !!*x*. (∗*f*∗ *ln*) *x* = *hlog* ((∗*f*∗ *exp*) *1*) *x*
by (*transfer*, *rule log-ln*)

lemma *powhr-hlog-cancel* [*simp*]:
!!*a x*. [| *0* < *a*; *a* ≠ *1*; *0* < *x* |] ==> *a powhr* (*hlog a x*) = *x*
by (*transfer*, *simp*)

lemma *hlog-powhr-cancel* [*simp*]:
!!*a y*. [| *0* < *a*; *a* ≠ *1* |] ==> *hlog a* (*a powhr y*) = *y*
by (*transfer*, *simp*)

lemma *hlog-mult*:
!!*a x y*. [| *0* < *a*; *a* ≠ *1*; *0* < *x*; *0* < *y* |]
==> *hlog a* (*x* ∗ *y*) = *hlog a x* + *hlog a y*
by (*transfer*, *rule log-mult*)

lemma *hlog-as-starfun*:
!!*a x*. [| *0* < *a*; *a* ≠ *1* |] ==> *hlog a x* = (∗*f*∗ *ln*) *x* / (∗*f*∗ *ln*) *a*
by (*transfer*, *simp add*: *log-def*)

lemma *hlog-eq-div-starfun-ln-mult-hlog*:
!!*a b x*. [| *0* < *a*; *a* ≠ *1*; *0* < *b*; *b* ≠ *1*; *0* < *x* |]
==> *hlog a x* = ((∗*f*∗ *ln*) *b*/(∗*f*∗*ln*) *a*) ∗ *hlog b x*
by (*transfer*, *rule log-eq-div-ln-mult-log*)

lemma *powhr-as-starfun*: !!*a x*. *x powhr a* = (∗*f*∗ *exp*) (*a* ∗ (∗*f*∗ *ln*) *x*)
by (*transfer*, *simp add*: *powr-def*)

lemma *HInfinite-powhr*:
[| *x* : *HInfinite*; *0* < *x*; *a* : *HFinite* − *Infinitesimal*;
0 < *a* |] ==> *x powhr a* : *HInfinite*
apply (*auto intro*!: *starfun-ln-ge-zero starfun-ln-HInfinite HInfinite-HFinite-not-Infinitesimal-mult2 starfun-exp-HInfinite*
simp add: *order-less-imp-le HInfinite-gt-zero-gt-one powhr-as-starfun zero-le-mult-iff*)
done

lemma *hlog-hrabs-HInfinite-Infinitesimal*:
[| *x* : *HFinite* − *Infinitesimal*; *a* : *HInfinite*; *0* < *a* |]
==> *hlog a* (*abs x*) : *Infinitesimal*
apply (*frule HInfinite-gt-zero-gt-one*)
apply (*auto intro*!: *starfun-ln-HFinite-not-Infinitesimal*
HInfinite-inverse-Infinitesimal Infinitesimal-HFinite-mult2
simp add: *starfun-ln-HInfinite not-Infinitesimal-not-zero*
hlog-as-starfun hypreal-not-refl2 [*THEN not-sym*] *divide-inverse*)
done

lemma *hlog-HInfinite-as-starfun*:
[| *a* : *HInfinite*; *0* < *a* |] ==> *hlog a x* = (**f** *ln*) *x* / (**f** *ln*) *a*
by (*rule hlog-as-starfun*, *auto*)

lemma *hlog-one* [*simp*]: !!*a*. *hlog a 1* = *0*
by (*transfer*, *simp*)

lemma *hlog-eq-one* [*simp*]: !!*a*. [| *0* < *a*; *a* ≠ *1* |] ==> *hlog a a* = *1*
by (*transfer*, *rule log-eq-one*)

lemma *hlog-inverse*:
[| *0* < *a*; *a* ≠ *1*; *0* < *x* |] ==> *hlog a* (*inverse x*) = − *hlog a x*
apply (*rule add-left-cancel* [*of hlog a x*, *THEN iffD1*])
apply (*simp add*: *hlog-mult* [*symmetric*])
done

lemma *hlog-divide*:
[| *0* < *a*; *a* ≠ *1*; *0* < *x*; *0* < *y*|] ==> *hlog a* (*x*/*y*) = *hlog a x* − *hlog a y*
by (*simp add*: *hlog-mult hlog-inverse divide-inverse*)

lemma *hlog-less-cancel-iff* [*simp*]:
!!*a x y*. [| *1* < *a*; *0* < *x*; *0* < *y* |] ==> (*hlog a x* < *hlog a y*) = (*x* < *y*)
by (*transfer*, *simp*)

lemma *hlog-le-cancel-iff* [*simp*]:
[| *1* < *a*; *0* < *x*; *0* < *y* |] ==> (*hlog a x* ≤ *hlog a y*) = (*x* ≤ *y*)
by (*simp add*: *linorder-not-less* [*symmetric*])

end

theory *Hyperreal*
imports *Ln Deriv Taylor Integration HLog*
begin

end

55 Complex-Main: Comprehensive Complex Theory

theory *Complex-Main*
imports *Fundamental-Theorem-Algebra CLim ../Hyperreal/Hyperreal*
begin

end