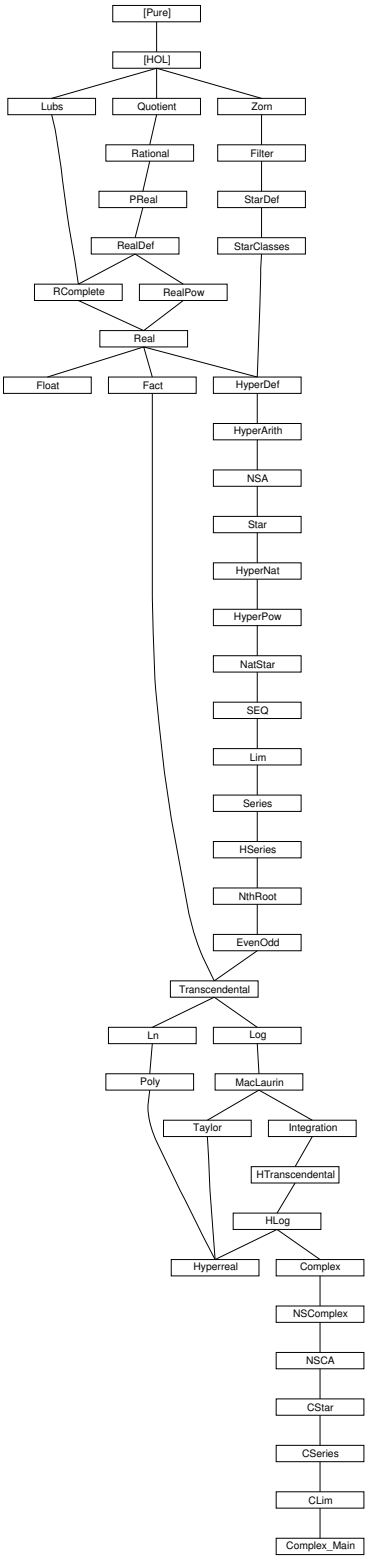
Isabelle/HOL-Complex — Higher-Order Logic with Complex Numbers

October 1, 2005

Contents


[Pure]
[HOL]
Lubs
Quotient
Zorn
Rational
Filter
PReal
StarDef
RealDef
StarClasses
RComplete
RealPow
Real
Float
Fact
HyperDef
HyperArith
NSA
Star
HyperNat
HyperPow
NatStar
SEQ
Lim
Series
HSeries
NthRoot
EvenOdd
Transcendental
Ln
Log
Poly
MacLaurin
Taylor
Integration
HTranscendental
HLog
Hyperreal
Complex
NSComplex
NSCA
CStar
CSeries
CLim
Complex_Main

1 Lubs: Definitions of Upper Bounds and Least Upper Bounds

theory *Lubs*
imports *Main*
begin

Thanks to suggestions by James Margetson

constdefs

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

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

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

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

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

ubs :: ['*a set*, '*a*::*ord set*] => '*a set*
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

ML
⟪
val setle-def = thm setle-def;
val setge-def = thm setge-def;
val leastP-def = thm leastP-def;
val isLub-def = thm isLub-def;
val isUb-def = thm isUb-def;
val ubs-def = thm ubs-def;

val setleI = thm setleI;
val setleD = thm setleD;
val setgeI = thm setgeI;
val setgeD = thm setgeD;
val leastPD1 = thm leastPD1;
val leastPD2 = thm leastPD2;
val leastPD3 = thm leastPD3;
val isLubD1 = thm isLubD1;
val isLubD1a = thm isLubD1a;
val isLub-isUb = thm isLub-isUb;
val isLubD2 = thm isLubD2;
val isLubD3 = thm isLubD3;
val isLubI1 = thm isLubI1;
val isLubI2 = thm isLubI2;
val isUbD = thm isUbD;
val isUbD2 = thm isUbD2;
val isUbD2a = thm isUbD2a;
val isUbI = thm isUbI;
val isLub-le-isUb = thm isLub-le-isUb;
val isLub-ubs = thm isLub-ubs;
⟫

end

2 Quotient: Quotient types

theory *Quotient*
imports *Main*
begin

We introduce the notion of quotient types over equivalence relations via axiomatic type classes.

2.1 Equivalence relations and quotient types

Type class *equiv* models equivalence relations $\sim :: 'a => 'a => bool$.

axclass *eqv* $\subseteq$ *type*

consts
eqv :: ('*a*::*eqv*) => '*a* => *bool* (**infixl** ∼ *50*)

axclass *equiv* ⊆ *eqv*
equiv-refl [*intro*]: *x* ∼ *x*
equiv-trans [*trans*]: *x* ∼ *y* ==> *y* ∼ *z* ==> *x* ∼ *z*
equiv-sym [*sym*]: *x* ∼ *y* ==> *y* ∼ *x*

lemma *equiv-not-sym* [*sym*]: ¬ (*x* ∼ *y*) ==> ¬ (*y* ∼ (*x*::'*a*::*equiv*))
proof −
assume ¬ (*x* ∼ *y*) **thus** ¬ (*y* ∼ *x*)
by (*rule contrapos-nn*) (*rule equiv-sym*)
qed

lemma *not-equiv-trans1* [*trans*]: ¬ (*x* ∼ *y*) ==> *y* ∼ *z* ==> ¬ (*x* ∼ (*z*::'*a*::*equiv*))
proof −
assume ¬ (*x* ∼ *y*) **and** *yz*: *y* ∼ *z*
show ¬ (*x* ∼ *z*)
proof
assume *x* ∼ *z*
also from *yz* **have** *z* ∼ *y* **..**
finally have *x* ∼ *y* **.**
thus *False* **by** *contradiction*
qed
qed

lemma *not-equiv-trans2* [*trans*]: *x* ∼ *y* ==> ¬ (*y* ∼ *z*) ==> ¬ (*x* ∼ (*z*::'*a*::*equiv*))
proof −
assume ¬ (*y* ∼ *z*) **hence** ¬ (*z* ∼ *y*) **..**
also assume *x* ∼ *y* **hence** *y* ∼ *x* **..**
finally have ¬ (*z* ∼ *x*) **. thus** (¬ *x* ∼ *z*) **..**
qed

The quotient type '*a quot* consists of all *equivalence classes* over elements of the base type '*a*.

typedef '*a quot* = {{*x*. *a* ∼ *x*} | *a*::'*a*::*eqv*. *True*}
by *blast*

lemma *quotI* [*intro*]: {*x*. *a* ∼ *x*} ∈ *quot*
by (*unfold quot-def*) *blast*

lemma *quotE* [*elim*]: *R* ∈ *quot* ==> (!!*a*. *R* = {*x*. *a* ∼ *x*} ==> *C*) ==> *C*
by (*unfold quot-def*) *blast*

Abstracted equivalence classes are the canonical representation of elements of a quotient type.

constdefs
class :: '*a*::*equiv* => '*a quot* (⌊-⌋)

$\lfloor a \rfloor$ == *Abs-quot* $\{x.\ a \sim x\}$

theorem *quot-exhaust*: $\exists\, a.\ A = \lfloor a \rfloor$
proof (*cases A*)
fix R **assume** R: $A =$ *Abs-quot* R
assume $R \in$ *quot* **hence** $\exists\, a.\ R = \{x.\ a \sim x\}$ **by** *blast*
with R **have** $\exists\, a.\ A =$ *Abs-quot* $\{x.\ a \sim x\}$ **by** *blast*
thus *?thesis* **by** (*unfold class-def*)
qed

lemma *quot-cases* [*cases type*: *quot*]: $(!!a.\ A = \lfloor a \rfloor \Longrightarrow C) \Longrightarrow C$
by (*insert quot-exhaust*) *blast*

2.2 Equality on quotients

Equality of canonical quotient elements coincides with the original relation.

theorem *quot-equality* [*iff?*]: $(\lfloor a \rfloor = \lfloor b \rfloor) = (a \sim b)$
proof
assume *eq*: $\lfloor a \rfloor = \lfloor b \rfloor$
show $a \sim b$
proof −
from *eq* **have** $\{x.\ a \sim x\} = \{x.\ b \sim x\}$
by (*simp only*: *class-def Abs-quot-inject quotI*)
moreover have $a \sim a$ **..**
ultimately have $a \in \{x.\ b \sim x\}$ **by** *blast*
hence $b \sim a$ **by** *blast*
thus *?thesis* **..**
qed
next
assume *ab*: $a \sim b$
show $\lfloor a \rfloor = \lfloor b \rfloor$
proof −
have $\{x.\ a \sim x\} = \{x.\ b \sim x\}$
proof (*rule Collect-cong*)
fix x **show** $(a \sim x) = (b \sim x)$
proof
from *ab* **have** $b \sim a$ **..**
also assume $a \sim x$
finally show $b \sim x$ **.**
next
note *ab*
also assume $b \sim x$
finally show $a \sim x$ **.**
qed
qed
thus *?thesis* **by** (*simp only*: *class-def*)
qed
qed

2.3 Picking representing elements

constdefs
  pick :: *'a*::*equiv quot* => *'a*
  pick A == *SOME a*. *A* = ⌊*a*⌋

theorem *pick-equiv* [*intro*]: *pick* ⌊*a*⌋ ∼ *a*
proof (*unfold pick-def*)
  show (*SOME x*. ⌊*a*⌋ = ⌊*x*⌋) ∼ *a*
  proof (*rule someI2*)
    show ⌊*a*⌋ = ⌊*a*⌋ **..**
    fix *x* **assume** ⌊*a*⌋ = ⌊*x*⌋
    hence *a* ∼ *x* **.. thus** *x* ∼ *a* **..**
  qed
qed

theorem *pick-inverse* [*intro*]: ⌊*pick A*⌋ = *A*
proof (*cases A*)
  fix *a* **assume** *a*: *A* = ⌊*a*⌋
  hence *pick A* ∼ *a* **by** (*simp only*: *pick-equiv*)
  hence ⌊*pick A*⌋ = ⌊*a*⌋ **..**
  with *a* **show** *?thesis* **by** *simp*
qed

The following rules support canonical function definitions on quotient types (with up to two arguments). Note that the stripped-down version without additional conditions is sufficient most of the time.

theorem *quot-cond-function*:
  (!!*X Y*. *P X Y* ==> *f X Y* == *g* (*pick X*) (*pick Y*)) ==>
    (!!*x x' y y'*. ⌊*x*⌋ = ⌊*x'*⌋ ==> ⌊*y*⌋ = ⌊*y'*⌋
      ==> *P* ⌊*x*⌋ ⌊*y*⌋ ==> *P* ⌊*x'*⌋ ⌊*y'*⌋ ==> *g x y* = *g x' y'*) ==>
    P ⌊*a*⌋ ⌊*b*⌋ ==> *f* ⌊*a*⌋ ⌊*b*⌋ = *g a b*
  (**is** *PROP ?eq* ==> *PROP ?cong* ==> - ==> -)
proof −
  assume *cong*: *PROP ?cong*
  assume *PROP ?eq* **and** *P* ⌊*a*⌋ ⌊*b*⌋
  hence *f* ⌊*a*⌋ ⌊*b*⌋ = *g* (*pick* ⌊*a*⌋) (*pick* ⌊*b*⌋) **by** (*simp only*:)
  also have ... = *g a b*
  proof (*rule cong*)
    show ⌊*pick* ⌊*a*⌋⌋ = ⌊*a*⌋ **..**
    moreover
    show ⌊*pick* ⌊*b*⌋⌋ = ⌊*b*⌋ **..**
    moreover
    show *P* ⌊*a*⌋ ⌊*b*⌋ **.**
    ultimately show *P* ⌊*pick* ⌊*a*⌋⌋ ⌊*pick* ⌊*b*⌋⌋ **by** (*simp only*:)
  qed
  finally show *?thesis* **.**
qed

theorem *quot-function*:
(!!*X Y*. *f X Y* == *g* (*pick X*) (*pick Y*)) ==>
(!!*x x' y y'*. ⌊*x*⌋ = ⌊*x'*⌋ ==> ⌊*y*⌋ = ⌊*y'*⌋ ==> *g x y* = *g x' y'*) ==>
f ⌊*a*⌋ ⌊*b*⌋ = *g a b*
proof −
case *rule-context* **from** *this TrueI*
show *?thesis* **by** (*rule quot-cond-function*)
qed

theorem *quot-function'*:
(!!*X Y*. *f X Y* == *g* (*pick X*) (*pick Y*)) ==>
(!!*x x' y y'*. *x* ∼ *x'* ==> *y* ∼ *y'* ==> *g x y* = *g x' y'*) ==>
f ⌊*a*⌋ ⌊*b*⌋ = *g a b*
by (*rule quot-function*) (*simp only*: *quot-equality*)+

end

3 Rational: Rational numbers

theory *Rational*
imports *Quotient*
uses (*rat-arith.ML*)
begin

3.1 Fractions

3.1.1 The type of fractions

typedef *fraction* = {(*a*, *b*) :: *int* × *int* | *a b*. *b* ≠ *0*}
proof
show (*0*, *1*) ∈ *?fraction* **by** *simp*
qed

constdefs
fract :: *int* => *int* => *fraction*
fract a b == *Abs-fraction* (*a*, *b*)
num :: *fraction* => *int*
num Q == *fst* (*Rep-fraction Q*)
den :: *fraction* => *int*
den Q == *snd* (*Rep-fraction Q*)

lemma *fract-num* [*simp*]: *b* ≠ *0* ==> *num* (*fract a b*) = *a*
by (*simp add*: *fract-def num-def fraction-def Abs-fraction-inverse*)

lemma *fract-den* [*simp*]: *b* ≠ *0* ==> *den* (*fract a b*) = *b*
by (*simp add*: *fract-def den-def fraction-def Abs-fraction-inverse*)

lemma *fraction-cases* [*case-names fract*, *cases type*: *fraction*]:

```
  (!!a b. Q = fract a b ==> b ≠ 0 ==> C) ==> C
proof −
  assume r: !!a b. Q = fract a b ==> b ≠ 0 ==> C
  obtain a b where Q = fract a b and b ≠ 0
    by (cases Q) (auto simp add: fract-def fraction-def)
  thus C by (rule r)
qed

lemma fraction-induct [case-names fract, induct type: fraction]:
    (!!a b. b ≠ 0 ==> P (fract a b)) ==> P Q
  by (cases Q) simp
```

3.1.2 Equivalence of fractions

```
instance fraction :: eqv ..

defs (overloaded)
  equiv-fraction-def: Q ∼ R == num Q * den R = num R * den Q

lemma equiv-fraction-iff [iff]:
    b ≠ 0 ==> b' ≠ 0 ==> (fract a b ∼ fract a' b') = (a * b' = a' * b)
  by (simp add: equiv-fraction-def)

instance fraction :: equiv
proof
  fix Q R S :: fraction
  {
    show Q ∼ Q
    proof (induct Q)
      fix a b :: int
      assume b ≠ 0 and b ≠ 0
      with refl show fract a b ∼ fract a b ..
    qed
  next
    assume Q ∼ R and R ∼ S
    show Q ∼ S
    proof (insert prems, induct Q, induct R, induct S)
      fix a b a' b' a'' b'' :: int
      assume b: b ≠ 0 and b': b' ≠ 0 and b'': b'' ≠ 0
      assume fract a b ∼ fract a' b' hence eq1: a * b' = a' * b ..
      assume fract a' b' ∼ fract a'' b'' hence eq2: a' * b'' = a'' * b' ..
      have a * b'' = a'' * b
      proof cases
        assume a' = 0
        with b' eq1 eq2 have a = 0 ∧ a'' = 0 by auto
        thus ?thesis by simp
      next
        assume a': a' ≠ 0
        from eq1 eq2 have (a * b') * (a' * b'') = (a' * b) * (a'' * b') by simp
```

```
      hence (a * b'') * (a' * b') = (a'' * b) * (a' * b') by (simp only: mult-ac)
      with a' b' show ?thesis by simp
    qed
    thus fract a b ~ fract a'' b'' ..
  qed
 next
  show Q ~ R ==> R ~ Q
  proof (induct Q, induct R)
    fix a b a' b' :: int
    assume b: b ≠ 0 and b': b' ≠ 0
    assume fract a b ~ fract a' b'
    hence a * b' = a' * b ..
    hence a' * b = a * b' ..
    thus fract a' b' ~ fract a b ..
  qed
 }
qed

lemma eq-fraction-iff [iff]:
   b ≠ 0 ==> b' ≠ 0 ==> (⌊fract a b⌋ = ⌊fract a' b'⌋) = (a * b' = a' * b)
 by (simp add: equiv-fraction-iff quot-equality)
```

3.1.3 Operations on fractions

We define the basic arithmetic operations on fractions and demonstrate their "well-definedness", i.e. congruence with respect to equivalence of fractions.

```
instance fraction :: {zero, one, plus, minus, times, inverse, ord} ..

defs (overloaded)
  zero-fraction-def: 0 == fract 0 1
  one-fraction-def: 1 == fract 1 1
  add-fraction-def: Q + R ==
    fract (num Q * den R + num R * den Q) (den Q * den R)
  minus-fraction-def: -Q == fract (-(num Q)) (den Q)
  mult-fraction-def: Q * R == fract (num Q * num R) (den Q * den R)
  inverse-fraction-def: inverse Q == fract (den Q) (num Q)
  le-fraction-def: Q ≤ R ==
    (num Q * den R) * (den Q * den R) ≤ (num R * den Q) * (den Q * den R)

lemma is-zero-fraction-iff: b ≠ 0 ==> (⌊fract a b⌋ = ⌊0⌋) = (a = 0)
 by (simp add: zero-fraction-def eq-fraction-iff)

theorem add-fraction-cong:
  ⌊fract a b⌋ = ⌊fract a' b'⌋ ==> ⌊fract c d⌋ = ⌊fract c' d'⌋
    ==> b ≠ 0 ==> b' ≠ 0 ==> d ≠ 0 ==> d' ≠ 0
    ==> ⌊fract a b + fract c d⌋ = ⌊fract a' b' + fract c' d'⌋
proof -
  assume neq: b ≠ 0  b' ≠ 0  d ≠ 0  d' ≠ 0
  assume ⌊fract a b⌋ = ⌊fract a' b'⌋ hence eq1: a * b' = a' * b ..
```

```
  assume ⌊fract c d⌋ = ⌊fract c' d'⌋ hence eq2: c * d' = c' * d ..
  have ⌊fract (a * d + c * b) (b * d)⌋ = ⌊fract (a' * d' + c' * b') (b' * d')⌋
  proof
    show (a * d + c * b) * (b' * d') = (a' * d' + c' * b') * (b * d)
      (is ?lhs = ?rhs)
    proof -
      have ?lhs = (a * b') * (d * d') + (c * d') * (b * b')
        by (simp add: int-distrib mult-ac)
      also have ... = (a' * b) * (d * d') + (c' * d) * (b * b')
        by (simp only: eq1 eq2)
      also have ... = ?rhs
        by (simp add: int-distrib mult-ac)
      finally show ?lhs = ?rhs .
    qed
    from neq show b * d ≠ 0 by simp
    from neq show b' * d' ≠ 0 by simp
  qed
  with neq show ?thesis by (simp add: add-fraction-def)
qed

theorem minus-fraction-cong:
  ⌊fract a b⌋ = ⌊fract a' b'⌋ ==> b ≠ 0 ==> b' ≠ 0
    ==> ⌊-(fract a b)⌋ = ⌊-(fract a' b')⌋
proof -
  assume neq: b ≠ 0  b' ≠ 0
  assume ⌊fract a b⌋ = ⌊fract a' b'⌋
  hence a * b' = a' * b ..
  hence -a * b' = -a' * b by simp
  hence ⌊fract (-a) b⌋ = ⌊fract (-a') b'⌋ ..
  with neq show ?thesis by (simp add: minus-fraction-def)
qed

theorem mult-fraction-cong:
  ⌊fract a b⌋ = ⌊fract a' b'⌋ ==> ⌊fract c d⌋ = ⌊fract c' d'⌋
    ==> b ≠ 0 ==> b' ≠ 0 ==> d ≠ 0 ==> d' ≠ 0
    ==> ⌊fract a b * fract c d⌋ = ⌊fract a' b' * fract c' d'⌋
proof -
  assume neq: b ≠ 0  b' ≠ 0  d ≠ 0  d' ≠ 0
  assume ⌊fract a b⌋ = ⌊fract a' b'⌋ hence eq1: a * b' = a' * b ..
  assume ⌊fract c d⌋ = ⌊fract c' d'⌋ hence eq2: c * d' = c' * d ..
  have ⌊fract (a * c) (b * d)⌋ = ⌊fract (a' * c') (b' * d')⌋
  proof
    from eq1 eq2 have (a * b') * (c * d') = (a' * b) * (c' * d) by simp
    thus (a * c) * (b' * d') = (a' * c') * (b * d) by (simp add: mult-ac)
    from neq show b * d ≠ 0 by simp
    from neq show b' * d' ≠ 0 by simp
  qed
  with neq show ⌊fract a b * fract c d⌋ = ⌊fract a' b' * fract c' d'⌋
    by (simp add: mult-fraction-def)
```

qed

theorem *inverse-fraction-cong*:
⌊*fract a b*⌋ = ⌊*fract a′ b′*⌋ ==> ⌊*fract a b*⌋ ≠ ⌊*0*⌋ ==> ⌊*fract a′ b′*⌋ ≠ ⌊*0*⌋
==> *b* ≠ *0* ==> *b′* ≠ *0*
==> ⌊*inverse* (*fract a b*)⌋ = ⌊*inverse* (*fract a′ b′*)⌋
proof −
assume *neq*: *b* ≠ *0* *b′* ≠ *0*
assume ⌊*fract a b*⌋ ≠ ⌊*0*⌋ **and** ⌊*fract a′ b′*⌋ ≠ ⌊*0*⌋
with *neq* **obtain** *a* ≠ *0* **and** *a′* ≠ *0* **by** (*simp add*: *is-zero-fraction-iff*)
assume ⌊*fract a b*⌋ = ⌊*fract a′ b′*⌋
hence *a* ∗ *b′* = *a′* ∗ *b* **..**
hence *b* ∗ *a′* = *b′* ∗ *a* **by** (*simp only*: *mult-ac*)
hence ⌊*fract b a*⌋ = ⌊*fract b′ a′*⌋ **..**
with *neq* **show** *?thesis* **by** (*simp add*: *inverse-fraction-def*)
qed

theorem *le-fraction-cong*:
⌊*fract a b*⌋ = ⌊*fract a′ b′*⌋ ==> ⌊*fract c d*⌋ = ⌊*fract c′ d′*⌋
==> *b* ≠ *0* ==> *b′* ≠ *0* ==> *d* ≠ *0* ==> *d′* ≠ *0*
==> (*fract a b* ≤ *fract c d*) = (*fract a′ b′* ≤ *fract c′ d′*)
proof −
assume *neq*: *b* ≠ *0* *b′* ≠ *0* *d* ≠ *0* *d′* ≠ *0*
assume ⌊*fract a b*⌋ = ⌊*fract a′ b′*⌋ **hence** *eq1*: *a* ∗ *b′* = *a′* ∗ *b* **..**
assume ⌊*fract c d*⌋ = ⌊*fract c′ d′*⌋ **hence** *eq2*: *c* ∗ *d′* = *c′* ∗ *d* **..**

let *?le* = λ*a b c d*. ((*a* ∗ *d*) ∗ (*b* ∗ *d*) ≤ (*c* ∗ *b*) ∗ (*b* ∗ *d*))
{
fix *a b c d x* :: *int* **assume** *x*: *x* ≠ *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*) ≤ (*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* ≠ *0* **by** *simp*
from *neq* **have** *?D′* ≠ *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′* ≤ (*c* ∗ *d′*) ∗ *?D* ∗ *?D′* ∗ *b* ∗ *b′*)
by (*simp add*: *mult-ac*)
also have ... = ((*a′* ∗ *b*) ∗ *?D* ∗ *?D′* ∗ *d* ∗ *d′* ≤ (*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 have *?le a b c d* = *?le a' b' c' d'* **.**
with *neq* **show** *?thesis* **by** (*simp add*: *le-fraction-def*)
qed

3.2 Rational numbers

3.2.1 The type of rational numbers

typedef (*Rat*)
rat = *UNIV* :: *fraction quot set* **..**

lemma *RatI* [*intro*, *simp*]: *Q* ∈ *Rat*
by (*simp add*: *Rat-def*)

constdefs
fraction-of :: *rat* => *fraction*
fraction-of q == *pick* (*Rep-Rat q*)
rat-of :: *fraction* => *rat*
rat-of Q == *Abs-Rat* ⌊*Q*⌋

theorem *rat-of-equality* [*iff?*]: (*rat-of Q* = *rat-of Q'*) = (⌊*Q*⌋ = ⌊*Q'*⌋)
by (*simp add*: *rat-of-def Abs-Rat-inject*)

lemma *rat-of*: ⌊*Q*⌋ = ⌊*Q'*⌋ ==> *rat-of Q* = *rat-of Q'* **..**

constdefs
Fract :: *int* => *int* => *rat*
Fract a b == *rat-of* (*fract a b*)

theorem *Fract-inverse*: ⌊*fraction-of* (*Fract a b*)⌋ = ⌊*fract a b*⌋
by (*simp add*: *fraction-of-def rat-of-def Fract-def Abs-Rat-inverse pick-inverse*)

theorem *Fract-equality* [*iff?*]:
(*Fract a b* = *Fract c d*) = (⌊*fract a b*⌋ = ⌊*fract c d*⌋)
by (*simp add*: *Fract-def rat-of-equality*)

theorem *eq-rat*:
b ≠ *0* ==> *d* ≠ *0* ==> (*Fract a b* = *Fract c d*) = (*a* * *d* = *c* * *b*)
by (*simp add*: *Fract-equality eq-fraction-iff*)

theorem *Rat-cases* [*case-names Fract*, *cases type*: *rat*]:
(!!*a b*. *q* = *Fract a b* ==> *b* ≠ *0* ==> *C*) ==> *C*
proof −
assume *r*: !!*a b*. *q* = *Fract a b* ==> *b* ≠ *0* ==> *C*
obtain *x* **where** *q* = *Abs-Rat x* **by** (*cases q*)

```
  moreover obtain Q where x = ⌊Q⌋ by (cases x)
  moreover obtain a b where Q = fract a b and b ≠ 0 by (cases Q)
  ultimately have q = Fract a b by (simp only: Fract-def rat-of-def)
  thus ?thesis by (rule r)
qed

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

3.2.2 Canonical function definitions

Note that the unconditional version below is much easier to read.

```
theorem rat-cond-function:
  (!!q r. P ⌊fraction-of q⌋ ⌊fraction-of r⌋ ==>
      f q r == g (fraction-of q) (fraction-of r)) ==>
    (!!a b a' b' c d c' d'.
      ⌊fract a b⌋ = ⌊fract a' b'⌋ ==> ⌊fract c d⌋ = ⌊fract c' d'⌋ ==>
      P ⌊fract a b⌋ ⌊fract c d⌋ ==> P ⌊fract a' b'⌋ ⌊fract c' d'⌋ ==>
      b ≠ 0 ==> b' ≠ 0 ==> d ≠ 0 ==> d' ≠ 0 ==>
      g (fract a b) (fract c d) = g (fract a' b') (fract c' d')) ==>
    P ⌊fract a b⌋ ⌊fract c d⌋ ==>
      f (Fract a b) (Fract c d) = g (fract a b) (fract c d)
  (is PROP ?eq ==> PROP ?cong ==> ?P ==> -)
proof −
  assume eq: PROP ?eq and cong: PROP ?cong and P: ?P
  have f (Abs-Rat ⌊fract a b⌋) (Abs-Rat ⌊fract c d⌋) = g (fract a b) (fract c d)
  proof (rule quot-cond-function)
    fix X Y assume P X Y
    with eq show f (Abs-Rat X) (Abs-Rat Y) == g (pick X) (pick Y)
      by (simp add: fraction-of-def pick-inverse Abs-Rat-inverse)
  next
    fix Q Q' R R' :: fraction
    show ⌊Q⌋ = ⌊Q'⌋ ==> ⌊R⌋ = ⌊R'⌋ ==>
        P ⌊Q⌋ ⌊R⌋ ==> P ⌊Q'⌋ ⌊R'⌋ ==> g Q R = g Q' R'
      by (induct Q, induct Q', induct R, induct R') (rule cong)
  qed
  thus ?thesis by (unfold Fract-def rat-of-def)
qed

theorem rat-function:
  (!!q r. f q r == g (fraction-of q) (fraction-of r)) ==>
    (!!a b a' b' c d c' d'.
      ⌊fract a b⌋ = ⌊fract a' b'⌋ ==> ⌊fract c d⌋ = ⌊fract c' d'⌋ ==>
      b ≠ 0 ==> b' ≠ 0 ==> d ≠ 0 ==> d' ≠ 0 ==>
      g (fract a b) (fract c d) = g (fract a' b') (fract c' d')) ==>
    f (Fract a b) (Fract c d) = g (fract a b) (fract c d)
proof −
  case rule-context from this TrueI
```

show *?thesis* **by** (*rule rat-cond-function*)
qed

3.2.3 Standard operations on rational numbers

instance *rat* :: {*zero*, *one*, *plus*, *minus*, *times*, *inverse*, *ord*} **..**

defs (**overloaded**)
zero-rat-def: $0 == rat\text{-}of\ 0$
one-rat-def: $1 == rat\text{-}of\ 1$
add-rat-def: $q + r == rat\text{-}of\ (fraction\text{-}of\ q + fraction\text{-}of\ r)$
minus-rat-def: $-q == rat\text{-}of\ (-(fraction\text{-}of\ q))$
diff-rat-def: $q - r == q + (-(r::rat))$
mult-rat-def: $q * r == rat\text{-}of\ (fraction\text{-}of\ q * fraction\text{-}of\ r)$
inverse-rat-def: $inverse\ q ==$
$if\ q=0\ then\ 0\ else\ rat\text{-}of\ (inverse\ (fraction\text{-}of\ q))$
divide-rat-def: $q\ /\ r == q * inverse\ (r::rat)$
le-rat-def: $q \leq r == fraction\text{-}of\ q \leq fraction\text{-}of\ r$
less-rat-def: $q < r == q \leq r \wedge q \neq (r::rat)$
abs-rat-def: $|q| == if\ q < 0\ then\ -q\ else\ (q::rat)$

theorem *zero-rat*: $0 = Fract\ 0\ 1$
by (*simp add*: *zero-rat-def zero-fraction-def rat-of-def Fract-def*)

theorem *one-rat*: $1 = Fract\ 1\ 1$
by (*simp add*: *one-rat-def one-fraction-def rat-of-def Fract-def*)

theorem *add-rat*: $b \neq 0 ==> d \neq 0 ==>$
$Fract\ a\ b + Fract\ c\ d = Fract\ (a * d + c * b)\ (b * d)$
proof −
have $Fract\ a\ b + Fract\ c\ d = rat\text{-}of\ (fract\ a\ b + fract\ c\ d)$
by (*rule rat-function*, *rule add-rat-def*, *rule rat-of*, *rule add-fraction-cong*)
also
assume $b \neq 0\ \ d \neq 0$
hence $fract\ a\ b + fract\ c\ d = fract\ (a * d + c * b)\ (b * d)$
by (*simp add*: *add-fraction-def*)
finally show *?thesis* **by** (*unfold Fract-def*)
qed

theorem *minus-rat*: $b \neq 0 ==> -(Fract\ a\ b) = Fract\ (-a)\ b$
proof −
have $-(Fract\ a\ b) = rat\text{-}of\ (-(fract\ a\ b))$
by (*rule rat-function*, *rule minus-rat-def*, *rule rat-of*, *rule minus-fraction-cong*)
also assume $b \neq 0$ **hence** $-(fract\ a\ b) = fract\ (-a)\ b$
by (*simp add*: *minus-fraction-def*)
finally show *?thesis* **by** (*unfold Fract-def*)
qed

theorem *diff-rat*: $b \neq 0 ==> d \neq 0 ==>$

```
    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 ≠ 0 ==> d ≠ 0 ==>
  Fract a b * Fract c d = Fract (a * c) (b * d)
proof −
  have Fract a b * Fract c d = rat-of (fract a b * fract c d)
    by (rule rat-function, rule mult-rat-def, rule rat-of, rule mult-fraction-cong)
  also
  assume b ≠ 0  d ≠ 0
  hence fract a b * fract c d = fract (a * c) (b * d)
    by (simp add: mult-fraction-def)
  finally show ?thesis by (unfold Fract-def)
qed

theorem inverse-rat: Fract a b ≠ 0 ==> b ≠ 0 ==>
  inverse (Fract a b) = Fract b a
proof −
  assume neq: b ≠ 0 and nonzero: Fract a b ≠ 0
  hence ⌊fract a b⌋ ≠ ⌊0⌋
    by (simp add: zero-rat eq-rat is-zero-fraction-iff)
  with - inverse-fraction-cong [THEN rat-of]
  have inverse (Fract a b) = rat-of (inverse (fract a b))
  proof (rule rat-cond-function)
    fix q assume cond: ⌊fraction-of q⌋ ≠ ⌊0⌋
    have q ≠ 0
    proof (cases q)
      fix a b assume b ≠ 0 and q = Fract a b
      from this cond show ?thesis
        by (simp add: Fract-inverse is-zero-fraction-iff zero-rat eq-rat)
    qed
    thus inverse q == rat-of (inverse (fraction-of q))
      by (simp add: inverse-rat-def)
  qed
  also from neq nonzero have inverse (fract a b) = fract b a
    by (simp add: inverse-fraction-def)
  finally show ?thesis by (unfold Fract-def)
qed

theorem divide-rat: Fract c d ≠ 0 ==> b ≠ 0 ==> d ≠ 0 ==>
  Fract a b / Fract c d = Fract (a * d) (b * c)
proof −
  assume neq: b ≠ 0  d ≠ 0 and nonzero: Fract c d ≠ 0
  hence c ≠ 0 by (simp add: zero-rat eq-rat)
  with neq nonzero show ?thesis
    by (simp add: divide-rat-def inverse-rat mult-rat)
qed

theorem le-rat: b ≠ 0 ==> d ≠ 0 ==>
```

```
  (Fract a b ≤ Fract c d) = ((a * d) * (b * d) ≤ (c * b) * (b * d))
proof −
  have (Fract a b ≤ Fract c d) = (fract a b ≤ fract c d)
    by (rule rat-function, rule le-rat-def, rule le-fraction-cong)
  also
  assume b ≠ 0  d ≠ 0
  hence (fract a b ≤ fract c d) = ((a * d) * (b * d) ≤ (c * b) * (b * d))
    by (simp add: le-fraction-def)
  finally show ?thesis .
qed

theorem less-rat: b ≠ 0 ==> d ≠ 0 ==>
    (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 ≠ 0 ==> |Fract a b| = Fract |a| |b|
  by (simp add: abs-rat-def minus-rat zero-rat less-rat eq-rat)
    (auto simp add: mult-less-0-iff zero-less-mult-iff order-le-less
          split: abs-split)
```

3.2.4 The ordered field of rational numbers

```
lemma rat-add-assoc: (q + r) + s = q + (r + (s::rat))
  by (induct q, induct r, induct s)
    (simp add: add-rat add-ac mult-ac int-distrib)

lemma rat-add-0: 0 + q = (q::rat)
  by (induct q) (simp add: zero-rat add-rat)

lemma rat-left-minus: (−q) + q = (0::rat)
  by (induct q) (simp add: zero-rat minus-rat add-rat eq-rat)


instance rat :: field
proof
  fix q r s :: rat
  show (q + r) + s = q + (r + s)
    by (rule rat-add-assoc)
  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 add-rat)
  show (−q) + q = 0
    by (rule rat-left-minus)
  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 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 zero-rat eq-rat)
  show q / r = q * inverse r
    by (simp add: divide-rat-def)
  show 0 ≠ (1::rat)
    by (simp add: zero-rat one-rat 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-ac, arith)
  }
qed

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 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

instance rat :: division-by-zero
proof
  show inverse 0 = (0::rat)  by (simp add: inverse-rat-def)
qed
```

3.3 Various Other Results

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

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 less-rat order-less-imp-not-eq2 zero-less-mult-iff*)

lemma *Fract-add-one*: *n ≠ 0 ==> Fract (m + n) n = Fract m n + 1*
apply (*insert add-rat* [*of* **concl**: *m n 1 1*])
apply (*simp add*: *one-rat* [*symmetric*])
done

lemma *Fract-of-nat-eq*: *Fract (of-nat k) 1 = of-nat k*
apply (*induct k*)
apply (*simp add*: *zero-rat*)
apply (*simp add*: *Fract-add-one*)
done

lemma *Fract-of-int-eq*: *Fract k 1 = of-int k*
proof (*cases k rule*: *int-cases*)
case (*nonneg n*)
thus *?thesis* **by** (*simp add*: *int-eq-of-nat Fract-of-nat-eq*)
next
case (*neg n*)
hence *Fract k 1 = − (Fract (of-nat (Suc n)) 1)*
by (*simp only*: *minus-rat int-eq-of-nat*)
also have *... = − (of-nat (Suc n))*
by (*simp only*: *Fract-of-nat-eq*)
finally show *?thesis*
by (*simp add*: *only*: *prems int-eq-of-nat of-int-minus of-int-of-nat-eq*)
qed

3.4 Numerals and Arithmetic

instance *rat* :: *number* **..**

defs (**overloaded**)
rat-number-of-def: (*number-of w* :: *rat*) == *of-int* (*Rep-Bin w*)
— the type constraint is essential!

instance *rat* :: *number-ring*
by (*intro-classes*, *simp add*: *rat-number-of-def*)

declare *diff-rat-def* [*symmetric*]

use *rat-arith.ML*

setup *rat-arith-setup*

end

4 PReal: Positive real numbers

theory *PReal*
imports *Rational*
begin

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

lemma *add-eq-exists*: $\exists\, x.\ a+x = (b::rat)$
by (*rule-tac* $x=b-a$ **in** *exI*, *simp*)

As a special case, the sum of two positives is positive. One of the premises could be weakened to the relation $\leq$.

lemma *pos-add-strict*: $[|0<a;\ b<c|] ==> b < a + (c::'a::ordered\text{-}semidom)$
by (*insert add-strict-mono* [*of 0 a b c*], *simp*)

lemma *interval-empty-iff*:
$(\{y::'a::ordered\text{-}field.\ x < y\ \&\ y < z\} = \{\}) = (\sim(x < z))$
by (*blast dest*: *dense intro*: *order-less-trans*)

constdefs
cut :: *rat set* => *bool*
cut A == $\{\} \subset A\ \&$
$A < \{r.\ 0 < r\}\ \&$
$(\forall\, y \in A.\ ((\forall\, z.\ 0<z\ \&\ z < y \text{ --> } z \in A)\ \&\ (\exists\, u \in A.\ y < u)))$

lemma *cut-of-rat*:
assumes *q*: $0 < q$ **shows** *cut* $\{r::rat.\ 0 < r\ \&\ r < q\}$
proof −
let *?A* = $\{r::rat.\ 0 < r\ \&\ r < q\}$
from *q* **have** *pos*: *?A* < $\{r.\ 0 < r\}$ **by** *force*
have *nonempty*: $\{\} \subset$ *?A*
proof
show $\{\} \subseteq$ *?A* **by** *simp*
show $\{\} \neq$ *?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*])

instance *preal* :: {*ord*, *plus*, *minus*, *times*, *inverse*} **..**

constdefs
preal-of-rat :: *rat* => *preal*
preal-of-rat q == *Abs-preal*({*x*::*rat*. 0 < *x* & *x* < *q*})

psup :: *preal set* => *preal*
psup(*P*) == *Abs-preal*($\bigcup X \in P$. *Rep-preal*(*X*))

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

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

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

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

defs (**overloaded**)

preal-less-def:
R < (*S*::*preal*) == *Rep-preal R* < *Rep-preal S*

preal-le-def:
R ≤ (*S*::*preal*) == *Rep-preal R* ⊆ *Rep-preal S*

preal-add-def:
R + *S* == *Abs-preal* (*add-set* (*Rep-preal R*) (*Rep-preal S*))

preal-diff-def:
R − *S* == *Abs-preal* (*diff-set* (*Rep-preal R*) (*Rep-preal S*))

preal-mult-def:
R * *S* == *Abs-preal*(*mult-set* (*Rep-preal R*) (*Rep-preal S*))

preal-inverse-def:
inverse R == *Abs-preal*(*inverse-set* (*Rep-preal R*))

Reduces equality on abstractions to equality on representatives

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

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

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 *mem-Rep-preal-Ex*: $\exists x.\ x \in Rep\text{-}preal\ X$
apply (*insert Rep-preal* [*of X*])
apply (*unfold preal-def cut-def*, *blast*)
done

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

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

lemma *Rep-preal-exists-bound*: $\exists x.\ 0 < x\ \&\ x \notin Rep\text{-}preal\ X$
apply (*cut-tac* $x = X$ **in** *Rep-preal*)
apply (*drule preal-imp-psubset-positives*)
apply (*auto simp add*: *psubset-def*)
done

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

lemma *rat-less-set-mem-preal*: $0 < y \Longrightarrow \{u{::}rat.\ 0 < u\ \&\ u < y\} \in preal$
apply (*auto simp add*: *preal-def cut-def intro*: *order-less-trans*)
apply (*force simp only*: *eq-commute* [*of* {}] *interval-empty-iff*)
apply (*blast dest*: *dense intro*: *order-less-trans*)
done

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*|] ==> *x* = *y*
by (*blast intro*: *rat-subset-imp-le order-antisym*)

4.2 Theorems for Ordering

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* ∈ *preal*
and *notx*: *x* ∉ *A*
and *y*: *y* ∈ *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* **by** *assumption*
qed

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

4.3 The ≤ Ordering

lemma *preal-le-refl*: *w* ≤ (*w*::*preal*)
by (*simp add*: *preal-le-def*)

lemma *preal-le-trans*: [| *i* ≤ *j*; *j* ≤ *k* |] ==> *i* ≤ (*k*::*preal*)
by (*force simp add*: *preal-le-def*)

lemma *preal-le-anti-sym*: [| *z* ≤ *w*; *w* ≤ *z* |] ==> *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* ≤ *z* & *w* ≠ *z*)
by (*simp add*: *preal-le-def preal-less-def Rep-preal-inject psubset-def*)

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*; $r \in A$|] ==> $0 < r$
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*)

4.4 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*: $a \in A$ ==> $\{\} \subset A$
by *blast*

Part 1 of Dedekind sections definition

lemma *add-set-not-empty*:
[|$A \in$ *preal*; $B \in$ *preal*|] ==> $\{\} \subset$ *add-set A B*
apply (*insert preal-nonempty* [*of A*] *preal-nonempty* [*of B*])
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*:
[|$A \in$ *preal*; $B \in$ *preal*|] ==> $\exists q.\ 0 < q$ & $q \notin$ *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 not-in-preal-ub*, *assumption*+)+
apply (*force dest*: *add-strict-mono*)
done

lemma *add-set-not-rat-set*:
assumes *A*: $A \in$ *preal*
and *B*: $B \in$ *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
show ∃ *y'* ∈ *B*. *z* = *x*∗*?f* + *y'*
proof
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
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* ∈ *preal*; *B* ∈ *preal*; *y* ∈ *add-set A B*|] ==> ∃ *u* ∈ *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* ∈ *preal*; *B* ∈ *preal*|] ==> *add-set A B* ∈ *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

lemma *preal-add-left-commute*: *x* + (*y* + *z*) = *y* + ((*x* + *z*)::*preal*)
apply (*rule mk-left-commute* [*of op* +])
apply (*rule preal-add-assoc*)
apply (*rule preal-add-commute*)
done

Positive Real addition is an AC operator

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

4.5 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*:
  [|$A \in$ *preal*; $B \in$ *preal*|] ==> {} $\subset$ *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*: $A \in$ *preal*
    and *B*: $B \in$ *preal*
  shows $\exists q.\ 0 < q$ & $q \notin$ *mult-set A B*
proof −
 from *preal-exists-bound* [*OF A*]
 obtain *x* **where** [*simp*]: $0 < x$ $x \notin A$ **by** *blast*
 from *preal-exists-bound* [*OF B*]
 obtain *y* **where** [*simp*]: $0 < y$ $y \notin B$ **by** *blast*
 show *?thesis*
 proof (*intro exI conjI*)
  show $0 < x*y$ **by** (*simp add*: *mult-pos-pos*)
  show $x * y \notin$ *mult-set A B*
  proof −
   { fix *u*::*rat* **and** *v*::*rat*
      assume $u \in A$ **and** $v \in B$ **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** $0 \leq v$
        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*: $A \in$ *preal*
    and *B*: $B \in$ *preal*
  shows *mult-set A B* $<$ {*r*. $0 < r$}
proof
 show *mult-set A B* $\subseteq$ {*r*. $0 < r$}
  by (*force simp add*: *mult-set-def*
      intro: *preal-imp-pos* [*OF A*] *preal-imp-pos* [*OF B*] *mult-pos-pos*)
next
 show *mult-set A B* $\neq$ {*r*. $0 < r$}
  by (*insert preal-not-mem-mult-set-Ex* [*OF A B*], *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 $y \in B$ **.**
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

lemma *preal-mult-left-commute*: $x * (y * z) = y * ((x * z)::preal)$
apply (*rule mk-left-commute* [*of op* $*$])
apply (*rule preal-mult-assoc*)
apply (*rule preal-mult-commute*)
done

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

lemma *preal-mult-1*: $(preal\text{-}of\text{-}rat\ 1) * z = z$
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 .
      qed
    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


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

4.6 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 linorder-le-cases [case-names le ge]:
   ((x::'a::linorder) <= y ==> P) ==> (y <= x ==> P) ==> P
 apply (insert linorder-linear, blast)
 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)
```

4.7 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

4.8 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
```

4.9 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
```

4.10 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 = (preal-of-rat 1)
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 = (preal-of-rat 1)
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* ∈ *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* ∈ *Rep-preal R* **and** *notin*: *r* + *y* ∉ *Rep-preal R* **..**
have *r* + *y* ∈ *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*) ≠ *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 psubset-def*)
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* ≠ *x* + (*y*::*preal*)
by (*insert preal-self-less-add-left* [*of x y*], *auto*)

4.11 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* ==> {} ⊂ *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*:
∃ *q*. *0* < *q* & *q* ∉ *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 $R + D \leq S$

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 ∈ 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) ≤ 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
```

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

```
lemma lemma-sum-mem-Rep-preal-ex:
    x ∈ Rep-preal S ==> ∃ e. 0 < e & x + e ∈ 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 ∈ Rep-preal S
    and xnot: x ∉  Rep-preal R
  shows ∃ u v z. 0 < v & 0 < z & u ∈ Rep-preal R & z ∉ Rep-preal R &
                 z + v ∈ 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 ∈ Rep-preal S by blast
  from  Gleason9-34 [OF Rep-preal epos]
  obtain r where r: r ∈ Rep-preal R and notin: r + e ∉ 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 ∈ Rep-preal R by (rule r)
    show r + e ∉ Rep-preal R by (rule notin)
    show r + e + y ∈ 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-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

4.13 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*:
∀ *X* ∈ *P*. *X* ≤ *Y* ==> (⋃ *X* ∈ *P*. *Rep-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*:
$[|P \neq \{\}; \forall X \in P.\ X \leq Y;\ u \in (\bigcup X \in P.\ Rep\text{-}preal(X));\ 0 < z;\ z < u|]$
$==>\ z \in (\bigcup X \in P.\ Rep\text{-}preal(X))$
by (*auto elim*: *Rep-preal* [*THEN preal-downwards-closed*])

Part 4 of Dedekind sections definition

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

lemma *preal-sup*:
$[|P \neq \{\}; \forall X \in P.\ X \leq Y|] ==> (\bigcup X \in P.\ Rep\text{-}preal(X)) \in preal$
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*:
$[|\ \forall X \in P.\ X \leq Y;\ \ x \in P\ |] ==> x \leq psup\ P$
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*: $[|\ P \neq \{\}; \forall X \in P.\ X \leq Y\ |] ==> psup\ P \leq Y$
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*:
$[|\ P \neq \{\}; \forall X \in P.\ X \leq Y\ |] ==> (\exists X \in P.\ Z < X) = (Z < psup\ P)$
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

4.14 The Embadding 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-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 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-of-rat* $((x{::}rat) * y) =$ *preal-of-rat* $x *$ *preal-of-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* ≤ *preal-of-rat y*) = (*x* ≤ *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*)

ML
⟪
val mem-Rep-preal-Ex = *thmmem-Rep-preal-Ex*;
val preal-add-commute = *thmpreal-add-commute*;
val preal-add-assoc = *thmpreal-add-assoc*;
val preal-add-left-commute = *thmpreal-add-left-commute*;
val preal-mult-commute = *thmpreal-mult-commute*;
val preal-mult-assoc = *thmpreal-mult-assoc*;
val preal-mult-left-commute = *thmpreal-mult-left-commute*;
val preal-add-mult-distrib2 = *thmpreal-add-mult-distrib2*;
val preal-add-mult-distrib = *thmpreal-add-mult-distrib*;
val preal-self-less-add-left = *thmpreal-self-less-add-left*;
val preal-self-less-add-right = *thmpreal-self-less-add-right*;
val less-add-left = *thmless-add-left*;
val preal-add-less2-mono1 = *thmpreal-add-less2-mono1*;
val preal-add-less2-mono2 = *thmpreal-add-less2-mono2*;
val preal-add-right-less-cancel = *thmpreal-add-right-less-cancel*;
val preal-add-left-less-cancel = *thmpreal-add-left-less-cancel*;
val preal-add-right-cancel = *thmpreal-add-right-cancel*;
val preal-add-left-cancel = *thmpreal-add-left-cancel*;
val preal-add-left-cancel-iff = *thmpreal-add-left-cancel-iff*;
val preal-add-right-cancel-iff = *thmpreal-add-right-cancel-iff*;
val preal-psup-le = *thmpreal-psup-le*;
val psup-le-ub = *thmpsup-le-ub*;
val preal-complete = *thmpreal-complete*;
val preal-of-rat-add = *thmpreal-of-rat-add*;
val preal-of-rat-mult = *thmpreal-of-rat-mult*;

val preal-add-ac = *thmspreal-add-ac*;
val preal-mult-ac = *thmspreal-mult-ac*;
⟫

end

5 RealDef: Defining the Reals from the Positive Reals

theory *RealDef*
imports *PReal*

uses (*real-arith.ML*)
begin

constdefs
realrel :: ((*preal* * *preal*) * (*preal* * *preal*)) *set*
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*)

instance *real* :: {*ord*, *zero*, *one*, *plus*, *times*, *minus*, *inverse*} **..**

constdefs

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

consts

Reals :: 'a *set*

real :: 'a => *real*

syntax (*xsymbols*)
Reals :: 'a *set* (ℝ)

defs (**overloaded**)

real-zero-def:
0 == *Abs-Real*(*realrel*``{(*preal-of-rat 1*, *preal-of-rat 1*)})

real-one-def:
1 == *Abs-Real*(*realrel*``
{(*preal-of-rat 1* + *preal-of-rat 1*,
preal-of-rat 1)})

real-minus-def:
− *r* == *contents* (⋃(*x*,*y*) ∈ *Rep-Real*(*r*). { *Abs-Real*(*realrel*``{(*y*,*x*)}) })

real-add-def:
z + *w* ==
contents (⋃(*x*,*y*) ∈ *Rep-Real*(*z*). ⋃(*u*,*v*) ∈ *Rep-Real*(*w*).
{ *Abs-Real*(*realrel*``{(*x*+*u*, *y*+*v*)}) })

real-diff-def:
$r - (s::real) == r + - s$

real-mult-def:
$z * w ==$
$\quad contents\ (\bigcup(x,y) \in Rep\text{-}Real(z).\ \bigcup(u,v) \in Rep\text{-}Real(w).$
$\quad\quad \{\ Abs\text{-}Real(realrel``\{(x*u + y*v,\ x*v + y*u)\})\ \})$

real-inverse-def:
$inverse\ (R::real) == (SOME\ S.\ (R = 0\ \&\ S = 0) \mid S * R = 1)$

real-divide-def:
$R\ /\ (S::real) == R * inverse\ S$

real-le-def:
$z \leq (w::real) ==$
$\exists x\ y\ u\ v.\ x+v \leq u+y\ \&\ (x,y) \in Rep\text{-}Real\ z\ \&\ (u,v) \in Rep\text{-}Real\ w$

real-less-def: $(x < (y::real)) == (x \leq y\ \&\ x \neq y)$

real-abs-def: $abs\ (r::real) == (if\ 0 \leq r\ then\ r\ else\ -r)$

5.1 Proving that realrel is an equivalence relation

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*: *preal-add-ac*)
also have ... $= (x2 + y) + x1$ **by** (*simp add*: *prems*)
also have ... $= x2 + (x1 + y)$ **by** (*simp add*: *preal-add-ac*)
also have ... $= x2 + (x + y1)$ **by** (*simp add*: *prems*)
also have ... $= (x2 + y1) + x$ **by** (*simp add*: *preal-add-ac*)
finally have $(x1 + y2) + x = (x2 + y1) + x$ **.**
thus *?thesis* **by** (*simp add*: *preal-add-right-cancel-iff*)
qed

lemma *realrel-iff* [*simp*]: $(((x1,y1),(x2,y2)) \in realrel) = (x1 + y2 = x2 + y1)$
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: $(realrel\ ``\ \{x\} = realrel\ ``\ \{y\}) = ((x,\ y) \in realrel)$

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*)

lemma *inj-on-Abs-Real*: *inj-on Abs-Real Real*
apply (*rule inj-on-inverseI*)
apply (*erule Abs-Real-inverse*)
done

declare *inj-on-Abs-Real* [*THEN inj-on-iff*, *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

5.2 Congruence property for addition

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*: *preal-add-assoc*)
apply (*rule preal-add-left-commute* [*of ab*, *THEN ssubst*])
apply (*simp add*: *preal-add-assoc* [*symmetric*])
apply (*simp add*: *preal-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-add-commute*: $(z::real) + w = w + z$
by (*cases z*, *cases w*, *simp add*: *real-add preal-add-ac*)

lemma *real-add-assoc*: $((z1::real) + z2) + z3 = z1 + (z2 + z3)$
by (*cases z1*, *cases z2*, *cases z3*, *simp add*: *real-add preal-add-assoc*)

lemma *real-add-zero-left*: $(0::real) + z = z$
by (*cases z*, *simp add*: *real-add real-zero-def preal-add-ac*)

instance *real* :: *comm-monoid-add*
by (*intro-classes*,
(*assumption* |
rule real-add-commute real-add-assoc real-add-zero-left)+)

5.3 Additive Inverse on real

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 preal-add-commute*)
thus *?thesis*
by (*simp add*: *real-minus-def UN-equiv-class* [*OF equiv-realrel*])
qed

lemma *real-add-minus-left*: $(-z) + z = (0::real)$
by (*cases z*, *simp add*: *real-minus real-add real-zero-def preal-add-commute*)

5.4 Congruence property for 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*: *preal-add-left-commute preal-add-assoc* [*symmetric*])
apply (*simp add*: *preal-add-assoc preal-add-mult-distrib2* [*symmetric*])
apply (*simp add*: *preal-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*: *preal-mult-commute preal-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 preal-add-ac preal-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 preal-add-mult-distrib2 preal-add-ac preal-mult-ac)
done

lemma real-mult-1: (1::real) * z = z
apply (cases z)
apply (simp add: real-mult real-one-def preal-add-mult-distrib2
            preal-mult-1-right preal-mult-ac preal-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 preal-add-mult-distrib2
            preal-add-ac preal-mult-ac)
done
```

one and zero are distinct

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

5.5 existence of inverse

```
lemma real-zero-iff: Abs-Real (realrel `` {(x, x)}) = 0
by (simp add: real-zero-def preal-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 ≠ 0 ==> ∃ 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 `` { (preal-of-rat 1,
                          inverse (D) + preal-of-rat 1)})
      in exI)
```

apply (*rule-tac* [*2*]
  x = *Abs-Real* (*realrel* `` { (*inverse* (*D*) + *preal-of-rat 1*,
    preal-of-rat 1)})
 in *exI*)
apply (*auto simp add*: *real-mult preal-mult-1-right*
  preal-add-mult-distrib2 preal-add-mult-distrib preal-mult-1
  preal-mult-inverse-right preal-add-ac preal-mult-ac)
done

lemma *real-mult-inverse-left*: *x* ≠ *0* ==> *inverse*(*x*)∗*x* = (*1*::*real*)
apply (*simp add*: *real-inverse-def*)
apply (*frule real-mult-inverse-left-ex*, *safe*)
apply (*rule someI2*, *auto*)
done

5.6 The Real Numbers form a Field

instance *real* :: *field*
proof
 fix *x y z* :: *real*
 show − *x* + *x* = *0* **by** (*rule real-add-minus-left*)
 show *x* − *y* = *x* + (−*y*) **by** (*simp add*: *real-diff-def*)
 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** (*simp add*: *real-add-mult-distrib*)
 show *0* ≠ (*1*::*real*) **by** (*rule real-zero-not-eq-one*)
 show *x* ≠ *0* ==> *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

declare *minus-mult-right* [*symmetric*, *simp*]
  minus-mult-left [*symmetric*, *simp*]

lemma *real-mult-1-right*: *z* ∗ (*1*::*real*) = *z*
 by (*rule OrderedGroup.mult-1-right*)

5.7 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 preal-cancels*)
  hence $a+b \leq a+d$ **by** (*simp add*: *prems*)
  thus $b \leq d$ **by** (*simp add*: *preal-cancels*)
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*: *preal-add-ac*)
  also have ... $\leq (x2+y1) + (u2+v1)$
      by (*simp add*: *prems preal-add-le-cancel-left*)
  finally show *?thesis* **by** (*simp add*: *preal-add-le-cancel-right*)
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*: *preal-add-ac*)
  also have ... $\leq (u+y) + (u+v')$
    by (*simp add*: *preal-add-le-cancel-right prems*)

```
  also have ... ≤ (u+y) + (u'+v)
    by (simp add: preal-add-le-cancel-left prems)
  also have ... = (u'+y) + (u+v)  by (simp add: preal-add-ac)
  finally show ?thesis by (simp add: preal-add-le-cancel-right)
qed

lemma real-le-trans: [| i ≤ j; j ≤ k |] ==> i ≤ (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 ≤ z & w ≠ 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) ≤ w | w ≤ z
apply (cases z, cases w)
apply (auto simp add: real-le real-zero-def preal-add-ac preal-cancels)
done


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


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

lemma real-add-left-mono:
  assumes le: x ≤ y shows z + x ≤ 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) $\leq$ relation
apply (*auto dest*!: *less-add-left-Ex*
simp add: *preal-add-ac preal-mult-ac*
preal-add-mult-distrib2 preal-cancels 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

lemma for proving *0 < 1*

lemma *real-zero-le-one*: $0 \leq (1::real)$
by (*simp add*: *real-zero-def real-one-def real-le*
preal-self-less-add-left order-less-imp-le)

5.8 The Reals Form an Ordered Field

instance *real* :: *ordered-field*
proof
fix *x y z* :: *real*
show $x \leq y ==> z + x \leq z + y$ **by** (*rule real-add-left-mono*)
show $x < y ==> 0 < z ==> z * x < z * y$ **by** (*simp add*: *real-mult-less-mono2*)
show $|x| = (if\ x < 0\ then\ -x\ else\ x)$
by (*auto dest*: *order-le-less-trans simp add*: *real-abs-def linorder-not-le*)
qed

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 preal-add-mult-distrib preal-mult-1*
preal-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 preal-add-mult-distrib2*

preal-mult-1 preal-mult-1-right preal-add-ac preal-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 preal-add-ac*)
apply (*cut-tac x* = *x* **and** *y* = *y* **in** *linorder-less-linear*)
apply (*auto dest*!: *less-add-left-Ex simp add*: *preal-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 preal-cancels*)

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*]
preal-cancels)

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 (*auto simp add*: *real-zero-def real-of-preal-def real-less-def real-le-def*
preal-add-ac preal-cancels)
apply (*simp-all add*: *preal-add-assoc* [*symmetric*] *preal-cancels*)
apply (*blast intro*: *preal-self-less-add-left order-less-imp-le*)
apply (*insert preal-not-eq-self* [*of preal-of-rat 1 m*])
apply (*simp add*: *preal-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

5.9 Theorems About the Ordering

obsolete but used a lot

lemma *real-not-refl2*: $x < y \Longrightarrow x \neq (y::real)$
by *blast*

lemma *real-le-imp-less-or-eq*: $!!(x::real).\ x \leq y \Longrightarrow x < y \mid x = y$
by (*simp add*: *order-le-less*)

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*])

lemma *real-add-less-le-mono*: $[|\ w'<w;\ z'\leq z\ |] \Longrightarrow w' + z' < w + (z::real)$
by (*rule OrderedGroup.add-less-le-mono*)

lemma *real-add-le-less-mono*:
$!!z\ z'::real.\ [|\ w'\leq w;\ z'<z\ |] \Longrightarrow w' + z' < w + z$
by (*rule OrderedGroup.add-le-less-mono*)

lemma *real-le-square* [*simp*]: $(0::real) \leq x*x$
by (*rule Ring-and-Field.zero-le-square*)

5.10 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*

The precondition could be weakened to $(0::'a) \leq x$

lemma *real-mult-less-mono*:
[| $u<v$; $x<y$; $(0::real) < v$; $0 < x$ |] ==> $u*x < v* y$
by (*simp add*: *Ring-and-Field.mult-strict-mono order-less-imp-le*)

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

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

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

Only two uses?

lemma *real-mult-less-mono'*:
[| $x < y$; $r1 < r2$; $(0::real) \leq r1$; $0 \leq x$|] ==> $r1 * x < r2 * y$
by (*rule Ring-and-Field.mult-strict-mono'*)

FIXME: delete or at least combine the next two lemmas

lemma *real-sum-squares-cancel*: $x * x + y * y = 0$ ==> $x = (0::real)$
apply (*drule OrderedGroup.equals-zero-I* [*THEN sym*])
apply (*cut-tac* $x = y$ **in** *real-le-square*)
apply (*auto*, *drule order-antisym*, *auto*)
done

lemma *real-sum-squares-cancel2*: $x * x + y * y = 0$ ==> $y = (0::real)$
apply (*rule-tac* $y = x$ **in** *real-sum-squares-cancel*)
apply (*simp add*: *add-commute*)
done

lemma *real-add-order*: [| $0 < x$; $0 < y$ |] ==> $(0::real) < x + y$
by (*drule add-strict-mono* [*of* **concl**: *0 0*], *assumption*, *simp*)

lemma *real-le-add-order*: [| $0 \leq x$; $0 \leq y$ |] ==> $(0::real) \leq x + y$
apply (*drule order-le-imp-less-or-eq*)+
apply (*auto intro*: *real-add-order order-less-imp-le*)
done

lemma *real-inverse-unique*: $x*y = (1::real)$ ==> $y = inverse\ x$
apply (*case-tac* $x \neq 0$)
apply (*rule-tac c1* $= x$ **in** *real-mult-left-cancel* [*THEN iffD1*], *auto*)
done

lemma *real-inverse-gt-one*: [| $(0::real) < x$; $x < 1$ |] ==> $1 < inverse\ x$
by (*auto dest*: *less-imp-inverse-less*)

lemma *real-mult-self-sum-ge-zero*: $(0::real) \leq x*x + y*y$
proof −
have $0 + 0 \leq x*x + y*y$ **by** (*blast intro*: *add-mono zero-le-square*)
thus *?thesis* **by** *simp*
qed

5.11 Embedding the Integers into the Reals

defs (**overloaded**)
real-of-nat-def: *real z* == *of-nat z*
real-of-int-def: *real z* == *of-int z*

lemma *real-eq-of-nat*: *real* = *of-nat*
apply (*rule ext*)
apply (*unfold real-of-nat-def*)
apply (*rule refl*)
done

lemma *real-eq-of-int*: *real* = *of-int*
apply (*rule ext*)
apply (*unfold real-of-int-def*)
apply (*rule refl*)
done

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-eq-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*)

5.12 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*)

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*)

lemma *real-of-nat-gt-zero-cancel-iff* [*simp*]: (*0* < *real* (*n*::*nat*)) = (*0* < *n*)
by (*simp add*: *add*: *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*)) = (*0* ≤ *n*)
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-eq-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 assumption
  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 assumption
  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

5.13 Numerals and Arithmetic

instance *real* :: *number* **..**

defs (**overloaded**)
real-number-of-def: (*number-of w* :: *real*) == *of-int* (*Rep-Bin w*)
— the type constraint is essential!

instance *real* :: *number-ring*
by (*intro-classes*, *simp add*: *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*

setup *real-arith-setup*

5.14 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* ≤ (*0*::*real*)) = (*y* ≤ −*x*)

by *auto*

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

5.14.1 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*

5.15 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-interval-iff*: $(abs\ x < r) = (-r < x \ \& \ x < (r::real))$
by (*force simp add*: *Ring-and-Field.abs-less-iff*)

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 (*simp add*: *real-of-nat-ge-zero*)

lemma *abs-add-one-not-less-self* [*simp*]: $\sim\ abs(x) + (1::real) < x$
apply (*simp add*: *linorder-not-less*)
apply (*auto intro*: *abs-ge-self* [*THEN order-trans*])
done

Used only in Hyperreal/Lim.ML

lemma *abs-sum-triangle-ineq*: $abs\ ((x::real) + y + (-l + -m)) \leq abs(x + -l) + abs(y + -m)$
apply (*simp add*: *real-add-assoc*)
apply (*rule-tac a1* = y **in** *add-left-commute* [*THEN ssubst*])
apply (*rule real-add-assoc* [*THEN subst*])

apply (*rule abs-triangle-ineq*)
done

ML
⟨⟨
val real-lbound-gt-zero = thmreal-lbound-gt-zero;
val real-less-half-sum = thmreal-less-half-sum;
val real-gt-half-sum = thmreal-gt-half-sum;

val abs-interval-iff = thmabs-interval-iff;
val abs-le-interval-iff = thmabs-le-interval-iff;
val abs-add-one-gt-zero = thmabs-add-one-gt-zero;
val abs-add-one-not-less-self = thmabs-add-one-not-less-self;
val abs-sum-triangle-ineq = thmabs-sum-triangle-ineq;
⟩⟩

end

6 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*

6.1 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-of-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-of-preal* x
      using *neg-y* **by** $(rule\ real\text{-}less\text{-}all\text{-}real2)$
    thus $y <$ *real-of-preal* $(psup\ ?pP)$ **..**
  next
    assume $y <$ *real-of-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-of-preal py*
    by $(auto\ simp\ add{:}\ real\text{-}gt\text{-}zero\text{-}preal\text{-}Ex)$

  obtain a **where** *a-in-P*: $a \in P$ **using** *not-empty-P* **..**
  have *a-pos*: $0 < a$ **using** *positive-P* **..**
  then obtain *pa* **where** $a =$ *real-of-preal pa*
    by $(auto\ simp\ add{:}\ real\text{-}gt\text{-}zero\text{-}preal\text{-}Ex)$
  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* **..**
  hence $a <$ *sup* **..**
  hence $0 <$ *sup* **using** *a-pos* **by** *arith*
  then obtain *possup* **where** *sup* = *real-of-preal possup*
    by $(auto\ simp\ add{:}\ real\text{-}gt\text{-}zero\text{-}preal\text{-}Ex)$
  hence $\forall\, X \in$ *?pP*. $X \leq$ *possup*
    using *sup* **by** $(auto\ simp\ add{:}\ real\text{-}of\text{-}preal\text{-}lessI)$
  with *pP-not-empty* **have** *psup*: $\bigwedge Z.\ (\exists\, X \in$ *?pP*. $Z < X) = (Z <$ *psup ?pP*$)$
    by $(rule\ preal\text{-}complete)$

  show *?thesis*
  proof
    assume $\exists\, x \in P.\ y < x$
    then obtain x **where** *x-in-P*: $x \in P$ **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 $(auto\ simp\ add{:}\ real\text{-}gt\text{-}zero\text{-}preal\text{-}Ex)$

    have *py-less-X*: $\exists\, X \in$ *?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: ∀ x ∈ S. 0 < x
   and not-empty-S: ∃ x. x ∈ S
   and upper-bound-Ex: ∃ u. isUb (UNIV::real set) S u
 shows ∃ t. isLub (UNIV::real set) S t
proof
 let ?pS = {w. real-of-preal w ∈ S}

 obtain u where isUb UNIV S u using upper-bound-Ex ..
 hence sup: ∀ x ∈ S. x ≤ 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*: $\forall\, y{\in}S.\ y \leq x$
have $real\text{-}of\text{-}preal\ (psup\ ?pS) \leq x$
proof −
obtain s **where** *s-in-S*: $s \in S$ **using** *not-empty-S* **..**
hence *s-pos*: $0 < s$ **using** *positive-S* **by** *simp*

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

from *x-ub-S* **have** $s \leq x$ **using** *s-in-S* **..**
hence $0 < x$ **using** *s-pos* **by** *simp*
hence $\exists\ px.\ x = real\text{-}of\text{-}preal\ px$ **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 $\exists\, Z.$ *isUb UNIV ?SHIFT Z* **..**
moreover have $\forall\, y \in$ *?SHIFT*. $0 < y$ **by** *auto*
moreover have *shifted-not-empty*: $\exists\, u.\ u \in$ *?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 $t \leq (x + (-X) + 1)$
using *t-is-Lub* **by** (*simp add*: *isLub-le-isUb*)
hence $t + X + -1 \leq x$ **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*: $y \in S$
have $y \leq t + X + -1$
proof −
obtain u **where** *u-in-shift*: $u \in$ *?SHIFT* **using** *shifted-not-empty* **..**
hence $\exists\ x \in S.\ u = x + - X + 1$ **by** *simp*
then obtain x **where** *x-and-u*: $u = x + - X + 1$ **..**
have *u-le-t*: $u \leq t$ **using** *u-in-shift* **and** *t-is-Lub* **by** (*simp add*: *isLubD2*)

show *?thesis*
proof *cases*
assume $y \leq x$
moreover have $x = u + X + - 1$ **using** *x-and-u* **by** *arith*
moreover have $u + X + - 1 \leq t + X + -1$ **using** *u-le-t* **by** *arith*
ultimately show $y \leq t + X + -1$ **by** *arith*
next
assume $\sim(y \leq x)$
hence *x-less-y*: $x < y$ **by** *arith*

have $x + (-X) + 1 \in$ *?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 $y + (-X) + 1 \in$ *?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
```

6.2 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 add: real-mult-inverse-left mult-commute)
  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: mult-commute ring-eq-simps real-mult-inverse-left)
  thus ∃ (n::nat). x < real n ..
qed

lemma reals-Archimedean3:
  assumes x-greater-zero: 0 < x
  shows ∀ (y::real). ∃ (n::nat). y < real n * x
proof
  fix y
  have x-not-zero: x ≠ 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: mult-commute ring-eq-simps)
```

hence $y < real\ (n::nat) * x$
using *x-not-zero* **by** (*simp add*: *real-mult-inverse-left ring-eq-simps*)
thus $\exists (n::nat).\ y < real\ n * x$ **..**
qed

lemma *reals-Archimedean6*:
$0 \leq r ==> \exists (n::nat).\ real\ (n - 1) \leq r\ \&\ r < real\ (n)$
apply (*insert reals-Archimedean2* [*of r*], *safe*)
apply (*frule-tac* $P = \%k.\ r < real\ k$ **and** $k = n$ **and** $m = \%x.\ x$
in *ex-has-least-nat*, *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*)
done

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

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

ML
⟪
val real-sum-of-halves = thm real-sum-of-halves;
val posreal-complete = thm posreal-complete;
val real-isLub-unique = thm real-isLub-unique;
val posreals-complete = thm posreals-complete;
val reals-complete = thm reals-complete;
val reals-Archimedean = thm reals-Archimedean;
val reals-Archimedean2 = thm reals-Archimedean2;
val reals-Archimedean3 = thm reals-Archimedean3;
⟫

6.3 Floor and Ceiling Functions from the Reals to the Integers

constdefs

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

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

syntax (*xsymbols*)
floor :: *real* => *int* (⌊-⌋)
ceiling :: *real* => *int* (⌈-⌉)

syntax (*HTML* **output**)
floor :: *real* => *int* (⌊-⌋)
ceiling :: *real* => *int* (⌈-⌉)

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-real-of-nat* [*THEN ssubst*])
apply (*drule-tac* [*2*] *real-of-int-less-iff* [*THEN iffD1*])
apply (*simp-all add*: *real-of-int-real-of-nat*)
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-real-of-nat* [*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 add*: *real-of-int-real-of-nat*)
done

lemma *floor-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-real-of-nat* [*THEN ssubst*])
apply (*drule-tac* [*2*] *real-of-int-less-iff* [*THEN iffD1*], *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 (*drule-tac* [*2*] *real-of-int-real-of-nat* [*THEN ssubst*])
apply (*drule-tac* [*2*] *real-of-int-less-iff* [*THEN iffD1*], *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* **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*: *real-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* ==> *ceiling x* ≤ *ceiling y*

by (*simp add*: *floor-mono ceiling-def*)

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

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

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

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

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

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

lemma *ceiling-one* [*simp*]: *ceiling* $1 = 1$
  by (*unfold ceiling-def*, *simp*)

lemma *real-of-int-ceiling-diff-one-le* [*simp*]: *real* (*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*]: *real* (*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*: $x <=$ *real* $a \Longrightarrow$ *ceiling* $x <= a$
  apply (*unfold ceiling-def*)
  apply (*subgoal-tac* $-a <=$ *floor*$(-\ x)$)

apply *simp*
apply (*rule le-floor*)
apply *simp*
done

lemma *ceiling-le-real*: *ceiling x <= a ==> x <= real a*
apply (*unfold ceiling-def*)
apply (*subgoal-tac real(− a) <= − x*)
apply *simp*
apply (*rule real-le-floor*)
apply *simp*
done

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

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

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

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

lemma *less-ceiling-eq*: (*a < ceiling x*) = (*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*]:
(*number-of n < ceiling x*) = (*number-of n < x*)
by (*simp add*: *less-ceiling-eq*)

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

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

lemma *ceiling-less-eq*: (*ceiling x < a*) = (*x <= 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

6.4 Versions for the natural numbers

constdefs
natfloor :: *real* => *nat*
natfloor x == *nat*(*floor x*)
natceiling :: *real* => *nat*
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*)
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*
apply (*subst nat-diff-distrib*)
apply *simp*
apply (*rule le-floor*)
apply *simp-all*
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 (*subst nat-int* [*THEN sym*])**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
  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
  apply (subst nat-diff-distrib)
  apply simp
  apply (rule order-trans)
  prefer 2
  apply (rule ceiling-mono2)
```

```
  apply assumption
  apply simp-all
done

lemma natfloor-div-nat: 1 <= x ==> 0 < y ==>
  natfloor (x / real y) = natfloor x div y
proof -
  assume 1 <= (x::real) and 0 < (y::nat)
  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-distrib ring-eq-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 *assumption*
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

ML
⟪
val number-of-less-real-of-int-iff = *thm number-of-less-real-of-int-iff*;
val number-of-less-real-of-int-iff2 = *thm number-of-less-real-of-int-iff2*;
val number-of-le-real-of-int-iff = *thm number-of-le-real-of-int-iff*;
val number-of-le-real-of-int-iff2 = *thm number-of-le-real-of-int-iff2*;
val floor-zero = *thm floor-zero*;
val floor-real-of-nat-zero = *thm floor-real-of-nat-zero*;
val floor-real-of-nat = *thm floor-real-of-nat*;
val floor-minus-real-of-nat = *thm floor-minus-real-of-nat*;
val floor-real-of-int = *thm floor-real-of-int*;
val floor-minus-real-of-int = *thm floor-minus-real-of-int*;
val reals-Archimedean6 = *thm reals-Archimedean6*;
val reals-Archimedean6a = *thm reals-Archimedean6a*;
val reals-Archimedean-6b-int = *thm reals-Archimedean-6b-int*;
val reals-Archimedean-6c-int = *thm reals-Archimedean-6c-int*;
val real-lb-ub-int = *thm real-lb-ub-int*;
val lemma-floor = *thm lemma-floor*;
val real-of-int-floor-le = *thm real-of-int-floor-le*;
(∗*val floor-le* = *thm floor-le*;
val floor-le2 = *thm floor-le2*;
∗)
val lemma-floor2 = *thm lemma-floor2*;

val real-of-int-floor-cancel = thm real-of-int-floor-cancel;
val floor-eq = thm floor-eq;
val floor-eq2 = thm floor-eq2;
val floor-eq3 = thm floor-eq3;
val floor-eq4 = thm floor-eq4;
val floor-number-of-eq = thm floor-number-of-eq;
val real-of-int-floor-ge-diff-one = thm real-of-int-floor-ge-diff-one;
val real-of-int-floor-add-one-ge = thm real-of-int-floor-add-one-ge;
val ceiling-zero = thm ceiling-zero;
val ceiling-real-of-nat = thm ceiling-real-of-nat;
val ceiling-real-of-nat-zero = thm ceiling-real-of-nat-zero;
val ceiling-floor = thm ceiling-floor;
val floor-ceiling = thm floor-ceiling;
val real-of-int-ceiling-ge = thm real-of-int-ceiling-ge;
(*
val ceiling-le = thm ceiling-le;
val ceiling-le2 = thm ceiling-le2;
*)
val real-of-int-ceiling-cancel = thm real-of-int-ceiling-cancel;
val ceiling-eq = thm ceiling-eq;
val ceiling-eq2 = thm ceiling-eq2;
val ceiling-eq3 = thm ceiling-eq3;
val ceiling-real-of-int = thm ceiling-real-of-int;
val ceiling-number-of-eq = thm ceiling-number-of-eq;
val real-of-int-ceiling-diff-one-le = thm real-of-int-ceiling-diff-one-le;
val real-of-int-ceiling-le-add-one = thm real-of-int-ceiling-le-add-one;
⟩⟩

end

theory *RealPow*
imports *RealDef*
begin

declare *abs-mult-self* [*simp*]

instance *real* :: *power* **..**

primrec (*realpow*)
  realpow-0: *r ^ 0 = 1*
  realpow-Suc: *r ^ (Suc n) = (r::real) * (r ^ n)*

instance *real* :: *recpower*
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

lemma *realpow-not-zero*: $r \neq (0::real) ==> r\ \hat{}\ n \neq 0$
by (*rule field-power-not-zero*)

lemma *realpow-zero-zero*: $r\ \hat{}\ n = (0::real) ==> r = 0$
by *simp*

lemma *realpow-two*: $(r::real)\ \hat{}\ (Suc\ (Suc\ 0)) = r * r$
by *simp*

Legacy: weaker version of the theorem *power-strict-mono*, used 6 times in NthRoot and Transcendental

lemma *realpow-less*:
$[|(0::real) < x; x < y; 0 < n|] ==> x\ \hat{}\ n < y\ \hat{}\ n$
apply (*rule power-strict-mono*, *auto*)
done

lemma *realpow-two-le* [*simp*]: $(0::real) \leq r\hat{}\ Suc\ (Suc\ 0)$
by (*simp add*: *real-le-square*)

lemma *abs-realpow-two* [*simp*]: $abs((x::real)\hat{}Suc\ (Suc\ 0)) = x\hat{}Suc\ (Suc\ 0)$
by (*simp add*: *abs-mult*)

lemma *realpow-two-abs* [*simp*]: $abs(x::real)\hat{}Suc\ (Suc\ 0) = x\hat{}Suc\ (Suc\ 0)$
by (*simp add*: *power-abs* [*symmetric*] *del*: *realpow-Suc*)

lemma *two-realpow-ge-one* [*simp*]: $(1::real) \leq 2\ \hat{}\ n$
by (*insert power-increasing* [*of 0 n 2*::*real*], *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 real-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*)

Used ONCE in Transcendental

lemma *realpow-Suc-less-one*: $[|\ 0 < r; r < (1::real)\ |] ==> r\ \hat{}\ Suc\ n < 1$
by (*insert power-strict-decreasing* [*of 0 Suc n r*], *simp*)

Used ONCE in Lim.ML

lemma *realpow-minus-mult* [*rule-format*]:
  $0 < n \longrightarrow (x::real)\ \hat{}\ (n - 1) * x = x\ \hat{}\ n$
apply (*simp split add*: *nat-diff-split*)
done

lemma *realpow-two-mult-inverse* [*simp*]:
  $r \neq 0 \Longrightarrow r * inverse\ r\ \hat{}Suc\ (Suc\ 0) = inverse\ (r::real)$
by (*simp add*: *realpow-two 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*: *right-distrib left-distrib mult-ac*)
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*:
  $[|(0::real) \leq x;\ 0 \leq y;\ x\ \hat{}\ Suc\ n \leq y\ \hat{}\ Suc\ n|] \Longrightarrow x \leq y$
 by (*rule power-le-imp-le-base*)

lemma *zero-less-realpow-abs-iff* [*simp*]:
  $(0 < (abs\ x)\ \hat{}n) = (x \neq (0::real) \mid n=0)$
apply (*induct n*)
apply (*auto simp add*: *zero-less-mult-iff*)
done

lemma *zero-le-realpow-abs* [*simp*]: $(0::real) \leq (abs\ x)\ \hat{}n$
apply (*induct n*)
apply (*auto simp add*: *zero-le-mult-iff*)
done

6.5 Literal Arithmetic Involving Powers, Type *real*

lemma *real-of-int-power*: *real* (*x*::*int*) ^ *n* = *real* (*x* ^ *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-of v* :: *real*) ^ *n* = *real* ((*number-of v* :: *int*) ^ *n*)
by (*simp only*: *real-number-of* [*symmetric*] *real-of-int-power*)

declare *power-real-number-of* [*of* - *number-of w*, *standard*, *simp*]

6.6 Various Other Theorems

Used several times in Hyperreal/Transcendental.ML

lemma *real-sum-squares-cancel-a*: *x* * *x* = −(*y* * *y*) ==> *x* = (*0*::*real*) & *y*=*0*
 apply (*auto dest*: *real-sum-squares-cancel simp add*: *real-add-eq-0-iff* [*symmetric*])
 apply (*auto dest*: *real-sum-squares-cancel simp add*: *add-commute*)
 done

lemma *real-squared-diff-one-factored*: *x***x* − (*1*::*real*) = (*x* + *1*)*(*x* − *1*)
by (*auto simp add*: *left-distrib right-distrib real-diff-def*)

lemma *real-mult-is-one* [*simp*]: (*x***x* = (*1*::*real*)) = (*x* = *1* | *x* = − *1*)
apply *auto*
apply (*drule right-minus-eq* [*THEN iffD2*])
apply (*auto simp add*: *real-squared-diff-one-factored*)
done

lemma *real-le-add-half-cancel*: (*x* + *y*/*2* ≤ (*y*::*real*)) = (*x* ≤ *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* |]
     ==> *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

Used once: in Hyperreal/Transcendental.ML

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 *auto*

lemma *inverse-real-of-nat-ge-zero* [*simp*]: *0* ≤ *inverse* (*real* (*Suc n*))
by *auto*

lemma *real-sum-squares-not-zero*: *x* ~= *0* ==> *x* * *x* + *y* * *y* ~= (*0*::*real*)
by (*blast dest*!: *real-sum-squares-cancel*)

lemma *real-sum-squares-not-zero2*: *y* ~= *0* ==> *x* * *x* + *y* * *y* ~= (*0*::*real*)
by (*blast dest*!: *real-sum-squares-cancel2*)

6.7 Various Other Theorems

lemma *realpow-divide*:
(*x*/*y*) ^ *n* = ((*x*::*real*) ^ *n*/ *y* ^ *n*)
apply (*unfold real-divide-def*)
apply (*auto simp add*: *power-mult-distrib power-inverse*)
done

lemma *realpow-two-sum-zero-iff* [*simp*]:
(*x* ^ *2* + *y* ^ *2* = (*0*::*real*)) = (*x* = *0* & *y* = *0*)
apply (*auto intro*: *real-sum-squares-cancel real-sum-squares-cancel2*
simp add: *power2-eq-square*)
done

lemma *realpow-two-le-add-order* [*simp*]: (*0*::*real*) ≤ *u* ^ *2* + *v* ^ *2*
apply (*rule real-le-add-order*)
apply (*auto simp add*: *power2-eq-square*)
done

lemma *realpow-two-le-add-order2* [*simp*]: (*0*::*real*) ≤ *u* ^ *2* + *v* ^ *2* + *w* ^ *2*
apply (*rule real-le-add-order*)+
apply (*auto simp add*: *power2-eq-square*)
done

lemma *real-sum-square-gt-zero*: *x* ~= *0* ==> (*0*::*real*) < *x* * *x* + *y* * *y*
apply (*cut-tac x* = *x* **and** *y* = *y* **in** *real-mult-self-sum-ge-zero*)
apply (*drule real-le-imp-less-or-eq*)
apply (*drule-tac y* = *y* **in** *real-sum-squares-not-zero*, *auto*)
done

lemma *real-sum-square-gt-zero2*: *y* ~= *0* ==> (*0*::*real*) < *x* * *x* + *y* * *y*
apply (*rule real-add-commute* [*THEN subst*])
apply (*erule real-sum-square-gt-zero*)

done

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 *realpow-num-eq-if*: $(m::real) \hat{} n = (if\ n=0\ then\ 1\ else\ m * m \hat{} (n - 1))$
by (*case-tac n*, *auto*)

lemma *real-num-zero-less-two-pow* [*simp*]: $0 < (2::real) \hat{} (4*d)$
apply (*induct d*)
apply (*auto simp add*: *realpow-num-eq-if*)
done

lemma *lemma-realpow-num-two-mono*:
$x * (4::real) < y ==> x * (2 \hat{} 8) < y * (2 \hat{} 6)$
apply (*subgoal-tac* $(2::real) \hat{} 8 = 4 * (2 \hat{} 6)$)
apply (*simp* (*no-asm-simp*) *add*: *real-mult-assoc* [*symmetric*])
apply (*auto simp add*: *realpow-num-eq-if*)
done

ML
⟪
val realpow-0 = thm realpow-0;
val realpow-Suc = thm realpow-Suc;

val realpow-not-zero = thm realpow-not-zero;
val realpow-zero-zero = thm realpow-zero-zero;
val realpow-two = thm realpow-two;
val realpow-less = thm realpow-less;
val realpow-two-le = thm realpow-two-le;
val abs-realpow-two = thm abs-realpow-two;
val realpow-two-abs = thm realpow-two-abs;
val two-realpow-ge-one = thm two-realpow-ge-one;
val two-realpow-gt = thm two-realpow-gt;
val realpow-Suc-le-self = thm realpow-Suc-le-self;
val realpow-Suc-less-one = thm realpow-Suc-less-one;
val realpow-minus-mult = thm realpow-minus-mult;
val realpow-two-mult-inverse = thm realpow-two-mult-inverse;
val realpow-two-minus = thm realpow-two-minus;
val realpow-two-disj = thm realpow-two-disj;
val realpow-real-of-nat = thm realpow-real-of-nat;
val realpow-real-of-nat-two-pos = thm realpow-real-of-nat-two-pos;
val realpow-increasing = thm realpow-increasing;
val zero-less-realpow-abs-iff = thm zero-less-realpow-abs-iff;
val zero-le-realpow-abs = thm zero-le-realpow-abs;

val real-of-int-power = thm real-of-int-power;
val power-real-number-of = thm power-real-number-of;
val real-sum-squares-cancel-a = thm real-sum-squares-cancel-a;
val real-mult-inverse-cancel2 = thm real-mult-inverse-cancel2;
val real-squared-diff-one-factored = thm real-squared-diff-one-factored;
val real-mult-is-one = thm real-mult-is-one;
val real-le-add-half-cancel = thm real-le-add-half-cancel;
val real-minus-half-eq = thm real-minus-half-eq;
val real-mult-inverse-cancel = thm real-mult-inverse-cancel;
val real-mult-inverse-cancel2 = thm real-mult-inverse-cancel2;
val inverse-real-of-nat-gt-zero = thm inverse-real-of-nat-gt-zero;
val inverse-real-of-nat-ge-zero = thm inverse-real-of-nat-ge-zero;
val real-sum-squares-not-zero = thm real-sum-squares-not-zero;
val real-sum-squares-not-zero2 = thm real-sum-squares-not-zero2;

val realpow-divide = thm realpow-divide;
val realpow-two-sum-zero-iff = thm realpow-two-sum-zero-iff;
val realpow-two-le-add-order = thm realpow-two-le-add-order;
val realpow-two-le-add-order2 = thm realpow-two-le-add-order2;
val real-sum-square-gt-zero = thm real-sum-square-gt-zero;
val real-sum-square-gt-zero2 = thm real-sum-square-gt-zero2;
val real-minus-mult-self-le = thm real-minus-mult-self-le;
val realpow-square-minus-le = thm realpow-square-minus-le;
val realpow-num-eq-if = thm realpow-num-eq-if;
val real-num-zero-less-two-pow = thm real-num-zero-less-two-pow;
val lemma-realpow-num-two-mono = thm lemma-realpow-num-two-mono;
⟩⟩

end

theory *Real*
imports *RComplete RealPow*
begin
end

theory *Float* **imports** *Real* **begin**

constdefs
pow2 :: *int* ⇒ *real*
pow2 a == *if* (*0* <= *a*) *then* (*2*^(*nat a*)) *else* (*inverse* (*2*^(*nat* (−*a*))))
float :: *int* ∗ *int* ⇒ *real*
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*])
    apply (*auto simp add*: *h*)
    apply *arith*
    done
  show *?thesis*
  proof (*induct a*)
    case (*1 n*)
    from *pos* **show** *?case* **by** (*simp add*: *ring-eq-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-eq-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-eq-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-eq-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-eq-simps)
    done
 qed
qed

lemma float (a, e) + float (b, e) = float (a + b, e)
by (simp add: float-def ring-eq-simps)

constdefs
  int-of-real :: real ⇒ int
  int-of-real x == SOME y. real y = x
  real-is-int :: real ⇒ bool
  real-is-int x == ? (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::real) ^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*::*bin*⇒*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 !! *y*. *real-is-int* ((*number-of*::*bin*⇒*real*) (*Abs-Bin x*))
      apply (*simp add*: *number-of-eq*)
      apply (*subst Abs-Bin-inverse*)
      apply (*simp add*: *Bin-def*)
      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 :: bin
  have ? u. x = Abs-Bin u
   apply (rule exI[where x = Rep-Bin x])
   apply (simp add: Rep-Bin-inverse)
   done
 }
 then obtain u::int where x = Abs-Bin 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-eq-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*)
apply (*subst split-div*, *auto*)
apply (*subst split-zdiv*, *auto*)
apply (*rule-tac a*=*int* (*b* ∗ *i*) + *int j* **and** *b*=*int b* **and** *r*=*int j* **and** *r′*=*ja* **in** *IntDiv.unique-quotient*)
apply (*auto simp add*: *IntDiv.quorem-def int-eq-of-nat*)
done

lemma *int-mod-zmod*: *int* (*a mod b*) = (*int a*) *mod* (*int b*)
apply (*subst split-mod*, *auto*)
apply (*subst split-zmod*, *auto*)
apply (*rule-tac a*=*int* (*b* ∗ *i*) + *int j* **and** *b*=*int b* **and** *q*=*int i* **and** *q′*=*ia* **in** *IntDiv.unique-remainder*)
apply (*auto simp add*: *IntDiv.quorem-def int-eq-of-nat*)
done

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

ML ⟪ *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*: *terminating-norm-float*)
ML ⟪ *simp-depth-limit* := *1000* ⟫

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 pow2-int: pow2 (int n) = 2^n
 by (simp add: pow2-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-eq-simps)
 apply (auto simp add: pow2-int[symmetric] pow2-add[symmetric])
 done

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 *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*: $1 * a = (a::'a::semiring\text{-}1)$
by *simp*

lemma *mult-right-one*: $a * 1 = (a::'a::semiring\text{-}1)$
by *simp*

lemma *int-pow-0*: $(a::int)\hat{}(Numeral0) = 1$
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* =
Pls-0-eq Min-1-eq
bin-pred-Pls bin-pred-Min bin-pred-1 bin-pred-0
bin-succ-Pls bin-succ-Min bin-succ-1 bin-succ-0
bin-add-Pls bin-add-Min bin-add-BIT-0 bin-add-BIT-10
bin-add-BIT-11 bin-minus-Pls bin-minus-Min bin-minus-1
bin-minus-0 bin-mult-Pls bin-mult-Min bin-mult-1 bin-mult-0
bin-add-Pls-right bin-add-Min-right

lemma *int-eq-number-of-eq*: $(((number\text{-}of\ v)::int)=(number\text{-}of\ w)) = iszero\ ((number\text{-}of\ (bin\text{-}add\ v\ (bin\text{-}minus\ w)))::int)$
by *simp*

lemma *int-iszero-number-of-Pls*: $iszero\ (Numeral0::int)$
by (*simp only*: *iszero-number-of-Pls*)

lemma *int-nonzero-number-of-Min*: $\sim(iszero\ ((-1)::int))$
by *simp*

lemma *int-iszero-number-of-0*: $iszero\ ((number\text{-}of\ (w\ BIT\ bit.B0))::int) = iszero\ ((number\text{-}of\ w)::int)$
by *simp*

lemma *int-iszero-number-of-1*: $\neg\ iszero\ ((number\text{-}of\ (w\ BIT\ bit.B1))::int)$
by *simp*

lemma *int-less-number-of-eq-neg*: $(((number\text{-}of\ x)::int)\ <\ number\text{-}of\ y)\ =\ neg\ ((number\text{-}of\ (bin\text{-}add\ x\ (bin\text{-}minus\ y)))::int)$
by *simp*

lemma *int-not-neg-number-of-Pls*: $\neg\ (neg\ (Numeral0::int))$
by *simp*

lemma *int-neg-number-of-Min*: $neg\ (-1::int)$
by *simp*

lemma *int-neg-number-of-BIT*: $neg\ ((number\text{-}of\ (w\ BIT\ x))::int) = neg\ ((number\text{-}of\ w)::int)$
by *simp*

lemma *int-le-number-of-eq*: $(((number\text{-}of\ x)::int) \leq number\text{-}of\ y) = (\neg\ neg\ ((number\text{-}of\ (bin\text{-}add\ y\ (bin\text{-}minus\ x)))::int))$
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-0
lift-bool[*OF int-iszero-number-of-1*] *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-BIT int-le-number-of-eq

lemma *int-number-of-add-sym*: ((*number-of v*)::*int*) + *number-of w* = *number-of* (*bin-add v w*)
by *simp*

lemma *int-number-of-diff-sym*: ((*number-of v*)::*int*) − *number-of w* = *number-of* (*bin-add v* (*bin-minus w*))
by *simp*

lemma *int-number-of-mult-sym*: ((*number-of v*)::*int*) ∗ *number-of w* = *number-of* (*bin-mult v w*)
by *simp*

lemma *int-number-of-minus-sym*: − ((*number-of v*)::*int*) = *number-of* (*bin-minus 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-mult float-minus float-abs zero-le-float float-pprt float-nprt*

lemmas *arith* = *binarith intarith intarithrel natarith powerarith floatarith not-false-eq-true not-true-eq-false*

end

7 Zorn: Zorn's Lemma

theory *Zorn*
imports *Main*

begin

The lemma and section numbers refer to an unpublished article [**?**].

constdefs
chain :: *'a set set => 'a set set set*
chain S == {*F*. *F* ⊆ *S* & (∀ *x* ∈ *F*. ∀ *y* ∈ *F*. *x* ⊆ *y* | *y* ⊆ *x*)}

super :: [*'a set set*,*'a set set*] => *'a set set set*
super S c == {*d*. *d* ∈ *chain S* & *c* ⊂ *d*}

maxchain :: *'a set set => 'a set set set*
maxchain S == {*c*. *c* ∈ *chain S* & *super S c* = {}}

succ :: [*'a set set*,*'a set set*] => *'a set set*
succ S c ==
if c ∉ *chain S* | *c* ∈ *maxchain S*
then c else SOME c'. *c'* ∈ *super S c*

consts
TFin :: *'a set set => 'a set set set*

inductive *TFin S*
intros
succI: *x* ∈ *TFin S* ==> *succ S x* ∈ *TFin S*
Pow-UnionI: *Y* ∈ *Pow*(*TFin S*) ==> *Union*(*Y*) ∈ *TFin S*
monos *Pow-mono*

7.1 Mathematical Preamble

lemma *Union-lemma0*:
(∀ *x* ∈ *C*. *x* ⊆ *A* | *B* ⊆ *x*) ==> *Union*(*C*)<=*A* | *B* ⊆ *Union*(*C*)
by *blast*

This is theorem *increasingD2* of ZF/Zorn.thy

lemma *Abrial-axiom1*: *x* ⊆ *succ S x*
apply (*unfold succ-def*)
apply (*rule split-if* [*THEN iffD2*])
apply (*auto simp add*: *super-def maxchain-def psubset-def*)
apply (*rule swap*, *assumption*)
apply (*rule someI2*, *blast*+)
done

lemmas *TFin-UnionI* = *TFin.Pow-UnionI* [*OF PowI*]

lemma *TFin-induct*:
[| *n* ∈ *TFin S*;
!!*x*. [| *x* ∈ *TFin S*; *P*(*x*) |] ==> *P*(*succ S x*);
!!*Y*. [| *Y* ⊆ *TFin S*; *Ball Y P* |] ==> *P*(*Union Y*) |]
==> *P*(*n*)

```
 apply (erule TFin.induct)
  apply blast+
 done
```

```
lemma succ-trans: x ⊆ y ==> x ⊆ succ S y
 apply (erule subset-trans)
 apply (rule Abrial-axiom1)
 done
```

Lemma 1 of section 3.1

```
lemma TFin-linear-lemma1:
    [| n ∈ TFin S;  m ∈ TFin S;
       ∀x ∈ TFin S. x ⊆ m --> x = m | succ S x ⊆ m
    |] ==> n ⊆ m | succ S m ⊆ 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 ∈ TFin S ==> ∀n ∈ TFin S. n ⊆ m --> n=m | succ S n ⊆ 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 ⊆ m;  m ∈ TFin S;  n ∈ TFin S |] ==> n=m | succ S n ⊆ m
 by (rule TFin-linear-lemma2 [rule-format])
```

Consequences from section 3.3 – Property 3.2, the ordering is total

lemma *TFin-subset-linear*: [| $m \in TFin\ S$; $n \in TFin\ S$|] ==> $n \subseteq m \mid m \subseteq n$
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 (*rule-tac* [*2*] *eq-succ-upper* [*THEN Union-least*])
apply (*assumption+*)
apply (*erule ssubst*)
apply (*rule Abrial-axiom1* [*THEN equalityI*])
apply (*blast del*: *subsetI intro*: *subsetI TFin-UnionI TFin.succI*)
done

7.2 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*) *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*: $c \in chain\ S - maxchain\ S$ ==> $?\ d.\ d \in super\ S\ c$
by (*unfold super-def maxchain-def*) *auto*

lemma *select-super*: $c \in chain\ S - maxchain\ S$ ==>
$(\epsilon\ c'.\ c' \text{: } super\ S\ c) \text{: } super\ S\ c$
apply (*erule mem-super-Ex* [*THEN exE*])
apply (*rule someI2*, *auto*)

done

lemma *select-not-equals*: $c \in chain\ S - maxchain\ S \Longrightarrow (\epsilon\, c'.\ c' \colon super\ S\ c) \neq c$
apply (*rule notI*)
apply (*drule select-super*)
apply (*simp add*: *super-def psubset-def*)
done

lemma *succI3*: $c \in chain\ S - maxchain\ S \Longrightarrow succ\ S\ c = (\epsilon\, c'.\ c' \colon super\ S\ c)$
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*: $c \in TFin\ S \Longrightarrow (c :: 'a\ set\ set) \colon chain\ S$
apply (*erule TFin-induct*)
apply (*simp add*: *succ-def select-super* [*THEN super-subset-chain*[*THEN subsetD*]])
apply (*unfold chain-def*)
apply (*rule CollectI*, *safe*)
apply (*drule bspec*, *assumption*)
apply (*rule-tac* [*2*] *m1* = *Xa* **and** *n1* = *X* **in** *TFin-subset-linear* [*THEN disjE*], *blast*+)
done

theorem *Hausdorff*: $\exists\, c.\ (c :: 'a\ set\ set) \colon maxchain\ S$
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

7.3 Zorn's Lemma: If All Chains Have Upper Bounds Then There Is a Maximal Element

lemma *chain-extend*:
$[\!|\ c \in chain\ S;\ z \in S;\ \forall x \in c.\ x <= (z :: 'a\ set)\ |\!] \Longrightarrow \{z\}\ Un\ c \in chain\ S$
by (*unfold chain-def*) *blast*

lemma *chain-Union-upper*: $[\!|\ c \in chain\ S;\ x \in c\ |\!] \Longrightarrow x \subseteq Union(c)$
by (*unfold chain-def*) *auto*

lemma *chain-ball-Union-upper*: $c \in chain\ S \Longrightarrow \forall x \in c.\ x \subseteq Union(c)$
by (*unfold chain-def*) *auto*

lemma *maxchain-Zorn*:
$[|\ c \in maxchain\ S;\ u \in S;\ Union(c) \subseteq u\ |] \Longrightarrow Union(c) = u$
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 psubset-def*)
apply (*blast intro*: *chain-extend dest*: *chain-Union-upper*)
done

theorem *Zorn-Lemma*:
$\forall c \in chain\ S.\ Union(c)$: $S \Longrightarrow \exists y \in S.\ \forall z \in S.\ y \subseteq z \longrightarrow y = z$
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

7.4 Alternative version of Zorn's Lemma

lemma *Zorn-Lemma2*:
$\forall c \in chain\ S.\ \exists y \in S.\ \forall x \in c.\ x \subseteq y$
$\Longrightarrow \exists y \in S.\ \forall x \in S.\ (y :: 'a\ set) \subseteq x \longrightarrow y = x$
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 psubset-def*)
apply (*blast elim*!: *equalityCE*)
done

Various other lemmas

lemma *chainD*: $[|\ c \in chain\ S;\ x \in c;\ y \in c\ |] \Longrightarrow x \subseteq y \mid y \subseteq x$
by (*unfold chain-def*) *blast*

lemma *chainD2*: !!(*c* :: *'a set set*). *c* ∈ *chain S* ==> *c* ⊆ *S*
by (*unfold chain-def*) *blast*

end

8 Filter: Filters and Ultrafilters

theory *Filter*
imports *Zorn*
begin

8.1 Definitions and basic properties

8.1.1 Filters

locale *filter* =
fixes *F* :: *'a set set*
assumes *UNIV* [*iff*]: *UNIV* ∈ *F*
assumes *empty* [*iff*]: {} ∉ *F*
assumes *Int*: ⟦*u* ∈ *F*; *v* ∈ *F*⟧ ⟹ *u* ∩ *v* ∈ *F*
assumes *subset*: ⟦*u* ∈ *F*; *u* ⊆ *v*⟧ ⟹ *v* ∈ *F*

lemma (**in** *filter*) *memD*: *A* ∈ *F* ⟹ − *A* ∉ *F*
proof
assume *A* ∈ *F* **and** − *A* ∈ *F*
hence *A* ∩ (− *A*) ∈ *F* **by** (*rule Int*)
thus *False* **by** *simp*
qed

lemma (**in** *filter*) *not-memI*: − *A* ∈ *F* ⟹ *A* ∉ *F*
by (*drule memD*, *simp*)

lemma (**in** *filter*) *Int-iff*: (*x* ∩ *y* ∈ *F*) = (*x* ∈ *F* ∧ *y* ∈ *F*)
by (*auto elim*: *subset intro*: *Int*)

8.1.2 Ultrafilters

locale *ultrafilter* = *filter* +
assumes *ultra*: *A* ∈ *F* ∨ − *A* ∈ *F*

lemma (**in** *ultrafilter*) *memI*: − *A* ∉ *F* ⟹ *A* ∈ *F*
by (*cut-tac ultra* [*of A*], *simp*)

lemma (**in** *ultrafilter*) *not-memD*: *A* ∉ *F* ⟹ − *A* ∈ *F*
by (*rule memI*, *simp*)

lemma (**in** *ultrafilter*) *not-mem-iff*: (*A* ∉ *F*) = (− *A* ∈ *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

8.1.3 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*) *filter*: *filter* F **.**

lemma (**in** *freeultrafilter*) *ultrafilter*: *ultrafilter* F
by (*rule ultrafilter.intro*)

8.2 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 $\{n.\ P\ n\ (SOME\ x.\ P\ n\ x)\} \in F$
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

8.3 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 $F \subseteq G$ **.**
  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

8.4 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* $\equiv \{A.\ finite\ (-A)\}$
  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* ∈ *?F* **by** *simp*
next
  show {} ∉ *?F* **by** *simp*
next
  fix *u v* **assume** *u* ∈ *?F* **and** *v* ∈ *?F*
  thus *u* ∩ *v* ∈ *?F* **by** *simp*
next
  fix *u v* **assume** *uv*: *u* ⊆ *v* **and** *u*: *u* ∈ *?F*
  from *uv* **have** *vu*: − *v* ⊆ − *u* **by** *simp*
  from *u* **show** *v* ∈ *?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* ∈ *superfrechet*
by (*rule superfrechetI* [*OF filter-frechet subset-refl*])

lemma (**in** *UFT*) *lemma-mem-chain-filter*:
  ⟦*c* ∈ *chain superfrechet*; *x* ∈ *c*⟧ $\Longrightarrow$ *filter x*
by (*unfold chain-def superfrechet-def*, *blast*)

8.4.1 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* ∈ *chain superfrechet*; *c* ≠ {}⟧ $\Longrightarrow$ *UNIV* ∈ $\bigcup c$
proof −
  assume *1*: *c* ∈ *chain superfrechet* **and** *2*: *c* ≠ {}

from *2* **obtain** x **where** *3*: $x \in c$ **by** *blast*
from *1 3* **have** *filter x* **by** (*rule lemma-mem-chain-filter*)
hence $UNIV \in x$ **by** (*rule filter.UNIV*)
with *3* **show** $UNIV \in \bigcup c$ **by** *blast*
qed

lemma (**in** *UFT*) *Union-chain-empty*:
$c \in$ *chain superfrechet* $\Longrightarrow \{\} \notin \bigcup c$
proof
assume *1*: $c \in$ *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*:
$[\![c \in$ *chain superfrechet*; $u \in \bigcup c$; $v \in \bigcup c]\!] \Longrightarrow u \cap v \in \bigcup c$
proof −
assume *c*: $c \in$ *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*:
$[\![c \in$ *chain superfrechet*; $u \in \bigcup c$; $u \subseteq v]\!] \Longrightarrow v \in \bigcup c$
proof −
assume *c*: $c \in$ *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 $c \in$ *chain superfrechet* **and** $c \neq \{\}$
shows *filter* $(\bigcup c)$
proof (*rule filter.intro*)

show $UNIV \in \bigcup c$ **by** (*rule Union-chain-UNIV*)
next
show $\{\} \notin \bigcup c$ **by** (*rule Union-chain-empty*)
next
fix u v **assume** $u \in \bigcup c$ **and** $v \in \bigcup c$
show $u \cap v \in \bigcup c$ **by** (*rule Union-chain-Int*)
next
fix u v **assume** $u \in \bigcup c$ **and** $u \subseteq v$
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

8.4.2 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*:
$[\![U \in superfrechet;\ A \in U]\!] \Longrightarrow infinite\ A$
proof
assume *U*: $U \in superfrechet$ **and** *A*: $A \in U$ **and** *fin*: $finite\ A$
from *U* **have** *fil*: $filter\ U$ **and** *fre*: $frechet \subseteq U$

```
    by (simp-all add: superfrechet-def)
  from fin have - A ∈ frechet by (simp add: frechet-def)
  with fre have cA: - A ∈ U by (rule subsetD)
  from fil A cA have A ∩ - A ∈ 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:
  ∃ U::'a set set. freeultrafilter U
proof -
  from max-cofinite-filter-Ex obtain U
    where U: U ∈ superfrechet
      and max [rule-format]: ∀ G∈superfrechet. U ⊆ G ⟶ U = G ..
  from U have fil: filter U by (rule superfrechetD1)
  from U have fre: frechet ⊆ U by (rule superfrechetD2)
  have ultra: ultrafilter-axioms U
  proof (rule filter.max-filter-ultrafilter [OF fil])
    fix G assume G: filter G and UG: U ⊆ G
    from fre UG have frechet ⊆ G by simp
    with G have G ∈ 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 ∈ U
    with U show infinite A by (rule mem-superfrechet-all-infinite)
  qed
  from fil ultra free have freeultrafilter U
    by (rule freeultrafilter.intro)
  thus ?thesis ..
qed

lemmas freeultrafilter-Ex = UFT.freeultrafilter-ex

end
```

9 StarDef: Construction of Star Types Using Ultrafilters

```
theory StarDef
imports Filter
uses (transfer.ML)
begin
```

9.1 A Free Ultrafilter over the Naturals

constdefs
FreeUltrafilterNat :: *nat set set* ($\mathcal{U}$)
$\mathcal{U} \equiv$ *SOME U. freeultrafilter U*

lemma *freeultrafilter-FUFNat*: *freeultrafilter* $\mathcal{U}$
apply (*unfold FreeUltrafilterNat-def*)
apply (*rule someI-ex*)
apply (*rule freeultrafilter-Ex*)
apply (*rule nat-infinite*)
done

interpretation *FUFNat*: *freeultrafilter* [*FreeUltrafilterNat*]
by (*cut-tac* [!] *freeultrafilter-FUFNat*, *simp-all add*: *freeultrafilter-def*)

This rule takes the place of the old ultra tactic

lemma *ultra*:
$[\![\{n.\ P\ n\} \in \mathcal{U};\ \{n.\ P\ n \longrightarrow Q\ n\} \in \mathcal{U}]\!] \Longrightarrow \{n.\ Q\ n\} \in \mathcal{U}$
by (*simp add*: *Collect-imp-eq FUFNat.F.Un-iff FUFNat.F.Compl-iff*)

9.2 Definition of *star* type constructor

constdefs
starrel :: $((nat \Rightarrow 'a) \times (nat \Rightarrow 'a))$ *set*
starrel $\equiv \{(X,Y).\ \{n.\ X\ n = Y\ n\} \in \mathcal{U}\}$

typedef *'a star* = (*UNIV* :: $(nat \Rightarrow 'a)$ *set*) // *starrel*
by (*auto intro*: *quotientI*)

constdefs
star-n :: $(nat \Rightarrow 'a) \Rightarrow$ *'a star*
star-n X $\equiv$ *Abs-star* (*starrel* `` $\{X\}$)

theorem *star-cases* [*case-names star-n*, *cases type*: *star*]:
$(\bigwedge X.\ x = \text{star-n}\ X \Longrightarrow P) \Longrightarrow P$
by (*cases x*, *unfold star-n-def star-def*, *erule quotientE*, *fast*)

lemma *all-star-eq*: $(\forall x.\ P\ x) = (\forall X.\ P\ (\text{star-n}\ X))$
by (*auto*, *rule-tac x=x* **in** *star-cases*, *simp*)

lemma *ex-star-eq*: $(\exists x.\ P\ x) = (\exists X.\ P\ (\text{star-n}\ X))$
by (*auto*, *rule-tac x=x* **in** *star-cases*, *auto*)

Proving that *starrel* is an equivalence relation

lemma *starrel-iff* [*iff*]: $((X,Y) \in \text{starrel}) = (\{n.\ X\ n = Y\ n\} \in \mathcal{U})$
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*: (*star-n X* = *star-n Y*) = ({*n*. *X n* = *Y n*} ∈ *U*)
by (*simp add*: *star-n-def Abs-star-inject starrel-in-star equiv-starrel-iff*)

9.3 Transfer principle

This introduction rule starts each transfer proof.

lemma *transfer-start*:
P ≡ {*n*. *Q*} ∈ *U* ⟹ *Trueprop P* ≡ *Trueprop Q*
by (*subgoal-tac P* ≡ *Q*, *simp*, *simp add*: *atomize-eq*)

Initialize transfer tactic.

use *transfer.ML*
setup *Transfer.setup*

Transfer introduction rules.

lemma *transfer-ex* [*transfer-intro*]:
⟦⋀*X*. *p* (*star-n X*) ≡ {*n*. *P n* (*X n*)} ∈ *U*⟧
⟹ ∃ *x*::'*a star*. *p x* ≡ {*n*. ∃ *x*. *P n x*} ∈ *U*
by (*simp only*: *ex-star-eq FUFNat.F.Collect-ex*)

lemma *transfer-all* [*transfer-intro*]:
⟦⋀*X*. *p* (*star-n X*) ≡ {*n*. *P n* (*X n*)} ∈ *U*⟧
⟹ ∀ *x*::'*a star*. *p x* ≡ {*n*. ∀ *x*. *P n x*} ∈ *U*
by (*simp only*: *all-star-eq FUFNat.F.Collect-all*)

lemma *transfer-not* [*transfer-intro*]:
⟦*p* ≡ {*n*. *P n*} ∈ *U*⟧ ⟹ ¬ *p* ≡ {*n*. ¬ *P n*} ∈ *U*
by (*simp only*: *FUFNat.F.Collect-not*)

lemma *transfer-conj* [*transfer-intro*]:
⟦*p* ≡ {*n*. *P n*} ∈ *U*; *q* ≡ {*n*. *Q n*} ∈ *U*⟧
⟹ *p* ∧ *q* ≡ {*n*. *P n* ∧ *Q n*} ∈ *U*
by (*simp only*: *FUFNat.F.Collect-conj*)

lemma *transfer-disj* [*transfer-intro*]:
⟦*p* ≡ {*n*. *P n*} ∈ *U*; *q* ≡ {*n*. *Q n*} ∈ *U*⟧
⟹ *p* ∨ *q* ≡ {*n*. *P n* ∨ *Q n*} ∈ *U*

by (*simp only*: *FUFNat.F.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*)

9.4 Standard elements

constdefs
star-of :: *'a* ⇒ *'a star*
star-of x ≡ *star-n* ($\lambda n.$ *x*)

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*: $(\mathit{star\text{-}of}\ x = \mathit{star\text{-}of}\ y) = (x = y)$
by (*transfer*, *rule refl*)

9.5 Internal functions

constdefs
$\mathit{Ifun} :: ('a \Rightarrow 'b)\ \mathit{star} \Rightarrow 'a\ \mathit{star} \Rightarrow 'b\ \mathit{star}$ (- $\star$ - [*300*,*301*] *300*)
$\mathit{Ifun}\ f \equiv \lambda x.\ \mathit{Abs\text{-}star}$
$(\bigcup F{\in}\mathit{Rep\text{-}star}\ f.\ \bigcup X{\in}\mathit{Rep\text{-}star}\ x.\ \mathit{starrel}``\{\lambda n.\ F\ n\ (X\ n)\})$

lemma *Ifun-congruent2*:
$(\lambda F\ X.\ \mathit{starrel}``\{\lambda n.\ F\ n\ (X\ n)\})\ \mathit{respects2}\ \mathit{starrel}$
by (*auto simp add*: *congruent2-def equiv-starrel-iff elim*!: *ultra*)

lemma *Ifun-star-n*: $\mathit{star\text{-}n}\ F \star \mathit{star\text{-}n}\ X = \mathit{star\text{-}n}\ (\lambda n.\ F\ n\ (X\ n))$
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 $\langle\!\langle$ [*Transfer.add-const StarDef.Ifun*] $\rangle\!\rangle$

lemma *transfer-Ifun* [*transfer-intro*]:
$[\![f \equiv \mathit{star\text{-}n}\ F;\ x \equiv \mathit{star\text{-}n}\ X]\!] \Longrightarrow f \star x \equiv \mathit{star\text{-}n}\ (\lambda n.\ F\ n\ (X\ n))$
by (*simp only*: *Ifun-star-n*)

lemma *Ifun-star-of* [*simp*]: $\mathit{star\text{-}of}\ f \star \mathit{star\text{-}of}\ x = \mathit{star\text{-}of}\ (f\ x)$
by (*transfer*, *rule refl*)

Nonstandard extensions of functions

constdefs
$\mathit{starfun} :: ('a \Rightarrow 'b) \Rightarrow ('a\ \mathit{star} \Rightarrow 'b\ \mathit{star})$
$(*f*$ - [*80*] *80*)
$\mathit{starfun}\ f \equiv \lambda x.\ \mathit{star\text{-}of}\ f \star x$

$\mathit{starfun2} :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow ('a\ \mathit{star} \Rightarrow 'b\ \mathit{star} \Rightarrow 'c\ \mathit{star})$
$(*f2*$ - [*80*] *80*)
$\mathit{starfun2}\ f \equiv \lambda x\ y.\ \mathit{star\text{-}of}\ f \star x \star y$

declare *starfun-def* [*transfer-unfold*]
declare *starfun2-def* [*transfer-unfold*]

lemma *starfun-star-n*: $(\ *f*\ f)\ (\mathit{star\text{-}n}\ X) = \mathit{star\text{-}n}\ (\lambda n.\ f\ (X\ n))$
by (*simp only*: *starfun-def star-of-def Ifun-star-n*)

lemma *starfun2-star-n*:
$(\ *f2*\ f)\ (\mathit{star\text{-}n}\ X)\ (\mathit{star\text{-}n}\ Y) = \mathit{star\text{-}n}\ (\lambda n.\ f\ (X\ n)\ (Y\ n))$
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*)

9.6 Internal predicates

constdefs
unstar :: *bool star* ⇒ *bool*
unstar b ≡ *b* = *star-of True*

lemma *unstar-star-n*: $unstar\ (star\text{-}n\ P) = (\{n.\ P\ n\} \in \mathcal{U})$
by (*simp add*: *unstar-def star-of-def star-n-eq-iff*)

lemma *unstar-star-of* [*simp*]: $unstar\ (star\text{-}of\ p) = p$
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*]:
$p \equiv star\text{-}n\ P \Longrightarrow unstar\ p \equiv \{n.\ P\ n\} \in \mathcal{U}$
by (*simp only*: *unstar-star-n*)

constdefs
starP :: ('*a* ⇒ *bool*) ⇒ '*a star* ⇒ *bool*
(∗*p*∗ - [*80*] *80*)
$*p* P \equiv \lambda x.\ unstar\ (star\text{-}of\ P \star x)$

starP2 :: ('*a* ⇒ '*b* ⇒ *bool*) ⇒ '*a star* ⇒ '*b star* ⇒ *bool*
(∗*p2*∗ - [*80*] *80*)
$*p2* P \equiv \lambda x\ y.\ unstar\ (star\text{-}of\ P \star x \star y)$

declare *starP-def* [*transfer-unfold*]
declare *starP2-def* [*transfer-unfold*]

lemma *starP-star-n*: $(*p* P) \ (star\text{-}n\ X) = (\{n.\ P\ (X\ n)\} \in \mathcal{U})$
by (*simp only*: *starP-def star-of-def Ifun-star-n unstar-star-n*)

lemma *starP2-star-n*:
$(*p2* P) \ (star\text{-}n\ X) \ (star\text{-}n\ Y) = (\{n.\ P\ (X\ n)\ (Y\ n)\} \in \mathcal{U})$
by (*simp only*: *starP2-def star-of-def Ifun-star-n unstar-star-n*)

lemma *starP-star-of* [*simp*]: $(*p* P) \ (star\text{-}of\ x) = P\ x$
by (*transfer*, *rule refl*)

lemma *starP2-star-of* [*simp*]: $(*p2* P) \ (star\text{-}of\ x) = *p* P\ x$
by (*transfer*, *rule refl*)

9.7 Internal sets

constdefs
Iset :: *'a set star ⇒ 'a star set*
Iset A ≡ {*x*. (*∗p2∗ op* ∈) *x A*}

lemma *Iset-star-n*:
(*star-n X* ∈ *Iset* (*star-n A*)) = ({*n*. *X n* ∈ *A n*} ∈ $\mathcal{U}$)
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*} ∈ $\mathcal{U}$
by (*simp only*: *Iset-star-n*)

lemma *transfer-Collect* [*transfer-intro*]:
⟦⋀*X*. *p* (*star-n X*) ≡ {*n*. *P n* (*X n*)} ∈ $\mathcal{U}$⟧
⟹ *Collect p* ≡ *Iset* (*star-n* (*λn*. *Collect* (*P n*)))
by (*simp add*: *atomize-eq expand-set-eq all-star-eq Iset-star-n*)

lemma *transfer-set-eq* [*transfer-intro*]:
⟦*a* ≡ *Iset* (*star-n A*); *b* ≡ *Iset* (*star-n B*)⟧
⟹ *a* = *b* ≡ {*n*. *A n* = *B n*} ∈ $\mathcal{U}$
by (*simp only*: *expand-set-eq transfer-all transfer-iff transfer-mem*)

lemma *transfer-ball* [*transfer-intro*]:
⟦*a* ≡ *Iset* (*star-n A*); ⋀*X*. *p* (*star-n X*) ≡ {*n*. *P n* (*X n*)} ∈ $\mathcal{U}$⟧
⟹ ∀ *x*∈*a*. *p x* ≡ {*n*. ∀ *x*∈*A n*. *P n x*} ∈ $\mathcal{U}$
by (*simp only*: *Ball-def transfer-all transfer-imp transfer-mem*)

lemma *transfer-bex* [*transfer-intro*]:
⟦*a* ≡ *Iset* (*star-n A*); ⋀*X*. *p* (*star-n X*) ≡ {*n*. *P n* (*X n*)} ∈ $\mathcal{U}$⟧
⟹ ∃ *x*∈*a*. *p x* ≡ {*n*. ∃ *x*∈*A n*. *P n x*} ∈ $\mathcal{U}$
by (*simp only*: *Bex-def transfer-ex transfer-conj transfer-mem*)

lemma *transfer-Iset* [*transfer-intro*]:
⟦*a* ≡ *star-n A*⟧ ⟹ *Iset a* ≡ *Iset* (*star-n* (*λn*. *A n*))
by *simp*

Nonstandard extensions of sets.

constdefs
starset :: *'a set ⇒ 'a star set* (*∗s∗* - [*80*] *80*)
starset A ≡ *Iset* (*star-of A*)

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

lemma *starset-mem*: (*star-of x* ∈ *∗s∗ A*) = (*x* ∈ *A*)

by (*transfer*, *rule refl*)

lemma *starset-UNIV*: $*s*\ (UNIV::'a\ set) = (UNIV::'a\ star\ set)$
by (*transfer UNIV-def*, *rule refl*)

lemma *starset-empty*: $*s*\ \{\} = \{\}$
by (*transfer empty-def*, *rule refl*)

lemma *starset-insert*: $*s*\ (insert\ x\ A) = insert\ (star\text{-}of\ x)\ (\ *s*\ A)$
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-def*, *rule refl*)

lemma *starset-diff*: $*s*\ (A - B) = *s*\ A - *s*\ B$
by (*transfer set-diff-def*, *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-def*, *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*

end

10 StarClasses: Class Instances

theory *StarClasses*

imports *StarDef*
begin

10.1 Syntactic classes

instance *star* :: (*ord*) *ord* **..**
instance *star* :: (*zero*) *zero* **..**
instance *star* :: (*one*) *one* **..**
instance *star* :: (*plus*) *plus* **..**
instance *star* :: (*times*) *times* **..**
instance *star* :: (*minus*) *minus* **..**
instance *star* :: (*inverse*) *inverse* **..**
instance *star* :: (*number*) *number* **..**
instance *star* :: (*Divides.div*) *Divides.div* **..**
instance *star* :: (*power*) *power* **..**

defs (**overloaded**)
star-zero-def: $0 \equiv$ *star-of 0*
star-one-def: $1 \equiv$ *star-of 1*
star-number-def: *number-of b* $\equiv$ *star-of* (*number-of b*)
star-add-def: $(op +) \equiv *f2*\ (op +)$
star-diff-def: $(op -) \equiv *f2*\ (op -)$
star-minus-def: *uminus* $\equiv *f*$ *uminus*
star-mult-def: $(op *) \equiv *f2*\ (op *)$
star-divide-def: $(op /) \equiv *f2*\ (op /)$
star-inverse-def: *inverse* $\equiv *f*$ *inverse*
star-le-def: $(op \leq) \equiv *p2*\ (op \leq)$
star-less-def: $(op <) \equiv *p2*\ (op <)$
star-abs-def: *abs* $\equiv *f*$ *abs*
star-div-def: $(op\ div) \equiv *f2*\ (op\ div)$
star-mod-def: $(op\ mod) \equiv *f2*\ (op\ mod)$
star-power-def: $(op\ \hat{}) \equiv \lambda x\ n.\ (*f*\ (\lambda x.\ x\ \hat{}\ n))\ x$

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-div-def *star-mod-def* *star-power-def*

star-of preserves class operations

lemma *star-of-add*: *star-of* $(x + y) =$ *star-of* $x +$ *star-of* y
by *transfer* (*rule refl*)

lemma *star-of-diff*: *star-of* $(x - y) =$ *star-of* $x -$ *star-of* y
by *transfer* (*rule refl*)

lemma *star-of-minus*: *star-of* $(-x) = -$ *star-of* x
by *transfer* (*rule refl*)

lemma *star-of-mult*: *star-of* $(x * y) = star\text{-}of\ x * star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-divide*: *star-of* $(x\ /\ y) = star\text{-}of\ x\ /\ star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-inverse*: *star-of* $(inverse\ x) = inverse\ (star\text{-}of\ x)$
by *transfer* (*rule refl*)

lemma *star-of-div*: *star-of* $(x\ div\ y) = star\text{-}of\ x\ div\ star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-mod*: *star-of* $(x\ mod\ y) = star\text{-}of\ x\ mod\ star\text{-}of\ y$
by *transfer* (*rule refl*)

lemma *star-of-power*: *star-of* $(x\ \hat{}\ n) = star\text{-}of\ x\ \hat{}\ n$
by *transfer* (*rule refl*)

lemma *star-of-abs*: *star-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* $(number\text{-}of\ x) = number\text{-}of\ x$
by *transfer* (*rule refl*)

star-of preserves orderings

lemma *star-of-less*: $(star\text{-}of\ x < star\text{-}of\ y) = (x < y)$
by *transfer* (*rule refl*)

lemma *star-of-le*: $(star\text{-}of\ x \leq star\text{-}of\ y) = (x \leq y)$
by *transfer* (*rule refl*)

lemma *star-of-eq*: $(star\text{-}of\ x = star\text{-}of\ y) = (x = y)$
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*

10.2 Ordering classes

instance *star* :: (*order*) *order*
apply (*intro-classes*)
apply (*transfer*, *rule order-refl*)
apply (*transfer*, *erule* (*1*) *order-trans*)
apply (*transfer*, *erule* (*1*) *order-antisym*)
apply (*transfer*, *rule order-less-le*)
done

instance *star* :: (*linorder*) *linorder*
by (*intro-classes*, *transfer*, *rule linorder-linear*)

10.3 Lattice ordering classes

Some extra trouble is necessary because the class axioms for *meet* and *join* use quantification over function spaces.

lemma *ex-star-fun*:
$\exists f{::}('a \Rightarrow 'b)\ star.\ P\ (\lambda x.\ f \star x)$
$\Longrightarrow \exists f{::}'a\ star \Rightarrow 'b\ star.\ P\ f$
by (*erule exE*, *erule exI*)

lemma *ex-star-fun2*:
$\exists f{::}('a \Rightarrow 'b \Rightarrow 'c)\ star.\ P\ (\lambda x\ y.\ f \star x \star y)$
$\Longrightarrow \exists f{::}'a\ star \Rightarrow 'b\ star \Rightarrow 'c\ star.\ P\ f$
by (*erule exE*, *erule exI*)

instance *star* :: (*join-semilorder*) *join-semilorder*
apply (*intro-classes*)
apply (*rule ex-star-fun2*)
apply (*transfer is-join-def*)
apply (*rule join-exists*)
done

instance *star* :: (*meet-semilorder*) *meet-semilorder*
apply (*intro-classes*)
apply (*rule ex-star-fun2*)
apply (*transfer is-meet-def*)
apply (*rule meet-exists*)
done

instance *star* :: (*lorder*) *lorder* **..**

lemma *star-join-def* [*transfer-unfold*]: *join* $\equiv$ $*f2*$ *join*
apply (*rule is-join-unique* [*OF is-join-join*, *THEN eq-reflection*])
apply (*transfer is-join-def*, *rule is-join-join*)
done

lemma *star-meet-def* [*transfer-unfold*]: *meet* $\equiv$ $*f2*$ *meet*
apply (*rule is-meet-unique* [*OF is-meet-meet*, *THEN eq-reflection*])
apply (*transfer is-meet-def*, *rule is-meet-meet*)
done

10.4 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.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-ab-group-add*) *pordered-ab-group-add* **..**
instance *star* :: (*ordered-cancel-ab-semigroup-add*) *ordered-cancel-ab-semigroup-add*
..
instance *star* :: (*lordered-ab-group-meet*) *lordered-ab-group-meet* **..**

instance *star* :: (*lordered-ab-group-meet*) *lordered-ab-group-meet* **..**
instance *star* :: (*lordered-ab-group*) *lordered-ab-group* **..**

instance *star* :: (*lordered-ab-group-abs*) *lordered-ab-group-abs*
by (*intro-classes*, *transfer*, *rule abs-lattice*)

10.5 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* **..**
instance *star* :: (*semiring-0-cancel*) *semiring-0-cancel* **..**

instance *star* :: (*comm-semiring*) *comm-semiring*
by (*intro-classes*, *transfer*, *rule distrib*)

instance *star* :: (*comm-semiring-0*) *comm-semiring-0* **..**
instance *star* :: (*comm-semiring-0-cancel*) *comm-semiring-0-cancel* **..**

instance *star* :: (*axclass-0-neq-1*) *axclass-0-neq-1*
by (*intro-classes*, *transfer*, *rule zero-neq-one*)

instance *star* :: (*semiring-1*) *semiring-1* **..**
instance *star* :: (*comm-semiring-1*) *comm-semiring-1* **..**

instance *star* :: (*axclass-no-zero-divisors*) *axclass-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* :: (*idom*) *idom* **..**

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 pordered-comm-semiring-class.mult-mono*)

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.mult-strict-mono*)

instance *star* :: (*pordered-ring*) *pordered-ring* **..**
instance *star* :: (*lordered-ring*) *lordered-ring* **..**

instance *star* :: (*axclass-abs-if*) *axclass-abs-if*
by (*intro-classes*, *transfer*, *rule abs-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* **..**

10.6 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 ^ 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

10.7 Number classes

lemma *star-of-nat-def* [*transfer-unfold*]: *of-nat n* ≡ *star-of* (*of-nat n*)
by (*rule eq-reflection*, *induct-tac n*, *simp-all*)

lemma *star-of-of-nat* [*simp*]: *star-of* (*of-nat n*) = *of-nat n*
by *transfer* (*rule refl*)

lemma *int-diff-cases*:
assumes *prem*: $\bigwedge m\ n.\ z = int\ m - int\ n \Longrightarrow P$ **shows** *P*
apply (*rule-tac z=z* **in** *int-cases*)
apply (*rule-tac m=n* **and** *n=0* **in** *prem*, *simp*)
apply (*rule-tac m=0* **and** *n=Suc n* **in** *prem*, *simp*)
done — Belongs in Integ/IntDef.thy

lemma *star-of-int-def* [*transfer-unfold*]: *of-int z* ≡ *star-of* (*of-int z*)
by (*rule eq-reflection*, *rule-tac z=z* **in** *int-diff-cases*, *simp*)

lemma *star-of-of-int* [*simp*]: *star-of* (*of-int z*) = *of-int z*
by *transfer* (*rule refl*)

instance *star* :: (*number-ring*) *number-ring*
by (*intro-classes*, *simp only*: *star-number-def star-of-int-def number-of-eq*)

10.8 Finite class

lemma *starset-finite*: *finite A* ⟹ $*s*\ A$ = *star-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

11 HyperDef: Construction of Hyperreals Using Ultrafilters

theory *HyperDef*
imports *StarClasses* ../*Real*/*Real*

uses (*fuf.ML*)
begin

types *hypreal* = *real star*

syntax *hypreal-of-real* :: *real* => *real star*
translations *hypreal-of-real* => *star-of* :: *real* => *real star*

constdefs

omega :: *hypreal* — an infinite number = [<*1*,*2*,*3*,...>]
omega == *star-n* (%*n*. *real* (*Suc n*))

epsilon :: *hypreal* — an infinitesimal number = [<*1*,*1*/*2*,*1*/*3*,...>]
epsilon == *star-n* (%*n*. *inverse* (*real* (*Suc n*)))

syntax (*xsymbols*)
omega :: *hypreal* (ω)
epsilon :: *hypreal* (ε)

syntax (*HTML* **output**)
omega :: *hypreal* (ω)
epsilon :: *hypreal* (ε)

11.1 Existence of Free Ultrafilter over the Naturals

Also, proof of various properties of $\mathcal{U}$: an arbitrary free ultrafilter

lemma *FreeUltrafilterNat-Ex*: $\exists$ *U*::*nat set set*. *freeultrafilter U*
by (*rule nat-infinite* [*THEN freeultrafilter-Ex*])

lemma *FreeUltrafilterNat-mem*: *freeultrafilter FreeUltrafilterNat*
apply (*unfold FreeUltrafilterNat-def*)
apply (*rule someI-ex*)
apply (*rule FreeUltrafilterNat-Ex*)
done

lemma *UltrafilterNat-mem*: *ultrafilter FreeUltrafilterNat*
by (*rule FreeUltrafilterNat-mem* [*THEN freeultrafilter.ultrafilter*])

lemma *FilterNat-mem*: *filter FreeUltrafilterNat*
by (*rule FreeUltrafilterNat-mem* [*THEN freeultrafilter.filter*])

lemma *FreeUltrafilterNat-finite*: *finite x* ==> *x* $\notin$ *FreeUltrafilterNat*
by (*rule FreeUltrafilterNat-mem* [*THEN freeultrafilter.finite*])

lemma *FreeUltrafilterNat-not-finite*: *x* $\in$ *FreeUltrafilterNat* ==> ~ *finite x*
by (*rule FreeUltrafilterNat-mem* [*THEN freeultrafilter.infinite*])

lemma *FreeUltrafilterNat-empty* [*simp*]: {} $\notin$ *FreeUltrafilterNat*

by (*rule FilterNat-mem* [*THEN filter.empty*])

lemma *FreeUltrafilterNat-Int*:
[| $X \in$ *FreeUltrafilterNat*; $Y \in$ *FreeUltrafilterNat* |]
==> X *Int* $Y \in$ *FreeUltrafilterNat*
by (*rule FilterNat-mem* [*THEN filter.Int*])

lemma *FreeUltrafilterNat-subset*:
[| $X \in$ *FreeUltrafilterNat*; $X \subseteq Y$ |]
==> $Y \in$ *FreeUltrafilterNat*
by (*rule FilterNat-mem* [*THEN filter.subset*])

lemma *FreeUltrafilterNat-Compl*:
$X \in$ *FreeUltrafilterNat* ==> $-X \notin$ *FreeUltrafilterNat*
apply (*erule contrapos-pn*)
apply (*erule UltrafilterNat-mem* [*THEN ultrafilter.not-mem-iff*, *THEN iffD2*])
done

lemma *FreeUltrafilterNat-Compl-mem*:
$X \notin$ *FreeUltrafilterNat* ==> $-X \in$ *FreeUltrafilterNat*
by (*rule UltrafilterNat-mem* [*THEN ultrafilter.not-mem-iff*, *THEN iffD1*])

lemma *FreeUltrafilterNat-Compl-iff1*:
($X \notin$ *FreeUltrafilterNat*) = ($-X \in$ *FreeUltrafilterNat*)
by (*rule UltrafilterNat-mem* [*THEN ultrafilter.not-mem-iff*])

lemma *FreeUltrafilterNat-Compl-iff2*:
($X \in$ *FreeUltrafilterNat*) = ($-X \notin$ *FreeUltrafilterNat*)
by (*auto simp add*: *FreeUltrafilterNat-Compl-iff1* [*symmetric*])

lemma *cofinite-mem-FreeUltrafilterNat*: *finite* $(-X)$ ==> $X \in$ *FreeUltrafilterNat*
apply (*drule FreeUltrafilterNat-finite*)
apply (*simp add*: *FreeUltrafilterNat-Compl-iff2* [*symmetric*])
done

lemma *FreeUltrafilterNat-UNIV* [*iff*]: *UNIV* $\in$ *FreeUltrafilterNat*
by (*rule FilterNat-mem* [*THEN filter.UNIV*])

lemma *FreeUltrafilterNat-Nat-set-refl* [*intro*]:
$\{n.\ P(n) = P(n)\} \in$ *FreeUltrafilterNat*
by *simp*

lemma *FreeUltrafilterNat-P*: $\{n$::*nat*. $P\} \in$ *FreeUltrafilterNat* ==> P
by (*rule ccontr*, *simp*)

lemma *FreeUltrafilterNat-Ex-P*: $\{n.\ P(n)\} \in$ *FreeUltrafilterNat* ==> $\exists n.\ P(n)$
by (*rule ccontr*, *simp*)

lemma *FreeUltrafilterNat-all*: $\forall n.\ P(n)$ ==> $\{n.\ P(n)\} \in$ *FreeUltrafilterNat*

by (*auto*)

Define and use Ultrafilter tactics

use *fuf.ML*

method-setup *fuf* = ⟪
Method.ctxt-args (*fn ctxt* =>
Method.METHOD (*fn facts* =>
fuf-tac (*local-clasimpset-of ctxt*) *1*)) ⟫
free ultrafilter tactic

method-setup *ultra* = ⟪
Method.ctxt-args (*fn ctxt* =>
Method.METHOD (*fn facts* =>
ultra-tac (*local-clasimpset-of ctxt*) *1*)) ⟫
ultrafilter tactic

One further property of our free ultrafilter

lemma *FreeUltrafilterNat-Un*:
X Un Y ∈ *FreeUltrafilterNat*
==> *X* ∈ *FreeUltrafilterNat* | *Y* ∈ *FreeUltrafilterNat*
by (*auto*, *ultra*)

11.2 Properties of *starrel*

Proving that *starrel* is an equivalence relation

lemma *starrel-iff*: ((*X*, *Y*) ∈ *starrel*) = ({*n*. *X n* = *Y n*} ∈ *FreeUltrafilterNat*)
by (*rule StarDef.starrel-iff*)

lemma *starrel-refl*: (*x*,*x*) ∈ *starrel*
by (*simp add*: *starrel-def*)

lemma *starrel-sym* [*rule-format* (*no-asm*)]: (*x*,*y*) ∈ *starrel* −−> (*y*,*x*) ∈ *starrel*
by (*simp add*: *starrel-def eq-commute*)

lemma *starrel-trans*:
[|(*x*,*y*) ∈ *starrel*; (*y*,*z*) ∈ *starrel*|] ==> (*x*,*z*) ∈ *starrel*
by (*simp add*: *starrel-def*, *ultra*)

lemma *equiv-starrel*: *equiv UNIV starrel*
by (*rule StarDef.equiv-starrel*)

lemmas *equiv-starrel-iff* =
eq-equiv-class-iff [*OF equiv-starrel UNIV-I UNIV-I*, *simp*]

lemma *starrel-in-hypreal* [*simp*]: *starrel*``{*x*}:*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*]

lemmas *eq-starrelD* = *eq-equiv-class* [*OF* - *equiv-starrel*]

lemma *lemma-starrel-refl* [*simp*]: $x \in \mathit{starrel}\ ``\ \{x\}$
by (*simp add*: *starrel-def*)

lemma *hypreal-empty-not-mem* [*simp*]: $\{\} \notin \mathit{star}$
apply (*simp add*: *star-def*)
apply (*auto elim*!: *quotientE equalityCE*)
done

lemma *Rep-hypreal-nonempty* [*simp*]: *Rep-star x* $\neq \{\}$
by (*insert Rep-star* [*of x*], *auto*)

11.3 *star-of*: the Injection from *real* to *hypreal*

lemma *inj-hypreal-of-real*: *inj*(*hypreal-of-real*)
by (*rule inj-onI*, *simp*)

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*

11.4 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*:
$\mathit{inverse}\ (\mathit{star\text{-}n}\ X) = \mathit{star\text{-}n}\ (\%n.\ \mathit{inverse}(X\ n))$
by (*simp only*: *star-inverse-def starfun-star-n*)

lemma *star-n-le*:
star-n X $\leq$ *star-n Y* =
({*n*. *X n* $\leq$ *Y n*} $\in$ *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*} $\in$ *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*)

11.5 Misc Others

lemma *hypreal-not-refl2*: !!(*x*::*hypreal*). *x* < *y* ==> *x* $\neq$ *y*
by (*auto*)

lemma *hypreal-eq-minus-iff*: ((*x*::*hypreal*) = *y*) = (*x* + - *y* = *0*)
by *auto*

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

lemma *hypreal-mult-right-cancel*: (*c*::*hypreal*) $\neq$ *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*)

11.6 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*} = {} |
($\exists$ *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*)

ML
⟪
val omega-def = *thm omega-def*;
val epsilon-def = *thm epsilon-def*;

val FreeUltrafilterNat-Ex = *thm FreeUltrafilterNat-Ex*;
val FreeUltrafilterNat-mem = *thm FreeUltrafilterNat-mem*;
val FreeUltrafilterNat-finite = *thm FreeUltrafilterNat-finite*;
val FreeUltrafilterNat-not-finite = *thm FreeUltrafilterNat-not-finite*;
val FreeUltrafilterNat-empty = *thm FreeUltrafilterNat-empty*;
val FreeUltrafilterNat-Int = *thm FreeUltrafilterNat-Int*;
val FreeUltrafilterNat-subset = *thm FreeUltrafilterNat-subset*;
val FreeUltrafilterNat-Compl = *thm FreeUltrafilterNat-Compl*;

val FreeUltrafilterNat-Compl-mem = thm FreeUltrafilterNat-Compl-mem;
val FreeUltrafilterNat-Compl-iff1 = thm FreeUltrafilterNat-Compl-iff1;
val FreeUltrafilterNat-Compl-iff2 = thm FreeUltrafilterNat-Compl-iff2;
val FreeUltrafilterNat-UNIV = thm FreeUltrafilterNat-UNIV;
val FreeUltrafilterNat-Nat-set-refl = thm FreeUltrafilterNat-Nat-set-refl;
val FreeUltrafilterNat-P = thm FreeUltrafilterNat-P;
val FreeUltrafilterNat-Ex-P = thm FreeUltrafilterNat-Ex-P;
val FreeUltrafilterNat-all = thm FreeUltrafilterNat-all;
val FreeUltrafilterNat-Un = thm FreeUltrafilterNat-Un;
val starrel-iff = thm starrel-iff;
val starrel-in-hypreal = thm starrel-in-hypreal;
val Abs-star-inverse = thm Abs-star-inverse;
val lemma-starrel-refl = thm lemma-starrel-refl;
val hypreal-empty-not-mem = thm hypreal-empty-not-mem;
val Rep-hypreal-nonempty = thm Rep-hypreal-nonempty;
val inj-hypreal-of-real = thm inj-hypreal-of-real;
(∗ *val eq-Abs-star = thm eq-Abs-star;* ∗)
val star-n-minus = thm star-n-minus;
val star-n-add = thm star-n-add;
val star-n-diff = thm star-n-diff;
val star-n-mult = thm star-n-mult;
val star-n-inverse = thm star-n-inverse;
val hypreal-mult-left-cancel = thm hypreal-mult-left-cancel;
val hypreal-mult-right-cancel = thm hypreal-mult-right-cancel;
val hypreal-not-refl2 = thm hypreal-not-refl2;
val star-n-less = thm star-n-less;
val hypreal-eq-minus-iff = thm hypreal-eq-minus-iff;
val star-n-le = thm star-n-le;
val star-n-zero-num = thm star-n-zero-num;
val star-n-one-num = thm star-n-one-num;
val hypreal-omega-gt-zero = thm hypreal-omega-gt-zero;

val lemma-omega-empty-singleton-disj = thmlemma-omega-empty-singleton-disj;
val lemma-finite-omega-set = thmlemma-finite-omega-set;
val not-ex-hypreal-of-real-eq-omega = thmnot-ex-hypreal-of-real-eq-omega;
val hypreal-of-real-not-eq-omega = thmhypreal-of-real-not-eq-omega;
val not-ex-hypreal-of-real-eq-epsilon = thmnot-ex-hypreal-of-real-eq-epsilon;
val hypreal-of-real-not-eq-epsilon = thmhypreal-of-real-not-eq-epsilon;
val hypreal-epsilon-not-zero = thmhypreal-epsilon-not-zero;
val hypreal-epsilon-inverse-omega = thmhypreal-epsilon-inverse-omega;
⟩⟩

end

12 HyperArith: Binary arithmetic and Simplification for the Hyperreals

theory *HyperArith*
imports *HyperDef*
uses (*hypreal-arith.ML*)
begin

12.1 Numerals and Arithmetic

use *hypreal-arith.ML*

setup *hypreal-arith-setup*

12.2 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*)

used once in NSA

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*::*hypreal*) | *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*)

lemma *hypreal-of-real-hrabs*:
abs (*hypreal-of-real r*) = *hypreal-of-real* (*abs r*)
by (*rule star-of-abs* [*symmetric*])

12.3 Embedding the Naturals into the Hyperreals

constdefs
hypreal-of-nat :: *nat* => *hypreal*
hypreal-of-nat m == *of-nat m*

lemma *SNat-eq*: *Nats* = {*n*. ∃ *N*. *n* = *hypreal-of-nat N*}
by (*force simp add*: *hypreal-of-nat-def Nats-def*)

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

lemma *hypreal-of-nat-mult*: *hypreal-of-nat* $(m * n) = hypreal\text{-}of\text{-}nat\ m * hypreal\text{-}of\text{-}nat\ n$
by (*simp add*: *hypreal-of-nat-def*)
declare *hypreal-of-nat-mult* [*simp*]

lemma *hypreal-of-nat-less-iff*:
$(n < m) = (hypreal\text{-}of\text{-}nat\ n < hypreal\text{-}of\text{-}nat\ m)$
apply (*simp add*: *hypreal-of-nat-def*)
done
declare *hypreal-of-nat-less-iff* [*symmetric*, *simp*]

lemma *hypreal-of-nat-eq*:
$hypreal\text{-}of\text{-}nat\ (n::nat) = hypreal\text{-}of\text{-}real\ (real\ n)$
apply (*induct n*)
apply (*simp-all add*: *hypreal-of-nat-def real-of-nat-def*)
done

lemma *hypreal-of-nat*:
$hypreal\text{-}of\text{-}nat\ m = star\text{-}n\ (\%n.\ real\ m)$
apply (*fold star-of-def*)
apply (*induct m*)
apply (*simp-all add*: *hypreal-of-nat-def real-of-nat-def star-n-add*)
done

lemma *hypreal-of-nat-Suc*:
$hypreal\text{-}of\text{-}nat\ (Suc\ n) = hypreal\text{-}of\text{-}nat\ n + (1::hypreal)$
by (*simp add*: *hypreal-of-nat-def*)

lemma *hypreal-of-nat-number-of* [*simp*]:
$hypreal\text{-}of\text{-}nat\ (number\text{-}of\ v :: nat) =$
$(if\ neg\ (number\text{-}of\ v :: int)\ then\ 0$
$else\ (number\text{-}of\ v :: hypreal))$
by (*simp add*: *hypreal-of-nat-eq*)

lemma *hypreal-of-nat-zero* [*simp*]: $hypreal\text{-}of\text{-}nat\ 0 = 0$
by (*simp add*: *hypreal-of-nat-def*)

lemma *hypreal-of-nat-one* [*simp*]: $hypreal\text{-}of\text{-}nat\ 1 = 1$
by (*simp add*: *hypreal-of-nat-def*)

lemma *hypreal-of-nat-le-iff* [*simp*]:
$(hypreal\text{-}of\text{-}nat\ n \leq hypreal\text{-}of\text{-}nat\ m) = (n \leq m)$
by (*simp add*: *hypreal-of-nat-def*)

lemma *hypreal-of-nat-ge-zero* [*simp*]: $0 \leq$ *hypreal-of-nat n*
by (*simp add*: *hypreal-of-nat-def*)

ML
⟪
val hypreal-of-nat-def = thmhypreal-of-nat-def;

val hrabs-add-less = thm hrabs-add-less;
val hrabs-disj = thm hrabs-disj;
val hrabs-add-lemma-disj = thm hrabs-add-lemma-disj;
val hypreal-of-real-hrabs = thm hypreal-of-real-hrabs;
val hypreal-of-nat-add = thm hypreal-of-nat-add;
val hypreal-of-nat-mult = thm hypreal-of-nat-mult;
val hypreal-of-nat-less-iff = thm hypreal-of-nat-less-iff;
val hypreal-of-nat-Suc = thm hypreal-of-nat-Suc;
val hypreal-of-nat-number-of = thm hypreal-of-nat-number-of;
val hypreal-of-nat-zero = thm hypreal-of-nat-zero;
val hypreal-of-nat-one = thm hypreal-of-nat-one;
val hypreal-of-nat-le-iff = thmhypreal-of-nat-le-iff;
val hypreal-of-nat-ge-zero = thmhypreal-of-nat-ge-zero;
val hypreal-of-nat = thmhypreal-of-nat;
⟫

end

13 NSA: Infinite Numbers, Infinitesimals, Infinitely Close Relation

theory *NSA*
imports *HyperArith ../Real/RComplete*
begin

constdefs

Infinitesimal :: *hypreal set*
Infinitesimal == $\{x.\ \forall r \in Reals.\ 0 < r \longrightarrow abs\ x < r\}$

HFinite :: *hypreal set*
HFinite == $\{x.\ \exists r \in Reals.\ abs\ x < r\}$

HInfinite :: *hypreal set*
HInfinite == $\{x.\ \forall r \in Reals.\ r < abs\ x\}$

approx :: [*hypreal*, *hypreal*] => *bool* (**infixl** @= *50*)
— the 'infinitely close' relation
x @= y == $(x + -y) \in$ *Infinitesimal*

st :: *hypreal* => *hypreal*
— the standard part of a hyperreal
st == $(\%x.$ @$r.$ $x \in$ *HFinite* & $r \in$ *Reals* & r @= $x)$

monad :: *hypreal* => *hypreal set*
monad x == $\{y.$ x @= $y\}$

galaxy :: *hypreal* => *hypreal set*
galaxy x == $\{y.$ $(x + -y) \in$ *HFinite*$\}$

defs (**overloaded**)

SReal-def: *Reals* == $\{x.$ $\exists r.$ $x =$ *hypreal-of-real* $r\}$
— the standard real numbers as a subset of the hyperreals

syntax (*xsymbols*)
approx :: [*hypreal*, *hypreal*] => *bool* (**infixl** $\approx$ *50*)

syntax (*HTML* **output**)
approx :: [*hypreal*, *hypreal*] => *bool* (**infixl** $\approx$ *50*)

13.1 Closure Laws for the Standard Reals

lemma *SReal-add* [*simp*]:
[| (x::*hypreal*) $\in$ *Reals*; $y \in$ *Reals* |] ==> $x + y \in$ *Reals*
apply (*auto simp add*: *SReal-def*)
apply (*rule-tac* $x = r + ra$ **in** *exI*, *simp*)
done

lemma *SReal-mult*: [| (x::*hypreal*) $\in$ *Reals*; $y \in$ *Reals* |] ==> $x * y \in$ *Reals*
apply (*simp add*: *SReal-def*, *safe*)
apply (*rule-tac* $x = r * ra$ **in** *exI*)
apply (*simp* (*no-asm*))
done

lemma *SReal-inverse*: (x::*hypreal*) $\in$ *Reals* ==> *inverse* $x \in$ *Reals*
apply (*simp add*: *SReal-def*)
apply (*blast intro*: *star-of-inverse* [*symmetric*])
done

lemma *SReal-divide*: [| (x::*hypreal*) $\in$ *Reals*; $y \in$ *Reals* |] ==> $x/y \in$ *Reals*
by (*simp* (*no-asm-simp*) *add*: *SReal-mult SReal-inverse divide-inverse*)

lemma *SReal-minus*: (x::*hypreal*) $\in$ *Reals* ==> $-x \in$ *Reals*

apply (*simp add*: *SReal-def*)
apply (*blast intro*: *star-of-minus* [*symmetric*])
done

lemma *SReal-minus-iff* [*simp*]: $(-x \in Reals) = ((x::hypreal) \in Reals)$
apply *auto*
apply (*erule-tac* [*2*] *SReal-minus*)
apply (*drule SReal-minus*, *auto*)
done

lemma *SReal-add-cancel*:
$[\!| (x::hypreal) + y \in Reals; y \in Reals |\!] ==> x \in Reals$
apply (*drule-tac* $x = y$ **in** *SReal-minus*)
apply (*drule SReal-add*, *assumption*, *auto*)
done

lemma *SReal-hrabs*: $(x::hypreal) \in Reals ==> abs\ x \in Reals$
apply (*auto simp add*: *SReal-def*)
apply (*rule-tac x=abs r* **in** *exI*)
apply *simp*
done

lemma *SReal-hypreal-of-real* [*simp*]: *hypreal-of-real x* $\in$ *Reals*
by (*simp add*: *SReal-def*)

lemma *SReal-number-of* [*simp*]: (*number-of w* ::*hypreal*) $\in$ *Reals*
apply (*simp only*: *star-of-number-of* [*symmetric*])
apply (*rule SReal-hypreal-of-real*)
done

lemma *Reals-0* [*simp*]: $(0::hypreal) \in Reals$
apply (*subst numeral-0-eq-0* [*symmetric*])
apply (*rule SReal-number-of*)
done

lemma *Reals-1* [*simp*]: $(1::hypreal) \in Reals$
apply (*subst numeral-1-eq-1* [*symmetric*])
apply (*rule SReal-number-of*)
done

lemma *SReal-divide-number-of*: $r \in Reals ==> r/(number\text{-}of\ w::hypreal) \in Reals$
apply (*simp only*: *divide-inverse*)
apply (*blast intro*!: *SReal-number-of SReal-mult SReal-inverse*)
done

epsilon is not in Reals because it is an infinitesimal

lemma *SReal-epsilon-not-mem*: *epsilon* $\notin$ *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* $\notin$ *Reals*
apply (*simp add*: *SReal-def*)
apply (*auto simp add*: *hypreal-of-real-not-eq-omega* [*THEN not-sym*])
done

lemma *SReal-UNIV-real*: $\{x.\ \mathit{hypreal\text{-}of\text{-}real}\ x \in \mathit{Reals}\} = (\mathit{UNIV}::\mathit{real\ set})$
by (*simp add*: *SReal-def*)

lemma *SReal-iff*: $(x \in \mathit{Reals}) = (\exists y.\ x = \mathit{hypreal\text{-}of\text{-}real}\ y)$
by (*simp add*: *SReal-def*)

lemma *hypreal-of-real-image*: *hypreal-of-real* ‘$(\mathit{UNIV}::\mathit{real\ set}) = \mathit{Reals}$
by (*auto simp add*: *SReal-def*)

lemma *inv-hypreal-of-real-image*: *inv hypreal-of-real* ‘ *Reals* = *UNIV*
apply (*auto simp add*: *SReal-def*)
apply (*rule inj-hypreal-of-real* [*THEN inv-f-f*, *THEN subst*], *blast*)
done

lemma *SReal-hypreal-of-real-image*:
$[\!|\ \exists x.\ x\text{: } P;\ P \subseteq \mathit{Reals}\ |\!] \Longrightarrow \exists Q.\ P = \mathit{hypreal\text{-}of\text{-}real}\ \text{‘}\ Q$
apply (*simp add*: *SReal-def*, *blast*)
done

lemma *SReal-dense*:
$[\!|\ (x::\mathit{hypreal}) \in \mathit{Reals};\ y \in \mathit{Reals};\ x<y\ |\!] \Longrightarrow \exists r \in \mathit{Reals}.\ x<r\ \&\ r<y$
apply (*auto simp add*: *SReal-iff*)
apply (*drule dense*, *safe*)
apply (*rule-tac* $x = \mathit{hypreal\text{-}of\text{-}real}\ r$ **in** *bexI*, *auto*)
done

Completeness of Reals, but both lemmas are unused.

lemma *SReal-sup-lemma*:
$P \subseteq \mathit{Reals} \Longrightarrow ((\exists x \in P.\ y < x) =$
$(\exists X.\ \mathit{hypreal\text{-}of\text{-}real}\ X \in P\ \&\ y < \mathit{hypreal\text{-}of\text{-}real}\ X))$
by (*blast dest*!: *SReal-iff* [*THEN iffD1*])

lemma *SReal-sup-lemma2*:
$[\!|\ P \subseteq \mathit{Reals};\ \exists x.\ x \in P;\ \exists y \in \mathit{Reals}.\ \forall x \in P.\ x < y\ |\!]$
$\Longrightarrow (\exists X.\ X \in \{w.\ \mathit{hypreal\text{-}of\text{-}real}\ w \in P\})\ \&$
$(\exists Y.\ \forall X \in \{w.\ \mathit{hypreal\text{-}of\text{-}real}\ w \in P\}.\ X < Y)$
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

13.2 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

13.3 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*!: *SReal-add hrabs-add-less*)
done

lemma *HFinite-mult*: $[|x \in HFinite; y \in HFinite|] ==> x*y \in HFinite$
apply (*simp add*: *HFinite-def abs-mult*)
apply (*blast intro*!: *SReal-mult abs-mult-less*)
done

lemma *HFinite-minus-iff*: $(-x \in HFinite) = (x \in HFinite)$
by (*simp add*: *HFinite-def*)

lemma *SReal-subset-HFinite*: $Reals \subseteq HFinite$
apply (*auto simp add*: *SReal-def HFinite-def*)
apply (*rule-tac* $x = 1 + abs$ (*hypreal-of-real* r) **in** *exI*)
apply (*rule conjI*, *rule-tac* $x = 1 + abs\ r$ **in** *exI*)
apply *simp-all*
done

lemma *HFinite-hypreal-of-real* [*simp*]: *hypreal-of-real* $x \in HFinite$
by (*auto intro*: *SReal-subset-HFinite* [*THEN subsetD*])

lemma *HFiniteD*: $x \in HFinite ==> \exists t \in Reals.\ abs\ x < t$
by (*simp add*: *HFinite-def*)

lemma *HFinite-hrabs-iff* [*iff*]: $(abs\ x \in HFinite) = (x \in HFinite)$
by (*simp add*: *HFinite-def*)

lemma *HFinite-number-of* [*simp*]: *number-of* $w \in HFinite$
by (*rule SReal-number-of* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])

lemma *HFinite-0* [*simp*]: $0 \in HFinite$
apply (*subst numeral-0-eq-0* [*symmetric*])
apply (*rule HFinite-number-of*)
done

lemma *HFinite-1* [*simp*]: $1 \in HFinite$

apply (*subst numeral-1-eq-1* [*symmetric*])
apply (*rule HFinite-number-of*)
done

lemma *HFinite-bounded*: [|$x \in$ *HFinite*; $y \leq x$; $0 \leq y$ |] ==> $y \in$ *HFinite*
apply (*case-tac* $x \leq 0$)
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

13.4 Set of Infinitesimals is a Subring of the Hyperreals

lemma *InfinitesimalD*:
$x \in$ *Infinitesimal* ==> $\forall r \in$ *Reals*. $0 < r$ --> *abs* $x < r$
by (*simp add*: *Infinitesimal-def*)

lemma *Infinitesimal-zero* [*iff*]: $0 \in$ *Infinitesimal*
by (*simp add*: *Infinitesimal-def*)

lemma *hypreal-sum-of-halves*: $x/(2::hypreal) + x/(2::hypreal) = x$
by *auto*

lemma *Infinitesimal-add*:
[| $x \in$ *Infinitesimal*; $y \in$ *Infinitesimal* |] ==> $(x+y) \in$ *Infinitesimal*
apply (*auto simp add*: *Infinitesimal-def*)
apply (*rule hypreal-sum-of-halves* [*THEN subst*])
apply (*drule half-gt-zero*)
apply (*blast intro*: *hrabs-add-less SReal-divide-number-of*)
done

lemma *Infinitesimal-minus-iff* [*simp*]: $(-x:Infinitesimal) = (x:Infinitesimal)$
by (*simp add*: *Infinitesimal-def*)

lemma *Infinitesimal-diff*:
[| $x \in$ *Infinitesimal*; $y \in$ *Infinitesimal* |] ==> $x-y \in$ *Infinitesimal*
by (*simp add*: *diff-def Infinitesimal-add*)

lemma *Infinitesimal-mult*:
[| $x \in$ *Infinitesimal*; $y \in$ *Infinitesimal* |] ==> $(x * y) \in$ *Infinitesimal*
apply (*auto simp add*: *Infinitesimal-def abs-mult*)
apply (*case-tac* $y=0$, *simp*)
apply (*cut-tac* $a = abs\ x$ **and** $b = 1$ **and** $c = abs\ y$ **and** $d = r$
in *mult-strict-mono*, *auto*)
done

lemma *Infinitesimal-HFinite-mult*:
[| $x \in$ *Infinitesimal*; $y \in$ *HFinite* |] ==> $(x * y) \in$ *Infinitesimal*

apply (*auto dest*!: *HFiniteD simp add*: *Infinitesimal-def abs-mult*)
apply (*frule hrabs-less-gt-zero*)
apply (*drule-tac x* = *r*/*t* **in** *bspec*)
apply (*blast intro*: *SReal-divide*)
apply (*cut-tac a* = *abs x* **and** *b* = *r*/*t* **and** *c* = *abs y* **in** *mult-strict-mono*)
apply (*auto simp add*: *zero-less-divide-iff*)
done

lemma *Infinitesimal-HFinite-mult2*:
[| *x* ∈ *Infinitesimal*; *y* ∈ *HFinite* |] ==> (*y* ∗ *x*) ∈ *Infinitesimal*
by (*auto dest*: *Infinitesimal-HFinite-mult simp add*: *mult-commute*)

lemma *HInfinite-inverse-Infinitesimal*:
x ∈ *HInfinite* ==> *inverse x*: *Infinitesimal*
apply (*auto simp add*: *HInfinite-def Infinitesimal-def*)
apply (*erule-tac x* = *inverse r* **in** *ballE*)
apply (*frule-tac a1* = *r* **and** *z* = *abs x* **in** *positive-imp-inverse-positive* [*THEN order-less-trans*], *assumption*)
apply (*drule inverse-inverse-eq* [*symmetric*, *THEN subst*])
apply (*rule inverse-less-iff-less* [*THEN iffD1*])
apply (*auto simp add*: *SReal-inverse*)
done

lemma *HInfinite-mult*: [|*x* ∈ *HInfinite*;*y* ∈ *HInfinite*|] ==> (*x*∗*y*) ∈ *HInfinite*
apply (*auto simp add*: *HInfinite-def abs-mult*)
apply (*erule-tac x* = *1* **in** *ballE*)
apply (*erule-tac x* = *r* **in** *ballE*)
apply (*case-tac y*=*0*, *simp*)
apply (*cut-tac c* = *1* **and** *d* = *abs x* **and** *a* = *r* **and** *b* = *abs y* **in** *mult-strict-mono*)
apply (*auto simp add*: *mult-ac*)
done

lemma *hypreal-add-zero-less-le-mono*: [|*r* < *x*; (*0*::*hypreal*) ≤ *y*|] ==> *r* < *x*+*y*
by (*auto dest*: *add-less-le-mono*)

lemma *HInfinite-add-ge-zero*:
[|*x* ∈ *HInfinite*; *0* ≤ *y*; *0* ≤ *x*|] ==> (*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* ∈ *HInfinite*; *0* ≤ *y*; *0* ≤ *x*|] ==> (*y* + *x*): *HInfinite*
by (*auto intro*!: *HInfinite-add-ge-zero simp add*: *add-commute*)

lemma *HInfinite-add-gt-zero*:
[|*x* ∈ *HInfinite*; *0* < *y*; *0* < *x*|] ==> (*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*:
$[|x \in HInfinite; y \leq 0; x \leq 0|] ==> (x + y)$: *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*:
$[|x \in HInfinite; y < 0; x < 0|] ==> (x + y)$: *HInfinite*
by (*blast intro*: *HInfinite-add-le-zero order-less-imp-le*)

lemma *HFinite-sum-squares*:
$[|a$: *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 *Infinitesimal-hrabs-iff* [*iff*]:
$(abs\ x \in Infinitesimal) = (x \in Infinitesimal)$
by (*auto simp add*: *abs-if*)

lemma *HFinite-diff-Infinitesimal-hrabs*:
$x \in HFinite - Infinitesimal ==> abs\ x \in HFinite - Infinitesimal$
by *blast*

lemma *hrabs-less-Infinitesimal*:
$[|\ e \in Infinitesimal; abs\ x < e\ |] ==> x \in Infinitesimal$
by (*auto simp add*: *Infinitesimal-def abs-less-iff*)

lemma *hrabs-le-Infinitesimal*:
$[|\ e \in Infinitesimal; abs\ x \leq e\ |] ==> x \in Infinitesimal$
by (*blast dest*: *order-le-imp-less-or-eq intro*: *hrabs-less-Infinitesimal*)

lemma *Infinitesimal-interval*:
$[|\ e \in Infinitesimal; e' \in Infinitesimal; e' < x\ ;\ x < e\ |]$
$==> x \in Infinitesimal$
by (*auto simp add*: *Infinitesimal-def abs-less-iff*)

lemma *Infinitesimal-interval2*:
$[|\ e \in Infinitesimal; e' \in Infinitesimal;$
$e' \leq x\ ;\ x \leq e\ |] ==> x \in Infinitesimal$

by (*auto intro*: *Infinitesimal-interval simp add*: *order-le-less*)

lemma *not-Infinitesimal-mult*:
[| $x \notin$ *Infinitesimal*; $y \notin$ *Infinitesimal*|] ==> $(x*y) \notin$*Infinitesimal*
apply (*unfold Infinitesimal-def*, *clarify*)
apply (*simp add*: *linorder-not-less abs-mult*)
apply (*erule-tac* $x = r*ra$ **in** *ballE*)
prefer *2* **apply** (*fast intro*: *SReal-mult*)
apply (*auto simp add*: *zero-less-mult-iff*)
apply (*cut-tac* $c = ra$ **and** $d = abs\ y$ **and** $a = r$ **and** $b = abs\ x$ **in** *mult-mono*, *auto*)
done

lemma *Infinitesimal-mult-disj*:
$x*y \in$ *Infinitesimal* ==> $x \in$ *Infinitesimal* | $y \in$ *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* ==> $x \neq 0$
by *blast*

lemma *HFinite-Infinitesimal-diff-mult*:
[| $x \in$ *HFinite* − *Infinitesimal*;
$y \in$ *HFinite* − *Infinitesimal*
|] ==> $(x*y) \in$ *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-hypreal-of-real-mult*:
$x \in$ *Infinitesimal* ==> $x *$ *hypreal-of-real* $r \in$ *Infinitesimal*
by (*erule HFinite-hypreal-of-real* [*THEN* [*2*] *Infinitesimal-HFinite-mult*])

lemma *Infinitesimal-hypreal-of-real-mult2*:
$x \in$ *Infinitesimal* ==> *hypreal-of-real* $r * x \in$ *Infinitesimal*
by (*erule HFinite-hypreal-of-real* [*THEN* [*2*] *Infinitesimal-HFinite-mult2*])

13.5 The Infinitely Close Relation

lemma *mem-infmal-iff*: $(x \in$ *Infinitesimal*$) = (x$ @= *0*$)$
by (*simp add*: *Infinitesimal-def approx-def*)

lemma *approx-minus-iff*: $(x \mathbin{@=} y) = (x + -y \mathbin{@=} 0)$
by (*simp add*: *approx-def*)

lemma *approx-minus-iff2*: $(x \mathbin{@=} y) = (-y + x \mathbin{@=} 0)$
by (*simp add*: *approx-def add-commute*)

lemma *approx-refl* [*iff*]: $x \mathbin{@=} x$
by (*simp add*: *approx-def Infinitesimal-def*)

lemma *hypreal-minus-distrib1*: $-(y + -(x::hypreal)) = x + -y$
by (*simp add*: *add-commute*)

lemma *approx-sym*: $x \mathbin{@=} y \Longrightarrow y \mathbin{@=} x$
apply (*simp add*: *approx-def*)
apply (*rule hypreal-minus-distrib1* [*THEN subst*])
apply (*erule Infinitesimal-minus-iff* [*THEN iffD2*])
done

lemma *approx-trans*: $[\!| x \mathbin{@=} y; y \mathbin{@=} z |\!] \Longrightarrow x \mathbin{@=} z$
apply (*simp add*: *approx-def*)
apply (*drule Infinitesimal-add*, *assumption*, *auto*)
done

lemma *approx-trans2*: $[\!| r \mathbin{@=} x; s \mathbin{@=} x |\!] \Longrightarrow r \mathbin{@=} s$
by (*blast intro*: *approx-sym approx-trans*)

lemma *approx-trans3*: $[\!| x \mathbin{@=} r; x \mathbin{@=} s|\!] \Longrightarrow r \mathbin{@=} s$
by (*blast intro*: *approx-sym approx-trans*)

lemma *number-of-approx-reorient*: $(number\text{-}of\ w \mathbin{@=} x) = (x \mathbin{@=} number\text{-}of\ w)$
by (*blast intro*: *approx-sym*)

lemma *zero-approx-reorient*: $(0 \mathbin{@=} x) = (x \mathbin{@=} 0)$
by (*blast intro*: *approx-sym*)

lemma *one-approx-reorient*: $(1 \mathbin{@=} x) = (x \mathbin{@=} 1)$
by (*blast intro*: *approx-sym*)

ML
⟪
val SReal-add = thm SReal-add;
val SReal-mult = thm SReal-mult;
val SReal-inverse = thm SReal-inverse;
val SReal-divide = thm SReal-divide;
val SReal-minus = thm SReal-minus;
val SReal-minus-iff = thm SReal-minus-iff;
val SReal-add-cancel = thm SReal-add-cancel;

val SReal-hrabs = thm SReal-hrabs;
val SReal-hypreal-of-real = thm SReal-hypreal-of-real;
val SReal-number-of = thm SReal-number-of;
val Reals-0 = thm Reals-0;
val Reals-1 = thm Reals-1;
val SReal-divide-number-of = thm SReal-divide-number-of;
val SReal-epsilon-not-mem = thm SReal-epsilon-not-mem;
val SReal-omega-not-mem = thm SReal-omega-not-mem;
val SReal-UNIV-real = thm SReal-UNIV-real;
val SReal-iff = thm SReal-iff;
val hypreal-of-real-image = thm hypreal-of-real-image;
val inv-hypreal-of-real-image = thm inv-hypreal-of-real-image;
val SReal-hypreal-of-real-image = thm SReal-hypreal-of-real-image;
val SReal-dense = thm SReal-dense;
val hypreal-of-real-isUb-iff = thm hypreal-of-real-isUb-iff;
val hypreal-of-real-isLub1 = thm hypreal-of-real-isLub1;
val hypreal-of-real-isLub2 = thm hypreal-of-real-isLub2;
val hypreal-of-real-isLub-iff = thm hypreal-of-real-isLub-iff;
val lemma-isUb-hypreal-of-real = thm lemma-isUb-hypreal-of-real;
val lemma-isLub-hypreal-of-real = thm lemma-isLub-hypreal-of-real;
val lemma-isLub-hypreal-of-real2 = thm lemma-isLub-hypreal-of-real2;
val SReal-complete = thm SReal-complete;
val HFinite-add = thm HFinite-add;
val HFinite-mult = thm HFinite-mult;
val HFinite-minus-iff = thm HFinite-minus-iff;
val SReal-subset-HFinite = thm SReal-subset-HFinite;
val HFinite-hypreal-of-real = thm HFinite-hypreal-of-real;
val HFiniteD = thm HFiniteD;
val HFinite-hrabs-iff = thm HFinite-hrabs-iff;
val HFinite-number-of = thm HFinite-number-of;
val HFinite-0 = thm HFinite-0;
val HFinite-1 = thm HFinite-1;
val HFinite-bounded = thm HFinite-bounded;
val InfinitesimalD = thm InfinitesimalD;
val Infinitesimal-zero = thm Infinitesimal-zero;
val hypreal-sum-of-halves = thm hypreal-sum-of-halves;
val Infinitesimal-add = thm Infinitesimal-add;
val Infinitesimal-minus-iff = thm Infinitesimal-minus-iff;
val Infinitesimal-diff = thm Infinitesimal-diff;
val Infinitesimal-mult = thm Infinitesimal-mult;
val Infinitesimal-HFinite-mult = thm Infinitesimal-HFinite-mult;
val Infinitesimal-HFinite-mult2 = thm Infinitesimal-HFinite-mult2;
val HInfinite-inverse-Infinitesimal = thm HInfinite-inverse-Infinitesimal;
val HInfinite-mult = thm HInfinite-mult;
val HInfinite-add-ge-zero = thm HInfinite-add-ge-zero;
val HInfinite-add-ge-zero2 = thm HInfinite-add-ge-zero2;
val HInfinite-add-gt-zero = thm HInfinite-add-gt-zero;
val HInfinite-minus-iff = thm HInfinite-minus-iff;
val HInfinite-add-le-zero = thm HInfinite-add-le-zero;

```
val HInfinite-add-lt-zero = thm HInfinite-add-lt-zero;
val HFinite-sum-squares = thm HFinite-sum-squares;
val not-Infinitesimal-not-zero = thm not-Infinitesimal-not-zero;
val not-Infinitesimal-not-zero2 = thm not-Infinitesimal-not-zero2;
val Infinitesimal-hrabs-iff = thm Infinitesimal-hrabs-iff;
val HFinite-diff-Infinitesimal-hrabs = thm HFinite-diff-Infinitesimal-hrabs;
val hrabs-less-Infinitesimal = thm hrabs-less-Infinitesimal;
val hrabs-le-Infinitesimal = thm hrabs-le-Infinitesimal;
val Infinitesimal-interval = thm Infinitesimal-interval;
val Infinitesimal-interval2 = thm Infinitesimal-interval2;
val not-Infinitesimal-mult = thm not-Infinitesimal-mult;
val Infinitesimal-mult-disj = thm Infinitesimal-mult-disj;
val HFinite-Infinitesimal-not-zero = thm HFinite-Infinitesimal-not-zero;
val HFinite-Infinitesimal-diff-mult = thm HFinite-Infinitesimal-diff-mult;
val Infinitesimal-subset-HFinite = thm Infinitesimal-subset-HFinite;
val Infinitesimal-hypreal-of-real-mult = thm Infinitesimal-hypreal-of-real-mult;
val Infinitesimal-hypreal-of-real-mult2 = thm Infinitesimal-hypreal-of-real-mult2;
val mem-infmal-iff = thm mem-infmal-iff;
val approx-minus-iff = thm approx-minus-iff;
val approx-minus-iff2 = thm approx-minus-iff2;
val approx-refl = thm approx-refl;
val approx-sym = thm approx-sym;
val approx-trans = thm approx-trans;
val approx-trans2 = thm approx-trans2;
val approx-trans3 = thm approx-trans3;
val number-of-approx-reorient = thm number-of-approx-reorient;
val zero-approx-reorient = thm zero-approx-reorient;
val one-approx-reorient = thm one-approx-reorient;

(*** 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 = zero-approx-reorient RS eq-reflection;
val meta-one-approx-reorient = one-approx-reorient RS eq-reflection;
val meta-number-of-approx-reorient = 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(0, -) => NONE
    | Const(1, -) => NONE
    | Const(Numeral.number-of, -) $ - => NONE
    | - => SOME (case t of
                Const(0, -) => meta-zero-approx-reorient
              | Const(1, -) => meta-one-approx-reorient
              | Const(Numeral.number-of, -) $ - =>
```

meta-number-of-approx-reorient);

val approx-reorient-simproc =
Bin-Simprocs.prep-simproc
(*reorient-simproc*, [*0@=x*, *1@=x*, *number-of w @= x*], *reorient-proc*);

Addsimprocs [*approx-reorient-simproc*];
⟫

lemma *Infinitesimal-approx-minus*: $(x-y \in Infinitesimal) = (x \mathbin{@=} y)$
by (*auto simp add*: *diff-def 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*:
$[\!| x \in Infinitesimal; y \in Infinitesimal |\!] \Longrightarrow x \mathbin{@=} y$
apply (*simp add*: *mem-infmal-iff*)
apply (*blast intro*: *approx-trans approx-sym*)
done

lemma *approx-add*: $[\!| a \mathbin{@=} b; c \mathbin{@=} d |\!] \Longrightarrow a+c \mathbin{@=} b+d$
proof (*unfold approx-def*)
assume *inf*: $a + -b \in Infinitesimal$ $c + -d \in Infinitesimal$
have $a + c + -(b + d) = (a + -b) + (c + -d)$ **by** *arith*
also have $... \in Infinitesimal$ **using** *inf* **by** (*rule Infinitesimal-add*)
finally show $a + c + -(b + d) \in Infinitesimal$ **.**
qed

lemma *approx-minus*: $a \mathbin{@=} b \Longrightarrow -a \mathbin{@=} -b$
apply (*rule approx-minus-iff* [*THEN iffD2*, *THEN approx-sym*])
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*simp* (*no-asm*) *add*: *add-commute*)
done

lemma *approx-minus2*: $-a \mathbin{@=} -b \Longrightarrow a \mathbin{@=} b$
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*: $[\!| a \mathbin{@=} b; c \mathbin{@=} d |\!] \Longrightarrow a + -c \mathbin{@=} b + -d$
by (*blast intro*!: *approx-add approx-minus*)

lemma *approx-mult1*: $[\!| a \mathbin{@=} b; c: HFinite |\!] \Longrightarrow a*c \mathbin{@=} b*c$
by (*simp add*: *approx-def Infinitesimal-HFinite-mult minus-mult-left*
left-distrib [*symmetric*]

del: *minus-mult-left* [*symmetric*])

lemma *approx-mult2*: [|*a* @= *b*; *c*: *HFinite*|] ==> *c*∗*a* @= *c*∗*b*
by (*simp add*: *approx-mult1 mult-commute*)

lemma *approx-mult-subst*: [|*u* @= *v*∗*x*; *x* @= *y*; *v* ∈ *HFinite*|] ==> *u* @= *v*∗*y*
by (*blast intro*: *approx-mult2 approx-trans*)

lemma *approx-mult-subst2*: [| *u* @= *x*∗*v*; *x* @= *y*; *v* ∈ *HFinite* |] ==> *u* @= *y*∗*v*
by (*blast intro*: *approx-mult1 approx-trans*)

lemma *approx-mult-subst-SReal*:
[| *u* @= *x*∗*hypreal-of-real v*; *x* @= *y* |] ==> *u* @= *y*∗*hypreal-of-real 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* ∈ *Infinitesimal* ==> *x* @= *y* + *x*
by (*auto dest*: *Infinitesimal-add-approx-self simp add*: *add-commute*)

lemma *Infinitesimal-add-minus-approx-self*: *y* ∈ *Infinitesimal* ==> *x* @= *x* + −*y*
by (*blast intro*!: *Infinitesimal-add-approx-self Infinitesimal-minus-iff* [*THEN iffD2*])

lemma *Infinitesimal-add-cancel*: [| *y* ∈ *Infinitesimal*; *x*+*y* @= *z*|] ==> *x* @= *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 @= $z + y$|] ==> x @= 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*: $d + b$ @= $d + c$ ==> b @= c
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*simp add*: *approx-minus-iff* [*symmetric*] *add-ac*)
done

lemma *approx-add-right-cancel*: $b + d$ @= $c + d$ ==> b @= c
apply (*rule approx-add-left-cancel*)
apply (*simp add*: *add-commute*)
done

lemma *approx-add-mono1*: b @= c ==> $d + b$ @= $d + c$
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*simp add*: *approx-minus-iff* [*symmetric*] *add-ac*)
done

lemma *approx-add-mono2*: b @= c ==> $b + a$ @= $c + a$
by (*simp add*: *add-commute approx-add-mono1*)

lemma *approx-add-left-iff* [*simp*]: $(a + b$ @= $a + c) = (b$ @= $c)$
by (*fast elim*: *approx-add-left-cancel approx-add-mono1*)

lemma *approx-add-right-iff* [*simp*]: $(b + a$ @= $c + a) = (b$ @= $c)$
by (*simp add*: *add-commute*)

lemma *approx-HFinite*: [| $x \in$ *HFinite*; x @= 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-hypreal-of-real-HFinite*: x @= *hypreal-of-real* D ==> $x \in$ *HFinite*
by (*rule approx-sym* [*THEN* [*2*] *approx-HFinite*], *auto*)

lemma *approx-mult-HFinite*:
[|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 *approx-mult-hypreal-of-real*:
[|*a* @= *hypreal-of-real b*; *c* @= *hypreal-of-real d* |]
==> *a*∗*c* @= *hypreal-of-real b*∗*hypreal-of-real d*
by (*blast intro*!: *approx-mult-HFinite approx-hypreal-of-real-HFinite*
HFinite-hypreal-of-real)

lemma *approx-SReal-mult-cancel-zero*:
[| *a* ∈ *Reals*; *a* ≠ *0*; *a*∗*x* @= *0* |] ==> *x* @= *0*
apply (*drule SReal-inverse* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])
apply (*auto dest*: *approx-mult2 simp add*: *mult-assoc* [*symmetric*])
done

lemma *approx-mult-SReal1*: [| *a* ∈ *Reals*; *x* @= *0* |] ==> *x*∗*a* @= *0*
by (*auto dest*: *SReal-subset-HFinite* [*THEN subsetD*] *approx-mult1*)

lemma *approx-mult-SReal2*: [| *a* ∈ *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* ∈ *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* ∈ *Reals*; *a* ≠ *0*; *a*∗ *w* @= *a*∗*z* |] ==> *w* @= *z*
apply (*drule SReal-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* ∈ *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* ≤ *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

13.6 Zero is the Only Infinitesimal that is also a Real

lemma *Infinitesimal-less-SReal*:
[| *x* ∈ *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*:
    $y \in Infinitesimal \Longrightarrow \forall r \in Reals.\ 0 < r \longrightarrow y < r$
by (*blast intro*: *Infinitesimal-less-SReal*)

lemma *SReal-not-Infinitesimal*:
    $[|\ 0 < y;\ \ y \in Reals|] \Longrightarrow y \notin Infinitesimal$
apply (*simp add*: *Infinitesimal-def*)
apply (*auto simp add*: *abs-if*)
done

lemma *SReal-minus-not-Infinitesimal*:
    $[|\ y < 0;\ \ y \in Reals\ |] \Longrightarrow y \notin Infinitesimal$
apply (*subst Infinitesimal-minus-iff* [*symmetric*])
apply (*rule SReal-not-Infinitesimal*, *auto*)
done

lemma *SReal-Int-Infinitesimal-zero*: $Reals\ Int\ Infinitesimal = \{0\}$
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*: $[|\ x \in Reals; x \in Infinitesimal|] \Longrightarrow x = 0$
by (*cut-tac SReal-Int-Infinitesimal-zero*, *blast*)

lemma *SReal-HFinite-diff-Infinitesimal*:
    $[|\ x \in Reals; x \neq 0\ |] \Longrightarrow x \in HFinite - Infinitesimal$
by (*auto dest*: *SReal-Infinitesimal-zero SReal-subset-HFinite* [*THEN subsetD*])

lemma *hypreal-of-real-HFinite-diff-Infinitesimal*:
    $hypreal\text{-}of\text{-}real\ x \neq 0 \Longrightarrow hypreal\text{-}of\text{-}real\ x \in HFinite - Infinitesimal$
by (*rule SReal-HFinite-diff-Infinitesimal*, *auto*)

lemma *hypreal-of-real-Infinitesimal-iff-0* [*iff*]:
    $(hypreal\text{-}of\text{-}real\ x \in Infinitesimal) = (x=0)$
apply *auto*
apply (*rule ccontr*)
apply (*rule hypreal-of-real-HFinite-diff-Infinitesimal* [*THEN DiffD2*], *auto*)
done

lemma *number-of-not-Infinitesimal* [*simp*]:
    $number\text{-}of\ w \neq (0::hypreal) \Longrightarrow number\text{-}of\ w \notin Infinitesimal$
by (*fast dest*: *SReal-number-of* [*THEN SReal-Infinitesimal-zero*])

lemma *one-not-Infinitesimal* [*simp*]: *1 ∉ Infinitesimal*
apply (*subst numeral-1-eq-1* [*symmetric*])
apply (*rule number-of-not-Infinitesimal*)
apply (*simp* (*no-asm*))
done

lemma *approx-SReal-not-zero*: [| *y ∈ 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*:
[| *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 ∈ Infinitesimal*; *y ∈ Reals* |] ==> *x/y ∈ Infinitesimal*
apply (*simp add*: *divide-inverse*)
apply (*auto intro*!: *Infinitesimal-HFinite-mult*
dest!: *SReal-inverse* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])
done

13.7 Uniqueness: Two Infinitely Close Reals are Equal

lemma *SReal-approx-iff*: [|*x ∈ Reals*; *y ∈ Reals*|] ==> (*x @= y*) = (*x = y*)
apply *auto*
apply (*simp add*: *approx-def*)
apply (*drule-tac x = y* **in** *SReal-minus*)
apply (*drule SReal-add*, *assumption*)
apply (*drule SReal-Infinitesimal-zero*, *assumption*)
apply (*drule sym*)
apply (*simp add*: *hypreal-eq-minus-iff* [*symmetric*])
done

lemma *number-of-approx-iff* [*simp*]:
(*number-of v @= number-of w*) = (*number-of v* = (*number-of w* :: *hypreal*))

by (*auto simp add*: *SReal-approx-iff*)

lemma [*simp*]: (*0* @= *number-of w*) = ((*number-of w* :: *hypreal*) = *0*)
(*number-of w* @= *0*) = ((*number-of w* :: *hypreal*) = *0*)
(*1* @= *number-of w*) = ((*number-of w* :: *hypreal*) = *1*)
(*number-of w* @= *1*) = ((*number-of w* :: *hypreal*) = *1*)
~ (*0* @= *1*) ~ (*1* @= *0*)
by (*auto simp only*: *SReal-number-of SReal-approx-iff Reals-0 Reals-1*)

lemma *hypreal-of-real-approx-iff* [*simp*]:
(*hypreal-of-real k* @= *hypreal-of-real m*) = (*k* = *m*)
apply *auto*
apply (*rule inj-hypreal-of-real* [*THEN injD*])
apply (*rule SReal-approx-iff* [*THEN iffD1*], *auto*)
done

lemma *hypreal-of-real-approx-number-of-iff* [*simp*]:
(*hypreal-of-real k* @= *number-of w*) = (*k* = *number-of w*)
by (*subst hypreal-of-real-approx-iff* [*symmetric*], *auto*)

lemma [*simp*]: (*hypreal-of-real k* @= *0*) = (*k* = *0*)
(*hypreal-of-real k* @= *1*) = (*k* = *1*)
by (*simp-all add*: *hypreal-of-real-approx-iff* [*symmetric*])

lemma *approx-unique-real*:
[| *r* ∈ *Reals*; *s* ∈ *Reals*; *r* @= *x*; *s* @= *x*|] ==> *r* = *s*
by (*blast intro*: *SReal-approx-iff* [*THEN iffD1*] *approx-trans2*)

13.8 Existence of Unique Real Infinitely Close

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 ∈ *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* ∈ *HFinite* ==> ∃ *y*. *y* ∈ {*s*. *s* ∈ *Reals* & *s* < *x*}

apply (*drule HFiniteD*, *safe*)
apply (*drule SReal-minus*)
apply (*rule-tac* $x = -t$ **in** *exI*)
apply (*auto simp add*: *abs-less-iff*)
done

lemma *lemma-st-part-subset*: $\{s.\ s \in Reals\ \&\ s < x\} \subseteq Reals$
by *auto*

lemma *lemma-st-part-lub*:
$x \in HFinite \Longrightarrow \exists t.\ isLub\ Reals\ \{s.\ s \in Reals\ \&\ s < x\}\ t$
by (*blast intro*!: *SReal-complete lemma-st-part-ub lemma-st-part-nonempty lemma-st-part-subset*)

lemma *lemma-hypreal-le-left-cancel*: $((t::hypreal) + r \leq t) = (r \leq 0)$
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*:
$[|\ x \in HFinite;\ isLub\ Reals\ \{s.\ s \in Reals\ \&\ s < x\}\ t;$
$r \in Reals;\ 0 < r\ |] \Longrightarrow x \leq t + r$
apply (*frule isLubD1a*)
apply (*rule ccontr*, *drule linorder-not-le* [*THEN iffD2*])
apply (*drule-tac* $x = t$ **in** *SReal-add*, *assumption*)
apply (*drule-tac* $y = t + r$ **in** *isLubD1* [*THEN setleD*], *auto*)
done

lemma *hypreal-setle-less-trans*:
$!!x::hypreal.\ [|\ S *<= x;\ x < y\ |] \Longrightarrow S *<= y$
apply (*simp add*: *setle-def*)
apply (*auto dest*!: *bspec order-le-less-trans intro*: *order-less-imp-le*)
done

lemma *hypreal-gt-isUb*:
$!!x::hypreal.\ [|\ isUb\ R\ S\ x;\ x < y;\ y \in R\ |] \Longrightarrow isUb\ R\ S\ y$
apply (*simp add*: *isUb-def*)
apply (*blast intro*: *hypreal-setle-less-trans*)
done

lemma *lemma-st-part-gt-ub*:
$[|\ x \in HFinite;\ x < y;\ y \in Reals\ |]$
$\Longrightarrow isUb\ Reals\ \{s.\ s \in Reals\ \&\ s < x\}\ y$
by (*auto dest*: *order-less-trans intro*: *order-less-imp-le intro*!: *isUbI setleI*)

lemma *lemma-minus-le-zero*: $t \leq t + -r \Longrightarrow r \leq (0::hypreal)$
apply (*drule-tac* $c = -t$ **in** *add-left-mono*)
apply (*auto simp add*: *add-assoc* [*symmetric*])

done

lemma *lemma-st-part-le2*:
[| $x \in$ *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 SReal-minus*, *drule-tac* $x = t$ **in** *SReal-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*:
[| $x \in$ *HFinite*;
isLub Reals $\{s.\ s \in Reals\ \&\ s < x\}\ t$;
$r \in$ *Reals*; $0 < r$ |]
==> $x + -t \leq r$
apply (*subgoal-tac* $x \leq t+r$)
apply (*auto intro*: *lemma-st-part-le1*)
done

lemma *lemma-st-part2a*:
[| $x \in$ *HFinite*;
isLub Reals $\{s.\ s \in Reals\ \&\ s < x\}\ t$;
$r \in$ *Reals*; $0 < r$ |]
==> $-(x + -t) \leq r$
apply (*subgoal-tac* $(t + -r \leq x)$)
apply (*auto intro*: *lemma-st-part-le2*)
done

lemma *lemma-SReal-ub*:
(x::*hypreal*) $\in$ *Reals* ==> *isUb Reals* $\{s.\ s \in Reals\ \&\ s < x\}\ x$
by (*auto intro*: *isUbI setleI order-less-imp-le*)

lemma *lemma-SReal-lub*:
(x::*hypreal*) $\in$ *Reals* ==> *isLub Reals* $\{s.\ s \in Reals\ \&\ s < x\}\ x$
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*:
[| $x \in HFinite$;
isLub Reals $\{s.\ s \in Reals\ \&\ s < x\}\ t$;
$r \in Reals$; $0 < r$ |]
==> $x + -t \neq r$
apply *auto*
apply (*frule isLubD1a* [*THEN SReal-minus*])
apply (*drule SReal-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*:
[| $x \in HFinite$;
isLub Reals $\{s.\ s \in Reals\ \&\ s < x\}\ t$;
$r \in Reals$; $0 < r$ |]
==> $-(x + -t) \neq r$
apply (*auto*)
apply (*frule isLubD1a*)
apply (*drule SReal-add-cancel*, *assumption*)
apply (*drule-tac* $x = -x$ **in** *SReal-minus*, *simp*)
apply (*drule-tac* $x = x$ **in** *lemma-SReal-lub*)
apply (*drule hypreal-isLub-unique*, *assumption*, *auto*)
done

lemma *lemma-st-part-major*:
[| $x \in HFinite$;
isLub Reals $\{s.\ s \in Reals\ \&\ s < x\}\ t$;
$r \in Reals$; $0 < r$ |]
==> $abs\ (x + -t) < r$
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*:
[| $x \in HFinite$; *isLub Reals* $\{s.\ s \in Reals\ \&\ s < x\}\ t$ |]
==> $\forall r \in Reals.\ 0 < r \longrightarrow abs\ (x + -t) < r$
by (*blast dest*!: *lemma-st-part-major*)

Existence of real and Standard Part Theorem

lemma *lemma-st-part-Ex*:
$x \in HFinite$ ==> $\exists t \in Reals.\ \forall r \in Reals.\ 0 < r \longrightarrow abs\ (x + -t) < r$
apply (*frule lemma-st-part-lub*, *safe*)
apply (*frule isLubD1a*)
apply (*blast dest*: *lemma-st-part-major2*)
done

lemma *st-part-Ex*:
  $x \in HFinite \Longrightarrow \exists t \in Reals.\ x\ @=\ t$
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*: $x \in HFinite \Longrightarrow EX!\ t.\ t \in Reals\ \&\ x\ @=\ t$
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

13.9 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*: $x \in HFinite$ **shows** $x \notin HInfinite$
proof
  assume *x'*: $x \in HInfinite$
  with *x* **have** $x \in HFinite \cap HInfinite$ **by** *blast*
  thus *False* **by** *auto*
qed

lemma *not-HFinite-HInfinite*: $x \notin HFinite \Longrightarrow x \in HInfinite$
apply (*simp add*: *HInfinite-def HFinite-def*, *auto*)
apply (*drule-tac x* = *r* + *1* **in** *bspec*)
apply (*auto*)
done

lemma *HInfinite-HFinite-disj*: $x \in HInfinite \mid x \in HFinite$
by (*blast intro*: *not-HFinite-HInfinite*)

lemma *HInfinite-HFinite-iff*: $(x \in HInfinite) = (x \notin HFinite)$
by (*blast dest*: *HFinite-not-HInfinite not-HFinite-HInfinite*)

lemma *HFinite-HInfinite-iff*: $(x \in HFinite) = (x \notin HInfinite)$
by (*simp add*: *HInfinite-HFinite-iff*)

lemma *HInfinite-diff-HFinite-Infinitesimal-disj*:
  $x \notin Infinitesimal \Longrightarrow x \in HInfinite \mid x \in HFinite - Infinitesimal$
by (*fast intro*: *not-HFinite-HInfinite*)

lemma *HFinite-inverse*:

[| $x \in HFinite$; $x \notin Infinitesimal$ |] ==> $inverse\ x \in HFinite$
apply (*cut-tac* $x = inverse\ x$ **in** *HInfinite-HFinite-disj*)
apply (*auto dest*!: *HInfinite-inverse-Infinitesimal*)
done

lemma *HFinite-inverse2*: $x \in HFinite - Infinitesimal$ ==> $inverse\ x \in HFinite$
by (*blast intro*: *HFinite-inverse*)

lemma *Infinitesimal-inverse-HFinite*:
$x \notin Infinitesimal$ ==> $inverse(x) \in HFinite$
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*:
$x \in HFinite - Infinitesimal$ ==> $inverse\ x \in HFinite - Infinitesimal$
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*:
[| 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 *hypreal-of-real-approx-inverse* = *hypreal-of-real-HFinite-diff-Infinitesimal* [*THEN* [*2*] *approx-inverse*]

lemma *inverse-add-Infinitesimal-approx*:
[| $x \in HFinite - Infinitesimal$;
$h \in Infinitesimal$ |] ==> $inverse(x + h)$ @= $inverse\ x$
apply (*auto intro*: *approx-inverse approx-sym Infinitesimal-add-approx-self*)
done

lemma *inverse-add-Infinitesimal-approx2*:
[| $x \in HFinite - Infinitesimal$;
$h \in Infinitesimal$ |] ==> $inverse(h + x)$ @= $inverse\ x$
apply (*rule add-commute* [*THEN subst*])

apply (*blast intro*: *inverse-add-Infinitesimal-approx*)
done

lemma *inverse-add-Infinitesimal-approx-Infinitesimal*:
[| $x \in$ *HFinite* $-$ *Infinitesimal*;
h $\in$ *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*: $(x \in$ *Infinitesimal*$) = (x*x \in$ *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*]: $(x*x \in$ *HFinite*$) = (x \in$ *HFinite*$)$
apply (*auto intro*: *HFinite-mult*)
apply (*auto dest*: *HInfinite-mult simp add*: *HFinite-HInfinite-iff*)
done

lemma *HInfinite-square-iff* [*simp*]: $(x*x \in$ *HInfinite*$) = (x \in$ *HInfinite*$)$
by (*auto simp add*: *HInfinite-HFinite-iff*)

lemma *approx-HFinite-mult-cancel*:
[| *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*:
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*:
[| $x \in$ *HInfinite*; $x \leq y$; $0 \leq x$ |] ==> $y \in$ *HInfinite*
by (*auto intro*: *HFinite-bounded simp add*: *HInfinite-HFinite-iff*)

lemma *Infinitesimal-inverse-HInfinite*:
[| $x \in$ *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*:
[| $x \in$ *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*:
[| $x \in$ *HInfinite*; $y \in$ *HFinite* $-$ *Infinitesimal* |]
==> $y * x \in$ *HInfinite*
by (*auto simp add*: *mult-commute HInfinite-HFinite-not-Infinitesimal-mult*)

lemma *HInfinite-gt-SReal*: [| $x \in$ *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*: [| $x \in$ *HInfinite*; $0 < x$ |] ==> $1 < x$
by (*auto intro*: *HInfinite-gt-SReal*)

lemma *not-HInfinite-one* [*simp*]: $1 \notin$ *HInfinite*
apply (*simp* (*no-asm*) *add*: *HInfinite-HFinite-iff*)
done

lemma *approx-hrabs-disj*: *abs* x @= x | *abs* x @= $-x$
by (*cut-tac* $x = x$ **in** *hrabs-disj*, *auto*)

13.10 Theorems about Monads

lemma *monad-hrabs-Un-subset*: *monad* (*abs* x) $\leq$ *monad*(x) *Un* *monad*($-x$)
by (*rule-tac x1* $= x$ **in** *hrabs-disj* [*THEN disjE*], *auto*)

lemma *Infinitesimal-monad-eq*: $e \in$ *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*: $(u \in monad\ x) = (-u \in monad\ (-x))$
by (*simp add*: *monad-def*)

lemma *Infinitesimal-monad-zero-iff*: $(x \in Infinitesimal) = (x \in monad\ 0)$
by (*auto intro*: *approx-sym simp add*: *monad-def mem-infmal-iff*)

lemma *monad-zero-minus-iff*: $(x \in monad\ 0) = (-x \in monad\ 0)$
apply (*simp* (*no-asm*) *add*: *Infinitesimal-monad-zero-iff* [*symmetric*])
done

lemma *monad-zero-hrabs-iff*: $(x \in monad\ 0) = (abs\ x \in monad\ 0)$
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*]: $x \in monad\ x$
by (*simp add*: *monad-def*)

13.11 Proof that $x \approx y$ implies $|x| \approx |y|$

lemma *approx-subset-monad*: $x\ @=\ y \Longrightarrow \{x,y\} \leq monad\ x$
apply (*simp* (*no-asm*))
apply (*simp add*: *approx-monad-iff*)
done

lemma *approx-subset-monad2*: $x\ @=\ y \Longrightarrow \{x,y\} \leq monad\ y$
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; $x \in$ *Infinitesimal* |] ==> *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*:
[| $0 < x$; $x \notin$*Infinitesimal*; e :*Infinitesimal* |] ==> $e < x$
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*:
[| $0 < x$; $x \notin$ *Infinitesimal*; $u \in$ *monad* x |] ==> $0 < u$
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*:
[| $x < 0$; $x \notin$ *Infinitesimal*; $u \in$ *monad* x |] ==> $u < 0$
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*:
[|$0 < x$; $x \notin$ *Infinitesimal*; x @= y|] ==> $0 < y$
by (*blast dest*: *Ball-mem-monad-gt-zero approx-subset-monad*)

lemma *lemma-approx-less-zero*:
[|$x < 0$; $x \notin$ *Infinitesimal*; x @= y|] ==> $y < 0$
by (*blast dest*: *Ball-mem-monad-less-zero approx-subset-monad*)

theorem *approx-hrabs*: x @= y ==> *abs* x @= *abs* y
apply (*case-tac* $x \in$ *Infinitesimal*)
apply (*simp add*: *Infinitesimal-approx-hrabs*)
apply (*rule linorder-cases* [*of 0 x*])
apply (*frule lemma-approx-gt-zero* [*of x y*])
apply (*auto simp add*: *lemma-approx-less-zero* [*of x y*] *abs-of-neg*)
done

lemma *approx-hrabs-zero-cancel*: *abs*(x) @= *0* ==> x @= *0*
apply (*cut-tac* $x = x$ **in** *hrabs-disj*)
apply (*auto dest*: *approx-minus*)
done

lemma *approx-hrabs-add-Infinitesimal*: $e \in$ *Infinitesimal* ==> *abs* x @= *abs*(x+e)

by (*fast intro*: *approx-hrabs Infinitesimal-add-approx-self*)

lemma *approx-hrabs-add-minus-Infinitesimal*:
  $e \in Infinitesimal \Longrightarrow abs\ x \mathbin{@=} abs(x + -e)$
by (*fast intro*: *approx-hrabs Infinitesimal-add-minus-approx-self*)

lemma *hrabs-add-Infinitesimal-cancel*:
  $[|\ e \in Infinitesimal;\ e' \in Infinitesimal;$
    $abs(x+e) = abs(y+e')|] \Longrightarrow abs\ x \mathbin{@=} abs\ y$
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*:
  $[|\ e \in Infinitesimal;\ e' \in Infinitesimal;$
    $abs(x + -e) = abs(y + -e')|] \Longrightarrow abs\ x \mathbin{@=} abs\ y$
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

lemma *Infinitesimal-add-hypreal-of-real-less*:
  $[|\ x < y;\ \ u \in Infinitesimal\ |]$
  $\Longrightarrow hypreal\text{-}of\text{-}real\ x + u < hypreal\text{-}of\text{-}real\ y$
apply (*simp add*: *Infinitesimal-def*)
apply (*drule-tac* $x = hypreal\text{-}of\text{-}real\ y + -hypreal\text{-}of\text{-}real\ x$ **in** *bspec*, *simp*)
apply (*simp add*: *abs-less-iff*)
done

lemma *Infinitesimal-add-hrabs-hypreal-of-real-less*:
  $[|\ x \in Infinitesimal;\ abs(hypreal\text{-}of\text{-}real\ r) < hypreal\text{-}of\text{-}real\ y\ |]$
  $\Longrightarrow abs\ (hypreal\text{-}of\text{-}real\ r + x) < hypreal\text{-}of\text{-}real\ y$
apply (*drule-tac* $x = hypreal\text{-}of\text{-}real\ r$ **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: *hypreal-of-real-hrabs*)
done

lemma *Infinitesimal-add-hrabs-hypreal-of-real-less2*:
  $[|\ x \in Infinitesimal;\ \ abs(hypreal\text{-}of\text{-}real\ r) < hypreal\text{-}of\text{-}real\ y\ |]$
  $\Longrightarrow abs\ (x + hypreal\text{-}of\text{-}real\ r) < hypreal\text{-}of\text{-}real\ y$
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*:

[| $u \in$ *Infinitesimal*; $v \in$ *Infinitesimal*;
hypreal-of-real $x + u \leq$ *hypreal-of-real* $y + v$ |]
==> *hypreal-of-real* $x \leq$ *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*:
[| $u \in$ *Infinitesimal*; $v \in$ *Infinitesimal*;
hypreal-of-real $x + u \leq$ *hypreal-of-real* $y + v$ |]
==> $x \leq y$
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* $x < e$; $e \in$ *Infinitesimal* |] ==> *hypreal-of-real* $x \leq 0$
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*:
[| $h \in$ *Infinitesimal*; $x \neq 0$ |] ==> *hypreal-of-real* $x + h \neq 0$
apply *auto*
apply (*subgoal-tac* $h = -$ *hypreal-of-real* x, *auto*)
done

lemma *Infinitesimal-square-cancel* [*simp*]:
$x*x + y*y \in$ *Infinitesimal* ==> $x*x \in$ *Infinitesimal*
apply (*rule Infinitesimal-interval2*)
apply (*rule-tac* [*3*] *zero-le-square*, *assumption*)
apply (*auto simp add*: *zero-le-square*)
done

lemma *HFinite-square-cancel* [*simp*]: $x*x + y*y \in$ *HFinite* ==> $x*x \in$ *HFinite*
apply (*rule HFinite-bounded*, *assumption*)
apply (*auto simp add*: *zero-le-square*)
done

lemma *Infinitesimal-square-cancel2* [*simp*]:
$x*x + y*y \in$ *Infinitesimal* ==> $y*y \in$ *Infinitesimal*
apply (*rule Infinitesimal-square-cancel*)
apply (*rule add-commute* [*THEN subst*])
apply (*simp* (*no-asm*))
done

lemma *HFinite-square-cancel2* [*simp*]: $x*x + y*y \in$ *HFinite* ==> $y*y \in$ *HFinite*

apply (*rule HFinite-square-cancel*)
apply (*rule add-commute* [*THEN subst*])
apply (*simp* (*no-asm*))
done

lemma *Infinitesimal-sum-square-cancel* [*simp*]:
*x***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*)
done

lemma *HFinite-sum-square-cancel* [*simp*]:
*x***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*)
done

lemma *Infinitesimal-sum-square-cancel2* [*simp*]:
*y***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***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***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***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*:
  $x \in monad\ (hypreal\text{-}of\text{-}real\ \ a) \Longrightarrow x \in HFinite$
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

13.12 Theorems about Standard Part

lemma *st-approx-self*: $x \in HFinite \Longrightarrow st\ x \mathbin{@=} x$
apply (*simp add*: *st-def*)
apply (*frule st-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *st-SReal*: $x \in HFinite \Longrightarrow st\ x \in Reals$
apply (*simp add*: *st-def*)
apply (*frule st-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
done

lemma *st-HFinite*: $x \in HFinite \Longrightarrow st\ x \in HFinite$
by (*erule st-SReal* [*THEN SReal-subset-HFinite* [*THEN subsetD*]])

lemma *st-SReal-eq*: $x \in Reals \Longrightarrow st\ x = x$
apply (*simp add*: *st-def*)
apply (*rule some-equality*)
apply (*fast intro*: *SReal-subset-HFinite* [*THEN subsetD*])
apply (*blast dest*: *SReal-approx-iff* [*THEN iffD1*])
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*: $[\!|\ x \in HFinite; y \in HFinite; st\ x = st\ y\ |\!] \Longrightarrow x \mathbin{@=} y$
by (*auto dest*!: *st-approx-self elim*!: *approx-trans3*)

lemma *approx-st-eq*:
  assumes $x \in HFinite$ **and** $y \in HFinite$ **and** $x \mathbin{@=} y$
  shows $st\ x = st\ y$
proof −

have *st x @= x st y @= y st x ∈ Reals st y ∈ 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 ∈ HFinite*; *y ∈ HFinite*|]
==> (*x @= y*) = (*st x* = *st y*)
by (*blast intro*: *approx-st-eq st-eq-approx*)

lemma *st-Infinitesimal-add-SReal*:
[| *x ∈ Reals*; *e ∈ Infinitesimal* |] ==> *st*(*x* + *e*) = *x*
apply (*frule st-SReal-eq* [*THEN subst*])
prefer *2* **apply** *assumption*
apply (*frule SReal-subset-HFinite* [*THEN subsetD*])
apply (*frule Infinitesimal-subset-HFinite* [*THEN subsetD*])
apply (*drule st-SReal-eq*)
apply (*rule approx-st-eq*)
apply (*auto intro*: *HFinite-add simp add*: *Infinitesimal-add-approx-self* [*THEN approx-sym*])
done

lemma *st-Infinitesimal-add-SReal2*:
[| *x ∈ Reals*; *e ∈ Infinitesimal* |] ==> *st*(*e* + *x*) = *x*
apply (*rule add-commute* [*THEN subst*])
apply (*blast intro*!: *st-Infinitesimal-add-SReal*)
done

lemma *HFinite-st-Infinitesimal-add*:
x ∈ HFinite ==> ∃ *e ∈ Infinitesimal*. *x* = *st*(*x*) + *e*
by (*blast dest*!: *st-approx-self* [*THEN approx-sym*] *bex-Infinitesimal-iff2* [*THEN iffD2*])

lemma *st-add*:
assumes *x*: *x ∈ HFinite* **and** *y*: *y ∈ HFinite*
shows *st* (*x* + *y*) = *st*(*x*) + *st*(*y*)
proof −
from *HFinite-st-Infinitesimal-add* [*OF x*]
obtain *ex* **where** *ex*: *ex ∈ Infinitesimal st x* + *ex* = *x*
by (*blast intro*: *sym*)
from *HFinite-st-Infinitesimal-add* [*OF y*]
obtain *ey* **where** *ey*: *ey ∈ Infinitesimal st y* + *ey* = *y*
by (*blast intro*: *sym*)
have *st* (*x* + *y*) = *st* ((*st x* + *ex*) + (*st y* + *ey*))
by (*simp add*: *ex ey*)
also have ... = *st* ((*ex* + *ey*) + (*st x* + *st y*)) **by** (*simp add*: *add-ac*)
also have ... = *st x* + *st y*
by (*simp add*: *prems st-SReal Infinitesimal-add*

st-Infinitesimal-add-SReal2)
finally show *?thesis* **.**
qed

lemma *st-number-of* [*simp*]: *st* (*number-of w*) = *number-of w*
by (*rule SReal-number-of* [*THEN st-SReal-eq*])

lemma [*simp*]: *st 0 = 0 st 1 = 1*
by (*simp-all add*: *st-SReal-eq*)

lemma *st-minus*: **assumes** *y* ∈ *HFinite* **shows** *st*(−*y*) = −*st*(*y*)
proof −
have *st* (− *y*) + *st y* = *0*
by (*simp add*: *prems st-add* [*symmetric*] *HFinite-minus-iff*)
thus *?thesis* **by** *arith*
qed

lemma *st-diff*: [| *x* ∈ *HFinite*; *y* ∈ *HFinite* |] ==> *st* (*x*−*y*) = *st*(*x*) − *st*(*y*)
apply (*simp add*: *diff-def*)
apply (*frule-tac y1* = *y* **in** *st-minus* [*symmetric*])
apply (*drule-tac x1* = *y* **in** *HFinite-minus-iff* [*THEN iffD2*])
apply (*simp* (*no-asm-simp*) *add*: *st-add*)
done

lemma *lemma-st-mult*:
[| *x* ∈ *HFinite*; *y* ∈ *HFinite*; *e* ∈ *Infinitesimal*; *ea* ∈ *Infinitesimal* |]
==> *e*∗*y* + *x*∗*ea* + *e*∗*ea* ∈ *Infinitesimal*
apply (*frule-tac x* = *e* **and** *y* = *y* **in** *Infinitesimal-HFinite-mult*)
apply (*frule-tac* [*2*] *x* = *ea* **and** *y* = *x* **in** *Infinitesimal-HFinite-mult*)
apply (*drule-tac* [*3*] *Infinitesimal-mult*)
apply (*auto intro*: *Infinitesimal-add simp add*: *add-ac mult-ac*)
done

lemma *st-mult*: [| *x* ∈ *HFinite*; *y* ∈ *HFinite* |] ==> *st* (*x* ∗ *y*) = *st*(*x*) ∗ *st*(*y*)
apply (*frule HFinite-st-Infinitesimal-add*)
apply (*frule-tac x* = *y* **in** *HFinite-st-Infinitesimal-add*, *safe*)
apply (*subgoal-tac st* (*x* ∗ *y*) = *st* ((*st x* + *e*) ∗ (*st y* + *ea*)))
apply (*drule-tac* [*2*] *sym*, *drule-tac* [*2*] *sym*)
prefer *2* **apply** *simp*
apply (*erule-tac V* = *x* = *st x* + *e* **in** *thin-rl*)
apply (*erule-tac V* = *y* = *st y* + *ea* **in** *thin-rl*)
apply (*simp add*: *left-distrib right-distrib*)
apply (*drule st-SReal*)+
apply (*simp* (*no-asm-use*) *add*: *add-assoc*)
apply (*rule st-Infinitesimal-add-SReal*)
apply (*blast intro*!: *SReal-mult*)
apply (*drule SReal-subset-HFinite* [*THEN subsetD*])+
apply (*rule add-assoc* [*THEN subst*])

apply (*blast intro*!: *lemma-st-mult*)
done

lemma *st-Infinitesimal*: $x \in Infinitesimal \Longrightarrow st\ x = 0$
apply (*subst numeral-0-eq-0* [*symmetric*])
apply (*rule st-number-of* [*THEN subst*])
apply (*rule approx-st-eq*)
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
simp *add*: *mem-infmal-iff* [*symmetric*])
done

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*:
$[\!| x \in HFinite; y \in HFinite; u \in Infinitesimal; st\ x \leq st\ y + u |\!] \Longrightarrow st\ x \leq st\ y$
apply (*simp add*: *linorder-not-less* [*symmetric*])
apply (*auto dest*: *Infinitesimal-add-st-less*)
done

lemma *st-le*: $[\!| x \in HFinite; y \in HFinite; x \leq y |\!] \Longrightarrow st(x) \leq st(y)$
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* ≤ *x*; *x* ∈ *HFinite* |] ==> *0* ≤ *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* ≤ *0*; *x* ∈ *HFinite* |] ==> *st x* ≤ *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* ∈ *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

13.13 Alternative Definitions for *HFinite* using Free Ultrafilter

lemma *FreeUltrafilterNat-Rep-hypreal*:
[| *X* ∈ *Rep-star x*; *Y* ∈ *Rep-star x* |]
==> {*n*. *X n* = *Y n*} ∈ *FreeUltrafilterNat*
by (*cases x*, *unfold star-n-def*, *auto*, *ultra*)

lemma *HFinite-FreeUltrafilterNat*:
x ∈ *HFinite*
==> ∃ *X* ∈ *Rep-star x*. ∃ *u*. {*n*. *abs* (*X n*) < *u*} ∈ *FreeUltrafilterNat*
apply (*cases x*)
apply (*auto simp add*: *HFinite-def abs-less-iff minus-less-iff* [*of x*]
star-of-def
star-n-less SReal-iff star-n-minus)
apply (*rule-tac x*=*X* **in** *bexI*)
apply (*rule-tac x*=*y* **in** *exI*, *ultra*)
apply *simp*
done

lemma *FreeUltrafilterNat-HFinite*:
∃ *X* ∈ *Rep-star x*.
∃ *u*. {*n*. *abs* (*X n*) < *u*} ∈ *FreeUltrafilterNat*
==> *x* ∈ *HFinite*
apply (*cases x*)
apply (*auto simp add*: *HFinite-def abs-less-iff minus-less-iff* [*of x*])

apply (*rule-tac x = hypreal-of-real u* **in** *bexI*)
apply (*auto simp add*: *star-n-less SReal-iff star-n-minus star-of-def*)
apply *ultra+*
done

lemma *HFinite-FreeUltrafilterNat-iff*:
(*x* ∈ *HFinite*) = (∃ *X* ∈ *Rep-star x*.
∃ *u*. {*n*. *abs* (*X n*) < *u*} ∈ *FreeUltrafilterNat*)
by (*blast intro*!: *HFinite-FreeUltrafilterNat FreeUltrafilterNat-HFinite*)

13.14 Alternative Definitions for *HInfinite* using Free Ultrafilter

lemma *lemma-Compl-eq*: − {*n*. (*u*::*real*) < *abs* (*xa n*)} = {*n*. *abs* (*xa n*) ≤ *u*}
by *auto*

lemma *lemma-Compl-eq2*: − {*n*. *abs* (*xa n*) < (*u*::*real*)} = {*n*. *u* ≤ *abs* (*xa n*)}
by *auto*

lemma *lemma-Int-eq1*:
{*n*. *abs* (*xa n*) ≤ (*u*::*real*)} *Int* {*n*. *u* ≤ *abs* (*xa n*)}
= {*n*. *abs*(*xa n*) = *u*}
by *auto*

lemma *lemma-FreeUltrafilterNat-one*:
{*n*. *abs* (*xa n*) = *u*} ≤ {*n*. *abs* (*xa n*) < *u* + (*1*::*real*)}
by *auto*

lemma *FreeUltrafilterNat-const-Finite*:
[| *xa*: *Rep-star x*;
{*n*. *abs* (*xa n*) = *u*} ∈ *FreeUltrafilterNat*
|] ==> *x* ∈ *HFinite*
apply (*rule FreeUltrafilterNat-HFinite*)
apply (*rule-tac x = xa* **in** *bexI*)
apply (*rule-tac x = u + 1* **in** *exI*)
apply (*ultra*, *assumption*)
done

lemma *HInfinite-FreeUltrafilterNat*:
x ∈ *HInfinite* ==> ∃ *X* ∈ *Rep-star x*.
∀ *u*. {*n*. *u* < *abs* (*X n*)} ∈ *FreeUltrafilterNat*
apply (*frule HInfinite-HFinite-iff* [*THEN iffD1*])
apply (*cut-tac x = x* **in** *Rep-hypreal-nonempty*)
apply (*auto simp del*: *Rep-hypreal-nonempty simp add*: *HFinite-FreeUltrafilterNat-iff Bex-def*)
apply (*drule spec*)+
apply *auto*
apply (*drule-tac x = u* **in** *spec*)

apply (*drule FreeUltrafilterNat-Compl-mem*)+
apply (*drule FreeUltrafilterNat-Int*, *assumption*)
apply (*simp add*: *lemma-Compl-eq lemma-Compl-eq2 lemma-Int-eq1*)
apply (*auto dest*: *FreeUltrafilterNat-const-Finite simp*
add: *HInfinite-HFinite-iff* [*THEN iffD1*])
done

lemma *lemma-Int-HI*:
$\{n.\ abs\ (Xa\ n) < u\}\ Int\ \{n.\ X\ n = Xa\ n\} \subseteq \{n.\ abs\ (X\ n) < (u::real)\}$
by *auto*

lemma *lemma-Int-HIa*: $\{n.\ u < abs\ (X\ n)\}\ Int\ \{n.\ abs\ (X\ n) < (u::real)\} = \{\}$
by (*auto intro*: *order-less-asym*)

lemma *FreeUltrafilterNat-HInfinite*:
$\exists X \in Rep\text{-}star\ x.\ \forall u.$
$\{n.\ u < abs\ (X\ n)\} \in FreeUltrafilterNat$
$==>\ x \in HInfinite$
apply (*rule HInfinite-HFinite-iff* [*THEN iffD2*])
apply (*safe*, *drule HFinite-FreeUltrafilterNat*, *auto*)
apply (*drule-tac* $x = u$ **in** *spec*)
apply (*drule FreeUltrafilterNat-Rep-hypreal*, *assumption*)
apply (*drule-tac* $Y = \{n.\ X\ n = Xa\ n\}$ **in** *FreeUltrafilterNat-Int*, *simp*)
apply (*drule lemma-Int-HI* [*THEN* [*2*] *FreeUltrafilterNat-subset*])
apply (*drule-tac* $Y = \{n.\ abs\ (X\ n) < u\}$ **in** *FreeUltrafilterNat-Int*)
apply (*auto simp add*: *lemma-Int-HIa*)
done

lemma *HInfinite-FreeUltrafilterNat-iff*:
$(x \in HInfinite) = (\exists X \in Rep\text{-}star\ x.$
$\forall u.\ \{n.\ u < abs\ (X\ n)\} \in FreeUltrafilterNat)$
by (*blast intro*!: *HInfinite-FreeUltrafilterNat FreeUltrafilterNat-HInfinite*)

13.15 Alternative Definitions for *Infinitesimal* using Free Ultrafilter

lemma *Infinitesimal-FreeUltrafilterNat*:
$x \in Infinitesimal ==> \exists X \in Rep\text{-}star\ x.$
$\forall u.\ 0 < u --> \{n.\ abs\ (X\ n) < u\} \in FreeUltrafilterNat$
apply (*simp add*: *Infinitesimal-def*)
apply (*auto simp add*: *abs-less-iff minus-less-iff* [*of x*])
apply (*cases x*)
apply (*auto*, *rule bexI* [*OF - Rep-star-star-n*], *safe*)
apply (*drule star-of-less* [*THEN iffD2*])
apply (*drule-tac* $x = hypreal\text{-}of\text{-}real\ u$ **in** *bspec*, *auto*)
apply (*auto simp add*: *star-n-less star-n-minus star-of-def*, *ultra*)
done

lemma *FreeUltrafilterNat-Infinitesimal*:

$\exists X \in \mathit{Rep\text{-}star}\ x.$
$\forall u.\ 0 < u \longrightarrow \{n.\ \mathit{abs}\ (X\ n) < u\} \in \mathit{FreeUltrafilterNat}$
$\Longrightarrow x \in \mathit{Infinitesimal}$
apply (*simp add*: *Infinitesimal-def*)
apply (*cases x*)
apply (*auto simp add*: *abs-less-iff abs-interval-iff minus-less-iff* [*of x*])
apply (*auto simp add*: *SReal-iff*)
apply (*drule-tac* [!] *x=y* **in** *spec*)
apply (*auto simp add*: *star-n-less star-n-minus star-of-def*, *ultra+*)
done

lemma *Infinitesimal-FreeUltrafilterNat-iff*:
$(x \in \mathit{Infinitesimal}) = (\exists X \in \mathit{Rep\text{-}star}\ x.$
$\forall u.\ 0 < u \longrightarrow \{n.\ \mathit{abs}\ (X\ n) < u\} \in \mathit{FreeUltrafilterNat})$
by (*blast intro*!: *Infinitesimal-FreeUltrafilterNat FreeUltrafilterNat-Infinitesimal*)

lemma *lemma-Infinitesimal*:
$(\forall r.\ 0 < r \longrightarrow x < r) = (\forall n.\ x < \mathit{inverse}(\mathit{real}\ (\mathit{Suc}\ n)))$
apply (*auto simp add*: *real-of-nat-Suc-gt-zero*)
apply (*blast dest*!: *reals-Archimedean intro*: *order-less-trans*)
done

lemma *of-nat-in-Reals* [*simp*]: (*of-nat n*::*hypreal*) $\in \mathbb{R}$
apply (*induct n*)
apply (*simp-all*)
done

lemma *lemma-Infinitesimal2*:
$(\forall r \in \mathit{Reals}.\ 0 < r \longrightarrow x < r) =$
$(\forall n.\ x < \mathit{inverse}(\mathit{hypreal\text{-}of\text{-}nat}\ (\mathit{Suc}\ n)))$
apply *safe*
apply (*drule-tac x* = *inverse* (*hypreal-of-real* (*real* (*Suc n*))) **in** *bspec*)
apply (*simp* (*no-asm-use*) *add*: *SReal-inverse*)
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-Suc-gt-zero hypreal-of-nat-eq*)
apply (*auto dest*!: *reals-Archimedean simp add*: *SReal-iff*)
apply (*drule star-of-less* [*THEN iffD2*])
apply (*simp add*: *real-of-nat-Suc-gt-zero hypreal-of-nat-eq*)
apply (*blast intro*: *order-less-trans*)
done

lemma *Infinitesimal-hypreal-of-nat-iff*:
Infinitesimal = $\{x.\ \forall n.\ \mathit{abs}\ x < \mathit{inverse}\ (\mathit{hypreal\text{-}of\text{-}nat}\ (\mathit{Suc}\ n))\}$
apply (*simp add*: *Infinitesimal-def*)

apply (*auto simp add*: *lemma-Infinitesimal2*)
done

13.16 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-Compl-mem*)
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-Compl-mem simp add*: *Compl-real-le-eq*)
done

theorem *HInfinite-omega* [*simp*]: *omega* $\in$ *HInfinite*
apply (*simp add*: *omega-def star-n-def*)
apply (*auto intro*!: *FreeUltrafilterNat-HInfinite*)
apply (*rule bexI*)
apply (*rule-tac* [*2*] *lemma-starrel-refl*, *auto*)
apply (*simp* (*no-asm*) *add*: *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*:
$0 < u \implies (u < inverse\ (real(Suc\ n))) = (real(Suc\ n) < inverse\ u)$
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*:
$0 < u \implies finite\ \{n.\ u < inverse(real(Suc\ n))\}$
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*:
$\{n.\ u \leq inverse(real(Suc\ n))\} =$
$\{n.\ u < inverse(real(Suc\ n))\}\ Un\ \{n.\ u = inverse(real(Suc\ n))\}$

apply (*auto dest*: *order-le-imp-less-or-eq simp add*: *order-less-imp-le*)
done

lemma *real-of-nat-inverse-le-iff*:
(*inverse* (*real*(*Suc n*)) ≤ *r*) = (*1* ≤ *r* ∗ *real*(*Suc n*))
apply (*simp* (*no-asm*) *add*: *linorder-not-less* [*symmetric*])
apply (*simp* (*no-asm*) *add*: *inverse-eq-divide*)
apply (*subst pos-less-divide-eq*)
apply (*simp* (*no-asm*) *add*: *real-of-nat-Suc-gt-zero*)
apply (*simp* (*no-asm*) *add*: *real-mult-commute*)
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 real-not-refl2* [*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* ==>
{*n*. *inverse*(*real*(*Suc n*)) < *u*} ∈ *FreeUltrafilterNat*
apply (*cut-tac u* = *u* **in** *inverse-real-of-posnat-ge-real-FreeUltrafilterNat*)
apply (*auto dest*: *FreeUltrafilterNat-Compl-mem simp add*: *Compl-le-inverse-eq*)
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*:

$\forall n.\ abs(X\ n + -x) < inverse(real(Suc\ n))$
$==>\ star\text{-}n\ X + -hypreal\text{-}of\text{-}real\ x \in Infinitesimal$
apply (*auto intro*!: *bexI dest*: *FreeUltrafilterNat-inverse-real-of-posnat FreeUltrafilterNat-all FreeUltrafilterNat-Int intro*: *order-less-trans FreeUltrafilterNat-subset simp add*: *star-n-minus star-of-def star-n-add Infinitesimal-FreeUltrafilterNat-iff star-n-inverse*)
done

lemma *real-seq-to-hypreal-approx*:
$\forall n.\ abs(X\ n + -x) < inverse(real(Suc\ n))$
$==>\ star\text{-}n\ X\ @=\ hypreal\text{-}of\text{-}real\ x$
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*:
$\forall n.\ abs(x + -X\ n) < inverse(real(Suc\ n))$
$==>\ star\text{-}n\ X\ @=\ hypreal\text{-}of\text{-}real\ x$
apply (*simp add*: *abs-minus-add-cancel real-seq-to-hypreal-approx*)
done

lemma *real-seq-to-hypreal-Infinitesimal2*:
$\forall n.\ abs(X\ n + -Y\ n) < inverse(real(Suc\ n))$
$==>\ star\text{-}n\ X + -star\text{-}n\ Y \in Infinitesimal$
by (*auto intro*!: *bexI*
dest: *FreeUltrafilterNat-inverse-real-of-posnat*
FreeUltrafilterNat-all FreeUltrafilterNat-Int
intro: *order-less-trans FreeUltrafilterNat-subset*
simp add: *Infinitesimal-FreeUltrafilterNat-iff star-n-minus*
star-n-add star-n-inverse)

ML
⟪
val Infinitesimal-def = thmInfinitesimal-def;
val HFinite-def = thmHFinite-def;
val HInfinite-def = thmHInfinite-def;
val st-def = thmst-def;
val monad-def = thmmonad-def;
val galaxy-def = thmgalaxy-def;
val approx-def = thmapprox-def;
val SReal-def = thmSReal-def;

val Infinitesimal-approx-minus = thm Infinitesimal-approx-minus;
val approx-monad-iff = thm approx-monad-iff;
val Infinitesimal-approx = thm Infinitesimal-approx;
val approx-add = thm approx-add;
val approx-minus = thm approx-minus;
val approx-minus2 = thm approx-minus2;

val approx-minus-cancel = thm approx-minus-cancel;
val approx-add-minus = thm approx-add-minus;
val approx-mult1 = thm approx-mult1;
val approx-mult2 = thm approx-mult2;
val approx-mult-subst = thm approx-mult-subst;
val approx-mult-subst2 = thm approx-mult-subst2;
val approx-mult-subst-SReal = thm approx-mult-subst-SReal;
val approx-eq-imp = thm approx-eq-imp;
val Infinitesimal-minus-approx = thm Infinitesimal-minus-approx;
val bex-Infinitesimal-iff = thm bex-Infinitesimal-iff;
val bex-Infinitesimal-iff2 = thm bex-Infinitesimal-iff2;
val Infinitesimal-add-approx = thm Infinitesimal-add-approx;
val Infinitesimal-add-approx-self = thm Infinitesimal-add-approx-self;
val Infinitesimal-add-approx-self2 = thm Infinitesimal-add-approx-self2;
val Infinitesimal-add-minus-approx-self = thm Infinitesimal-add-minus-approx-self;
val Infinitesimal-add-cancel = thm Infinitesimal-add-cancel;
val Infinitesimal-add-right-cancel = thm Infinitesimal-add-right-cancel;
val approx-add-left-cancel = thm approx-add-left-cancel;
val approx-add-right-cancel = thm approx-add-right-cancel;
val approx-add-mono1 = thm approx-add-mono1;
val approx-add-mono2 = thm approx-add-mono2;
val approx-add-left-iff = thm approx-add-left-iff;
val approx-add-right-iff = thm approx-add-right-iff;
val approx-HFinite = thm approx-HFinite;
val approx-hypreal-of-real-HFinite = thm approx-hypreal-of-real-HFinite;
val approx-mult-HFinite = thm approx-mult-HFinite;
val approx-mult-hypreal-of-real = thm approx-mult-hypreal-of-real;
val approx-SReal-mult-cancel-zero = thm approx-SReal-mult-cancel-zero;
val approx-mult-SReal1 = thm approx-mult-SReal1;
val approx-mult-SReal2 = thm approx-mult-SReal2;
val approx-mult-SReal-zero-cancel-iff = thm approx-mult-SReal-zero-cancel-iff;
val approx-SReal-mult-cancel = thm approx-SReal-mult-cancel;
val approx-SReal-mult-cancel-iff1 = thm approx-SReal-mult-cancel-iff1;
val approx-le-bound = thm approx-le-bound;
val Infinitesimal-less-SReal = thm Infinitesimal-less-SReal;
val Infinitesimal-less-SReal2 = thm Infinitesimal-less-SReal2;
val SReal-not-Infinitesimal = thm SReal-not-Infinitesimal;
val SReal-minus-not-Infinitesimal = thm SReal-minus-not-Infinitesimal;
val SReal-Int-Infinitesimal-zero = thm SReal-Int-Infinitesimal-zero;
val SReal-Infinitesimal-zero = thm SReal-Infinitesimal-zero;
val SReal-HFinite-diff-Infinitesimal = thm SReal-HFinite-diff-Infinitesimal;
val hypreal-of-real-HFinite-diff-Infinitesimal = thm hypreal-of-real-HFinite-diff-Infinitesimal;
val hypreal-of-real-Infinitesimal-iff-0 = thm hypreal-of-real-Infinitesimal-iff-0;
val number-of-not-Infinitesimal = thm number-of-not-Infinitesimal;
val one-not-Infinitesimal = thm one-not-Infinitesimal;
val approx-SReal-not-zero = thm approx-SReal-not-zero;
val HFinite-diff-Infinitesimal-approx = thm HFinite-diff-Infinitesimal-approx;
val Infinitesimal-ratio = thm Infinitesimal-ratio;
val SReal-approx-iff = thm SReal-approx-iff;

val number-of-approx-iff = thm number-of-approx-iff;
val hypreal-of-real-approx-iff = thm hypreal-of-real-approx-iff;
val hypreal-of-real-approx-number-of-iff = thm hypreal-of-real-approx-number-of-iff;
val approx-unique-real = thm approx-unique-real;
val hypreal-isLub-unique = thm hypreal-isLub-unique;
val hypreal-setle-less-trans = thm hypreal-setle-less-trans;
val hypreal-gt-isUb = thm hypreal-gt-isUb;
val st-part-Ex = thm st-part-Ex;
val st-part-Ex1 = thm st-part-Ex1;
val HFinite-Int-HInfinite-empty = thm HFinite-Int-HInfinite-empty;
val HFinite-not-HInfinite = thm HFinite-not-HInfinite;
val not-HFinite-HInfinite = thm not-HFinite-HInfinite;
val HInfinite-HFinite-disj = thm HInfinite-HFinite-disj;
val HInfinite-HFinite-iff = thm HInfinite-HFinite-iff;
val HFinite-HInfinite-iff = thm HFinite-HInfinite-iff;
val HInfinite-diff-HFinite-Infinitesimal-disj = thm HInfinite-diff-HFinite-Infinitesimal-disj;
val HFinite-inverse = thm HFinite-inverse;
val HFinite-inverse2 = thm HFinite-inverse2;
val Infinitesimal-inverse-HFinite = thm Infinitesimal-inverse-HFinite;
val HFinite-not-Infinitesimal-inverse = thm HFinite-not-Infinitesimal-inverse;
val approx-inverse = thm approx-inverse;
val hypreal-of-real-approx-inverse = thm hypreal-of-real-approx-inverse;
val inverse-add-Infinitesimal-approx = thm inverse-add-Infinitesimal-approx;
val inverse-add-Infinitesimal-approx2 = thm inverse-add-Infinitesimal-approx2;
val inverse-add-Infinitesimal-approx-Infinitesimal = thm inverse-add-Infinitesimal-approx-Infinitesimal;
val Infinitesimal-square-iff = thm Infinitesimal-square-iff;
val HFinite-square-iff = thm HFinite-square-iff;
val HInfinite-square-iff = thm HInfinite-square-iff;
val approx-HFinite-mult-cancel = thm approx-HFinite-mult-cancel;
val approx-HFinite-mult-cancel-iff1 = thm approx-HFinite-mult-cancel-iff1;
val approx-hrabs-disj = thm approx-hrabs-disj;
val monad-hrabs-Un-subset = thm monad-hrabs-Un-subset;
val Infinitesimal-monad-eq = thm Infinitesimal-monad-eq;
val mem-monad-iff = thm mem-monad-iff;
val Infinitesimal-monad-zero-iff = thm Infinitesimal-monad-zero-iff;
val monad-zero-minus-iff = thm monad-zero-minus-iff;
val monad-zero-hrabs-iff = thm monad-zero-hrabs-iff;
val mem-monad-self = thm mem-monad-self;
val approx-subset-monad = thm approx-subset-monad;
val approx-subset-monad2 = thm approx-subset-monad2;
val mem-monad-approx = thm mem-monad-approx;
val approx-mem-monad = thm approx-mem-monad;
val approx-mem-monad2 = thm approx-mem-monad2;
val approx-mem-monad-zero = thm approx-mem-monad-zero;
val Infinitesimal-approx-hrabs = thm Infinitesimal-approx-hrabs;
val less-Infinitesimal-less = thm less-Infinitesimal-less;
val Ball-mem-monad-gt-zero = thm Ball-mem-monad-gt-zero;
val Ball-mem-monad-less-zero = thm Ball-mem-monad-less-zero;
val approx-hrabs = thm approx-hrabs;

val approx-hrabs-zero-cancel = thm approx-hrabs-zero-cancel;
val approx-hrabs-add-Infinitesimal = thm approx-hrabs-add-Infinitesimal;
val approx-hrabs-add-minus-Infinitesimal = thm approx-hrabs-add-minus-Infinitesimal;
val hrabs-add-Infinitesimal-cancel = thm hrabs-add-Infinitesimal-cancel;
val hrabs-add-minus-Infinitesimal-cancel = thm hrabs-add-minus-Infinitesimal-cancel;
val Infinitesimal-add-hypreal-of-real-less = thm Infinitesimal-add-hypreal-of-real-less;
val Infinitesimal-add-hrabs-hypreal-of-real-less = thm Infinitesimal-add-hrabs-hypreal-of-real-less;
val Infinitesimal-add-hrabs-hypreal-of-real-less2 = thm Infinitesimal-add-hrabs-hypreal-of-real-less2;
val hypreal-of-real-le-add-Infininitesimal-cancel2 = thmhypreal-of-real-le-add-Infininitesimal-cancel2;
val hypreal-of-real-less-Infinitesimal-le-zero = thm hypreal-of-real-less-Infinitesimal-le-zero;
val Infinitesimal-add-not-zero = thm Infinitesimal-add-not-zero;
val Infinitesimal-square-cancel = thm Infinitesimal-square-cancel;
val HFinite-square-cancel = thm HFinite-square-cancel;
val Infinitesimal-square-cancel2 = thm Infinitesimal-square-cancel2;
val HFinite-square-cancel2 = thm HFinite-square-cancel2;
val Infinitesimal-sum-square-cancel = thm Infinitesimal-sum-square-cancel;
val HFinite-sum-square-cancel = thm HFinite-sum-square-cancel;
val Infinitesimal-sum-square-cancel2 = thm Infinitesimal-sum-square-cancel2;
val HFinite-sum-square-cancel2 = thm HFinite-sum-square-cancel2;
val Infinitesimal-sum-square-cancel3 = thm Infinitesimal-sum-square-cancel3;
val HFinite-sum-square-cancel3 = thm HFinite-sum-square-cancel3;
val monad-hrabs-less = thm monad-hrabs-less;
val mem-monad-SReal-HFinite = thm mem-monad-SReal-HFinite;
val st-approx-self = thm st-approx-self;
val st-SReal = thm st-SReal;
val st-HFinite = thm st-HFinite;
val st-SReal-eq = thm st-SReal-eq;
val st-hypreal-of-real = thm st-hypreal-of-real;
val st-eq-approx = thm st-eq-approx;
val approx-st-eq = thm approx-st-eq;
val st-eq-approx-iff = thm st-eq-approx-iff;
val st-Infinitesimal-add-SReal = thm st-Infinitesimal-add-SReal;
val st-Infinitesimal-add-SReal2 = thm st-Infinitesimal-add-SReal2;
val HFinite-st-Infinitesimal-add = thm HFinite-st-Infinitesimal-add;
val st-add = thm st-add;
val st-number-of = thm st-number-of;
val st-minus = thm st-minus;
val st-diff = thm st-diff;
val st-mult = thm st-mult;
val st-Infinitesimal = thm st-Infinitesimal;
val st-not-Infinitesimal = thm st-not-Infinitesimal;
val st-inverse = thm st-inverse;
val st-divide = thm st-divide;
val st-idempotent = thm st-idempotent;
val Infinitesimal-add-st-less = thm Infinitesimal-add-st-less;
val Infinitesimal-add-st-le-cancel = thm Infinitesimal-add-st-le-cancel;
val st-le = thm st-le;
val st-zero-le = thm st-zero-le;
val st-zero-ge = thm st-zero-ge;

val st-hrabs = thm st-hrabs;
val FreeUltrafilterNat-HFinite = thm FreeUltrafilterNat-HFinite;
val HFinite-FreeUltrafilterNat-iff = thm HFinite-FreeUltrafilterNat-iff;
val FreeUltrafilterNat-const-Finite = thm FreeUltrafilterNat-const-Finite;
val FreeUltrafilterNat-HInfinite = thm FreeUltrafilterNat-HInfinite;
val HInfinite-FreeUltrafilterNat-iff = thm HInfinite-FreeUltrafilterNat-iff;
val Infinitesimal-FreeUltrafilterNat = thm Infinitesimal-FreeUltrafilterNat;
val FreeUltrafilterNat-Infinitesimal = thm FreeUltrafilterNat-Infinitesimal;
val Infinitesimal-FreeUltrafilterNat-iff = thm Infinitesimal-FreeUltrafilterNat-iff;
val Infinitesimal-hypreal-of-nat-iff = thm Infinitesimal-hypreal-of-nat-iff;
val Suc-Un-eq = thm Suc-Un-eq;
val finite-nat-segment = thm finite-nat-segment;
val finite-real-of-nat-segment = thm finite-real-of-nat-segment;
val finite-real-of-nat-less-real = thm finite-real-of-nat-less-real;
val finite-real-of-nat-le-real = thm finite-real-of-nat-le-real;
val finite-rabs-real-of-nat-le-real = thm finite-rabs-real-of-nat-le-real;
val rabs-real-of-nat-le-real-FreeUltrafilterNat = thm rabs-real-of-nat-le-real-FreeUltrafilterNat;
val FreeUltrafilterNat-nat-gt-real = thm FreeUltrafilterNat-nat-gt-real;
val FreeUltrafilterNat-omega = thm FreeUltrafilterNat-omega;
val HInfinite-omega = thm HInfinite-omega;
val Infinitesimal-epsilon = thm Infinitesimal-epsilon;
val HFinite-epsilon = thm HFinite-epsilon;
val epsilon-approx-zero = thm epsilon-approx-zero;
val real-of-nat-less-inverse-iff = thm real-of-nat-less-inverse-iff;
val finite-inverse-real-of-posnat-gt-real = thm finite-inverse-real-of-posnat-gt-real;
val real-of-nat-inverse-le-iff = thm real-of-nat-inverse-le-iff;
val real-of-nat-inverse-eq-iff = thm real-of-nat-inverse-eq-iff;
val finite-inverse-real-of-posnat-ge-real = thm finite-inverse-real-of-posnat-ge-real;
val inverse-real-of-posnat-ge-real-FreeUltrafilterNat = thm inverse-real-of-posnat-ge-real-FreeUltrafilterNat;
val FreeUltrafilterNat-inverse-real-of-posnat = thm FreeUltrafilterNat-inverse-real-of-posnat;
val real-seq-to-hypreal-Infinitesimal = thm real-seq-to-hypreal-Infinitesimal;
val real-seq-to-hypreal-approx = thm real-seq-to-hypreal-approx;
val real-seq-to-hypreal-approx2 = thm real-seq-to-hypreal-approx2;
val real-seq-to-hypreal-Infinitesimal2 = thm real-seq-to-hypreal-Infinitesimal2;
val HInfinite-HFinite-add = thm HInfinite-HFinite-add;
val HInfinite-ge-HInfinite = thm HInfinite-ge-HInfinite;
val Infinitesimal-inverse-HInfinite = thm Infinitesimal-inverse-HInfinite;
val HInfinite-HFinite-not-Infinitesimal-mult = thm HInfinite-HFinite-not-Infinitesimal-mult;
val HInfinite-HFinite-not-Infinitesimal-mult2 = thm HInfinite-HFinite-not-Infinitesimal-mult2;
val HInfinite-gt-SReal = thm HInfinite-gt-SReal;
val HInfinite-gt-zero-gt-one = thm HInfinite-gt-zero-gt-one;
val not-HInfinite-one = thm not-HInfinite-one;
⟩⟩

end

14 Star: Star-Transforms in Non-Standard Analysis

theory *Star*
imports *NSA*
begin

constdefs

starset-n :: (*nat* => *'a set*) => *'a star set* (∗*sn*∗ - [*80*] *80*)
∗*sn*∗ *As* == *Iset* (*star-n As*)

InternalSets :: *'a star set set*
InternalSets == {*X*. ∃ *As*. *X* = ∗*sn*∗ *As*}

is-starext :: [*'a star* => *'a star*, *'a* => *'a*] => *bool*
is-starext F f == (∀ *x y*. ∃ *X* ∈ *Rep-star*(*x*). ∃ *Y* ∈ *Rep-star*(*y*).
((*y* = (*F x*)) = ({*n*. *Y n* = *f*(*X n*)} : *FreeUltrafilterNat*)))

starfun-n :: (*nat* => (*'a* => *'b*)) => *'a star* => *'b star*
(∗*fn*∗ - [*80*] *80*)
∗*fn*∗ *F* == *Ifun* (*star-n F*)

InternalFuns :: (*'a star* => *'b star*) *set*
InternalFuns == {*X*. ∃ *F*. *X* = ∗*fn*∗ *F*}

lemma *no-choice*: ∀ *x*. ∃ *y*. *Q x y* ==> ∃(*f* :: *nat* => *nat*). ∀ *x*. *Q x* (*f x*)
apply (*rule-tac x* = %*x*. *LEAST y*. *Q x y* **in** *exI*)
apply (*blast intro*: *LeastI*)
done

14.1 Properties of the Star-transform Applied to Sets of Reals

lemma *STAR-UNIV-set*: ∗*s*∗(*UNIV*::*'a set*) = (*UNIV*::*'a star set*)
by (*transfer UNIV-def*, *rule refl*)

lemma *STAR-empty-set*: ∗*s*∗ {} = {}
by (*transfer empty-def*, *rule refl*)

lemma *STAR-Un*: ∗*s*∗ (*A Un B*) = ∗*s*∗ *A Un* ∗*s*∗ *B*
by (*transfer Un-def*, *rule refl*)

lemma *STAR-Int*: $*s*\ (A\ Int\ B) = *s*\ A\ Int\ *s*\ B$
by (*transfer Int-def*, *rule refl*)

lemma *STAR-Compl*: $*s*\ -A = -(\ *s*\ A)$
by (*transfer Compl-def*, *rule refl*)

lemma *STAR-mem-Compl*: $!!x.\ x \notin *s*\ F ==> x : *s*\ (-\ F)$
by (*transfer Compl-def*, *simp*)

lemma *STAR-diff*: $*s*\ (A - B) = *s*\ A - *s*\ B$
by (*transfer set-diff-def*, *rule refl*)

lemma *STAR-subset*: $A <= B ==> *s*\ A <= *s*\ B$
by (*transfer subset-def*, *simp*)

lemma *STAR-mem*: $a \in A ==>$ *star-of* $a : *s*\ A$
by *transfer*

lemma *STAR-mem-iff*: (*star-of* $x \in *s*\ A) = (x \in A)$
by (*transfer*, *rule refl*)

lemma *STAR-star-of-image-subset*: *star-of* ‘ $A <= *s*\ A$
by (*auto simp add*: *STAR-mem*)

lemma *STAR-hypreal-of-real-Int*: $*s*\ X\ Int\ Reals =$ *hypreal-of-real* ‘ X
by (*auto simp add*: *SReal-def STAR-mem-iff*)

lemma *lemma-not-hyprealA*: $x \notin$ *hypreal-of-real* ‘ $A ==> \forall\, y \in A.\ x \neq$ *hypreal-of-real* y
by *auto*

lemma *lemma-Compl-eq*: $-\ \{n.\ X\ n = xa\} = \{n.\ X\ n \neq xa\}$
by *auto*

lemma *STAR-real-seq-to-hypreal*:
$\forall\, n.\ (X\ n) \notin M ==>$ *star-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-of* $x\}$
by *simp*

lemma *STAR-not-mem*: $x \notin F ==>$ *star-of* $x \notin *s*\ F$
by *transfer*

lemma *STAR-subset-closed*: $[|\ x : *s*\ A;\ A <= B\ |] ==> x : *s*\ B$
by (*blast dest*: *STAR-subset*)

Nonstandard extension of a set (defined using a constant sequence) as a

special case of an internal set

lemma *starset-n-starset*: $\forall n.\ (As\ n = A) ==> *sn*\ As = *s*\ A$
apply (*drule expand-fun-eq* [*THEN iffD2*])
apply (*simp add*: *starset-n-def starset-def star-of-def*)
done

lemma *starfun-n-starfun*: $\forall n.\ (F\ n = f) ==> *fn*\ F = *f*\ f$
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

lemma *Rep-star-FreeUltrafilterNat*:
[| $X \in$ *Rep-star z*; $Y \in$ *Rep-star z* |]
==> $\{n.\ X\ n = Y\ n\}$: *FreeUltrafilterNat*
by (*rule FreeUltrafilterNat-Rep-hypreal*)

Nonstandard extension of functions

lemma *starfun*:
$(*f*\ f)\ (star\text{-}n\ X) = star\text{-}n\ (\%n.\ f\ (X\ n))$
by (*simp add*: *starfun-def star-of-def Ifun-star-n*)

lemma *starfun-if-eq*:
$!!w.\ w \neq star\text{-}of\ x$
==> $(*f*\ (\lambda z.\ if\ z = x\ then\ a\ else\ g\ z))\ w = (*f*\ g)\ w$

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 @= *hypreal-of-real* a ==> ($*f*$ (%x. x)) x @= *hypreal-of-real* 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 dest*!: *FreeUltrafilterNat-Compl-mem simp add*: *starfun*, *ultra*)
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* [*simp*]: ($*f*$ *f*) (*star-of a*) = *star-of* (*f a*)
by (*transfer*, *rule refl*)

lemma *starfun-approx*: ($*f*$ *f*) (*star-of a*) @= *hypreal-of-real* (*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*: [| ($*f*$ *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 (*rule hrabs-is-starext-rabs* [*THEN is-starext-starfun-iff* [*THEN iffD1*], *symmetric*])

lemma *starfun-inverse-inverse* [*simp*]: $(\ *f*\ inverse)\ x = inverse(x)$
by (*unfold star-inverse-def*, *rule refl*)

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*:
$!!x.\ (\ *f*\ f)\ x : *s*\ A ==> x : *s*\ \{x.\ f\ x\ \in A\}$
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*:
$abs\ (star\text{-}n\ X) = star\text{-}n\ (\%n.\ abs\ (X\ n))$
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*:
$*s*\ \{x.\ abs\ (x + -\ y) < r\} =$
$\{x.\ abs(x + -hypreal\text{-}of\text{-}real\ y) < hypreal\text{-}of\text{-}real\ r\}$
by (*transfer*, *rule refl*)

lemma *STAR-starfun-rabs-add-minus*:
$*s*\ \{x.\ abs\ (f\ x + -\ y) < r\} =$
$\{x.\ abs((\ *f*\ f)\ x + -hypreal\text{-}of\text{-}real\ y) < hypreal\text{-}of\text{-}real\ r\}$
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*:
$(x \in Infinitesimal) =$
$(\exists X \in Rep\text{-}star(x).$
$\forall m.\ \{n.\ abs(X\ n) < inverse(real(Suc\ m))\}$
$\in\ FreeUltrafilterNat)$
apply (*cases x*)
apply (*auto intro*!: *bexI lemma-starrel-refl*
simp add: *Infinitesimal-hypreal-of-nat-iff star-of-def*
star-n-inverse star-n-abs star-n-less hypreal-of-nat-eq)
apply (*drule-tac x* = *n* **in** *spec*, *ultra*)
done

lemma *approx-FreeUltrafilterNat-iff*: *star-n X* @= *star-n Y* =
$(\forall m.\ \{n.\ abs\ (X\ n + -\ Y\ n) <$
$inverse(real(Suc\ m))\} : FreeUltrafilterNat)$
apply (*subst approx-minus-iff*)
apply (*rule mem-infmal-iff* [*THEN subst*])
apply (*auto simp add*: *star-n-minus star-n-add Infinitesimal-FreeUltrafilterNat-iff2*)
apply (*drule-tac x* = *m* **in** *spec*, *ultra*)
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

ML
⟪
val starset-n-def = *thmstarset-n-def*;
val InternalSets-def = *thmInternalSets-def*;
val is-starext-def = *thmis-starext-def*;
val starfun-n-def = *thmstarfun-n-def*;
val InternalFuns-def = *thmInternalFuns-def*;

val no-choice = *thm no-choice*;
val STAR-UNIV-set = *thm STAR-UNIV-set*;
val STAR-empty-set = *thm STAR-empty-set*;
val STAR-Un = *thm STAR-Un*;
val STAR-Int = *thm STAR-Int*;
val STAR-Compl = *thm STAR-Compl*;
val STAR-mem-Compl = *thm STAR-mem-Compl*;
val STAR-diff = *thm STAR-diff*;
val STAR-subset = *thm STAR-subset*;
val STAR-mem = *thm STAR-mem*;
val STAR-star-of-image-subset = *thm STAR-star-of-image-subset*;

val STAR-hypreal-of-real-Int = thm STAR-hypreal-of-real-Int;
val STAR-real-seq-to-hypreal = thm STAR-real-seq-to-hypreal;
val STAR-singleton = thm STAR-singleton;
val STAR-not-mem = thm STAR-not-mem;
val STAR-subset-closed = thm STAR-subset-closed;
val starset-n-starset = thm starset-n-starset;
val starfun-n-starfun = thm starfun-n-starfun;
val hrabs-is-starext-rabs = thm hrabs-is-starext-rabs;
val Rep-star-FreeUltrafilterNat = thm Rep-star-FreeUltrafilterNat;
val starfun = thm starfun;
val starfun-mult = thm starfun-mult;
val starfun-add = thm starfun-add;
val starfun-minus = thm starfun-minus;
val starfun-add-minus = thm starfun-add-minus;
val starfun-diff = thm starfun-diff;
val starfun-o2 = thm starfun-o2;
val starfun-o = thm starfun-o;
val starfun-const-fun = thm starfun-const-fun;
val starfun-Idfun-approx = thm starfun-Idfun-approx;
val starfun-Id = thm starfun-Id;
val is-starext-starfun = thm is-starext-starfun;
val is-starfun-starext = thm is-starfun-starext;
val is-starext-starfun-iff = thm is-starext-starfun-iff;
val starfun-eq = thm starfun-eq;
val starfun-approx = thm starfun-approx;
val starfun-lambda-cancel = thm starfun-lambda-cancel;
val starfun-lambda-cancel2 = thm starfun-lambda-cancel2;
val starfun-mult-HFinite-approx = thm starfun-mult-HFinite-approx;
val starfun-add-approx = thm starfun-add-approx;
val starfun-rabs-hrabs = thm starfun-rabs-hrabs;
val starfun-inverse-inverse = thm starfun-inverse-inverse;
val starfun-inverse = thm starfun-inverse;
val starfun-divide = thm starfun-divide;
val starfun-inverse2 = thm starfun-inverse2;
val starfun-mem-starset = thm starfun-mem-starset;
val hypreal-hrabs = thm hypreal-hrabs;
val STAR-rabs-add-minus = thm STAR-rabs-add-minus;
val STAR-starfun-rabs-add-minus = thm STAR-starfun-rabs-add-minus;
val Infinitesimal-FreeUltrafilterNat-iff2 = thm Infinitesimal-FreeUltrafilterNat-iff2;
val approx-FreeUltrafilterNat-iff = thm approx-FreeUltrafilterNat-iff;
val inj-starfun = thm inj-starfun;
⟩⟩

end

15 HyperNat: Hypernatural numbers

theory *HyperNat*

imports *Star*
begin

types *hypnat* = *nat star*

syntax *hypnat-of-nat* :: *nat* => *nat star*
translations *hypnat-of-nat* => *star-of* :: *nat* => *nat star*

15.1 Properties Transferred from Naturals

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* ≤ (*0*::*hypnat*)) = (*n* = *0*)

by *transfer* (*rule le-0-eq*)

lemma *hypnat-mult-is-0* [*simp*]: !!*m n*. (*m*∗*n* = (*0*::*hypnat*)) = (*m*=*0* | *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-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

15.2 Properties of the set of embedded natural numbers

lemma *hypnat-of-nat-def*: *hypnat-of-nat m* == *of-nat m*
by (*transfer*, *simp*)

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

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

lemma *of-nat-eq-add* [*rule-format*]:
  $\forall$ *d*::*hypnat*. *of-nat m* = *of-nat n* + *d* $\longrightarrow$ *d* $\in$ *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* $\in$ *Nats*; *b* $\in$ *Nats*|] ==> (*a*−*b* :: *hypnat*) $\in$ *Nats*
by (*auto simp add*: *of-nat-eq-add Nats-def split*: *hypnat-diff-split*)

15.3 Existence of an infinite hypernatural number

consts *whn* :: *hypnat*

defs

hypnat-omega-def: *whn* == *star-n* (%*n*::*nat*. *n*)

Existence of infinite number not corresponding to any natural number follows because member $\mathcal{U}$ is not finite. See *HyperDef.thy* for similar argument.

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-Compl-mem*, *ultra*)
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 (*force simp add*: *hypnat-of-nat-def Nats-def*)

lemma *hypnat-omega-gt-SHNat*:
$n \in Nats \Longrightarrow n < whn$
by (*auto simp add*: *hypnat-of-nat-eq star-n-less hypnat-omega-def SHNat-eq*)

lemma *SHNAT-omega-not-mem* [*simp*]: $whn \notin Nats$
by (*blast dest*: *hypnat-omega-gt-SHNat*)

lemma *hypnat-of-nat-less-whn* [*simp*]: $hypnat\text{-}of\text{-}nat\ n < whn$
apply (*insert hypnat-omega-gt-SHNat* [*of hypnat-of-nat n*])
apply (*simp add*: *hypnat-of-nat-def*)
done

lemma *hypnat-of-nat-le-whn* [*simp*]: $hypnat\text{-}of\text{-}nat\ n \leq whn$
by (*rule hypnat-of-nat-less-whn* [*THEN order-less-imp-le*])

lemma *hypnat-zero-less-hypnat-omega* [*simp*]: $0 < whn$
by (*simp add*: *hypnat-omega-gt-SHNat*)

lemma *hypnat-one-less-hypnat-omega* [*simp*]: $(1::hypnat) < whn$
by (*simp add*: *hypnat-omega-gt-SHNat*)

15.4 Infinite Hypernatural Numbers − *HNatInfinite*

constdefs

HNatInfinite :: *hypnat set*
$HNatInfinite == \{n.\ n \notin Nats\}$

lemma *HNatInfinite-whn* [*simp*]: $whn \in HNatInfinite$
by (*simp add*: *HNatInfinite-def*)

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*)

15.4.1 Alternative characterization of the set of infinite hypernaturals

$HNatInfinite = \{N.\ \forall\, n{\in}\mathbb{N}.\ n < N\}$

lemma *HNatInfinite-FreeUltrafilterNat-lemma*:
$\forall\, N{::}nat.\ \{n.\ f\ n \neq N\} \in FreeUltrafilterNat$
$\Longrightarrow \{n.\ N < f\ n\} \in FreeUltrafilterNat$
apply (*induct-tac N*)
apply (*drule-tac x = 0* **in** *spec*)
apply (*rule ccontr*, *drule FreeUltrafilterNat-Compl-mem*, *drule FreeUltrafilterNat-Int*, *assumption*, *simp*)
apply (*drule-tac x = Suc n* **in** *spec*, *ultra*)
done

lemma *HNatInfinite-iff*: $HNatInfinite = \{N.\ \forall\, n \in Nats.\ n < N\}$
apply (*auto simp add*: *HNatInfinite-def SHNat-eq hypnat-of-nat-eq*)
apply (*rule-tac x = x* **in** *star-cases*)
apply (*auto elim*: *HNatInfinite-FreeUltrafilterNat-lemma*
simp add: *star-n-less FreeUltrafilterNat-Compl-iff1*
star-n-eq-iff Collect-neg-eq [*symmetric*])
done

15.4.2 Alternative Characterization of *HNatInfinite* using Free Ultrafilter

lemma *HNatInfinite-FreeUltrafilterNat*:
$x \in HNatInfinite$
$\Longrightarrow \exists\, X \in Rep\text{-}star\ x.\ \forall\, u.\ \{n.\ u < X\ n\}:\ FreeUltrafilterNat$
apply (*cases x*)
apply (*auto simp add*: *HNatInfinite-iff SHNat-eq hypnat-of-nat-eq*)
apply (*rule bexI* [*OF - Rep-star-star-n*], *clarify*)
apply (*auto simp add*: *hypnat-of-nat-def star-n-less*)
done

lemma *FreeUltrafilterNat-HNatInfinite*:
$\exists\, X \in Rep\text{-}star\ x.\ \forall\, u.\ \{n.\ u < X\ n\}:\ FreeUltrafilterNat$
$\Longrightarrow x \in HNatInfinite$
apply (*cases x*)
apply (*auto simp add*: *star-n-less HNatInfinite-iff SHNat-eq hypnat-of-nat-eq*)
apply (*drule spec*, *ultra*, *auto*)
done

lemma *HNatInfinite-FreeUltrafilterNat-iff*:

$(x \in HNatInfinite) =$
$(\exists X \in Rep\text{-}star\ x.\ \forall u.\ \{n.\ u < X\ n\}:\ FreeUltrafilterNat)$
by (*blast intro*: *HNatInfinite-FreeUltrafilterNat*
FreeUltrafilterNat-HNatInfinite)

lemma *HNatInfinite-gt-one* [*simp*]: $x \in HNatInfinite \Longrightarrow (1::hypnat) < x$
by (*auto simp add*: *HNatInfinite-iff*)

lemma *zero-not-mem-HNatInfinite* [*simp*]: $0 \notin HNatInfinite$
apply (*auto simp add*: *HNatInfinite-iff*)
apply (*drule-tac a* = $(1::hypnat)$ **in** *equals0D*)
apply *simp*
done

lemma *HNatInfinite-not-eq-zero*: $x \in HNatInfinite \Longrightarrow 0 < x$
apply (*drule HNatInfinite-gt-one*)
apply (*auto simp add*: *order-less-trans* [*OF zero-less-one*])
done

lemma *HNatInfinite-ge-one* [*simp*]: $x \in HNatInfinite \Longrightarrow (1::hypnat) \leq x$
by (*blast intro*: *order-less-imp-le HNatInfinite-gt-one*)

15.4.3 Closure Rules

lemma *HNatInfinite-add*:
$[|\ x \in HNatInfinite; y \in HNatInfinite\ |] \Longrightarrow x + y \in HNatInfinite$
apply (*auto simp add*: *HNatInfinite-iff*)
apply (*drule bspec*, *assumption*)
apply (*drule bspec* [*OF - Nats-0*])
apply (*drule add-strict-mono*, *assumption*, *simp*)
done

lemma *HNatInfinite-SHNat-add*:
$[|\ x \in HNatInfinite; y \in Nats\ |] \Longrightarrow x + y \in HNatInfinite$
apply (*auto simp add*: *HNatInfinite-not-Nats-iff*)
apply (*drule-tac a* = $x + y$ **in** *Nats-diff*, *auto*)
done

lemma *HNatInfinite-Nats-imp-less*: $[|\ x \in HNatInfinite; y \in Nats\ |] \Longrightarrow y < x$
by (*simp add*: *HNatInfinite-iff*)

lemma *HNatInfinite-SHNat-diff*:
assumes *x*: $x \in HNatInfinite$ **and** *y*: $y \in Nats$
shows $x - y \in HNatInfinite$
proof −
have $y < x$ **by** (*simp add*: *HNatInfinite-Nats-imp-less prems*)
hence $x - y + y = x$ **by** (*simp add*: *order-less-imp-le*)
with *x* **show** *?thesis*
by (*force simp add*: *HNatInfinite-not-Nats-iff*

dest: *Nats-add* [*of* $x-y$, *OF* - y])
qed

lemma *HNatInfinite-add-one*:
$x \in HNatInfinite \Longrightarrow x + (1::hypnat) \in HNatInfinite$
by (*auto intro*: *HNatInfinite-SHNat-add*)

lemma *HNatInfinite-is-Suc*: $x \in HNatInfinite \Longrightarrow \exists y.\ x = y + (1::hypnat)$
apply (*rule-tac* $x = x - (1::hypnat)$ **in** *exI*)
apply *auto*
done

15.5 Embedding of the Hypernaturals into the Hyperreals

Obtained using the nonstandard extension of the naturals

constdefs
hypreal-of-hypnat :: *hypnat* => *hypreal*
hypreal-of-hypnat == $*f*$ *real*

declare *hypreal-of-hypnat-def* [*transfer-unfold*]

lemma *HNat-hypreal-of-nat* [*simp*]: *hypreal-of-nat N* $\in$ *Nats*
by (*simp add*: *hypreal-of-nat-def*)

lemma *hypreal-of-hypnat*:
hypreal-of-hypnat (*star-n X*) = *star-n* (%n. *real* (*X n*))
by (*simp add*: *hypreal-of-hypnat-def starfun*)

lemma *hypreal-of-hypnat-inject* [*simp*]:
!!m n. (*hypreal-of-hypnat* m = *hypreal-of-hypnat* n) = (m=n)
by (*transfer*, *simp*)

lemma *hypreal-of-hypnat-zero*: *hypreal-of-hypnat 0* = *0*
by (*simp add*: *star-n-zero-num hypreal-of-hypnat*)

lemma *hypreal-of-hypnat-one*: *hypreal-of-hypnat* (*1*::*hypnat*) = *1*
by (*simp add*: *star-n-one-num hypreal-of-hypnat*)

lemma *hypreal-of-hypnat-add* [*simp*]:
!!m n. *hypreal-of-hypnat* (m + n) = *hypreal-of-hypnat* m + *hypreal-of-hypnat* n
by (*transfer*, *rule real-of-nat-add*)

lemma *hypreal-of-hypnat-mult* [*simp*]:
!!m n. *hypreal-of-hypnat* (m * n) = *hypreal-of-hypnat* m * *hypreal-of-hypnat* n
by (*transfer*, *rule real-of-nat-mult*)

lemma *hypreal-of-hypnat-less-iff* [*simp*]:
!!m n. (*hypreal-of-hypnat* n < *hypreal-of-hypnat* m) = (n < m)

by (*transfer*, *simp*)

lemma *hypreal-of-hypnat-eq-zero-iff*: (*hypreal-of-hypnat N = 0*) = (*N = 0*)
by (*simp add*: *hypreal-of-hypnat-zero* [*symmetric*])
declare *hypreal-of-hypnat-eq-zero-iff* [*simp*]

lemma *hypreal-of-hypnat-ge-zero* [*simp*]: !!*n*. *0 ≤ hypreal-of-hypnat n*
by (*transfer*, *simp*)

lemma *HNatInfinite-inverse-Infinitesimal* [*simp*]:
n ∈ HNatInfinite ==> inverse (*hypreal-of-hypnat n*) *∈ Infinitesimal*
apply (*cases n*)
apply (*auto simp add*: *hypreal-of-hypnat star-n-inverse*
HNatInfinite-FreeUltrafilterNat-iff Infinitesimal-FreeUltrafilterNat-iff2)
apply (*rule bexI* [*OF - Rep-star-star-n*], *auto*)
apply (*drule-tac x = m + 1* **in** *spec*, *ultra*)
done

lemma *HNatInfinite-hypreal-of-hypnat-gt-zero*:
N ∈ HNatInfinite ==> 0 < hypreal-of-hypnat N
apply (*rule ccontr*)
apply (*simp add*: *hypreal-of-hypnat-zero* [*symmetric*] *linorder-not-less*)
done

ML
⟪
val hypnat-of-nat-def = thmhypnat-of-nat-def;
val HNatInfinite-def = thmHNatInfinite-def;
val hypreal-of-hypnat-def = thmhypreal-of-hypnat-def;
val hypnat-omega-def = thmhypnat-omega-def;

val starrel-iff = thm starrel-iff;
val lemma-starrel-refl = thm lemma-starrel-refl;
val hypnat-minus-zero = thm hypnat-minus-zero;
val hypnat-diff-0-eq-0 = thm hypnat-diff-0-eq-0;
val hypnat-add-is-0 = thm hypnat-add-is-0;
val hypnat-diff-diff-left = thm hypnat-diff-diff-left;
val hypnat-diff-commute = thm hypnat-diff-commute;
val hypnat-diff-add-inverse = thm hypnat-diff-add-inverse;
val hypnat-diff-add-inverse2 = thm hypnat-diff-add-inverse2;
val hypnat-diff-cancel = thm hypnat-diff-cancel;
val hypnat-diff-cancel2 = thm hypnat-diff-cancel2;
val hypnat-diff-add-0 = thm hypnat-diff-add-0;
val hypnat-diff-mult-distrib = thm hypnat-diff-mult-distrib;
val hypnat-diff-mult-distrib2 = thm hypnat-diff-mult-distrib2;
val hypnat-mult-is-0 = thm hypnat-mult-is-0;
val hypnat-not-less0 = thm hypnat-not-less0;
val hypnat-less-one = thm hypnat-less-one;

val hypnat-add-diff-inverse = thm hypnat-add-diff-inverse;
val hypnat-le-add-diff-inverse = thm hypnat-le-add-diff-inverse;
val hypnat-le-add-diff-inverse2 = thm hypnat-le-add-diff-inverse2;
val hypnat-le0 = thm hypnat-le0;
val hypnat-add-self-le = thm hypnat-add-self-le;
val hypnat-add-one-self-less = thm hypnat-add-one-self-less;
val hypnat-neq0-conv = thm hypnat-neq0-conv;
val hypnat-gt-zero-iff = thm hypnat-gt-zero-iff;
val hypnat-gt-zero-iff2 = thm hypnat-gt-zero-iff2;
val SHNat-eq = thmSHNat-eq
val hypnat-of-nat-one = thm hypnat-of-nat-one;
val hypnat-of-nat-Suc = thm hypnat-of-nat-Suc;
val SHNAT-omega-not-mem = thm SHNAT-omega-not-mem;
val cofinite-mem-FreeUltrafilterNat = thm cofinite-mem-FreeUltrafilterNat;
val hypnat-omega-gt-SHNat = thm hypnat-omega-gt-SHNat;
val hypnat-of-nat-less-whn = thm hypnat-of-nat-less-whn;
val hypnat-of-nat-le-whn = thm hypnat-of-nat-le-whn;
val hypnat-zero-less-hypnat-omega = thm hypnat-zero-less-hypnat-omega;
val hypnat-one-less-hypnat-omega = thm hypnat-one-less-hypnat-omega;
val HNatInfinite-whn = thm HNatInfinite-whn;
val HNatInfinite-iff = thm HNatInfinite-iff;
val HNatInfinite-FreeUltrafilterNat = thm HNatInfinite-FreeUltrafilterNat;
val FreeUltrafilterNat-HNatInfinite = thm FreeUltrafilterNat-HNatInfinite;
val HNatInfinite-FreeUltrafilterNat-iff = thm HNatInfinite-FreeUltrafilterNat-iff;
val HNatInfinite-gt-one = thm HNatInfinite-gt-one;
val zero-not-mem-HNatInfinite = thm zero-not-mem-HNatInfinite;
val HNatInfinite-not-eq-zero = thm HNatInfinite-not-eq-zero;
val HNatInfinite-ge-one = thm HNatInfinite-ge-one;
val HNatInfinite-add = thm HNatInfinite-add;
val HNatInfinite-SHNat-add = thm HNatInfinite-SHNat-add;
val HNatInfinite-SHNat-diff = thm HNatInfinite-SHNat-diff;
val HNatInfinite-add-one = thm HNatInfinite-add-one;
val HNatInfinite-is-Suc = thm HNatInfinite-is-Suc;
val HNat-hypreal-of-nat = thm HNat-hypreal-of-nat;
val hypreal-of-hypnat = thm hypreal-of-hypnat;
val hypreal-of-hypnat-zero = thm hypreal-of-hypnat-zero;
val hypreal-of-hypnat-one = thm hypreal-of-hypnat-one;
val hypreal-of-hypnat-add = thm hypreal-of-hypnat-add;
val hypreal-of-hypnat-mult = thm hypreal-of-hypnat-mult;
val hypreal-of-hypnat-less-iff = thm hypreal-of-hypnat-less-iff;
val hypreal-of-hypnat-ge-zero = thm hypreal-of-hypnat-ge-zero;
val HNatInfinite-inverse-Infinitesimal = thm HNatInfinite-inverse-Infinitesimal;
⟩

end

16 HyperPow: Exponentials on the Hyperreals

theory *HyperPow*
imports *HyperArith HyperNat*
begin

lemma *hpowr-0* [*simp*]: $r \hat{} 0 = (1::hypreal)$
by (*rule power-0*)

lemma *hpowr-Suc* [*simp*]: $r \hat{} (Suc\ n) = (r::hypreal) * (r \hat{} n)$
by (*rule power-Suc*)

consts
pow :: [*hypreal*,*hypnat*] => *hypreal* (**infixr** *pow* 80)

defs

hyperpow-def [*transfer-unfold*]:
$(R::hypreal)\ pow\ (N::hypnat) == (\ *f2*\ op\ \hat{})\ R\ N$

lemma *hrealpow-two*: $(r::hypreal) \hat{} Suc\ (Suc\ 0) = r * r$
by *simp*

lemma *hrealpow-two-le* [*simp*]: $(0::hypreal) \leq r \hat{} Suc\ (Suc\ 0)$
by (*auto simp add*: *zero-le-mult-iff*)

lemma *hrealpow-two-le-add-order* [*simp*]:
$(0::hypreal) \leq u \hat{} Suc\ (Suc\ 0) + v \hat{} Suc\ (Suc\ 0)$
by (*simp only*: *hrealpow-two-le add-nonneg-nonneg*)

lemma *hrealpow-two-le-add-order2* [*simp*]:
$(0::hypreal) \leq u \hat{} Suc\ (Suc\ 0) + v \hat{} Suc\ (Suc\ 0) + w \hat{} Suc\ (Suc\ 0)$
by (*simp only*: *hrealpow-two-le add-nonneg-nonneg*)

lemma *hypreal-add-nonneg-eq-0-iff*:
$[|\ 0 \leq x;\ 0 \leq y\ |] ==> (x+y = 0) = (x = 0\ \&\ y = (0::hypreal))$
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*]:

$(x \hat{} Suc\ (Suc\ 0) + y \hat{} Suc\ (Suc\ 0) + z \hat{} Suc\ (Suc\ 0) = (0::hypreal)) =$
$(x = 0\ \&\ y = 0\ \&\ z = 0)$
by (*simp only*: *hypreal-three-squares-add-zero-iff hrealpow-two*)

lemma *hrabs-hrealpow-two* [*simp*]:
$abs(x \hat{} Suc\ (Suc\ 0)) = (x::hypreal) \hat{} Suc\ (Suc\ 0)$
by (*simp add*: *abs-mult*)

lemma *two-hrealpow-ge-one* [*simp*]: $(1::hypreal) \leq 2 \hat{} n$
by (*insert power-increasing* [*of 0 n 2*::*hypreal*], *simp*)

lemma *two-hrealpow-gt* [*simp*]: *hypreal-of-nat n* $< 2 \hat{} n$
apply (*induct-tac n*)
apply (*auto simp add*: *hypreal-of-nat-Suc left-distrib*)
apply (*cut-tac n* = *n* **in** *two-hrealpow-ge-one*, *arith*)
done

lemma *hrealpow*:
$star\text{-}n\ X \hat{} m = star\text{-}n\ (\%n.\ (X\ n::real) \hat{} m)$
apply (*induct-tac m*)
apply (*auto simp add*: *star-n-one-num star-n-mult power-0*)
done

lemma *hrealpow-sum-square-expand*:
$(x + (y::hypreal)) \hat{} Suc\ (Suc\ 0) =$
$x \hat{} Suc\ (Suc\ 0) + y \hat{} Suc\ (Suc\ 0) + (hypreal\text{-}of\text{-}nat\ (Suc\ (Suc\ 0)))*x*y$
by (*simp add*: *right-distrib left-distrib hypreal-of-nat-Suc*)

16.1 Literal Arithmetic Involving Powers and Type *hypreal*

lemma *power-hypreal-of-real-number-of*:
$(number\text{-}of\ v :: hypreal) \hat{} n = hypreal\text{-}of\text{-}real\ ((number\text{-}of\ v) \hat{} n)$
by *simp*

declare *power-hypreal-of-real-number-of* [*of - number-of w*, *standard*, *simp*]

lemma *hrealpow-HFinite*: $x \in HFinite \Longrightarrow x \hat{} n \in HFinite$
apply (*induct-tac n*)
apply (*auto intro*: *HFinite-mult*)
done

16.2 Powers with Hypernatural Exponents

lemma *hyperpow*: *star-n X pow star-n Y* = *star-n* $(\%n.\ X\ n \hat{} Y\ n)$
by (*simp add*: *hyperpow-def starfun2-star-n*)

lemma *hyperpow-zero* [*simp*]: $!!n.\ (0::hypreal)\ pow\ (n + (1::hypnat)) = 0$
by (*transfer*, *simp*)

lemma *hyperpow-not-zero*: !!*r n*. *r* ≠ (*0*::*hypreal*) ==> *r pow n* ≠ *0*
by (*transfer*, *simp*)

lemma *hyperpow-inverse*:
!!*r n*. *r* ≠ (*0*::*hypreal*) ==> *inverse*(*r pow n*) = (*inverse r*) *pow n*
by (*transfer*, *rule power-inverse*)

lemma *hyperpow-hrabs*: !!*r n*. *abs r pow n* = *abs* (*r pow n*)
by (*transfer*, *rule power-abs* [*symmetric*])

lemma *hyperpow-add*: !!*r n m*. *r pow* (*n* + *m*) = (*r pow n*) ∗ (*r pow m*)
by (*transfer*, *rule power-add*)

lemma *hyperpow-one* [*simp*]: !!*r*. *r pow* (*1*::*hypnat*) = *r*
by (*transfer*, *simp*)

lemma *hyperpow-two*:
!!*r*. *r pow* ((*1*::*hypnat*) + (*1*::*hypnat*)) = *r* ∗ *r*
by (*transfer*, *simp*)

lemma *hyperpow-gt-zero*: !!*r n*. (*0*::*hypreal*) < *r* ==> *0* < *r pow n*
by (*transfer*, *rule zero-less-power*)

lemma *hyperpow-ge-zero*: !!*r n*. (*0*::*hypreal*) ≤ *r* ==> *0* ≤ *r pow n*
by (*transfer*, *rule zero-le-power*)

lemma *hyperpow-le*:
!!*x y n*. [|(*0*::*hypreal*) < *x*; *x* ≤ *y*|] ==> *x pow n* ≤ *y pow n*
by (*transfer*, *rule power-mono* [*OF - order-less-imp-le*])

lemma *hyperpow-eq-one* [*simp*]: !!*n*. *1 pow n* = (*1*::*hypreal*)
by (*transfer*, *simp*)

lemma *hrabs-hyperpow-minus-one* [*simp*]: !!*n*. *abs*(−*1 pow n*) = (*1*::*hypreal*)
by (*transfer*, *simp*)

lemma *hyperpow-mult*: !!*r s n*. (*r* ∗ *s*) *pow n* = (*r pow n*) ∗ (*s pow n*)
by (*transfer*, *rule power-mult-distrib*)

lemma *hyperpow-two-le* [*simp*]: *0* ≤ *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 pow* (*1* + *1*)
by (*simp add*: *abs-if hyperpow-two-le linorder-not-less*)

lemma *hyperpow-two-hrabs* [*simp*]: *abs*(*x*) *pow* (*1* + *1*) = *x pow* (*1* + *1*)
by (*simp add*: *hyperpow-hrabs*)

The precondition could be weakened to (*0*::′*a*) ≤ *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*: *1* < *r* ==> *1* < *r pow* (*1* + *1*)
apply (*auto simp add*: *hyperpow-two*)
apply (*rule-tac y* = *1*∗*1* **in** *order-le-less-trans*)
apply (*rule-tac* [*2*] *hypreal-mult-less-mono*, *auto*)
done

lemma *hyperpow-two-ge-one*:
1 ≤ *r* ==> *1* ≤ *r pow* (*1* + *1*)
by (*auto dest*!: *order-le-imp-less-or-eq intro*: *hyperpow-two-gt-one order-less-imp-le*)

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*, *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 (*simp add*: *star-of-def hypnat-of-nat-eq hyperpow*)

lemma *hyperpow-SReal* [*simp*]:
(*hypreal-of-real r*) *pow* (*hypnat-of-nat n*) ∈ *Reals*
by (*simp del*: *star-of-power add*: *hyperpow-realpow SReal-def*)

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 *lemma-Infinitesimal-hyperpow*:
[| *x* ∈ *Infinitesimal*; *0* < *N* |] ==> *abs* (*x pow N*) ≤ *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* ∈ *Infinitesimal*; *0* < *N* |] ==> *x pow N* ∈ *Infinitesimal*
apply (*rule hrabs-le-Infinitesimal*)
apply (*rule-tac* [*2*] *lemma-Infinitesimal-hyperpow*, *auto*)
done

lemma *hrealpow-hyperpow-Infinitesimal-iff*:
(*x* ^ *n* ∈ *Infinitesimal*) = (*x pow* (*hypnat-of-nat n*) ∈ *Infinitesimal*)
apply (*cases x*)
apply (*simp add*: *hrealpow hyperpow hypnat-of-nat-eq*)
done

lemma *Infinitesimal-hrealpow*:
[| *x* ∈ *Infinitesimal*; *0* < *n* |] ==> *x* ^ *n* ∈ *Infinitesimal*
by (*simp add*: *hrealpow-hyperpow-Infinitesimal-iff Infinitesimal-hyperpow*)

ML
⟪
val hrealpow-two = *thm hrealpow-two*;
val hrealpow-two-le = *thm hrealpow-two-le*;
val hrealpow-two-le-add-order = *thm hrealpow-two-le-add-order*;
val hrealpow-two-le-add-order2 = *thm hrealpow-two-le-add-order2*;
val hypreal-add-nonneg-eq-0-iff = *thm hypreal-add-nonneg-eq-0-iff*;
val hypreal-three-squares-add-zero-iff = *thm hypreal-three-squares-add-zero-iff*;
val hrealpow-three-squares-add-zero-iff = *thm hrealpow-three-squares-add-zero-iff*;
val hrabs-hrealpow-two = *thm hrabs-hrealpow-two*;
val two-hrealpow-ge-one = *thm two-hrealpow-ge-one*;
val two-hrealpow-gt = *thm two-hrealpow-gt*;
val hrealpow = *thm hrealpow*;
val hrealpow-sum-square-expand = *thm hrealpow-sum-square-expand*;
val power-hypreal-of-real-number-of = *thm power-hypreal-of-real-number-of*;
val hrealpow-HFinite = *thm hrealpow-HFinite*;
val hyperpow = *thm hyperpow*;
val hyperpow-zero = *thm hyperpow-zero*;

val hyperpow-not-zero = thm hyperpow-not-zero;
val hyperpow-inverse = thm hyperpow-inverse;
val hyperpow-hrabs = thm hyperpow-hrabs;
val hyperpow-add = thm hyperpow-add;
val hyperpow-one = thm hyperpow-one;
val hyperpow-two = thm hyperpow-two;
val hyperpow-gt-zero = thm hyperpow-gt-zero;
val hyperpow-ge-zero = thm hyperpow-ge-zero;
val hyperpow-le = thm hyperpow-le;
val hyperpow-eq-one = thm hyperpow-eq-one;
val hrabs-hyperpow-minus-one = thm hrabs-hyperpow-minus-one;
val hyperpow-mult = thm hyperpow-mult;
val hyperpow-two-le = thm hyperpow-two-le;
val hrabs-hyperpow-two = thm hrabs-hyperpow-two;
val hyperpow-two-hrabs = thm hyperpow-two-hrabs;
val hyperpow-two-gt-one = thm hyperpow-two-gt-one;
val hyperpow-two-ge-one = thm hyperpow-two-ge-one;
val two-hyperpow-ge-one = thm two-hyperpow-ge-one;
val hyperpow-minus-one2 = thm hyperpow-minus-one2;
val hyperpow-less-le = thm hyperpow-less-le;
val hyperpow-SHNat-le = thm hyperpow-SHNat-le;
val hyperpow-realpow = thm hyperpow-realpow;
val hyperpow-SReal = thm hyperpow-SReal;
val hyperpow-zero-HNatInfinite = thm hyperpow-zero-HNatInfinite;
val hyperpow-le-le = thm hyperpow-le-le;
val hyperpow-Suc-le-self2 = thm hyperpow-Suc-le-self2;
val lemma-Infinitesimal-hyperpow = thm lemma-Infinitesimal-hyperpow;
val Infinitesimal-hyperpow = thm Infinitesimal-hyperpow;
val hrealpow-hyperpow-Infinitesimal-iff = thm hrealpow-hyperpow-Infinitesimal-iff;
val Infinitesimal-hrealpow = thm Infinitesimal-hrealpow;
⟩⟩

end

17 NatStar: Star-transforms for the Hypernaturals

theory *NatStar*
imports *HyperPow*
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*:
[| $X \in$ *InternalSets*; $Y \in$ *InternalSets* |]
==> $(X\ Un\ Y) \in$ *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*:
[| $X \in$ *InternalSets*; $Y \in$ *InternalSets* |]
==> $(X\ Int\ Y) \in$ *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-def*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-Compl*: $X \in$ *InternalSets* ==> $-X \in$ *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-def*)
apply (*rule-tac x*=*whn* **in** *spec*, *transfer*, *simp*)
done

lemma *InternalSets-diff*:
[| $X \in$ *InternalSets*; $Y \in$ *InternalSets* |]
==> $(X\ -\ Y) \in$ *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 STAR-mem-iff*)

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*:
$X \in \mathit{InternalSets} \Longrightarrow \mathit{UNIV} - X \in \mathit{InternalSets}$
apply (*subgoal-tac* $\mathit{UNIV} - X = - X$)
by (*auto intro*: *InternalSets-Compl*)

17.1 Nonstandard Extensions of Functions

Example of transfer of a property from reals to hyperreals — used for limit comparison of sequences

lemma *starfun-le-mono*:
$\forall n.\ N \leq n \longrightarrow f\ n \leq g\ n$
$\Longrightarrow \forall n.\ \mathit{hypnat\text{-}of\text{-}nat}\ N \leq n \longrightarrow (\ {*f*}\ f)\ n \leq (\ {*f*}\ g)\ n$
by *transfer*

lemma *starfun-less-mono*:
$\forall n.\ N \leq n \longrightarrow f\ n < g\ n$
$\Longrightarrow \forall n.\ \mathit{hypnat\text{-}of\text{-}nat}\ N \leq n \longrightarrow (\ {*f*}\ f)\ n < (\ {*f*}\ g)\ n$
by *transfer*

Nonstandard extension when we increment the argument by one

lemma *starfun-shift-one*:
$!!N.\ (\ {*f*}\ (\%n.\ f\ (\mathit{Suc}\ n)))\ N = (\ {*f*}\ f)\ (N + (1{::}\mathit{hypnat}))$
by (*transfer*, *simp*)

Nonstandard extension with absolute value

lemma *starfun-abs*: $!!N.\ (\ {*f*}\ (\%n.\ \mathit{abs}\ (f\ n)))\ N = \mathit{abs}((\ {*f*}\ f)\ N)$
by (*transfer*, *rule refl*)

The hyperpow function as a nonstandard extension of realpow

lemma *starfun-pow*: $!!N.\ (\ {*f*}\ (\%n.\ r\ \hat{}\ n))\ N = (\mathit{hypreal\text{-}of\text{-}real}\ r)\ \mathit{pow}\ N$
by (*transfer*, *rule refl*)

lemma *starfun-pow2*:
$!!N.\ (\ {*f*}\ (\%n.\ (X\ n)\ \hat{}\ m))\ N = (\ {*f*}\ X)\ N\ \mathit{pow}\ \mathit{hypnat\text{-}of\text{-}nat}\ m$
by (*transfer*, *rule refl*)

lemma *starfun-pow3*: $!!R.\ (\ {*f*}\ (\%r.\ r\ \hat{}\ n))\ R = (R)\ \mathit{pow}\ \mathit{hypnat\text{-}of\text{-}nat}\ n$
by (*transfer*, *rule refl*)

The *hypreal-of-hypnat* function as a nonstandard extension of *real-of-nat*

lemma *starfunNat-real-of-nat*: $(\ {*f*}\ \mathit{real}) = \mathit{hypreal\text{-}of\text{-}hypnat}$
by (*transfer*, *rule refl*)

lemma *starfun-inverse-real-of-nat-eq*:
$N \in \mathit{HNatInfinite}$
$\Longrightarrow (\ {*f*}\ (\%x{::}\mathit{nat}.\ \mathit{inverse}(\mathit{real}\ x)))\ N = \mathit{inverse}(\mathit{hypreal\text{-}of\text{-}hypnat}\ N)$
apply (*rule-tac f1* $= \mathit{inverse}$ **in** *starfun-o2* [*THEN subst*])

apply (*subgoal-tac hypreal-of-hypnat N ~= 0*)
apply (*simp-all add*: *HNatInfinite-not-eq-zero starfunNat-real-of-nat starfun-inverse-inverse*)
done

Internal functions - some redundancy with *f* now

lemma *starfun-n*: (*fn* *) (*star-n X*) = *star-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-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-of n*) = *star-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 ∈ *HNatInfinite* ==> (∗*f*∗ (%*x*. *inverse* (*real x*))) *N* ∈ *Infinitesimal*
apply (*rule-tac f1* = *inverse* **in** *starfun-o2* [*THEN subst*])
apply (*subgoal-tac hypreal-of-hypnat N* ~= *0*)
apply (*simp-all add*: *HNatInfinite-not-eq-zero starfunNat-real-of-nat*)
done

ML
⟪
val starset-n-Un = *thm starset-n-Un*;
val InternalSets-Un = *thm InternalSets-Un*;
val starset-n-Int = *thm starset-n-Int*;
val InternalSets-Int = *thm InternalSets-Int*;
val starset-n-Compl = *thm starset-n-Compl*;
val InternalSets-Compl = *thm InternalSets-Compl*;
val starset-n-diff = *thm starset-n-diff*;
val InternalSets-diff = *thm InternalSets-diff*;
val NatStar-SHNat-subset = *thm NatStar-SHNat-subset*;
val NatStar-hypreal-of-real-Int = *thm NatStar-hypreal-of-real-Int*;
val starset-starset-n-eq = *thm starset-starset-n-eq*;
val InternalSets-starset-n = *thm InternalSets-starset-n*;
val InternalSets-UNIV-diff = *thm InternalSets-UNIV-diff*;
val starset-n-starset = *thm starset-n-starset*;
val starfun-const-fun = *thm starfun-const-fun*;
val starfun-le-mono = *thm starfun-le-mono*;
val starfun-less-mono = *thm starfun-less-mono*;
val starfun-shift-one = *thm starfun-shift-one*;
val starfun-abs = *thm starfun-abs*;
val starfun-pow = *thm starfun-pow*;
val starfun-pow2 = *thm starfun-pow2*;
val starfun-pow3 = *thm starfun-pow3*;
val starfunNat-real-of-nat = *thm starfunNat-real-of-nat*;
val starfun-inverse-real-of-nat-eq = *thm starfun-inverse-real-of-nat-eq*;
val starfun-n = *thm starfun-n*;
val starfun-n-mult = *thm starfun-n-mult*;
val starfun-n-add = *thm starfun-n-add*;
val starfun-n-add-minus = *thm starfun-n-add-minus*;
val starfun-n-const-fun = *thm starfun-n-const-fun*;
val starfun-n-minus = *thm starfun-n-minus*;
val starfun-n-eq = *thm starfun-n-eq*;
val starfun-eq-iff = *thm starfun-eq-iff*;
val starfunNat-inverse-real-of-nat-Infinitesimal = *thm starfunNat-inverse-real-of-nat-Infinitesimal*;
⟫

17.2 Nonstandard Characterization of Induction

constdefs
hSuc :: *hypnat* => *hypnat*
hSuc n == *n* + *1*

lemma *starP*: $((*p*\ P)\ (star\text{-}n\ X)) = (\{n.\ P\ (X\ n)\} \in FreeUltrafilterNat)$
by (*rule starP-star-n*)

lemma *hypnat-induct-obj*:
$!!n.\ ((*p*\ P)\ (0::hypnat)\ \&$
$(\forall n.\ (*p*\ P)(n) \longrightarrow (*p*\ P)(n + 1)))$
$\longrightarrow (*p*\ P)(n)$
by (*transfer*, *induct-tac n*, *auto*)

lemma *hypnat-induct*:
$!!n.\ [|\ (*p*\ P)\ (0::hypnat);$
$!!n.\ (*p*\ P)(n) \Longrightarrow (*p*\ P)(n + 1)|]$
$\Longrightarrow (*p*\ P)(n)$
by (*transfer*, *induct-tac n*, *auto*)

lemma *starP2*:
$((*p2*\ P)\ (star\text{-}n\ X)\ (star\text{-}n\ Y)) =$
$(\{n.\ P\ (X\ n)\ (Y\ n)\} \in FreeUltrafilterNat)$
by (*rule starP2-star-n*)

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 *hSuc-not-zero* [*iff*]: $hSuc\ m \neq 0$
by (*simp add*: *hSuc-def*)

lemmas *zero-not-hSuc* = *hSuc-not-zero* [*THEN not-sym*, *standard*, *iff*]

lemma *hSuc-hSuc-eq* [*iff*]: $(hSuc\ m = hSuc\ n) = (m = n)$
by (*simp add*: *hSuc-def star-n-one-num*)

lemma *nonempty-nat-set-Least-mem*: $c \in (S :: nat\ set) \Longrightarrow (LEAST\ n.\ n\ \in S) \in S$
by (*erule LeastI*)

lemma *nonempty-set-star-has-least*:
$!!S::nat\ set\ star.\ Iset\ S \neq \{\} \Longrightarrow \exists n \in Iset\ S.\ \forall m \in Iset\ S.\ n \leq m$
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*:
$[|\ (S::hypnat\ set) \in InternalSets;\ S \neq \{\}\ |] \Longrightarrow \exists n \in S.\ \forall m \in S.\ n \leq m$

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

18 SEQ: Sequences and Series

theory *SEQ*
imports *NatStar*
begin

constdefs

LIMSEQ :: [*nat*=>*real*,*real*] => *bool* (((-)/ −−−−> (-)) [*60*, *60*] *60*)
— Standard definition of convergence of sequence
X −−−−> *L* == (∀ *r*. *0* < *r* −−> (∃ *no*. ∀ *n*. *no* ≤ *n* −−> |*X n* + −*L*| < *r*))

NSLIMSEQ :: [*nat*=>*real*,*real*] => *bool* (((-)/ −−−−*NS*> (-)) [*60*, *60*] *60*)
— Nonstandard definition of convergence of sequence
X −−−−*NS*> *L* == (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* ≈ *hypreal-of-real L*)

lim :: (*nat* => *real*) => *real*
— Standard definition of limit using choice operator
lim X == (@*L*. (*X* −−−−> *L*))

nslim :: (*nat* => *real*) => *real*
— Nonstandard definition of limit using choice operator
nslim X == (@*L*. (*X* −−−−*NS*> *L*))

convergent :: (*nat* => *real*) => *bool*
— Standard definition of convergence
convergent X == (∃ *L*. (*X* −−−−> *L*))

NSconvergent :: (*nat* => *real*) => *bool*
— Nonstandard definition of convergence
NSconvergent X == (∃ *L*. (*X* −−−−*NS*> *L*))

Bseq :: (*nat* => *real*) => *bool*
— Standard definition for bounded sequence
Bseq X == ∃ *K*>*0*.∀ *n*. |*X n*| ≤ *K*

NSBseq :: (*nat*=>*real*) => *bool*
— Nonstandard definition for bounded sequence
NSBseq X == (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* : *HFinite*)

monoseq :: (*nat*=>*real*)=>*bool*
— Definition for monotonicity
monoseq X == (∀ *m*. ∀ *n*≥*m*. *X m* ≤ *X n*) | (∀ *m*. ∀ *n*≥*m*. *X n* ≤ *X m*)

subseq :: (*nat* => *nat*) => *bool*
— Definition of subsequence
subseq f == ∀ *m*. ∀ *n*>*m*. (*f m*) < (*f n*)

Cauchy :: (*nat* => *real*) => *bool*
— Standard definition of the Cauchy condition
Cauchy X == ∀ *e*>*0*. ∃ *M*. ∀ *m* ≥ *M*. ∀ *n* ≥ *M*. *abs*((*X m*) + −(*X n*)) < *e*

NSCauchy :: (*nat* => *real*) => *bool*
— Nonstandard definition
NSCauchy X == (∀ *M* ∈ *HNatInfinite*. ∀ *N* ∈ *HNatInfinite*.
(∗*f*∗ *X*) *M* ≈ (∗*f*∗ *X*) *N*)

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 Infinitesimal-FreeUltrafilterNat-iff starfun*)
apply (*simp add*: *star-n-inverse*)
apply (*rule bexI* [*OF - Rep-star-star-n*])
apply (*simp add*: *real-of-nat-Suc-gt-zero FreeUltrafilterNat-inverse-real-of-posnat*)
done

18.1 LIMSEQ and NSLIMSEQ

lemma *LIMSEQ-iff*:
(*X* −−−−> *L*) = (∀ *r*>*0*. ∃ *no*. ∀ *n* ≥ *no*. |*X n* + −*L*| < *r*)
by (*simp add*: *LIMSEQ-def*)

lemma *NSLIMSEQ-iff*:
(*X* −−−−*NS*> *L*) = (∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* ≈ *hypreal-of-real L*)
by (*simp add*: *NSLIMSEQ-def*)

LIMSEQ ==¿ NSLIMSEQ

lemma *LIMSEQ-NSLIMSEQ*:
X −−−−> *L* ==> *X* −−−−*NS*> *L*
apply (*simp add*: *LIMSEQ-def NSLIMSEQ-def*)
apply (*auto simp add*: *HNatInfinite-FreeUltrafilterNat-iff*)
apply (*rule-tac x* = *N* **in** *star-cases*)
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*auto simp add*: *starfun mem-infmal-iff* [*symmetric*] *star-of-def*
star-n-minus star-n-add Infinitesimal-FreeUltrafilterNat-iff)
apply (*rule bexI* [*OF - Rep-star-star-n*], *safe*)
apply (*drule-tac x* = *u* **in** *spec*, *safe*)
apply (*drule-tac x* = *no* **in** *spec*, *fuf*)
apply (*blast dest*: *less-imp-le*)
done

NSLIMSEQ ==¿ LIMSEQ

lemma *lemma-NSLIMSEQ1*: !!(*f*::*nat*=>*nat*). ∀ *n*. *n* ≤ *f n*
==> {*n*. *f n* = *0*} = {*0*} | {*n*. *f n* = *0*} = {}
apply *auto*
apply (*drule-tac x* = *xa* **in** *spec*)
apply (*drule-tac* [*2*] *x* = *x* **in** *spec*, *auto*)
done

lemma *lemma-NSLIMSEQ2*: {*n*. *f n* ≤ *Suc u*} = {*n*. *f n* ≤ *u*} *Un* {*n*. *f n* = *Suc u*}
by (*auto simp add*: *le-Suc-eq*)

lemma *lemma-NSLIMSEQ3*:
!!(*f*::*nat*=>*nat*). ∀ *n*. *n* ≤ *f n* ==> {*n*. *f n* = *Suc u*} ≤ {*n*. *n* ≤ *Suc u*}
apply *auto*
apply (*drule-tac x* = *x* **in** *spec*, *auto*)
done

the following sequence *f n* defines a hypernatural

lemma *NSLIMSEQ-finite-set*:
!!(*f*::*nat*=>*nat*). ∀ *n*. *n* ≤ *f n* ==> *finite* {*n*. *f n* ≤ *u*}
apply (*induct u*)
apply (*auto simp add*: *lemma-NSLIMSEQ2*)
apply (*auto intro*: *lemma-NSLIMSEQ3* [*THEN finite-subset*] *finite-atMost* [*unfolded atMost-def*])
apply (*drule lemma-NSLIMSEQ1*, *safe*)
apply (*simp-all* (*no-asm-simp*))
done

lemma *Compl-less-set*: $-\{n.\ u < (f::nat=>nat)\ n\} = \{n.\ f\ n \leq u\}$
by (*auto dest*: *less-le-trans simp add*: *le-def*)

the index set is in the free ultrafilter

lemma *FreeUltrafilterNat-NSLIMSEQ*:
$!!(f::nat=>nat).\ \forall n.\ n \leq f\ n ==> \{n.\ u < f\ n\} : FreeUltrafilterNat$
apply (*rule FreeUltrafilterNat-Compl-iff2* [*THEN iffD2*])
apply (*rule FreeUltrafilterNat-finite*)
apply (*auto dest*: *NSLIMSEQ-finite-set simp add*: *Compl-less-set*)
done

thus, the sequence defines an infinite hypernatural!

lemma *HNatInfinite-NSLIMSEQ*: $\forall n.\ n \leq f\ n$
$==>$ *star-n f* : *HNatInfinite*
apply (*auto simp add*: *HNatInfinite-FreeUltrafilterNat-iff*)
apply (*rule bexI* [*OF - Rep-star-star-n*], *safe*)
apply (*erule FreeUltrafilterNat-NSLIMSEQ*)
done

lemma *lemmaLIM*:
$\{n.\ X\ (f\ n) + - L = Y\ n\}$ *Int* $\{n.\ |Y\ n| < r\} \leq$
$\{n.\ |X\ (f\ n) + - L| < r\}$
by *auto*

lemma *lemmaLIM2*:
$\{n.\ |X\ (f\ n) + - L| < r\}$ *Int* $\{n.\ r \leq abs\ (X\ (f\ n) + - (L::real))\} = \{\}$
by *auto*

lemma *lemmaLIM3*: $[|\ 0 < r; \forall n.\ r \leq |X\ (f\ n) + - L|;$
$(\ *f*\ X)\ (star\text{-}n\ f) +$
$- hypreal\text{-}of\text{-}real\ \ L \approx 0\ |] ==> False$
apply (*auto simp add*: *starfun mem-infmal-iff* [*symmetric*] *star-of-def star-n-minus star-n-add Infinitesimal-FreeUltrafilterNat-iff*)
apply (*rename-tac Y*)
apply (*drule-tac x = r* **in** *spec*, *safe*)
apply (*drule FreeUltrafilterNat-Int*, *assumption*)
apply (*drule lemmaLIM* [*THEN* [*2*] *FreeUltrafilterNat-subset*])
apply (*drule FreeUltrafilterNat-all*)
apply (*erule-tac V = {n. |Y n| < r} : FreeUltrafilterNat* **in** *thin-rl*)
apply (*drule FreeUltrafilterNat-Int*, *assumption*)
apply (*simp add*: *lemmaLIM2*)
done

lemma *NSLIMSEQ-LIMSEQ*: $X\ ----NS>\ L ==>\ X\ ----->\ L$
apply (*simp add*: *LIMSEQ-def NSLIMSEQ-def*)
apply (*rule ccontr*, *simp*, *safe*)

skolemization step

apply (*drule choice*, *safe*)

apply (*drule-tac x = star-n f* **in** *bspec*)
apply (*drule-tac* [*2*] *approx-minus-iff* [*THEN iffD1*])
apply (*simp-all add*: *linorder-not-less*)
apply (*blast intro*: *HNatInfinite-NSLIMSEQ*)
apply (*blast intro*: *lemmaLIM3*)
done

Now, the all-important result is trivially proved!

theorem *LIMSEQ-NSLIMSEQ-iff*: $(f \longrightarrow L) = (f \longrightarrow_{NS} L)$
by (*blast intro*: *LIMSEQ-NSLIMSEQ NSLIMSEQ-LIMSEQ*)

18.2 Theorems About Sequences

lemma *NSLIMSEQ-const*: $(\%n.\ k) \longrightarrow_{NS} k$
by (*simp add*: *NSLIMSEQ-def*)

lemma *LIMSEQ-const*: $(\%n.\ k) \longrightarrow k$
by (*simp add*: *LIMSEQ-def*)

lemma *NSLIMSEQ-add*:
$[\!|\ X \longrightarrow_{NS} a;\ Y \longrightarrow_{NS} b\ |\!] \Longrightarrow (\%n.\ X\ n + Y\ n) \longrightarrow_{NS} a + b$
by (*auto intro*: *approx-add simp add*: *NSLIMSEQ-def starfun-add* [*symmetric*])

lemma *LIMSEQ-add*: $[\!|\ X \longrightarrow a;\ Y \longrightarrow b\ |\!] \Longrightarrow (\%n.\ X\ n + Y\ n) \longrightarrow a + b$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-add*)

lemma *LIMSEQ-add-const*: $f \longrightarrow a \Longrightarrow (\%n.(f\ n + b)) \longrightarrow a + b$
apply (*subgoal-tac* $\%n.\ (f\ n + b) == \%n.\ (f\ n + (\%n.\ b)\ n)$)
apply (*subgoal-tac* $(\%n.\ b) \longrightarrow b$)
apply (*auto simp add*: *LIMSEQ-add LIMSEQ-const*)
done

lemma *NSLIMSEQ-add-const*: $f \longrightarrow_{NS} a \Longrightarrow (\%n.(f\ n + b)) \longrightarrow_{NS} a + b$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*THEN sym*] *LIMSEQ-add-const*)

lemma *NSLIMSEQ-mult*:
$[\!|\ X \longrightarrow_{NS} a;\ Y \longrightarrow_{NS} b\ |\!] \Longrightarrow (\%n.\ X\ n * Y\ n) \longrightarrow_{NS} a * b$
by (*auto intro*!: *approx-mult-HFinite*
simp add: *NSLIMSEQ-def starfun-mult* [*symmetric*])

lemma *LIMSEQ-mult*: $[\!|\ X \longrightarrow a;\ Y \longrightarrow b\ |\!] \Longrightarrow (\%n.\ X\ n * Y\ n) \longrightarrow a * b$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-mult*)

lemma *NSLIMSEQ-minus*: $X \longrightarrow_{NS} a \Longrightarrow (\%n.\ -(X\ n)) \longrightarrow_{NS} -a$

by (*auto simp add*: *NSLIMSEQ-def starfun-minus* [*symmetric*])

lemma *LIMSEQ-minus*: X ----> a ==> (%n. $-(X\ n)$) ----> $-a$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-minus*)

lemma *LIMSEQ-minus-cancel*: (%n. $-(X\ n)$) ----> $-a$ ==> X ----> a
by (*drule LIMSEQ-minus*, *simp*)

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* [*of* Y])

lemma *LIMSEQ-add-minus*:
[| X ----> a; Y ----> b |] ==> (%n. $X\ n + -Y\ n$) ----> $a + -b$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-add-minus*)

lemma *LIMSEQ-diff*: [| X ----> a; Y ----> b |] ==> (%n. $X\ n - Y\ n$) ----> $a - b$
apply (*simp add*: *diff-minus*)
apply (*blast intro*: *LIMSEQ-add-minus*)
done

lemma *NSLIMSEQ-diff*:
[| X ----*NS*> a; Y ----*NS*> b |] ==> (%n. $X\ n - Y\ n$) ----*NS*> $a - b$
apply (*simp add*: *diff-minus*)
apply (*blast intro*: *NSLIMSEQ-add-minus*)
done

lemma *LIMSEQ-diff-const*: f ----> a ==> (%n.($f\ n - b$)) ----> $a - b$
apply (*subgoal-tac* %n. ($f\ n - b$) == %n. ($f\ n - $ (%n. b) n))
apply (*subgoal-tac* (%n. b) ----> b)
apply (*auto simp add*: *LIMSEQ-diff LIMSEQ-const*)
done

lemma *NSLIMSEQ-diff-const*: f ----*NS*> a ==> (%n.($f\ n - b$)) ----*NS*> $a - b$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*THEN sym*] *LIMSEQ-diff-const*)

Proof is like that of *NSLIM-inverse*.

lemma *NSLIMSEQ-inverse*:
[| X ----*NS*> a; $a \neq 0$ |] ==> (%n. *inverse*($X\ n$)) ----*NS*> *inverse*(a)
by (*simp add*: *NSLIMSEQ-def starfun-inverse* [*symmetric*]
hypreal-of-real-approx-inverse)

Standard version of theorem

lemma *LIMSEQ-inverse*:
[| X ----> a; a ~= 0 |] ==> (%n. *inverse*(X n)) ----> *inverse*(a)
by (*simp add*: *NSLIMSEQ-inverse LIMSEQ-NSLIMSEQ-iff*)

lemma *NSLIMSEQ-mult-inverse*:
[| 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 *LIMSEQ-divide*:
[| X ----> a; Y ----> b; b ~= 0 |] ==> (%n. X n / Y n) ----> a/b
by (*simp add*: *LIMSEQ-mult LIMSEQ-inverse divide-inverse*)

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 *LIMSEQ-unique*: [| X ----> a; X ----> b |] ==> $a = b$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-unique*)

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*: *setsum-def LIMSEQ-const*)
qed

lemma *LIMSEQ-setprod*:
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-ignore-initial-segment*: *f* ----> *a*
==> (%*n*. *f*(*n* + *k*)) ----> *a*
apply (*unfold LIMSEQ-def*)
apply (*clarify*)
apply (*drule-tac x* = *r* **in** *spec*)
apply (*clarify*)
apply (*rule-tac x* = *no* + *k* **in** *exI*)
by *auto*

lemma *LIMSEQ-offset*: (%*x*. *f* (*x* + *k*)) ----> *a* ==>
f ----> *a*
apply (*unfold LIMSEQ-def*)
apply *clarsimp*
apply (*drule-tac x* = *r* **in** *spec*)
apply *clarsimp*
apply (*rule-tac x* = *no* + *k* **in** *exI*)
apply *clarsimp*
apply (*drule-tac x* = *n* − *k* **in** *spec*)
apply (*frule mp*)
apply *arith*
apply *simp*
done

lemma *LIMSEQ-diff-approach-zero*:
g ----> *L* ==> (%*x*. *f x* − *g x*) ----> *0* ==>
f ----> *L*
apply (*drule LIMSEQ-add*)
apply *assumption*
apply *simp*
done

lemma *LIMSEQ-diff-approach-zero2*:
f −−−−> *L* ==> (%*x*. *f x* − *g x*) −−−−> *0* ==>
g −−−−> *L*
apply (*drule LIMSEQ-diff*)
apply *assumption*
apply *simp*
done

18.3 Nslim and Lim

lemma *limI*: *X* −−−−> *L* ==> *lim X* = *L*
apply (*simp add*: *lim-def*)
apply (*blast intro*: *LIMSEQ-unique*)
done

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*)

18.4 Convergence

lemma *convergentD*: *convergent X* ==> ∃ *L*. (*X* −−−−> *L*)
by (*simp add*: *convergent-def*)

lemma *convergentI*: (*X* −−−−> *L*) ==> *convergent X*
by (*auto simp add*: *convergent-def*)

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*: *someI simp add*: *NSconvergent-def nslim-def*)

lemma *convergent-LIMSEQ-iff*: *convergent X* = (*X* −−−−> *lim X*)
by (*auto intro*: *someI simp add*: *convergent-def lim-def*)

Subsequence (alternative definition, (e.g. Hoskins)

lemma *subseq-Suc-iff*: *subseq f* = (∀ *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

18.5 Monotonicity

lemma *monoseq-Suc*:
  monoseq X = ((∀ *n*. *X n* ≤ *X* (*Suc n*))
    | (∀ *n*. *X* (*Suc n*) ≤ *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 swap*)
apply (*induct-tac k*)
apply (*auto intro*: *order-trans*)
done

lemma *monoI1*: ∀ *m*. ∀ *n* ≥ *m*. *X m* ≤ *X n* ==> *monoseq X*
by (*simp add*: *monoseq-def*)

lemma *monoI2*: ∀ *m*. ∀ *n* ≥ *m*. *X n* ≤ *X m* ==> *monoseq X*
by (*simp add*: *monoseq-def*)

lemma *mono-SucI1*: ∀ *n*. *X n* ≤ *X* (*Suc n*) ==> *monoseq X*
by (*simp add*: *monoseq-Suc*)

lemma *mono-SucI2*: ∀ *n*. *X* (*Suc n*) ≤ *X n* ==> *monoseq X*
by (*simp add*: *monoseq-Suc*)

18.6 Bounded Sequence

lemma *BseqD*: *Bseq X* ==> ∃ *K*. *0* < *K* & (∀ *n*. |*X n*| ≤ *K*)
by (*simp add*: *Bseq-def*)

lemma *BseqI*: [| *0* < *K*; ∀ *n*. |*X n*| ≤ *K* |] ==> *Bseq X*
by (*auto simp add*: *Bseq-def*)

lemma *lemma-NBseq-def*:
    (∃ *K* > *0*. ∀ *n*. |*X n*| ≤ *K*) =
     (∃ *N*. ∀ *n*. |*X n*| ≤ *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* = (∃ *N*. ∀ *n*. |*X n*| ≤ *real*(*Suc N*))
apply (*simp add*: *Bseq-def*)
apply (*simp* (*no-asm*) *add*: *lemma-NBseq-def*)
done

lemma *lemma-NBseq-def2*:
(∃ *K* > *0*. ∀ *n*. |*X n*| ≤ *K*) = (∃ *N*. ∀ *n*. |*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* = (∃ *N*. ∀ *n*. |*X n*| < *real*(*Suc N*))
by (*simp add*: *Bseq-def lemma-NBseq-def2*)

lemma *NSBseqD*: [| *NSBseq X*; *N*: *HNatInfinite* |] ==> (∗*f*∗ *X*) *N* : *HFinite*
by (*simp add*: *NSBseq-def*)

lemma *NSBseqI*: ∀ *N* ∈ *HNatInfinite*. (∗*f*∗ *X*) *N* : *HFinite* ==> *NSBseq X*
by (*simp add*: *NSBseq-def*)

The standard definition implies the nonstandard definition

lemma *lemma-Bseq*: ∀ *n*. |*X n*| ≤ *K* ==> ∀ *n*. *abs*(*X*((*f*::*nat*=>*nat*) *n*)) ≤ *K*
by *auto*

lemma *Bseq-NSBseq*: *Bseq X* ==> *NSBseq X*
apply (*simp add*: *Bseq-def NSBseq-def*, *safe*)
apply (*rule-tac x* = *N* **in** *star-cases*)
apply (*auto simp add*: *starfun HFinite-FreeUltrafilterNat-iff*
HNatInfinite-FreeUltrafilterNat-iff)
apply (*rule bexI* [*OF - Rep-star-star-n*])
apply (*drule-tac f* = *Xa* **in** *lemma-Bseq*)
apply (*rule-tac x* = *K*+*1* **in** *exI*)
apply (*drule-tac P*=%*n*. *?f n* ≤ *K* **in** *FreeUltrafilterNat-all*, *ultra*)
done

The nonstandard definition implies the standard definition

We need to get rid of the real variable and do so by proving the following, which relies on the Archimedean property of the reals. When we skolemize we then get the required function *f*. Otherwise, we would be stuck with a skolem function *f* which woulid be useless.

lemma *lemmaNSBseq*:
$\forall K > 0. \exists n. K < |X\ n|$
$==> \forall N. \exists n. real(Suc\ N) < |X\ n|$
apply *safe*
apply (*cut-tac* $n = N$ **in** *real-of-nat-Suc-gt-zero*, *blast*)
done

lemma *lemmaNSBseq2*: $\forall K > 0. \exists n. K < |X\ n|$
$==> \exists f. \forall N. real(Suc\ N) < |X\ (f\ N)|$
apply (*drule lemmaNSBseq*)
apply (*drule choice*, *blast*)
done

lemma *real-seq-to-hypreal-HInfinite*:
$\forall N. real(Suc\ N) < |X\ (f\ N)|$
$==>$ *star-n* $(X\ o\ f)$: *HInfinite*
apply (*auto simp add*: *HInfinite-FreeUltrafilterNat-iff o-def*)
apply (*rule bexI* [*OF - Rep-star-star-n*], *clarify*)
apply (*cut-tac* $u = u$ **in** *FreeUltrafilterNat-nat-gt-real*)
apply (*drule FreeUltrafilterNat-all*)
apply (*erule FreeUltrafilterNat-Int* [*THEN FreeUltrafilterNat-subset*])
apply (*auto simp add*: *real-of-nat-Suc*)
done

Now prove that we can get out an infinite hypernatural as well defined using the skolem function f above

lemma *lemma-finite-NSBseq*:
$\{n. f\ n \leq Suc\ u\ \&\ real(Suc\ n) < |X\ (f\ n)|\} \leq$
$\{n. f\ n \leq u\ \&\ real(Suc\ n) < |X\ (f\ n)|\}\ Un$
$\{n. real(Suc\ n) < |X\ (Suc\ u)|\}$
by (*auto dest*!: *le-imp-less-or-eq*)

lemma *lemma-finite-NSBseq2*:
finite $\{n. f\ n \leq (u::nat)\ \&\ real(Suc\ n) < |X(f\ n)|\}$
apply (*induct u*)
apply (*rule-tac* [*2*] *lemma-finite-NSBseq* [*THEN finite-subset*])
apply (*rule-tac* $B = \{n. real\ (Suc\ n) < |X\ 0|\ \}$ **in** *finite-subset*)
apply (*auto intro*: *finite-real-of-nat-less-real*
simp add: *real-of-nat-Suc less-diff-eq* [*symmetric*])
done

lemma *HNatInfinite-skolem-f*:
$\forall N. real(Suc\ N) < |X\ (f\ N)|$
$==>$ *star-n f* : *HNatInfinite*
apply (*auto simp add*: *HNatInfinite-FreeUltrafilterNat-iff*)
apply (*rule bexI* [*OF - Rep-star-star-n*], *safe*)
apply (*rule ccontr*, *drule FreeUltrafilterNat-Compl-mem*)
apply (*rule lemma-finite-NSBseq2* [*THEN FreeUltrafilterNat-finite*, *THEN notE*])

apply (*subgoal-tac* $\{n.\ f\ n \leq u\ \&\ real\ (Suc\ n) < |X\ (f\ n)|\} =$
$\{n.\ f\ n \leq u\} \cap \{N.\ real\ (Suc\ N) < |X\ (f\ N)|\}$)
apply (*erule ssubst*)
apply (*auto simp add*: *linorder-not-less Compl-def*)
done

lemma *NSBseq-Bseq*: *NSBseq X ==> Bseq X*
apply (*simp add*: *Bseq-def NSBseq-def*)
apply (*rule ccontr*)
apply (*auto simp add*: *linorder-not-less* [*symmetric*])
apply (*drule lemmaNSBseq2*, *safe*)
apply (*frule-tac X = X* **and** *f = f* **in** *real-seq-to-hypreal-HInfinite*)
apply (*drule HNatInfinite-skolem-f* [*THEN* [*2*] *bspec*])
apply (*auto simp add*: *starfun o-def HFinite-HInfinite-iff*)
done

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-hypreal-of-real 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*)

18.7 Upper Bounds and Lubs of Bounded Sequences

lemma *Bseq-isUb*:
!!(*X*::*nat*=>*real*). *Bseq X ==> ∃ U. isUb* (*UNIV*::*real set*) $\{x.\ \exists n.\ X\ n = x\}$ *U*
by (*auto intro*: *isUbI setleI simp add*: *Bseq-def abs-le-interval-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*) $\{x.\ \exists n.\ X\ n = x\}$ *U*
by (*blast intro*: *reals-complete Bseq-isUb*)

lemma *NSBseq-isUb*: *NSBseq X ==> ∃ U. isUb UNIV* $\{x.\ \exists n.\ X\ n = x\}$ *U*
by (*simp add*: *Bseq-NSBseq-iff* [*symmetric*] *Bseq-isUb*)

lemma *NSBseq-isLub*: *NSBseq X ==> ∃ U. isLub UNIV {x. ∃ n. X n = x} U*
by (*simp add*: *Bseq-NSBseq-iff* [*symmetric*] *Bseq-isLub*)

18.8 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

Now, the same theorem in terms of NS limit

lemma *Bmonoseq-NSLIMSEQ*: ∀ *n* ≥ *m*. *X n* = *X m* ==> ∃ *L*. (*X* ----*NS*> *L*)
by (*auto dest*!: *Bmonoseq-LIMSEQ simp add*: *LIMSEQ-NSLIMSEQ-iff*)

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*
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

Nonstandard version of the theorem

lemma *NSBseq-mono-NSconvergent*:
[| *NSBseq X*; ∀ *m*. ∀ *n* ≥ *m*. *X m* ≤ *X n* |] ==> *NSconvergent X*
by (*auto intro*: *Bseq-mono-convergent*
simp add: *convergent-NSconvergent-iff* [*symmetric*]
Bseq-NSBseq-iff [*symmetric*])

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

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

18.9 A Few More Equivalence Theorems for Boundedness

alternative formulation for boundedness

lemma *Bseq-iff2*: $Bseq\ X = (\exists k > 0.\ \exists x.\ \forall n.\ |X(n) + -x| \leq k)$
apply (*unfold Bseq-def*, *safe*)
apply (*rule-tac* [*2*] $x = k + |x|$ **in** *exI*)
apply (*rule-tac* $x = K$ **in** *exI*, *simp*)
apply (*rule exI* [**where** $x = 0$], *auto*)
apply (*drule-tac* $x{=}n$ **in** *spec*, *arith*)+
done

alternative formulation for boundedness

lemma *Bseq-iff3*: $Bseq\ X = (\exists k > 0.\ \exists N.\ \forall n.\ abs(X(n) + -X(N)) \leq k)$
apply *safe*
apply (*simp add*: *Bseq-def*, *safe*)
apply (*rule-tac* $x = K + |X\ N|$ **in** *exI*)
apply *auto*
apply *arith*
apply (*rule-tac* $x = N$ **in** *exI*, *safe*)
apply (*drule-tac* $x = n$ **in** *spec*, *arith*)
apply (*auto simp add*: *Bseq-iff2*)
done

lemma *BseqI2*: $(\forall n.\ k \leq f\ n\ \&\ f\ n \leq K) ==> Bseq\ f$
apply (*simp add*: *Bseq-def*)
apply (*rule-tac* $x = (|k| + |K|) + 1$ **in** *exI*, *auto*)
apply (*drule-tac* [*2*] $x = n$ **in** *spec*, *arith*+)
done

18.10 Equivalence Between NS and Standard Cauchy Sequences

18.10.1 Standard Implies Nonstandard

lemma *lemmaCauchy1*:
$star\text{-}n\ x : HNatInfinite$
$==> \{n.\ M \leq x\ n\} : FreeUltrafilterNat$
apply (*auto simp add*: *HNatInfinite-FreeUltrafilterNat-iff*)
apply (*drule-tac* $x = M$ **in** *spec*, *ultra*)
done

lemma *lemmaCauchy2*:
$\{n.\ \forall m\ n.\ M \leq m\ \&\ M \leq (n{::}nat) --> |X\ m + -\ X\ n| < u\}\ Int$
$\{n.\ M \leq xa\ n\}\ Int\ \{n.\ M \leq x\ n\} \leq$
$\{n.\ abs\ (X\ (xa\ n) + -\ X\ (x\ n)) < u\}$
by *blast*

lemma *Cauchy-NSCauchy*: $Cauchy\ X ==> NSCauchy\ X$
apply (*simp add*: *Cauchy-def NSCauchy-def*, *safe*)

apply (*rule-tac* $x = M$ **in** *star-cases*)
apply (*rule-tac* $x = N$ **in** *star-cases*)
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*rule mem-infmal-iff* [*THEN iffD1*])
apply (*auto simp add*: *starfun star-n-minus star-n-add Infinitesimal-FreeUltrafilterNat-iff*)
apply (*rule bexI*, *auto*)
apply (*drule spec*, *auto*)
apply (*drule-tac* $M = M$ **in** *lemmaCauchy1*)
apply (*drule-tac* $M = M$ **in** *lemmaCauchy1*)
apply (*rule-tac x1* = *Xaa* **in** *lemmaCauchy2* [*THEN* [*2*] *FreeUltrafilterNat-subset*])
apply (*rule FreeUltrafilterNat-Int*)
apply (*auto intro*: *FreeUltrafilterNat-Int*)
done

18.10.2 Nonstandard Implies Standard

lemma *NSCauchy-Cauchy*: *NSCauchy X* ==> *Cauchy X*
apply (*auto simp add*: *Cauchy-def NSCauchy-def*)
apply (*rule ccontr*, *simp*)
apply (*auto dest*!: *choice HNatInfinite-NSLIMSEQ simp add*: *all-conj-distrib*)
apply (*drule bspec*, *assumption*)
apply (*drule-tac* $x =$ *star-n fa* **in** *bspec*)
apply (*auto simp add*: *starfun*)
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*drule mem-infmal-iff* [*THEN iffD2*])
apply (*auto simp add*: *star-n-minus star-n-add Infinitesimal-FreeUltrafilterNat-iff*)
apply (*rename-tac Y*)
apply (*drule-tac* $x = e$ **in** *spec*, *auto*)
apply (*drule FreeUltrafilterNat-Int*, *assumption*)
apply (*subgoal-tac* $\{n.\ |X\ (f\ n) + -\ X\ (fa\ n)| < e\} \in \mathcal{U}$)
prefer *2* **apply** (*erule FreeUltrafilterNat-subset*, *force*)
apply (*rule FreeUltrafilterNat-empty* [*THEN notE*])
apply (*subgoal-tac*
$\{n.\ abs\ (X\ (f\ n) + -\ X\ (fa\ n)) < e\}$ *Int*
$\{M.\ \sim\ abs\ (X\ (f\ M) + -\ X\ (fa\ M)) < e\}\ =\ \{\}$)
apply *auto*
done

theorem *NSCauchy-Cauchy-iff*: *NSCauchy X* = *Cauchy X*
by (*blast intro*!: *NSCauchy-Cauchy Cauchy-NSCauchy*)

A Cauchy sequence is bounded – this is the standard proof mechanization rather than the nonstandard proof

lemma *lemmaCauchy*: $\forall n \geq M.\ |X\ M + -\ X\ n| < (1::real)$
==> $\forall n \geq M.\ |X\ n| < 1 + |X\ M|$
apply *safe*
apply (*drule spec*, *auto*, *arith*)
done

lemma *less-Suc-cancel-iff*: $(n < Suc\ M) = (n \leq M)$
by *auto*

FIXME: Long. Maximal element in subsequence

lemma *SUP-rabs-subseq*:
$\exists m \leq M.\ \forall n \leq M.\ |(X::nat=> real)\ n| \leq |X\ m|$
apply (*induct M*)
apply (*rule-tac x = 0* **in** *exI*, *simp*, *safe*)
apply (*cut-tac x = |X (Suc M)|* **and** *y = |X m|* **in** *linorder-less-linear*)
apply *safe*
apply (*rule-tac x = m* **in** *exI*)
apply (*rule-tac [2] x = m* **in** *exI*)
apply (*rule-tac [3] x = Suc M* **in** *exI*, *simp-all*, *safe*)
apply (*erule-tac* [!] *m1 = n* **in** *le-imp-less-or-eq* [*THEN disjE*])
apply (*simp-all add*: *less-Suc-cancel-iff*)
apply (*blast intro*: *order-le-less-trans* [*THEN order-less-imp-le*])
done

lemma *lemma-Nat-covered*:
$[|\ \forall m::nat.\ m \leq M\ --> P\ M\ m;$
$\forall m \geq M.\ P\ M\ m\ |]$
$==> \forall m.\ P\ M\ m$
by (*auto elim*: *less-asym simp add*: *le-def*)

lemma *lemma-trans1*:
$[|\ \forall n \leq M.\ |(X::nat=>real)\ n| \leq a;\ \ a < b\ |]$
$==> \forall n \leq M.\ |X\ n| \leq b$
by (*blast intro*: *order-le-less-trans* [*THEN order-less-imp-le*])

lemma *lemma-trans2*:
$[|\ \forall n \geq M.\ |(X::nat=>real)\ n| < a;\ a < b\ |]$
$==> \forall n \geq M.\ |X\ n| \leq b$
by (*blast intro*: *order-less-trans* [*THEN order-less-imp-le*])

lemma *lemma-trans3*:
$[|\ \forall n \leq M.\ |X\ n| \leq a;\ a = b\ |]$
$==> \forall n \leq M.\ |X\ n| \leq b$
by *auto*

lemma *lemma-trans4*: $\forall n \geq M.\ |(X::nat=>real)\ n| < a$
$==> \forall n \geq M.\ |X\ n| \leq a$
by (*blast intro*: *order-less-imp-le*)

Proof is more involved than outlines sketched by various authors would suggest

lemma *Cauchy-Bseq*: *Cauchy X ==> Bseq X*
apply (*simp add*: *Cauchy-def Bseq-def*)

apply (*drule-tac x = 1* **in** *spec*)
apply (*erule zero-less-one* [*THEN* [*2*] *impE*], *safe*)
apply (*drule-tac x = M* **in** *spec*, *simp*)
apply (*drule lemmaCauchy*)
apply (*cut-tac M = M* **and** *X = X* **in** *SUP-rabs-subseq*, *safe*)
apply (*cut-tac x = |X m|* **and** *y = 1 + |X M|* **in** *linorder-less-linear*)
apply *safe*
apply (*drule lemma-trans1*, *assumption*)
apply (*drule-tac* [*3*] *lemma-trans2*, *erule-tac* [*3*] *asm-rl*)
apply (*drule-tac* [*2*] *lemma-trans3*, *erule-tac* [*2*] *asm-rl*)
apply (*drule-tac* [*3*] *abs-add-one-gt-zero* [*THEN order-less-trans*])
apply (*drule lemma-trans4*)
apply (*drule-tac* [*2*] *lemma-trans4*)
apply (*rule-tac x = 1 + |X M|* **in** *exI*)
apply (*rule-tac* [*2*] *x = 1 + |X M|* **in** *exI*)
apply (*rule-tac* [*3*] *x = |X m|* **in** *exI*)
apply (*auto elim*!: *lemma-Nat-covered*)
done

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*)

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 *NSCauchy-NSconvergent-iff*: *NSCauchy X = NSconvergent X*
apply (*simp add*: *NSconvergent-def NSLIMSEQ-def*, *safe*)
apply (*frule NSCauchy-NSBseq*)
apply (*auto intro*: *approx-trans2 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

Standard proof for free

lemma *Cauchy-convergent-iff*: *Cauchy X = convergent X*
by (*simp add*: *NSCauchy-Cauchy-iff* [*symmetric*] *convergent-NSconvergent-iff NSCauchy-NSconvergent-iff*)

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;
$\exists N.\ \forall n \geq N.\ f(n) \leq g(n)$
|] ==> $l \leq m$
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 *LIMSEQ-le*:
[| f −−−−> l; g −−−−> m; $\exists N.\ \forall n \geq N.\ f(n) \leq g(n)$ |]
==> $l \leq m$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-le*)

lemma *LIMSEQ-le-const*: [| X −−−−> r; $\forall n.\ a \leq X\ n$ |] ==> $a \leq r$
apply (*rule LIMSEQ-le*)
apply (*rule LIMSEQ-const*, *auto*)
done

lemma *NSLIMSEQ-le-const*: [| X −−−−*NS*> r; $\forall n.\ a \leq X\ n$ |] ==> $a \leq r$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff LIMSEQ-le-const*)

lemma *LIMSEQ-le-const2*: [| X −−−−> r; $\forall n.\ X\ n \leq a$ |] ==> $r \leq a$
apply (*rule LIMSEQ-le*)
apply (*rule-tac* [*2*] *LIMSEQ-const*, *auto*)
done

lemma *NSLIMSEQ-le-const2*: [| X −−−−*NS*> r; $\forall n.\ X\ n \leq a$ |] ==> $r \leq a$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff LIMSEQ-le-const2*)

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 (*frule NSconvergentI* [*THEN NSCauchy-NSconvergent-iff* [*THEN iffD2*]])
apply (*auto simp add*: *NSCauchy-def NSLIMSEQ-def starfun-shift-one*)
apply (*drule bspec*, *assumption*)
apply (*drule bspec*, *assumption*)
apply (*drule Nats-1* [*THEN* [*2*] *HNatInfinite-SHNat-add*])
apply (*blast intro*: *approx-trans3*)
done

standard version

lemma *LIMSEQ-Suc*: f −−−−> l ==> (%n. f(*Suc* n)) −−−−> l
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-Suc*)

lemma *NSLIMSEQ-imp-Suc*: $(\%n.\ f(Suc\ n))\ ----NS>\ l \Longrightarrow f\ ----NS>\ l$
apply (*frule NSconvergentI* [*THEN NSCauchy-NSconvergent-iff* [*THEN iffD2*]])
apply (*auto simp add*: *NSCauchy-def NSLIMSEQ-def starfun-shift-one*)
apply (*drule bspec*, *assumption*)
apply (*drule bspec*, *assumption*)
apply (*frule Nats-1* [*THEN* [*2*] *HNatInfinite-SHNat-diff*])
apply (*drule-tac x=N − 1* **in** *bspec*)
apply (*auto intro*: *approx-trans3*)
done

lemma *LIMSEQ-imp-Suc*: $(\%n.\ f(Suc\ n))\ ----> l \Longrightarrow f\ ----> l$
apply (*simp add*: *LIMSEQ-NSLIMSEQ-iff*)
apply (*erule NSLIMSEQ-imp-Suc*)
done

lemma *LIMSEQ-Suc-iff*: $((\%n.\ f(Suc\ n))\ ----> l) = (f\ ----> l)$
by (*blast intro*: *LIMSEQ-imp-Suc LIMSEQ-Suc*)

lemma *NSLIMSEQ-Suc-iff*: $((\%n.\ f(Suc\ n))\ ----NS> l) = (f\ ----NS> l)$
by (*blast intro*: *NSLIMSEQ-imp-Suc NSLIMSEQ-Suc*)

A sequence tends to zero iff its abs does

lemma *LIMSEQ-rabs-zero*: $((\%n.\ |f\ n|)\ ----> 0) = (f\ ----> 0)$
by (*simp add*: *LIMSEQ-def*)

We prove the NS version from the standard one, since the NS proof seems more complicated than the standard one above!

lemma *NSLIMSEQ-rabs-zero*: $((\%n.\ |f\ n|)\ ----NS> 0) = (f\ ----NS> 0)$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-rabs-zero*)

Generalization to other limits

lemma *NSLIMSEQ-imp-rabs*: $f\ ----NS> l \Longrightarrow (\%n.\ |f\ n|)\ ----NS> |l|$
apply (*simp add*: *NSLIMSEQ-def*)
apply (*auto intro*: *approx-hrabs*
simp add: *starfun-abs hypreal-of-real-hrabs* [*symmetric*])
done

standard version

lemma *LIMSEQ-imp-rabs*: $f\ ----> l \Longrightarrow (\%n.\ |f\ n|)\ ----> |l|$
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-imp-rabs*)

An unbounded sequence's inverse tends to 0

standard proof seems easier

lemma *LIMSEQ-inverse-zero*:
$\forall y.\ \exists N.\ \forall n \geq N.\ y < f(n) \Longrightarrow (\%n.\ inverse(f\ n))\ ----> 0$

apply (*simp add*: *LIMSEQ-def*, *safe*)
apply (*drule-tac* $x = inverse\ r$ **in** *spec*, *safe*)
apply (*rule-tac* $x = N$ **in** *exI*, *safe*)
apply (*drule spec*, *auto*)
apply (*frule positive-imp-inverse-positive*)
apply (*frule order-less-trans*, *assumption*)
apply (*frule-tac* $a = f\ n$ **in** *positive-imp-inverse-positive*)
apply (*simp add*: *abs-if*)
apply (*rule-tac* $t = r$ **in** *inverse-inverse-eq* [*THEN subst*])
apply (*auto intro*: *inverse-less-iff-less* [*THEN iffD2*]
simp *del*: *inverse-inverse-eq*)
done

lemma *NSLIMSEQ-inverse-zero*:
$\forall y.\ \exists N.\ \forall n \geq N.\ y < f(n)$
$==>$ $(\%n.\ inverse(f\ n))$ $----NS>$ *0*
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff* [*symmetric*] *LIMSEQ-inverse-zero*)

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* $x = y$ **in** *reals-Archimedean2*)
apply (*safe*, *rule-tac* $x = n$ **in** *exI*)
apply (*auto simp add*: *real-of-nat-Suc*)
done

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*)

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 *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 *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 *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 *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 *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*)

Real Powers

lemma *NSLIMSEQ-pow* [*rule-format*]:
(X −−−−NS> a) −−> ((%n. (X n) ^ m) −−−−NS> a ^ m)
apply (*induct m*)
apply (*auto intro*: *NSLIMSEQ-mult NSLIMSEQ-const*)
done

lemma *LIMSEQ-pow*: X −−−−> a ==> (%n. (X n) ^ m) −−−−> a ^ m
by (*simp add*: *LIMSEQ-NSLIMSEQ-iff NSLIMSEQ-pow*)

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 ≤ x; x ≤ 1 |] ==> *Bseq* (%n. x ^ n)
apply (*simp add*: *Bseq-def*)
apply (*rule-tac x = 1* **in** *exI*)
apply (*simp add*: *power-abs*)
apply (*auto dest*: *power-mono intro*: *order-less-imp-le simp add*: *abs-if*)
done

lemma *monoseq-realpow*: [| 0 ≤ x; x ≤ 1 |] ==> *monoseq* (%n. x ^ 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 ≤ x; x ≤ 1 |] ==> *convergent* (%n. x ^ n)
by (*blast intro*!: *Bseq-monoseq-convergent Bseq-realpow monoseq-realpow*)

We now use NS criterion to bring proof of theorem through

lemma *NSLIMSEQ-realpow-zero*: [| 0 ≤ x; x < 1 |] ==> (%n. x ^ 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-SReal*, *assumption*)
apply (*drule approx-trans3*, *assumption*)
apply (*auto simp del*: *star-of-mult simp add*: *star-of-mult* [*symmetric*])
done

standard version

lemma *LIMSEQ-realpow-zero*: [| *0* ≤ *x*; *x* < *1* |] ==> (%*n*. *x* ^ *n*) ----> *0*
by (*simp add*: *NSLIMSEQ-realpow-zero LIMSEQ-NSLIMSEQ-iff*)

lemma *LIMSEQ-divide-realpow-zero*: *1* < *x* ==> (%*n*. *a* / (*x* ^ *n*)) ----> *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* ^ *n* for |*c*| < (*1*::'*a*)

lemma *LIMSEQ-rabs-realpow-zero*: |*c*| < *1* ==> (%*n*. |*c*| ^ *n*) ----> *0*
by (*blast intro*!: *LIMSEQ-realpow-zero abs-ge-zero*)

lemma *NSLIMSEQ-rabs-realpow-zero*: |*c*| < *1* ==> (%*n*. |*c*| ^ *n*) ----*NS*> *0*
by (*simp add*: *LIMSEQ-rabs-realpow-zero LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

lemma *LIMSEQ-rabs-realpow-zero2*: |*c*| < *1* ==> (%*n*. *c* ^ *n*) ----> *0*
apply (*rule LIMSEQ-rabs-zero* [*THEN iffD1*])
apply (*auto intro*: *LIMSEQ-rabs-realpow-zero simp add*: *power-abs*)
done

lemma *NSLIMSEQ-rabs-realpow-zero2*: |*c*| < *1* ==> (%*n*. *c* ^ *n*) ----*NS*> *0*
by (*simp add*: *LIMSEQ-rabs-realpow-zero2 LIMSEQ-NSLIMSEQ-iff* [*symmetric*])

18.11 Hyperreals and Sequences

A bounded sequence is a finite hyperreal

lemma *NSBseq-HFinite-hypreal*: *NSBseq X* ==> *star-n X* : *HFinite*
by (*auto intro*!: *bexI lemma-starrel-refl*
intro: *FreeUltrafilterNat-all* [*THEN FreeUltrafilterNat-subset*]
simp add: *HFinite-FreeUltrafilterNat-iff Bseq-NSBseq-iff* [*symmetric*]
Bseq-iff1a)

A sequence converging to zero defines an infinitesimal

lemma *NSLIMSEQ-zero-Infinitesimal-hypreal*:
X ----*NS*> *0* ==> *star-n X* : *Infinitesimal*
apply (*simp add*: *NSLIMSEQ-def*)
apply (*drule-tac x = whn* **in** *bspec*)
apply (*simp add*: *HNatInfinite-whn*)

apply (*auto simp add*: *hypnat-omega-def mem-infmal-iff* [*symmetric*] *starfun*)
done

ML
⟪
val Cauchy-def = thmCauchy-def;
val SEQ-Infinitesimal = thm SEQ-Infinitesimal;
val LIMSEQ-iff = thm LIMSEQ-iff;
val NSLIMSEQ-iff = thm NSLIMSEQ-iff;
val LIMSEQ-NSLIMSEQ = thm LIMSEQ-NSLIMSEQ;
val NSLIMSEQ-finite-set = thm NSLIMSEQ-finite-set;
val Compl-less-set = thm Compl-less-set;
val FreeUltrafilterNat-NSLIMSEQ = thm FreeUltrafilterNat-NSLIMSEQ;
val HNatInfinite-NSLIMSEQ = thm HNatInfinite-NSLIMSEQ;
val NSLIMSEQ-LIMSEQ = thm NSLIMSEQ-LIMSEQ;
val LIMSEQ-NSLIMSEQ-iff = thm LIMSEQ-NSLIMSEQ-iff;
val NSLIMSEQ-const = thm NSLIMSEQ-const;
val LIMSEQ-const = thm LIMSEQ-const;
val NSLIMSEQ-add = thm NSLIMSEQ-add;
val LIMSEQ-add = thm LIMSEQ-add;
val NSLIMSEQ-mult = thm NSLIMSEQ-mult;
val LIMSEQ-mult = thm LIMSEQ-mult;
val NSLIMSEQ-minus = thm NSLIMSEQ-minus;
val LIMSEQ-minus = thm LIMSEQ-minus;
val LIMSEQ-minus-cancel = thm LIMSEQ-minus-cancel;
val NSLIMSEQ-minus-cancel = thm NSLIMSEQ-minus-cancel;
val NSLIMSEQ-add-minus = thm NSLIMSEQ-add-minus;
val LIMSEQ-add-minus = thm LIMSEQ-add-minus;
val LIMSEQ-diff = thm LIMSEQ-diff;
val NSLIMSEQ-diff = thm NSLIMSEQ-diff;
val NSLIMSEQ-inverse = thm NSLIMSEQ-inverse;
val LIMSEQ-inverse = thm LIMSEQ-inverse;
val NSLIMSEQ-mult-inverse = thm NSLIMSEQ-mult-inverse;
val LIMSEQ-divide = thm LIMSEQ-divide;
val NSLIMSEQ-unique = thm NSLIMSEQ-unique;
val LIMSEQ-unique = thm LIMSEQ-unique;
val limI = thm limI;
val nslimI = thm nslimI;
val lim-nslim-iff = thm lim-nslim-iff;
val convergentD = thm convergentD;
val convergentI = thm convergentI;
val NSconvergentD = thm NSconvergentD;
val NSconvergentI = thm NSconvergentI;
val convergent-NSconvergent-iff = thm convergent-NSconvergent-iff;
val NSconvergent-NSLIMSEQ-iff = thm NSconvergent-NSLIMSEQ-iff;
val convergent-LIMSEQ-iff = thm convergent-LIMSEQ-iff;

val subseq-Suc-iff = thm subseq-Suc-iff;
val monoseq-Suc = thm monoseq-Suc;
val monoI1 = thm monoI1;
val monoI2 = thm monoI2;
val mono-SucI1 = thm mono-SucI1;
val mono-SucI2 = thm mono-SucI2;
val BseqD = thm BseqD;
val BseqI = thm BseqI;
val Bseq-iff = thm Bseq-iff;
val Bseq-iff1a = thm Bseq-iff1a;
val NSBseqD = thm NSBseqD;
val NSBseqI = thm NSBseqI;
val Bseq-NSBseq = thm Bseq-NSBseq;
val real-seq-to-hypreal-HInfinite = thm real-seq-to-hypreal-HInfinite;
val HNatInfinite-skolem-f = thm HNatInfinite-skolem-f;
val NSBseq-Bseq = thm NSBseq-Bseq;
val Bseq-NSBseq-iff = thm Bseq-NSBseq-iff;
val NSconvergent-NSBseq = thm NSconvergent-NSBseq;
val convergent-Bseq = thm convergent-Bseq;
val Bseq-isUb = thm Bseq-isUb;
val Bseq-isLub = thm Bseq-isLub;
val NSBseq-isUb = thm NSBseq-isUb;
val NSBseq-isLub = thm NSBseq-isLub;
val Bmonoseq-LIMSEQ = thm Bmonoseq-LIMSEQ;
val Bmonoseq-NSLIMSEQ = thm Bmonoseq-NSLIMSEQ;
val Bseq-mono-convergent = thm Bseq-mono-convergent;
val NSBseq-mono-NSconvergent = thm NSBseq-mono-NSconvergent;
val convergent-minus-iff = thm convergent-minus-iff;
val Bseq-minus-iff = thm Bseq-minus-iff;
val Bseq-monoseq-convergent = thm Bseq-monoseq-convergent;
val Bseq-iff2 = thm Bseq-iff2;
val Bseq-iff3 = thm Bseq-iff3;
val BseqI2 = thm BseqI2;
val Cauchy-NSCauchy = thm Cauchy-NSCauchy;
val NSCauchy-Cauchy = thm NSCauchy-Cauchy;
val NSCauchy-Cauchy-iff = thm NSCauchy-Cauchy-iff;
val less-Suc-cancel-iff = thm less-Suc-cancel-iff;
val SUP-rabs-subseq = thm SUP-rabs-subseq;
val Cauchy-Bseq = thm Cauchy-Bseq;
val NSCauchy-NSBseq = thm NSCauchy-NSBseq;
val NSCauchy-NSconvergent-iff = thm NSCauchy-NSconvergent-iff;
val Cauchy-convergent-iff = thm Cauchy-convergent-iff;
val NSLIMSEQ-le = thm NSLIMSEQ-le;
val LIMSEQ-le = thm LIMSEQ-le;
val LIMSEQ-le-const = thm LIMSEQ-le-const;
val NSLIMSEQ-le-const = thm NSLIMSEQ-le-const;
val LIMSEQ-le-const2 = thm LIMSEQ-le-const2;
val NSLIMSEQ-le-const2 = thm NSLIMSEQ-le-const2;
val NSLIMSEQ-Suc = thm NSLIMSEQ-Suc;

val LIMSEQ-Suc = thm LIMSEQ-Suc;
val NSLIMSEQ-imp-Suc = thm NSLIMSEQ-imp-Suc;
val LIMSEQ-imp-Suc = thm LIMSEQ-imp-Suc;
val LIMSEQ-Suc-iff = thm LIMSEQ-Suc-iff;
val NSLIMSEQ-Suc-iff = thm NSLIMSEQ-Suc-iff;
val LIMSEQ-rabs-zero = thm LIMSEQ-rabs-zero;
val NSLIMSEQ-rabs-zero = thm NSLIMSEQ-rabs-zero;
val NSLIMSEQ-imp-rabs = thm NSLIMSEQ-imp-rabs;
val LIMSEQ-imp-rabs = thm LIMSEQ-imp-rabs;
val LIMSEQ-inverse-zero = thm LIMSEQ-inverse-zero;
val NSLIMSEQ-inverse-zero = thm NSLIMSEQ-inverse-zero;
val LIMSEQ-inverse-real-of-nat = thm LIMSEQ-inverse-real-of-nat;
val NSLIMSEQ-inverse-real-of-nat = thm NSLIMSEQ-inverse-real-of-nat;
val LIMSEQ-inverse-real-of-nat-add = thm LIMSEQ-inverse-real-of-nat-add;
val NSLIMSEQ-inverse-real-of-nat-add = thm NSLIMSEQ-inverse-real-of-nat-add;
val LIMSEQ-inverse-real-of-nat-add-minus = thm LIMSEQ-inverse-real-of-nat-add-minus;
val NSLIMSEQ-inverse-real-of-nat-add-minus = thm NSLIMSEQ-inverse-real-of-nat-add-minus;
val LIMSEQ-inverse-real-of-nat-add-minus-mult = thm LIMSEQ-inverse-real-of-nat-add-minus-mult;
val NSLIMSEQ-inverse-real-of-nat-add-minus-mult = thm NSLIMSEQ-inverse-real-of-nat-add-minus-mult;
val NSLIMSEQ-pow = thm NSLIMSEQ-pow;
val LIMSEQ-pow = thm LIMSEQ-pow;
val Bseq-realpow = thm Bseq-realpow;
val monoseq-realpow = thm monoseq-realpow;
val convergent-realpow = thm convergent-realpow;
val NSLIMSEQ-realpow-zero = thm NSLIMSEQ-realpow-zero;
⟫

end

19 Lim: Limits, Continuity and Differentiation

theory *Lim*
imports *SEQ*
begin

Standard and Nonstandard Definitions

constdefs

```
  LIM :: [real=>real,real,real] => bool
                              (((-)/ -- (-)/ --> (-)) [60, 0, 60] 60)
  f -- a --> L ==
     ∀ r > 0. ∃ s > 0. ∀ x. x ≠ a & |x + -a| < s
                          --> |f x + -L| < r

  NSLIM :: [real=>real,real,real] => bool
                            (((-)/ -- (-)/ --NS> (-)) [60, 0, 60] 60)
  f -- a --NS> L == (∀ x. (x ≠ hypreal-of-real a &
                     x @= hypreal-of-real a -->
```

$(\ {*f*}\ f)\ x\ @{=}\ \textit{hypreal-of-real}\ L))$

$\textit{isCont} :: [\textit{real}{=>}\textit{real},\textit{real}] => \textit{bool}$
$\textit{isCont}\ f\ a == (f\ --\ a\ --{>}\ (f\ a))$

$\textit{isNSCont} :: [\textit{real}{=>}\textit{real},\textit{real}] => \textit{bool}$
— NS definition dispenses with limit notions
$\textit{isNSCont}\ f\ a == (\forall\, y.\ y\ @{=}\ \textit{hypreal-of-real}\ a\ --{>}$
$(\ {*f*}\ f)\ y\ @{=}\ \textit{hypreal-of-real}\ (f\ a))$

$\textit{deriv}:: [\textit{real}{=>}\textit{real},\textit{real},\textit{real}] => \textit{bool}$
— Differentiation: D is derivative of function f at x
$((\textit{DERIV}\ (\text{-})/\ (\text{-})/\ :>\ (\text{-}))\ [1000,\ 1000,\ 60]\ 60)$
$\textit{DERIV}\ f\ x :> D == ((\%h.\ (f(x + h) + -f\ x)/h)\ --\ 0\ --{>}\ D)$

$\textit{nsderiv} :: [\textit{real}{=>}\textit{real},\textit{real},\textit{real}] => \textit{bool}$
$((\textit{NSDERIV}\ (\text{-})/\ (\text{-})/\ :>\ (\text{-}))\ [1000,\ 1000,\ 60]\ 60)$
$\textit{NSDERIV}\ f\ x :> D == (\forall\, h \in \textit{Infinitesimal} - \{0\}.$
$(({*f*}\ f)(\textit{hypreal-of-real}\ x + h) +$
$- \textit{hypreal-of-real}\ (f\ x))/h\ @{=}\ \textit{hypreal-of-real}\ D)$

$\textit{differentiable} :: [\textit{real}{=>}\textit{real},\textit{real}] => \textit{bool}\quad (\textbf{infixl}\ \textit{differentiable}\ 60)$
$f\ \textit{differentiable}\ x == (\exists\, D.\ \textit{DERIV}\ f\ x :> D)$

$\textit{NSdifferentiable} :: [\textit{real}{=>}\textit{real},\textit{real}] => \textit{bool}$
$(\textbf{infixl}\ \textit{NSdifferentiable}\ 60)$
$f\ \textit{NSdifferentiable}\ x == (\exists\, D.\ \textit{NSDERIV}\ f\ x :> D)$

$\textit{increment} :: [\textit{real}{=>}\textit{real},\textit{real},\textit{hypreal}] => \textit{hypreal}$
$\textit{increment}\ f\ x\ h == (@\textit{inc}.\ f\ \textit{NSdifferentiable}\ x\ \&$
$\textit{inc} = (\ {*f*}\ f)(\textit{hypreal-of-real}\ x + h) + -\textit{hypreal-of-real}\ (f\ x))$

$\textit{isUCont} :: (\textit{real}{=>}\textit{real}) => \textit{bool}$
$\textit{isUCont}\ f == \forall\, r > 0.\ \exists\, s > 0.\ \forall\, x\ y.\ |x + -y| < s\ --{>}\ |f\ x + -f\ y| < r$

$\textit{isNSUCont} :: (\textit{real}{=>}\textit{real}) => \textit{bool}$
$\textit{isNSUCont}\ f == (\forall\, x\ y.\ x\ @{=}\ y\ --{>}\ (\ {*f*}\ f)\ x\ @{=}\ (\ {*f*}\ f)\ y)$

consts
$\textit{Bolzano-bisect} :: [\textit{real}{*}\textit{real}{=>}\textit{bool},\ \textit{real},\ \textit{real},\ \textit{nat}] => (\textit{real}{*}\textit{real})$
— Used in the proof of the Bolzano theorem

primrec
$\textit{Bolzano-bisect}\ P\ a\ b\ 0 = (a,b)$
$\textit{Bolzano-bisect}\ P\ a\ b\ (\textit{Suc}\ n) =$
$(\textit{let}\ (x,y) = \textit{Bolzano-bisect}\ P\ a\ b\ n$
$\textit{in if}\ P(x,\ (x+y)/2)\ \textit{then}\ ((x+y)/2,\ y)$

else $(x, (x+y)/2)$)

20 Some Purely Standard Proofs

lemma *LIM-eq*:
$f \dashrightarrow a \longrightarrow L =$
$(\forall r{>}0.\exists s{>}0.\forall x.\ x \neq a\ \&\ |x-a| < s \longrightarrow |f\ x - L| < r)$
by (*simp add*: *LIM-def diff-minus*)

lemma *LIM-D*:
$[|\ f \dashrightarrow a \longrightarrow L;\ 0{<}r\ |]$
$\Longrightarrow \exists s{>}0.\forall x.\ x \neq a\ \&\ |x-a| < s \longrightarrow |f\ x - L| < r$
by (*simp add*: *LIM-eq*)

lemma *LIM-const* [*simp*]: $(\%x.\ k) \dashrightarrow x \longrightarrow k$
by (*simp add*: *LIM-def*)

lemma *LIM-add*:
assumes *f*: $f \dashrightarrow a \longrightarrow L$ **and** *g*: $g \dashrightarrow a \longrightarrow M$
shows $(\%x.\ f\ x + g(x)) \dashrightarrow a \longrightarrow (L + M)$
proof (*simp add*: *LIM-eq*, *clarify*)
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*: $\forall x.\ x \neq a\ \&\ |x-a| < fs \longrightarrow |f\ x - L| < r/2$
by *blast*
from *LIM-D* [*OF g half-gt-zero* [*OF r*]]
obtain *gs*
where *gs*: $0 < gs$
and *gs-lt*: $\forall x.\ x \neq a\ \&\ |x-a| < gs \longrightarrow |g\ x - M| < r/2$
by *blast*
show $\exists s{>}0.\forall x.\ x \neq a \wedge |x-a| < s \longrightarrow |f\ x + g\ x - (L + M)| < r$
proof (*intro exI conjI strip*)
show $0 < min\ fs\ gs$ **by** (*simp add*: *fs gs*)
fix x :: *real*
assume $x \neq a \wedge |x-a| < min\ fs\ gs$
with *fs-lt gs-lt*
have $|f\ x - L| < r/2$ **and** $|g\ x - M| < r/2$ **by** (*auto simp add*: *fs-lt*)
hence $|f\ x - L| + |g\ x - M| < r$ **by** *arith*
thus $|f\ x + g\ x - (L + M)| < r$
by (*blast intro*: *abs-diff-triangle-ineq order-le-less-trans*)
qed
qed

lemma *LIM-minus*: $f \dashrightarrow a \longrightarrow L \Longrightarrow (\%x.\ -f(x)) \dashrightarrow a \longrightarrow -L$
apply (*simp add*: *LIM-eq*)
apply (*subgoal-tac* $\forall x.\ |-f\ x + L| = |f\ x - L|$)

apply (*simp-all add*: *abs-if*)
done

lemma *LIM-add-minus*:
[| f –– x ––> l; g –– x ––> m |] ==> (%x. $f(x)$ + −$g(x)$) –– x ––> (l + −m)
by (*blast dest*: *LIM-add LIM-minus*)

lemma *LIM-diff*:
[| f –– x ––> l; g –– x ––> m |] ==> (%x. $f(x)$ − $g(x)$) –– x ––> l−m
by (*simp add*: *diff-minus LIM-add-minus*)

lemma *LIM-const-not-eq*: $k \neq L$ ==> ~ ((%x. k) –– a ––> L)
proof (*simp add*: *linorder-neq-iff LIM-eq*, *elim disjE*)
assume k: $k < L$
show ∃ r>0. ∀ s>0. (∃ x. ($x < a \vee a < x$) ∧ |x−a| < s) ∧ ¬ |k−L| < r
proof (*intro exI conjI strip*)
show $0 < L-k$ **by** (*simp add*: *k compare-rls*)
fix s :: *real*
assume s: $0<s$
{ from s **show** $s/2 + a < a \vee a < s/2 + a$ **by** *arith*
next
from s **show** |s / 2 + a − a| < s **by** (*simp add*: *abs-if*)
next
from s **show** ~ |k−L| < L−k **by** (*simp add*: *abs-if*) **}**
qed
next
assume k: $L < k$
show ∃ r>0.∀ s>0. (∃ x. ($x < a \vee a < x$) ∧ |x−a| < s) ∧ ¬ |k−L| < r
proof (*intro exI conjI strip*)
show $0 < k-L$ **by** (*simp add*: *k compare-rls*)
fix s :: *real*
assume s: $0<s$
{ from s **show** $s/2 + a < a \vee a < s/2 + a$ **by** *arith*
next
from s **show** |s / 2 + a − a| < s **by** (*simp add*: *abs-if*)
next
from s **show** ~ |k−L| < k−L **by** (*simp add*: *abs-if*) **}**
qed
qed

lemma *LIM-const-eq*: (%x. k) –– x ––> L ==> $k = L$
apply (*rule ccontr*)
apply (*blast dest*: *LIM-const-not-eq*)
done

lemma *LIM-unique*: [| f –– a ––> L; f –– a ––> M |] ==> $L = M$
apply (*drule LIM-diff*, *assumption*)

apply (*auto dest*!: *LIM-const-eq*)
done

lemma *LIM-mult-zero*:
  assumes *f*: f −− a −−> 0 **and** *g*: g −− a −−> 0
  shows $(\%x.\ f(x) * g(x))$ −− a −−> 0
proof (*simp add*: *LIM-eq abs-mult*, *clarify*)
  fix r :: *real*
  assume *r*: $0<r$
  from *LIM-D* [*OF f zero-less-one*]
  obtain *fs*
    where *fs*: $0 < fs$
      and *fs-lt*: $\forall x.\ x \neq a \ \&\ |x-a| < fs$ −−> $|f\ x| < 1$
  by *auto*
  from *LIM-D* [*OF g r*]
  obtain *gs*
    where *gs*: $0 < gs$
      and *gs-lt*: $\forall x.\ x \neq a \ \&\ |x-a| < gs$ −−> $|g\ x| < r$
  by *auto*
  show $\exists s.\ 0 < s \wedge (\forall x.\ x \neq a \wedge |x-a| < s \longrightarrow |f\ x| * |g\ x| < r)$
  proof (*intro exI conjI strip*)
    show $0 < min\ fs\ gs$ **by** (*simp add*: *fs gs*)
    fix x :: *real*
    assume $x \neq a \wedge |x-a| < min\ fs\ gs$
    with *fs-lt gs-lt*
    have $|f\ x| < 1$ **and** $|g\ x| < r$ **by** (*auto simp add*: *fs-lt*)
    hence $|f\ x| * |g\ x| < 1*r$ **by** (*rule abs-mult-less*)
    thus $|f\ x| * |g\ x| < r$ **by** *simp*
  qed
qed

lemma *LIM-self*: $(\%x.\ x)$ −− a −−> a
by (*auto simp add*: *LIM-def*)

Limits are equal for functions equal except at limit point

lemma *LIM-equal*:
    [| $\forall x.\ x \neq a$ −−> $(f\ x = g\ x)$ |] ==> $(f$ −− a −−> $l) = (g$ −− a −−> $l)$
by (*simp add*: *LIM-def*)

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

20.1 Relationships Between Standard and Nonstandard Concepts

Standard and NS definitions of Limit

lemma *LIM-NSLIM*:
$f \dashrightarrow x \longrightarrow L \Longrightarrow f \dashrightarrow x \longrightarrow_{NS} L$
apply (*simp add*: *LIM-def NSLIM-def approx-def*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff*, *safe*)
apply (*rule-tac* $x = xa$ **in** *star-cases*)
apply (*auto simp add*: *real-add-minus-iff starfun star-n-minus star-of-def star-n-add star-n-eq-iff*)
apply (*rule bexI* [*OF - Rep-star-star-n*], *clarify*)
apply (*drule-tac* $x = u$ **in** *spec*, *clarify*)
apply (*drule-tac* $x = s$ **in** *spec*, *clarify*)
apply (*subgoal-tac* $\forall n{::}nat.\ (Xa\ n) \neq x\ \&\ |(Xa\ n) + -x| < s \longrightarrow |f\ (Xa\ n) + -L| < u$)
prefer *2* **apply** *blast*
apply (*drule FreeUltrafilterNat-all*, *ultra*)
done

20.1.1 Limit: The NS definition implies the standard definition.

lemma *lemma-LIM*: $\forall s{>}0.\exists xa.\ \ xa \neq x\ \&$
$|xa + -x| < s\ \ \&\ r \leq |f\ xa + -L|$
$\Longrightarrow \forall n{::}nat.\ \exists xa.\ \ xa \neq x\ \&$
$|xa + -x| < inverse(real(Suc\ n))\ \&\ r \leq |f\ xa + -L|$
apply *clarify*
apply (*cut-tac* $n1 = n$ **in** *real-of-nat-Suc-gt-zero* [*THEN positive-imp-inverse-positive*], *auto*)
done

lemma *lemma-skolemize-LIM2*:
$\forall s{>}0.\exists xa.\ \ xa \neq x\ \&\ |xa + -x| < s\ \ \&\ r \leq |f\ xa + -L|$
$\Longrightarrow \exists X.\ \forall n{::}nat.\ X\ n \neq x\ \&$
$|X\ n + -x| < inverse(real(Suc\ n))\ \&\ r \leq abs(f\ (X\ n) + -L)$
apply (*drule lemma-LIM*)
apply (*drule choice*, *blast*)
done

lemma *lemma-simp*: $\forall n.\ X\ n \neq x\ \&$
$|X\ n + -x| < inverse\ (real(Suc\ n))\ \&$
$r \leq abs\ (f\ (X\ n) + -L) \Longrightarrow$
$\forall n.\ |X\ n + -x| < inverse\ (real(Suc\ n))$
by *auto*

NSLIM =¿ LIM

lemma *NSLIM-LIM*: $f \dashrightarrow x \longrightarrow_{NS} L \Longrightarrow f \dashrightarrow x \longrightarrow L$
apply (*simp add*: *LIM-def NSLIM-def approx-def*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff*, *clarify*)

apply (*rule ccontr*, *simp*)
apply (*simp add*: *linorder-not-less*)
apply (*drule lemma-skolemize-LIM2*, *safe*)
apply (*drule-tac x = star-n X* **in** *spec*)
apply (*auto simp add*: *starfun star-n-minus star-of-def star-n-add star-n-eq-iff*)
apply (*drule lemma-simp* [*THEN real-seq-to-hypreal-Infinitesimal*])
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff star-of-def star-n-minus star-n-add star-n-eq-iff*, *blast*)
apply (*drule spec*, *drule mp*, *assumption*)
apply (*drule FreeUltrafilterNat-all*, *ultra*)
done

theorem *LIM-NSLIM-iff*: $(f -- x --> L) = (f -- x --NS> L)$
by (*blast intro*: *LIM-NSLIM NSLIM-LIM*)

Proving properties of limits using nonstandard definition. The properties hold for standard limits as well!

lemma *NSLIM-mult*:
$[| f -- x --NS> l; g -- x --NS> m |] ==> (\%x.\ f(x) * g(x)) -- x --NS> (l * m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-mult-HFinite*)

lemma *LIM-mult2*:
$[| f -- x --> l; g -- x --> m |] ==> (\%x.\ f(x) * g(x)) -- x --> (l * m)$
by (*simp add*: *LIM-NSLIM-iff NSLIM-mult*)

lemma *NSLIM-add*:
$[| f -- x --NS> l; g -- x --NS> m |] ==> (\%x.\ f(x) + g(x)) -- x --NS> (l + m)$
by (*auto simp add*: *NSLIM-def intro*!: *approx-add*)

lemma *LIM-add2*:
$[| f -- x --> l; g -- x --> m |] ==> (\%x.\ f(x) + g(x)) -- x --> (l + m)$
by (*simp add*: *LIM-NSLIM-iff NSLIM-add*)

lemma *NSLIM-const* [*simp*]: $(\%x.\ k) -- x --NS> k$
by (*simp add*: *NSLIM-def*)

lemma *LIM-const2*: $(\%x.\ k) -- x --> k$
by (*simp add*: *LIM-NSLIM-iff*)

lemma *NSLIM-minus*: $f -- a --NS> L ==> (\%x.\ -f(x)) -- a --NS> -L$
by (*simp add*: *NSLIM-def*)

lemma *LIM-minus2*: $f -- a --> L ==> (\%x.\ -f(x)) -- a --> -L$

by (*simp add*: *LIM-NSLIM-iff NSLIM-minus*)

lemma *NSLIM-add-minus*: [| f $--$ x $--NS>$ l; g $--$ x $--NS>$ m |] ==> (%x. $f(x)$ + $-g(x)$) $--$ x $--NS>$ (l + $-m$)
by (*blast dest*: *NSLIM-add NSLIM-minus*)

lemma *LIM-add-minus2*: [| f $--$ x $-->$ l; g $--$ x $-->$ m |] ==> (%x. $f(x)$ + $-g(x)$) $--$ x $-->$ (l + $-m$)
by (*simp add*: *LIM-NSLIM-iff NSLIM-add-minus*)

lemma *NSLIM-inverse*:
[| 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*: *hypreal-of-real-approx-inverse*)
done

lemma *LIM-inverse*: [| f $--$ a $-->$ L; $L \neq 0$ |] ==> (%x. *inverse*($f(x)$)) $--$ a $-->$ (*inverse* L)
by (*simp add*: *LIM-NSLIM-iff NSLIM-inverse*)

lemma *NSLIM-zero*:
assumes f: f $--$ a $--NS>$ l **shows** (%x. $f(x)$ + $-l$) $--$ a $--NS>$ 0
proof $-$
have (λx. $f\ x$ + $-$ l) $--$ a $--NS>$ l + $-l$
by (*rule NSLIM-add-minus* [*OF f NSLIM-const*])
thus *?thesis* **by** *simp*
qed

lemma *LIM-zero2*: f $--$ a $-->$ l ==> (%x. $f(x)$ + $-l$) $--$ a $-->$ 0
by (*simp add*: *LIM-NSLIM-iff NSLIM-zero*)

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 *LIM-zero-cancel*: (%x. $f(x)$ $-$ l) $--$ x $-->$ 0 ==> f $--$ x $-->$ l
apply (*drule-tac* g = %x. l **and** M = l **in** *LIM-add*)
apply (*auto simp add*: *diff-minus add-assoc*)
done

lemma *NSLIM-not-zero*: $k \neq 0$ ==> $\sim$ ((%x. k) $--$ x $--NS>$ 0)
apply (*simp add*: *NSLIM-def*)

apply (*rule-tac* x = *hypreal-of-real* x + *epsilon* **in** *exI*)
apply (*auto intro*: *Infinitesimal-add-approx-self* [*THEN approx-sym*]
simp *add*: *hypreal-epsilon-not-zero*)
done

lemma *NSLIM-const-not-eq*: $k \neq L \Longrightarrow \sim ((\%x.\ k) -- x --NS> L)$
apply (*simp add*: *NSLIM-def*)
apply (*rule-tac* x = *hypreal-of-real* x + *epsilon* **in** *exI*)
apply (*auto intro*: *Infinitesimal-add-approx-self* [*THEN approx-sym*]
simp *add*: *hypreal-epsilon-not-zero*)
done

lemma *NSLIM-const-eq*: $(\%x.\ k) -- x --NS> L \Longrightarrow k = L$
apply (*rule ccontr*)
apply (*blast dest*: *NSLIM-const-not-eq*)
done

can actually be proved more easily by unfolding the definition!

lemma *NSLIM-unique*: $[|\ f -- x --NS> L; f -- x --NS> M\ |] \Longrightarrow L = M$
apply (*drule NSLIM-minus*)
apply (*drule NSLIM-add*, *assumption*)
apply (*auto dest*!: *NSLIM-const-eq* [*symmetric*])
done

lemma *LIM-unique2*: $[|\ f -- x --> L; f -- x --> M\ |] \Longrightarrow L = M$
by (*simp add*: *LIM-NSLIM-iff NSLIM-unique*)

lemma *NSLIM-mult-zero*: $[|\ f -- x --NS> 0; g -- x --NS> 0\ |] \Longrightarrow (\%x.\ f(x)*g(x)) -- x --NS> 0$
by (*drule NSLIM-mult*, *auto*)

lemma *LIM-mult-zero2*: $[|\ f -- x --> 0; g -- x --> 0\ |] \Longrightarrow (\%x.\ f(x)*g(x)) -- x --> 0$
by (*drule LIM-mult2*, *auto*)

lemma *NSLIM-self*: $(\%x.\ x) -- a --NS> a$
by (*simp add*: *NSLIM-def*)

20.2 Derivatives and Continuity: NS and Standard properties

20.2.1 Continuity

lemma *isNSContD*: [| *isNSCont f a*; *y* ≈ *hypreal-of-real a* |] ==> (∗*f*∗ *f*) *y* ≈ *hypreal-of-real* (*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 (*rule-tac Q* = *y* = *hypreal-of-real a* **in** *excluded-middle* [*THEN disjE*], *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* ==> *isNSCont f a*
by (*erule isNSCont-isCont-iff* [*THEN iffD2*])

NS continuity ==¿ Standard continuity

lemma *isNSCont-isCont*: *isNSCont f a* ==> *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* = *hypreal-of-real a* + *x* **in** *spec*)
apply (*drule-tac* [*2*] *x* = −*hypreal-of-real a* + *x* **in** *spec*, *safe*, *simp*)
apply (*rule mem-infmal-iff* [*THEN iffD2*, *THEN Infinitesimal-add-approx-self* [*THEN approx-sym*]])

apply (*rule-tac* [*4*] *approx-minus-iff2* [*THEN iffD1*])
prefer *3* **apply** (*simp add*: *add-commute*)
apply (*rule-tac* [*2*] *x* = *x* **in** *star-cases*)
apply (*rule-tac* [*4*] *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 *LIM-isCont-iff*: (*f* −− *a* −−> *f a*) = ((%*h*. *f*(*a* + *h*)) −− *0* −−> *f*(*a*))
by (*simp add*: *LIM-NSLIM-iff NSLIM-isCont-iff*)

lemma *isCont-iff*: (*isCont f x*) = ((%*h*. *f*(*x* + *h*)) −− *0* −−> *f*(*x*))
by (*simp add*: *isCont-def LIM-isCont-iff*)

Immediate application of nonstandard criterion for continuity can offer very simple proofs of some standard property of continuous functions

sum continuous

lemma *isCont-add*: [| *isCont f a*; *isCont g a* |] ==> *isCont* (%*x*. *f*(*x*) + *g*(*x*)) *a*
by (*auto intro*: *approx-add simp add*: *isNSCont-isCont-iff* [*symmetric*] *isNSCont-def*)

mult continuous

lemma *isCont-mult*: [| *isCont f a*; *isCont g a* |] ==> *isCont* (%*x*. *f*(*x*) * *g*(*x*)) *a*
by (*auto intro*!: *starfun-mult-HFinite-approx*
simp del: *starfun-mult* [*symmetric*]
simp add: *isNSCont-isCont-iff* [*symmetric*] *isNSCont-def*)

composition of continuous functions Note very short straightforard proof!

lemma *isCont-o*: [| *isCont f a*; *isCont g* (*f a*) |] ==> *isCont* (*g o f*) *a*
by (*auto simp add*: *isNSCont-isCont-iff* [*symmetric*] *isNSCont-def starfun-o* [*symmetric*])

lemma *isCont-o2*: [| *isCont f a*; *isCont g* (*f a*) |] ==> *isCont* (%*x*. *g* (*f x*)) *a*
by (*auto dest*: *isCont-o simp add*: *o-def*)

lemma *isNSCont-minus*: *isNSCont f a* ==> *isNSCont* (%*x*. − *f x*) *a*
by (*simp add*: *isNSCont-def*)

lemma *isCont-minus*: *isCont f a* ==> *isCont* (%*x*. − *f x*) *a*
by (*auto simp add*: *isNSCont-isCont-iff* [*symmetric*] *isNSCont-minus*)

lemma *isCont-inverse*:
[| *isCont f x*; *f x* ≠ *0* |] ==> *isCont* (%*x*. *inverse* (*f x*)) *x*
apply (*simp add*: *isCont-def*)
apply (*blast intro*: *LIM-inverse*)
done

lemma *isNSCont-inverse*: [| *isNSCont f x*; *f x* ≠ *0* |] ==> *isNSCont* (%*x*. *inverse* (*f x*)) *x*
by (*auto intro*: *isCont-inverse simp add*: *isNSCont-isCont-iff*)

lemma *isCont-diff*:
[| *isCont f a*; *isCont g a* |] ==> *isCont* (%*x*. *f*(*x*) − *g*(*x*)) *a*
apply (*simp add*: *diff-minus*)
apply (*auto intro*: *isCont-add isCont-minus*)
done

lemma *isCont-const* [*simp*]: *isCont* (%*x*. *k*) *a*
by (*simp add*: *isCont-def*)

lemma *isNSCont-const* [*simp*]: *isNSCont* (%*x*. *k*) *a*
by (*simp add*: *isNSCont-def*)

lemma *isNSCont-abs* [*simp*]: *isNSCont abs a*
apply (*simp add*: *isNSCont-def*)
apply (*auto intro*: *approx-hrabs simp add*: *hypreal-of-real-hrabs* [*symmetric*] *starfun-rabs-hrabs*)
done

lemma *isCont-abs* [*simp*]: *isCont abs a*
by (*auto simp add*: *isNSCont-isCont-iff* [*symmetric*])

Uniform continuity

lemma *isNSUContD*: [| *isNSUCont f*; *x* ≈ *y*|] ==> (∗*f*∗ *f*) *x* ≈ (∗*f*∗ *f*) *y*
by (*simp add*: *isNSUCont-def*)

lemma *isUCont-isCont*: *isUCont f* ==> *isCont f x*
by (*simp add*: *isUCont-def isCont-def LIM-def*, *meson*)

lemma *isUCont-isNSUCont*: *isUCont f* ==> *isNSUCont f*
apply (*simp add*: *isNSUCont-def isUCont-def approx-def*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff*, *safe*)
apply (*rule-tac x* = *x* **in** *star-cases*)
apply (*rule-tac x* = *y* **in** *star-cases*)
apply (*auto simp add*: *starfun star-n-minus star-n-add*)
apply (*rule bexI* [*OF - Rep-star-star-n*], *safe*)
apply (*drule-tac x* = *u* **in** *spec*, *clarify*)
apply (*drule-tac x* = *s* **in** *spec*, *clarify*)
apply (*subgoal-tac* ∀ *n*::*nat*. *abs* ((*Xa n*) + − (*Xb n*)) < *s* −−> *abs* (*f* (*Xa n*) + − *f* (*Xb n*)) < *u*)
prefer *2* **apply** *blast*
apply (*erule-tac V* = ∀ *x y*. |*x* + − *y*| < *s* −−> |*f x* + − *f y*| < *u* **in** *thin-rl*)
apply (*drule FreeUltrafilterNat-all*, *ultra*)
done

lemma *lemma-LIMu*: ∀ *s*>*0*.∃ *z y*. |*z* + − *y*| < *s* & *r* ≤ |*f z* + −*f y*|

$==> \forall n{::}nat.\ \exists z\ y.\ |z + -y| < inverse(real(Suc\ n))\ \&\ r \leq |f\ z + -f\ y|$
apply *clarify*
apply (*cut-tac n1 = n*
in *real-of-nat-Suc-gt-zero* [*THEN positive-imp-inverse-positive*], *auto*)
done

lemma *lemma-skolemize-LIM2u*:
$\forall s{>}0.\exists z\ y.\ |z + -\ y| < s\ \ \&\ r \leq |f\ z + -f\ y|$
$==> \exists X\ Y.\ \forall n{::}nat.$
$abs(X\ n + -(Y\ n)) < inverse(real(Suc\ n))\ \&$
$r \leq abs(f\ (X\ n) + -f\ (Y\ n))$
apply (*drule lemma-LIMu*)
apply (*drule choice*, *safe*)
apply (*drule choice*, *blast*)
done

lemma *lemma-simpu*: $\forall n.\ |X\ n + -Y\ n| < inverse\ (real(Suc\ n))\ \&$
$r \leq abs\ (f\ (X\ n) + -\ f(Y\ n)) ==>$
$\forall n.\ |X\ n + -\ Y\ n| < inverse\ (real(Suc\ n))$
by *auto*

lemma *isNSUCont-isUCont*:
$isNSUCont\ f ==> isUCont\ f$
apply (*simp add*: *isNSUCont-def isUCont-def approx-def*)
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff*, *safe*)
apply (*rule ccontr*, *simp*)
apply (*simp add*: *linorder-not-less*)
apply (*drule lemma-skolemize-LIM2u*, *safe*)
apply (*drule-tac x = star-n X* **in** *spec*)
apply (*drule-tac x = star-n Y* **in** *spec*)
apply (*simp add*: *starfun star-n-minus star-n-add*, *auto*)
apply (*drule lemma-simpu* [*THEN real-seq-to-hypreal-Infinitesimal2*])
apply (*simp add*: *Infinitesimal-FreeUltrafilterNat-iff star-n-minus star-n-add*, *blast*)
apply (*drule-tac x = r* **in** *spec*, *clarify*)
apply (*drule FreeUltrafilterNat-all*, *ultra*)
done

Derivatives

lemma *DERIV-iff*: $(DERIV\ f\ x\ :>\ D) = ((\%h.\ (f(x + h) + -\ f(x))/h)\ --\ 0\ --> D)$
by (*simp add*: *deriv-def*)

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 *DERIV-D*: $DERIV\ f\ x\ :>\ D ==> (\%h.\ (f(x + h) + -\ f(x))/h)\ --\ 0\ --> D$
by (*simp add*: *deriv-def*)

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*)

20.2.2 Uniqueness

lemma *DERIV-unique*:
[| *DERIV f x :> D*; *DERIV f x :> E* |] ==> *D = E*
apply (*simp add*: *deriv-def*)
apply (*blast intro*: *LIM-unique*)
done

lemma *NSDeriv-unique*:
[| *NSDERIV f x :> D*; *NSDERIV f x :> E* |] ==> *D = E*
apply (*simp add*: *nsderiv-def*)
apply (*cut-tac Infinitesimal-epsilon hypreal-epsilon-not-zero*)
apply (*auto dest*!: *bspec* [**where** *x=epsilon*]
intro!: *inj-hypreal-of-real* [*THEN injD*]
dest: *approx-trans3*)
done

20.2.3 Differentiable

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 *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*)

20.2.4 Alternative definition for differentiability

lemma *LIM-I*:
(!!*r*. *0<r ==> ∃ s>0.∀ x*. *x ≠ a & |x-a| < s --> |f x - L| < r*)
==> f -- a --> L
by (*simp add*: *LIM-eq*)

lemma *DERIV-LIM-iff*:
((%*h*. (*f*(*a* + *h*) - *f*(*a*)) / *h*) -- *0* --> *D*) =
((%*x*. (*f*(*x*)-*f*(*a*)) / (*x*-*a*)) -- *a* --> *D*)
proof (*intro iffI LIM-I*)
fix *r*::*real*
assume *r*: *0<r*
assume (λ*h*. (*f* (*a* + *h*) - *f a*) / *h*) -- *0* --> *D*

from *LIM-D* [*OF this r*]
obtain *s*
where *s*: $0 < s$
and *s-lt*: $\forall x.\ x \neq 0\ \&\ |x| < s \longrightarrow |(f\ (a + x) - f\ a) / x - D| < r$
by *auto*
show $\exists s.\ 0 < s \wedge$
$(\forall x.\ x \neq a \wedge |x-a| < s \longrightarrow |(f\ x - f\ a) / (x-a) - D| < r)$
proof (*intro exI conjI strip*)
show $0 < s$ **by** (*rule s*)
next
fix *x*::*real*
assume $x \neq a \wedge |x-a| < s$
with *s-lt* [*THEN spec* [**where** *x*=*x*−*a*]]
show $|(f\ x - f\ a) / (x-a) - D| < r$ **by** *auto*
qed
next
fix *r*::*real*
assume *r*: $0<r$
assume $(\lambda x.\ (f\ x - f\ a) / (x-a)) \dashrightarrow a \longrightarrow D$
from *LIM-D* [*OF this r*]
obtain *s*
where *s*: $0 < s$
and *s-lt*: $\forall x.\ x \neq a\ \&\ |x-a| < s \longrightarrow |(f\ x - f\ a)/(x-a) - D| < r$
by *auto*
show $\exists s.\ 0 < s \wedge$
$(\forall x.\ x \neq 0\ \&\ |x - 0| < s \longrightarrow |(f\ (a + x) - f\ a) / x - D| < r)$
proof (*intro exI conjI strip*)
show $0 < s$ **by** (*rule s*)
next
fix *x*::*real*
assume $x \neq 0 \wedge |x - 0| < s$
with *s-lt* [*THEN spec* [**where** *x*=*x*+*a*]]
show $|(f\ (a + x) - f\ a) / x - D| < r$ **by** (*auto simp add*: *add-ac*)
qed
qed

lemma *DERIV-iff2*: $(DERIV\ f\ x :> D) = ((\%z.\ (f(z) - f(x)) / (z-x)) \dashrightarrow x \longrightarrow D)$
by (*simp add*: *deriv-def diff-minus* [*symmetric*] *DERIV-LIM-iff*)

20.3 Equivalence of NS and standard definitions of differentiation

20.3.1 First NSDERIV in terms of NSLIM

first equivalence

lemma *NSDERIV-NSLIM-iff*:
$(NSDERIV\ f\ x :> D) = ((\%h.\ (f(x + h) + - f(x))/h) \dashrightarrow 0\ --NS> D)$
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 ≠ *hypreal-of-real x* & *w* ≈ *hypreal-of-real x* −−>
(∗*f*∗ (%*z*. (*f z* − *f x*) / (*z*−*x*))) *w* ≈ *hypreal-of-real D*)
by (*simp add*: *NSDERIV-NSLIM-iff2 NSLIM-def*)

lemma *hypreal-not-eq-minus-iff*: (*x* ≠ *a*) = (*x* + −*a* ≠ (*0*::*hypreal*))
by (*auto dest*: *hypreal-eq-minus-iff* [*THEN iffD2*])

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*: *diff-minus*
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

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*)

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* = −*hypreal-of-real x* + *xa* **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* + −*hypreal-of-real x* **in** *approx-mult1*)
apply (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
simp add: *mult-assoc*)
apply (*drule-tac x3*=*D* **in**
HFinite-hypreal-of-real [*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

Now Sandard proof

lemma *DERIV-isCont*: *DERIV f x :> D ==> isCont f x*
by (*simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *isNSCont-isCont-iff* [*symmetric*]
NSDERIV-isNSCont)

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*)

lemma *DERIV-const* [*simp*]: (*DERIV* (%*x*. *k*) *x* :> *0*)
by (*simp add*: *NSDERIV-DERIV-iff* [*symmetric*])

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 dest*!: *spec*)
apply (*drule-tac b* = *hypreal-of-real Da* **and** *d* = *hypreal-of-real Db* **in** *approx-add*)
apply (*auto simp add*: *add-ac*)
done

lemma *DERIV-add*: [| *DERIV f x* :> *Da*; *DERIV g x* :> *Db* |]
==> *DERIV* (%*x*. *f x* + *g x*) *x* :> *Da* + *Db*
apply (*simp add*: *NSDERIV-add NSDERIV-DERIV-iff* [*symmetric*])
done

Product of functions - Proof is trivial but tedious and long due to rearrangement of terms

lemma *lemma-nsderiv1*: ((*a*::*hypreal*)∗*b*) + −(*c*∗*d*) = (*b*∗(*a* + −*c*)) + (*c*∗(*b* + −*d*))
by (*simp add*: *right-distrib*)

lemma *lemma-nsderiv2*: [| (*x* + *y*) / *z* = *hypreal-of-real D* + *yb*; *z* ≠ *0*;
z ∈ *Infinitesimal*; *yb* ∈ *Infinitesimal* |]
==> *x* + *y* ≈ *0*
apply (*frule-tac c1* = *z* **in** *hypreal-mult-right-cancel* [*THEN iffD2*], *assumption*)
apply (*erule-tac V* = (*x* + *y*) / *z* = *hypreal-of-real D* + *yb* **in** *thin-rl*)
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*)
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*)
apply (*drule-tac* [*4*]
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* = *hypreal-of-real Db* ∗ *hypreal-of-real* (*f x*)
in *add-commute* [*THEN subst*])
apply (*auto intro*!: *Infinitesimal-add-approx-self2* [*THEN approx-sym*]
Infinitesimal-add Infinitesimal-mult
Infinitesimal-hypreal-of-real-mult
Infinitesimal-hypreal-of-real-mult2
simp add: *add-assoc* [*symmetric*])
done

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 (*simp add*: *NSDERIV-mult NSDERIV-DERIV-iff* [*symmetric*])

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-distrib [*symmetric*])
apply (*erule NSLIM-const* [*THEN NSLIM-mult*])
done

lemma *DERIV-cmult*:
DERIV f x :> *D* ==> *DERIV* (%*x*. *c* ∗ *f x*) *x* :> *c*∗*D*
apply (*simp only*: *deriv-def times-divide-eq-right* [*symmetric*]
NSDERIV-NSLIM-iff minus-mult-right right-distrib [*symmetric*])
apply (*erule LIM-const* [*THEN LIM-mult2*])
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*)
with *deriv*
show (λ*h*. (− *f* (*x* + *h*) + *f x*) / *h*) −− *0* −−*NS*> − *D* **by** *simp*
qed

lemma *DERIV-minus*: *DERIV f x* :> *D* ==> *DERIV* (%*x*. −(*f x*)) *x* :> −*D*

by (*simp add*: *NSDERIV-minus NSDERIV-DERIV-iff* [*symmetric*])

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 *DERIV-add-minus*: [| *DERIV f x :> Da*; *DERIV g x :> Db* |] ==> *DERIV* (%*x*. *f x* + −*g x*) *x :> Da* + −*Db*
by (*blast dest*: *DERIV-add DERIV-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

lemma *DERIV-diff*:
[| *DERIV f x :> Da*; *DERIV g x :> Db* |]
==> *DERIV* (%*x*. *f x* − *g x*) *x :> Da*−*Db*
apply (*simp add*: *diff-minus*)
apply (*blast intro*: *DERIV-add-minus*)
done

(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

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*) (*hypreal-of-real*(*x*) + *xa*) = *hypreal-of-real*(*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*) (*hypreal-of-real*(*x*) + *h*) + −*hypreal-of-real*(*f x*) ≈ *0*
apply (*simp add*: *nsderiv-def*)
apply (*simp add*: *mem-infmal-iff* [*symmetric*])
apply (*rule Infinitesimal-ratio*)
apply (*rule-tac* [*3*] *approx-hypreal-of-real-HFinite*, *auto*)
done

lemma *NSDERIVD1*: [| *NSDERIV f* (*g x*) :> *Da*;
(∗*f*∗ *g*) (*hypreal-of-real*(*x*) + *xa*) ≠ *hypreal-of-real* (*g x*);
(∗*f*∗ *g*) (*hypreal-of-real*(*x*) + *xa*) ≈ *hypreal-of-real* (*g x*)
|] ==> ((∗*f*∗ *f*) ((∗*f*∗ *g*) (*hypreal-of-real*(*x*) + *xa*))

$+ - \mathit{hypreal\text{-}of\text{-}real}\ (f\ (g\ x)))$
$/\ ((\ {*f*}\ g)\ (\mathit{hypreal\text{-}of\text{-}real}(x) + xa) + - \mathit{hypreal\text{-}of\text{-}real}\ (g\ x))$
$\approx \mathit{hypreal\text{-}of\text{-}real}(Da)$
by (*auto simp add*: *NSDERIV-NSLIM-iff2 NSLIM-def diff-minus* [*symmetric*])

lemma *NSDERIVD2*: [| *NSDERIV g x :> Db*; *xa* ∈ *Infinitesimal*; *xa* ≠ *0* |]
==> ((*f* g) (*hypreal-of-real*(*x*) + *xa*) + - *hypreal-of-real*(*g x*)) / *xa*
≈ *hypreal-of-real*(*Db*)
by (*auto simp add*: *NSDERIV-NSLIM-iff NSLIM-def mem-infmal-iff starfun-lambda-cancel*)

lemma *lemma-chain*: (*z*::*hypreal*) ≠ *0* ==> *x***y* = (*x***inverse*(*z*))*(*z***y*)
by *auto*

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) (*hypreal-of-real* (*x*) + *xa*) = *hypreal-of-real* (*g x*))
apply (*drule-tac g* = *g* **in** *NSDERIV-zero*)
apply (*auto simp add*: *divide-inverse*)
apply (*rule-tac z1* = (*f* g) (*hypreal-of-real* (*x*) + *xa*) + −*hypreal-of-real* (*g x*)
and *y1* = *inverse xa* **in** *lemma-chain* [*THEN ssubst*])
apply (*erule hypreal-not-eq-minus-iff* [*THEN iffD1*])
apply (*rule approx-mult-hypreal-of-real*)
apply (*simp-all add*: *divide-inverse* [*symmetric*])
apply (*blast intro*: *NSDERIVD1 approx-minus-iff* [*THEN iffD2*])
apply (*blast intro*: *NSDERIVD2*)
done

lemma *DERIV-chain*: [| *DERIV f* (*g x*) :> *Da*; *DERIV g x* :> *Db* |] ==> *DERIV*
(*f o g*) *x* :> *Da* * *Db*
by (*simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *NSDERIV-chain*)

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*)

Differentiation of natural number powers

lemma *NSDERIV-Id* [*simp*]: *NSDERIV* (%*x*. *x*) *x* :> *1*
by (*simp add*: *NSDERIV-NSLIM-iff NSLIM-def divide-self del*: *divide-self-if*)

lemma *DERIV-Id* [*simp*]: *DERIV* (%*x*. *x*) *x* :> *1*

by (*simp add*: *NSDERIV-DERIV-iff* [*symmetric*])

lemmas *isCont-Id* = *DERIV-Id* [*THEN DERIV-isCont*, *standard*]

lemma *DERIV-cmult-Id* [*simp*]: *DERIV* (*op* ∗ *c*) *x* :> *c*
by (*cut-tac c* = *c* **and** *x* = *x* **in** *DERIV-Id* [*THEN DERIV-cmult*], *simp*)

lemma *NSDERIV-cmult-Id* [*simp*]: *NSDERIV* (*op* ∗ *c*) *x* :> *c*
by (*simp add*: *NSDERIV-DERIV-iff*)

lemma *DERIV-pow*: *DERIV* (%*x*. *x* ^ *n*) *x* :> *real n* ∗ (*x* ^ (*n* − *Suc 0*))
apply (*induct n*)
apply (*drule-tac* [*2*] *DERIV-Id* [*THEN DERIV-mult*])
apply (*auto simp add*: *real-of-nat-Suc left-distrib*)
apply (*case-tac 0* < *n*)
apply (*drule-tac x* = *x* **in** *realpow-minus-mult*)
apply (*auto simp add*: *mult-assoc add-commute*)
done

lemma *NSDERIV-pow*: *NSDERIV* (%*x*. *x* ^ *n*) *x* :> *real n* ∗ (*x* ^ (*n* − *Suc 0*))
by (*simp add*: *NSDERIV-DERIV-iff DERIV-pow*)

Power of -1

lemma *NSDERIV-inverse*:
  x ≠ *0* ==> *NSDERIV* (%*x*. *inverse*(*x*)) *x* :> (− (*inverse x* ^ *Suc* (*Suc 0*)))
apply (*simp add*: *nsderiv-def*)
apply (*rule ballI*, *simp*, *clarify*)
apply (*frule Infinitesimal-add-not-zero*)
prefer *2* **apply** (*simp add*: *add-commute*)
apply (*auto simp add*: *starfun-inverse-inverse realpow-two*
    simp del: *minus-mult-left* [*symmetric*] *minus-mult-right* [*symmetric*])
apply (*simp add*: *inverse-add inverse-mult-distrib* [*symmetric*]
      inverse-minus-eq [*symmetric*] *add-ac mult-ac*
    del: *inverse-mult-distrib inverse-minus-eq*
      minus-mult-left [*symmetric*] *minus-mult-right* [*symmetric*])
apply (*simp* (*no-asm-simp*) *add*: *mult-assoc* [*symmetric*] *right-distrib*
    del: *minus-mult-left* [*symmetric*] *minus-mult-right* [*symmetric*])
apply (*rule-tac y* = *inverse* (− *hypreal-of-real x* ∗ *hypreal-of-real 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-mult2*, *auto*)
done

lemma *DERIV-inverse*: $x \neq 0 \Longrightarrow DERIV\ (\%x.\ inverse(x))\ x :> (-(inverse\ x\ \hat{}\ Suc\ (Suc\ 0)))$
by (*simp add*: *NSDERIV-inverse NSDERIV-DERIV-iff* [*symmetric*] *del*: *realpow-Suc*)

Derivative of inverse

lemma *DERIV-inverse-fun*: [| $DERIV\ f\ x :> d;\ f(x) \neq 0$ |]
$\Longrightarrow DERIV\ (\%x.\ inverse(f\ x))\ x :> (-\ (d * inverse(f(x)\ \hat{}\ Suc\ (Suc\ 0))))$
apply (*simp only*: *mult-commute* [*of d*] *minus-mult-left power-inverse*)
apply (*fold o-def*)
apply (*blast intro*!: *DERIV-chain DERIV-inverse*)
done

lemma *NSDERIV-inverse-fun*: [| $NSDERIV\ f\ x :> d;\ f(x) \neq 0$ |]
$\Longrightarrow NSDERIV\ (\%x.\ inverse(f\ x))\ x :> (-\ (d * inverse(f(x)\ \hat{}\ Suc\ (Suc\ 0))))$
by (*simp add*: *NSDERIV-DERIV-iff DERIV-inverse-fun del*: *realpow-Suc*)

Derivative of quotient

lemma *DERIV-quotient*: [| $DERIV\ f\ x :> d;\ DERIV\ g\ x :> e;\ g(x) \neq 0$ |]
$\Longrightarrow DERIV\ (\%y.\ f(y)\ /\ (g\ y))\ x :> (d*g(x) + -(e*f(x)))\ /\ (g(x)\ \hat{}\ Suc\ (Suc\ 0))$
apply (*drule-tac f* = *g* **in** *DERIV-inverse-fun*)
apply (*drule-tac* [*2*] *DERIV-mult*)
apply (*assumption*+)
apply (*simp add*: *divide-inverse right-distrib power-inverse minus-mult-left mult-ac*
del: *realpow-Suc minus-mult-right* [*symmetric*] *minus-mult-left* [*symmetric*])
done

lemma *NSDERIV-quotient*: [| $NSDERIV\ f\ x :> d;\ DERIV\ g\ x :> e;\ g(x) \neq 0$ |]
$\Longrightarrow NSDERIV\ (\%y.\ f(y)\ /\ (g\ y))\ x :> (d*g(x) + -(e*f(x)))\ /\ (g(x)\ \hat{}\ Suc\ (Suc\ 0))$
by (*simp add*: *NSDERIV-DERIV-iff DERIV-quotient del*: *realpow-Suc*)

lemma *CARAT-DERIV*:
$(DERIV\ f\ x :> l) =$
$(\exists g.\ (\forall z.\ f\ z - f\ x = g\ z * (z-x))\ \&\ isCont\ g\ x\ \&\ g\ x = l)$
(**is** *?lhs* = *?rhs*)
proof
assume *der*: $DERIV\ f\ x :> l$
show $\exists g.\ (\forall z.\ f\ z - f\ x = g\ z * (z-x)) \wedge isCont\ g\ x \wedge g\ x = l$
proof (*intro exI conjI*)
let *?g* = $(\%z.\ if\ z = x\ then\ l\ else\ (f\ z - f\ x)\ /\ (z-x))$
show $\forall z.\ f\ z - f\ x = ?g\ z * (z-x)$ **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 diff-minus
          cong: LIM-equal [rule-format])
qed


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)

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 ≠ hypreal-of-real x ∧ w ≈ hypreal-of-real x ⟶
       ( ∗f∗ g) w ∗ (w − hypreal-of-real x) / (w − hypreal-of-real x) ≈
       hypreal-of-real (g x)
   by (simp add: diff-minus isNSCont-def)
 thus ?thesis using all
   by (simp add: NSDERIV-iff2 starfun-if-eq cong: if-cong)
qed
```

Lemmas about nested intervals and proof by bisection (cf.Harrison). All considerably tidied by lcp.

```
lemma lemma-f-mono-add [rule-format (no-asm)]: (∀n. (f::nat=>real) n ≤ f
(Suc n)) −−> f m ≤ f(m + no)
apply (induct no)
apply (auto intro: order-trans)
done

lemma f-inc-g-dec-Beq-f: [| ∀n. f(n) ≤ f(Suc n);
       ∀n. g(Suc n) ≤ g(n);
       ∀n. f(n) ≤ g(n) |]
     ==> Bseq f
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*
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*: [| $\forall n.\ f\ n \leq f\ (Suc\ n)$; *convergent f* |] ==> $f\ n \leq lim\ f$
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* $P = \%na.\ no \leq na$ --> *?Q na* **and** $x = no + n$ **in** *spec*, *auto*)
apply (*subgoal-tac* $lim\ f \leq f\ (no + n)$)
apply (*induct-tac* [*2*] *no*)
apply (*auto intro*: *order-trans simp add*: *diff-minus abs-if*)
apply (*drule-tac no=no* **and** *m=n* **in** *lemma-f-mono-add*)
apply (*auto simp add*: *add-commute*)
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*: [| $\forall n.\ g(Suc\ n) \leq g(n)$; *convergent g* |] ==> $lim\ g \leq g\ n$
apply (*subgoal-tac* $-\ (g\ n) \leq -\ (lim\ g)$)
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*: [| $\forall n.\ f(n) \leq f(Suc\ n)$;
$\forall n.\ g(Suc\ n) \leq g(n)$;
$\forall n.\ f(n) \leq g(n)$ |]
==> $\exists l\ m.\ l \leq m\ \&\ ((\forall n.\ f(n) \leq l)\ \&\ f$ ----> $l)\ \&$
$((\forall n.\ m \leq g(n))\ \&\ g$ ----> $m)$
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*: [| $\forall n.\ f(n) \leq f(Suc\ n)$;
$\forall n.\ g(Suc\ n) \leq g(n)$;
$\forall n.\ f(n) \leq g(n)$;
$(\%n.\ f(n) - g(n)) \longrightarrow 0$ |]
$\Longrightarrow \exists l.\ ((\forall n.\ f(n) \leq l)\ \&\ f \longrightarrow l)\ \&$
$((\forall n.\ l \leq g(n))\ \&\ g \longrightarrow l)$
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 ≤ b ==>
     snd(Bolzano-bisect P a b n) − fst(Bolzano-bisect P a b n) =
     (b−a) / (2 ^ 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 ≤ b; b ≤ c|] ==> P(a,c)
      and notP: ~ P(a,b)
      and le:   a ≤ b
  shows ~ P(fst(Bolzano-bisect P a b n), snd(Bolzano-bisect P a b n))
proof (induct n)
  case 0 thus ?case 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*) ≤ *y*; *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* $x \leq aa$, *simp-all*)
done

lemma *IVT2*: [| $f(b) \leq y$; $y \leq f(a)$;
$a \leq b$;
$(\forall x.\ a \leq x\ \&\ x \leq b\ \text{--->}\ isCont\ f\ x)$
|] ==> $\exists x.\ a \leq x\ \&\ x \leq b\ \&\ f(x) = y$
apply (*subgoal-tac* $-f\,a \leq -y\ \&\ -y \leq -f\,b$, *clarify*)
apply (*drule IVT* [**where** $f = \%x.\ -f\,x$], *assumption*)
apply (*auto intro*: *isCont-minus*)
done

lemma *IVT-objl*: $(f(a) \leq y\ \&\ y \leq f(b)\ \&\ a \leq b\ \&$
$(\forall x.\ a \leq x\ \&\ x \leq b\ \text{--->}\ isCont\ f\ x))$
$\text{--->}\ (\exists x.\ a \leq x\ \&\ x \leq b\ \&\ f(x) = y)$
apply (*blast intro*: *IVT*)
done

lemma *IVT2-objl*: $(f(b) \leq y\ \&\ y \leq f(a)\ \&\ a \leq b\ \&$
$(\forall x.\ a \leq x\ \&\ x \leq b\ \text{--->}\ isCont\ f\ x))$
$\text{--->}\ (\exists x.\ a \leq x\ \&\ x \leq b\ \&\ f(x) = y)$
apply (*blast intro*: *IVT2*)
done

20.5 By bisection, function continuous on closed interval is bounded above

lemma *isCont-bounded*:
[| $a \leq b$; $\forall x.\ a \leq x\ \&\ x \leq b\ \text{--->}\ isCont\ f\ x$ |]
==> $\exists M.\ \forall x.\ a \leq x\ \&\ x \leq b\ \text{--->}\ f(x) \leq M$
apply (*cut-tac* $P = \%\ (u,v)\ .\ a \leq u\ \&\ u \leq v\ \&\ v \leq b\ \text{--->}\ (\exists M.\ \forall x.\ u \leq x\ \&$
$x \leq v\ \text{--->}\ f\,x \leq M)$ **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* $V = \forall M.\ \exists x.\ a \le x\ \&\ x \le b\ \&\ \sim f\,x \le M$ **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*, *arith+*)
done

Refine the above to existence of least upper bound

lemma *lemma-reals-complete*: $((\exists x.\ x \in S)\ \&\ (\exists y.\ isUb\ UNIV\ S\ (y::real))) \dashrightarrow$
$(\exists t.\ isLub\ UNIV\ S\ t)$
by (*blast intro*: *reals-complete*)

lemma *isCont-has-Ub*: $[|\ a \le b;\ \forall x.\ a \le x\ \&\ x \le b \dashrightarrow isCont\ f\ x\ |]$
$\Longrightarrow \exists M.\ (\forall x.\ a \le x\ \&\ x \le b \dashrightarrow f(x) \le M)\ \&$
$(\forall N.\ N < M \dashrightarrow (\exists x.\ a \le x\ \&\ x \le b\ \&\ N < f(x)))$
apply (*cut-tac* $S = Collect\ (\%y.\ \exists x.\ a \le x\ \&\ x \le b\ \&\ y = f\,x)$
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*: $a \le b$
and *con*: $\forall x.\ a \le x\ \&\ x \le b \dashrightarrow isCont\ f\ x$
shows $\exists M.\ (\forall x.\ a \le x\ \&\ x \le b \dashrightarrow f(x) \le M)\ \&$
$(\exists x.\ a \le x\ \&\ x \le b\ \&\ f(x) = M)$
proof −
from *isCont-has-Ub* [*OF le con*]
obtain M **where** *M1*: $\forall x.\ a \le x \wedge x \le b \longrightarrow f\,x \le M$
and *M2*: $!!N.\ N<M \Longrightarrow \exists x.\ a \le x \wedge x \le b \wedge N < f\,x$ **by** *blast*
show *?thesis*
proof (*intro exI*, *intro conjI*)
show $\forall x.\ a \le x \wedge x \le b \longrightarrow f\,x \le M$ **by** (*rule M1*)
show $\exists x.\ a \le x \wedge x \le b \wedge f\,x = M$
proof (*rule ccontr*)
assume $\neg\ (\exists x.\ a \le x \wedge x \le b \wedge f\,x = M)$
with *M1* **have** *M3*: $\forall x.\ a \le x\ \&\ x \le b \dashrightarrow f\,x < M$
by (*fastsimp simp add*: *linorder-not-le* [*symmetric*])
hence $\forall x.\ a \le x\ \&\ x \le b \dashrightarrow isCont\ (\%x.\ inverse\ (M - f\,x))\ x$
by (*auto simp add*: *isCont-inverse isCont-diff con*)
from *isCont-bounded* [*OF le this*]
obtain k **where** k: $!!x.\ a \le x\ \&\ x \le b \dashrightarrow inverse\ (M - f\,x) \le k$ **by** *auto*

```
    have Minv: !!x. a ≤ x & x ≤ b --> 0 < inverse (M - f (x))
      by (simp add: M3 compare-rls)
    have !!x. a ≤ x & x ≤ b --> inverse (M - f x) < k+1 using k
      by (auto intro: order-le-less-trans [of - k])
    with Minv
    have !!x. a ≤ x & x ≤ b --> inverse(k+1) < inverse(inverse(M - f x))
      by (intro strip less-imp-inverse-less, simp-all)
    hence invlt: !!x. a ≤ x & x ≤ 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 ≤ x & x ≤ 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 ≤ b; ∀x. a ≤ x & x ≤ b --> isCont f x |]
        ==> ∃M. (∀x. a ≤ x & x ≤ b --> M ≤ f(x)) &
                (∃x. a ≤ x & x ≤ b & f(x) = M)
apply (subgoal-tac ∀x. a ≤ x & x ≤ b --> 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 ≤ b; ∀x. a ≤ x & x ≤ b --> isCont f x |]
     ==> ∃L M. (∀x. a ≤ x & x ≤ b --> L ≤ f(x) & f(x) ≤ M) &
         (∀y. L ≤ y & y ≤ M --> (∃x. a ≤ x & x ≤ 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
```

20.6 If $(0::'a) < f'\,x$ then x is Locally Strictly Increasing At The Right

```
lemma DERIV-left-inc:
```

```
  assumes der: DERIV f x :> l
      and l:   0 < l
  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 .
    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-left-dec:
  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 .
    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:
  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 show ?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 ∧ e < d ∧ 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: ∀ h > 0. h < d' ⟶ f x < f (x + h) by blast
  from real-lbound-gt-zero [OF d d']
  obtain e where 0 < e ∧ e < d ∧ 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:
    [| DERIV f x :> l; 0 < d; ∀ y. |x−y| < d −−> f(x) ≤ 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:
    [| DERIV f x :> l; 0 < d; ∀ 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 |]
     ==> ∃ d::real. 0 < d & (∀ y. |x−y| < d −−> a < y & y < b)
```

apply (*simp add*: *abs-interval-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*: [| $a < x$; $x < b$ |] ==>
$\exists d::real.\ 0 < d\ \&\ (\forall y.\ |x-y| < d \longrightarrow a \leq y\ \&\ y \leq b)$
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*: $\forall x.\ a \leq x\ \&\ x \leq b \longrightarrow isCont\ f\ x$
and *dif* [*rule-format*]: $\forall x.\ a < x\ \&\ x < b \longrightarrow f\ differentiable\ x$
shows $\exists z.\ a < z\ \&\ z < b\ \&\ DERIV\ f\ z :> 0$
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*: ~ $(a < x\ \&\ x < b)$
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*: $\sim\ (a < x' \ \&\ x' < b)$
— f is constant througout the interval
hence *x'eqab*: $x'=a \mid x'=b$ **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*: $\forall\, y.\ |r-y| < d \longrightarrow a \leq y \wedge y \leq b$
by *blast*
have *eq-fb*: $\forall\, z.\ a \leq z \dashrightarrow z \leq b \dashrightarrow f\ z = f\ b$
proof (*clarify*)
fix *z*::*real*
assume *az*: $a \leq z$ **and** *zb*: $z \leq b$
show $f\ z = f\ b$
proof (*rule order-antisym*)
show $f\ z \leq f\ b$ **by** (*simp add*: *fb-eq-fx x-max az zb*)
show $f\ b \leq f\ z$ **by** (*simp add*: *fb-eq-fx' x'-min az zb*)
qed
qed
have *bound'*: $\forall\, y.\ |r-y| < d \longrightarrow f\ r = f\ y$
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

20.7 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 $a\neq b$
hence *ba*: $b-a \neq 0$ **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*: $\forall x.\ a \leq x \ \&\ x \leq b \longrightarrow isCont\ f\ x$
and *dif* [*rule-format*]: $\forall x.\ a < x \ \&\ x < b \longrightarrow f\ differentiable\ x$
shows $\exists l\ z.\ a < z \ \&\ z < b \ \&\ DERIV\ f\ z :> l \ \&$
$(f(b) - f(a) = (b-a) * l)$
proof −
let *?F* = $\%x.\ f\,x - ((f\,b - f\,a)\ /\ (b-a)) * x$
have *contF*: $\forall x.\ a \leq x \wedge x \leq b \longrightarrow isCont\ ?F\ x$ **using** *con*
by (*fast intro*: *isCont-diff isCont-const isCont-mult isCont-Id*)
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 $a < z$ **.**

```
    show z < b .
    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: [| a < b;
        ∀x. a ≤ x & x ≤ b −−> isCont f x;
        ∀x. a < x & x < b −−> 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: [| a < b;
        ∀x. a ≤ x & x ≤ b −−> isCont f x;
        ∀x. a < x & x < b −−> DERIV f x :> 0 |]
       ==> ∀x. a ≤ x & x ≤ 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: [| a < b;
        ∀x. a ≤ x & x ≤ b −−> isCont f x;
        ∀x. a < x & x < b −−> DERIV f x :> 0;
        a ≤ x; x ≤ b |]
       ==> f x = f a
apply (blast dest: DERIV-isconst1)
done

lemma DERIV-isconst-all: ∀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:
    [|a ≠ b; ∀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: left-distrib diff-minus)
done

lemma DERIV-const-ratio-const2:
    [|a ≠ b; ∀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*:
  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*:
  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 ≤ x
    and cont': ∀ z. x − d ≤ z ∧ z ≤ x ⟶ 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-interval-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-interval-iff)
 qed
qed
```

Similar version for lower bound.

```
lemma lemma-isCont-inj2:
    [|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:
  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-interval-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 .
    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-interval-iff)
    qed
  qed
qed
```

Continuity of inverse function

```
lemma isCont-inverse-function:
  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*: $\forall z.\ |z-x| \leq e \dashrightarrow g\ (f\ z) = z$
$\forall z.\ |z-x| \leq e \dashrightarrow isCont\ f\ z$ **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 $0{<}e'$ **.**
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

ML
⟪
val LIM-def = thmLIM-def;
val NSLIM-def = thmNSLIM-def;
val isCont-def = thmisCont-def;
val isNSCont-def = thmisNSCont-def;
val deriv-def = thmderiv-def;
val nsderiv-def = thmnsderiv-def;
val differentiable-def = thmdifferentiable-def;
val NSdifferentiable-def = thmNSdifferentiable-def;
val increment-def = thmincrement-def;
val isUCont-def = thmisUCont-def;
val isNSUCont-def = thmisNSUCont-def;

val half-gt-zero-iff = thm half-gt-zero-iff;
val half-gt-zero = thms half-gt-zero;
val abs-diff-triangle-ineq = thm abs-diff-triangle-ineq;
val LIM-eq = thm LIM-eq;
val LIM-D = thm LIM-D;
val LIM-const = thm LIM-const;
val LIM-add = thm LIM-add;
val LIM-minus = thm LIM-minus;
val LIM-add-minus = thm LIM-add-minus;
val LIM-diff = thm LIM-diff;
val LIM-const-not-eq = thm LIM-const-not-eq;
val LIM-const-eq = thm LIM-const-eq;
val LIM-unique = thm LIM-unique;
val LIM-mult-zero = thm LIM-mult-zero;
val LIM-self = thm LIM-self;

val LIM-equal = thm LIM-equal;
val LIM-trans = thm LIM-trans;
val LIM-NSLIM = thm LIM-NSLIM;
val NSLIM-LIM = thm NSLIM-LIM;
val LIM-NSLIM-iff = thm LIM-NSLIM-iff;
val NSLIM-mult = thm NSLIM-mult;
val LIM-mult2 = thm LIM-mult2;
val NSLIM-add = thm NSLIM-add;
val LIM-add2 = thm LIM-add2;
val NSLIM-const = thm NSLIM-const;
val LIM-const2 = thm LIM-const2;
val NSLIM-minus = thm NSLIM-minus;
val LIM-minus2 = thm LIM-minus2;
val NSLIM-add-minus = thm NSLIM-add-minus;
val LIM-add-minus2 = thm LIM-add-minus2;
val NSLIM-inverse = thm NSLIM-inverse;
val LIM-inverse = thm LIM-inverse;
val NSLIM-zero = thm NSLIM-zero;
val LIM-zero2 = thm LIM-zero2;
val NSLIM-zero-cancel = thm NSLIM-zero-cancel;
val LIM-zero-cancel = thm LIM-zero-cancel;
val NSLIM-not-zero = thm NSLIM-not-zero;
val NSLIM-const-not-eq = thm NSLIM-const-not-eq;
val NSLIM-const-eq = thm NSLIM-const-eq;
val NSLIM-unique = thm NSLIM-unique;
val LIM-unique2 = thm LIM-unique2;
val NSLIM-mult-zero = thm NSLIM-mult-zero;
val LIM-mult-zero2 = thm LIM-mult-zero2;
val NSLIM-self = thm NSLIM-self;
val isNSContD = thm isNSContD;
val isNSCont-NSLIM = thm isNSCont-NSLIM;
val NSLIM-isNSCont = thm NSLIM-isNSCont;
val isNSCont-NSLIM-iff = thm isNSCont-NSLIM-iff;
val isNSCont-LIM-iff = thm isNSCont-LIM-iff;
val isNSCont-isCont-iff = thm isNSCont-isCont-iff;
val isCont-isNSCont = thm isCont-isNSCont;
val isNSCont-isCont = thm isNSCont-isCont;
val NSLIM-h-iff = thm NSLIM-h-iff;
val NSLIM-isCont-iff = thm NSLIM-isCont-iff;
val LIM-isCont-iff = thm LIM-isCont-iff;
val isCont-iff = thm isCont-iff;
val isCont-add = thm isCont-add;
val isCont-mult = thm isCont-mult;
val isCont-o = thm isCont-o;
val isCont-o2 = thm isCont-o2;
val isNSCont-minus = thm isNSCont-minus;
val isCont-minus = thm isCont-minus;
val isCont-inverse = thm isCont-inverse;
val isNSCont-inverse = thm isNSCont-inverse;

val isCont-diff = thm isCont-diff;
val isCont-const = thm isCont-const;
val isNSCont-const = thm isNSCont-const;
val isNSUContD = thm isNSUContD;
val isUCont-isCont = thm isUCont-isCont;
val isUCont-isNSUCont = thm isUCont-isNSUCont;
val isNSUCont-isUCont = thm isNSUCont-isUCont;
val DERIV-iff = thm DERIV-iff;
val DERIV-NS-iff = thm DERIV-NS-iff;
val DERIV-D = thm DERIV-D;
val NS-DERIV-D = thm NS-DERIV-D;
val DERIV-unique = thm DERIV-unique;
val NSDeriv-unique = thm NSDeriv-unique;
val differentiableD = thm differentiableD;
val differentiableI = thm differentiableI;
val NSdifferentiableD = thm NSdifferentiableD;
val NSdifferentiableI = thm NSdifferentiableI;
val LIM-I = thm LIM-I;
val DERIV-LIM-iff = thm DERIV-LIM-iff;
val DERIV-iff2 = thm DERIV-iff2;
val NSDERIV-NSLIM-iff = thm NSDERIV-NSLIM-iff;
val NSDERIV-NSLIM-iff2 = thm NSDERIV-NSLIM-iff2;
val NSDERIV-iff2 = thm NSDERIV-iff2;
val hypreal-not-eq-minus-iff = thm hypreal-not-eq-minus-iff;
val NSDERIVD5 = thm NSDERIVD5;
val NSDERIVD4 = thm NSDERIVD4;
val NSDERIVD3 = thm NSDERIVD3;
val NSDERIV-DERIV-iff = thm NSDERIV-DERIV-iff;
val NSDERIV-isNSCont = thm NSDERIV-isNSCont;
val DERIV-isCont = thm DERIV-isCont;
val NSDERIV-const = thm NSDERIV-const;
val DERIV-const = thm DERIV-const;
val NSDERIV-add = thm NSDERIV-add;
val DERIV-add = thm DERIV-add;
val NSDERIV-mult = thm NSDERIV-mult;
val DERIV-mult = thm DERIV-mult;
val NSDERIV-cmult = thm NSDERIV-cmult;
val DERIV-cmult = thm DERIV-cmult;
val NSDERIV-minus = thm NSDERIV-minus;
val DERIV-minus = thm DERIV-minus;
val NSDERIV-add-minus = thm NSDERIV-add-minus;
val DERIV-add-minus = thm DERIV-add-minus;
val NSDERIV-diff = thm NSDERIV-diff;
val DERIV-diff = thm DERIV-diff;
val incrementI = thm incrementI;
val incrementI2 = thm incrementI2;
val increment-thm = thm increment-thm;
val increment-thm2 = thm increment-thm2;
val increment-approx-zero = thm increment-approx-zero;

val NSDERIV-zero = thm NSDERIV-zero;
val NSDERIV-approx = thm NSDERIV-approx;
val NSDERIVD1 = thm NSDERIVD1;
val NSDERIVD2 = thm NSDERIVD2;
val NSDERIV-chain = thm NSDERIV-chain;
val DERIV-chain = thm DERIV-chain;
val DERIV-chain2 = thm DERIV-chain2;
val NSDERIV-Id = thm NSDERIV-Id;
val DERIV-Id = thm DERIV-Id;
val isCont-Id = thms isCont-Id;
val DERIV-cmult-Id = thm DERIV-cmult-Id;
val NSDERIV-cmult-Id = thm NSDERIV-cmult-Id;
val DERIV-pow = thm DERIV-pow;
val NSDERIV-pow = thm NSDERIV-pow;
val NSDERIV-inverse = thm NSDERIV-inverse;
val DERIV-inverse = thm DERIV-inverse;
val DERIV-inverse-fun = thm DERIV-inverse-fun;
val NSDERIV-inverse-fun = thm NSDERIV-inverse-fun;
val DERIV-quotient = thm DERIV-quotient;
val NSDERIV-quotient = thm NSDERIV-quotient;
val CARAT-DERIV = thm CARAT-DERIV;
val CARAT-NSDERIV = thm CARAT-NSDERIV;
val hypreal-eq-minus-iff3 = thm hypreal-eq-minus-iff3;
val starfun-if-eq = thm starfun-if-eq;
val CARAT-DERIVD = thm CARAT-DERIVD;
val f-inc-g-dec-Beq-f = thm f-inc-g-dec-Beq-f;
val f-inc-g-dec-Beq-g = thm f-inc-g-dec-Beq-g;
val f-inc-imp-le-lim = thm f-inc-imp-le-lim;
val lim-uminus = thm lim-uminus;
val g-dec-imp-lim-le = thm g-dec-imp-lim-le;
val Bolzano-bisect-le = thm Bolzano-bisect-le;
val Bolzano-bisect-fst-le-Suc = thm Bolzano-bisect-fst-le-Suc;
val Bolzano-bisect-Suc-le-snd = thm Bolzano-bisect-Suc-le-snd;
val eq-divide-2-times-iff = thm eq-divide-2-times-iff;
val Bolzano-bisect-diff = thm Bolzano-bisect-diff;
val Bolzano-nest-unique = thms Bolzano-nest-unique;
val not-P-Bolzano-bisect = thm not-P-Bolzano-bisect;
val not-P-Bolzano-bisect = thm not-P-Bolzano-bisect;
val lemma-BOLZANO2 = thm lemma-BOLZANO2;
val IVT = thm IVT;
val IVT2 = thm IVT2;
val IVT-objl = thm IVT-objl;
val IVT2-objl = thm IVT2-objl;
val isCont-bounded = thm isCont-bounded;
val isCont-has-Ub = thm isCont-has-Ub;
val isCont-eq-Ub = thm isCont-eq-Ub;
val isCont-eq-Lb = thm isCont-eq-Lb;
val isCont-Lb-Ub = thm isCont-Lb-Ub;
val DERIV-left-inc = thm DERIV-left-inc;

val DERIV-left-dec = thm DERIV-left-dec;
val DERIV-local-max = thm DERIV-local-max;
val DERIV-local-min = thm DERIV-local-min;
val DERIV-local-const = thm DERIV-local-const;
val Rolle = thm Rolle;
val MVT = thm MVT;
val DERIV-isconst-end = thm DERIV-isconst-end;
val DERIV-isconst1 = thm DERIV-isconst1;
val DERIV-isconst2 = thm DERIV-isconst2;
val DERIV-isconst-all = thm DERIV-isconst-all;
val DERIV-const-ratio-const = thm DERIV-const-ratio-const;
val DERIV-const-ratio-const2 = thm DERIV-const-ratio-const2;
val real-average-minus-first = thm real-average-minus-first;
val real-average-minus-second = thm real-average-minus-second;
val DERIV-const-average = thm DERIV-const-average;
val isCont-inj-range = thm isCont-inj-range;
val isCont-inverse-function = thm isCont-inverse-function;
⟩⟩

end

21 Series: Finite Summation and Infinite Series

theory *Series*
imports *SEQ Lim*
begin

declare *atLeastLessThan-iff*[*iff*]
declare *setsum-op-ivl-Suc*[*simp*]

constdefs
  sums :: (*nat* => *real*) => *real* => *bool* (**infixr** *sums* 80)
  f sums s == (%*n*. *setsum f* {0..<*n*}) ----> *s*

  summable :: (*nat*=>*real*) => *bool*
  summable f == (∃ *s*. *f sums s*)

  suminf :: (*nat*=>*real*) => *real*
  suminf f == *SOME s*. *f sums s*

syntax
  -suminf :: *idt* => *real* => *real* (∑ -. - [0, 10] 10)
translations
  ∑ *i*. *b* == *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* ⟹ *f*(*p*) ≤ *K*)
⟹ *setsum f {0..<n::nat}* ≤ *real n* ∗ *K*
using *setsum-bounded*[**where** *A* = {*0..<n*}]
by (*auto simp*:*real-of-nat-def*)

lemma *sumr-minus-one-realpow-zero* [*simp*]:
(∑ *i*=*0..<2*∗*n*. (−*1*) ^ *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*
apply (*induct n*)
apply (*case-tac* [*2*] *n*, *auto*)
done

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-offset*:
(∑ *m*=*0..<n*::*nat*. *f*(*m*+*k*)::*real*) = *setsum f* {*0..<n*+*k*} − *setsum f* {*0..<k*}
by (*induct n*, *auto*)

lemma *sumr-offset2*:
∀ *f*. (∑ *m*=*0..<n*::*nat*. *f*(*m*+*k*)::*real*) = *setsum f* {*0..<n*+*k*} − *setsum f* {*0..<k*}
by (*induct n*, *auto*)

lemma *sumr-offset3*:
setsum f {*0*::*nat..<n*+*k*} = (∑ *m*=*0..<n*. *f* (*m*+*k*)::*real*) + *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 (*simp add*: *sumr-offset*)

21.1 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*)
apply (*blast intro*: *someI2*)
done

lemma *summable-sumr-LIMSEQ-suminf*:
summable f ==> (%n. setsum f {0..<n}) ----> (suminf f)
apply (*simp add*: *summable-def suminf-def sums-def*)
apply (*blast intro*: *someI2*)
done

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*: (%*n*. *0*) *sums 0*
apply (*unfold sums-def*)
apply *simp*
apply (*rule LIMSEQ-const*)
done

lemma *summable-zero*: *summable* (%*n*. *0*)
apply (*rule sums-summable*)
apply (*rule sums-zero*)
done

lemma *suminf-zero*: *suminf* (%*n*. *0*) = *0*
apply (*rule sym*)
apply (*rule sums-unique*)
apply (*rule sums-zero*)
done

lemma *sums-mult*: *f sums a* ==> (%*n*. *c* * *f n*) *sums* (*c* * *a*)
by (*auto simp add*: *sums-def setsum-mult* [*symmetric*]
intro!: *LIMSEQ-mult intro*: *LIMSEQ-const*)

lemma *summable-mult*: *summable f* ==> *summable* (%*n*. *c* * *f n*)
apply (*unfold summable-def*)
apply (*auto intro*: *sums-mult*)
done

lemma *suminf-mult*: *summable f* ==> *suminf* (%*n*. *c* * *f n*) = *c* * *suminf f*
apply (*rule sym*)
apply (*rule sums-unique*)
apply (*rule sums-mult*)
apply (*erule summable-sums*)
done

lemma *sums-mult2*: *f sums a* ==> (%*n*. *f n* * *c*) *sums* (*a* * *c*)
apply (*subst mult-commute*)
apply (*subst mult-commute*)**back**
apply (*erule sums-mult*)
done

lemma *summable-mult2*: *summable f* ==> *summable* (%*n*. *f n* * *c*)
apply (*unfold summable-def*)
apply (*auto intro*: *sums-mult2*)
done

lemma *suminf-mult2*: *summable f ==> suminf f * c = (∑ n. f n * c)*
by (*auto intro*!: *sums-unique sums-mult summable-sums simp add*: *mult-commute*)

lemma *sums-divide*: *f sums a ==> (%n. (f n)/c) sums (a/c)*
by (*simp add*: *real-divide-def sums-mult mult-commute* [*of - inverse c*])

lemma *summable-divide*: *summable f ==> summable (%n. (f n) / c)*
  apply (*unfold summable-def*)
  apply (*auto intro*: *sums-divide*)
done

lemma *suminf-divide*: *summable f ==> suminf (%n. (f n) / c) = (suminf f) / c*
  apply (*rule sym*)
  apply (*rule sums-unique*)
  apply (*rule sums-divide*)
  apply (*erule summable-sums*)
done

lemma *sums-add*: [| *x sums x0*; *y sums y0* |] *==> (%n. x n + y n) sums (x0+y0)*
by (*auto simp add*: *sums-def setsum-addf intro*: *LIMSEQ-add*)

lemma *summable-add*: *summable f ==> summable g ==> summable (%x. f x + g x)*
  apply (*unfold summable-def*)
  apply *clarify*
  apply (*rule exI*)
  apply (*erule sums-add*)
  apply *assumption*
done

lemma *suminf-add*:
    [| *summable f*; *summable g* |]
     ==> suminf f + suminf g = (∑ n. f n + g n)
by (*auto intro*!: *sums-add sums-unique summable-sums*)

lemma *sums-diff*: [| *x sums x0*; *y sums y0* |] *==> (%n. x n − y n) sums (x0−y0)*
by (*auto simp add*: *sums-def setsum-subtractf intro*: *LIMSEQ-diff*)

lemma *summable-diff*: *summable f ==> summable g ==> summable (%x. f x − g x)*
  apply (*unfold summable-def*)
  apply *clarify*
  apply (*rule exI*)
  apply (*erule sums-diff*)
  apply *assumption*
done

lemma *suminf-diff*:
    [| *summable f*; *summable g* |]

```
    ==> suminf f - suminf g  = (∑ n. f n - g n)
by (auto intro!: sums-diff sums-unique summable-sums)

lemma sums-minus: f sums s ==> (%x. - f x) sums (- s)
  by (simp add: sums-def setsum-negf LIMSEQ-minus)

lemma summable-minus: summable f ==> summable (%x. - f x)
  by (auto simp add: summable-def intro: sums-minus)

lemma suminf-minus: summable f ==> suminf (%x. - f x) = - (suminf f)
  apply (rule sym)
  apply (rule sums-unique)
  apply (rule sums-minus)
  apply (erule summable-sums)
done

lemma sums-group:
    [|summable f; 0 < k |] ==> (%n. setsum f {n*k..<n*k+k}) sums (suminf f)
apply (drule summable-sums)
apply (auto simp add: sums-def LIMSEQ-def sumr-group)
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 (auto dest!: not-leE)
apply (drule-tac j = no in less-le-trans, auto)
done

lemma sumr-pos-lt-pair-lemma:
  [|∀ d. - f (n + (d + d)) < (f (Suc (n + (d + d))) :: real) |]
   ==> setsum f {0..<n+Suc(Suc 0)} ≤ setsum f {0..<Suc(Suc 0) * Suc no +
n}
apply (induct no, auto)
apply (drule-tac x = Suc no in spec)
apply (simp add: add-ac)

apply simp
done


lemma sumr-pos-lt-pair:
    [|summable f;
      ∀ d. 0 < (f(n + (Suc(Suc 0) * d))) + f(n + ((Suc(Suc 0) * d) + 1))|]
     ==> setsum f {0..<n} < suminf f
apply (drule summable-sums)
apply (auto simp add: sums-def LIMSEQ-def)
apply (drule-tac x = f (n) + f (n + 1) in spec)
apply (auto iff: real-0-less-add-iff)
  — legacy proof: not necessarily better!
apply (rule-tac [2] ccontr, drule-tac [2] linorder-not-less [THEN iffD1])
```

apply (*frule-tac* [*2*] *no=no* **in** *sumr-pos-lt-pair-lemma*)
apply (*drule-tac x = 0* **in** *spec*, *simp*)
apply (*rotate-tac 1*, *drule-tac x = Suc (Suc 0) * (Suc no) + n* **in** *spec*)
apply (*safe*, *simp*)
apply (*subgoal-tac suminf f + (f (n) + f (n + 1)) ≤*
*setsum f {0 ..< Suc (Suc 0) * (Suc no) + n}*)
apply (*rule-tac* [*2*] *y = setsum f {0..<n+ Suc (Suc 0)}* **in** *order-trans*)
prefer *3* **apply** *assumption*
apply (*rule-tac* [*2*] *y = setsum f {0..<n} + (f (n) + f (n + 1))* **in** *order-trans*)
apply *simp-all*
apply (*subgoal-tac suminf f ≤ setsum f {0..< Suc (Suc 0) * (Suc no) + n}*)
apply (*rule-tac* [*2*] *y = suminf f + (f (n) + f (n + 1))* **in** *order-trans*)
prefer *3* **apply** *simp*
apply (*drule-tac* [*2*] *x = 0* **in** *spec*)
prefer *2* **apply** *simp*
apply (*subgoal-tac 0 ≤ setsum f {0 ..< Suc (Suc 0) * Suc no + n} + − suminf f*)
apply (*simp add*: *abs-if*)
apply (*auto simp add*: *linorder-not-less* [*symmetric*])
done

A summable series of positive terms has limit that is at least as great as any partial sum.

lemma *series-pos-le*:
[| *summable f*; ∀ *m ≥ n. 0 ≤ f(m)* |] ==> *setsum f {0..<n} ≤ 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*:
[| *summable f*; ∀ *m ≥ n. 0 < f(m)* |] ==> *setsum f {0..<n} < suminf f*
apply (*rule-tac y = setsum f {0..<Suc n}* **in** *order-less-le-trans*)
apply (*rule-tac* [*2*] *series-pos-le*, *auto*)
apply (*drule-tac x = m* **in** *spec*, *auto*)
done

Sum of a geometric progression.

lemmas *sumr-geometric = geometric-sum* [**where** *'a = real*]

lemma *geometric-sums*: *abs(x) < 1* ==> *(%n. x ^ n) sums (1/(1 − x))*
apply (*case-tac x = 1*)
apply (*auto dest*!: *LIMSEQ-rabs-realpow-zero2*
simp add: *sumr-geometric sums-def diff-minus add-divide-distrib*)
apply (*subgoal-tac 1 / (1 + −x) = 0/ (x − 1) + − 1/ (x − 1)*)

apply (*erule ssubst*)
apply (*rule LIMSEQ-add*, *rule LIMSEQ-divide*)
apply (*auto intro*: *LIMSEQ-const simp add*: *diff-minus minus-divide-right LIMSEQ-rabs-realpow-zero2*)
done

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-Cauchy*:
summable f =
(∀ *e* > *0*. ∃ *N*. ∀ *m* ≥ *N*. ∀ *n*. *abs*(*setsum f* {*m*..<*n*}) < *e*)
apply (*auto simp add*: *summable-convergent-sumr-iff Cauchy-convergent-iff* [*symmetric*] *Cauchy-def diff-minus*[*symmetric*])
apply (*drule-tac* [!] *spec*, *auto*)
apply (*rule-tac x = M* **in** *exI*)
apply (*rule-tac* [*2*] *x = N* **in** *exI*, *auto*)
apply (*cut-tac* [!] *m = m* **and** *n = n* **in** *less-linear*, *auto*)
apply (*frule le-less-trans* [*THEN less-imp-le*], *assumption*)
apply (*drule-tac x = n* **in** *spec*, *simp*)
apply (*drule-tac x = m* **in** *spec*)
apply(*simp add*: *setsum-diff*[*symmetric*])
apply(*subst abs-minus-commute*)
apply(*simp add*: *setsum-diff*[*symmetric*])
apply(*simp add*: *setsum-diff*[*symmetric*])
done

Comparison test

lemma *summable-comparison-test*:
[| ∃ *N*. ∀ *n* ≥ *N*. *abs*(*f n*) ≤ *g n*; *summable g* |] ==> *summable f*
apply (*auto simp add*: *summable-Cauchy*)
apply (*drule spec*, *auto*)
apply (*rule-tac x = N + Na* **in** *exI*, *auto*)
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* = ∑*k*=*m*..<*n*. *abs*(*f k*) **in** *order-le-less-trans*)
apply (*rule setsum-abs*)
apply (*rule-tac y = setsum g* {*m*..<*n*} **in** *order-le-less-trans*)
apply (*auto intro*: *setsum-mono simp add*: *abs-interval-iff*)
done

lemma *summable-rabs-comparison-test*:
[| ∃ *N*. ∀ *n* ≥ *N*. *abs*(*f n*) ≤ *g n*; *summable g* |]
==> *summable* (%*k*. *abs* (*f k*))
apply (*rule summable-comparison-test*)
apply (*auto*)
done

Limit comparison property for series (c.f. jrh)

lemma *summable-le*:
[|∀ *n*. *f n* ≤ *g n*; *summable f*; *summable g* |] ==> *suminf f* ≤ *suminf g*
apply (*drule summable-sums*)+
apply (*auto intro*!: *LIMSEQ-le simp add*: *sums-def*)
apply (*rule exI*)
apply (*auto intro*!: *setsum-mono*)
done

lemma *summable-le2*:
[|∀ *n*. *abs*(*f n*) ≤ *g n*; *summable g* |]
==> *summable f* & *suminf f* ≤ *suminf g*
apply (*auto intro*: *summable-comparison-test intro*!: *summable-le*)
apply (*simp add*: *abs-le-interval-iff*)
done

Absolute convergence imples normal convergence

lemma *summable-rabs-cancel*: *summable* (%*n*. *abs* (*f n*)) ==> *summable f*
apply (*auto simp add*: *summable-Cauchy*)
apply (*drule spec*, *auto*)
apply (*rule-tac x* = *N* **in** *exI*, *auto*)
apply (*drule spec*, *auto*)
apply (*rule-tac y* = ∑ *n*=*m*..<*n*. *abs*(*f n*) **in** *order-le-less-trans*)
apply (*auto*)
done

Absolute convergence of series

lemma *summable-rabs*:
summable (%*n*. *abs* (*f n*)) ==> *abs*(*suminf f*) ≤ (∑ *n*. *abs*(*f n*))
by (*auto intro*: *LIMSEQ-le LIMSEQ-imp-rabs summable-rabs-cancel summable-sumr-LIMSEQ-suminf*)

21.2 The Ratio Test

lemma *rabs-ratiotest-lemma*: [| *c* ≤ *0*; *abs x* ≤ *c* * *abs y* |] ==> *x* = (*0*::*real*)
apply (*drule order-le-imp-less-or-eq*, *auto*)
apply (*subgoal-tac 0* ≤ *c* * *abs y*)
apply (*simp add*: *zero-le-mult-iff*, *arith*)
done

lemma *le-Suc-ex*: (*k*::*nat*) ≤ *l* ==> (∃ *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*) ≤ *l*) = (∃ *n*. *l* = *k* + *n*)
by (*auto simp add*: *le-Suc-ex*)

lemma *ratio-test-lemma2*:

```
    [| ∀ n ≥ N. abs(f(Suc n)) ≤ c*abs(f n) |]
     ==> 0 < c | summable f
apply (simp (no-asm) add: linorder-not-le [symmetric])
apply (simp add: summable-Cauchy)
apply (safe, subgoal-tac ∀ n. N < n --> 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: rabs-ratiotest-lemma)
apply (rule-tac x = Suc N in exI, clarify)
apply(simp cong:setsum-ivl-cong)
done

lemma ratio-test:
    [| c < 1; ∀ n ≥ N. abs(f(Suc n)) ≤ c*abs(f n) |]
     ==> summable f
apply (frule ratio-test-lemma2, auto)
apply (rule-tac g = %n. (abs (f N) / (c ^ N))*c ^ 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 realpow-not-zero)
apply (induct-tac na, auto)
apply (rule-tac y = c*abs (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 = abs (f N) * (1/ (1 - c)) / (c ^ N) in exI)
apply (rule sums-divide)
apply (rule sums-mult)
apply (auto intro!: geometric-sums)
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) x :> (∑ r=m..<n. f' r x)
apply (induct n)
apply (auto intro: DERIV-add)
done
```

ML

```
⟪
val sums-def = thmsums-def;
val summable-def = thmsummable-def;
val suminf-def = thmsuminf-def;

val sumr-minus-one-realpow-zero = thm sumr-minus-one-realpow-zero;
val sumr-one-lb-realpow-zero = thm sumr-one-lb-realpow-zero;
```

val sumr-group = thm sumr-group;
val sums-summable = thm sums-summable;
val summable-sums = thm summable-sums;
val summable-sumr-LIMSEQ-suminf = thm summable-sumr-LIMSEQ-suminf;
val sums-unique = thm sums-unique;
val series-zero = thm series-zero;
val sums-mult = thm sums-mult;
val sums-divide = thm sums-divide;
val sums-diff = thm sums-diff;
val suminf-mult = thm suminf-mult;
val suminf-mult2 = thm suminf-mult2;
val suminf-diff = thm suminf-diff;
val sums-minus = thm sums-minus;
val sums-group = thm sums-group;
val sumr-pos-lt-pair-lemma = thm sumr-pos-lt-pair-lemma;
val sumr-pos-lt-pair = thm sumr-pos-lt-pair;
val series-pos-le = thm series-pos-le;
val series-pos-less = thm series-pos-less;
val sumr-geometric = thm sumr-geometric;
val geometric-sums = thm geometric-sums;
val summable-convergent-sumr-iff = thm summable-convergent-sumr-iff;
val summable-Cauchy = thm summable-Cauchy;
val summable-comparison-test = thm summable-comparison-test;
val summable-rabs-comparison-test = thm summable-rabs-comparison-test;
val summable-le = thm summable-le;
val summable-le2 = thm summable-le2;
val summable-rabs-cancel = thm summable-rabs-cancel;
val summable-rabs = thm summable-rabs;
val rabs-ratiotest-lemma = thm rabs-ratiotest-lemma;
val le-Suc-ex = thm le-Suc-ex;
val le-Suc-ex-iff = thm le-Suc-ex-iff;
val ratio-test-lemma2 = thm ratio-test-lemma2;
val ratio-test = thm ratio-test;
val DERIV-sumr = thm DERIV-sumr;
⟩⟩

end

22 HSeries: Finite Summation and Infinite Series for Hyperreals

theory *HSeries*
imports *Series*
begin

constdefs
sumhr :: (*hypnat* ∗ *hypnat* ∗ (*nat*=>*real*)) => *hypreal*

sumhr ==
  %(*M*,*N*,*f*). *starfun2* (%*m n*. *setsum f* {*m*..<*n*}) *M N*

NSsums :: [*nat*=>*real*,*real*] => *bool* (**infixr** *NSsums 80*)
f NSsums s == (%*n*. *setsum f* {*0*..<*n*}) −−−−*NS*> *s*

NSsummable :: (*nat*=>*real*) => *bool*
NSsummable f == (∃ *s*. *f NSsums s*)

NSsuminf :: (*nat*=>*real*) => *real*
NSsuminf f == (@*s*. *f NSsums s*)

lemma *sumhr*:
  sumhr(*star-n M*, *star-n N*, *f*) =
   star-n (%*n*. *setsum f* {*M n*..<*N n*})
by (*simp add*: *sumhr-def starfun2-star-n*)

Base case in definition of *sumr*

lemma *sumhr-zero* [*simp*]: *sumhr* (*m*,*0*,*f*) = *0*
apply (*cases m*)
apply (*simp add*: *star-n-zero-num sumhr symmetric*)
done

Recursive case in definition of *sumr*

lemma *sumhr-if*:
  sumhr(*m*,*n*+*1*,*f*) =
   (*if n* + *1* ≤ *m then 0 else sumhr*(*m*,*n*,*f*) + (∗*f*∗ *f*) *n*)
apply (*cases m*, *cases n*)
apply (*auto simp add*: *star-n-one-num sumhr star-n-add star-n-le starfun*
      star-n-zero-num star-n-eq-iff, *ultra*+)
done

lemma *sumhr-Suc-zero* [*simp*]: *sumhr* (*n* + *1*, *n*, *f*) = *0*
apply (*cases n*)
apply (*simp add*: *star-n-one-num sumhr star-n-add star-n-zero-num*)
done

lemma *sumhr-eq-bounds* [*simp*]: *sumhr* (*n*,*n*,*f*) = *0*
apply (*cases n*)
apply (*simp add*: *sumhr star-n-zero-num*)
done

lemma *sumhr-Suc* [*simp*]: *sumhr* (*m*,*m* + *1*,*f*) = (∗*f*∗ *f*) *m*
apply (*cases m*)
apply (*simp add*: *sumhr star-n-one-num star-n-add starfun*)
done

lemma *sumhr-add-lbound-zero* [*simp*]: *sumhr*(*m*+*k*,*k*,*f*) = *0*

apply (*cases m*, *cases k*)
apply (*simp add*: *sumhr star-n-add star-n-zero-num*)
done

lemma *sumhr-add*: *sumhr* (*m*,*n*,*f*) + *sumhr*(*m*,*n*,*g*) = *sumhr*(*m*,*n*,%*i*. *f i* + *g i*)
apply (*cases m*, *cases n*)
apply (*simp add*: *sumhr star-n-add setsum-addf*)
done

lemma *sumhr-mult*: *hypreal-of-real r* ∗ *sumhr*(*m*,*n*,*f*) = *sumhr*(*m*,*n*,%*n*. *r* ∗ *f n*)
apply (*cases m*, *cases n*)
apply (*simp add*: *sumhr star-of-def star-n-mult setsum-mult*)
done

lemma *sumhr-split-add*: *n* < *p* ==> *sumhr*(*0*,*n*,*f*) + *sumhr*(*n*,*p*,*f*) = *sumhr*(*0*,*p*,*f*)
apply (*cases n*, *cases p*)
apply (*auto elim*!: *FreeUltrafilterNat-subset simp*
add: *star-n-zero-num sumhr star-n-add star-n-less setsum-add-nat-ivl star-n-eq-iff*)
done

lemma *sumhr-split-diff*: *n*<*p* ==> *sumhr*(*0*,*p*,*f*) − *sumhr*(*0*,*n*,*f*) = *sumhr*(*n*,*p*,*f*)
by (*drule-tac f1* = *f* **in** *sumhr-split-add* [*symmetric*], *simp*)

lemma *sumhr-hrabs*: *abs*(*sumhr*(*m*,*n*,*f*)) ≤ *sumhr*(*m*,*n*,%*i*. *abs*(*f i*))
apply (*cases n*, *cases m*)
apply (*simp add*: *sumhr star-n-le star-n-abs setsum-abs*)
done

other general version also needed

lemma *sumhr-fun-hypnat-eq*:
(∀ *r*. *m* ≤ *r* & *r* < *n* −−> *f r* = *g r*) −−>
sumhr(*hypnat-of-nat m*, *hypnat-of-nat n*, *f*) =
sumhr(*hypnat-of-nat m*, *hypnat-of-nat n*, *g*)
by (*fastsimp simp add*: *sumhr hypnat-of-nat-eq intro*:*setsum-cong*)

lemma *sumhr-const*:
sumhr(*0*, *n*, %*i*. *r*) = *hypreal-of-hypnat n* ∗ *hypreal-of-real r*
apply (*cases n*)
apply (*simp add*: *sumhr star-n-zero-num hypreal-of-hypnat*
star-of-def star-n-mult real-of-nat-def)
done

lemma *sumhr-less-bounds-zero* [*simp*]: *n* < *m* ==> *sumhr*(*m*,*n*,*f*) = *0*
apply (*cases m*, *cases n*)
apply (*auto elim*: *FreeUltrafilterNat-subset*
simp add: *sumhr star-n-less star-n-zero-num star-n-eq-iff*)
done

lemma *sumhr-minus*: *sumhr*(*m*, *n*, %*i*. − *f i*) = − *sumhr*(*m*, *n*, *f*)
apply (*cases m*, *cases n*)
apply (*simp add*: *sumhr star-n-minus setsum-negf*)
done

lemma *sumhr-shift-bounds*:
sumhr(*m*+*hypnat-of-nat k*,*n*+*hypnat-of-nat k*,*f*) = *sumhr*(*m*,*n*,%*i*. *f*(*i* + *k*))
apply (*cases m*, *cases n*)
apply (*simp add*: *sumhr star-n-add setsum-shift-bounds-nat-ivl hypnat-of-nat-eq*)
done

22.1 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-of-hypnat whn*
by (*simp add*: *hypnat-omega-def star-n-zero-num sumhr hypreal-of-hypnat real-of-nat-def*)

lemma *sumhr-hypreal-omega-minus-one*: *sumhr*(*0*, *whn*, %*i*. *1*) = *omega* − *1*
by (*simp add*: *hypnat-omega-def star-n-zero-num omega-def star-n-one-num sumhr star-n-diff real-of-nat-def*)

lemma *sumhr-minus-one-realpow-zero* [*simp*]:
sumhr(*0*, *whn* + *whn*, %*i*. (*−1*) ^ (*i*+*1*)) = *0*
by (*simp del*: *realpow-Suc*
add: *sumhr star-n-add nat-mult-2* [*symmetric*] *star-n-zero-num star-n-zero-num hypnat-omega-def*)

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*)
by(*simp add*: *sumhr hypreal-of-hypnat-eq hypnat-of-nat-eq real-of-nat-def star-of-def star-n-mult cong*: *setsum-ivl-cong*)

lemma *starfunNat-sumr*: (∗*f*∗ (%*n*. *setsum f* {*0*..<*n*})) *N* = *sumhr*(*0*,*N*,*f*)
apply (*cases N*)
apply (*simp add*: *star-n-zero-num starfun sumhr*)
done

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*)
apply (*blast intro*: *someI2*)
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

Easy to prove stsandard case now

lemma *summable-LIMSEQ-zero*: *summable f ==> f ----> 0*
by (*simp add*: *summable-NSsummable-iff LIMSEQ-NSLIMSEQ-iff NSsummable-NSLIMSEQ-zero*)

Nonstandard comparison test

lemma *NSsummable-comparison-test*:
[| ∃ *N*. ∀ *n*. *N* ≤ *n* --> *abs*(*f n*) ≤ *g n*; *NSsummable g* |] ==> *NSsummable f*
by (*auto intro*: *summable-comparison-test*
simp add: *summable-NSsummable-iff* [*symmetric*])

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

ML
⟪
val sumhr = thm sumhr;
val sumhr-zero = thm sumhr-zero;
val sumhr-if = thm sumhr-if;
val sumhr-Suc-zero = thm sumhr-Suc-zero;
val sumhr-eq-bounds = thm sumhr-eq-bounds;
val sumhr-Suc = thm sumhr-Suc;
val sumhr-add-lbound-zero = thm sumhr-add-lbound-zero;
val sumhr-add = thm sumhr-add;
val sumhr-mult = thm sumhr-mult;
val sumhr-split-add = thm sumhr-split-add;
val sumhr-split-diff = thm sumhr-split-diff;
val sumhr-hrabs = thm sumhr-hrabs;
val sumhr-fun-hypnat-eq = thm sumhr-fun-hypnat-eq;
val sumhr-less-bounds-zero = thm sumhr-less-bounds-zero;
val sumhr-minus = thm sumhr-minus;
val sumhr-shift-bounds = thm sumhr-shift-bounds;
val sumhr-hypreal-of-hypnat-omega = thm sumhr-hypreal-of-hypnat-omega;
val sumhr-hypreal-omega-minus-one = thm sumhr-hypreal-omega-minus-one;
val sumhr-minus-one-realpow-zero = thm sumhr-minus-one-realpow-zero;
val sumhr-interval-const = thm sumhr-interval-const;
val starfunNat-sumr = thm starfunNat-sumr;
val sumhr-hrabs-approx = thm sumhr-hrabs-approx;
val sums-NSsums-iff = thm sums-NSsums-iff;
val summable-NSsummable-iff = thm summable-NSsummable-iff;
val suminf-NSsuminf-iff = thm suminf-NSsuminf-iff;
val NSsums-NSsummable = thm NSsums-NSsummable;
val NSsummable-NSsums = thm NSsummable-NSsums;
val NSsums-unique = thm NSsums-unique;
val NSseries-zero = thm NSseries-zero;

val NSsummable-NSCauchy = thm NSsummable-NSCauchy;
val NSsummable-NSLIMSEQ-zero = thm NSsummable-NSLIMSEQ-zero;
val summable-LIMSEQ-zero = thm summable-LIMSEQ-zero;
val NSsummable-comparison-test = thm NSsummable-comparison-test;
val NSsummable-rabs-comparison-test = thm NSsummable-rabs-comparison-test;
⟩⟩

end

23 NthRoot: Existence of Nth Root

theory *NthRoot*
imports *SEQ HSeries*
begin

Various lemmas needed for this result. We follow the proof given by John Lindsay Orr (`jorr@math.unl.edu`) in his Analysis Webnotes available at http://www.math.unl.edu/~webnotes.

Lemmas about sequences of reals are used to reach the result.

lemma *lemma-nth-realpow-non-empty*:
[| (*0*::*real*) < *a*; *0* < *n* |] ==> ∃ *s*. *s* : {*x*. *x* ^ *n* <= *a* & *0* < *x*}
apply (*case-tac 1* <= *a*)
apply (*rule-tac x* = *1* **in** *exI*)
apply (*drule-tac* [*2*] *linorder-not-le* [*THEN iffD1*])
apply (*drule-tac* [*2*] *less-not-refl2* [*THEN not0-implies-Suc*], *simp*)
apply (*force intro*!: *realpow-Suc-le-self simp del*: *realpow-Suc*)
done

Used only just below

lemma *realpow-ge-self2*: [| (*1*::*real*) ≤ *r*; *0* < *n* |] ==> *r* ≤ *r* ^ *n*
by (*insert power-increasing* [*of 1 n r*], *simp*)

lemma *lemma-nth-realpow-isUb-ex*:
[| (*0*::*real*) < *a*; *0* < *n* |]
==> ∃ *u*. *isUb* (*UNIV*::*real set*) {*x*. *x* ^ *n* <= *a* & *0* < *x*} *u*
apply (*case-tac 1* <= *a*)
apply (*rule-tac x* = *a* **in** *exI*)
apply (*drule-tac* [*2*] *linorder-not-le* [*THEN iffD1*])
apply (*rule-tac* [*2*] *x* = *1* **in** *exI*)
apply (*rule-tac* [!] *setleI* [*THEN isUbI*], *safe*)
apply (*simp-all* (*no-asm*))
apply (*rule-tac* [!] *ccontr*)
apply (*drule-tac* [!] *linorder-not-le* [*THEN iffD1*])
apply (*drule realpow-ge-self2*, *assumption*)
apply (*drule-tac n* = *n* **in** *realpow-less*)
apply (*assumption*+)
apply (*drule real-le-trans*, *assumption*)

apply (*drule-tac y = y ^ n* **in** *order-less-le-trans*, *assumption*, *simp*)
apply (*drule-tac n = n* **in** *zero-less-one* [*THEN realpow-less*], *auto*)
done

lemma *nth-realpow-isLub-ex*:
[| (*0*::*real*) < *a*; *0* < *n* |]
==> ∃ *u*. *isLub* (*UNIV*::*real set*) {*x*. *x* ^ *n* <= *a* & *0* < *x*} *u*
by (*blast intro*: *lemma-nth-realpow-isUb-ex lemma-nth-realpow-non-empty reals-complete*)

23.1 First Half – Lemmas First

lemma *lemma-nth-realpow-seq*:
isLub (*UNIV*::*real set*) {*x*. *x* ^ *n* <= *a* & (*0*::*real*) < *x*} *u*
==> *u* + *inverse*(*real* (*Suc k*)) ~: {*x*. *x* ^ *n* <= *a* & *0* < *x*}
apply (*safe*, *drule isLubD2*, *blast*)
apply (*simp add*: *linorder-not-less* [*symmetric*])
done

lemma *lemma-nth-realpow-isLub-gt-zero*:
[| *isLub* (*UNIV*::*real set*) {*x*. *x* ^ *n* <= *a* & (*0*::*real*) < *x*} *u*;
0 < *a*; *0* < *n* |] ==> *0* < *u*
apply (*drule lemma-nth-realpow-non-empty*, *auto*)
apply (*drule-tac y = s* **in** *isLub-isUb* [*THEN isUbD*])
apply (*auto intro*: *order-less-le-trans*)
done

lemma *lemma-nth-realpow-isLub-ge*:
[| *isLub* (*UNIV*::*real set*) {*x*. *x* ^ *n* <= *a* & (*0*::*real*) < *x*} *u*;
0 < *a*; *0* < *n* |] ==> *ALL k*. *a* <= (*u* + *inverse*(*real* (*Suc k*))) ^ *n*
apply *safe*
apply (*frule lemma-nth-realpow-seq*, *safe*)
apply (*auto elim*: *order-less-asym simp add*: *linorder-not-less* [*symmetric*]
iff: *real-0-less-add-iff*) — legacy iff rule!
apply (*simp add*: *linorder-not-less*)
apply (*rule order-less-trans* [*of - 0*])
apply (*auto intro*: *lemma-nth-realpow-isLub-gt-zero*)
done

First result we want

lemma *realpow-nth-ge*:
[| (*0*::*real*) < *a*; *0* < *n*;
isLub (*UNIV*::*real set*)
{*x*. *x* ^ *n* <= *a* & *0* < *x*} *u* |] ==> *a* <= *u* ^ *n*
apply (*frule lemma-nth-realpow-isLub-ge*, *safe*)
apply (*rule LIMSEQ-inverse-real-of-nat-add* [*THEN LIMSEQ-pow*, *THEN LIMSEQ-le-const*])
apply (*auto simp add*: *real-of-nat-def*)
done

23.2 Second Half

lemma *less-isLub-not-isUb*:
[| *isLub* (*UNIV*::*real set*) *S u*; *x* < *u* |]
==> ~ *isUb* (*UNIV*::*real set*) *S x*
apply *safe*
apply (*drule isLub-le-isUb*, *assumption*)
apply (*drule order-less-le-trans*, *auto*)
done

lemma *not-isUb-less-ex*:
~ *isUb* (*UNIV*::*real set*) *S u* ==> ∃*x* ∈ *S*. *u* < *x*
apply (*rule ccontr*, *erule swap*)
apply (*rule setleI* [*THEN isUbI*])
apply (*auto simp add*: *linorder-not-less* [*symmetric*])
done

lemma *real-mult-less-self*: *0* < *r* ==> *r* * (*1* + −*inverse*(*real* (*Suc n*))) < *r*
apply (*simp* (*no-asm*) *add*: *right-distrib*)
apply (*rule add-less-cancel-left* [*of* −*r*, *THEN iffD1*])
apply (*auto intro*: *mult-pos-pos*
simp add: *add-assoc* [*symmetric*] *neg-less-0-iff-less*)
done

lemma *real-mult-add-one-minus-ge-zero*:
0 < *r* ==> *0* <= *r*(*1* + −*inverse*(*real* (*Suc n*)))
by (*simp add*: *zero-le-mult-iff real-of-nat-inverse-le-iff real-0-le-add-iff*)

lemma *lemma-nth-realpow-isLub-le*:
[| *isLub* (*UNIV*::*real set*) {*x*. *x* ^ *n* <= *a* & (*0*::*real*) < *x*} *u*;
0 < *a*; *0* < *n* |] ==> *ALL k*. (*u*(*1* + −*inverse*(*real* (*Suc k*)))) ^ *n* <= *a*
apply *safe*
apply (*frule less-isLub-not-isUb* [*THEN not-isUb-less-ex*])
apply (*rule-tac n* = *k* **in** *real-mult-less-self*)
apply (*blast intro*: *lemma-nth-realpow-isLub-gt-zero*, *safe*)
apply (*drule-tac n* = *k* **in**
lemma-nth-realpow-isLub-gt-zero [*THEN real-mult-add-one-minus-ge-zero*],
assumption+)
apply (*blast intro*: *order-trans order-less-imp-le power-mono*)
done

Second result we want

lemma *realpow-nth-le*:
[| (*0*::*real*) < *a*; *0* < *n*;
isLub (*UNIV*::*real set*)
{*x*. *x* ^ *n* <= *a* & *0* < *x*} *u* |] ==> *u* ^ *n* <= *a*
apply (*frule lemma-nth-realpow-isLub-le*, *safe*)
apply (*rule LIMSEQ-inverse-real-of-nat-add-minus-mult*
[*THEN LIMSEQ-pow*, *THEN LIMSEQ-le-const2*])
apply (*auto simp add*: *real-of-nat-def*)

done

The theorem at last!

lemma *realpow-nth*: [| (*0*::*real*) < *a*; *0* < *n* |] ==> ∃ *r*. *r* ^ *n* = *a*
apply (*frule nth-realpow-isLub-ex*, *auto*)
apply (*auto intro*: *realpow-nth-le realpow-nth-ge order-antisym*)
done

lemma *realpow-pos-nth*: [| (*0*::*real*) < *a*; *0* < *n* |] ==> ∃ *r*. *0* < *r* & *r* ^ *n* = *a*
apply (*frule nth-realpow-isLub-ex*, *auto*)
apply (*auto intro*: *realpow-nth-le realpow-nth-ge order-antisym lemma-nth-realpow-isLub-gt-zero*)
done

lemma *realpow-pos-nth2*: (*0*::*real*) < *a* ==> ∃ *r*. *0* < *r* & *r* ^ *Suc n* = *a*
by (*blast intro*: *realpow-pos-nth*)

lemma *realpow-pos-nth-unique*:
[| (*0*::*real*) < *a*; *0* < *n* |] ==> *EX*! *r*. *0* < *r* & *r* ^ *n* = *a*
apply (*auto intro*!: *realpow-pos-nth*)
apply (*cut-tac x* = *r* **and** *y* = *y* **in** *linorder-less-linear*, *auto*)
apply (*drule-tac x* = *r* **in** *realpow-less*)
apply (*drule-tac* [*4*] *x* = *y* **in** *realpow-less*, *auto*)
done

ML
⟪
val nth-realpow-isLub-ex = *thmnth-realpow-isLub-ex*;
val realpow-nth-ge = *thmrealpow-nth-ge*;
val less-isLub-not-isUb = *thmless-isLub-not-isUb*;
val not-isUb-less-ex = *thmnot-isUb-less-ex*;
val realpow-nth-le = *thmrealpow-nth-le*;
val realpow-nth = *thmrealpow-nth*;
val realpow-pos-nth = *thmrealpow-pos-nth*;
val realpow-pos-nth2 = *thmrealpow-pos-nth2*;
val realpow-pos-nth-unique = *thmrealpow-pos-nth-unique*;
⟫

end

24 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 ≤ inverse (real (fact n))*
by (*auto intro*: *order-less-imp-le*)

lemma *fact-diff-Suc* [*rule-format*]:
    *∀ m. n < Suc m --> fact (Suc m − n) = (Suc m − n) * fact (m − n)*
apply (*induct n*, *auto*)
apply (*drule-tac x = m − 1* **in** *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 (*case-tac 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 (*case-tac 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 (*case-tac m*, *auto*)

end

25 EvenOdd: Even and Odd Numbers: Compatibility file for Parity

theory *EvenOdd*
imports *NthRoot*
begin

25.1 General Lemmas About Division

lemma *Suc-times-mod-eq*: *1<k ==> Suc (k * m) mod k = 1*
apply (*induct m*)
apply (*simp-all add*: *mod-Suc*)
done

declare *Suc-times-mod-eq* [*of number-of w*, *standard*, *simp*]

lemma [*simp*]: *n div k ≤ (Suc n) div k*
by (*simp add*: *div-le-mono*)

lemma *Suc-n-div-2-gt-zero* [*simp*]: *(0::nat) < n ==> 0 < (n + 1) div 2*
by *arith*

lemma *div-2-gt-zero* [*simp*]: *(1::nat) < n ==> 0 < n div 2*
by *arith*

lemma *mod-mult-self3* [*simp*]: *(k*n + m) mod n = m mod (n::nat)*
by (*simp add*: *mult-ac add-ac*)

lemma *mod-mult-self4* [*simp*]: *Suc (k*n + m) mod n = Suc m mod n*
proof −
have *Suc (k * n + m) mod n = (k * n + Suc m) mod n* **by** *simp*
also have *... = Suc m mod n* **by** (*rule mod-mult-self3*)

finally show *?thesis* **.**
qed

lemma *mod-Suc-eq-Suc-mod*: *Suc m mod n = Suc (m mod n) mod n*
apply (*subst mod-Suc* [*of m*])
apply (*subst mod-Suc* [*of m mod n*], *simp*)
done

25.2 More Even/Odd Results

lemma *even-mult-two-ex*: *even(n) = (∃m::nat. n = 2*m)*
by (*simp add*: *even-nat-equiv-def2 numeral-2-eq-2*)

lemma *odd-Suc-mult-two-ex*: *odd(n) = (∃m. n = Suc (2*m))*
by (*simp add*: *odd-nat-equiv-def2 numeral-2-eq-2*)

lemma *even-add* [*simp*]: *even(m + n::nat) = (even m = even n)*
by *auto*

lemma *odd-add* [*simp*]: *odd(m + n::nat) = (odd m ≠ odd n)*
by *auto*

lemma *lemma-even-div2* [*simp*]: *even (n::nat) ==> (n + 1) div 2 = n div 2*
apply (*simp add*: *numeral-2-eq-2*)
apply (*subst div-Suc*)
apply (*simp add*: *even-nat-mod-two-eq-zero*)
done

lemma *lemma-not-even-div2* [*simp*]: *~even n ==> (n + 1) div 2 = Suc (n div 2)*
apply (*simp add*: *numeral-2-eq-2*)
apply (*subst div-Suc*)
apply (*simp add*: *odd-nat-mod-two-eq-one*)
done

lemma *even-num-iff*: *0 < n ==> even n = (~ even(n − 1 :: nat))*
by (*case-tac n*, *auto*)

lemma *even-even-mod-4-iff*: *even (n::nat) = even (n mod 4)*
apply (*induct n*, *simp*)
apply (*subst mod-Suc*, *simp*)
done

lemma *lemma-odd-mod-4-div-2*: *n mod 4 = (3::nat) ==> odd((n − 1) div 2)*
apply (*rule-tac t = n* **and** *n1 = 4* **in** *mod-div-equality* [*THEN subst*])
apply (*simp add*: *even-num-iff*)
done

lemma *lemma-even-mod-4-div-2*: *n mod 4 = (1::nat) ==> even ((n − 1) div 2)*

by (*rule-tac t = n* **and** *n1 = 4* **in** *mod-div-equality* [*THEN subst*], *simp*)

ML
⟪
val even-nat-Suc = thmParity.even-nat-Suc;

val even-mult-two-ex = thm even-mult-two-ex;
val odd-Suc-mult-two-ex = thm odd-Suc-mult-two-ex;
val even-add = thm even-add;
val odd-add = thm odd-add;
val Suc-n-div-2-gt-zero = thm Suc-n-div-2-gt-zero;
val div-2-gt-zero = thm div-2-gt-zero;
val even-num-iff = thm even-num-iff;
val nat-mod-div-trivial = thm nat-mod-div-trivial;
val nat-mod-mod-trivial = thm nat-mod-mod-trivial;
val mod-Suc-eq-Suc-mod = thm mod-Suc-eq-Suc-mod;
val even-even-mod-4-iff = thm even-even-mod-4-iff;
val lemma-odd-mod-4-div-2 = thm lemma-odd-mod-4-div-2;
val lemma-even-mod-4-div-2 = thm lemma-even-mod-4-div-2;
⟫

end

26 Transcendental: Power Series, Transcendental Functions etc.

theory *Transcendental*
imports *NthRoot Fact HSeries EvenOdd Lim*
begin

constdefs
root :: [*nat*,*real*] => *real*
root n x == (@*u*. ((*0*::*real*) < *x* −−> *0* < *u*) & (*u* ^ *n* = *x*))

sqrt :: *real* => *real*
sqrt x == *root 2 x*

exp :: *real* => *real*
exp x == ∑ *n*. *inverse*(*real* (*fact n*)) * (*x* ^ *n*)

sin :: *real* => *real*
sin x == ∑ *n*. (*if even*(*n*) *then 0 else*
((− *1*) ^ ((*n* − *Suc 0*) *div 2*))/(*real* (*fact n*))) * *x* ^ *n*

diffs :: (*nat* => *real*) => *nat* => *real*
diffs c == (%*n*. *real* (*Suc n*) * *c*(*Suc n*))

cos :: *real* => *real*
cos x == $\sum$ *n*. (*if even*(*n*) *then* ((− *1*) ^ (*n div 2*))/(*real* (*fact n*))
else 0) ∗ *x* ^ *n*

ln :: *real* => *real*
ln x == (@*u*. *exp u* = *x*)

pi :: *real*
pi == *2* ∗ (@*x*. *0* ≤ (*x*::*real*) & *x* ≤ *2* & *cos x* = *0*)

tan :: *real* => *real*
tan x == (*sin x*)/(*cos x*)

arcsin :: *real* => *real*
arcsin y == (@*x*. −(*pi*/*2*) ≤ *x* & *x* ≤ *pi*/*2* & *sin x* = *y*)

arcos :: *real* => *real*
arcos y == (@*x*. *0* ≤ *x* & *x* ≤ *pi* & *cos x* = *y*)

arctan :: *real* => *real*
arctan y == (@*x*. −(*pi*/*2*) < *x* & *x* < *pi*/*2* & *tan x* = *y*)

lemma *real-root-zero* [*simp*]: *root* (*Suc n*) *0* = *0*
apply (*simp add*: *root-def*)
apply (*safe intro*!: *some-equality power-0-Suc elim*!: *realpow-zero-zero*)
done

lemma *real-root-pow-pos*:
0 < *x* ==> (*root*(*Suc n*) *x*) ^ (*Suc n*) = *x*
apply (*simp add*: *root-def*)
apply (*drule-tac n* = *n* **in** *realpow-pos-nth2*)
apply (*auto intro*: *someI2*)
done

lemma *real-root-pow-pos2*: *0* ≤ *x* ==> (*root*(*Suc n*) *x*) ^ (*Suc n*) = *x*
by (*auto dest*!: *real-le-imp-less-or-eq dest*: *real-root-pow-pos*)

lemma *real-root-pos*:
0 < *x* ==> *root*(*Suc n*) (*x* ^ (*Suc n*)) = *x*
apply (*simp add*: *root-def*)
apply (*rule some-equality*)
apply (*frule-tac* [*2*] *n* = *n* **in** *zero-less-power*)
apply (*auto simp add*: *zero-less-mult-iff*)
apply (*rule-tac x* = *u* **and** *y* = *x* **in** *linorder-cases*)
apply (*drule-tac n1* = *n* **and** *x* = *u* **in** *zero-less-Suc* [*THEN* [*3*] *realpow-less*])
apply (*drule-tac* [*4*] *n1* = *n* **and** *x* = *x* **in** *zero-less-Suc* [*THEN* [*3*] *realpow-less*])
apply (*auto*)

done

lemma *real-root-pos2*: $0 \leq x \Longrightarrow root(Suc\ n)\ (x\ \hat{}\ (Suc\ n)) = x$
by (*auto dest*!: *real-le-imp-less-or-eq real-root-pos*)

lemma *real-root-pos-pos*:
$0 < x \Longrightarrow 0 \leq root(Suc\ n)\ x$
apply (*simp add*: *root-def*)
apply (*drule-tac n* = *n* **in** *realpow-pos-nth2*)
apply (*safe*, *rule someI2*)
apply (*auto intro*!: *order-less-imp-le dest*: *zero-less-power*
simp add: *zero-less-mult-iff*)
done

lemma *real-root-pos-pos-le*: $0 \leq x \Longrightarrow 0 \leq root(Suc\ n)\ x$
by (*auto dest*!: *real-le-imp-less-or-eq dest*: *real-root-pos-pos*)

lemma *real-root-one* [*simp*]: *root* (*Suc n*) *1* = *1*
apply (*simp add*: *root-def*)
apply (*rule some-equality*, *auto*)
apply (*rule ccontr*)
apply (*rule-tac x* = *u* **and** *y* = *1* **in** *linorder-cases*)
apply (*drule-tac n* = *n* **in** *realpow-Suc-less-one*)
apply (*drule-tac* [*4*] *n* = *n* **in** *power-gt1-lemma*)
apply (*auto*)
done

26.1 Square Root

needed because 2 is a binary numeral!

lemma *root-2-eq* [*simp*]: *root 2* = *root* (*Suc* (*Suc 0*))
by (*simp del*: *nat-numeral-0-eq-0 nat-numeral-1-eq-1*
add: *nat-numeral-0-eq-0* [*symmetric*])

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

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

lemma *real-sqrt-pow2-iff* [*iff*]: $((sqrt\ x)^2 = x) = (0 \leq x)$
apply (*simp add*: *sqrt-def*)
apply (*rule iffI*)
apply (*cut-tac r* = *root 2 x* **in** *realpow-two-le*)
apply (*simp add*: *numeral-2-eq-2*)
apply (*subst numeral-2-eq-2*)
apply (*erule real-root-pow-pos2*)
done

lemma [*simp*]: $(sqrt(u2^2 + v2^2))^2 = u2^2 + v2^2$
by (*rule realpow-two-le-add-order* [*THEN real-sqrt-pow2-iff* [*THEN iffD2*]])

lemma *real-sqrt-pow2* [*simp*]: $0 \leq x \Longrightarrow (sqrt\ x)^2 = x$
by (*simp*)

lemma *real-sqrt-abs-abs* [*simp*]: $sqrt|x|\ \hat{}\ 2 = |x|$
by (*rule real-sqrt-pow2-iff* [*THEN iffD2*], *arith*)

lemma *real-pow-sqrt-eq-sqrt-pow*:
$0 \leq x \Longrightarrow (sqrt\ x)^2 = sqrt(x^2)$
apply (*simp add*: *sqrt-def*)
apply (*simp only*: *numeral-2-eq-2 real-root-pow-pos2 real-root-pos2*)
done

lemma *real-pow-sqrt-eq-sqrt-abs-pow2*:
$0 \leq x \Longrightarrow (sqrt\ x)^2 = sqrt(|x|\ \hat{}\ 2)$
by (*simp add*: *real-pow-sqrt-eq-sqrt-pow* [*symmetric*])

lemma *real-sqrt-pow-abs*: $0 \leq x \Longrightarrow (sqrt\ x)^2 = |x|$
apply (*rule real-sqrt-abs-abs* [*THEN subst*])
apply (*rule-tac x1* $= x$ **in** *real-pow-sqrt-eq-sqrt-abs-pow2* [*THEN ssubst*])
apply (*rule-tac* [*2*] *real-pow-sqrt-eq-sqrt-pow* [*symmetric*])
apply (*assumption*, *arith*)
done

lemma *not-real-square-gt-zero* [*simp*]: $(\sim\ (0::real) < x*x) = (x = 0)$
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-gt-zero*: $0 < x \Longrightarrow 0 < sqrt(x)$
apply (*simp add*: *sqrt-def root-def*)
apply (*drule realpow-pos-nth2* [**where** *n=1*], *safe*)
apply (*rule someI2*, *auto*)
done

lemma *real-sqrt-ge-zero*: $0 \leq x \Longrightarrow 0 \leq sqrt(x)$
by (*auto intro*: *real-sqrt-gt-zero simp add*: *order-le-less*)

lemma *real-sqrt-mult-self-sum-ge-zero* [*simp*]: $0 \leq sqrt(x*x + y*y)$
by (*rule real-sqrt-ge-zero* [*OF real-mult-self-sum-ge-zero*])

lemma *sqrt-eqI*: $[|r^2 = a;\ 0 \leq r|] \Longrightarrow sqrt\ a = r$
apply (*unfold sqrt-def root-def*)

apply (*rule someI2* [*of* - *r*], *auto*)
apply (*auto simp add*: *numeral-2-eq-2 simp del*: *realpow-Suc*
dest: *power-inject-base*)
done

lemma *real-sqrt-mult-distrib*:
[| $0 \leq x$; $0 \leq y$ |] ==> $sqrt(x*y) = sqrt(x) * sqrt(y)$
apply (*rule sqrt-eqI*)
apply (*simp add*: *power-mult-distrib*)
apply (*simp add*: *zero-le-mult-iff real-sqrt-ge-zero*)
done

lemma *real-sqrt-mult-distrib2*:
[|$0 \leq x$; $0 \leq y$ |] ==> $sqrt(x*y) = sqrt(x) * sqrt(y)$
by (*auto intro*: *real-sqrt-mult-distrib simp add*: *order-le-less*)

lemma *real-sqrt-sum-squares-ge-zero* [*simp*]: $0 \leq sqrt\ (x^2 + y^2)$
by (*auto intro*!: *real-sqrt-ge-zero*)

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 simp del*: *realpow-Suc*)

lemma *real-sqrt-abs* [*simp*]: $sqrt(x^2) = |x|$
apply (*rule abs-realpow-two* [*THEN subst*])
apply (*rule real-sqrt-abs-abs* [*THEN subst*])
apply (*subst real-pow-sqrt-eq-sqrt-pow*)
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *real-sqrt-abs2* [*simp*]: $sqrt(x*x) = |x|$
apply (*rule realpow-two* [*THEN subst*])
apply (*subst numeral-2-eq-2* [*symmetric*])
apply (*rule real-sqrt-abs*)
done

lemma *real-sqrt-pow2-gt-zero*: $0 < x$ ==> $0 < (sqrt\ x)^2$
by *simp*

lemma *real-sqrt-not-eq-zero*: $0 < x$ ==> $sqrt\ x \neq 0$
apply (*frule real-sqrt-pow2-gt-zero*)
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *real-inv-sqrt-pow2*: $0 < x$ ==> $inverse\ (sqrt(x))\ \hat{}\ 2 = inverse\ x$

by (*cut-tac n1 = 2* **and** *a1 = sqrt x* **in** *power-inverse* [*symmetric*], *auto*)

lemma *real-sqrt-eq-zero-cancel*: [| *0 ≤ x*; *sqrt(x) = 0*|] ==> *x = 0*
apply (*drule real-le-imp-less-or-eq*)
apply (*auto dest*: *real-sqrt-not-eq-zero*)
done

lemma *real-sqrt-eq-zero-cancel-iff* [*simp*]: *0 ≤ x ==> ((sqrt x = 0) = (x=0))*
by (*auto simp add*: *real-sqrt-eq-zero-cancel*)

lemma *real-sqrt-sum-squares-ge1* [*simp*]: *x ≤ sqrt(x² + y²)*
apply (*subgoal-tac x ≤ 0 | 0 ≤ x*, *safe*)
apply (*rule real-le-trans*)
apply (*auto simp del*: *realpow-Suc*)
apply (*rule-tac n = 1* **in** *realpow-increasing*)
apply (*auto simp add*: *numeral-2-eq-2* [*symmetric*] *simp del*: *realpow-Suc*)
done

lemma *real-sqrt-sum-squares-ge2* [*simp*]: *y ≤ sqrt(z² + y²)*
apply (*simp* (*no-asm*) *add*: *real-add-commute del*: *realpow-Suc*)
done

lemma *real-sqrt-ge-one*: *1 ≤ x ==> 1 ≤ sqrt x*
apply (*rule-tac n = 1* **in** *realpow-increasing*)
apply (*auto simp add*: *numeral-2-eq-2* [*symmetric*] *real-sqrt-ge-zero simp*
del: *realpow-Suc*)
done

26.2 Exponential Function

lemma *summable-exp*: *summable* (%*n*. *inverse* (*real* (*fact n*)) * *x* ^ *n*)
apply (*cut-tac 'a = real* **in** *zero-less-one* [*THEN dense*], *safe*)
apply (*cut-tac x = r* **in** *reals-Archimedean3*, *auto*)
apply (*drule-tac x = |x|* **in** *spec*, *safe*)
apply (*rule-tac N = n* **and** *c = r* **in** *ratio-test*)
apply (*auto simp add*: *abs-mult mult-assoc* [*symmetric*] *simp del*: *fact-Suc*)
apply (*rule mult-right-mono*)
apply (*rule-tac b1 = |x|* **in** *mult-commute* [*THEN ssubst*])
apply (*subst fact-Suc*)
apply (*subst real-of-nat-mult*)
apply (*auto*)
apply (*auto simp add*: *mult-assoc* [*symmetric*] *positive-imp-inverse-positive*)
apply (*rule order-less-imp-le*)
apply (*rule-tac z1 = real* (*Suc na*) **in** *real-mult-less-iff1* [*THEN iffD1*])
apply (*auto simp add*: *real-not-refl2* [*THEN not-sym*] *mult-assoc*)
apply (*erule order-less-trans*)
apply (*auto simp add*: *mult-less-cancel-left mult-ac*)
done

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* [*simp*]:
(*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* [*simp*]:
0 < *n* −−>
(− *1*) ^ (*n div 2*)/(*real* (*fact n*)) ∗ *0* ^ *n* = *0*
by (*induct n*, *auto*)

lemma *lemma-STAR-cos1* [*simp*]:
0 < *n* −−>
(−*1*) ^ (*n div 2*)/(*real* (*fact n*)) ∗ *0* ^ *n* = *0*
by (*induct n*, *auto*)

lemma *lemma-STAR-cos2* [*simp*]:
(∑ *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*: (%*n*. *inverse* (*real* (*fact n*)) ∗ *x* ^ *n*) *sums exp*(*x*)
apply (*simp add*: *exp-def*)
apply (*rule summable-exp* [*THEN summable-sums*])
done

lemma *sin-converges*:

(%n. (if even n then 0
else (− 1) ^ ((n − Suc 0) div 2)/(real (fact n))) *
x ^ n) sums sin(x)
apply (simp add: sin-def)
apply (rule summable-sin [THEN summable-sums])
done

lemma cos-converges:
(%n. (if even n then
(− 1) ^ (n div 2)/(real (fact n))
else 0) * x ^ n) sums cos(x)
apply (simp add: cos-def)
apply (rule summable-cos [THEN summable-sums])
done

lemma lemma-realpow-diff [rule-format (no-asm)]:
p ≤ n −−> y ^ (Suc n − p) = ((y::real) ^ (n − p)) * y
apply (induct n, auto)
apply (subgoal-tac p = Suc n)
apply (simp (no-asm-simp), auto)
apply (drule sym)
apply (simp add: Suc-diff-le mult-commute realpow-Suc [symmetric]
del: realpow-Suc)
done

26.3 Properties of Power Series

lemma lemma-realpow-diff-sumr:
(∑ p=0..<Suc n. (x ^ p) * y ^ ((Suc n) − p)) =
y * (∑ p=0..<Suc n. (x ^ p) * (y ^ (n − p))::real)
by (auto simp add: setsum-mult lemma-realpow-diff mult-ac
simp del: setsum-op-ivl-Suc cong: strong-setsum-cong)

lemma lemma-realpow-diff-sumr2:
x ^ (Suc n) − y ^ (Suc n) =
(x − y) * (∑ p=0..<Suc n. (x ^ p) * (y ^(n − p))::real)
apply (induct n, simp)
apply (auto simp del: setsum-op-ivl-Suc)
apply (subst setsum-op-ivl-Suc)
apply (drule sym)
apply (auto simp add: lemma-realpow-diff-sumr right-distrib diff-minus mult-ac
simp del: setsum-op-ivl-Suc)
done

lemma lemma-realpow-rev-sumr:
(∑ p=0..<Suc n. (x ^ p) * (y ^ (n − p))) =
(∑ p=0..<Suc n. (x ^ (n − p)) * (y ^ p)::real)
apply (case-tac x = y)
apply (auto simp add: mult-commute power-add [symmetric] simp del: setsum-op-ivl-Suc)

apply (*rule-tac c1 = x − y* **in** *real-mult-left-cancel* [*THEN iffD1*])
apply (*rule-tac* [*2*] *minus-minus* [*THEN subst*], *simp*)
apply (*subst minus-mult-left*)
apply (*simp add*: *lemma-realpow-diff-sumr2* [*symmetric*] *del*: *setsum-op-ivl-Suc*)
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*:
[| *summable* (%*n*. *f*(*n*) ∗ (*x* ^ *n*)); |*z*| < |*x*| |]
==> *summable* (%*n*. |*f*(*n*)| ∗ (*z* ^ *n*))
apply (*drule summable-LIMSEQ-zero*)
apply (*drule convergentI*)
apply (*simp add*: *Cauchy-convergent-iff* [*symmetric*])
apply (*drule Cauchy-Bseq*)
apply (*simp add*: *Bseq-def*, *safe*)
apply (*rule-tac g* = %*n*. *K* ∗ |*z* ^ *n*| ∗ *inverse* (|*x* ^ *n*|) **in** *summable-comparison-test*)
apply (*rule-tac x* = *0* **in** *exI*, *safe*)
apply (*subgoal-tac* *0* < |*x* ^ *n*|)
apply (*rule-tac c*=|*x* ^ *n*| **in** *mult-right-le-imp-le*)
apply (*auto simp add*: *mult-assoc power-abs abs-mult*)
prefer *2*
apply (*drule-tac x* = *0* **in** *spec*, *force*)
apply (*auto simp add*: *power-abs mult-ac*)
apply (*rule-tac a2* = *z* ^ *n*
in *abs-ge-zero* [*THEN real-le-imp-less-or-eq*, *THEN disjE*])
apply (*auto intro*!: *mult-right-mono simp add*: *mult-assoc* [*symmetric*] *power-abs summable-def power-0-left*)
apply (*rule-tac x* = *K* ∗ *inverse* (*1* − (|*z*| ∗ *inverse* (|*x*|))) **in** *exI*)
apply (*auto intro*!: *sums-mult simp add*: *mult-assoc*)
apply (*subgoal-tac* |*z* ^ *n*| ∗ *inverse* (|*x*| ^ *n*) = (|*z*| ∗ *inverse* (|*x*|)) ^ *n*)
apply (*auto simp add*: *power-abs* [*symmetric*])
apply (*subgoal-tac x* ≠ *0*)
apply (*subgoal-tac* [*3*] *x* ≠ *0*)
apply (*auto simp del*: *abs-inverse*
simp add: *abs-inverse* [*symmetric*] *realpow-not-zero*
abs-mult [*symmetric*] *power-inverse power-mult-distrib* [*symmetric*])
apply (*auto intro*!: *geometric-sums simp add*: *power-abs inverse-eq-divide*)
done

lemma *powser-inside*:
[| *summable* (%*n*. *f*(*n*) ∗ (*x* ^ *n*)); |*z*| < |*x*| |]
==> *summable* (%*n*. *f*(*n*) ∗ (*z* ^ *n*))
apply (*drule-tac z* = |*z*| **in** *powser-insidea*)
apply (*auto intro*: *summable-rabs-cancel simp add*: *abs-mult power-abs* [*symmetric*])
done

26.4 Differentiation of Power Series

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*. *real n* ∗ *c*(*n*) ∗ (*x* ^ (*n* − *Suc 0*))) +
(*real 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*. *real n* ∗ *c*(*n*) ∗ (*x* ^ (*n* − *Suc 0*))) =
(∑ *n*=*0*..<*n*. (*diffs c*)(*n*) ∗ (*x* ^ *n*)) −
(*real n* ∗ *c*(*n*) ∗ *x* ^ (*n* − *Suc 0*))
by (*auto simp add*: *lemma-diffs*)

lemma *diffs-equiv*:
summable (%*n*. (*diffs c*)(*n*) ∗ (*x* ^ *n*)) ==>
(%*n*. *real n* ∗ *c*(*n*) ∗ (*x* ^ (*n* − *Suc 0*))) *sums*
(∑ *n*. (*diffs c*)(*n*) ∗ (*x* ^ *n*))
apply (*subgoal-tac* (%*n*. *real 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*=(λ*n*. ∑ *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

26.5 Term-by-Term Differentiability of Power Series

lemma *lemma-termdiff1*:
(∑ *p*=*0*..<*m*. (((*z* + *h*) ^ (*m* − *p*)) ∗ (*z* ^ *p*)) − (*z* ^ *m*)) =
(∑ *p*=*0*..<*m*. (*z* ^ *p*) ∗ (((*z* + *h*) ^ (*m* − *p*)) − (*z* ^ (*m* − *p*)))::*real*)
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 *lemma-termdiff2*:
$h \neq 0 \Longrightarrow$
$(((z + h) \hat{} n) - (z \hat{} n)) * inverse\ h - real\ n * (z \hat{} (n - Suc\ 0)) =$
$h * (\sum p{=}0..{<} n - Suc\ 0.\ (z \hat{} p) *$
$(\sum q{=}0..{<} (n - Suc\ 0) - p.\ ((z + h) \hat{} q) * (z \hat{} (((n - 2) - p) - q))))$
apply (*rule real-mult-left-cancel* [*THEN iffD1*], *simp* (*no-asm-simp*))
apply (*simp add*: *right-diff-distrib mult-ac*)
apply (*simp add*: *mult-assoc* [*symmetric*])
apply (*case-tac n*)
apply (*auto simp add*: *lemma-realpow-diff-sumr2*
right-diff-distrib [*symmetric*] *mult-assoc*
simp del: *realpow-Suc setsum-op-ivl-Suc*)
apply (*auto simp add*: *lemma-realpow-rev-sumr simp del*: *setsum-op-ivl-Suc*)
apply (*auto simp add*: *real-of-nat-Suc sumr-diff-mult-const left-distrib*
sumdiff lemma-termdiff1 setsum-mult)
apply (*auto intro*!: *setsum-cong*[*OF refl*] *simp add*: *diff-minus real-add-assoc*)
apply (*simp add*: *diff-minus* [*symmetric*] *less-iff-Suc-add*)
apply (*auto simp add*: *setsum-mult lemma-realpow-diff-sumr2 mult-ac simp*
del: *setsum-op-ivl-Suc realpow-Suc*)
done

lemma *lemma-termdiff3*:
$[|\ h \neq 0; |z| \leq K; |z + h| \leq K\ |]$
$\Longrightarrow abs\ (((z + h) \hat{} n - z \hat{} n) * inverse\ h - real\ n * z \hat{} (n - Suc\ 0))$
$\leq real\ n * real\ (n - Suc\ 0) * K \hat{} (n - 2) * |h|$
apply (*subst lemma-termdiff2*, *assumption*)
apply (*simp add*: *mult-commute abs-mult*)
apply (*simp add*: *mult-commute* [*of - K* $\hat{}$ $(n - 2)$])
apply (*rule setsum-abs* [*THEN real-le-trans*])
apply (*simp add*: *mult-assoc* [*symmetric*] *abs-mult*)
apply (*simp add*: *mult-commute* [*of - real* $(n - Suc\ 0)$])
apply (*auto intro*!: *real-setsum-nat-ivl-bounded*)
apply (*case-tac n*, *auto*)
apply (*drule less-add-one*)

apply *clarify*
apply (*subgoal-tac K* $\hat{}$ *p* $*$ *K* $\hat{}$ *d* $*$ *real* (*Suc* (*Suc* (*p* $+$ *d*))) $=$
K $\hat{}$ *p* $*$ (*real* (*Suc* (*Suc* (*p* $+$ *d*))) $*$ *K* $\hat{}$ *d*))
apply (*simp* (*no-asm-simp*) *add*: *power-add del*: *setsum-op-ivl-Suc*)
apply (*auto intro*!: *mult-mono simp del*: *setsum-op-ivl-Suc*)
apply (*auto intro*!: *power-mono simp add*: *power-abs*
simp del: *setsum-op-ivl-Suc*)
apply (*rule-tac j* $=$ *real* (*Suc d*) $*$ (*K* $\hat{}$ *d*) **in** *real-le-trans*)
apply (*subgoal-tac* [*2*] $0 \leq K$)
apply (*drule-tac* [*2*] $n = d$ **in** *zero-le-power*)
apply (*auto simp del*: *setsum-op-ivl-Suc*)
apply (*rule setsum-abs* [*THEN real-le-trans*])
apply (*rule real-setsum-nat-ivl-bounded*)

apply (*auto dest*!: *less-add-one intro*!: *mult-mono simp add*: *power-add abs-mult*)
apply (*auto intro*!: *power-mono zero-le-power simp add*: *power-abs*, *arith*+)
done

lemma *lemma-termdiff4*:
[| *0* < *k*;
(∀ *h*. *0* < |*h*| & |*h*| < *k* −−> |*f h*| ≤ *K* ∗ |*h*|) |]
==> *f* −− *0* −−> *0*
apply (*simp add*: *LIM-def*, *auto*)
apply (*subgoal-tac* *0* ≤ *K*)
prefer *2*
apply (*drule-tac* *x* = *k*/*2* **in** *spec*)
apply (*simp add*:)
apply (*subgoal-tac* *0* ≤ *K*∗*k*, *simp add*: *zero-le-mult-iff*)
apply (*force intro*: *order-trans* [*of* - |*f* (*k* / *2*)| ∗ *2*])
apply (*drule real-le-imp-less-or-eq*, *auto*)
apply (*subgoal-tac* *0* < (*r* ∗ *inverse K*) / *2*)
apply (*drule-tac* *?d1.0* = (*r* ∗ *inverse K*) / *2* **and** *?d2.0* = *k* **in** *real-lbound-gt-zero*)
apply (*auto simp add*: *positive-imp-inverse-positive zero-less-mult-iff zero-less-divide-iff*)
apply (*rule-tac* *x* = *e* **in** *exI*, *auto*)
apply (*rule-tac* *y* = *K* ∗ |*x*| **in** *order-le-less-trans*)
apply (*force*)
apply (*rule-tac* *y* = *K* ∗ *e* **in** *order-less-trans*)
apply (*simp add*: *mult-less-cancel-left*)
apply (*rule-tac* *c* = *inverse K* **in** *mult-right-less-imp-less*)
apply (*auto simp add*: *mult-ac*)
done

lemma *lemma-termdiff5*:
[| *0* < *k*;
summable f;
∀ *h*. *0* < |*h*| & |*h*| < *k* −−>
(∀ *n*. *abs*(*g*(*h*) (*n*::*nat*)) ≤ (*f*(*n*) ∗ |*h*|)) |]
==> (%*h*. *suminf*(*g h*)) −− *0* −−> *0*
apply (*drule summable-sums*)
apply (*subgoal-tac* ∀ *h*. *0* < |*h*| & |*h*| < *k* −−> |*suminf* (*g h*)| ≤ *suminf f* ∗ |*h*|)
apply (*auto intro*!: *lemma-termdiff4 simp add*: *sums-summable* [*THEN suminf-mult*, *symmetric*])
apply (*subgoal-tac summable* (%*n*. *f n* ∗ |*h*|))
prefer *2*
apply (*simp add*: *summable-def*)
apply (*rule-tac* *x* = *suminf f* ∗ |*h*| **in** *exI*)
apply (*drule-tac* *c* = |*h*| **in** *sums-mult*)
apply (*simp add*: *mult-ac*)
apply (*subgoal-tac summable* (%*n*. *abs* (*g* (*h*::*real*) (*n*::*nat*))))
apply (*rule-tac* [*2*] *g* = %*n*. *f n* ∗ |*h*| **in** *summable-comparison-test*)
apply (*rule-tac* [*2*] *x* = *0* **in** *exI*, *auto*)
apply (*rule-tac* *j* = ∑ *n*. |*g h n*| **in** *real-le-trans*)
apply (*auto intro*: *summable-rabs summable-le simp add*: *sums-summable* [*THEN*

suminf-mult2])
done

FIXME: Long proofs

lemma *termdiffs-aux*:
[|*summable* (λ*n*. *diffs* (*diffs c*) *n* ∗ *K* ^ *n*); |*x*| < |*K*| |]
==> (λ*h*. ∑ *n*. *c n* ∗
(((*x* + *h*) ^ *n* − *x* ^ *n*) ∗ *inverse h* −
real n ∗ *x* ^ (*n* − *Suc 0*)))
−− *0* −−> *0*
apply (*drule dense*, *safe*)
apply (*frule real-less-sum-gt-zero*)
apply (*drule-tac*
f = %*n*. |*c n*| ∗ *real n* ∗ *real* (*n* − *Suc 0*) ∗ (*r* ^ (*n* − *2*))
and *g* = %*h n*. *c* (*n*) ∗ (((((*x* + *h*) ^ *n*) − (*x* ^ *n*)) ∗ *inverse h*)
− (*real n* ∗ (*x* ^ (*n* − *Suc 0*))))
in *lemma-termdiff5*)
apply (*auto simp add*: *add-commute*)
apply (*subgoal-tac summable* (%*n*. |*diffs* (*diffs c*) *n*| ∗ (*r* ^ *n*)))
apply (*rule-tac* [*2*] *x* = *K* **in** *powser-insidea*, *auto*)
apply (*subgoal-tac* [*2*] |*r*| = *r*, *auto*)
apply (*rule-tac* [*2*] *y1* = |*x*| **in** *order-trans* [*THEN abs-of-nonneg*], *auto*)
apply (*simp add*: *diffs-def mult-assoc* [*symmetric*])
apply (*subgoal-tac*
∀ *n*. *real* (*Suc n*) ∗ *real* (*Suc* (*Suc n*)) ∗ |*c* (*Suc* (*Suc n*))| ∗ (*r* ^ *n*)
= *diffs* (*diffs* (%*n*. |*c n*|)) *n* ∗ (*r* ^ *n*))
apply (*auto simp add*: *abs-mult*)
apply (*drule diffs-equiv*)
apply (*drule sums-summable*)
apply (*simp-all add*: *diffs-def*)
apply (*simp add*: *diffs-def mult-ac*)
apply (*subgoal-tac* (%*n*. *real n* ∗ (*real* (*Suc n*) ∗ (|*c* (*Suc n*)| ∗ (*r* ^ (*n* − *Suc* *0*))))) = (%*n*. *diffs* (%*m*. *real* (*m* − *Suc 0*) ∗ |*c m*| ∗ *inverse r*) *n* ∗ (*r* ^ *n*)))
apply *auto*
prefer *2*
apply (*rule ext*)
apply (*simp add*: *diffs-def*)
apply (*case-tac n*, *auto*)

23

apply (*drule abs-ge-zero* [*THEN order-le-less-trans*])
apply (*simp add*: *mult-ac*)
apply (*drule abs-ge-zero* [*THEN order-le-less-trans*])
apply (*simp add*: *mult-ac*)
apply (*drule diffs-equiv*)
apply (*drule sums-summable*)
apply (*subgoal-tac*
summable
(λ*n*. *real n* ∗ (*real* (*n* − *Suc 0*) ∗ |*c n*| ∗ *inverse r*) ∗

```
            r ^ (n - Suc 0)) =
        summable
         (λn. real n * (|c n| * (real (n - Suc 0) * r ^ (n - 2)))))
apply simp
apply (rule-tac f = summable in arg-cong, rule ext)

33

apply (case-tac n, auto)
apply (case-tac nat, auto)
apply (drule abs-ge-zero [THEN order-le-less-trans], auto)
apply (drule abs-ge-zero [THEN order-le-less-trans])
apply (simp add: mult-assoc)
apply (rule mult-left-mono)
 prefer 2 apply arith
apply (subst add-commute)
apply (simp (no-asm) add: mult-assoc [symmetric])
apply (rule lemma-termdiff3)
apply (auto intro: abs-triangle-ineq [THEN order-trans], arith)
done

lemma termdiffs:
   [| summable(%n. c(n) * (K ^ n));
      summable(%n. (diffs c)(n) * (K ^ n));
      summable(%n. (diffs(diffs c))(n) * (K ^ n));
      |x| < |K| |]
   ==> DERIV (%x. ∑ n. c(n) * (x ^ n))  x :>
         (∑ n. (diffs c)(n) * (x ^ n))
apply (simp add: deriv-def)
apply (rule-tac g = %h. ∑ n. ((c (n) * ( (x + h) ^ n)) - (c (n) * (x ^ n))) *
inverse h in LIM-trans)
apply (simp add: LIM-def, safe)
apply (rule-tac x = |K| - |x| in exI)
apply (auto simp add: less-diff-eq)
apply (drule abs-triangle-ineq [THEN order-le-less-trans])
apply (rule-tac y = 0 in order-le-less-trans, auto)
apply (subgoal-tac  (%n. (c n) * (x ^ n)) sums (∑ n. (c n) * (x ^ n)) & (%n. (c
n) * ((x + xa) ^ n)) sums (∑ n. (c n) * ( (x + xa) ^ n)))
apply (auto intro!: summable-sums)
apply (rule-tac [2] powser-inside, rule-tac [4] powser-inside)
apply (auto simp add: add-commute)
apply (drule-tac x=(λn. c n * (xa + x) ^ n) in sums-diff, assumption)
apply (drule-tac f = (%n. c n * (xa + x) ^ n - c n * x ^ n)  and c = inverse
xa in sums-mult)
apply (rule sums-unique)
apply (simp add: diff-def divide-inverse add-ac mult-ac)
apply (rule LIM-zero-cancel)
apply (rule-tac g = %h. ∑ n. c (n) * ((( ((x + h) ^ n) - (x ^ n)) * inverse h)
- (real n * (x ^ (n - Suc 0))))) in LIM-trans)
 prefer 2 apply (blast intro: termdiffs-aux)
```

apply (*simp add*: *LIM-def*, *safe*)
apply (*rule-tac* $x = |K| - |x|$ **in** *exI*)
apply (*auto simp add*: *less-diff-eq*)
apply (*drule abs-triangle-ineq* [*THEN order-le-less-trans*])
apply (*rule-tac* $y = 0$ **in** *order-le-less-trans*, *auto*)
apply (*subgoal-tac summable* (%*n*. (*diffs c*) (*n*) ∗ (*x* ^ *n*)))
apply (*rule-tac* [*2*] *powser-inside*, *auto*)
apply (*drule-tac* $c = c$ **and** $x = x$ **in** *diffs-equiv*)
apply (*frule sums-unique*, *auto*)
apply (*subgoal-tac* (%*n*. (*c n*) ∗ (*x* ^ *n*)) *sums* (∑ *n*. (*c n*) ∗ (*x* ^ *n*)) & (%*n*. (*c n*) ∗ ((*x* + *xa*) ^ *n*)) *sums* (∑ *n*. (*c n*) ∗ ((*x* + *xa*) ^ *n*)))
apply *safe*
apply (*auto intro*!: *summable-sums*)
apply (*rule-tac* [*2*] *powser-inside*, *rule-tac* [*4*] *powser-inside*)
apply (*auto simp add*: *add-commute*)
apply (*frule-tac* $x = (\%n.\ c\ n * (xa + x)\ \hat{}\ n)$ **and** $y = (\%n.\ c\ n * x\ \hat{}\ n)$ **in** *sums-diff*, *assumption*)
apply (*simp add*: *suminf-diff* [*OF sums-summable sums-summable*]
right-diff-distrib [*symmetric*])
apply (*subst suminf-diff*)
apply (*rule summable-mult2*)
apply (*erule sums-summable*)
apply (*erule sums-summable*)
apply (*simp add*: *ring-eq-simps*)
done

26.6 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*] *del*: *mult-Suc*)

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 simp del*: *mult-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 (*auto intro*!: *ext*
simp add: *diffs-def divide-inverse simp del*: *mult-Suc*)

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*
simp del: *mult-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 (*auto intro*!: *ext*
simp add: *diffs-def divide-inverse odd-Suc-mult-two-ex*
simp del: *mult-Suc*)

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* = (%*x*. ∑ *n*. *inverse* (*real* (*fact n*)) ∗ *x* ^ *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* (%*u*. ∑ *n*. *inverse* (*real* (*fact n*)) ∗ *u* ^ *n*) *x* :> (∑ *n*. *diffs* (%*n*. *inverse* (*real* (*fact n*))) *n* ∗ *x* ^ *n*))
apply (*rule-tac* [*2*] *K* = *1* + |*x*| **in** *termdiffs*)
apply (*auto intro*: *exp-converges* [*THEN sums-summable*] *simp add*: *exp-fdiffs*, *arith*)
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*. ∑ *n*.
(*if even n then* (− *1*) ^ (*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*, *arith*)
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*, *arith*)
done

26.7 Properties of the Exponential Function

lemma *exp-zero* [*simp*]: *exp 0 = 1*
proof −
  have $(\sum n = 0..<1.\ inverse\ (real\ (fact\ n)) * 0 \hat{}\ n) =$
      $(\sum n.\ inverse\ (real\ (fact\ n)) * 0 \hat{}\ n)$
    by (*rule series-zero* [*rule-format*, *THEN sums-unique*],
      case-tac m, *auto*)
  thus *?thesis* **by** (*simp add*: *exp-def*)
qed

lemma *exp-ge-add-one-self-aux*: *0 ≤ x ==> (1 + x) ≤ exp(x)*
apply (*drule real-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-power zero-le-mult-iff*)
done

lemma *exp-gt-one* [*simp*]: *0 < x ==> 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* ∘ (λ*x. x* + *y*)) *x :> exp* (*x* + *y*) ∗ (*1*+*0*)
    by (*fast intro*: *DERIV-chain DERIV-add DERIV-exp DERIV-Id 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-Id*)
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* (λ*x*. *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*
qed

lemma *exp-add-mult-minus* [*simp*]: *exp*(*x* + *y*)∗*exp*(−*x*) = *exp*(*y*)
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*
proof −
have *exp* (*x* + *0*) ∗ *exp* (− *x*) = *exp 0* **by** (*rule exp-add-mult-minus*)
thus *?thesis* **by** *simp*
qed

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*])

lemma *exp-add*: *exp*(*x* + *y*) = *exp*(*x*) ∗ *exp*(*y*)
proof −
have *exp x* ∗ *exp y* = *exp x* ∗ (*exp* (*x* + *y*) ∗ *exp* (− *x*)) **by** *simp*
thus *?thesis* **by** (*simp* (*no-asm-simp*) *add*: *mult-ac*)
qed

Proof: because every exponential can be seen as a square.

lemma *exp-ge-zero* [*simp*]: *0* ≤ *exp x*

apply (*rule-tac* $t = x$ **in** *real-sum-of-halves* [*THEN subst*])
apply (*subst exp-add*, *auto*)
done

lemma *exp-not-eq-zero* [*simp*]: $exp\ x \neq 0$
apply (*cut-tac* $x = x$ **in** *exp-mult-minus2*)
apply (*auto simp del*: *exp-mult-minus2*)
done

lemma *exp-gt-zero* [*simp*]: $0 < exp\ x$
by (*simp add*: *order-less-le*)

lemma *inv-exp-gt-zero* [*simp*]: $0 < inverse(exp\ x)$
by (*auto intro*: *positive-imp-inverse-positive*)

lemma *abs-exp-cancel* [*simp*]: $|exp\ x| = exp\ x$
by *auto*

lemma *exp-real-of-nat-mult*: $exp(real\ n * x) = exp(x)\ \hat{}\ n$
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*:
  assumes *xy*: $x < y$ **shows** $exp\ x < exp\ y$
proof −
  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*: $exp\ x < exp\ y \Longrightarrow x < y$
apply (*simp add*: *linorder-not-le* [*symmetric*])
apply (*auto simp add*: *order-le-less exp-less-mono*)
done

lemma *exp-less-cancel-iff* [*iff*]: $(exp(x) < exp(y)) = (x < y)$
by (*auto intro*: *exp-less-mono exp-less-cancel*)

lemma *exp-le-cancel-iff* [*iff*]: $(exp(x) \leq exp(y)) = (x \leq y)$
by (*auto simp add*: *linorder-not-less* [*symmetric*])

lemma *exp-inj-iff* [*iff*]: $(exp\ x = exp\ y) = (x = y)$
by (*simp add*: *order-eq-iff*)

lemma *lemma-exp-total*: $1 \leq y \Longrightarrow \exists x.\ 0 \leq x\ \&\ x \leq y - 1\ \&\ exp(x) = y$
apply (*rule IVT*)
apply (*auto intro*: *DERIV-exp* [*THEN DERIV-isCont*] *simp add*: *le-diff-eq*)
apply (*subgoal-tac* $1 + (y - 1) \leq exp\ (y - 1)$)
apply *simp*
apply (*rule exp-ge-add-one-self-aux*, *simp*)
done

lemma *exp-total*: $0 < y \Longrightarrow \exists x.\ exp\ x = y$
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

26.8 Properties of the Logarithmic Function

lemma *ln-exp*[*simp*]: $ln(exp\ x) = x$
by (*simp add*: *ln-def*)

lemma *exp-ln-iff*[*simp*]: $(exp(ln\ x) = x) = (0 < x)$
apply (*auto dest*: *exp-total*)
apply (*erule subst*, *simp*)
done

lemma *ln-mult*: $[|\ 0 < x;\ 0 < y\ |] \Longrightarrow ln(x * y) = ln(x) + ln(y)$
apply (*rule exp-inj-iff* [*THEN iffD1*])
apply (*frule real-mult-order*)
apply (*auto simp add*: *exp-add exp-ln-iff* [*symmetric*] *simp del*: *exp-inj-iff exp-ln-iff*)
done

lemma *ln-inj-iff*[*simp*]: $[|\ 0 < x;\ 0 < y\ |] \Longrightarrow (ln\ x = ln\ y) = (x = y)$
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*: $0 < x \Longrightarrow ln(inverse\ x) = -\ ln\ x$
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*:
$[|0 < x;\ 0 < y|] \Longrightarrow ln(x/y) = ln\ x - ln\ y$
apply (*simp add*: *divide-inverse*)
apply (*auto simp add*: *positive-imp-inverse-positive ln-mult ln-inverse*)
done

lemma *ln-less-cancel-iff* [*simp*]: $[|\ 0 < x;\ 0 < y|] \Longrightarrow (ln\ x < ln\ y) = (x < y)$
apply (*simp only*: *exp-ln-iff* [*symmetric*])
apply (*erule subst*)+
apply *simp*
done

lemma *ln-le-cancel-iff* [*simp*]: $[|\ 0 < x;\ 0 < y|] \Longrightarrow (ln\ x \leq ln\ y) = (x \leq y)$
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 exp-ln-iff* [*symmetric*] *del*: *exp-ln-iff*)
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 ≤ ln x* **by** *simp*
thus *?thesis* **by** (*simp add*: *x del*: *ln-one*)
qed

lemma *ln-ge-zero-iff* [*simp*]: *0 < x ==> (0 ≤ ln x) = (1 ≤ x)*
by (*blast intro*: *ln-ge-zero ln-ge-zero-imp-ge-one*)

lemma *ln-less-zero-iff* [*simp*]: *0 < x ==> (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 exp-ln-iff* [*symmetric*] *del*: *exp-ln-iff*)
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*

26.9 Basic Properties of the Trigonometric Functions

lemma *sin-zero* [*simp*]: *sin 0 = 0*
by (*auto intro*!: *sums-unique* [*symmetric*] *LIMSEQ-const*
simp add: *sin-def sums-def simp del*: *power-0-left*)

lemma *lemma-series-zero2*:
(∀ *m*. *n ≤ m* −−> *f m = 0*) −−> *f sums setsum f* {*0..<n*}
by (*auto intro*: *series-zero*)

lemma *cos-zero* [*simp*]: *cos 0 = 1*
apply (*simp add*: *cos-def*)
apply (*rule sums-unique* [*symmetric*])
apply (*cut-tac n = 1* **and** *f* = (%*n*. (*if even n then* (− *1*) ^ (*n div 2*) / (*real* (*fact n*)) *else 0*) ∗ *0* ^ *n*) **in** *lemma-series-zero2*)
apply *auto*
done

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*]:
  ∀ *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 real-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* ≤ *y* |] ==> *1* < *x* + (*y*::*real*)
by *arith*

lemma *abs-sin-le-one* [*simp*]: |*sin x*| ≤ *1*
apply (*auto simp add*: *linorder-not-less* [*symmetric*])
apply (*drule-tac n* = *Suc 0* **in** *power-gt1*)
apply (*auto simp del*: *realpow-Suc*)
apply (*drule-tac r1* = *cos x* **in** *realpow-two-le* [*THEN* [*2*] *real-gt-one-ge-zero-add-less*])
apply (*simp add*: *numeral-2-eq-2* [*symmetric*] *del*: *realpow-Suc*)
done

lemma *sin-ge-minus-one* [*simp*]: −*1* ≤ *sin x*
apply (*insert abs-sin-le-one* [*of x*])
apply (*simp add*: *abs-le-interval-iff del*: *abs-sin-le-one*)
done

lemma *sin-le-one* [*simp*]: *sin x* ≤ *1*
apply (*insert abs-sin-le-one* [*of x*])
apply (*simp add*: *abs-le-interval-iff del*: *abs-sin-le-one*)
done

lemma *abs-cos-le-one* [*simp*]: |*cos x*| ≤ *1*
apply (*auto simp add*: *linorder-not-less* [*symmetric*])
apply (*drule-tac n* = *Suc 0* **in** *power-gt1*)
apply (*auto simp del*: *realpow-Suc*)
apply (*drule-tac r1* = *sin x* **in** *realpow-two-le* [*THEN* [*2*] *real-gt-one-ge-zero-add-less*])
apply (*simp add*: *numeral-2-eq-2* [*symmetric*] *del*: *realpow-Suc*)
done

lemma *cos-ge-minus-one* [*simp*]: −*1* ≤ *cos x*
apply (*insert abs-cos-le-one* [*of x*])
apply (*simp add*: *abs-le-interval-iff del*: *abs-cos-le-one*)
done

lemma *cos-le-one* [*simp*]: *cos x* ≤ *1*
apply (*insert abs-cos-le-one* [*of x*])
apply (*simp add*: *abs-le-interval-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-Id 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 (*auto dest*!: *real-sum-squares-cancel-a*
    simp add: *numeral-2-eq-2 real-add-eq-0-iff 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 (*auto dest*!: *real-sum-squares-cancel-a*
simp add: *numeral-2-eq-2 real-add-eq-0-iff 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 (*auto simp add*: *numeral-2-eq-2*)
done

lemma *sin-minus* [*simp*]: *sin* (−*x*) = −*sin*(*x*)
apply (*cut-tac x* = *x* **in** *sin-cos-minus*)
apply (*auto dest*!: *real-sum-squares-cancel-a*
simp add: *numeral-2-eq-2 real-add-eq-0-iff simp del*: *sin-cos-minus*)
done

lemma *cos-minus* [*simp*]: *cos* (−*x*) = *cos*(*x*)
apply (*cut-tac x* = *x* **in** *sin-cos-minus*)
apply (*auto dest*!: *real-sum-squares-cancel-a*
simp add: *numeral-2-eq-2 simp del*: *sin-cos-minus*)
done

lemma *sin-diff*: *sin* (*x* − *y*) = *sin x* ∗ *cos y* − *cos x* ∗ *sin y*
apply (*simp add*: *diff-minus*)
apply (*simp* (*no-asm*) *add*: *sin-add*)
done

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*
apply (*simp add*: *diff-minus*)
apply (*simp* (*no-asm*) *add*: *cos-add*)
done

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 (*auto simp add*: *numeral-2-eq-2*)
done

26.10 The Constant Pi

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
(∑ *n*. (*if even n then 0*
else (− *1*) ^ ((*n* − *Suc 0*) *div 2*) / *real* (*fact n*)) ∗
x ^ *n*)
by (*rule sin-converges* [*THEN sums-summable*, *THEN sums-group*], *simp*)
thus *?thesis* **by** (*simp add*: *mult-ac sin-def*)
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 (λ*n*. ∑ *k* = *n* ∗ *2*..<*n* ∗ *2* + *2*.
(*if even k then* (− *1*) ^ (*k div 2*) / *real* (*fact k*) *else 0*) ∗
x ^ *k*)
sums
(∑ *n*. (*if even n then* (− *1*) ^ (*n div 2*) / *real* (*fact n*) *else 0*) ∗
x ^ *n*)
by (*rule cos-converges* [*THEN sums-summable*, *THEN sums-group*], *simp*)
thus *?thesis* **by** (*simp add*: *mult-ac cos-def*)
qed

declare *zero-less-power* [*simp*]

lemma *fact-lemma*: *real* (*n*::*nat*) ∗ *4* = *real* (*4* ∗ *n*)
by *simp*

lemma *cos-two-less-zero*: *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 real-mult-less-mono*, *force*)
apply (*rule-tac* [*3*] *real-of-nat-fact-gt-zero*)
prefer *2* **apply** *force*
apply (*rule real-of-nat-less-iff* [*THEN iffD2*])
apply (*rule fact-less-mono*, *auto*)
done
declare *cos-two-less-zero* [*simp*]
declare *cos-two-less-zero* [*THEN real-not-refl2*, *simp*]
declare *cos-two-less-zero* [*THEN order-less-imp-le*, *simp*]

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* (∀ *x*. *cos differentiable x*) & (∀ *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*: $pi/2 = (@x.\ 0 \leq x\ \&\ x \leq 2\ \&\ cos\ x = 0)$
by (*simp add*: *pi-def*)

lemma *cos-pi-half* [*simp*]: *cos* (*pi* / *2*) = *0*
apply (*rule cos-is-zero* [*THEN ex1E*])
apply (*auto intro*!: *someI2 simp add*: *pi-half*)
done

lemma *pi-half-gt-zero*: *0* < *pi* / *2*
apply (*rule cos-is-zero* [*THEN ex1E*])
apply (*auto simp add*: *pi-half*)
apply (*rule someI2*, *blast*, *safe*)
apply (*drule-tac y* = *xa* **in** *real-le-imp-less-or-eq*)
apply (*safe*, *simp*)
done
declare *pi-half-gt-zero* [*simp*]
declare *pi-half-gt-zero* [*THEN real-not-refl2*, *THEN not-sym*, *simp*]
declare *pi-half-gt-zero* [*THEN order-less-imp-le*, *simp*]

lemma *pi-half-less-two*: *pi* / *2* < *2*
apply (*rule cos-is-zero* [*THEN ex1E*])
apply (*auto simp add*: *pi-half*)
apply (*rule someI2*, *blast*, *safe*)
apply (*drule-tac x* = *xa* **in** *order-le-imp-less-or-eq*)
apply (*safe*, *simp*)
done
declare *pi-half-less-two* [*simp*]
declare *pi-half-less-two* [*THEN real-not-refl2*, *simp*]
declare *pi-half-less-two* [*THEN order-less-imp-le*, *simp*]

lemma *pi-gt-zero* [*simp*]: *0* < *pi*
apply (*insert pi-half-gt-zero*)
apply (*simp add*:)
done

lemma *pi-neq-zero* [*simp*]: *pi* ≠ *0*
by (*rule pi-gt-zero* [*THEN real-not-refl2*, *THEN not-sym*])

lemma *pi-not-less-zero* [*simp*]: ~ (*pi* < *0*)
apply (*insert pi-gt-zero*)
apply (*blast elim*: *order-less-asym*)
done

lemma *pi-ge-zero* [*simp*]: *0* ≤ *pi*
by (*auto intro*: *order-less-imp-le*)

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 (*auto simp add*: *numeral-2-eq-2*)
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*)

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 *sin-cos-eq* [*symmetric*, *simp*] *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* ^ *n*
apply (*induct n*)
apply (*auto simp add*: *real-of-nat-Suc left-distrib*)
done

lemma *cos-npi2* [*simp*]: *cos* (*pi* ∗ *real n*) = −*1* ^ *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 (*force intro*: *DERIV-isCont DERIV-cos*)
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 swap*)
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*: *real-add-assoc*)
apply (*rotate-tac 3*)
apply (*drule-tac* $x = xa + pi/2$ **in** *spec*, *safe*, *simp-all*)
done

lemma *reals-Archimedean4*:
[| $0 < y$; $0 \leq x$ |] ==> $\exists n.\ real\ n * y \leq x\ \&\ x < real\ (Suc\ n) * y$

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* $\sim x < real\ nat * y$)
prefer *2* **apply** (*rule not-less-Least*, *simp*, *force*)
done

lemma *cos-zero-lemma*:
$[|\ 0 \leq x;\ cos\ x = 0\ |] ==>$
$\exists n{::}nat.\ \sim even\ n\ \&\ x = real\ n * (pi/2)$
apply (*drule pi-gt-zero* [*THEN reals-Archimedean4*], *safe*)
apply (*subgoal-tac* $0 \leq x - real\ n * pi\ \&$
$(x - real\ n * pi) \leq pi\ \&\ (cos\ (x - real\ n * pi) = 0)$)
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*! $x.\ 0 \leq x\ \&\ x \leq pi\ \&\ cos\ x = 0$)
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*:
$[|\ 0 \leq x;\ sin\ x = 0\ |] ==>$
$\exists n{::}nat.\ even\ n\ \&\ x = real\ n * (pi/2)$
apply (*subgoal-tac* $\exists n{::}nat.\ \sim even\ n\ \&\ x + pi/2 = real\ n * (pi/2)$)
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*:
$(cos\ x = 0) =$
$((\exists n{::}nat.\ \sim even\ n\ \&\ (x = real\ n * (pi/2)))\ |$
$(\exists n{::}nat.\ \sim even\ n\ \&\ (x = -(real\ n * (pi/2)))))$
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

26.11 Tangent

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 *LIM-cos-div-sin* [*simp*]: (%*x*. *cos*(*x*)/*sin*(*x*)) −− *pi/2* −−> *0*
apply (*subgoal-tac* (λ*x*. *cos x * inverse* (*sin x*)) −− *pi * inverse 2* −−> *0*1*)
apply (*simp add*: *divide-inverse* [*symmetric*])
apply (*rule LIM-mult2*)
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*: $0 < y \Longrightarrow \exists x.\ 0 < x \ \&\ x < pi/2 \ \&\ y < tan\ x$
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*: $0 \leq y \Longrightarrow \exists x.\ 0 \leq x \ \&\ x < pi/2 \ \&\ tan\ x = y$
apply (*frule real-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 real-not-refl2*, *THEN not-sym*])
done

lemma *arcsin-pi*:
  [| $-1 \leq y$; $y \leq 1$ |]
  ==> $-(pi/2) \leq arcsin\ y$ & $arcsin\ y \leq pi$ & $sin(arcsin\ y) = y$
apply (*drule sin-total*, *assumption*)
apply (*erule ex1E*)
apply (*simp add*: *arcsin-def*)
apply (*rule someI2*, *blast*)
apply (*force intro*: *order-trans*)
done

lemma *arcsin*:
  [| $-1 \leq y$; $y \leq 1$ |]
  ==> $-(pi/2) \leq arcsin\ y$ &
    $arcsin\ y \leq pi/2$ & $sin(arcsin\ y) = y$
apply (*unfold arcsin-def*)
apply (*drule sin-total*, *assumption*)
apply (*fast intro*: *someI2*)
done

lemma *sin-arcsin* [*simp*]: [| $-1 \leq y$; $y \leq 1$ |] ==> $sin(arcsin\ y) = y$
by (*blast dest*: *arcsin*)

lemma *arcsin-bounded*:
  [| $-1 \leq y$; $y \leq 1$ |] ==> $-(pi/2) \leq arcsin\ y$ & $arcsin\ y \leq pi/2$
by (*blast dest*: *arcsin*)

lemma *arcsin-lbound*: [| $-1 \leq y$; $y \leq 1$ |] ==> $-(pi/2) \leq arcsin\ y$
by (*blast dest*: *arcsin*)

lemma *arcsin-ubound*: [| $-1 \leq y$; $y \leq 1$ |] ==> $arcsin\ y \leq pi/2$
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*: [|$-(pi/2) \leq x$; $x \leq pi/2$ |] ==> $arcsin(sin\ x) = x$
apply (*unfold arcsin-def*)

apply (*rule some1-equality*)
apply (*rule sin-total*, *auto*)
done

lemma *arcos*:
[| $-1 \leq y$; $y \leq 1$ |]
==> $0 \leq$ *arcos y* & *arcos y* $\leq$ *pi* & *cos*(*arcos y*) = *y*
apply (*simp add*: *arcos-def*)
apply (*drule cos-total*, *assumption*)
apply (*fast intro*: *someI2*)
done

lemma *cos-arcos* [*simp*]: [| $-1 \leq y$; $y \leq 1$ |] ==> *cos*(*arcos y*) = *y*
by (*blast dest*: *arcos*)

lemma *arcos-bounded*: [| $-1 \leq y$; $y \leq 1$ |] ==> $0 \leq$ *arcos y* & *arcos y* $\leq$ *pi*
by (*blast dest*: *arcos*)

lemma *arcos-lbound*: [| $-1 \leq y$; $y \leq 1$ |] ==> $0 \leq$ *arcos y*
by (*blast dest*: *arcos*)

lemma *arcos-ubound*: [| $-1 \leq y$; $y \leq 1$ |] ==> *arcos y* $\leq$ *pi*
by (*blast dest*: *arcos*)

lemma *arcos-lt-bounded*:
[| $-1 < y$; $y < 1$ |]
==> $0 <$ *arcos y* & *arcos y* $<$ *pi*
apply (*frule order-less-imp-le*)
apply (*frule-tac y = y* **in** *order-less-imp-le*)
apply (*frule arcos-bounded*, *auto*)
apply (*drule-tac y = arcos 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 *arcos-cos*: [|$0 \leq x$; $x \leq$ *pi* |] ==> *arcos*(*cos x*) = *x*
apply (*simp add*: *arcos-def*)
apply (*auto intro*!: *some1-equality cos-total*)
done

lemma *arcos-cos2*: [|$x \leq 0$; $-pi \leq x$ |] ==> *arcos*(*cos x*) = $-x$
apply (*simp add*: *arcos-def*)
apply (*auto intro*!: *some1-equality cos-total*)
done

lemma *arctan* [*simp*]:
$-(pi/2) <$ *arctan y* & *arctan y* $< pi/2$ & *tan* (*arctan y*) = *y*
apply (*cut-tac y = y* **in** *tan-total*)
apply (*simp add*: *arctan-def*)

apply (*fast intro*: *someI2*)
done

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 some1-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*: *realpow-not-zero*
simp add: *power-mult-distrib left-distrib realpow-divide tan-def*
mult-assoc power-inverse [*symmetric*]
simp del: *realpow-Suc*)
done

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 3/2 = (1+1 / 2::real)*)
apply (*simp only*: *left-distrib*)
apply (*auto simp add*: *cos-add mult-ac*)
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 3/2 = (1+1 / 2::real)*)
apply (*simp only*: *left-distrib*)
apply (*auto simp add*: *sin-add mult-ac*)
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 *isCont-cos* [*simp*]: *isCont cos x*
by (*rule DERIV-cos* [*THEN DERIV-isCont*])

lemma *isCont-sin* [*simp*]: *isCont sin x*
by (*rule DERIV-sin* [*THEN DERIV-isCont*])

lemma *isCont-exp* [*simp*]: *isCont exp x*
by (*rule DERIV-exp* [*THEN DERIV-isCont*])

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* = *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*)

lemma *real-root-less-mono*:
[| *0* ≤ *x*; *x* < *y* |] ==> *root*(*Suc n*) *x* < *root*(*Suc n*) *y*
apply (*frule order-le-less-trans*, *assumption*)
apply (*frule-tac n1* = *n* **in** *real-root-pow-pos2* [*THEN ssubst*])
apply (*rotate-tac 1*, *assumption*)
apply (*frule-tac n1* = *n* **in** *real-root-pow-pos* [*THEN ssubst*])
apply (*rotate-tac 3*, *assumption*)
apply (*drule-tac y* = *root* (*Suc n*) *y* ^ *Suc n* **in** *order-less-imp-le*)
apply (*frule-tac n* = *n* **in** *real-root-pos-pos-le*)
apply (*frule-tac n* = *n* **in** *real-root-pos-pos*)
apply (*drule-tac x* = *root* (*Suc n*) *x* **and** *y* = *root* (*Suc n*) *y* **in** *realpow-increasing*)
apply (*assumption*, *assumption*)
apply (*drule-tac x* = *root* (*Suc n*) *x* **in** *order-le-imp-less-or-eq*)
apply *auto*
apply (*drule-tac f* = %*x*. *x* ^ (*Suc n*) **in** *arg-cong*)
apply (*auto simp add*: *real-root-pow-pos2 simp del*: *realpow-Suc*)

done

lemma *real-root-le-mono*:
[| $0 \leq x$; $x \leq y$ |] ==> $root(Suc\ n)\ x \leq root(Suc\ n)\ y$
apply (*drule-tac* $y = y$ **in** *order-le-imp-less-or-eq*)
apply (*auto dest*: *real-root-less-mono intro*: *order-less-imp-le*)
done

lemma *real-root-less-iff* [*simp*]:
[| $0 \leq x$; $0 \leq y$ |] ==> $(root(Suc\ n)\ x < root(Suc\ n)\ y) = (x < y)$
apply (*auto intro*: *real-root-less-mono*)
apply (*rule ccontr*, *drule linorder-not-less* [*THEN iffD1*])
apply (*drule-tac* $x = y$ **and** $n = n$ **in** *real-root-le-mono*, *auto*)
done

lemma *real-root-le-iff* [*simp*]:
[| $0 \leq x$; $0 \leq y$ |] ==> $(root(Suc\ n)\ x \leq root(Suc\ n)\ y) = (x \leq y)$
apply (*auto intro*: *real-root-le-mono*)
apply (*simp* (*no-asm*) *add*: *linorder-not-less* [*symmetric*])
apply *auto*
apply (*drule-tac* $x = y$ **and** $n = n$ **in** *real-root-less-mono*, *auto*)
done

lemma *real-root-eq-iff* [*simp*]:
[| $0 \leq x$; $0 \leq y$ |] ==> $(root(Suc\ n)\ x = root(Suc\ n)\ y) = (x = y)$
apply (*auto intro*!: *order-antisym*)
apply (*rule-tac n1* $= n$ **in** *real-root-le-iff* [*THEN iffD1*])
apply (*rule-tac* [*4*] *n1* $= n$ **in** *real-root-le-iff* [*THEN iffD1*], *auto*)
done

lemma *real-root-pos-unique*:
[| $0 \leq x$; $0 \leq y$; $y\ \hat{}\ (Suc\ n) = x$ |] ==> $root\ (Suc\ n)\ x = y$
by (*auto dest*: *real-root-pos2 simp del*: *realpow-Suc*)

lemma *real-root-mult*:
[| $0 \leq x$; $0 \leq y$ |]
==> $root(Suc\ n)\ (x * y) = root(Suc\ n)\ x * root(Suc\ n)\ y$
apply (*rule real-root-pos-unique*)
apply (*auto intro*!: *real-root-pos-pos-le*
simp add: *power-mult-distrib zero-le-mult-iff real-root-pow-pos2*
simp del: *realpow-Suc*)
done

lemma *real-root-inverse*:
$0 \leq x$ ==> $(root(Suc\ n)\ (inverse\ x) = inverse(root(Suc\ n)\ x))$
apply (*rule real-root-pos-unique*)
apply (*auto intro*: *real-root-pos-pos-le*
simp add: *power-inverse* [*symmetric*] *real-root-pow-pos2*
simp del: *realpow-Suc*)

done

lemma *real-root-divide*:
[| $0 \leq x$; $0 \leq y$ |]
==> $(root(Suc\ n)\ (x\ /\ y) = root(Suc\ n)\ x\ /\ root(Suc\ n)\ y)$
apply (*simp add*: *divide-inverse*)
apply (*auto simp add*: *real-root-mult real-root-inverse*)
done

lemma *real-sqrt-less-mono*: [| $0 \leq x$; $x < y$ |] ==> $sqrt(x) < sqrt(y)$
by (*simp add*: *sqrt-def*)

lemma *real-sqrt-le-mono*: [| $0 \leq x$; $x \leq y$ |] ==> $sqrt(x) \leq sqrt(y)$
by (*simp add*: *sqrt-def*)

lemma *real-sqrt-less-iff* [*simp*]:
[| $0 \leq x$; $0 \leq y$ |] ==> $(sqrt(x) < sqrt(y)) = (x < y)$
by (*simp add*: *sqrt-def*)

lemma *real-sqrt-le-iff* [*simp*]:
[| $0 \leq x$; $0 \leq y$ |] ==> $(sqrt(x) \leq sqrt(y)) = (x \leq y)$
by (*simp add*: *sqrt-def*)

lemma *real-sqrt-eq-iff* [*simp*]:
[| $0 \leq x$; $0 \leq y$ |] ==> $(sqrt(x) = sqrt(y)) = (x = y)$
by (*simp add*: *sqrt-def*)

lemma *real-sqrt-sos-less-one-iff* [*simp*]: $(sqrt(x^2 + y^2) < 1) = (x^2 + y^2 < 1)$
apply (*rule real-sqrt-one* [*THEN subst*], *safe*)
apply (*rule-tac* [*2*] *real-sqrt-less-mono*)
apply (*drule real-sqrt-less-iff* [*THEN* [*2*] *rev-iffD1*], *auto*)
done

lemma *real-sqrt-sos-eq-one-iff* [*simp*]: $(sqrt(x^2 + y^2) = 1) = (x^2 + y^2 = 1)$
apply (*rule real-sqrt-one* [*THEN subst*], *safe*)
apply (*drule real-sqrt-eq-iff* [*THEN* [*2*] *rev-iffD1*], *auto*)
done

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

26.12 Theorems About Sqrt, Transcendental Functions for Complex

lemma *le-real-sqrt-sumsq* [*simp*]: $x \leq sqrt\ (x * x + y * y)$
proof (*rule order-trans*)

show $x \leq sqrt(x*x)$ **by** (*simp add*: *abs-if*)
show $sqrt\ (x * x) \leq sqrt\ (x * x + y * y)$
by (*rule real-sqrt-le-mono*, *auto*)
qed

lemma *minus-le-real-sqrt-sumsq* [*simp*]: $-x \leq sqrt\ (x * x + y * y)$
proof (*rule order-trans*)
show $-x \leq sqrt(x*x)$ **by** (*simp add*: *abs-if*)
show $sqrt\ (x * x) \leq sqrt\ (x * x + y * y)$
by (*rule real-sqrt-le-mono*, *auto*)
qed

lemma *lemma-real-divide-sqrt-ge-minus-one*:
$0 < x ==> -1 \leq x/(sqrt\ (x * x + y * y))$
by (*simp add*: *divide-const-simps linorder-not-le* [*symmetric*])

lemma *real-sqrt-sum-squares-gt-zero1*: $x < 0 ==> 0 < sqrt\ (x * x + y * y)$
apply (*rule real-sqrt-gt-zero*)
apply (*subgoal-tac* $0 < x*x\ \&\ 0 \leq y*y$, *arith*)
apply (*auto simp add*: *zero-less-mult-iff*)
done

lemma *real-sqrt-sum-squares-gt-zero2*: $0 < x ==> 0 < sqrt\ (x * x + y * y)$
apply (*rule real-sqrt-gt-zero*)
apply (*subgoal-tac* $0 < x*x\ \&\ 0 \leq y*y$, *arith*)
apply (*auto simp add*: *zero-less-mult-iff*)
done

lemma *real-sqrt-sum-squares-gt-zero3*: $x \neq 0 ==> 0 < sqrt(x^2 + y^2)$
apply (*cut-tac* $x = x$ **and** $y = 0$ **in** *linorder-less-linear*)
apply (*auto intro*: *real-sqrt-sum-squares-gt-zero2 real-sqrt-sum-squares-gt-zero1 simp add*: *numeral-2-eq-2*)
done

lemma *real-sqrt-sum-squares-gt-zero3a*: $y \neq 0 ==> 0 < sqrt(x^2 + y^2)$
apply (*drule-tac* $y = x$ **in** *real-sqrt-sum-squares-gt-zero3*)
apply (*auto simp add*: *real-add-commute*)
done

lemma *real-sqrt-sum-squares-eq-cancel*: $sqrt(x^2 + y^2) = x ==> y = 0$
by (*drule-tac* $f = \%x.\ x^2$ **in** *arg-cong*, *auto*)

lemma *real-sqrt-sum-squares-eq-cancel2*: $sqrt(x^2 + y^2) = y ==> x = 0$
apply (*rule-tac* $x = y$ **in** *real-sqrt-sum-squares-eq-cancel*)
apply (*simp add*: *real-add-commute*)
done

lemma *lemma-real-divide-sqrt-le-one*: $x < 0 ==> x/(sqrt\ (x * x + y * y)) \leq 1$
by (*insert lemma-real-divide-sqrt-ge-minus-one* [*of* $-x\ y$], *simp*)

lemma *lemma-real-divide-sqrt-ge-minus-one2*:
$x < 0 \Longrightarrow -1 \leq x/(sqrt\ (x * x + y * y))$
apply (*simp add*: *divide-const-simps*)
apply (*insert minus-le-real-sqrt-sumsq* [*of x y*], *arith*)
done

lemma *lemma-real-divide-sqrt-le-one2*: $0 < x \Longrightarrow x/(sqrt\ (x * x + y * y)) \leq 1$
by (*cut-tac* $x = -x$ **and** $y = y$ **in** *lemma-real-divide-sqrt-ge-minus-one2*, *auto*)

lemma *minus-sqrt-le*: $-\ sqrt\ (x * x + y * y) \leq x$
by (*insert minus-le-real-sqrt-sumsq* [*of x y*], *arith*)

lemma *minus-sqrt-le2*: $-\ sqrt\ (x * x + y * y) \leq y$
by (*subst add-commute*, *simp add*: *minus-sqrt-le*)

lemma *not-neg-sqrt-sumsq*: $\sim\ sqrt\ (x * x + y * y) < 0$
by (*simp add*: *linorder-not-less*)

lemma *cos-x-y-ge-minus-one*: $-1 \leq x\ /\ sqrt\ (x * x + y * y)$
by (*simp add*: *minus-sqrt-le not-neg-sqrt-sumsq divide-const-simps*)

lemma *cos-x-y-ge-minus-one1a* [*simp*]: $-1 \leq y\ /\ sqrt\ (x * x + y * y)$
by (*subst add-commute*, *simp add*: *cos-x-y-ge-minus-one*)

lemma *cos-x-y-le-one* [*simp*]: $x\ /\ sqrt\ (x * x + y * y) \leq 1$
apply (*cut-tac* $x = x$ **and** $y = 0$ **in** *linorder-less-linear*, *safe*)
apply (*rule lemma-real-divide-sqrt-le-one*)
apply (*rule-tac* [*3*] *lemma-real-divide-sqrt-le-one2*, *auto*)
done

lemma *cos-x-y-le-one2* [*simp*]: $y\ /\ sqrt\ (x * x + y * y) \leq 1$
apply (*cut-tac* $x = y$ **and** $y = x$ **in** *cos-x-y-le-one*)
apply (*simp add*: *real-add-commute*)
done

declare *cos-arcos* [*OF cos-x-y-ge-minus-one cos-x-y-le-one*, *simp*]
declare *arcos-bounded* [*OF cos-x-y-ge-minus-one cos-x-y-le-one*, *simp*]

declare *cos-arcos* [*OF cos-x-y-ge-minus-one1a cos-x-y-le-one2*, *simp*]
declare *arcos-bounded* [*OF cos-x-y-ge-minus-one1a cos-x-y-le-one2*, *simp*]

lemma *cos-abs-x-y-ge-minus-one* [*simp*]:
$-1 \leq |x|\ /\ sqrt\ (x * x + y * y)$
by (*auto simp add*: *divide-const-simps abs-if linorder-not-le* [*symmetric*])

lemma *cos-abs-x-y-le-one* [*simp*]: $|x|\ /\ sqrt\ (x * x + y * y) \leq 1$
apply (*insert minus-le-real-sqrt-sumsq* [*of x y*] *le-real-sqrt-sumsq* [*of x y*])
apply (*auto simp add*: *divide-const-simps abs-if linorder-neq-iff*)

done

declare *cos-arcos* [*OF cos-abs-x-y-ge-minus-one cos-abs-x-y-le-one*, *simp*]
declare *arcos-bounded* [*OF cos-abs-x-y-ge-minus-one cos-abs-x-y-le-one*, *simp*]

lemma *minus-pi-less-zero*: $-pi < 0$
by *simp*

declare *minus-pi-less-zero* [*simp*]
declare *minus-pi-less-zero* [*THEN order-less-imp-le*, *simp*]

lemma *arcos-ge-minus-pi*: [| $-1 \leq y$; $y \leq 1$ |] ==> $-pi \leq arcos\ y$
apply (*rule real-le-trans*)
apply (*rule-tac* [*2*] *arcos-lbound*, *auto*)
done

declare *arcos-ge-minus-pi* [*OF cos-x-y-ge-minus-one cos-x-y-le-one*, *simp*]

lemma *lemma-divide-rearrange*:
    [| $x + (y::real) \neq 0$; $1 - z = x/(x + y)$ |] ==> $z = y/(x + y)$
apply (*rule-tac c1* $= x + y$ **in** *real-mult-right-cancel* [*THEN iffD1*])
apply (*frule-tac* [*2*] *c1* $= x + y$ **in** *real-mult-right-cancel* [*THEN iffD2*])
prefer *2* **apply** *assumption*
apply (*rotate-tac* [*2*] *2*)
apply (*drule-tac* [*2*] *mult-assoc* [*THEN subst*])
apply (*rotate-tac* [*2*] *2*)
apply (*frule-tac* [*2*] *left-inverse* [*THEN subst*])
prefer *2* **apply** *assumption*
apply (*erule-tac* [*2*] $V = (1 - z) * (x + y) = x\ /\ (x + y) * (x + y)$ **in** *thin-rl*)
apply (*erule-tac* [*2*] $V = 1 - z = x\ /\ (x + y)$ **in** *thin-rl*)
apply (*auto simp add*: *mult-assoc*)
apply (*auto simp add*: *right-distrib left-diff-distrib*)
done

lemma *lemma-cos-sin-eq*:
    [| $0 < x * x + y * y$;
       $1 - (sin\ xa)^2 = (x\ /\ sqrt\ (x * x + y * y))\ \hat{}\ 2$ |]
    ==> $(sin\ xa)^2 = (y\ /\ sqrt\ (x * x + y * y))\ \hat{}\ 2$
by (*auto intro*: *lemma-divide-rearrange*
        simp add: *realpow-divide power2-eq-square* [*symmetric*])

lemma *lemma-sin-cos-eq*:
    [| $0 < x * x + y * y$;
       $1 - (cos\ xa)^2 = (y\ /\ sqrt\ (x * x + y * y))\ \hat{}\ 2$ |]
    ==> $(cos\ xa)^2 = (x\ /\ sqrt\ (x * x + y * y))\ \hat{}\ 2$
apply (*auto simp add*: *realpow-divide power2-eq-square* [*symmetric*])
apply (*subst add-commute*)

apply (*rule lemma-divide-rearrange*, *simp add*: *real-add-eq-0-iff*)
apply (*simp add*: *add-commute*)
done

lemma *sin-x-y-disj*:
[| $x \neq 0$;
$cos\ xa = x\ /\ sqrt\ (x * x + y * y)$
|] ==> $sin\ xa = y\ /\ sqrt\ (x * x + y * y)$ |
$sin\ xa = -\ y\ /\ sqrt\ (x * x + y * y)$
apply (*drule-tac* $f = \%x.\ x^2$ **in** *arg-cong*)
apply (*frule-tac* $y = y$ **in** *real-sum-square-gt-zero*)
apply (*simp add*: *cos-squared-eq*)
apply (*subgoal-tac* $(sin\ xa)^2 = (y\ /\ sqrt\ (x * x + y * y))\ \hat{}\ 2$)
apply (*rule-tac* [2] *lemma-cos-sin-eq*)
apply (*auto simp add*: *realpow-two-disj numeral-2-eq-2 simp del*: *realpow-Suc*)
done

lemma *lemma-cos-not-eq-zero*: $x \neq 0 ==> x\ /\ sqrt\ (x * x + y * y) \neq 0$
apply (*simp add*: *divide-inverse*)
apply (*frule-tac* $y3 = y$ **in** *real-sqrt-sum-squares-gt-zero3* [*THEN real-not-refl2*, *THEN not-sym*, *THEN nonzero-imp-inverse-nonzero*])
apply (*auto simp add*: *power2-eq-square*)
done

lemma *cos-x-y-disj*:
[| $x \neq 0$;
$sin\ xa = y\ /\ sqrt\ (x * x + y * y)$
|] ==> $cos\ xa = x\ /\ sqrt\ (x * x + y * y)$ |
$cos\ xa = -\ x\ /\ sqrt\ (x * x + y * y)$
apply (*drule-tac* $f = \%x.\ x^2$ **in** *arg-cong*)
apply (*frule-tac* $y = y$ **in** *real-sum-square-gt-zero*)
apply (*simp add*: *sin-squared-eq del*: *realpow-Suc*)
apply (*subgoal-tac* $(cos\ xa)^2 = (x\ /\ sqrt\ (x * x + y * y))\ \hat{}\ 2$)
apply (*rule-tac* [2] *lemma-sin-cos-eq*)
apply (*auto simp add*: *realpow-two-disj numeral-2-eq-2 simp del*: *realpow-Suc*)
done

lemma *real-sqrt-divide-less-zero*: $0 < y ==> -\ y\ /\ sqrt\ (x * x + y * y) < 0$
apply (*case-tac* $x = 0$, *auto*)
apply (*drule-tac* $y = y$ **in** *real-sqrt-sum-squares-gt-zero3*)
apply (*auto simp add*: *zero-less-mult-iff divide-inverse power2-eq-square*)
done

lemma *polar-ex1*:
[| $x \neq 0$; $0 < y$ |] ==> $\exists r\ a.\ x = r * cos\ a\ \&\ y = r * sin\ a$
apply (*rule-tac* $x = sqrt\ (x^2 + y^2)$ **in** *exI*)
apply (*rule-tac* $x = arcos\ (x\ /\ sqrt\ (x * x + y * y))$ **in** *exI*)
apply *auto*
apply (*drule-tac* $y2 = y$ **in** *real-sqrt-sum-squares-gt-zero3* [*THEN real-not-refl2*,

THEN not-sym])
apply (*auto simp add*: *power2-eq-square*)
apply (*simp add*: *arcos-def*)
apply (*cut-tac x1* = *x* **and** *y1* = *y*
    in *cos-total* [*OF cos-x-y-ge-minus-one cos-x-y-le-one*])
apply (*rule someI2-ex*, *blast*)
apply (*erule-tac V* = *EX*! *xa*. *0* ≤ *xa* & *xa* ≤ *pi* & *cos xa* = *x* / *sqrt* (*x* ∗ *x* + *y* ∗ *y*) **in** *thin-rl*)
apply (*frule sin-x-y-disj*, *blast*)
apply (*drule-tac y2* = *y* **in** *real-sqrt-sum-squares-gt-zero3* [*THEN real-not-refl2*, *THEN not-sym*])
apply (*auto simp add*: *power2-eq-square*)
apply (*drule sin-ge-zero*, *assumption*)
apply (*drule-tac x* = *x* **in** *real-sqrt-divide-less-zero*, *auto*)
done

lemma *real-sum-squares-cancel2a*: *x* ∗ *x* = −(*y* ∗ *y*) ==> *y* = (*0*::*real*)
by (*auto intro*: *real-sum-squares-cancel iff*: *real-add-eq-0-iff*)

lemma *polar-ex2*:
    [| *x* ≠ *0*; *y* < *0* |] ==> ∃ *r a*. *x* = *r* ∗ *cos a* & *y* = *r* ∗ *sin a*
apply (*cut-tac x* = *0* **and** *y* = *x* **in** *linorder-less-linear*, *auto*)
apply (*rule-tac x* = *sqrt* (x^2 + y^2) **in** *exI*)
apply (*rule-tac x* = *arcsin* (*y* / *sqrt* (*x* ∗ *x* + *y* ∗ *y*)) **in** *exI*)
apply (*auto dest*: *real-sum-squares-cancel2a*
        simp add: *power2-eq-square real-0-le-add-iff real-add-eq-0-iff*)
apply (*unfold arcsin-def*)
apply (*cut-tac x1* = *x* **and** *y1* = *y*
    in *sin-total* [*OF cos-x-y-ge-minus-one1a cos-x-y-le-one2*])
apply (*rule someI2-ex*, *blast*)
apply (*erule-tac V* = *EX*! *v*. *?P v* **in** *thin-rl*)
apply (*cut-tac x*=*x* **and** *y*=*y* **in** *cos-x-y-disj*, *simp*, *blast*)
apply (*auto simp add*: *real-0-le-add-iff real-add-eq-0-iff*)
apply (*drule cos-ge-zero*, *force*)
apply (*drule-tac x* = *y* **in** *real-sqrt-divide-less-zero*)
apply (*auto simp add*: *add-commute*)
apply (*insert polar-ex1* [*of x* −*y*], *simp*, *clarify*)
apply (*rule-tac x* = *r* **in** *exI*)
apply (*rule-tac x* = −*a* **in** *exI*, *simp*)
done

lemma *polar-Ex*: ∃ *r a*. *x* = *r* ∗ *cos a* & *y* = *r* ∗ *sin a*
apply (*case-tac x* = *0*, *auto*)
apply (*rule-tac x* = *y* **in** *exI*)
apply (*rule-tac x* = *pi*/*2* **in** *exI*, *auto*)
apply (*cut-tac x* = *0* **and** *y* = *y* **in** *linorder-less-linear*, *auto*)
apply (*rule-tac* [*2*] *x* = *x* **in** *exI*)
apply (*rule-tac* [*2*] *x* = *0* **in** *exI*, *auto*)
apply (*blast intro*: *polar-ex1 polar-ex2*)+

done

lemma *real-sqrt-ge-abs1* [*simp*]: $|x| \leq sqrt\ (x^2 + y^2)$
apply (*rule-tac n = 1* **in** *realpow-increasing*)
apply (*auto simp add*: *numeral-2-eq-2* [*symmetric*] *power2-abs*)
done

lemma *real-sqrt-ge-abs2* [*simp*]: $|y| \leq sqrt\ (x^2 + y^2)$
apply (*rule real-add-commute* [*THEN subst*])
apply (*rule real-sqrt-ge-abs1*)
done
declare *real-sqrt-ge-abs1* [*simp*] *real-sqrt-ge-abs2* [*simp*]

lemma *real-sqrt-two-gt-zero* [*simp*]: *0 < sqrt 2*
by (*auto intro*: *real-sqrt-gt-zero*)

lemma *real-sqrt-two-ge-zero* [*simp*]: *0 ≤ sqrt 2*
by (*auto intro*: *real-sqrt-ge-zero*)

lemma *real-sqrt-two-gt-one* [*simp*]: *1 < sqrt 2*
apply (*rule order-less-le-trans* [*of - 7/5*], *simp*)
apply (*rule-tac n = 1* **in** *realpow-increasing*)
prefer *3* **apply** (*simp add*: *numeral-2-eq-2* [*symmetric*] *del*: *realpow-Suc*)
apply (*simp-all add*: *numeral-2-eq-2*)
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*)

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 ≤ x*; *0 ≤ 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-tac n = 1* **in** *realpow-increasing*)
apply (*auto simp add*: *real-0-le-divide-iff realpow-divide numeral-2-eq-2* [*symmetric*]

simp del: *realpow-Suc*)
apply (*rule-tac t = u²* **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

declare *real-sqrt-sum-squares-ge-zero* [*THEN abs-of-nonneg*, *simp*]

26.13 A Few Theorems Involving Ln, Derivatives, etc.

lemma *lemma-DERIV-ln*:
    *DERIV ln z :> l ==> DERIV (%x. exp (ln x)) z :> exp (ln z) * l*
by (*erule DERIV-fun-exp*)

lemma *STAR-exp-ln*: *0 < z ==> (*f* (%x. exp (ln x))) z = z*
apply (*cases z*)
apply (*auto simp add*: *starfun star-n-zero-num star-n-less star-n-eq-iff*)
done

lemma *hypreal-add-Infinitesimal-gt-zero*:
    [|*e* : *Infinitesimal*; *0 < x* |] ==> *0 < hypreal-of-real x + e*
apply (*rule-tac c1 = −e* **in** *add-less-cancel-right* [*THEN iffD1*])
apply (*auto intro*: *Infinitesimal-less-SReal*)
done

lemma *NSDERIV-exp-ln-one*: *0 < z ==> NSDERIV (%x. exp (ln x)) z :> 1*
apply (*simp add*: *nsderiv-def NSLIM-def*, *auto*)
apply (*rule ccontr*)
apply (*subgoal-tac 0 < hypreal-of-real z + h*)
apply (*drule STAR-exp-ln*)
apply (*rule-tac* [*2*] *hypreal-add-Infinitesimal-gt-zero*)
apply (*subgoal-tac h/h = 1*)
apply (*auto simp add*: *exp-ln-iff* [*symmetric*] *simp del*: *exp-ln-iff*)
done

lemma *DERIV-exp-ln-one*: *0 < z ==> DERIV (%x. exp (ln x)) z :> 1*
by (*auto intro*: *NSDERIV-exp-ln-one simp add*: *NSDERIV-DERIV-iff* [*symmetric*])

lemma *lemma-DERIV-ln2*:
    [| *0 < z*; *DERIV ln z :> l* |] ==> *exp (ln z) * l = 1*
apply (*rule DERIV-unique*)
apply (*rule lemma-DERIV-ln*)
apply (*rule-tac* [*2*] *DERIV-exp-ln-one*, *auto*)
done

lemma *lemma-DERIV-ln3*:
    [| *0 < z*; *DERIV ln z :> l* |] ==> *l = 1/(exp (ln z))*
apply (*rule-tac c1 = exp (ln z)* **in** *real-mult-left-cancel* [*THEN iffD1*])
apply (*auto intro*: *lemma-DERIV-ln2*)
done

lemma *lemma-DERIV-ln4*: [| *0 < z*; *DERIV ln z :> l* |] ==> *l = 1/z*
apply (*rule-tac t = z* **in** *exp-ln-iff* [*THEN iffD2*, *THEN subst*])
apply (*auto intro*: *lemma-DERIV-ln3*)
done

lemma *isCont-inv-fun*:
[| $0 < d$; $\forall z.\ |z - x| \leq d \dashrightarrow g(f(z)) = z$;
$\forall z.\ |z - x| \leq d \dashrightarrow isCont\ f\ z$ |]
==> *isCont g (f x)*
apply (*simp (no-asm) add*: *isCont-iff LIM-def*)
apply *safe*
apply (*drule-tac ?d1.0 = r* **in** *real-lbound-gt-zero*)
apply (*assumption*, *safe*)
apply (*subgoal-tac* $\forall z.\ |z - x| \leq e \dashrightarrow (g\ (f\ z) = z)$)
prefer *2* **apply** *force*
apply (*subgoal-tac* $\forall z.\ |z - x| \leq e \dashrightarrow isCont\ f\ z$)
prefer *2* **apply** *force*
apply (*drule-tac d = e* **in** *isCont-inj-range*)
prefer *2* **apply** (*assumption*, *assumption*, *safe*)
apply (*rule-tac x = ea* **in** *exI*, *auto*)
apply (*drule-tac x = f (x) + xa* **and** $P = \%y.\ |y - f\ x| \leq ea \longrightarrow (\exists z.\ |z - x| \leq e \wedge f\ z = y)$ **in** *spec*)
apply *auto*
apply (*drule sym*, *auto*, *arith*)
done

lemma *isCont-inv-fun-inv*:
[| $0 < d$;
$\forall z.\ |z - x| \leq d \dashrightarrow g(f(z)) = z$;
$\forall z.\ |z - x| \leq d \dashrightarrow isCont\ f\ z$ |]
==> $\exists e.\ 0 < e\ \&$
$(\forall y.\ 0 < |y - f(x)|\ \&\ |y - f(x)| < e \dashrightarrow f(g(y)) = y)$
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 -- c --> l; $0 < l$ |]
==> $\exists r.\ 0 < r\ \&\ (\forall x.\ x \neq c\ \&\ |c - x| < r \dashrightarrow 0 < f\ x)$
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-interval-iff*)
done

lemma *LIM-fun-less-zero*:
[| f -- c --> l; $l < 0$ |]

$\Longrightarrow \exists r.\ 0 < r\ \&\ (\forall x.\ x \neq c\ \&\ |c - x| < r \longrightarrow f\ x < 0)$
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-interval-iff*)
done

lemma *LIM-fun-not-zero*:
$[|\ f \dashrightarrow c \longrightarrow l;\ l \neq 0\ |]$
$\Longrightarrow \exists r.\ 0 < r\ \&\ (\forall x.\ x \neq c\ \&\ |c - x| < r \longrightarrow f\ x \neq 0)$
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

ML
⟪
val inverse-unique = thm inverse-unique;
val real-root-zero = thm real-root-zero;
val real-root-pow-pos = thm real-root-pow-pos;
val real-root-pow-pos2 = thm real-root-pow-pos2;
val real-root-pos = thm real-root-pos;
val real-root-pos2 = thm real-root-pos2;
val real-root-pos-pos = thm real-root-pos-pos;
val real-root-pos-pos-le = thm real-root-pos-pos-le;
val real-root-one = thm real-root-one;
val root-2-eq = thm root-2-eq;
val real-sqrt-zero = thm real-sqrt-zero;
val real-sqrt-one = thm real-sqrt-one;
val real-sqrt-pow2-iff = thm real-sqrt-pow2-iff;
val real-sqrt-pow2 = thm real-sqrt-pow2;
val real-sqrt-abs-abs = thm real-sqrt-abs-abs;
val real-pow-sqrt-eq-sqrt-pow = thm real-pow-sqrt-eq-sqrt-pow;
val real-pow-sqrt-eq-sqrt-abs-pow2 = thm real-pow-sqrt-eq-sqrt-abs-pow2;
val real-sqrt-pow-abs = thm real-sqrt-pow-abs;
val not-real-square-gt-zero = thm not-real-square-gt-zero;
val real-sqrt-gt-zero = thm real-sqrt-gt-zero;
val real-sqrt-ge-zero = thm real-sqrt-ge-zero;
val sqrt-eqI = thm sqrt-eqI;
val real-sqrt-mult-distrib = thm real-sqrt-mult-distrib;
val real-sqrt-mult-distrib2 = thm real-sqrt-mult-distrib2;
val real-sqrt-sum-squares-ge-zero = thm real-sqrt-sum-squares-ge-zero;
val real-sqrt-sum-squares-mult-ge-zero = thm real-sqrt-sum-squares-mult-ge-zero;
val real-sqrt-sum-squares-mult-squared-eq = thm real-sqrt-sum-squares-mult-squared-eq;
val real-sqrt-abs = thm real-sqrt-abs;
val real-sqrt-abs2 = thm real-sqrt-abs2;
val real-sqrt-pow2-gt-zero = thm real-sqrt-pow2-gt-zero;

val real-sqrt-not-eq-zero = thm real-sqrt-not-eq-zero;
val real-inv-sqrt-pow2 = thm real-inv-sqrt-pow2;
val real-sqrt-eq-zero-cancel = thm real-sqrt-eq-zero-cancel;
val real-sqrt-eq-zero-cancel-iff = thm real-sqrt-eq-zero-cancel-iff;
val real-sqrt-sum-squares-ge1 = thm real-sqrt-sum-squares-ge1;
val real-sqrt-sum-squares-ge2 = thm real-sqrt-sum-squares-ge2;
val real-sqrt-ge-one = thm real-sqrt-ge-one;
val summable-exp = thm summable-exp;
val summable-sin = thm summable-sin;
val summable-cos = thm summable-cos;
val exp-converges = thm exp-converges;
val sin-converges = thm sin-converges;
val cos-converges = thm cos-converges;
val powser-insidea = thm powser-insidea;
val powser-inside = thm powser-inside;
val diffs-minus = thm diffs-minus;
val diffs-equiv = thm diffs-equiv;
val less-add-one = thm less-add-one;
val termdiffs-aux = thm termdiffs-aux;
val termdiffs = thm termdiffs;
val exp-fdiffs = thm exp-fdiffs;
val sin-fdiffs = thm sin-fdiffs;
val sin-fdiffs2 = thm sin-fdiffs2;
val cos-fdiffs = thm cos-fdiffs;
val cos-fdiffs2 = thm cos-fdiffs2;
val DERIV-exp = thm DERIV-exp;
val DERIV-sin = thm DERIV-sin;
val DERIV-cos = thm DERIV-cos;
val exp-zero = thm exp-zero;
(∗ *val exp-ge-add-one-self = thm exp-ge-add-one-self*; ∗)
val exp-gt-one = thm exp-gt-one;
val DERIV-exp-add-const = thm DERIV-exp-add-const;
val DERIV-exp-minus = thm DERIV-exp-minus;
val DERIV-exp-exp-zero = thm DERIV-exp-exp-zero;
val exp-add-mult-minus = thm exp-add-mult-minus;
val exp-mult-minus = thm exp-mult-minus;
val exp-mult-minus2 = thm exp-mult-minus2;
val exp-minus = thm exp-minus;
val exp-add = thm exp-add;
val exp-ge-zero = thm exp-ge-zero;
val exp-not-eq-zero = thm exp-not-eq-zero;
val exp-gt-zero = thm exp-gt-zero;
val inv-exp-gt-zero = thm inv-exp-gt-zero;
val abs-exp-cancel = thm abs-exp-cancel;
val exp-real-of-nat-mult = thm exp-real-of-nat-mult;
val exp-diff = thm exp-diff;
val exp-less-mono = thm exp-less-mono;
val exp-less-cancel = thm exp-less-cancel;
val exp-less-cancel-iff = thm exp-less-cancel-iff;

val exp-le-cancel-iff = thm exp-le-cancel-iff;
val exp-inj-iff = thm exp-inj-iff;
val exp-total = thm exp-total;
val ln-exp = thm ln-exp;
val exp-ln-iff = thm exp-ln-iff;
val ln-mult = thm ln-mult;
val ln-inj-iff = thm ln-inj-iff;
val ln-one = thm ln-one;
val ln-inverse = thm ln-inverse;
val ln-div = thm ln-div;
val ln-less-cancel-iff = thm ln-less-cancel-iff;
val ln-le-cancel-iff = thm ln-le-cancel-iff;
val ln-realpow = thm ln-realpow;
val ln-add-one-self-le-self = thm ln-add-one-self-le-self;
val ln-less-self = thm ln-less-self;
val ln-ge-zero = thm ln-ge-zero;
val ln-gt-zero = thm ln-gt-zero;
val ln-less-zero = thm ln-less-zero;
val exp-ln-eq = thm exp-ln-eq;
val sin-zero = thm sin-zero;
val cos-zero = thm cos-zero;
val DERIV-sin-sin-mult = thm DERIV-sin-sin-mult;
val DERIV-sin-sin-mult2 = thm DERIV-sin-sin-mult2;
val DERIV-sin-realpow2 = thm DERIV-sin-realpow2;
val DERIV-sin-realpow2a = thm DERIV-sin-realpow2a;
val DERIV-cos-cos-mult = thm DERIV-cos-cos-mult;
val DERIV-cos-cos-mult2 = thm DERIV-cos-cos-mult2;
val DERIV-cos-realpow2 = thm DERIV-cos-realpow2;
val DERIV-cos-realpow2a = thm DERIV-cos-realpow2a;
val DERIV-cos-realpow2b = thm DERIV-cos-realpow2b;
val DERIV-cos-cos-mult3 = thm DERIV-cos-cos-mult3;
val DERIV-sin-circle-all = thm DERIV-sin-circle-all;
val DERIV-sin-circle-all-zero = thm DERIV-sin-circle-all-zero;
val sin-cos-squared-add = thm sin-cos-squared-add;
val sin-cos-squared-add2 = thm sin-cos-squared-add2;
val sin-cos-squared-add3 = thm sin-cos-squared-add3;
val sin-squared-eq = thm sin-squared-eq;
val cos-squared-eq = thm cos-squared-eq;
val real-gt-one-ge-zero-add-less = thm real-gt-one-ge-zero-add-less;
val abs-sin-le-one = thm abs-sin-le-one;
val sin-ge-minus-one = thm sin-ge-minus-one;
val sin-le-one = thm sin-le-one;
val abs-cos-le-one = thm abs-cos-le-one;
val cos-ge-minus-one = thm cos-ge-minus-one;
val cos-le-one = thm cos-le-one;
val DERIV-fun-pow = thm DERIV-fun-pow;
val DERIV-fun-exp = thm DERIV-fun-exp;
val DERIV-fun-sin = thm DERIV-fun-sin;
val DERIV-fun-cos = thm DERIV-fun-cos;

val DERIV-intros = thms DERIV-intros;
val sin-cos-add = thm sin-cos-add;
val sin-add = thm sin-add;
val cos-add = thm cos-add;
val sin-cos-minus = thm sin-cos-minus;
val sin-minus = thm sin-minus;
val cos-minus = thm cos-minus;
val sin-diff = thm sin-diff;
val sin-diff2 = thm sin-diff2;
val cos-diff = thm cos-diff;
val cos-diff2 = thm cos-diff2;
val sin-double = thm sin-double;
val cos-double = thm cos-double;
val sin-paired = thm sin-paired;
val sin-gt-zero = thm sin-gt-zero;
val sin-gt-zero1 = thm sin-gt-zero1;
val cos-double-less-one = thm cos-double-less-one;
val cos-paired = thm cos-paired;
val cos-two-less-zero = thm cos-two-less-zero;
val cos-is-zero = thm cos-is-zero;
val pi-half = thm pi-half;
val cos-pi-half = thm cos-pi-half;
val pi-half-gt-zero = thm pi-half-gt-zero;
val pi-half-less-two = thm pi-half-less-two;
val pi-gt-zero = thm pi-gt-zero;
val pi-neq-zero = thm pi-neq-zero;
val pi-not-less-zero = thm pi-not-less-zero;
val pi-ge-zero = thm pi-ge-zero;
val minus-pi-half-less-zero = thm minus-pi-half-less-zero;
val sin-pi-half = thm sin-pi-half;
val cos-pi = thm cos-pi;
val sin-pi = thm sin-pi;
val sin-cos-eq = thm sin-cos-eq;
val minus-sin-cos-eq = thm minus-sin-cos-eq;
val cos-sin-eq = thm cos-sin-eq;
val sin-periodic-pi = thm sin-periodic-pi;
val sin-periodic-pi2 = thm sin-periodic-pi2;
val cos-periodic-pi = thm cos-periodic-pi;
val sin-periodic = thm sin-periodic;
val cos-periodic = thm cos-periodic;
val cos-npi = thm cos-npi;
val sin-npi = thm sin-npi;
val sin-npi2 = thm sin-npi2;
val cos-two-pi = thm cos-two-pi;
val sin-two-pi = thm sin-two-pi;
val sin-gt-zero2 = thm sin-gt-zero2;
val sin-less-zero = thm sin-less-zero;
val pi-less-4 = thm pi-less-4;
val cos-gt-zero = thm cos-gt-zero;

val cos-gt-zero-pi = thm cos-gt-zero-pi;
val cos-ge-zero = thm cos-ge-zero;
val sin-gt-zero-pi = thm sin-gt-zero-pi;
val sin-ge-zero = thm sin-ge-zero;
val cos-total = thm cos-total;
val sin-total = thm sin-total;
val reals-Archimedean4 = thm reals-Archimedean4;
val cos-zero-lemma = thm cos-zero-lemma;
val sin-zero-lemma = thm sin-zero-lemma;
val cos-zero-iff = thm cos-zero-iff;
val sin-zero-iff = thm sin-zero-iff;
val tan-zero = thm tan-zero;
val tan-pi = thm tan-pi;
val tan-npi = thm tan-npi;
val tan-minus = thm tan-minus;
val tan-periodic = thm tan-periodic;
val add-tan-eq = thm add-tan-eq;
val tan-add = thm tan-add;
val tan-double = thm tan-double;
val tan-gt-zero = thm tan-gt-zero;
val tan-less-zero = thm tan-less-zero;
val DERIV-tan = thm DERIV-tan;
val LIM-cos-div-sin = thm LIM-cos-div-sin;
val tan-total-pos = thm tan-total-pos;
val tan-total = thm tan-total;
val arcsin-pi = thm arcsin-pi;
val arcsin = thm arcsin;
val sin-arcsin = thm sin-arcsin;
val arcsin-bounded = thm arcsin-bounded;
val arcsin-lbound = thm arcsin-lbound;
val arcsin-ubound = thm arcsin-ubound;
val arcsin-lt-bounded = thm arcsin-lt-bounded;
val arcsin-sin = thm arcsin-sin;
val arcos = thm arcos;
val cos-arcos = thm cos-arcos;
val arcos-bounded = thm arcos-bounded;
val arcos-lbound = thm arcos-lbound;
val arcos-ubound = thm arcos-ubound;
val arcos-lt-bounded = thm arcos-lt-bounded;
val arcos-cos = thm arcos-cos;
val arcos-cos2 = thm arcos-cos2;
val arctan = thm arctan;
val tan-arctan = thm tan-arctan;
val arctan-bounded = thm arctan-bounded;
val arctan-lbound = thm arctan-lbound;
val arctan-ubound = thm arctan-ubound;
val arctan-tan = thm arctan-tan;
val arctan-zero-zero = thm arctan-zero-zero;
val cos-arctan-not-zero = thm cos-arctan-not-zero;

val tan-sec = thm tan-sec;
val DERIV-sin-add = thm DERIV-sin-add;
val cos-2npi = thm cos-2npi;
val cos-3over2-pi = thm cos-3over2-pi;
val sin-2npi = thm sin-2npi;
val sin-3over2-pi = thm sin-3over2-pi;
val cos-pi-eq-zero = thm cos-pi-eq-zero;
val DERIV-cos-add = thm DERIV-cos-add;
val isCont-cos = thm isCont-cos;
val isCont-sin = thm isCont-sin;
val isCont-exp = thm isCont-exp;
val sin-zero-abs-cos-one = thm sin-zero-abs-cos-one;
val exp-eq-one-iff = thm exp-eq-one-iff;
val cos-one-sin-zero = thm cos-one-sin-zero;
val real-root-less-mono = thm real-root-less-mono;
val real-root-le-mono = thm real-root-le-mono;
val real-root-less-iff = thm real-root-less-iff;
val real-root-le-iff = thm real-root-le-iff;
val real-root-eq-iff = thm real-root-eq-iff;
val real-root-pos-unique = thm real-root-pos-unique;
val real-root-mult = thm real-root-mult;
val real-root-inverse = thm real-root-inverse;
val real-root-divide = thm real-root-divide;
val real-sqrt-less-mono = thm real-sqrt-less-mono;
val real-sqrt-le-mono = thm real-sqrt-le-mono;
val real-sqrt-less-iff = thm real-sqrt-less-iff;
val real-sqrt-le-iff = thm real-sqrt-le-iff;
val real-sqrt-eq-iff = thm real-sqrt-eq-iff;
val real-sqrt-sos-less-one-iff = thm real-sqrt-sos-less-one-iff;
val real-sqrt-sos-eq-one-iff = thm real-sqrt-sos-eq-one-iff;
val real-divide-square-eq = thm real-divide-square-eq;
val real-sqrt-sum-squares-gt-zero1 = thm real-sqrt-sum-squares-gt-zero1;
val real-sqrt-sum-squares-gt-zero2 = thm real-sqrt-sum-squares-gt-zero2;
val real-sqrt-sum-squares-gt-zero3 = thm real-sqrt-sum-squares-gt-zero3;
val real-sqrt-sum-squares-gt-zero3a = thm real-sqrt-sum-squares-gt-zero3a;
val cos-x-y-ge-minus-one = thm cos-x-y-ge-minus-one;
val cos-x-y-ge-minus-one1a = thm cos-x-y-ge-minus-one1a;
val cos-x-y-le-one = thm cos-x-y-le-one;
val cos-x-y-le-one2 = thm cos-x-y-le-one2;
val cos-abs-x-y-ge-minus-one = thm cos-abs-x-y-ge-minus-one;
val cos-abs-x-y-le-one = thm cos-abs-x-y-le-one;
val minus-pi-less-zero = thm minus-pi-less-zero;
val arcos-ge-minus-pi = thm arcos-ge-minus-pi;
val sin-x-y-disj = thm sin-x-y-disj;
val cos-x-y-disj = thm cos-x-y-disj;
val real-sqrt-divide-less-zero = thm real-sqrt-divide-less-zero;
val polar-ex1 = thm polar-ex1;
val polar-ex2 = thm polar-ex2;
val polar-Ex = thm polar-Ex;

val real-sqrt-ge-abs1 = thm real-sqrt-ge-abs1;
val real-sqrt-ge-abs2 = thm real-sqrt-ge-abs2;
val real-sqrt-two-gt-zero = thm real-sqrt-two-gt-zero;
val real-sqrt-two-ge-zero = thm real-sqrt-two-ge-zero;
val real-sqrt-two-gt-one = thm real-sqrt-two-gt-one;
val STAR-exp-ln = thm STAR-exp-ln;
val hypreal-add-Infinitesimal-gt-zero = thm hypreal-add-Infinitesimal-gt-zero;
val NSDERIV-exp-ln-one = thm NSDERIV-exp-ln-one;
val DERIV-exp-ln-one = thm DERIV-exp-ln-one;
val isCont-inv-fun = thm isCont-inv-fun;
val isCont-inv-fun-inv = thm isCont-inv-fun-inv;
val LIM-fun-gt-zero = thm LIM-fun-gt-zero;
val LIM-fun-less-zero = thm LIM-fun-less-zero;
val LIM-fun-not-zero = thm LIM-fun-not-zero;
⟫

end

27 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 (*unfold exp-def*, *simp*)
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
    apply (simp add: power2-eq-square)
    apply (subgoal-tac real (Suc (Suc 0)) = 2)
    apply (erule ssubst)
    apply simp
    apply simp
    done
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 assumption
    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 ^ 2 / 2 * (1 / 2) ^ n) sums x^2
proof -
 have (%n. (1 / 2)^n) sums (1 / (1 - (1/2)))
   apply (rule geometric-sums)
   by (simp add: abs-interval-iff)
 also have (1::real) / (1 - 1/2) = 2
   by simp
 finally have (%n. (1 / 2)^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 ==> 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 aux3: (0::real) <= x ==> (1 + x + x^2)/(1 + x^2) <= 1 + x
  apply (subst pos-divide-le-eq)
  apply (simp add: zero-compare-simps)
  apply (simp add: ring-eq-simps zero-compare-simps)
done

lemma aux4: 0 <= x ==> 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 (rule aux3)
  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-eq-simps power2-eq-square power3-eq-cube*)
also have *... <= 1*
by (*auto intro*: *zero-le-power 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)
  proof -
    have 1 / (1 - x) = (1 - x + x) / (1 - x)
      by auto
    also have ... = (1 - x) / (1 - x) + x / (1 - x)
      by (rule add-divide-distrib)
    also have ... = 1 + x / (1-x)
      apply (subst add-right-cancel)
      apply (insert a, simp)
      done
    finally show ?thesis .
  qed
  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 e: −x − 2 ∗ x^2 <= − x / (1 − x)
    apply (rule mult-imp-le-div-pos)
    apply (insert prems, force)
    apply (auto simp add: ring-eq-simps power2-eq-square)
    apply (subgoal-tac − (x ∗ x) + x ∗ (x ∗ (x ∗ 2)) = x^2 ∗ (2 ∗ x − 1))
    apply (erule ssubst)
    apply (rule mult-nonneg-nonpos)
    apply auto
    apply (auto simp add: ring-eq-simps power2-eq-square)
    done
  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 <= 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
  apply (rule sym)
  apply (subst exp-ln-iff)
  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 0 <= x
```

```
  assume x <= 1
  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 diff-minus* [*THEN sym*])+
apply (*subst ln-div* [*THEN sym*])
apply *arith*
apply (*auto simp add*: *ring-eq-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*)
apply *arith*
apply *auto*

apply (*subgoal-tac 0 < exp 1*)
apply *arith*
apply *auto*
done
have $x * ln\ y - x * ln\ x = x * (ln\ y - ln\ x)$
by (*simp add*: *ring-eq-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 $... = 1 + (y - x)\ /\ x$
apply (*simp only*: *add-divide-distrib*)
apply (*simp add*: *prems*)
apply (*insert a*, *arith*)
done
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 $... = y - x$
by (*insert a*, *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 $ln\ y <= (y * ln\ x)\ /\ x$
apply (*subst pos-le-divide-eq*)
apply (*rule a*)
apply (*simp add*: *mult-ac*)
done
also have $... = y * (ln\ x\ /\ x)$
by *simp*
finally show *?thesis*
apply (*subst pos-divide-le-eq*)
apply (*rule b*)

apply (*simp add*: *mult-ac*)
done
qed

end

28 Poly: Univariate Real Polynomials

theory *Poly*
imports *Ln*
begin

Application of polynomial as a real function.

consts *poly* :: *real list* => *real* => *real*
primrec
poly-Nil: *poly* [] *x* = *0*
poly-Cons: *poly* (*h*#*t*) *x* = *h* + *x* ∗ *poly t x*

28.1 Arithmetic Operations on Polynomials

addition

consts +++ :: [*real list*, *real list*] => *real list* (**infixl** *65*)
primrec
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

consts %∗ :: [*real*, *real list*] => *real list* (**infixl** *70*)
primrec
cmult-Nil: *c* %∗ [] = []
cmult-Cons: *c* %∗ (*h*#*t*) = (*c* ∗ *h*)#(*c* %∗ *t*)

Multiplication by a polynomial

consts ∗∗∗ :: [*real list*, *real list*] => *real list* (**infixl** *70*)
primrec
pmult-Nil: [] ∗∗∗ *l2* = []
pmult-Cons: (*h*#*t*) ∗∗∗ *l2* = (*if t* = [] *then h* %∗ *l2*
else (*h* %∗ *l2*) +++ ((*0*) # (*t* ∗∗∗ *l2*)))

Repeated multiplication by a polynomial

consts *mulexp* :: [*nat*, *real list*, *real list*] => *real list*
primrec
mulexp-zero: *mulexp 0 p q* = *q*
mulexp-Suc: *mulexp* (*Suc n*) *p q* = *p* ∗∗∗ *mulexp n p q*

Exponential

consts %^ :: [*real list*, *nat*] => *real list* (**infixl** *80*)
primrec
pexp-0: *p* %^ *0* = [*1*]
pexp-Suc: *p* %^ (*Suc n*) = *p* *** (*p* %^ *n*)

Quotient related value of dividing a polynomial by x + a

consts *pquot* :: [*real list*, *real*] => *real list*
primrec
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*))

Differentiation of polynomials (needs an auxiliary function).

consts *pderiv-aux* :: *nat* => *real list* => *real list*
primrec
pderiv-aux-Nil: *pderiv-aux n* [] = []
pderiv-aux-Cons: *pderiv-aux n* (*h*#*t*) =
(*real n* * *h*)#(*pderiv-aux* (*Suc n*) *t*)

normalization of polynomials (remove extra 0 coeff)

consts *pnormalize* :: *real list* => *real list*
primrec
pnormalize-Nil: *pnormalize* [] = []
pnormalize-Cons: *pnormalize* (*h*#*p*) = (*if* ((*pnormalize p*) = [])
then (*if* (*h* = *0*) *then* [] *else* [*h*])
else (*h*#(*pnormalize p*)))

Other definitions

constdefs
poly-minus :: *real list* => *real list* (−− - [*80*] *80*)
−− *p* == (− *1*) %* *p*

pderiv :: *real list* => *real list*
pderiv p == *if p* = [] *then* [] *else pderiv-aux 1* (*tl p*)

divides :: [*real list*,*real list*] => *bool* (**infixl** *divides 70*)
p1 divides p2 == ∃ *q*. *poly p2* = *poly*(*p1* *** *q*)

order :: *real* => *real list* => *nat*
— order of a polynomial
order a p == (@*n*. ([−*a*, *1*] %^ *n*) *divides p* &
~ (([−*a*, *1*] %^ (*Suc n*)) *divides p*))

degree :: *real list* => *nat*
— degree of a polynomial
degree p == *length* (*pnormalize p*)

rsquarefree :: *real list* => *bool*
— squarefree polynomials — NB with respect to real roots only.
rsquarefree p == *poly p* ≠ *poly* [] &
(∀ *a*. (*order a p* = *0*) | (*order a p* = *1*))

lemma *padd-Nil2*: *p* +++ [] = *p*
by (*induct p*, *auto*)
declare *padd-Nil2* [*simp*]

lemma *padd-Cons-Cons*: (*h1* # *p1*) +++ (*h2* # *p2*) = (*h1* + *h2*) # (*p1* +++ *p2*)
by *auto*

lemma *pminus-Nil*: −− [] = []
by (*simp add*: *poly-minus-def*)
declare *pminus-Nil* [*simp*]

lemma *pmult-singleton*: [*h1*] *** *p1* = *h1* %* *p1*
by *simp*

lemma *poly-ident-mult*: *1* %* *t* = *t*
by (*induct t*, *auto*)
declare *poly-ident-mult* [*simp*]

lemma *poly-simple-add-Cons*: [*a*] +++ ((*0*)#*t*) = (*a*#*t*)
by *simp*
declare *poly-simple-add-Cons* [*simp*]

Handy general properties

lemma *padd-commut*: *b* +++ *a* = *a* +++ *b*
apply (*subgoal-tac* ∀ *a*. *b* +++ *a* = *a* +++ *b*)
apply (*induct-tac* [*2*] *b*, *auto*)
apply (*rule padd-Cons* [*THEN ssubst*])
apply (*case-tac aa*, *auto*)
done

lemma *padd-assoc* [*rule-format*]: ∀ *b c*. (*a* +++ *b*) +++ *c* = *a* +++ (*b* +++ *c*)
apply (*induct a*, *simp*, *clarify*)
apply (*case-tac b*, *simp-all*)
done

lemma *poly-cmult-distr* [*rule-format*]:
∀ *q*. *a* %* (*p* +++ *q*) = (*a* %* *p* +++ *a* %* *q*)
apply (*induct p*, *simp*, *clarify*)
apply (*case-tac q*)
apply (*simp-all add*: *right-distrib*)
done

lemma *pmult-by-x*: $[0, 1] \ast\ast\ast t = ((0)\#t)$
apply (*induct t*, *simp*)
apply (*auto simp add*: *poly-ident-mult padd-commut*)
done
declare *pmult-by-x* [*simp*]

properties of evaluation of polynomials.

lemma *poly-add*: $poly\ (p1 +\!+\!+ p2)\ x = poly\ p1\ x + poly\ p2\ x$
apply (*subgoal-tac* $\forall p2.\ poly\ (p1 +\!+\!+ p2)\ x = poly\ (p1)\ x + poly\ (p2)\ x$)
apply (*induct-tac* [2] *p1*, *auto*)
apply (*case-tac p2*)
apply (*auto simp add*: *right-distrib*)
done

lemma *poly-cmult*: $poly\ (c \%\ast p)\ x = c \ast poly\ p\ x$
apply (*induct p*)
apply (*case-tac* [2] *x=0*)
apply (*auto simp add*: *right-distrib mult-ac*)
done

lemma *poly-minus*: $poly\ (-\!-\ p)\ x = -\ (poly\ p\ x)$
apply (*simp add*: *poly-minus-def*)
apply (*auto simp add*: *poly-cmult*)
done

lemma *poly-mult*: $poly\ (p1 \ast\ast\ast p2)\ x = poly\ p1\ x \ast poly\ p2\ x$
apply (*subgoal-tac* $\forall p2.\ poly\ (p1 \ast\ast\ast p2)\ x = poly\ p1\ x \ast poly\ p2\ x$)
apply (*simp* (*no-asm-simp*))
apply (*induct p1*)
apply (*auto simp add*: *poly-cmult*)
apply (*case-tac p1*)
apply (*auto simp add*: *poly-cmult poly-add left-distrib right-distrib mult-ac*)
done

lemma *poly-exp*: $poly\ (p \%\hat{}\ n)\ x = (poly\ p\ x)\ \hat{}\ n$
apply (*induct n*)
apply (*auto simp add*: *poly-cmult poly-mult*)
done

More Polynomial Evaluation Lemmas

lemma *poly-add-rzero*: $poly\ (a +\!+\!+ [])\ x = poly\ a\ x$
by *simp*
declare *poly-add-rzero* [*simp*]

lemma *poly-mult-assoc*: $poly\ ((a \ast\ast\ast b) \ast\ast\ast c)\ x = poly\ (a \ast\ast\ast (b \ast\ast\ast c))\ x$
by (*simp add*: *poly-mult real-mult-assoc*)

lemma *poly-mult-Nil2*: $poly\ (p \ast\ast\ast [])\ x = 0$

by (*induct p*, *auto*)
declare *poly-mult-Nil2* [*simp*]

lemma *poly-exp-add*: *poly* (*p* %^ (*n* + *d*)) *x* = *poly*(*p* %^ *n* *** *p* %^ *d*) *x*
apply (*induct n*)
apply (*auto simp add*: *poly-mult real-mult-assoc*)
done

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*) *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*) *x* :> *D*
by (*rule lemma-DERIV-subst*, *rule DERIV-add*, *auto*)

lemma *poly-DERIV*: *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
declare *poly-DERIV* [*simp*]

Consequences of the derivative theorem above

lemma *poly-differentiable*: (%*x*. *poly p x*) *differentiable x*

apply (*simp add*: *differentiable-def*)
apply (*blast intro*: *poly-DERIV*)
done
declare *poly-differentiable* [*simp*]

lemma *poly-isCont*: *isCont* (%*x*. *poly p x*) *x*
by (*rule poly-DERIV* [*THEN DERIV-isCont*])
declare *poly-isCont* [*simp*]

lemma *poly-IVT-pos*: [| *a* < *b*; *poly p a* < *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* < *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* ==>
   ∃*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*: ∀ *p2 n*. *poly* (*pderiv-aux n* (*p1* +++ *p2*)) *x* =
          poly (*pderiv-aux n p1* +++ *pderiv-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-aux n* (*p1* +++ *p2*)) *x* =
    poly (*pderiv-aux n p1* +++ *pderiv-aux n p2*) *x*

apply (*simp add*: *lemma-poly-pderiv-aux-add*)
done

lemma *lemma-poly-pderiv-aux-cmult*: $\forall n.$ *poly* (*pderiv-aux n* (*c* %∗ *p*)) *x* = *poly* (*c* %∗ *pderiv-aux n p*) *x*
apply (*induct p*)
apply (*auto simp add*: *poly-cmult mult-ac*)
done

lemma *poly-pderiv-aux-cmult*: *poly* (*pderiv-aux n* (*c* %∗ *p*)) *x* = *poly* (*c* %∗ *pderiv-aux n p*) *x*
by (*simp add*: *lemma-poly-pderiv-aux-cmult*)

lemma *poly-pderiv-aux-minus*:
  poly (*pderiv-aux n* (−− *p*)) *x* = *poly* (−− *pderiv-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-aux* (*Suc n*) *p*) *x* = *poly* ((*pderiv-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-aux* (*Suc n*) *p*) *x* = *poly* ((*pderiv-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

28.2 Key Property: if *f a* = (*0*::'*a*) then *x* − *a* divides *p x*

lemma *lemma-poly-linear-rem*: $\forall h.\ \exists q\ r.\ h\#t = [r]\ +++\ [-a,\ 1]\ ***\ q$
apply (*induct t*, *safe*)
apply (*rule-tac x* = [] **in** *exI*)
apply (*rule-tac x* = *h* **in** *exI*, *simp*)
apply (*drule-tac x* = *aa* **in** *spec*, *safe*)
apply (*rule-tac x* = *r*#*q* **in** *exI*)
apply (*rule-tac x* = *a***r* + *h* **in** *exI*)
apply (*case-tac q*, *auto*)
done

lemma *poly-linear-rem*: $\exists q\ r.\ h\#t = [r]\ +++\ [-a,\ 1]\ ***\ q$
by (*cut-tac t* = *t* **and** *a* = *a* **in** *lemma-poly-linear-rem*, *auto*)

lemma *poly-linear-divides*: (*poly p a* = *0*) = ((*p* = []) | ($\exists q.\ p = [-a,\ 1]\ ***\ q$))
apply (*auto simp add*: *poly-add poly-cmult right-distrib*)
apply (*case-tac p*, *simp*)

apply (*cut-tac h = aa* **and** *t = list* **and** *a = a* **in** *poly-linear-rem*, *safe*)
apply (*case-tac q*, *auto*)
apply (*drule-tac x = []* **in** *spec*, *simp*)
apply (*auto simp add*: *poly-add poly-cmult real-add-assoc*)
apply (*drule-tac x = aa#lista* **in** *spec*, *auto*)
done

lemma *lemma-poly-length-mult*: $\forall h\ k\ a.$ *length* (*k %* p +++ (h # (a %* p))*) = *Suc* (*length p*)
by (*induct p*, *auto*)
declare *lemma-poly-length-mult* [*simp*]

lemma *lemma-poly-length-mult2*: $\forall h\ k.$ *length* (*k %* p +++ (h # p)*) = *Suc* (*length p*)
by (*induct p*, *auto*)
declare *lemma-poly-length-mult2* [*simp*]

lemma *poly-length-mult*: *length*([−*a*,*1*] ***** *q*) = *Suc* (*length q*)
by *auto*
declare *poly-length-mult* [*simp*]

28.3 Polynomial length

lemma *poly-cmult-length*: *length* (*a %* p*) = *length p*
by (*induct p*, *auto*)
declare *poly-cmult-length* [*simp*]

lemma *poly-add-length* [*rule-format*]:
$\forall$ *p2*. *length* (*p1 +++ p2*) =
(*if* (*length p1 < length p2*) *then length p2 else length p1*)
apply (*induct p1*, *simp-all*, *arith*)
done

lemma *poly-root-mult-length*: *length*([*a*,*b*] ***** *p*) = *Suc* (*length p*)
by (*simp add*: *poly-cmult-length poly-add-length*)
declare *poly-root-mult-length* [*simp*]

lemma *poly-mult-not-eq-poly-Nil*: (*poly* (*p *** q*) *x* ≠ *poly* [] *x*) =
(*poly p x* ≠ *poly* [] *x* & *poly q x* ≠ *poly* [] *x*)
apply (*auto simp add*: *poly-mult*)
done
declare *poly-mult-not-eq-poly-Nil* [*simp*]

lemma *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 *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 *poly-roots-index-lemma* [*rule-format*]:
  $\forall p\ x.\ poly\ p\ x \neq poly\ []\ x\ \&\ length\ p = n$
  $--> (\exists i.\ \forall x.\ (poly\ p\ x = (0::real)) --> (\exists m.\ (m \leq n\ \&\ x = i\ m)))$
apply (*induct n*, *safe*)
apply (*rule ccontr*)
apply (*subgoal-tac* $\exists a.\ poly\ p\ a = 0$, *safe*)
apply (*drule poly-linear-divides* [*THEN iffD1*], *safe*)
apply (*drule-tac* $x = q$ **in** *spec*)
apply (*drule-tac* $x = x$ **in** *spec*)
apply (*simp del*: *poly-Nil pmult-Cons*)
apply (*erule exE*)
apply (*drule-tac* $x = \%m.\ if\ m = Suc\ n\ then\ a\ else\ i\ m$ **in** *spec*, *safe*)
apply (*drule poly-mult-eq-zero-disj* [*THEN iffD1*], *safe*)
apply (*drule-tac* $x = Suc\ (length\ q)$ **in** *spec*)
apply *simp*
apply (*drule-tac* $x = xa$ **in** *spec*, *safe*)
apply (*drule-tac* $x = m$ **in** *spec*, *simp*, *blast*)
done
lemmas *poly-roots-index-lemma2* = *conjI* [*THEN poly-roots-index-lemma*, *standard*]

lemma *poly-roots-index-length*: *poly p x* $\neq$ *poly* [] *x* ==>
    $\exists i.\ \forall x.\ (poly\ p\ x = 0) --> (\exists n.\ n \leq length\ p\ \&\ x = i\ n)$
by (*blast intro*: *poly-roots-index-lemma2*)

lemma *poly-roots-finite-lemma*: *poly p x* $\neq$ *poly* [] *x* ==>
    $\exists N\ i.\ \forall x.\ (poly\ p\ x = 0) --> (\exists n.\ (n::nat) < N\ \&\ x = i\ n)$
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 *real-finite-lemma* [*rule-format* (*no-asm*)]:
    $\forall P.\ (\forall x.\ P\ x --> (\exists n.\ n < N\ \&\ x = (j::nat=>real)\ n))$
    $--> (\exists a.\ \forall x.\ P\ x --> x < a)$
apply (*induct N*, *simp*, *safe*)
apply (*drule-tac* $x = \%z.\ P\ z\ \&\ (z \neq j\ N)$ **in** *spec*)
apply (*auto simp add*: *less-Suc-eq*)
apply (*rename-tac N P a*)
apply (*rule-tac* $x = abs\ a + abs\ (j\ N) + 1$ **in** *exI*)
apply *safe*
apply (*drule-tac* $x = x$ **in** *spec*, *safe*)
apply (*drule-tac* $x = j\ n$ **in** *spec*, *arith+*)
done

lemma *poly-roots-finite*: (*poly p* ≠ *poly* []) =
(∃ *N j*. ∀ *x*. *poly p x* = *0* −−> (∃ *n*. (*n*::*nat*) < *N* & *x* = *j n*))
apply *safe*
apply (*erule swap*, *rule ext*)
apply (*rule ccontr*)
apply (*clarify dest*!: *poly-roots-finite-lemma*)
apply (*clarify dest*!: *real-finite-lemma*)
apply (*drule-tac x* = *a* **in** *fun-cong*, *auto*)
done

Entirety and Cancellation for polynomials

lemma *poly-entire-lemma*: [| *poly p* ≠ *poly* [] ; *poly q* ≠ *poly* [] |]
==> *poly* (*p* *** *q*) ≠ *poly* []
apply (*auto simp add*: *poly-roots-finite*)
apply (*rule-tac x* = *N* + *Na* **in** *exI*)
apply (*rule-tac x* = %*n*. *if n* < *N then j n else ja* (*n* − *N*) **in** *exI*)
apply (*auto simp add*: *poly-mult-eq-zero-disj*, *force*)
done

lemma *poly-entire*: (*poly* (*p* *** *q*) = *poly* []) = ((*poly p* = *poly* []) | (*poly q* = *poly* []))
apply (*auto intro*: *ext dest*: *fun-cong simp add*: *poly-entire-lemma poly-mult*)
apply (*blast intro*: *ccontr dest*: *poly-entire-lemma poly-mult* [*THEN subst*])
done

lemma *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 *poly-add-minus-zero-iff*: (*poly* (*p* +++ −− *q*) = *poly* []) = (*poly p* = *poly q*)
by (*auto simp add*: *poly-add poly-minus-def fun-eq poly-cmult*)

lemma *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*)

lemma *poly-mult-left-cancel*: (*poly* (*p* *** *q*) = *poly* (*p* *** *r*)) = (*poly p* = *poly* [] | *poly q* = *poly r*)
apply (*rule-tac p1* = *p* *** *q* **in** *poly-add-minus-zero-iff* [*THEN subst*])
apply (*auto intro*: *ext simp add*: *poly-add-minus-mult-eq poly-entire poly-add-minus-zero-iff*)
done

lemma *real-mult-zero-disj-iff*: (*x* * *y* = *0*) = (*x* = (*0*::*real*) | *y* = *0*)
by *simp*

lemma *poly-exp-eq-zero*:
(*poly* (*p* %^ *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-tac n*)
apply (*auto simp add*: *poly-mult real-mult-zero-disj-iff*)
done
declare *poly-exp-eq-zero* [*simp*]

lemma *poly-prime-eq-zero*: *poly* [*a*,*1*] ≠ *poly* []
apply (*simp add*: *fun-eq*)
apply (*rule-tac x* = *1* − *a* **in** *exI*, *simp*)
done
declare *poly-prime-eq-zero* [*simp*]

lemma *poly-exp-prime-eq-zero*: (*poly* ([*a*, *1*] %^ *n*) ≠ *poly* [])
by *auto*
declare *poly-exp-prime-eq-zero* [*simp*]

A more constructive notion of polynomials being trivial

lemma *poly-zero-lemma*: *poly* (*h* # *t*) = *poly* [] ==> *h* = *0* & *poly t* = *poly* []
apply (*simp add*: *fun-eq*)
apply (*case-tac h* = *0*)
apply (*drule-tac* [*2*] *x* = *0* **in** *spec*, *auto*)
apply (*case-tac poly t* = *poly* [], *simp*)
apply (*auto simp add*: *poly-roots-finite real-mult-zero-disj-iff*)
apply (*drule real-finite-lemma*, *safe*)
apply (*drule-tac x* = *abs a* + *1* **in** *spec*)+
apply *arith*
done

lemma *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

declare *real-mult-zero-disj-iff* [*simp*]

lemma *pderiv-aux-iszero* [*rule-format*, *simp*]:
∀ *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*) ≠ *0*
==> (*list-all* (%*c*. *c* = *0*) (*pderiv-aux* (*number-of n*) *p*) =
list-all (%*c*. *c* = *0*) *p*)

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

lemma *pderiv-iszero* [*rule-format*]:
    poly (*pderiv p*) = *poly* [] −−> (∃ *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

Basics of divisibility.

lemma *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* = −*a* **in** *spec*)
apply (*auto simp add*: *poly-linear-divides poly-add poly-cmult left-distrib* [*symmetric*])
apply (*rule-tac x* = *qa* *** *q* **in** *exI*)
apply (*rule-tac* [*2*] *x* = *p* *** *qa* **in** *exI*)
apply (*auto simp add*: *poly-add poly-mult poly-cmult mult-ac*)
done

lemma *poly-divides-refl*: *p divides p*
apply (*simp add*: *divides-def*)
apply (*rule-tac x* = [*1*] **in** *exI*)
apply (*auto simp add*: *poly-mult fun-eq*)
done
declare *poly-divides-refl* [*simp*]

lemma *poly-divides-trans*: [| *p divides q*; *q divides r* |] ==> *p divides r*
apply (*simp add*: *divides-def*, *safe*)
apply (*rule-tac x* = *qa* *** *qaa* **in** *exI*)
apply (*auto simp add*: *poly-mult fun-eq real-mult-assoc*)
done

lemma *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 *poly-exp-divides*: [| (*p* %^ *n*) *divides q*; *m*≤*n* |] ==> (*p* %^ *m*) *divides q*
by (*blast intro*: *poly-divides-exp poly-divides-trans*)

lemma *poly-divides-add*:
[| *p divides q*; *p divides r* |] ==> *p divides* (*q* +++ *r*)
apply (*simp add*: *divides-def*, *auto*)
apply (*rule-tac x* = *qa* +++ *qaa* **in** *exI*)
apply (*auto simp add*: *poly-add fun-eq poly-mult right-distrib*)
done

lemma *poly-divides-diff*:
[| *p divides q*; *p divides* (*q* +++ *r*) |] ==> *p divides r*
apply (*simp add*: *divides-def*, *auto*)
apply (*rule-tac x* = *qaa* +++ −− *qa* **in** *exI*)
apply (*auto simp add*: *poly-add fun-eq poly-mult poly-minus right-diff-distrib compare-rls add-ac*)
done

lemma *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 *poly-divides-zero*: *poly p* = *poly* [] ==> *q divides p*
apply (*simp add*: *divides-def*)
apply (*auto simp add*: *fun-eq poly-mult*)
done

lemma *poly-divides-zero2*: *q divides* []
apply (*simp add*: *divides-def*)
apply (*rule-tac x* = [] **in** *exI*)
apply (*auto simp add*: *fun-eq*)
done

declare *poly-divides-zero2* [*simp*]

At last, we can consider the order of a root.

lemma *poly-order-exists-lemma* [*rule-format*]:
  ∀ *p*. *length p = d* −−> *poly p ≠ poly* []
    −−> (∃ *n q*. *p = mulexp n* [−*a*, *1*] *q* & *poly q a ≠ 0*)
apply (*induct d*)
apply (*simp add*: *fun-eq*, *safe*)
apply (*case-tac poly p a = 0*)
apply (*drule-tac poly-linear-divides* [*THEN iffD1*], *safe*)
apply (*drule-tac x = q* **in** *spec*)
apply (*drule-tac poly-entire-neg* [*THEN iffD1*], *safe*, *force*, *blast*)
apply (*rule-tac x = Suc n* **in** *exI*)
apply (*rule-tac x = qa* **in** *exI*)
apply (*simp del*: *pmult-Cons*)
apply (*rule-tac x = 0* **in** *exI*, *force*)
done

lemma *poly-order-exists*:
  [| *length p = d*; *poly p ≠ poly* [] |]
  ==> ∃ *n*. ([−*a*, *1*] %^ *n*) *divides p* &
      ~(([−*a*, *1*] %^ (*Suc n*)) *divides p*)
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* [− *a*, *1*] *q*) ≠ *poly* ([− *a*, *1*] %^ *Suc n* *** *qa*))

apply *simp*
apply (*induct-tac n*)
apply (*simp del*: *pmult-Cons pexp-Suc*)
apply (*erule-tac Pa = poly q a = 0* **in** *swap*)
apply (*simp add*: *poly-add poly-cmult*)
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

lemma *poly-one-divides*: [*1*] *divides p*
by (*simp add*: *divides-def*, *auto*)
declare *poly-one-divides* [*simp*]

lemma *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 m = y* **and** *n = 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 *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 *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 *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 *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 *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 *pexp-one*: *p* % ^ (*Suc 0*) = *p*
apply (*induct p*)
apply (*auto simp add*: *numeral-1-eq-1*)
done
declare *pexp-one* [*simp*]

lemma *lemma-order-root* [*rule-format*]:
  ∀ *p a*. *0 < n* & [− *a*, *1*] % ^ *n divides p* & ~ [− *a*, *1*] % ^ (*Suc n*) *divides p*
    −−> *poly p a = 0*

apply (*induct n*, *blast*)
apply (*auto simp add*: *divides-def poly-mult simp del*: *pmult-Cons*)
done

lemma *order-root*: (*poly p a = 0*) = ((*poly p = poly* []) | *order a p ≠ 0*)
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*)
apply (*blast intro*: *lemma-order-root*)
done

lemma *order-divides*: (([−*a*, *1*] %^ *n*) *divides p*) = ((*poly p = poly* []) | *n* ≤ *order a p*)
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

lemma *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 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 *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 swap*, *assumption*)
apply (*rotate-tac 3*, *erule swap*)
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 field-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)
apply (blast intro: lemma-order-pderiv)
done
```

Now justify the standard squarefree decomposition, i.e. f / gcd(f,f'). *) (* 'a la Harrison

```
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-root2*: *poly p* ≠ *poly* [] ==> (*poly p a* = *0*) = (*order a p* ≠ *0*)
by (*rule order-root* [*THEN ssubst*], *auto*)

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 (*drule spec*, *auto*)
done

lemma *pmult-one*: [*1*] *** *p* = *p*
by *auto*
declare *pmult-one* [*simp*]

lemma *poly-Nil-zero*: *poly* [] = *poly* [*0*]
by (*simp add*: *fun-eq*)

lemma *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 a1* = *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

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

Normalization of a polynomial.

lemma *poly-normalize*: *poly* (*pnormalize p*) = *poly p*
apply (*induct p*)
apply (*auto simp add*: *fun-eq*)
done
declare *poly-normalize* [*simp*]

The degree of a polynomial.

lemma *lemma-degree-zero* [*rule-format*]:
list-all (%*c*. *c* = *0*) *p* −−> *pnormalize p* = []
by (*induct p*, *auto*)

lemma *degree-zero*: *poly p* = *poly* [] ==> *degree p* = *0*
apply (*simp add*: *degree-def*)
apply (*case-tac pnormalize p* = [])
apply (*auto dest*: *lemma-degree-zero simp add*: *poly-zero*)
done

Tidier versions of finiteness of roots.

lemma *poly-roots-finite-set*: *poly p* ≠ *poly* [] ==> *finite* {*x*. *poly p x* = *0*}
apply (*auto simp add*: *poly-roots-finite*)
apply (*rule-tac B* = {*x*::*real*. ∃ *n*. (*n*::*nat*) < *N* & (*x* = *j n*) } **in** *finite-subset*)
apply (*induct-tac* [*2*] *N*, *auto*)
apply (*subgoal-tac* {*x*::*real*. ∃ *na*. *na* < *Suc n* & (*x* = *j na*) } = { (*j n*) } *Un* {*x*.
∃ *na*. *na* < *n* & (*x* = *j na*) })
apply (*auto simp add*: *less-Suc-eq*)
done

bound for polynomial.

lemma *poly-mono*: *abs*(*x*) ≤ *k* ==> *abs*(*poly p x*) ≤ *poly* (*map abs p*) *k*

apply (*induct p*, *auto*)
apply (*rule-tac j* = *abs a* + *abs* (*x* ∗ *poly p x*) **in** *real-le-trans*)
apply (*rule abs-triangle-ineq*)
apply (*auto intro*!: *mult-mono simp add*: *abs-mult*, *arith*)
done

ML
⟪
val padd-Nil2 = *thm padd-Nil2*;
val padd-Cons-Cons = *thm padd-Cons-Cons*;
val pminus-Nil = *thm pminus-Nil*;
val pmult-singleton = *thm pmult-singleton*;
val poly-ident-mult = *thm poly-ident-mult*;
val poly-simple-add-Cons = *thm poly-simple-add-Cons*;
val padd-commut = *thm padd-commut*;
val padd-assoc = *thm padd-assoc*;
val poly-cmult-distr = *thm poly-cmult-distr*;
val pmult-by-x = *thm pmult-by-x*;
val poly-add = *thm poly-add*;
val poly-cmult = *thm poly-cmult*;
val poly-minus = *thm poly-minus*;
val poly-mult = *thm poly-mult*;
val poly-exp = *thm poly-exp*;
val poly-add-rzero = *thm poly-add-rzero*;
val poly-mult-assoc = *thm poly-mult-assoc*;
val poly-mult-Nil2 = *thm poly-mult-Nil2*;
val poly-exp-add = *thm poly-exp-add*;
val pderiv-Nil = *thm pderiv-Nil*;
val pderiv-singleton = *thm pderiv-singleton*;
val pderiv-Cons = *thm pderiv-Cons*;
val DERIV-cmult2 = *thm DERIV-cmult2*;
val DERIV-pow2 = *thm DERIV-pow2*;
val lemma-DERIV-poly1 = *thm lemma-DERIV-poly1*;
val lemma-DERIV-poly = *thm lemma-DERIV-poly*;
val DERIV-add-const = *thm DERIV-add-const*;
val poly-DERIV = *thm poly-DERIV*;
val poly-differentiable = *thm poly-differentiable*;
val poly-isCont = *thm poly-isCont*;
val poly-IVT-pos = *thm poly-IVT-pos*;
val poly-IVT-neg = *thm poly-IVT-neg*;
val poly-MVT = *thm poly-MVT*;
val lemma-poly-pderiv-aux-add = *thm lemma-poly-pderiv-aux-add*;
val poly-pderiv-aux-add = *thm poly-pderiv-aux-add*;
val lemma-poly-pderiv-aux-cmult = *thm lemma-poly-pderiv-aux-cmult*;
val poly-pderiv-aux-cmult = *thm poly-pderiv-aux-cmult*;
val poly-pderiv-aux-minus = *thm poly-pderiv-aux-minus*;
val lemma-poly-pderiv-aux-mult1 = *thm lemma-poly-pderiv-aux-mult1*;
val lemma-poly-pderiv-aux-mult = *thm lemma-poly-pderiv-aux-mult*;
val lemma-poly-pderiv-add = *thm lemma-poly-pderiv-add*;

val poly-pderiv-add = thm poly-pderiv-add;
val poly-pderiv-cmult = thm poly-pderiv-cmult;
val poly-pderiv-minus = thm poly-pderiv-minus;
val lemma-poly-mult-pderiv = thm lemma-poly-mult-pderiv;
val poly-pderiv-mult = thm poly-pderiv-mult;
val poly-pderiv-exp = thm poly-pderiv-exp;
val poly-pderiv-exp-prime = thm poly-pderiv-exp-prime;
val lemma-poly-linear-rem = thm lemma-poly-linear-rem;
val poly-linear-rem = thm poly-linear-rem;
val poly-linear-divides = thm poly-linear-divides;
val lemma-poly-length-mult = thm lemma-poly-length-mult;
val lemma-poly-length-mult2 = thm lemma-poly-length-mult2;
val poly-length-mult = thm poly-length-mult;
val poly-cmult-length = thm poly-cmult-length;
val poly-add-length = thm poly-add-length;
val poly-root-mult-length = thm poly-root-mult-length;
val poly-mult-not-eq-poly-Nil = thm poly-mult-not-eq-poly-Nil;
val poly-mult-eq-zero-disj = thm poly-mult-eq-zero-disj;
val poly-normalized-nil = thm poly-normalized-nil;
val poly-roots-index-lemma = thm poly-roots-index-lemma;
val poly-roots-index-lemma2 = thms poly-roots-index-lemma2;
val poly-roots-index-length = thm poly-roots-index-length;
val poly-roots-finite-lemma = thm poly-roots-finite-lemma;
val real-finite-lemma = thm real-finite-lemma;
val poly-roots-finite = thm poly-roots-finite;
val poly-entire-lemma = thm poly-entire-lemma;
val poly-entire = thm poly-entire;
val poly-entire-neg = thm poly-entire-neg;
val fun-eq = thm fun-eq;
val poly-add-minus-zero-iff = thm poly-add-minus-zero-iff;
val poly-add-minus-mult-eq = thm poly-add-minus-mult-eq;
val poly-mult-left-cancel = thm poly-mult-left-cancel;
val real-mult-zero-disj-iff = thm real-mult-zero-disj-iff;
val poly-exp-eq-zero = thm poly-exp-eq-zero;
val poly-prime-eq-zero = thm poly-prime-eq-zero;
val poly-exp-prime-eq-zero = thm poly-exp-prime-eq-zero;
val poly-zero-lemma = thm poly-zero-lemma;
val poly-zero = thm poly-zero;
val pderiv-aux-iszero = thm pderiv-aux-iszero;
val pderiv-aux-iszero-num = thm pderiv-aux-iszero-num;
val pderiv-iszero = thm pderiv-iszero;
val pderiv-zero-obj = thm pderiv-zero-obj;
val pderiv-zero = thm pderiv-zero;
val poly-pderiv-welldef = thm poly-pderiv-welldef;
val poly-primes = thm poly-primes;
val poly-divides-refl = thm poly-divides-refl;
val poly-divides-trans = thm poly-divides-trans;
val poly-divides-exp = thm poly-divides-exp;
val poly-exp-divides = thm poly-exp-divides;

val poly-divides-add = thm poly-divides-add;
val poly-divides-diff = thm poly-divides-diff;
val poly-divides-diff2 = thm poly-divides-diff2;
val poly-divides-zero = thm poly-divides-zero;
val poly-divides-zero2 = thm poly-divides-zero2;
val poly-order-exists-lemma = thm poly-order-exists-lemma;
val poly-order-exists = thm poly-order-exists;
val poly-one-divides = thm poly-one-divides;
val poly-order = thm poly-order;
val some1-equalityD = thm some1-equalityD;
val order = thm order;
val order2 = thm order2;
val order-unique = thm order-unique;
val order-unique-lemma = thm order-unique-lemma;
val order-poly = thm order-poly;
val pexp-one = thm pexp-one;
val lemma-order-root = thm lemma-order-root;
val order-root = thm order-root;
val order-divides = thm order-divides;
val order-decomp = thm order-decomp;
val order-mult = thm order-mult;
val lemma-order-pderiv = thm lemma-order-pderiv;
val order-pderiv = thm order-pderiv;
val poly-squarefree-decomp-order = thm poly-squarefree-decomp-order;
val poly-squarefree-decomp-order2 = thm poly-squarefree-decomp-order2;
val order-root2 = thm order-root2;
val order-pderiv2 = thm order-pderiv2;
val rsquarefree-roots = thm rsquarefree-roots;
val pmult-one = thm pmult-one;
val poly-Nil-zero = thm poly-Nil-zero;
val rsquarefree-decomp = thm rsquarefree-decomp;
val poly-squarefree-decomp = thm poly-squarefree-decomp;
val poly-normalize = thm poly-normalize;
val lemma-degree-zero = thm lemma-degree-zero;
val degree-zero = thm degree-zero;
val poly-roots-finite-set = thm poly-roots-finite-set;
val poly-mono = thm poly-mono;
⟩⟩

end

29 Log: Logarithms: Standard Version

theory *Log*
imports *Transcendental*
begin

constdefs

powr :: [*real*,*real*] => *real* (**infixr** *powr* 80)
— exponentation with real exponent
x powr a == *exp*(*a* * *ln x*)

log :: [*real*,*real*] => *real*
— 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*]: $1 < x ==> (x\ powr\ a \leq x\ powr\ b) = (a \leq b)$
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*]:
$[|\ 0 < a;\ a \neq 1;\ 0 < x\ |] ==> a\ powr\ (log\ a\ x) = x$
by (*simp add*: *powr-def log-def*)

lemma *log-powr-cancel* [*simp*]: $[|\ 0 < a;\ a \neq 1\ |] ==> log\ a\ (a\ powr\ y) = y$
by (*simp add*: *log-def powr-def*)

lemma *log-mult*:
$[|\ 0 < a;\ a \neq 1;\ 0 < x;\ 0 < y\ |]$
$==> log\ a\ (x * y) = log\ a\ x + log\ a\ y$
by (*simp add*: *log-def ln-mult divide-inverse left-distrib*)

lemma *log-eq-div-ln-mult-log*:
$[|\ 0 < a;\ a \neq 1;\ 0 < b;\ b \neq 1;\ 0 < x\ |]$
$==> log\ a\ x = (ln\ b/ln\ a) * log\ b\ x$
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

ML
⟪
val powr-one-eq-one = thm powr-one-eq-one;
val powr-zero-eq-one = thm powr-zero-eq-one;
val powr-one-gt-zero-iff = thm powr-one-gt-zero-iff;
val powr-mult = thm powr-mult;
val powr-gt-zero = thm powr-gt-zero;
val powr-not-zero = thm powr-not-zero;
val powr-divide = thm powr-divide;
val powr-add = thm powr-add;
val powr-powr = thm powr-powr;
val powr-powr-swap = thm powr-powr-swap;
val powr-minus = thm powr-minus;
val powr-minus-divide = thm powr-minus-divide;
val powr-less-mono = thm powr-less-mono;
val powr-less-cancel = thm powr-less-cancel;
val powr-less-cancel-iff = thm powr-less-cancel-iff;
val powr-le-cancel-iff = thm powr-le-cancel-iff;
val log-ln = thm log-ln;
val powr-log-cancel = thm powr-log-cancel;
val log-powr-cancel = thm log-powr-cancel;
val log-mult = thm log-mult;
val log-eq-div-ln-mult-log = thm log-eq-div-ln-mult-log;
val log-base-10-eq1 = thm log-base-10-eq1;
val log-base-10-eq2 = thm log-base-10-eq2;
val log-one = thm log-one;
val log-eq-one = thm log-eq-one;
val log-inverse = thm log-inverse;

val log-divide = thm log-divide;
val log-less-cancel-iff = thm log-less-cancel-iff;
val log-le-cancel-iff = thm log-le-cancel-iff;
⟩⟩

end

30 MacLaurin: MacLaurin Series

theory *MacLaurin*
imports *Log*
begin

30.1 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.

ML
⟨⟨
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*;

val deriv-rulesI = [*DERIV-Id*,*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-Id,*DERIV-const*,*DERIV-cos*];

```
val deriv-tac =
  SUBGOAL (fn (prem,i) =>
   (resolve-tac deriv-rulesI i) ORELSE
    ((rtac (read-instantiate [(f,get-fun-name prem)]
                  DERIV-chain2) i) handle DERIV-name => no-tac));;

val DERIV-tac = ALLGOALS(fn i => REPEAT(deriv-tac i));
⟫

lemma Maclaurin-lemma2:
     [| ∀m t. m < n ∧ 0≤t ∧ t≤h ⟶ DERIV (diff m) t :> diff (Suc m) t;
        n = Suc k;
       difg =
       (λm 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:
```

```
    [|∀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 (subgoal-tac ∀ta. 0 ≤ ta & ta ≤ t --> (difg (Suc n)) differentiable ta)
apply (simp add: differentiable-def)
apply (blast dest!: DERIV-isCont)
apply (simp add: differentiable-def, clarify)
apply (rule-tac x = difg (Suc (Suc n)) ta in exI)
apply force
apply (simp add: differentiable-def, clarify)
apply (rule-tac x = difg (Suc (Suc n)) x in exI)
apply force
done

lemma Maclaurin:
  [| 0 < h; 0 < n; 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 (*subgoal-tac* ∀*t*. 0 ≤ *t* & *t* ≤ *h* −−> *g differentiable t*)
apply (*simp add*: *differentiable-def*)
apply (*blast dest*: *DERIV-isCont*)
apply (*simp add*: *differentiable-def*, *clarify*)
apply (*rule-tac x* = *difg* (*Suc* 0) *t* **in** *exI*)
apply *force*
apply (*simp add*: *differentiable-def*, *clarify*)

apply (*rule-tac x = difg (Suc 0) x* **in** *exI*)
apply *force*
apply *safe*
apply *force*
apply (*frule Maclaurin-lemma3*, *assumption+*, *safe*)
apply (*rule-tac x = ta* **in** *exI*, *force*)
done

lemma *Maclaurin-objl*:
0 < h & 0 < n & 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; 0 < n; 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-Id)
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 & 0 < n & 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)
```

30.2 More Convenient "Bidirectional" Version.

```
lemma Maclaurin-bi-le-lemma [rule-format]:
     0 < n −→
       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; 0 < n
 |] ==> ∃ 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 & 0 < n
--> (∃ 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)
 ==> 0 < n -->
     (∑ 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 (*insert linorder-le-less-linear* [*of n 0*])
apply (*erule disjE*, *force*)
apply (*case-tac x = 0*)
apply (*frule-tac diff = diff* **and** *n = n* **in** *Maclaurin-zero*, *assumption*)
apply (*drule gr-implies-not0* [*THEN 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*)

30.3 Version for Exponential Function

lemma *Maclaurin-exp-lt*: [| *x* ~= *0*; *0* < *n* |]
 ==> (∃ *t*. *0* < *abs t* &
  abs t < *abs x* &
  exp x = (∑ *m*=*0*..<*n*. (*x* ^ *m*) / *real* (*fact m*)) +
   (*exp t* / *real* (*fact n*)) * *x* ^ *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*:
 ∃ *t*. *abs t* ≤ *abs x* &
  exp x = (∑ *m*=*0*..<*n*. (*x* ^ *m*) / *real* (*fact m*)) +
   (*exp t* / *real* (*fact n*)) * *x* ^ *n*
by (*cut-tac diff* = %*n*. *exp* **and** *f* = *exp* **and** *x* = *x* **and** *n* = *n* **in** *Maclaurin-all-le-objl*, *auto*)

30.4 Version for Sine Function

lemma *MVT2*:
 [| *a* < *b*; ∀ *x*. *a* ≤ *x* & *x* ≤ *b* --> *DERIV f x* :> *f'*(*x*) |]
 ==> ∃ *z*. *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 | m mod 4 = 1 | m mod 4 = 2 | m mod 4 = (3::nat)
by (*case-tac m mod 4*, *auto*, *arith*)

lemma *Suc-Suc-mult-two-diff-two* [*rule-format*, *simp*]:
*0 < n --> Suc (Suc (2 * n - 2)) = 2*n*
by (*induct n*, *auto*)

lemma *lemma-Suc-Suc-4n-diff-2* [*rule-format*, *simp*]:
*0 < n --> Suc (Suc (4*n - 2)) = 4*n*
by (*induct n*, *auto*)

lemma *Suc-mult-two-diff-one* [*rule-format*, *simp*]:
*0 < n --> 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*)
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*:
[| *0 < n*; *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 ==>
∃ *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** *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

30.5 Maclaurin Expansion for Cosine Function

lemma *sumr-cos-zero-one* [*simp*]:
(∑ *m*=*0*..<(*Suc n*).
(*if even m then* (− *1*) ^ (*m div 2*)/(*real* (*fact m*)) *else 0*) ∗ *0* ^ *m*) = *1*
by (*induct n*, *auto*)

lemma *Maclaurin-cos-expansion*:
 ∃ *t*. *abs t* ≤ *abs x* &
 cos x =
 (∑ *m*=*0*..<*n*. (*if even m*
 then (− *1*) ^ (*m div 2*)/(*real* (*fact m*))
 else 0) ∗
 x ^ *m*)
 + ((*cos*(*t* + *1*/*2* ∗ *real* (*n*) ∗*pi*) / *real* (*fact n*)) ∗ *x* ^ *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*; *0* < *n* |] ==>
 ∃ *t*. *0* < *t* & *t* < *x* &
 cos x =
 (∑ *m*=*0*..<*n*. (*if even m*
 then (− *1*) ^ (*m div 2*)/(*real* (*fact m*))
 else 0) ∗
 x ^ *m*)
 + ((*cos*(*t* + *1*/*2* ∗ *real* (*n*) ∗*pi*) / *real* (*fact n*)) ∗ *x* ^ *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*; *0* < *n* |] ==>
 ∃ *t*. *x* < *t* & *t* < *0* &
 cos x =
 (∑ *m*=*0*..<*n*. (*if even m*
 then (− *1*) ^ (*m div 2*)/(*real* (*fact m*))

```
                    else 0) *
                    x ^ m)
     + ((cos(t + 1/2 * real (n) *pi) / real (fact n)) * x ^ 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 ≤ (v::real) |] ==> |(x + u) - y| ≤ 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]
  show ?thesis
    apply (cut-tac f=sin and n=n and x=x and
      diff = %n 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)
      in Maclaurin-all-le-objl)
    apply safe
    apply simp
    apply (subst (1 2 3) mod-Suc-eq-Suc-mod)
    apply (cut-tac m=m in mod-exhaust-less-4, safe, simp+)
    apply (rule DERIV-minus, simp+)
    apply (rule lemma-DERIV-subst, rule DERIV-minus, rule DERIV-cos, simp)
    apply (erule ssubst)
    apply (rule sin-bound-lemma)
    apply (rule setsum-cong[OF refl])
    apply (rule-tac f = %u. u * (x^xa) in arg-cong)
    apply (subst even-even-mod-4-iff)
    apply (cut-tac m=xa in mod-exhaust-less-4, simp, safe)
    apply (simp-all add:even-num-iff)
    apply (drule lemma-even-mod-4-div-2[simplified])
```

apply(*simp add*: *numeral-2-eq-2 divide-inverse*)
apply (*drule lemma-odd-mod-4-div-2*)
apply (*simp add*: *numeral-2-eq-2 divide-inverse*)
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

31 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*: *0 < n 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*. *c < t & t < b &*
f b = setsum (%*m*. (*diff m c* / *real* (*fact m*)) ∗ (*b* − *c*) ^ *m*) {*0..<n*} +
(*diff n t* / *real* (*fact n*)) ∗ (*b* − *c*) ^ *n*
proof −
from *INTERV* **have** *0 < b−c* **by** *arith*
moreover
from *INIT* **have** *0<n* ((λ*m x*. *diff m* (*x* + *c*)) *0*) = (λ*x*. *f* (*x* + *c*)) **by** *auto*
moreover
have *ALL m t*. *m < n & 0 <= t & t <= b − c −−> 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-Id* **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) ^
m) +
      diff n (t + c) / real (fact n) * (b − c) ^ n
    by (rule Maclaurin)
  show ?thesis
  proof −
    from EX obtain x where
      X: 0 < x & x < b − c &
       f (b − c + c) = (∑ m = 0..<n. diff m (0 + c) / real (fact m) * (b − c) ^
m) +
          diff n (x + c) / real (fact n) * (b − c) ^ n ..
    let ?H = x + c
    from X have c<?H & ?H<b ∧ f b = (∑ m = 0..<n. diff m c / real (fact m)
* (b − c) ^ m) +
      diff n ?H / real (fact n) * (b − c) ^ n
      by fastsimp
    thus ?thesis by fastsimp
  qed
qed

lemma taylor-down:
  assumes INIT: 0 < n 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 0<n ((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-Id 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: 0 < n 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 a ≤ x x ≤ b x ≠ c
  shows ∃ 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)^m) {0..<n} +
      (diff n t / real (fact n)) * (x - c)^n
proof (cases x<c)
  case True
  note INIT
  moreover from DERIV and INTERV
  have ∀ m t. m < n ∧ x ≤ t ∧ t ≤ b ⟶ DERIV (diff m) t :> diff (Suc m) t
    by fastsimp
  moreover note True
  moreover from INTERV have c ≤ b by simp
  ultimately have EX: ∃ t>x. t < c ∧ f x =
    (∑ m = 0..<n. diff m c / real (fact m) * (x - c) ^ m) +
      diff n t / real (fact n) * (x - c) ^ n
    by (rule taylor-down)
  with True show ?thesis by simp
next
  case False
  note INIT
  moreover from DERIV and INTERV
  have ∀ m t. m < n ∧ a ≤ t ∧ t ≤ x ⟶ DERIV (diff m) t :> diff (Suc m) t
    by fastsimp
  moreover from INTERV have a ≤ 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

32 Integration: Theory of Integration

theory *Integration*
imports *MacLaurin*
begin

We follow John Harrison in formalizing the Gauge integral.

constdefs

— Partitions and tagged partitions etc.

$partition :: [(real{*}real),nat => real] => bool$
$partition == \%(a,b)\ D.\ D\ 0 = a\ \&$
$\quad (\exists N.\ (\forall n < N.\ D(n) < D(Suc\ n))\ \&$
$\quad\quad (\forall n \geq N.\ D(n) = b))$

$psize :: (nat => real) => nat$
$psize\ D == SOME\ N.\ (\forall n < N.\ D(n) < D(Suc\ n))\ \&$
$\quad (\forall n \geq N.\ D(n) = D(N))$

$tpart :: [(real{*}real),((nat => real){*}(nat =>real))] => bool$
$tpart == \%(a,b)\ (D,p).\ partition(a,b)\ D\ \&$
$\quad (\forall n.\ D(n) \leq p(n)\ \&\ p(n) \leq D(Suc\ n))$

— Gauges and gauge-fine divisions

$gauge :: [real => bool,\ real => real] => bool$
$gauge\ E\ g == \forall x.\ E\ x\ --{>}\ 0 < g(x)$

$fine :: [real => real,\ ((nat => real){*}(nat => real))] => bool$
$fine == \%\ g\ (D,p).\ \forall n.\ n < (psize\ D)\ --{>}\ D(Suc\ n) - D(n) < g(p\ n)$

— Riemann sum

$rsum :: [((nat{=>}real){*}(nat{=>}real)),real{=>}real] => real$
$rsum == \%(D,p)\ f.\ \sum n{=}0..{<}psize(D).\ f(p\ n) * (D(Suc\ n) - D(n))$

— Gauge integrability (definite)

$Integral :: [(real*real),real=>real,real] => bool$
$Integral == \%(a,b)\ f\ k.\ \forall\, e > 0.$
$\quad (\exists\, g.\ gauge(\%x.\ a \leq x\ \&\ x \leq b)\ g\ \&$
$\quad (\forall\, D\ p.\ tpart(a,b)\ (D,p)\ \&\ fine(g)(D,p) --> |rsum(D,p)\ f\ -\ k| < e))$

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*]:
$\quad a \leq b ==> partition(a,b)(\%n.\ if\ n = 0\ then\ a\ else\ b)$
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*:
$\quad (partition(a,b)\ D) =$
$\quad ((D\ 0 = a)\ \&$
$\quad (\forall\, n < psize\ D.\ D\ n < D(Suc\ n))\ \&$
$\quad (\forall\, n \geq psize\ D.\ D\ n = b))$
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** $P = \%n.\ Na \leq n \longrightarrow D\ n = D\ Na$ **in** *spec*, *auto*)
done

lemma *partition-rhs*: $partition(a,b)\ D ==> (D(psize\ D) = b)$
by (*simp add*: *partition*)

lemma *partition-rhs2*: $[|partition(a,b)\ D;\ psize\ D \leq n\ |] ==> (D\ n = b)$

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*, *assumption*)
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* ≤ *b*; *gauge*(%*x*. *a* ≤ *x* & *x* ≤ *b*) *g* |]
==> ∃ *D p*. *tpart*(*a*,*b*) (*D*,*p*) & *fine g* (*D*,*p*)
apply (*cut-tac P* = %(*u*,*v*). *a* ≤ *u* & *v* ≤ *b* −−>
(∃ *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* ≤ *x* & *x* ≤ *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)
apply *arith*
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-interval-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-mult* [*symmetric*] *gauge-def abs-interval-iff*
right-diff-distrib [*symmetric*] *partition tpart-def*)
done

lemma *Integral-mult*:
[| *a ≤ 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-mult*[*symmetric*] *mult-assoc*)
apply (*rule-tac a2 = c* **in** *abs-ge-zero* [*THEN real-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*: ∀ *x. P(x) −−> (*∃ *y. Q x y) ==>* ∃ *f. (*∀ *x. P(x) −−> Q x (f x))*
by (*insert bchoice* [*of Collect P Q*], *simp*)

lemma *strad1*:
⟦∀ *xa*::*real. xa ≠ x* ∧ *|xa + − x| < s* ⟶
|(f xa − f x) / (xa − x) + − f′ x| ∗ 2 < e;
0 < e; *a ≤ x*; *x ≤ b*; *0 < s*⟧
⟹ ∀ *z. |z − x| < s −−>|f z − f x − f′ x ∗ (z − x)| ∗ 2 ≤ 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*:
$[|\ \forall x.\ a \leq x\ \&\ x \leq b\ \text{--}\text{>}\ DERIV\ f\ x\ :>\ f'(x);\ 0 < e\ |]$
$\text{==}\text{>}\ \exists g.\ gauge(\%x.\ a \leq x\ \&\ x \leq b)\ g\ \&$
$(\forall x\ u\ v.\ a \leq u\ \&\ u \leq x\ \&\ x \leq v\ \&\ v \leq b\ \&\ (v - u) < g(x)$
$\text{--}\text{>}\ |(f(v) - f(u)) - (f'(x) * (v - u))| \leq e * (v - u))$
apply (*simp add*: *gauge-def*)
apply (*subgoal-tac* $\forall x.\ a \leq x\ \&\ x \leq b\ \text{--}\text{>}$
$(\exists d > 0.\ \forall u\ v.\ u \leq x\ \&\ x \leq v\ \&\ (v - u) < d\ \text{--}\text{>}$
$|(f\ (v) - f\ (u)) - (f'\,(x) * (v - u))| \leq e * (v - u)))$
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*, *arith*)
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* [!] $V = \forall x'.\ x' \sim= x\ \&\ |x' + - x| < s\ \text{--}\text{>}$ *?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-interval-iff*)
apply (*simp add*: *right-diff-distrib*, *arith*)
done

lemma *FTC1*: $[|a \leq b;\ \forall x.\ a \leq x\ \&\ x \leq b\ \text{--}\text{>}\ DERIV\ f\ x\ :>\ f'(x)\ |]$
$\text{==}\text{>}\ Integral(a,b)\ f'\ (f(b) - f(a))$
apply (*drule order-le-imp-less-or-eq*, *auto*)
apply (*auto simp add*: *Integral-def*)

apply (*rule ccontr*)
apply (*subgoal-tac* $\forall e > 0.\ \exists g.$ *gauge* (%$x.\ a \le x$ & $x \le b$) g & ($\forall D\ p.$ *tpart* (a, b) (D, p) & *fine* g (D, p) −−> $|$*rsum* (D, p) $f' - (f\ b - f\ a)| \le e$))
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* ($\sum$ *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* = ($\sum$ *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-mult* [*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*:
[| $a \le b$; $b \le c$; *Integral*(*a*,*b*) f' *k1*; *Integral*(*b*,*c*) f' *k2*;
$\forall x.\ a \le x$ & $x \le c$ −−> *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 ==> ~ $(\exists n.\ c < D(n))$
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* $D\ n \leq c$ *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

32.1 Lemmas for Additivity Theorem of Gauge Integral

lemma *lemma-additivity1*:
    [| $a \leq D\ n$; $D\ n < b$; *partition*(a,b) D |] ==> $n <$ *psize* D
by (*auto simp add*: *partition linorder-not-less* [*symmetric*])

lemma *lemma-additivity2*: [| $a \leq D\ n$; *partition*$(a,D\ n)$ D |] ==> *psize* $D \leq n$
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 m* = *n* **and** *n* = *psize D* **in** *less-linear*, *auto*)
apply (*cut-tac m* = *m* **and** *n* = *n* **in** *less-linear*)
apply (*cut-tac m* = *m* **and** *n* = *psize D* **in** *less-linear*)
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 (*rule ccontr*, *drule leI*, *drule le-imp-less-or-eq*)
apply (*auto dest*: *partition-eq-bound*)
apply (*rule ccontr*, *drule leI*, *drule le-imp-less-or-eq*)
apply (*frule partition-eq-bound*, *assumption*)
apply (*drule-tac m* = *m* **in** *partition-eq-bound*, *auto*)
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 m* = *na* **and** *n* = *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*, *arith*)
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*
apply (*cut-tac m = n* **and** *n = Suc na* **in** *less-linear*, *auto*)
apply (*drule-tac* [*2*] *n = n* **in** *partition-lt-gen*, *auto*)
apply (*cut-tac m = psize D* **and** *n = na* **in** *less-linear*)
apply (*auto simp add*: *partition-rhs2 less-Suc-eq*)
apply (*drule-tac n = na* **in** *partition-lt-gen*, *auto*)
done

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 ≤ na ⟶ D na = D n* **in** *spec*, *auto*)
done

lemma *better-lemma-psize-right-eq*:
partition(a,b) D ==> psize (%x. D (x + n)) ≤ 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* |] ==> *psize (%x. D (x + n)) ≤ 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 ≤ 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 ==> psize (%x. D (x + n)) ≤ 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* ≤ *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* |] ==> *psize* (%*x*. *D* (*x* + *n*)) ≤ *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*:
[| $a \leq D\ n$; *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*)
apply (*drule-tac* $x = na + n$ **in** *spec*, *arith*)+
done

lemma *fine-right1*:
[| $a \leq D\ n$; *tpart* (a, b) (D, p); *gauge* $(\%x.\ D\ n \leq x\ \&\ x \leq b)$ *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*, *arith*)
apply (*simp add*: *tpart-def*, *safe*)
apply (*subgoal-tac* $D\ n \leq p\ (na + n)$)
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*], *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*:
[| $a \leq b$; *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*, *arith*)
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*:
[| ∀ *x*. *a* ≤ *x* & *x* ≤ *b* −−> *f x* ≤ *g x*;
tpart(*a*,*b*) (*D*,*p*)
|] ==> *rsum*(*D*,*p*) *f* ≤ *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* ≤ *b*;
∀ *x*. *a* ≤ *x* & *x* ≤ *b* −−> *f*(*x*) ≤ *g*(*x*);
Integral(*a*,*b*) *f k1*; *Integral*(*a*,*b*) *g k2*
|] ==> *k1* ≤ *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, *arith*)
done

lemma *Integral-imp-Cauchy*:
$(\exists\, k.\ Integral(a,b)\ f\ k) ==>$
$(\forall\, e > 0.\ \exists\, g.\ gauge\ (\%x.\ a \le x\ \&\ x \le b)\ g\ \&$
$(\forall\, D1\ D2\ p1\ p2.$
$tpart(a,b)\ (D1,\ p1)\ \&\ fine\ g\ (D1,p1)\ \&$
$tpart(a,b)\ (D2,\ p2)\ \&\ fine\ g\ (D2,p2)\ --> $
$|rsum(D1,p1)\ f\ -\ rsum(D2,p2)\ f| < e))$
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, *arith*)
done

lemma *Cauchy-iff2*:
$Cauchy\ X =$
$(\forall\, j.\ (\exists\, M.\ \forall\, m \ge M.\ \forall\, n \ge M.\ |X\ m\ +\ -\ X\ n| < inverse(real\ (Suc\ j))))$
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*:
$[|\ a \le b;\ \forall\, n.\ gauge\ (\%x.\ a \le x\ \&\ x \le b)\ (fa\ n)\ |]$
$==> \forall\, n.\ \exists\, D\ p.\ tpart\ (a,\ b)\ (D,\ p)\ \&\ fine\ (fa\ n)\ (D,\ p)$
by (*blast dest*: *partition-exists*)

lemma *monotonic-anti-derivative*:
$[|\ a \le b;\ \forall\, c.\ a \le c\ \&\ c \le b\ --> f'\ c \le g'\ c;$

$\forall x.\ DERIV\ f\ x :> f'\ x; \forall x.\ DERIV\ g\ x :> g'\ x\ |]$
$==> f\ b - f\ a \leq g\ b - g\ a$
apply (*rule Integral-le*, *assumption*)
apply (*auto intro*: *FTC1*)
done

end

33 HTranscendental: Nonstandard Extensions of Transcendental Functions

theory *HTranscendental*
imports *Transcendental Integration*
begin

really belongs in Transcendental

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≠0*
hence *pos*: *0<x* **using** *nneg* **by** *arith*
show *?thesis*
proof (*rule right-inverse-eq* [*THEN iffD1*, *THEN sym*])
show *sqrt x / x ≠ 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

constdefs

exphr :: *real => hypreal*
— define exponential function using standard part
*exphr x == st(sumhr (0, whn, %n. inverse(real (fact n)) * (x ^ n)))*

sinhr :: *real => hypreal*
sinhr x == st(sumhr (0, whn, %n. (if even(n) then 0 else
*((−1) ^ ((n − 1) div 2))/(real (fact n))) * (x ^ n)))*

coshr :: *real => hypreal*
coshr x == st(sumhr (0, whn, %n. (if even(n) then
*((−1) ^ (n div 2))/(real (fact n)) else 0) * x ^ n))*

33.1 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*: $((*f* sqrt)(x)\ \hat{}\ 2 = x) = (0 \leq x)$
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*: $!!x.\ 0 < x ==> (*f* sqrt)\ (x)\ \hat{}\ 2 = x$
by (*transfer*, *simp*)

lemma *hypreal-sqrt-pow2-gt-zero*: $0 < x ==> 0 < (*f* sqrt)\ (x)\ \hat{}\ 2$
by (*frule hypreal-sqrt-gt-zero-pow2*, *auto*)

lemma *hypreal-sqrt-not-zero*: $0 < x ==> (*f* sqrt)\ (x) \neq 0$
apply (*frule hypreal-sqrt-pow2-gt-zero*)
apply (*auto simp add*: *numeral-2-eq-2*)
done

lemma *hypreal-inverse-sqrt-pow2*:
$0 < x ==> inverse\ ((*f* sqrt)(x))\ \hat{}\ 2 = inverse\ x$
apply (*cut-tac n1* $= 2$ **and** *a1* $= (*f* sqrt)\ x$ **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*:
$[|0 \leq x;\ 0 \leq y\ |] ==>$
$(*f* sqrt)(x*y) = (*f* sqrt)(x) * (*f* sqrt)(y)$
by (*auto intro*: *hypreal-sqrt-mult-distrib simp add*: *order-le-less*)

lemma *hypreal-sqrt-approx-zero* [*simp*]:
$0 < x ==> ((*f* sqrt)(x)\ @= 0) = (x\ @= 0)$
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 simp add*: *zero-le-square*)
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 simp add*: *zero-le-square*)
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*:
  $[|\ x \in HFinite;\ 0 \leq x\ |] \Longrightarrow st((*f* sqrt)\ x) = (*f* sqrt)(st\ x)$
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*, *simp*)

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 simp add*: *zero-le-square*)
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* ≤ *x*; (∗*f*∗ *sqrt*) *x* ∈ *Infinitesimal* |] ==> *x* ∈ *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 ≤ x ==> ((*f* sqrt) x ∈ Infinitesimal) = (x ∈ Infinitesimal)*
by (*blast intro*: *Infinitesimal-hypreal-sqrt-imp-Infinitesimal Infinitesimal-hypreal-sqrt*)

lemma *Infinitesimal-sqrt-sum-squares* [*simp*]:
*((*f* sqrt)(x*x + y*y) ∈ Infinitesimal) = (x*x + y*y ∈ Infinitesimal)*
apply (*rule Infinitesimal-hypreal-sqrt-iff*)
apply (*rule add-nonneg-nonneg*)
apply (*auto simp add*: *zero-le-square*)
done

lemma *HInfinite-hypreal-sqrt*:
*[| 0 ≤ x; x ∈ HInfinite |] ==> (*f* sqrt) x ∈ 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 ≤ x; (*f* sqrt) x ∈ HInfinite |] ==> x ∈ 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 ≤ x ==> ((*f* sqrt) x ∈ HInfinite) = (x ∈ HInfinite)*
by (*blast intro*: *HInfinite-hypreal-sqrt HInfinite-hypreal-sqrt-imp-HInfinite*)

lemma *HInfinite-sqrt-sum-squares* [*simp*]:
*((*f* sqrt)(x*x + y*y) ∈ HInfinite) = (x*x + y*y ∈ HInfinite)*
apply (*rule HInfinite-hypreal-sqrt-iff*)
apply (*rule add-nonneg-nonneg*)
apply (*auto simp add*: *zero-le-square*)
done

lemma *HFinite-exp* [*simp*]:
*sumhr (0, whn, %n. inverse (real (fact n)) * x ^ n) ∈ HFinite*
by (*auto intro*!: *NSBseq-HFinite-hypreal NSconvergent-NSBseq*
simp add: *starfunNat-sumr* [*symmetric*] *starfun hypnat-omega-def*
convergent-NSconvergent-iff [*symmetric*]
summable-convergent-sumr-iff [*symmetric*] *summable-exp*)

lemma *exphr-zero* [*simp*]: *exphr 0 = 1*
apply (*simp add*: *exphr-def sumhr-split-add*
[*OF hypnat-one-less-hypnat-omega, symmetric*])

apply (*simp add*: *sumhr star-n-zero-num starfun star-n-one-num star-n-add*
hypnat-omega-def
del: *OrderedGroup.add-0*)
apply (*simp add*: *star-n-one-num* [*symmetric*])
done

lemma *coshr-zero* [*simp*]: *coshr 0 = 1*
apply (*simp add*: *coshr-def sumhr-split-add*
[*OF hypnat-one-less-hypnat-omega*, *symmetric*])
apply (*simp add*: *sumhr star-n-zero-num star-n-one-num hypnat-omega-def*)
apply (*simp add*: *star-n-one-num* [*symmetric*] *star-n-zero-num* [*symmetric*])
done

lemma *STAR-exp-zero-approx-one* [*simp*]: (*∗f∗ exp*) *0* @= *1*
by (*simp add*: *star-n-zero-num star-n-one-num starfun*)

lemma *STAR-exp-Infinitesimal*: *x* ∈ *Infinitesimal* ==> (*∗f∗ exp*) *x* @= *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*: *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-hypreal-of-real* [*THEN subst*])
apply (*rule approx-st-eq*, *auto*)
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*simp only*: *mem-infmal-iff* [*symmetric*])
apply (*auto simp add*: *mem-infmal-iff* [*symmetric*] *star-of-def star-n-zero-num*
hypnat-omega-def sumhr star-n-minus star-n-add)
apply (*rule NSLIMSEQ-zero-Infinitesimal-hypreal*)
apply (*insert exp-converges* [*of x*])
apply (*simp add*: *sums-def*)
apply (*drule LIMSEQ-const* [*THEN* [*2*] *LIMSEQ-add*, **where** *b* = − *exp x*])
apply (*simp add*: *LIMSEQ-NSLIMSEQ-iff*)
done

lemma *starfun-exp-ge-add-one-self* [*simp*]: !!*x*. *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* ∈ *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* ∈ *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*. *0* < *x* ==> *1* < (∗*f*∗ *exp*) *x*
by (*transfer*, *simp*)

lemma *starfun-ln-exp* [*simp*]: !!*x*. (∗*f*∗ *ln*) ((∗*f*∗ *exp*) *x*) = *x*
by (*transfer*, *simp*)

lemma *starfun-exp-ln-iff* [*simp*]: !!*x*. ((∗*f*∗ *exp*)((∗*f*∗ *ln*) *x*) = *x*) = (*0* < *x*)
by (*transfer*, *simp*)

lemma *starfun-exp-ln-eq*: (∗*f*∗ *exp*) *u* = *x* ==> (∗*f*∗ *ln*) *x* = *u*
by *auto*

lemma *starfun-ln-less-self* [*simp*]: !!*x*. *0* < *x* ==> (∗*f*∗ *ln*) *x* < *x*
by (*transfer*, *simp*)

lemma *starfun-ln-ge-zero* [*simp*]: !!*x*. *1* ≤ *x* ==> *0* ≤ (∗*f*∗ *ln*) *x*
by (*transfer*, *simp*)

lemma *starfun-ln-gt-zero* [*simp*]: !!*x* *.1* < *x* ==> *0* < (∗*f*∗ *ln*) *x*
by (*transfer*, *simp*)

lemma *starfun-ln-not-eq-zero* [*simp*]: !!x. [| $0 < x$; $x \neq 1$ |] ==> ($*f*$ *ln*) $x \neq 0$
by (*transfer*, *simp*)

lemma *starfun-ln-HFinite*: [| $x \in$ *HFinite*; $1 \leq x$ |] ==> ($*f*$ *ln*) $x \in$ *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-exp-HFinite*: $x \in$ *HFinite* ==> ($*f*$ *exp*) $x \in$ *HFinite*
apply (*cases* x)
apply (*auto simp add*: *starfun HFinite-FreeUltrafilterNat-iff*)
apply (*rule bexI* [*OF - Rep-star-star-n*], *auto*)
apply (*rule-tac* x = *exp* u **in** *exI*)
apply (*ultra*, *arith*)
done

lemma *starfun-exp-add-HFinite-Infinitesimal-approx*:
[|$x \in$ *Infinitesimal*; $z \in$ *HFinite* |] ==> ($*f*$ *exp*) ($z + x$) @= ($*f*$ *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*) $x \in$ *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*, *simp*)

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*
apply (*auto intro*!: *NSBseq-HFinite-hypreal NSconvergent-NSBseq*
simp add: *starfunNat-sumr* [*symmetric*] *starfun hypnat-omega-def*
convergent-NSconvergent-iff [*symmetric*]
summable-convergent-sumr-iff [*symmetric*])
apply (*simp only*: *One-nat-def summable-sin*)
done

lemma *STAR-sin-zero* [*simp*]: ($*f*$ *sin*) 0 = 0
by (*transfer*, *simp*)

lemma *STAR-sin-Infinitesimal* [*simp*]: $x \in$ *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*
by (*auto intro*!: *NSBseq-HFinite-hypreal NSconvergent-NSBseq*
      simp add: *starfunNat-sumr* [*symmetric*] *starfun hypnat-omega-def*
            convergent-NSconvergent-iff [*symmetric*]
            summable-convergent-sumr-iff [*symmetric*] *summable-cos*)

lemma *STAR-cos-zero* [*simp*]: (∗*f*∗ *cos*) *0* = *1*
by (*simp add*: *starfun star-n-zero-num star-n-one-num*)

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*], *auto*)
done

lemma *STAR-tan-zero* [*simp*]: (∗*f*∗ *tan*) *0* = *0*
by (*simp add*: *starfun star-n-zero-num*)

lemma *STAR-tan-Infinitesimal*: *x* ∈ *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*:

$N \in HNatInfinite$
$\Longrightarrow hypreal\text{-}of\text{-}real\ a =$
$hypreal\text{-}of\text{-}hypnat\ N * (inverse(hypreal\text{-}of\text{-}hypnat\ N) * hypreal\text{-}of\text{-}real\ a)$
by (*simp add*: *mult-assoc* [*symmetric*] *HNatInfinite-not-eq-zero*)

lemma *STAR-sin-Infinitesimal-divide*:
$[|x \in Infinitesimal; x \neq 0 |] \Longrightarrow (*f* sin)\ x/x \mathbin{@=} 1$
apply (*cut-tac* $x = 0$ **in** *DERIV-sin*)
apply (*simp add*: *NSDERIV-DERIV-iff* [*symmetric*] *nsderiv-def*)
done

lemma *lemma-sin-pi*:
$n \in HNatInfinite$
$\Longrightarrow (*f* sin)\ (inverse\ (hypreal\text{-}of\text{-}hypnat\ n))/(inverse\ (hypreal\text{-}of\text{-}hypnat\ n)) \mathbin{@=} 1$
apply (*rule STAR-sin-Infinitesimal-divide*)
apply (*auto simp add*: *HNatInfinite-not-eq-zero*)
done

lemma *STAR-sin-inverse-HNatInfinite*:
$n \in HNatInfinite$
$\Longrightarrow (*f* sin)\ (inverse\ (hypreal\text{-}of\text{-}hypnat\ n)) * hypreal\text{-}of\text{-}hypnat\ n \mathbin{@=} 1$
apply (*frule lemma-sin-pi*)
apply (*simp add*: *divide-inverse*)
done

lemma *Infinitesimal-pi-divide-HNatInfinite*:
$N \in HNatInfinite$
$\Longrightarrow hypreal\text{-}of\text{-}real\ pi/(hypreal\text{-}of\text{-}hypnat\ N) \in Infinitesimal$
apply (*simp add*: *divide-inverse*)
apply (*auto intro*: *Infinitesimal-HFinite-mult2*)
done

lemma *pi-divide-HNatInfinite-not-zero* [*simp*]:
$N \in HNatInfinite \Longrightarrow hypreal\text{-}of\text{-}real\ pi/(hypreal\text{-}of\text{-}hypnat\ N) \neq 0$
by (*simp add*: *HNatInfinite-not-eq-zero*)

lemma *STAR-sin-pi-divide-HNatInfinite-approx-pi*:
$n \in HNatInfinite$
$\Longrightarrow (*f* sin)\ (hypreal\text{-}of\text{-}real\ pi/(hypreal\text{-}of\text{-}hypnat\ n)) * hypreal\text{-}of\text{-}hypnat\ n$
$\mathbin{@=} hypreal\text{-}of\text{-}real\ pi$
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*: *SReal-inverse simp add*: *divide-inverse mult-ac*)
done

lemma *STAR-sin-pi-divide-HNatInfinite-approx-pi2*:
n ∈ *HNatInfinite*
==> *hypreal-of-hypnat n* ∗
(∗*f*∗ *sin*) (*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 ∈ *HNatInfinite* ==>
(∗*f*∗ *sin*) ((∗*f*∗ (%*x*. *pi* / *real x*)) *N*) @=
hypreal-of-real pi/(*hypreal-of-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 ∈ *Infinitesimal* ==> (∗*f*∗ *cos*) *x* @= *1* − *x* ^ *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 ∈ *Infinitesimal* ==> (∗*f*∗ *cos*) *x* @= *1* − (*x* ^ *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

ML
⟪
val STAR-sqrt-zero = thm STAR-sqrt-zero;
val STAR-sqrt-one = thm STAR-sqrt-one;
val hypreal-sqrt-pow2-iff = thm hypreal-sqrt-pow2-iff;
val hypreal-sqrt-gt-zero-pow2 = thm hypreal-sqrt-gt-zero-pow2;
val hypreal-sqrt-pow2-gt-zero = thm hypreal-sqrt-pow2-gt-zero;
val hypreal-sqrt-not-zero = thm hypreal-sqrt-not-zero;
val hypreal-inverse-sqrt-pow2 = thm hypreal-inverse-sqrt-pow2;
val hypreal-sqrt-mult-distrib = thm hypreal-sqrt-mult-distrib;
val hypreal-sqrt-mult-distrib2 = thm hypreal-sqrt-mult-distrib2;
val hypreal-sqrt-approx-zero = thm hypreal-sqrt-approx-zero;
val hypreal-sqrt-approx-zero2 = thm hypreal-sqrt-approx-zero2;
val hypreal-sqrt-sum-squares = thm hypreal-sqrt-sum-squares;
val hypreal-sqrt-sum-squares2 = thm hypreal-sqrt-sum-squares2;
val hypreal-sqrt-gt-zero = thm hypreal-sqrt-gt-zero;
val hypreal-sqrt-ge-zero = thm hypreal-sqrt-ge-zero;
val hypreal-sqrt-hrabs = thm hypreal-sqrt-hrabs;
val hypreal-sqrt-hrabs2 = thm hypreal-sqrt-hrabs2;
val hypreal-sqrt-hyperpow-hrabs = thm hypreal-sqrt-hyperpow-hrabs;
val star-sqrt-HFinite = thm star-sqrt-HFinite;
val st-hypreal-sqrt = thm st-hypreal-sqrt;
val hypreal-sqrt-sum-squares-ge1 = thm hypreal-sqrt-sum-squares-ge1;
val HFinite-hypreal-sqrt = thm HFinite-hypreal-sqrt;
val HFinite-hypreal-sqrt-imp-HFinite = thm HFinite-hypreal-sqrt-imp-HFinite;
val HFinite-hypreal-sqrt-iff = thm HFinite-hypreal-sqrt-iff;
val HFinite-sqrt-sum-squares = thm HFinite-sqrt-sum-squares;
val Infinitesimal-hypreal-sqrt = thm Infinitesimal-hypreal-sqrt;

val Infinitesimal-hypreal-sqrt-imp-Infinitesimal = thm Infinitesimal-hypreal-sqrt-imp-Infinitesimal;
val Infinitesimal-hypreal-sqrt-iff = thm Infinitesimal-hypreal-sqrt-iff;
val Infinitesimal-sqrt-sum-squares = thm Infinitesimal-sqrt-sum-squares;
val HInfinite-hypreal-sqrt = thm HInfinite-hypreal-sqrt;
val HInfinite-hypreal-sqrt-imp-HInfinite = thm HInfinite-hypreal-sqrt-imp-HInfinite;
val HInfinite-hypreal-sqrt-iff = thm HInfinite-hypreal-sqrt-iff;
val HInfinite-sqrt-sum-squares = thm HInfinite-sqrt-sum-squares;
val HFinite-exp = thm HFinite-exp;
val exphr-zero = thm exphr-zero;
val coshr-zero = thm coshr-zero;
val STAR-exp-zero-approx-one = thm STAR-exp-zero-approx-one;
val STAR-exp-Infinitesimal = thm STAR-exp-Infinitesimal;
val STAR-exp-epsilon = thm STAR-exp-epsilon;
val STAR-exp-add = thm STAR-exp-add;
val exphr-hypreal-of-real-exp-eq = thm exphr-hypreal-of-real-exp-eq;
val starfun-exp-ge-add-one-self = thm starfun-exp-ge-add-one-self;
val starfun-exp-HInfinite = thm starfun-exp-HInfinite;
val starfun-exp-minus = thm starfun-exp-minus;
val starfun-exp-Infinitesimal = thm starfun-exp-Infinitesimal;
val starfun-exp-gt-one = thm starfun-exp-gt-one;
val starfun-ln-exp = thm starfun-ln-exp;
val starfun-exp-ln-iff = thm starfun-exp-ln-iff;
val starfun-exp-ln-eq = thm starfun-exp-ln-eq;
val starfun-ln-less-self = thm starfun-ln-less-self;
val starfun-ln-ge-zero = thm starfun-ln-ge-zero;
val starfun-ln-gt-zero = thm starfun-ln-gt-zero;
val starfun-ln-not-eq-zero = thm starfun-ln-not-eq-zero;
val starfun-ln-HFinite = thm starfun-ln-HFinite;
val starfun-ln-inverse = thm starfun-ln-inverse;
val starfun-exp-HFinite = thm starfun-exp-HFinite;
val starfun-exp-add-HFinite-Infinitesimal-approx = thm starfun-exp-add-HFinite-Infinitesimal-approx;
val starfun-ln-HInfinite = thm starfun-ln-HInfinite;
val starfun-exp-HInfinite-Infinitesimal-disj = thm starfun-exp-HInfinite-Infinitesimal-disj;
val starfun-ln-HFinite-not-Infinitesimal = thm starfun-ln-HFinite-not-Infinitesimal;
val starfun-ln-Infinitesimal-HInfinite = thm starfun-ln-Infinitesimal-HInfinite;
val starfun-ln-less-zero = thm starfun-ln-less-zero;
val starfun-ln-Infinitesimal-less-zero = thm starfun-ln-Infinitesimal-less-zero;
val starfun-ln-HInfinite-gt-zero = thm starfun-ln-HInfinite-gt-zero;
val HFinite-sin = thm HFinite-sin;
val STAR-sin-zero = thm STAR-sin-zero;
val STAR-sin-Infinitesimal = thm STAR-sin-Infinitesimal;
val HFinite-cos = thm HFinite-cos;
val STAR-cos-zero = thm STAR-cos-zero;
val STAR-cos-Infinitesimal = thm STAR-cos-Infinitesimal;
val STAR-tan-zero = thm STAR-tan-zero;
val STAR-tan-Infinitesimal = thm STAR-tan-Infinitesimal;
val STAR-sin-cos-Infinitesimal-mult = thm STAR-sin-cos-Infinitesimal-mult;
val HFinite-pi = thm HFinite-pi;
val lemma-split-hypreal-of-real = thm lemma-split-hypreal-of-real;

val STAR-sin-Infinitesimal-divide = thm STAR-sin-Infinitesimal-divide;
val lemma-sin-pi = thm lemma-sin-pi;
val STAR-sin-inverse-HNatInfinite = thm STAR-sin-inverse-HNatInfinite;
val Infinitesimal-pi-divide-HNatInfinite = thm Infinitesimal-pi-divide-HNatInfinite;
val pi-divide-HNatInfinite-not-zero = thm pi-divide-HNatInfinite-not-zero;
val STAR-sin-pi-divide-HNatInfinite-approx-pi = thm STAR-sin-pi-divide-HNatInfinite-approx-pi;
val STAR-sin-pi-divide-HNatInfinite-approx-pi2 = thm STAR-sin-pi-divide-HNatInfinite-approx-pi2;
val starfunNat-pi-divide-n-Infinitesimal = thm starfunNat-pi-divide-n-Infinitesimal;
val STAR-sin-pi-divide-n-approx = thm STAR-sin-pi-divide-n-approx;
val NSLIMSEQ-sin-pi = thm NSLIMSEQ-sin-pi;
val NSLIMSEQ-cos-one = thm NSLIMSEQ-cos-one;
val NSLIMSEQ-sin-cos-pi = thm NSLIMSEQ-sin-cos-pi;
val STAR-cos-Infinitesimal-approx = thm STAR-cos-Infinitesimal-approx;
val STAR-cos-Infinitesimal-approx2 = thm STAR-cos-Infinitesimal-approx2;
⟫

end

34 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*

constdefs

powhr :: [*hypreal*,*hypreal*] => *hypreal* (**infixr** *powhr 80*)
x powhr a == *starfun2* (*op powr*) *x a*

hlog :: [*hypreal*,*hypreal*] => *hypreal*
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. [| $0 < a$; $a \neq 1$ |] ==> $hlog\ a\ a = 1$
by (*transfer*, *rule log-eq-one*)

lemma *hlog-inverse*:
[| $0 < a$; $a \neq 1$; $0 < x$ |] ==> $hlog\ a\ (inverse\ x) = -\ hlog\ a\ x$
apply (*rule add-left-cancel* [*of hlog a x*, *THEN iffD1*])
apply (*simp add*: *hlog-mult* [*symmetric*])
done

lemma *hlog-divide*:
[| $0 < a$; $a \neq 1$; $0 < x$; $0 < y$|] ==> $hlog\ a\ (x/y) = hlog\ a\ x - hlog\ a\ y$
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*]:
[| $1 < a$; $0 < x$; $0 < y$ |] ==> $(hlog\ a\ x \leq hlog\ a\ y) = (x \leq y)$
by (*simp add*: *linorder-not-less* [*symmetric*])

ML
⟪
val powhr = thm powhr;
val powhr-one-eq-one = thm powhr-one-eq-one;
val powhr-mult = thm powhr-mult;
val powhr-gt-zero = thm powhr-gt-zero;
val powhr-not-zero = thm powhr-not-zero;
val powhr-divide = thm powhr-divide;
val powhr-add = thm powhr-add;
val powhr-powhr = thm powhr-powhr;
val powhr-powhr-swap = thm powhr-powhr-swap;
val powhr-minus = thm powhr-minus;
val powhr-minus-divide = thm powhr-minus-divide;
val powhr-less-mono = thm powhr-less-mono;
val powhr-less-cancel = thm powhr-less-cancel;
val powhr-less-cancel-iff = thm powhr-less-cancel-iff;
val powhr-le-cancel-iff = thm powhr-le-cancel-iff;
val hlog = thm hlog;
val hlog-starfun-ln = thm hlog-starfun-ln;
val powhr-hlog-cancel = thm powhr-hlog-cancel;
val hlog-powhr-cancel = thm hlog-powhr-cancel;

val hlog-mult = thm hlog-mult;
val hlog-as-starfun = thm hlog-as-starfun;
val hlog-eq-div-starfun-ln-mult-hlog = thm hlog-eq-div-starfun-ln-mult-hlog;
val powhr-as-starfun = thm powhr-as-starfun;
val HInfinite-powhr = thm HInfinite-powhr;
val hlog-hrabs-HInfinite-Infinitesimal = thm hlog-hrabs-HInfinite-Infinitesimal;
val hlog-HInfinite-as-starfun = thm hlog-HInfinite-as-starfun;
val hlog-one = thm hlog-one;
val hlog-eq-one = thm hlog-eq-one;
val hlog-inverse = thm hlog-inverse;
val hlog-divide = thm hlog-divide;
val hlog-less-cancel-iff = thm hlog-less-cancel-iff;
val hlog-le-cancel-iff = thm hlog-le-cancel-iff;
⟩⟩

end

theory *Hyperreal*
imports *Poly Taylor HLog*
begin

end

35 Complex: Complex Numbers: Rectangular and Polar Representations

theory *Complex*
imports *../Hyperreal/HLog*
begin

datatype *complex = Complex real real*

instance *complex* :: {*zero*, *one*, *plus*, *times*, *minus*, *inverse*, *power*} **..**

consts
  ii :: *complex* (i)

consts *Re* :: *complex => real*
primrec *Re* (*Complex x y*) = *x*

consts *Im* :: *complex => real*
primrec *Im* (*Complex x y*) = *y*

lemma *complex-surj* [*simp*]: *Complex* (*Re z*) (*Im z*) = *z*
  by (*induct z*) *simp*

constdefs

$cmod :: complex => real$
$cmod\ z == sqrt(Re(z)\ \hat{}\ 2 + Im(z)\ \hat{}\ 2)$

$complex\text{-}of\text{-}real :: real => complex$
$complex\text{-}of\text{-}real\ r == Complex\ r\ 0$

$cnj :: complex => complex$
$cnj\ z == Complex\ (Re\ z)\ (-Im\ z)$

$sgn :: complex => complex$
$sgn\ z == z\ /\ complex\text{-}of\text{-}real(cmod\ z)$

$arg :: complex => real$
$arg\ z == @a.\ Re(sgn\ z) = cos\ a\ \&\ Im(sgn\ z) = sin\ a\ \&\ -pi < a\ \&\ a \leq pi$

defs (**overloaded**)

complex-zero-def:
$0 == Complex\ 0\ 0$

complex-one-def:
$1 == Complex\ 1\ 0$

i-def: $ii == Complex\ 0\ 1$

complex-minus-def: $-\ z == Complex\ (-\ Re\ z)\ (-\ Im\ z)$

complex-inverse-def:
$inverse\ z ==$
$Complex\ (Re\ z\ /\ ((Re\ z)^2 + (Im\ z)^2))\ (-\ Im\ z\ /\ ((Re\ z)^2 + (Im\ z)^2))$

complex-add-def:
$z + w == Complex\ (Re\ z + Re\ w)\ (Im\ z + Im\ w)$

complex-diff-def:
$z - w == z + -\ (w::complex)$

complex-mult-def:
*z * w == Complex (Re z * Re w − Im z * Im w) (Re z * Im w + Im z * Re w)*

complex-divide-def: *w / (z::complex) == w * inverse z*

constdefs

cis :: real => complex
cis a == Complex (cos a) (sin a)

rcis :: [real, real] => complex
*rcis r a == complex-of-real r * cis a*

expi :: complex => complex
*expi z == complex-of-real(exp (Re z)) * cis (Im z)*

lemma *complex-equality [intro?]*: *Re z = Re w ==> Im z = Im w ==> z = w*
by (*induct z*, *induct w*) *simp*

lemma *Re [simp]*: *Re(Complex x y) = x*
by *simp*

lemma *Im [simp]*: *Im(Complex x y) = y*
by *simp*

lemma *complex-Re-Im-cancel-iff*: *(w=z) = (Re(w) = Re(z) & Im(w) = Im(z))*
by (*induct w*, *induct z*, *simp*)

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-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-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 *Re-complex-of-real* [*simp*]: $Re(complex\text{-}of\text{-}real\ z) = z$
by (*simp add*: *complex-of-real-def*)

lemma *Im-complex-of-real* [*simp*]: $Im(complex\text{-}of\text{-}real\ z) = 0$
by (*simp add*: *complex-of-real-def*)

35.1 Unary Minus

lemma *complex-minus* [*simp*]: $-\ (Complex\ x\ y) = Complex\ (-x)\ (-y)$
by (*simp add*: *complex-minus-def*)

lemma *complex-Re-minus* [*simp*]: $Re\ (-z) = -\ Re\ z$
by (*simp add*: *complex-minus-def*)

lemma *complex-Im-minus* [*simp*]: $Im\ (-z) = -\ Im\ z$
by (*simp add*: *complex-minus-def*)

35.2 Addition

lemma *complex-add* [*simp*]:
$Complex\ x1\ y1\ +\ Complex\ x2\ y2 = Complex\ (x1+x2)\ (y1+y2)$
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-add-commute*: $(u::complex)\ +\ v = v\ +\ u$
by (*simp add*: *complex-add-def add-commute*)

lemma *complex-add-assoc*: $((u::complex)\ +\ v)\ +\ w = u\ +\ (v\ +\ w)$
by (*simp add*: *complex-add-def add-assoc*)

lemma *complex-add-zero-left*: $(0::complex)\ +\ z = z$
by (*simp add*: *complex-add-def complex-zero-def*)

lemma *complex-add-zero-right*: $z\ +\ (0::complex) = z$
by (*simp add*: *complex-add-def complex-zero-def*)

lemma *complex-add-minus-left*: $-z\ +\ z = (0::complex)$
by (*simp add*: *complex-add-def complex-minus-def complex-zero-def*)

lemma *complex-diff*:
$Complex\ x1\ y1\ -\ Complex\ x2\ y2 = Complex\ (x1-x2)\ (y1-y2)$
by (*simp add*: *complex-add-def complex-minus-def 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*)

35.3 Multiplication

lemma *complex-mult* [*simp*]:
$Complex\ x1\ y1 * Complex\ x2\ y2 = Complex\ (x1*x2 - y1*y2)\ (x1*y2 + y1*x2)$
by (*simp add*: *complex-mult-def*)

lemma *complex-mult-commute*: $(w::complex) * z = z * w$
by (*simp add*: *complex-mult-def mult-commute add-commute*)

lemma *complex-mult-assoc*: $((u::complex) * v) * w = u * (v * w)$
by (*simp add*: *complex-mult-def mult-ac add-ac*
right-diff-distrib right-distrib left-diff-distrib left-distrib)

lemma *complex-mult-one-left*: $(1::complex) * z = z$
by (*simp add*: *complex-mult-def complex-one-def*)

lemma *complex-mult-one-right*: $z * (1::complex) = z$
by (*simp add*: *complex-mult-def complex-one-def*)

35.4 Inverse

lemma *complex-inverse* [*simp*]:
$inverse\ (Complex\ x\ y) = Complex\ (x/(x\ \hat{}\ 2 + y\ \hat{}\ 2))\ (-y/(x\ \hat{}\ 2 + y\ \hat{}\ 2))$
by (*simp add*: *complex-inverse-def*)

lemma *complex-mult-inv-left*: $z \neq (0::complex) ==> inverse(z) * z = 1$
apply (*induct z*)
apply (*rename-tac x y*)
apply (*auto simp add*: *times-divide-eq complex-mult complex-inverse*
complex-one-def complex-zero-def add-divide-distrib [*symmetric*]
power2-eq-square mult-ac)
apply (*simp-all add*: *real-sum-squares-not-zero real-sum-squares-not-zero2*)
done

35.5 The field of complex numbers

instance *complex* :: *field*
proof
fix *z u v w* :: *complex*
show $(u + v) + w = u + (v + w)$
by (*rule complex-add-assoc*)
show $z + w = w + z$

by (*rule complex-add-commute*)
show *0 + z = z*
by (*rule complex-add-zero-left*)
show *−z + z = 0*
by (*rule complex-add-minus-left*)
show *z − w = z + −w*
by (*simp add*: *complex-diff-def*)
show *(u * v) * w = u * (v * w)*
by (*rule complex-mult-assoc*)
show *z * w = w * z*
by (*rule complex-mult-commute*)
show *1 * z = z*
by (*rule complex-mult-one-left*)
show *0 ≠ (1::complex)*
by (*simp add*: *complex-zero-def complex-one-def*)
show *(u + v) * w = u * w + v * w*
by (*simp add*: *complex-mult-def complex-add-def left-distrib diff-minus add-ac*)
show *z / w = z * inverse w*
by (*simp add*: *complex-divide-def*)
assume *w ≠ 0*
thus *inverse w * w = 1*
by (*simp add*: *complex-mult-inv-left*)
qed

instance *complex* :: *division-by-zero*
proof
show *inverse 0 = (0::complex)*
by (*simp add*: *complex-inverse-def complex-zero-def*)
qed

35.6 Embedding Properties for *complex-of-real* Map

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*: *i-def 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*: *i-def complex-of-real-def*)

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

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

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

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

lemma *complex-of-real-inverse* [*simp*]:
complex-of-real(*inverse x*) = *inverse*(*complex-of-real x*)
apply (*case-tac x=0*, *simp*)
apply (*simp add*: *complex-of-real-def divide-inverse power2-eq-square*)
done

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

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

lemma *complex-of-real-mult* [*simp*]:
complex-of-real (*x* ∗ *y*) = *complex-of-real x* ∗ *complex-of-real y*
by (*simp add*: *complex-mult complex-of-real-def*)

lemma *complex-of-real-divide* [*simp*]:
complex-of-real(*x*/*y*) = *complex-of-real x* / *complex-of-real y*
apply (*simp add*: *complex-divide-def*)
apply (*case-tac y=0*, *simp*)
apply (*simp add*: *complex-of-real-mult complex-of-real-inverse*
divide-inverse)
done

lemma *complex-mod* [*simp*]: *cmod* (*Complex x y*) = *sqrt*(*x* ^ *2* + *y* ^ *2*)
by (*simp add*: *cmod-def*)

lemma *complex-mod-zero* [*simp*]: *cmod*(*0*) = *0*

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

lemma *complex-mod-one* [*simp*]: *cmod*(*1*) = *1*
by (*simp add*: *cmod-def power2-eq-square*)

lemma *complex-mod-complex-of-real* [*simp*]: *cmod*(*complex-of-real x*) = *abs x*
by (*simp add*: *complex-of-real-def power2-eq-square complex-mod*)

lemma *complex-of-real-abs*:
complex-of-real (*abs x*) = *complex-of-real*(*cmod*(*complex-of-real x*))
by *simp*

35.7 The Functions *Re* and *Im*

lemma *complex-Re-mult-eq*: *Re* (*w* ∗ *z*) = *Re w* ∗ *Re z* − *Im w* ∗ *Im z*
by (*induct z*, *induct w*, *simp add*: *complex-mult*)

lemma *complex-Im-mult-eq*: *Im* (*w* ∗ *z*) = *Re w* ∗ *Im z* + *Im w* ∗ *Re z*
by (*induct z*, *induct w*, *simp add*: *complex-mult*)

lemma *Re-i-times* [*simp*]: *Re*(*ii* ∗ *z*) = − *Im z*
by (*simp add*: *complex-Re-mult-eq*)

lemma *Re-times-i* [*simp*]: *Re*(*z* ∗ *ii*) = − *Im z*
by (*simp add*: *complex-Re-mult-eq*)

lemma *Im-i-times* [*simp*]: *Im*(*ii* ∗ *z*) = *Re z*
by (*simp add*: *complex-Im-mult-eq*)

lemma *Im-times-i* [*simp*]: *Im*(*z* ∗ *ii*) = *Re z*
by (*simp add*: *complex-Im-mult-eq*)

lemma *complex-Re-mult*: [| *Im w* = *0*; *Im z* = *0* |] ==> *Re*(*w* ∗ *z*) = *Re*(*w*) ∗ *Re*(*z*)
by (*simp add*: *complex-Re-mult-eq*)

lemma *complex-Re-mult-complex-of-real* [*simp*]:
Re (*z* ∗ *complex-of-real c*) = *Re*(*z*) ∗ *c*
by (*simp add*: *complex-Re-mult-eq*)

lemma *complex-Im-mult-complex-of-real* [*simp*]:
Im (*z* ∗ *complex-of-real c*) = *Im*(*z*) ∗ *c*
by (*simp add*: *complex-Im-mult-eq*)

lemma *complex-Re-mult-complex-of-real2* [*simp*]:
Re (*complex-of-real c* ∗ *z*) = *c* ∗ *Re*(*z*)
by (*simp add*: *complex-Re-mult-eq*)

lemma *complex-Im-mult-complex-of-real2* [*simp*]:

Im (*complex-of-real c* ∗ *z*) = *c* ∗ *Im*(*z*)
by (*simp add*: *complex-Im-mult-eq*)

35.8 Conjugation is an Automorphism

lemma *complex-cnj*: *cnj* (*Complex x y*) = *Complex x* (−*y*)
by (*simp add*: *cnj-def*)

lemma *complex-cnj-cancel-iff* [*simp*]: (*cnj x* = *cnj y*) = (*x* = *y*)
by (*simp add*: *cnj-def complex-Re-Im-cancel-iff*)

lemma *complex-cnj-cnj* [*simp*]: *cnj* (*cnj z*) = *z*
by (*simp add*: *cnj-def*)

lemma *complex-cnj-complex-of-real* [*simp*]:
cnj (*complex-of-real x*) = *complex-of-real x*
by (*simp add*: *complex-of-real-def complex-cnj*)

lemma *complex-mod-cnj* [*simp*]: *cmod* (*cnj z*) = *cmod z*
by (*induct z*, *simp add*: *complex-cnj complex-mod power2-eq-square*)

lemma *complex-cnj-minus*: *cnj* (−*z*) = − *cnj z*
by (*simp add*: *cnj-def complex-minus complex-Re-minus complex-Im-minus*)

lemma *complex-cnj-inverse*: *cnj*(*inverse z*) = *inverse*(*cnj z*)
by (*induct z*, *simp add*: *complex-cnj complex-inverse power2-eq-square*)

lemma *complex-cnj-add*: *cnj*(*w* + *z*) = *cnj*(*w*) + *cnj*(*z*)
by (*induct w*, *induct z*, *simp add*: *complex-cnj complex-add*)

lemma *complex-cnj-diff*: *cnj*(*w* − *z*) = *cnj*(*w*) − *cnj*(*z*)
by (*simp add*: *diff-minus complex-cnj-add complex-cnj-minus*)

lemma *complex-cnj-mult*: *cnj*(*w* ∗ *z*) = *cnj*(*w*) ∗ *cnj*(*z*)
by (*induct w*, *induct z*, *simp add*: *complex-cnj complex-mult*)

lemma *complex-cnj-divide*: *cnj*(*w* / *z*) = (*cnj w*)/(*cnj z*)
by (*simp add*: *complex-divide-def complex-cnj-mult complex-cnj-inverse*)

lemma *complex-cnj-one* [*simp*]: *cnj 1* = *1*
by (*simp add*: *cnj-def complex-one-def*)

lemma *complex-add-cnj*: *z* + *cnj z* = *complex-of-real* (*2* ∗ *Re*(*z*))
by (*induct z*, *simp add*: *complex-add complex-cnj complex-of-real-def*)

lemma *complex-diff-cnj*: *z* − *cnj z* = *complex-of-real* (*2* ∗ *Im*(*z*)) ∗ *ii*
apply (*induct z*)
apply (*simp add*: *complex-add complex-cnj complex-of-real-def diff-minus*
complex-minus i-def complex-mult)

done

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

lemma *complex-cnj-zero-iff* [*iff*]: (*cnj z = 0*) = (*z = 0*)
by (*induct z*, *simp add*: *complex-zero-def complex-cnj*)

lemma *complex-mult-cnj*: *z * cnj z = complex-of-real* (*Re*(*z*) ^ *2* + *Im*(*z*) ^ *2*)
by (*induct z*,
simp add: *complex-cnj complex-mult complex-of-real-def power2-eq-square*)

35.9 Modulus

lemma *complex-mod-eq-zero-cancel* [*simp*]: (*cmod x = 0*) = (*x = 0*)
apply (*induct x*)
apply (*auto iff*: *real-0-le-add-iff*
intro: *real-sum-squares-cancel real-sum-squares-cancel2*
simp add: *complex-mod complex-zero-def power2-eq-square*)
done

lemma *complex-mod-complex-of-real-of-nat* [*simp*]:
cmod (*complex-of-real*(*real* (*n*::*nat*))) = *real n*
by *simp*

lemma *complex-mod-minus* [*simp*]: *cmod* (−*x*) = *cmod*(*x*)
by (*induct x*, *simp add*: *complex-mod complex-minus power2-eq-square*)

lemma *complex-mod-mult-cnj*: *cmod*(*z * cnj*(*z*)) = *cmod*(*z*) ^ *2*
apply (*induct z*, *simp add*: *complex-mod complex-cnj complex-mult*)
apply (*simp add*: *power2-eq-square abs-if linorder-not-less real-0-le-add-iff*)
done

lemma *complex-mod-squared*: *cmod*(*Complex x y*) ^ *2* = *x* ^ *2* + *y* ^ *2*
by (*simp add*: *cmod-def*)

lemma *complex-mod-ge-zero* [*simp*]: *0* ≤ *cmod x*
by (*simp add*: *cmod-def*)

lemma *abs-cmod-cancel* [*simp*]: *abs*(*cmod x*) = *cmod x*
by (*simp add*: *abs-if linorder-not-less*)

lemma *complex-mod-mult*: *cmod*(*x*∗*y*) = *cmod*(*x*) ∗ *cmod*(*y*)
apply (*induct x*, *induct y*)
apply (*auto simp add*: *complex-mult complex-mod real-sqrt-mult-distrib2*[*symmetric*])
apply (*rule-tac n = 1* **in** *power-inject-base*)
apply (*auto simp add*: *power2-eq-square* [*symmetric*] *simp del*: *realpow-Suc*)
apply (*auto simp add*: *real-diff-def power2-eq-square right-distrib left-distrib*
add-ac mult-ac)

done

lemma *cmod-unit-one* [*simp*]: *cmod* (*Complex* (*cos a*) (*sin a*)) = *1*
by (*simp add*: *cmod-def*)

lemma *cmod-complex-polar* [*simp*]:
cmod (*complex-of-real r* ∗ *Complex* (*cos a*) (*sin a*)) = *abs r*
by (*simp only*: *cmod-unit-one complex-mod-mult*, *simp*)

lemma *complex-mod-add-squared-eq*:
cmod(*x* + *y*) ^ *2* = *cmod*(*x*) ^ *2* + *cmod*(*y*) ^ *2* + *2* ∗ *Re*(*x* ∗ *cnj y*)
apply (*induct x*, *induct y*)
apply (*auto simp add*: *complex-add complex-mod-squared complex-mult complex-cnj real-diff-def simp del*: *realpow-Suc*)
apply (*auto simp add*: *right-distrib left-distrib power2-eq-square mult-ac add-ac*)
done

lemma *complex-Re-mult-cnj-le-cmod* [*simp*]: *Re*(*x* ∗ *cnj y*) ≤ *cmod*(*x* ∗ *cnj y*)
apply (*induct x*, *induct y*)
apply (*auto simp add*: *complex-mod complex-mult complex-cnj real-diff-def simp del*: *realpow-Suc*)
done

lemma *complex-Re-mult-cnj-le-cmod2* [*simp*]: *Re*(*x* ∗ *cnj y*) ≤ *cmod*(*x* ∗ *y*)
by (*insert complex-Re-mult-cnj-le-cmod* [*of x y*], *simp add*: *complex-mod-mult*)

lemma *real-sum-squared-expand*:
((*x*::*real*) + *y*) ^ *2* = *x* ^ *2* + *y* ^ *2* + *2* ∗ *x* ∗ *y*
by (*simp add*: *left-distrib right-distrib power2-eq-square*)

lemma *complex-mod-triangle-squared* [*simp*]:
cmod (*x* + *y*) ^ *2* ≤ (*cmod*(*x*) + *cmod*(*y*)) ^ *2*
by (*simp add*: *real-sum-squared-expand complex-mod-add-squared-eq real-mult-assoc complex-mod-mult* [*symmetric*])

lemma *complex-mod-minus-le-complex-mod* [*simp*]: − *cmod x* ≤ *cmod x*
by (*rule order-trans* [*OF - complex-mod-ge-zero*], *simp*)

lemma *complex-mod-triangle-ineq* [*simp*]: *cmod* (*x* + *y*) ≤ *cmod*(*x*) + *cmod*(*y*)
apply (*rule-tac n* = *1* **in** *realpow-increasing*)
apply (*auto intro*: *order-trans* [*OF - complex-mod-ge-zero*]
simp add: *add-increasing power2-eq-square* [*symmetric*])
done

lemma *complex-mod-triangle-ineq2* [*simp*]: *cmod*(*b* + *a*) − *cmod b* ≤ *cmod a*
by (*insert complex-mod-triangle-ineq* [*THEN add-right-mono*, *of b a*−*cmod b*], *simp*)

lemma *complex-mod-diff-commute*: *cmod* (*x* − *y*) = *cmod* (*y* − *x*)

apply (*induct x*, *induct y*)
apply (*auto simp add*: *complex-diff complex-mod right-diff-distrib power2-eq-square left-diff-distrib add-ac mult-ac*)
done

lemma *complex-mod-add-less*:
[| *cmod x* < *r*; *cmod y* < *s* |] ==> *cmod* (*x* + *y*) < *r* + *s*
by (*auto intro*: *order-le-less-trans complex-mod-triangle-ineq*)

lemma *complex-mod-mult-less*:
[| *cmod x* < *r*; *cmod y* < *s* |] ==> *cmod* (*x* * *y*) < *r* * *s*
by (*auto intro*: *real-mult-less-mono'* *simp add*: *complex-mod-mult*)

lemma *complex-mod-diff-ineq* [*simp*]: *cmod*(*a*) − *cmod*(*b*) ≤ *cmod*(*a* + *b*)
apply (*rule linorder-cases* [*of cmod*(*a*) *cmod* (*b*)])
apply *auto*
apply (*rule order-trans* [*of* - *0*], *rule order-less-imp-le*)
apply (*simp add*: *compare-rls*, *simp*)
apply (*simp add*: *compare-rls*)
apply (*rule complex-mod-minus* [*THEN subst*])
apply (*rule order-trans*)
apply (*rule-tac* [*2*] *complex-mod-triangle-ineq*)
apply (*auto simp add*: *add-ac*)
done

lemma *complex-Re-le-cmod* [*simp*]: *Re z* ≤ *cmod z*
by (*induct z*, *simp add*: *complex-mod del*: *realpow-Suc*)

lemma *complex-mod-gt-zero*: *z* ≠ *0* ==> *0* < *cmod z*
apply (*insert complex-mod-ge-zero* [*of z*])
apply (*drule order-le-imp-less-or-eq*, *auto*)
done

35.10 A Few More Theorems

lemma *complex-mod-inverse*: *cmod*(*inverse x*) = *inverse*(*cmod x*)
apply (*case-tac x=0*, *simp*)
apply (*rule-tac c1* = *cmod x* **in** *real-mult-left-cancel* [*THEN iffD1*])
apply (*auto simp add*: *complex-mod-mult* [*symmetric*])
done

lemma *complex-mod-divide*: *cmod*(*x*/*y*) = *cmod*(*x*)/(*cmod y*)
by (*simp add*: *complex-divide-def divide-inverse complex-mod-mult complex-mod-inverse*)

35.11 Exponentiation

primrec
complexpow-0: *z* ^ *0* = *1*
complexpow-Suc: *z* ^ (*Suc n*) = (*z*::*complex*) * (*z* ^ *n*)

instance *complex* :: *recpower*
proof
  fix *z* :: *complex*
  fix *n* :: *nat*
  show *z^0 = 1* **by** *simp*
  show *z^(Suc n) = z * (z^n)* **by** *simp*
qed

lemma *complex-of-real-pow*: *complex-of-real (x ^ n) = (complex-of-real x) ^ n*
apply (*induct-tac n*)
apply (*auto simp add*: *complex-of-real-mult* [*symmetric*])
done

lemma *complex-cnj-pow*: *cnj(z ^ n) = cnj(z) ^ n*
apply (*induct-tac n*)
apply (*auto simp add*: *complex-cnj-mult*)
done

lemma *complex-mod-complexpow*: *cmod(x ^ n) = cmod(x) ^ n*
apply (*induct-tac n*)
apply (*auto simp add*: *complex-mod-mult*)
done

lemma *complexpow-i-squared* [*simp*]: *ii ^ 2 = −(1::complex)*
by (*simp add*: *i-def complex-mult complex-one-def complex-minus numeral-2-eq-2*)

lemma *complex-i-not-zero* [*simp*]: *ii ≠ 0*
by (*simp add*: *i-def complex-zero-def*)

35.12 The Function *sgn*

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

lemma *sgn-one* [*simp*]: *sgn 1 = 1*
by (*simp add*: *sgn-def*)

lemma *sgn-minus*: *sgn (−z) = − sgn(z)*
by (*simp add*: *sgn-def*)

lemma *sgn-eq*: *sgn z = z / complex-of-real (cmod z)*
by (*simp add*: *sgn-def*)

lemma *i-mult-eq*: *ii * ii = complex-of-real (−1)*
by (*simp add*: *i-def complex-of-real-def complex-mult complex-add*)

lemma *i-mult-eq2* [*simp*]: *ii * ii = −(1::complex)*

by (*simp add*: *i-def complex-one-def complex-mult complex-minus*)

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 *complex-eq-cancel-iff2a* [*simp*]:
(*Complex x y* = *complex-of-real xa*) = (*x* = *xa* & *y* = *0*)
by (*simp add*: *complex-of-real-def*)

lemma *Complex-eq-0* [*simp*]: (*Complex x y* = *0*) = (*x* = *0* & *y* = *0*)
by (*simp add*: *complex-zero-def*)

lemma *Complex-eq-1* [*simp*]: (*Complex x y* = *1*) = (*x* = *1* & *y* = *0*)
by (*simp add*: *complex-one-def*)

lemma *Complex-eq-i* [*simp*]: (*Complex x y* = *ii*) = (*x* = *0* & *y* = *1*)
by (*simp add*: *i-def*)

lemma *Re-sgn* [*simp*]: *Re*(*sgn z*) = *Re*(*z*)/*cmod z*
proof (*induct z*)
case (*Complex x y*)
have *sqrt* (x^2 + y^2) * *inverse* (x^2 + y^2) = *inverse* (*sqrt* (x^2 + y^2))
by (*simp add*: *divide-inverse* [*symmetric*] *sqrt-divide-self-eq*)
thus *Re* (*sgn* (*Complex x y*)) = *Re* (*Complex x y*) /*cmod* (*Complex x y*)
by (*simp add*: *sgn-def complex-of-real-def divide-inverse*)
qed

lemma *Im-sgn* [*simp*]: *Im*(*sgn z*) = *Im*(*z*)/*cmod z*
proof (*induct z*)
case (*Complex x y*)
have *sqrt* (x^2 + y^2) * *inverse* (x^2 + y^2) = *inverse* (*sqrt* (x^2 + y^2))
by (*simp add*: *divide-inverse* [*symmetric*] *sqrt-divide-self-eq*)
thus *Im* (*sgn* (*Complex x y*)) = *Im* (*Complex x y*) /*cmod* (*Complex x y*)
by (*simp add*: *sgn-def complex-of-real-def divide-inverse*)
qed

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 complex-mult complex-add*
diff-minus complex-minus complex-inverse divide-inverse)

lemma *complex-of-real-zero-iff* [*simp*]: (*complex-of-real y = 0*) = (*y = 0*)
by (*auto simp add*: *complex-zero-def complex-of-real-def*)

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*)

35.13 Finally! Polar Form for Complex Numbers

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*))
apply (*simp add*: *cmod-def*)
apply (*rule real-sqrt-eq-iff* [*THEN iffD2*])
apply (*auto simp add*: *complex-mult-cnj*)
done

lemma *complex-Re-cnj* [*simp*]: *Re*(*cnj z*) = *Re z*
by (*induct z*, *simp add*: *complex-cnj*)

lemma *complex-Im-cnj* [*simp*]: *Im*(*cnj z*) = − *Im z*
by (*induct z*, *simp add*: *complex-cnj*)

lemma *complex-In-mult-cnj-zero* [*simp*]: *Im* (*z* ∗ *cnj z*) = *0*
by (*induct z*, *simp add*: *complex-cnj complex-mult*)

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 complex-minus*)

lemma *complex-i-mult-minus* [*simp*]: *ii* ∗ (*ii* ∗ *x*) = − *x*
by (*simp add*: *complex-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 complex-of-real-pow*)

lemma *cis-inverse* [*simp*]: *inverse*(*cis a*) = *cis* (−*a*)
by (*simp add*: *cis-def complex-inverse-complex-split complex-of-real-minus*
    diff-minus)

lemma *rcis-inverse*: *inverse*(*rcis r a*) = *rcis* (*1*/*r*) (−*a*)
by (*simp add*: *divide-inverse rcis-def complex-of-real-inverse*)

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 complex-Re-add exp-add complex-Im-add*
    cis-mult [*symmetric*] *complex-of-real-mult mult-ac*)

lemma *expi-zero* [*simp*]: *expi* (*0*::*complex*) = *1*
by (*simp add*: *expi-def*)

lemma *complex-expi-Ex*: $\exists\, a\ r.\ z = \text{complex-of-real}\ r * \text{expi}\ a$
apply (*insert rcis-Ex* [*of z*])
apply (*auto simp add*: *expi-def rcis-def complex-mult-assoc* [*symmetric*] *complex-of-real-mult*)
apply (*rule-tac x* = *ii* * *complex-of-real a* **in** *exI*, *auto*)
done

35.14 Numerals and Arithmetic

instance *complex* :: *number* **..**

defs (**overloaded**)
complex-number-of-def: (*number-of w* :: *complex*) == *of-int* (*Rep-Bin w*)
— the type constraint is essential!

instance *complex* :: *number-ring*
by (*intro-classes*, *simp add*: *complex-number-of-def*)

lemma *complex-of-real-of-nat* [*simp*]: *complex-of-real* (*of-nat n*) = *of-nat n*
by (*induct n*, *simp-all*)

lemma *complex-of-real-of-int* [*simp*]: *complex-of-real* (*of-int z*) = *of-int z*
proof (*cases z*)
case (*1 n*)
thus *?thesis* **by** *simp*
next
case (*2 n*)
thus *?thesis*
by (*simp only*: *of-int-minus complex-of-real-minus*, *simp*)
qed

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

lemma *complex-number-of* [*simp*]: *complex-of-real* (*number-of w*) = *number-of w*
by (*simp add*: *complex-number-of-def real-number-of-def*)

This theorem is necessary because theorems such as *iszero-number-of-0* only hold for ordered rings. They cannot be generalized to fields in general because they fail for finite fields. They work for type complex because the reals can be embedded in them.

lemma *iszero-complex-number-of* [*simp*]:
iszero (*number-of w* :: *complex*) = *iszero* (*number-of w* :: *real*)
by (*simp only*: *complex-of-real-zero-iff complex-number-of* [*symmetric*]
iszero-def)

lemma *complex-number-of-cnj* [*simp*]: *cnj*(*number-of v* :: *complex*) = *number-of v*
by (*simp only*: *complex-number-of* [*symmetric*] *complex-cnj-complex-of-real*)

lemma *complex-number-of-cmod*:

cmod(*number-of v* :: *complex*) = *abs* (*number-of v* :: *real*)
by (*simp only*: *complex-number-of* [*symmetric*] *complex-mod-complex-of-real*)

lemma *complex-number-of-Re* [*simp*]: *Re*(*number-of v* :: *complex*) = *number-of v*
by (*simp only*: *complex-number-of* [*symmetric*] *Re-complex-of-real*)

lemma *complex-number-of-Im* [*simp*]: *Im*(*number-of v* :: *complex*) = *0*
by (*simp only*: *complex-number-of* [*symmetric*] *Im-complex-of-real*)

lemma *expi-two-pi-i* [*simp*]: *expi*((*2*::*complex*) ∗ *complex-of-real pi* ∗ *ii*) = *1*
by (*simp add*: *expi-def complex-Re-mult-eq complex-Im-mult-eq cis-def*)

ML
⟪
val complex-zero-def = *thmcomplex-zero-def*;
val complex-one-def = *thmcomplex-one-def*;
val complex-minus-def = *thmcomplex-minus-def*;
val complex-divide-def = *thmcomplex-divide-def*;
val complex-mult-def = *thmcomplex-mult-def*;
val complex-add-def = *thmcomplex-add-def*;
val complex-of-real-def = *thmcomplex-of-real-def*;
val i-def = *thmi-def*;
val expi-def = *thmexpi-def*;
val cis-def = *thmcis-def*;
val rcis-def = *thmrcis-def*;
val cmod-def = *thmcmod-def*;
val cnj-def = *thmcnj-def*;
val sgn-def = *thmsgn-def*;
val arg-def = *thmarg-def*;
val complexpow-0 = *thmcomplexpow-0*;
val complexpow-Suc = *thmcomplexpow-Suc*;

val Re = *thmRe*;
val Im = *thmIm*;
val complex-Re-Im-cancel-iff = *thmcomplex-Re-Im-cancel-iff*;
val complex-Re-zero = *thmcomplex-Re-zero*;
val complex-Im-zero = *thmcomplex-Im-zero*;
val complex-Re-one = *thmcomplex-Re-one*;
val complex-Im-one = *thmcomplex-Im-one*;
val complex-Re-i = *thmcomplex-Re-i*;
val complex-Im-i = *thmcomplex-Im-i*;
val Re-complex-of-real = *thmRe-complex-of-real*;
val Im-complex-of-real = *thmIm-complex-of-real*;
val complex-minus = *thmcomplex-minus*;
val complex-Re-minus = *thmcomplex-Re-minus*;

val complex-Im-minus = thmcomplex-Im-minus;
val complex-add = thmcomplex-add;
val complex-Re-add = thmcomplex-Re-add;
val complex-Im-add = thmcomplex-Im-add;
val complex-add-commute = thmcomplex-add-commute;
val complex-add-assoc = thmcomplex-add-assoc;
val complex-add-zero-left = thmcomplex-add-zero-left;
val complex-add-zero-right = thmcomplex-add-zero-right;
val complex-diff = thmcomplex-diff;
val complex-mult = thmcomplex-mult;
val complex-mult-one-left = thmcomplex-mult-one-left;
val complex-mult-one-right = thmcomplex-mult-one-right;
val complex-inverse = thmcomplex-inverse;
val complex-of-real-one = thmcomplex-of-real-one;
val complex-of-real-zero = thmcomplex-of-real-zero;
val complex-of-real-eq-iff = thmcomplex-of-real-eq-iff;
val complex-of-real-minus = thmcomplex-of-real-minus;
val complex-of-real-inverse = thmcomplex-of-real-inverse;
val complex-of-real-add = thmcomplex-of-real-add;
val complex-of-real-diff = thmcomplex-of-real-diff;
val complex-of-real-mult = thmcomplex-of-real-mult;
val complex-of-real-divide = thmcomplex-of-real-divide;
val complex-of-real-pow = thmcomplex-of-real-pow;
val complex-mod = thmcomplex-mod;
val complex-mod-zero = thmcomplex-mod-zero;
val complex-mod-one = thmcomplex-mod-one;
val complex-mod-complex-of-real = thmcomplex-mod-complex-of-real;
val complex-of-real-abs = thmcomplex-of-real-abs;
val complex-cnj = thmcomplex-cnj;
val complex-cnj-cancel-iff = thmcomplex-cnj-cancel-iff;
val complex-cnj-cnj = thmcomplex-cnj-cnj;
val complex-cnj-complex-of-real = thmcomplex-cnj-complex-of-real;
val complex-mod-cnj = thmcomplex-mod-cnj;
val complex-cnj-minus = thmcomplex-cnj-minus;
val complex-cnj-inverse = thmcomplex-cnj-inverse;
val complex-cnj-add = thmcomplex-cnj-add;
val complex-cnj-diff = thmcomplex-cnj-diff;
val complex-cnj-mult = thmcomplex-cnj-mult;
val complex-cnj-divide = thmcomplex-cnj-divide;
val complex-cnj-one = thmcomplex-cnj-one;
val complex-cnj-pow = thmcomplex-cnj-pow;
val complex-add-cnj = thmcomplex-add-cnj;
val complex-diff-cnj = thmcomplex-diff-cnj;
val complex-cnj-zero = thmcomplex-cnj-zero;
val complex-cnj-zero-iff = thmcomplex-cnj-zero-iff;
val complex-mult-cnj = thmcomplex-mult-cnj;
val complex-mod-eq-zero-cancel = thmcomplex-mod-eq-zero-cancel;
val complex-mod-complex-of-real-of-nat = thmcomplex-mod-complex-of-real-of-nat;
val complex-mod-minus = thmcomplex-mod-minus;

val complex-mod-mult-cnj = thmcomplex-mod-mult-cnj;
val complex-mod-squared = thmcomplex-mod-squared;
val complex-mod-ge-zero = thmcomplex-mod-ge-zero;
val abs-cmod-cancel = thmabs-cmod-cancel;
val complex-mod-mult = thmcomplex-mod-mult;
val complex-mod-add-squared-eq = thmcomplex-mod-add-squared-eq;
val complex-Re-mult-cnj-le-cmod = thmcomplex-Re-mult-cnj-le-cmod;
val complex-Re-mult-cnj-le-cmod2 = thmcomplex-Re-mult-cnj-le-cmod2;
val real-sum-squared-expand = thmreal-sum-squared-expand;
val complex-mod-triangle-squared = thmcomplex-mod-triangle-squared;
val complex-mod-minus-le-complex-mod = thmcomplex-mod-minus-le-complex-mod;
val complex-mod-triangle-ineq = thmcomplex-mod-triangle-ineq;
val complex-mod-triangle-ineq2 = thmcomplex-mod-triangle-ineq2;
val complex-mod-diff-commute = thmcomplex-mod-diff-commute;
val complex-mod-add-less = thmcomplex-mod-add-less;
val complex-mod-mult-less = thmcomplex-mod-mult-less;
val complex-mod-diff-ineq = thmcomplex-mod-diff-ineq;
val complex-Re-le-cmod = thmcomplex-Re-le-cmod;
val complex-mod-gt-zero = thmcomplex-mod-gt-zero;
val complex-mod-complexpow = thmcomplex-mod-complexpow;
val complex-mod-inverse = thmcomplex-mod-inverse;
val complex-mod-divide = thmcomplex-mod-divide;
val complexpow-i-squared = thmcomplexpow-i-squared;
val complex-i-not-zero = thmcomplex-i-not-zero;
val sgn-zero = thmsgn-zero;
val sgn-one = thmsgn-one;
val sgn-minus = thmsgn-minus;
val sgn-eq = thmsgn-eq;
val i-mult-eq = thmi-mult-eq;
val i-mult-eq2 = thmi-mult-eq2;
val Re-sgn = thmRe-sgn;
val Im-sgn = thmIm-sgn;
val complex-inverse-complex-split = thmcomplex-inverse-complex-split;
val cos-arg-i-mult-zero = thmcos-arg-i-mult-zero;
val complex-of-real-zero-iff = thmcomplex-of-real-zero-iff;
val rcis-Ex = thmrcis-Ex;
val Re-rcis = thmRe-rcis;
val Im-rcis = thmIm-rcis;
val complex-mod-rcis = thmcomplex-mod-rcis;
val complex-mod-sqrt-Re-mult-cnj = thmcomplex-mod-sqrt-Re-mult-cnj;
val complex-Re-cnj = thmcomplex-Re-cnj;
val complex-Im-cnj = thmcomplex-Im-cnj;
val complex-In-mult-cnj-zero = thmcomplex-In-mult-cnj-zero;
val complex-Re-mult = thmcomplex-Re-mult;
val complex-Re-mult-complex-of-real = thmcomplex-Re-mult-complex-of-real;
val complex-Im-mult-complex-of-real = thmcomplex-Im-mult-complex-of-real;
val complex-Re-mult-complex-of-real2 = thmcomplex-Re-mult-complex-of-real2;
val complex-Im-mult-complex-of-real2 = thmcomplex-Im-mult-complex-of-real2;
val cis-rcis-eq = thmcis-rcis-eq;

val rcis-mult = thmrcis-mult;
val cis-mult = thmcis-mult;
val cis-zero = thmcis-zero;
val rcis-zero-mod = thmrcis-zero-mod;
val rcis-zero-arg = thmrcis-zero-arg;
val complex-of-real-minus-one = thmcomplex-of-real-minus-one;
val complex-i-mult-minus = thmcomplex-i-mult-minus;
val cis-real-of-nat-Suc-mult = thmcis-real-of-nat-Suc-mult;
val DeMoivre = thmDeMoivre;
val DeMoivre2 = thmDeMoivre2;
val cis-inverse = thmcis-inverse;
val rcis-inverse = thmrcis-inverse;
val cis-divide = thmcis-divide;
val rcis-divide = thmrcis-divide;
val Re-cis = thmRe-cis;
val Im-cis = thmIm-cis;
val cos-n-Re-cis-pow-n = thmcos-n-Re-cis-pow-n;
val sin-n-Im-cis-pow-n = thmsin-n-Im-cis-pow-n;
val expi-add = thmexpi-add;
val expi-zero = thmexpi-zero;
val complex-Re-mult-eq = thmcomplex-Re-mult-eq;
val complex-Im-mult-eq = thmcomplex-Im-mult-eq;
val complex-expi-Ex = thmcomplex-expi-Ex;
⟩⟩

end

36 NSComplex: Nonstandard Complex Numbers

theory *NSComplex*
imports *Complex*
begin

types *hcomplex = complex star*

syntax *hcomplex-of-complex :: real => real star*
translations *hcomplex-of-complex => star-of :: complex => complex star*

constdefs

hRe :: hcomplex => hypreal
*hRe == *f* Re*

hIm :: hcomplex => hypreal

$hIm == *f* Im$

$hcmod :: hcomplex => hypreal$
$hcmod == *f* cmod$

$iii :: hcomplex$
$iii == star\text{-}of\ ii$

$hcnj :: hcomplex => hcomplex$
$hcnj == *f* cnj$

$hsgn :: hcomplex => hcomplex$
$hsgn == *f* sgn$

$harg :: hcomplex => hypreal$
$harg == *f* arg$

$hcis :: hypreal => hcomplex$
$hcis == *f* cis$

$hcomplex\text{-}of\text{-}hypreal :: hypreal => hcomplex$
$hcomplex\text{-}of\text{-}hypreal == *f* complex\text{-}of\text{-}real$

$hrcis :: [hypreal, hypreal] => hcomplex$
$hrcis == *f2* rcis$

$hexpi :: hcomplex => hcomplex$
$hexpi == *f* expi$

$HComplex :: [hypreal,hypreal] => hcomplex$
$HComplex == *f2* Complex$

$hcpow :: [hcomplex,hypnat] => hcomplex\ \ (\textbf{infixr}\ hcpow\ 80)$
$(z::hcomplex)\ hcpow\ (n::hypnat) == (\ *f2* op\ \hat{}\ )\ z\ n$

lemmas *hcomplex-defs* [*transfer-unfold*] =
hRe-def hIm-def hcmod-def iii-def hcnj-def hsgn-def harg-def hcis-def
hcomplex-of-hypreal-def hrcis-def hexpi-def HComplex-def hcpow-def

36.1 Properties of Nonstandard Real and Imaginary Parts

lemma *hRe*: *hRe* (*star-n X*) = *star-n* (%*n*. *Re*(*X n*))
by (*simp add*: *hRe-def starfun*)

lemma *hIm*: *hIm* (*star-n X*) = *star-n* (%*n*. *Im*(*X n*))
by (*simp add*: *hIm-def starfun*)

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?*]: *hRe z* = *hRe w* ==> *hIm z* = *hIm w* ==> *z* = *w*
by (*simp add*: *hcomplex-hRe-hIm-cancel-iff*)

lemma *hcomplex-hRe-zero* [*simp*]: *hRe 0* = *0*
by (*simp add*: *hRe star-n-zero-num*)

lemma *hcomplex-hIm-zero* [*simp*]: *hIm 0* = *0*
by (*simp add*: *hIm star-n-zero-num*)

lemma *hcomplex-hRe-one* [*simp*]: *hRe 1* = *1*
by (*simp add*: *hRe star-n-one-num*)

lemma *hcomplex-hIm-one* [*simp*]: *hIm 1* = *0*
by (*simp add*: *hIm star-n-one-num star-n-zero-num*)

36.2 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*)

36.3 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 (*simp add*: *iii-def star-of-def star-n-mult star-n-one-num star-n-minus*)

lemma *hcomplex-i-mult-left* [*simp*]: $iii * (iii * z) = -z$
by (*simp add*: *mult-assoc* [*symmetric*])

lemma *hcomplex-i-not-zero* [*simp*]: $iii \neq 0$
by (*simp add*: *iii-def star-of-def star-n-zero-num star-n-eq-iff*)

36.4 More Multiplication Laws

lemma *hcomplex-mult-minus-one* [*simp*]: $-1 * (z::hcomplex) = -z$
by *simp*

lemma *hcomplex-mult-minus-one-right* [*simp*]: $(z::hcomplex) * -1 = -z$
by *simp*

lemma *hcomplex-mult-left-cancel*:
$(c::hcomplex) \neq (0::hcomplex) ==> (c*a=c*b) = (a=b)$
by (*simp add*: *field-mult-cancel-left*)

lemma *hcomplex-mult-right-cancel*:
$(c::hcomplex) \neq (0::hcomplex) ==> (a*c=b*c) = (a=b)$
by (*simp add*: *Ring-and-Field.field-mult-cancel-right*)

36.5 Subraction and Division

lemma *hcomplex-diff-eq-eq* [*simp*]: $((x::hcomplex) - y = z) = (x = z + y)$
by (*rule OrderedGroup.diff-eq-eq*)

lemma *hcomplex-add-divide-distrib*: $(x+y)/(z::hcomplex) = x/z + y/z$
by (*rule Ring-and-Field.add-divide-distrib*)

36.6 Embedding Properties for *hcomplex-of-hypreal* Map

lemma *hcomplex-of-hypreal*:
hcomplex-of-hypreal (*star-n X*) = *star-n* (%*n*. *complex-of-real* (*X n*))
by (*simp add*: *hcomplex-of-hypreal-def starfun*)

lemma *hcomplex-of-hypreal-cancel-iff* [*iff*]:
!!*x y*. (*hcomplex-of-hypreal x* = *hcomplex-of-hypreal y*) = (*x* = *y*)
by (*transfer*, *simp*)

lemma *hcomplex-of-hypreal-one* [*simp*]: *hcomplex-of-hypreal 1* = *1*
by (*simp add*: *hcomplex-of-hypreal star-n-one-num*)

lemma *hcomplex-of-hypreal-zero* [*simp*]: *hcomplex-of-hypreal 0 = 0*
by (*simp add*: *star-n-zero-num hcomplex-of-hypreal*)

lemma *hcomplex-of-hypreal-minus* [*simp*]:
!!*x*. *hcomplex-of-hypreal*(−*x*) = − *hcomplex-of-hypreal x*
by (*transfer*, *simp*)

lemma *hcomplex-of-hypreal-inverse* [*simp*]:
!!*x*. *hcomplex-of-hypreal*(*inverse x*) = *inverse*(*hcomplex-of-hypreal x*)
by (*transfer*, *simp*)

lemma *hcomplex-of-hypreal-add* [*simp*]:
!!*x y*. *hcomplex-of-hypreal* (*x* + *y*) =
hcomplex-of-hypreal x + *hcomplex-of-hypreal y*
by (*transfer*, *simp*)

lemma *hcomplex-of-hypreal-diff* [*simp*]:
!!*x y*. *hcomplex-of-hypreal* (*x* − *y*) =
hcomplex-of-hypreal x − *hcomplex-of-hypreal y*
by (*transfer*, *simp*)

lemma *hcomplex-of-hypreal-mult* [*simp*]:
!!*x y*. *hcomplex-of-hypreal* (*x* ∗ *y*) =
hcomplex-of-hypreal x ∗ *hcomplex-of-hypreal y*
by (*transfer*, *simp*)

lemma *hcomplex-of-hypreal-divide* [*simp*]:
!!*x y*. *hcomplex-of-hypreal*(*x*/*y*) =
hcomplex-of-hypreal x / *hcomplex-of-hypreal y*
by (*transfer*, *simp*)

lemma *hRe-hcomplex-of-hypreal* [*simp*]: !!*z*. *hRe*(*hcomplex-of-hypreal z*) = *z*
by (*transfer*, *simp*)

lemma *hIm-hcomplex-of-hypreal* [*simp*]: !!*z*. *hIm*(*hcomplex-of-hypreal z*) = *0*
by (*transfer*, *simp*)

lemma *hcomplex-of-hypreal-epsilon-not-zero* [*simp*]:
hcomplex-of-hypreal epsilon ≠ *0*
by (*simp add*: *hcomplex-of-hypreal epsilon-def star-n-zero-num star-n-eq-iff*)

36.7 HComplex theorems

lemma *hRe-HComplex* [*simp*]: !!*x y*. *hRe* (*HComplex x y*) = *x*
by (*transfer*, *simp*)

lemma *hIm-HComplex* [*simp*]: !!*x y*. *hIm* (*HComplex x y*) = *y*
by (*transfer*, *simp*)

Relates the two nonstandard constructions

lemma *HComplex-eq-Abs-star-Complex*:
HComplex (star-n X) (star-n Y) =
star-n (%n::nat. Complex (X n) (Y n))
by (*simp add*: *hcomplex-hRe-hIm-cancel-iff hRe hIm*)

lemma *hcomplex-surj* [*simp*]: *HComplex (hRe z) (hIm z) = z*
by (*simp add*: *hcomplex-equality*)

lemma *hcomplex-induct* [*case-names rect*]:
(⋀x y. P (HComplex x y)) ==> P z
by (*rule hcomplex-surj* [*THEN subst*], *blast*)

36.8 Modulus (Absolute Value) of Nonstandard Complex Number

lemma *hcmod*: *hcmod (star-n X) = star-n (%n. cmod (X n))*
by (*simp add*: *hcmod-def starfun*)

lemma *hcmod-zero* [*simp*]: *hcmod(0) = 0*
by (*simp add*: *star-n-zero-num hcmod*)

lemma *hcmod-one* [*simp*]: *hcmod(1) = 1*
by (*simp add*: *hcmod star-n-one-num*)

lemma *hcmod-hcomplex-of-hypreal* [*simp*]:
!!x. hcmod(hcomplex-of-hypreal x) = abs x
by (*transfer*, *simp*)

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*, *simp*)

lemma *HComplex-add* [*simp*]:
!!x1 y1 x2 y2. HComplex x1 y1 + HComplex x2 y2 = HComplex (x1+x2) (y1+y2)
by (*transfer*, *simp*)

lemma *HComplex-minus* [*simp*]: *!!x y. − HComplex x y = HComplex (−x) (−y)*
by (*transfer*, *simp*)

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*
apply (*transfer*)
apply (*simp add*: *complex-of-real-def*)
done

lemma *HComplex-add-hcomplex-of-hypreal* [*simp*]:
HComplex x y + *hcomplex-of-hypreal r* = *HComplex* (*x*+*r*) *y*
by (*simp add*: *hcomplex-of-hypreal-eq*)

lemma *hcomplex-of-hypreal-add-HComplex* [*simp*]:
hcomplex-of-hypreal r + *HComplex x y* = *HComplex* (*r*+*x*) *y*
by (*simp add*: *i-def hcomplex-of-hypreal-eq*)

lemma *HComplex-mult-hcomplex-of-hypreal*:
HComplex x y ∗ *hcomplex-of-hypreal r* = *HComplex* (*x*∗*r*) (*y*∗*r*)
by (*simp add*: *hcomplex-of-hypreal-eq*)

lemma *hcomplex-of-hypreal-mult-HComplex*:
hcomplex-of-hypreal r ∗ *HComplex x y* = *HComplex* (*r*∗*x*) (*r*∗*y*)
by (*simp add*: *i-def hcomplex-of-hypreal-eq*)

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*)

36.9 Conjugation

lemma *hcnj*: *hcnj* (*star-n X*) = *star-n* (%*n*. *cnj*(*X n*))
by (*simp add*: *hcnj-def starfun*)

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*)

36.10 More Theorems about the Function *hcmod*

lemma *hcomplex-hcmod-eq-zero-cancel* [*simp*]: !!x. (*hcmod* x = *0*) = (x = *0*)
by (*transfer*, *rule complex-mod-eq-zero-cancel*)

lemma *hcmod-hcomplex-of-hypreal-of-nat* [*simp*]:
hcmod (*hcomplex-of-hypreal*(*hypreal-of-nat* n)) = *hypreal-of-nat* n
by (*simp add*: *abs-if linorder-not-less*)

lemma *hcmod-hcomplex-of-hypreal-of-hypnat* [*simp*]:
hcmod (*hcomplex-of-hypreal*(*hypreal-of-hypnat* n)) = *hypreal-of-hypnat* n
by (*simp add*: *abs-if linorder-not-less*)

lemma *hcmod-minus* [*simp*]: !!x. *hcmod* $(-x)$ = *hcmod*(x)
by (*transfer*, *rule complex-mod-minus*)

lemma *hcmod-mult-hcnj*: !!z. *hcmod*$(z * hcnj(z))$ = *hcmod*(z) ^ *2*
by (*transfer*, *rule complex-mod-mult-cnj*)

lemma *hcmod-ge-zero* [*simp*]: !!x. $(0$::*hypreal*$)$ $\leq$ *hcmod x*
by (*transfer*, *rule complex-mod-ge-zero*)

lemma *hrabs-hcmod-cancel* [*simp*]: *abs*(*hcmod x*) = *hcmod x*
by (*simp add*: *abs-if linorder-not-less*)

lemma *hcmod-mult*: !!x y. *hcmod*$(x*y)$ = *hcmod*(x) * *hcmod*(y)
by (*transfer*, *rule complex-mod-mult*)

lemma *hcmod-add-squared-eq*:
!!x y. *hcmod*$(x + y)$ ^ *2* = *hcmod*(x) ^ *2* + *hcmod*(y) ^ *2* + *2* * *hRe*$(x * hcnj$
$y)$
by (*transfer*, *rule complex-mod-add-squared-eq*)

lemma *hcomplex-hRe-mult-hcnj-le-hcmod* [*simp*]:
!!x y. *hRe*$(x * hcnj\ y)$ $\leq$ *hcmod*$(x * hcnj\ y)$
by (*transfer*, *simp*)

lemma *hcomplex-hRe-mult-hcnj-le-hcmod2* [*simp*]:
!!x y. *hRe*$(x * hcnj\ y)$ $\leq$ *hcmod*$(x * y)$
by (*transfer*, *simp*)

lemma *hcmod-triangle-squared* [*simp*]:
!!x y. *hcmod* $(x + y)$ ^ *2* $\leq$ $(hcmod(x) + hcmod(y))$ ^ *2*
by (*transfer*, *simp*)

lemma *hcmod-triangle-ineq* [*simp*]:
!!x y. *hcmod* $(x + y)$ $\leq$ *hcmod*(x) + *hcmod*(y)
by (*transfer*, *simp*)

lemma *hcmod-triangle-ineq2* [*simp*]:
!!a b. *hcmod*$(b + a)$ $-$ *hcmod b* $\leq$ *hcmod a*
by (*transfer*, *simp*)

lemma *hcmod-diff-commute*: !!x y. *hcmod* $(x - y)$ = *hcmod* $(y - x)$
by (*transfer*, *rule complex-mod-diff-commute*)

lemma *hcmod-add-less*:
!!x y r s. [| *hcmod x* $<$ r; *hcmod y* $<$ s |] ==> *hcmod* $(x + y)$ $<$ $r + s$
by (*transfer*, *rule complex-mod-add-less*)

lemma *hcmod-mult-less*:
!!x y r s. [| *hcmod x* $<$ r; *hcmod y* $<$ s |] ==> *hcmod* $(x * y)$ $<$ $r * s$

by (*transfer*, *rule complex-mod-mult-less*)

lemma *hcmod-diff-ineq* [*simp*]: !!*a b*. *hcmod*(*a*) − *hcmod*(*b*) ≤ *hcmod*(*a* + *b*)
by (*transfer*, *simp*)

36.11 A Few Nonlinear Theorems

lemma *hcpow*: *star-n X hcpow star-n Y* = *star-n* (%*n*. *X n* ^ *Y n*)
by (*simp add*: *hcpow-def starfun2-star-n*)

lemma *hcomplex-of-hypreal-hyperpow*:
!!*x n*. *hcomplex-of-hypreal* (*x pow n*) = (*hcomplex-of-hypreal x*) *hcpow n*
by (*transfer*, *rule complex-of-real-pow*)

lemma *hcmod-hcpow*: !!*x n*. *hcmod*(*x hcpow n*) = *hcmod*(*x*) *pow n*
by (*transfer*, *rule complex-mod-complexpow*)

lemma *hcmod-hcomplex-inverse*: !!*x*. *hcmod*(*inverse x*) = *inverse*(*hcmod x*)
by (*transfer*, *rule complex-mod-inverse*)

lemma *hcmod-divide*: *hcmod*(*x*/*y*) = *hcmod*(*x*)/(*hcmod y*)
by (*simp add*: *divide-inverse hcmod-mult hcmod-hcomplex-inverse*)

36.12 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 (*simp add*: *power2-eq-square*)

lemma *hcomplex-of-hypreal-pow*:
hcomplex-of-hypreal (*x* ^ *n*) = (*hcomplex-of-hypreal x*) ^ *n*
apply (*induct-tac n*)
apply (*auto simp add*: *hcomplex-of-hypreal-mult* [*symmetric*])
done

lemma *hcomplex-hcnj-pow*: *hcnj*(*z* ^ *n*) = *hcnj*(*z*) ^ *n*
apply (*induct-tac n*)
apply (*auto simp add*: *hcomplex-hcnj-mult*)
done

lemma *hcmod-hcomplexpow*: *hcmod*(*x* ^ *n*) = *hcmod*(*x*) ^ *n*
apply (*induct-tac n*)
apply (*auto simp add*: *hcmod-mult*)
done

lemma *hcpow-minus*:
    !!*x n*. (−*x*::*hcomplex*) *hcpow n* =
     (*if* (∗*p*∗ *even*) *n then* (*x hcpow n*) *else* −(*x hcpow n*))
by (*transfer*, *rule neg-power-if*)

lemma *hcpow-mult*:
  !!*r s n*. ((*r*::*hcomplex*) ∗ *s*) *hcpow n* = (*r hcpow n*) ∗ (*s hcpow n*)
by (*transfer*, *rule power-mult-distrib*)

lemma *hcpow-zero* [*simp*]: !!*n*. *0 hcpow* (*n* + *1*) = *0*
by (*transfer*, *simp*)

lemma *hcpow-zero2* [*simp*]: *0 hcpow* (*hSuc n*) = *0*
by (*simp add*: *hSuc-def*)

lemma *hcpow-not-zero* [*simp*,*intro*]:
  !!*r n*. *r* ≠ *0* ==> *r hcpow n* ≠ (*0*::*hcomplex*)
by (*transfer*, *simp*)

lemma *hcpow-zero-zero*: *r hcpow n* = (*0*::*hcomplex*) ==> *r* = *0*
by (*blast intro*: *ccontr dest*: *hcpow-not-zero*)

lemma *star-n-divide*: *star-n X* / *star-n Y* = *star-n* (%*n*. *X n* / *Y n*)
by (*simp add*: *star-divide-def starfun2-star-n*)

36.13 The Function *hsgn*

lemma *hsgn*: *hsgn* (*star-n X*) = *star-n* (%*n*. *sgn* (*X n*))
by (*simp add*: *hsgn-def starfun*)

lemma *hsgn-zero* [*simp*]: *hsgn 0* = *0*
by (*simp add*: *star-n-zero-num hsgn*)

lemma *hsgn-one* [*simp*]: *hsgn 1* = *1*
by (*simp add*: *star-n-one-num hsgn*)

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-mod*)

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*]: (*HComplex x y = 0*) = (*x = 0 & y = 0*)
by (*insert hcomplex-eq-cancel-iff2* [*of - - 0*], *simp*)

lemma *HComplex-eq-1* [*simp*]: (*HComplex x y = 1*) = (*x = 1 & y = 0*)
by (*insert hcomplex-eq-cancel-iff2* [*of - - 1*], *simp*)

lemma *i-eq-HComplex-0-1*: *iii = HComplex 0 1*
by (*insert hcomplex-of-hypreal-i* [*of 1*], *simp*)

lemma *HComplex-eq-i* [*simp*]: (*HComplex x y = iii*) = (*x = 0 & y = 1*)
by (*simp add*: *i-eq-HComplex-0-1*)

lemma *hRe-hsgn* [*simp*]: !!*z*. *hRe*(*hsgn z*) = *hRe*(*z*)/*hcmod z*
by (*transfer*, *simp*)

lemma *hIm-hsgn* [*simp*]: !!*z*. *hIm*(*hsgn z*) = *hIm*(*z*)/*hcmod z*
by (*transfer*, *simp*)

lemma *real-two-squares-add-zero-iff* [*simp*]: (*x**x + y**y = 0*) = ((*x*::*real*) = *0* & *y = 0*)
by (*auto intro*: *real-sum-squares-cancel iff*: *real-add-eq-0-iff*)

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*]: *hcmod* (*hcomplex-of-hypreal y* ∗ *iii*) = *abs y*
by (*simp only*: *hcmod-mult-i mult-commute*)

lemma *harg*: *harg* (*star-n X*) = *star-n* (%*n*. *arg* (*X n*))
by (*simp add*: *harg-def starfun*)

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 ≠ *0* ==> (∗*f*∗ *cos*) (*harg*(*HComplex 0 y*)) = *0*
by (*auto simp add*: *linorder-neq-iff*
        cos-harg-i-mult-zero-pos cos-harg-i-mult-zero-neg)

lemma *hcomplex-of-hypreal-zero-iff* [*simp*]:
  !!*y*. (*hcomplex-of-hypreal y* = *0*) = (*y* = *0*)
by (*transfer*, *simp*)

36.14 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 *lemma-hypreal-P-EX2*:
  (∃ (*x*::*hypreal*) *y*. *P x y*) =
   (∃ *f g*. *P* (*star-n f*) (*star-n g*))
apply *auto*
apply (*rule-tac x* = *x* **in** *star-cases*)
apply (*rule-tac x* = *y* **in** *star-cases*, *auto*)
done

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*: *hcis* (*star-n X*) = *star-n* (%*n*. *cis* (*X n*))
by (*simp add*: *hcis-def starfun*)

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*: *hrcis* (*star-n X*) (*star-n Y*) = *star-n* (%*n*. *rcis* (*X n*) (*Y n*))
by (*simp add*: *hrcis-def starfun2-star-n*)

lemma *hrcis-Ex*: !!*z*. ∃*r a*. *z* = *hrcis r a*
by (*transfer*, *rule rcis-Ex*)

lemma *hRe-hcomplex-polar* [*simp*]:
    hRe (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) =
     r ∗ (∗*f*∗ *cos*) *a*
by (*simp add*: *hcomplex-of-hypreal-mult-HComplex*)

lemma *hRe-hrcis* [*simp*]: !!*r a*. *hRe*(*hrcis r a*) = *r* ∗ (∗*f*∗ *cos*) *a*
by (*transfer*, *rule Re-rcis*)

lemma *hIm-hcomplex-polar* [*simp*]:
    hIm (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) =
     r ∗ (∗*f*∗ *sin*) *a*
by (*simp add*: *hcomplex-of-hypreal-mult-HComplex*)

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*, *simp*)

lemma *hcmod-complex-polar* [*simp*]:
    hcmod (*hcomplex-of-hypreal r* ∗ *HComplex* ((∗*f*∗ *cos*) *a*) ((∗*f*∗ *sin*) *a*)) =
     abs r
by (*simp only*: *hcmod-mult hcmod-unit-one*, *simp*)

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*]: *iii* ∗ (*iii* ∗ *x*) = − *x*
by (*simp add*: *mult-assoc* [*symmetric*])

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 (*unfold hypreal-of-nat-def*)
apply *transfer*
apply (*fold real-of-nat-def*)
apply (*rule cis-real-of-nat-Suc-mult*)
done

lemma *NSDeMoivre*: (*hcis a*) ^ *n* = *hcis* (*hypreal-of-nat n* ∗ *a*)
apply (*induct-tac n*)
apply (*simp-all add*: *hcis-hypreal-of-nat-Suc-mult*)
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*, *simp add*: *cis-real-of-nat-Suc-mult*)

lemma *NSDeMoivre-ext*:
!!*a n*. (*hcis a*) *hcpow n* = *hcis* (*hypreal-of-hypnat n* ∗ *a*)
by (*transfer*, *rule DeMoivre*)

lemma *NSDeMoivre2*:
!!*a r*. (*hrcis r a*) ^ *n* = *hrcis* (*r* ^ *n*) (*hypreal-of-nat n* ∗ *a*)
apply (*unfold hypreal-of-nat-def*)
apply *transfer*

apply (*fold real-of-nat-def*)
apply (*rule DeMoivre2*)
done

lemma *DeMoivre2-ext*:
  !! *a r n*. (*hrcis r a*) *hcpow n* = *hrcis* (*r pow n*) (*hypreal-of-hypnat n* ∗ *a*)
by (*transfer*, *rule DeMoivre2*)

lemma *hcis-inverse* [*simp*]: !!*a*. *inverse*(*hcis a*) = *hcis* (−*a*)
by (*transfer*, *simp*)

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 hcpow n*)
by (*simp add*: *NSDeMoivre-ext*)

lemma *sin-n-hIm-hcis-hcpow-n*: (∗*f*∗ *sin*) (*hypreal-of-hypnat n* ∗ *a*) = *hIm*(*hcis a hcpow n*)
by (*simp add*: *NSDeMoivre-ext*)

lemma *hexpi-add*: !!*a b*. *hexpi*(*a* + *b*) = *hexpi*(*a*) ∗ *hexpi*(*b*)
by (*transfer*, *rule expi-add*)

36.15 *star-of*: the Injection from type *complex* to to *hcomplex*

lemma *inj-hcomplex-of-complex*: *inj*(*hcomplex-of-complex*)
by (*rule inj-onI*, *simp*)

lemma *hcomplex-of-complex-i*: *iii* = *hcomplex-of-complex ii*
by (*simp add*: *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*)

36.16 Numerals and Arithmetic

lemma *hcomplex-number-of-def*: (*number-of w* :: *hcomplex*) == *of-int* (*Rep-Bin 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 complex-number-of* [*symmetric*])

This theorem is necessary because theorems such as *iszero-number-of-0* only hold for ordered rings. They cannot be generalized to fields in general because they fail for finite fields. They work for type complex because the reals can be embedded in them.

lemma *iszero-hcomplex-number-of* [*simp*]:
iszero (*number-of w* :: *hcomplex*) = *iszero* (*number-of w* :: *real*)
by (*transfer iszero-def*, *simp*)

lemma *hcomplex-number-of-hcnj* [*simp*]:
hcnj (*number-of v* :: *hcomplex*) = *number-of v*
by (*transfer*, *rule complex-number-of-cnj*)

lemma *hcomplex-number-of-hcmod* [*simp*]:
hcmod(*number-of v* :: *hcomplex*) = *abs* (*number-of v* :: *hypreal*)
by (*transfer*, *rule complex-number-of-cmod*)

lemma *hcomplex-number-of-hRe* [*simp*]:
hRe(*number-of v* :: *hcomplex*) = *number-of v*

by (*transfer*, *simp*)

lemma *hcomplex-number-of-hIm* [*simp*]:
  hIm(*number-of v* :: *hcomplex*) = *0*
by (*transfer*, *simp*)

ML
⟪
val iii-def = *thmiii-def*;

val hRe = *thmhRe*;
val hIm = *thmhIm*;
val hcomplex-hRe-hIm-cancel-iff = *thmhcomplex-hRe-hIm-cancel-iff*;
val hcomplex-hRe-zero = *thmhcomplex-hRe-zero*;
val hcomplex-hIm-zero = *thmhcomplex-hIm-zero*;
val hcomplex-hRe-one = *thmhcomplex-hRe-one*;
val hcomplex-hIm-one = *thmhcomplex-hIm-one*;
val inj-hcomplex-of-complex = *thminj-hcomplex-of-complex*;
val hcomplex-of-complex-i = *thmhcomplex-of-complex-i*;
val star-n-add = *thmstar-n-add*;
val hRe-add = *thmhRe-add*;
val hIm-add = *thmhIm-add*;
val hRe-minus = *thmhRe-minus*;
val hIm-minus = *thmhIm-minus*;
val hcomplex-add-minus-eq-minus = *thmhcomplex-add-minus-eq-minus*;
val hcomplex-diff-eq-eq = *thmhcomplex-diff-eq-eq*;
val hcomplex-mult-minus-one = *thmhcomplex-mult-minus-one*;
val hcomplex-mult-minus-one-right = *thmhcomplex-mult-minus-one-right*;
val hcomplex-mult-left-cancel = *thmhcomplex-mult-left-cancel*;
val hcomplex-mult-right-cancel = *thmhcomplex-mult-right-cancel*;
val hcomplex-add-divide-distrib = *thmhcomplex-add-divide-distrib*;
val hcomplex-of-hypreal = *thmhcomplex-of-hypreal*;
val hcomplex-of-hypreal-cancel-iff = *thmhcomplex-of-hypreal-cancel-iff*;
val hcomplex-of-hypreal-minus = *thmhcomplex-of-hypreal-minus*;
val hcomplex-of-hypreal-inverse = *thmhcomplex-of-hypreal-inverse*;
val hcomplex-of-hypreal-add = *thmhcomplex-of-hypreal-add*;
val hcomplex-of-hypreal-diff = *thmhcomplex-of-hypreal-diff*;
val hcomplex-of-hypreal-mult = *thmhcomplex-of-hypreal-mult*;
val hcomplex-of-hypreal-divide = *thmhcomplex-of-hypreal-divide*;
val hcomplex-of-hypreal-one = *thmhcomplex-of-hypreal-one*;
val hcomplex-of-hypreal-zero = *thmhcomplex-of-hypreal-zero*;
val hcomplex-of-hypreal-pow = *thmhcomplex-of-hypreal-pow*;
val hRe-hcomplex-of-hypreal = *thmhRe-hcomplex-of-hypreal*;
val hIm-hcomplex-of-hypreal = *thmhIm-hcomplex-of-hypreal*;
val hcomplex-of-hypreal-epsilon-not-zero = *thmhcomplex-of-hypreal-epsilon-not-zero*;
val hcmod = *thmhcmod*;
val hcmod-zero = *thmhcmod-zero*;
val hcmod-one = *thmhcmod-one*;

val hcmod-hcomplex-of-hypreal = thmhcmod-hcomplex-of-hypreal;
val hcomplex-of-hypreal-abs = thmhcomplex-of-hypreal-abs;
val hcnj = thmhcnj;
val hcomplex-hcnj-cancel-iff = thmhcomplex-hcnj-cancel-iff;
val hcomplex-hcnj-hcnj = thmhcomplex-hcnj-hcnj;
val hcomplex-hcnj-hcomplex-of-hypreal = thmhcomplex-hcnj-hcomplex-of-hypreal;
val hcomplex-hmod-hcnj = thmhcomplex-hmod-hcnj;
val hcomplex-hcnj-minus = thmhcomplex-hcnj-minus;
val hcomplex-hcnj-inverse = thmhcomplex-hcnj-inverse;
val hcomplex-hcnj-add = thmhcomplex-hcnj-add;
val hcomplex-hcnj-diff = thmhcomplex-hcnj-diff;
val hcomplex-hcnj-mult = thmhcomplex-hcnj-mult;
val hcomplex-hcnj-divide = thmhcomplex-hcnj-divide;
val hcnj-one = thmhcnj-one;
val hcomplex-hcnj-pow = thmhcomplex-hcnj-pow;
val hcomplex-hcnj-zero = thmhcomplex-hcnj-zero;
val hcomplex-hcnj-zero-iff = thmhcomplex-hcnj-zero-iff;
val hcomplex-mult-hcnj = thmhcomplex-mult-hcnj;
val hcomplex-hcmod-eq-zero-cancel = thmhcomplex-hcmod-eq-zero-cancel;

val hcmod-hcomplex-of-hypreal-of-nat = thmhcmod-hcomplex-of-hypreal-of-nat;
val hcmod-hcomplex-of-hypreal-of-hypnat = thmhcmod-hcomplex-of-hypreal-of-hypnat;
val hcmod-minus = thmhcmod-minus;
val hcmod-mult-hcnj = thmhcmod-mult-hcnj;
val hcmod-ge-zero = thmhcmod-ge-zero;
val hrabs-hcmod-cancel = thmhrabs-hcmod-cancel;
val hcmod-mult = thmhcmod-mult;
val hcmod-add-squared-eq = thmhcmod-add-squared-eq;
val hcomplex-hRe-mult-hcnj-le-hcmod = thmhcomplex-hRe-mult-hcnj-le-hcmod;
val hcomplex-hRe-mult-hcnj-le-hcmod2 = thmhcomplex-hRe-mult-hcnj-le-hcmod2;
val hcmod-triangle-squared = thmhcmod-triangle-squared;
val hcmod-triangle-ineq = thmhcmod-triangle-ineq;
val hcmod-triangle-ineq2 = thmhcmod-triangle-ineq2;
val hcmod-diff-commute = thmhcmod-diff-commute;
val hcmod-add-less = thmhcmod-add-less;
val hcmod-mult-less = thmhcmod-mult-less;
val hcmod-diff-ineq = thmhcmod-diff-ineq;
val hcpow = thmhcpow;
val hcomplex-of-hypreal-hyperpow = thmhcomplex-of-hypreal-hyperpow;
val hcmod-hcomplexpow = thmhcmod-hcomplexpow;
val hcmod-hcpow = thmhcmod-hcpow;
val hcpow-minus = thmhcpow-minus;
val hcmod-hcomplex-inverse = thmhcmod-hcomplex-inverse;
val hcmod-divide = thmhcmod-divide;
val hcpow-mult = thmhcpow-mult;
val hcpow-zero = thmhcpow-zero;
val hcpow-zero2 = thmhcpow-zero2;
val hcpow-not-zero = thmhcpow-not-zero;
val hcpow-zero-zero = thmhcpow-zero-zero;

val hcomplex-i-mult-eq = thmhcomplex-i-mult-eq;
val hcomplexpow-i-squared = thmhcomplexpow-i-squared;
val hcomplex-i-not-zero = thmhcomplex-i-not-zero;
val star-n-divide = thmstar-n-divide;
val hsgn = thmhsgn;
val hsgn-zero = thmhsgn-zero;
val hsgn-one = thmhsgn-one;
val hsgn-minus = thmhsgn-minus;
val hsgn-eq = thmhsgn-eq;
val lemma-hypreal-P-EX2 = thmlemma-hypreal-P-EX2;
val hcmod-i = thmhcmod-i;
val hcomplex-eq-cancel-iff2 = thmhcomplex-eq-cancel-iff2;
val hRe-hsgn = thmhRe-hsgn;
val hIm-hsgn = thmhIm-hsgn;
val real-two-squares-add-zero-iff = thmreal-two-squares-add-zero-iff;
val hRe-mult-i-eq = thmhRe-mult-i-eq;
val hIm-mult-i-eq = thmhIm-mult-i-eq;
val hcmod-mult-i = thmhcmod-mult-i;
val hcmod-mult-i2 = thmhcmod-mult-i2;
val harg = thmharg;
val cos-harg-i-mult-zero = thmcos-harg-i-mult-zero;
val hcomplex-of-hypreal-zero-iff = thmhcomplex-of-hypreal-zero-iff;
val complex-split-polar2 = thmcomplex-split-polar2;
val hcomplex-split-polar = thmhcomplex-split-polar;
val hcis = thmhcis;
val hcis-eq = thmhcis-eq;
val hrcis = thmhrcis;
val hrcis-Ex = thmhrcis-Ex;
val hRe-hcomplex-polar = thmhRe-hcomplex-polar;
val hRe-hrcis = thmhRe-hrcis;
val hIm-hcomplex-polar = thmhIm-hcomplex-polar;
val hIm-hrcis = thmhIm-hrcis;
val hcmod-complex-polar = thmhcmod-complex-polar;
val hcmod-hrcis = thmhcmod-hrcis;
val hcis-hrcis-eq = thmhcis-hrcis-eq;
val hrcis-mult = thmhrcis-mult;
val hcis-mult = thmhcis-mult;
val hcis-zero = thmhcis-zero;
val hrcis-zero-mod = thmhrcis-zero-mod;
val hrcis-zero-arg = thmhrcis-zero-arg;
val hcomplex-i-mult-minus = thmhcomplex-i-mult-minus;
val hcomplex-i-mult-minus2 = thmhcomplex-i-mult-minus2;
val hcis-hypreal-of-nat-Suc-mult = thmhcis-hypreal-of-nat-Suc-mult;
val NSDeMoivre = thmNSDeMoivre;
val hcis-hypreal-of-hypnat-Suc-mult = thmhcis-hypreal-of-hypnat-Suc-mult;
val NSDeMoivre-ext = thmNSDeMoivre-ext;
val DeMoivre2 = thmDeMoivre2;
val DeMoivre2-ext = thmDeMoivre2-ext;
val hcis-inverse = thmhcis-inverse;

val hrcis-inverse = thmhrcis-inverse;
val hRe-hcis = thmhRe-hcis;
val hIm-hcis = thmhIm-hcis;
val cos-n-hRe-hcis-pow-n = thmcos-n-hRe-hcis-pow-n;
val sin-n-hIm-hcis-pow-n = thmsin-n-hIm-hcis-pow-n;
val cos-n-hRe-hcis-hcpow-n = thmcos-n-hRe-hcis-hcpow-n;
val sin-n-hIm-hcis-hcpow-n = thmsin-n-hIm-hcis-hcpow-n;
val hexpi-add = thmhexpi-add;
val hRe-hcomplex-of-complex = thmhRe-hcomplex-of-complex;
val hIm-hcomplex-of-complex = thmhIm-hcomplex-of-complex;
val hcmod-hcomplex-of-complex = thmhcmod-hcomplex-of-complex;
⟩

end

37 NSCA: Non-Standard Complex Analysis

theory *NSCA*
imports *NSComplex*
begin

constdefs

CInfinitesimal :: *hcomplex set*
CInfinitesimal == {*x*. ∀ *r* ∈ *Reals*. *0* < *r* −−> *hcmod x* < *r*}

capprox :: [*hcomplex*,*hcomplex*] => *bool* (**infixl** @*c*= *50*)
— the "infinitely close" relation
x @*c*= *y* == (*x* − *y*) ∈ *CInfinitesimal*

SComplex :: *hcomplex set*
SComplex == {*x*. ∃ *r*. *x* = *hcomplex-of-complex r*}

CFinite :: *hcomplex set*
CFinite == {*x*. ∃ *r* ∈ *Reals*. *hcmod x* < *r*}

CInfinite :: *hcomplex set*
CInfinite == {*x*. ∀ *r* ∈ *Reals*. *r* < *hcmod x*}

stc :: *hcomplex* => *hcomplex*
— standard part map
stc x == (@*r*. *x* ∈ *CFinite* & *r*:*SComplex* & *r* @*c*= *x*)

cmonad :: *hcomplex* => *hcomplex set*
cmonad x == {*y*. *x* @*c*= *y*}

cgalaxy :: *hcomplex* => *hcomplex set*

cgalaxy x == {y. (x − y) ∈ CFinite}

37.1 Closure Laws for SComplex, the Standard Complex Numbers

lemma *SComplex-add*: [| *x ∈ SComplex*; *y ∈ SComplex* |] ==> *x + y ∈ SComplex*
apply (*simp add*: *SComplex-def*, *safe*)
apply (*rule-tac x = r + ra* **in** *exI*, *simp*)
done

lemma *SComplex-mult*: [| *x ∈ SComplex*; *y ∈ SComplex* |] ==> *x * y ∈ SComplex*
apply (*simp add*: *SComplex-def*, *safe*)
apply (*rule-tac x = r * ra* **in** *exI*, *simp*)
done

lemma *SComplex-inverse*: *x ∈ SComplex* ==> *inverse x ∈ SComplex*
apply (*simp add*: *SComplex-def*)
apply (*blast intro*: *star-of-inverse* [*symmetric*])
done

lemma *SComplex-divide*: [| *x ∈ SComplex*; *y ∈ SComplex* |] ==> *x/y ∈ SComplex*
by (*simp add*: *SComplex-mult SComplex-inverse divide-inverse*)

lemma *SComplex-minus*: *x ∈ SComplex* ==> *−x ∈ SComplex*
apply (*simp add*: *SComplex-def*)
apply (*blast intro*: *star-of-minus* [*symmetric*])
done

lemma *SComplex-minus-iff* [*simp*]: *(−x ∈ SComplex) = (x ∈ SComplex)*
apply *auto*
apply (*erule-tac* [*2*] *SComplex-minus*)
apply (*drule SComplex-minus*, *auto*)
done

lemma *SComplex-diff*: [| *x ∈ SComplex*; *y ∈ SComplex* |] ==> *x − y ∈ SComplex*
by (*simp add*: *diff-minus SComplex-add*)

lemma *SComplex-add-cancel*:
[| *x + y ∈ SComplex*; *y ∈ SComplex* |] ==> *x ∈ SComplex*
by (*drule SComplex-diff*, *assumption*, *simp*)

lemma *SReal-hcmod-hcomplex-of-complex* [*simp*]:
hcmod (hcomplex-of-complex r) ∈ Reals
by (*auto simp add*: *hcmod SReal-def star-of-def*)

lemma *SReal-hcmod-number-of* [*simp*]: *hcmod (number-of w ::hcomplex) ∈ Reals*
apply (*subst star-of-number-of* [*symmetric*])
apply (*rule SReal-hcmod-hcomplex-of-complex*)
done

lemma *SReal-hcmod-SComplex*: $x \in SComplex \Longrightarrow hcmod\ x \in Reals$
by (*auto simp add*: *SComplex-def*)

lemma *SComplex-hcomplex-of-complex* [*simp*]: *hcomplex-of-complex x* $\in$ *SComplex*
by (*simp add*: *SComplex-def*)

lemma *SComplex-number-of* [*simp*]: (*number-of w* ::*hcomplex*) $\in$ *SComplex*
apply (*subst star-of-number-of* [*symmetric*])
apply (*rule SComplex-hcomplex-of-complex*)
done

lemma *SComplex-divide-number-of*:
$r \in SComplex \Longrightarrow r/(number\text{-}of\ w{::}hcomplex) \in SComplex$
apply (*simp only*: *divide-inverse*)
apply (*blast intro*!: *SComplex-number-of SComplex-mult SComplex-inverse*)
done

lemma *SComplex-UNIV-complex*:
$\{x.\ hcomplex\text{-}of\text{-}complex\ x \in SComplex\} = (UNIV{::}complex\ set)$
by (*simp add*: *SComplex-def*)

lemma *SComplex-iff*: $(x \in SComplex) = (\exists y.\ x = hcomplex\text{-}of\text{-}complex\ y)$
by (*simp add*: *SComplex-def*)

lemma *hcomplex-of-complex-image*:
$hcomplex\text{-}of\text{-}complex\ `(UNIV{::}complex\ set) = SComplex$
by (*auto simp add*: *SComplex-def*)

lemma *inv-hcomplex-of-complex-image*: *inv hcomplex-of-complex* `*SComplex* = *UNIV*
apply (*auto simp add*: *SComplex-def*)
apply (*rule inj-hcomplex-of-complex* [*THEN inv-f-f*, *THEN subst*], *blast*)
done

lemma *SComplex-hcomplex-of-complex-image*:
$[|\ \exists x.\ x{:}\ P;\ P \leq SComplex\ |] \Longrightarrow \exists Q.\ P = hcomplex\text{-}of\text{-}complex\ `\ Q$
apply (*simp add*: *SComplex-def*, *blast*)
done

lemma *SComplex-SReal-dense*:
$[|\ x \in SComplex;\ y \in SComplex;\ hcmod\ x < hcmod\ y\ |] \Longrightarrow \exists r \in Reals.\ hcmod\ x < r\ \&\ r < hcmod\ y$
apply (*auto intro*: *SReal-dense simp add*: *SReal-hcmod-SComplex*)
done

lemma *SComplex-hcmod-SReal*:
$z \in SComplex \Longrightarrow hcmod\ z \in Reals$
by (*auto simp add*: *SComplex-def SReal-def hcmod-def*)

lemma *SComplex-zero* [*simp*]: *0* ∈ *SComplex*
by (*simp add*: *SComplex-def*)

lemma *SComplex-one* [*simp*]: *1* ∈ *SComplex*
by (*simp add*: *SComplex-def*)

37.2 The Finite Elements form a Subring

lemma *CFinite-add*: [|*x* ∈ *CFinite*; *y* ∈ *CFinite*|] ==> (*x*+*y*) ∈ *CFinite*
apply (*simp add*: *CFinite-def*)
apply (*blast intro*!: *SReal-add hcmod-add-less*)
done

lemma *CFinite-mult*: [|*x* ∈ *CFinite*; *y* ∈ *CFinite*|] ==> *x*∗*y* ∈ *CFinite*
apply (*simp add*: *CFinite-def*)
apply (*blast intro*!: *SReal-mult hcmod-mult-less*)
done

lemma *CFinite-minus-iff* [*simp*]: (−*x* ∈ *CFinite*) = (*x* ∈ *CFinite*)
by (*simp add*: *CFinite-def*)

lemma *SComplex-subset-CFinite* [*simp*]: *SComplex* ≤ *CFinite*
apply (*auto simp add*: *SComplex-def CFinite-def*)
apply (*rule-tac x* = *1* + *hcmod* (*hcomplex-of-complex r*) **in** *bexI*)
apply (*auto intro*: *SReal-add*)
done

lemma *HFinite-hcmod-hcomplex-of-complex* [*simp*]:
hcmod (*hcomplex-of-complex r*) ∈ *HFinite*
by (*auto intro*!: *SReal-subset-HFinite* [*THEN subsetD*])

lemma *CFinite-hcomplex-of-complex* [*simp*]: *hcomplex-of-complex x* ∈ *CFinite*
by (*auto intro*!: *SComplex-subset-CFinite* [*THEN subsetD*])

lemma *CFiniteD*: *x* ∈ *CFinite* ==> ∃ *t* ∈ *Reals*. *hcmod x* < *t*
by (*simp add*: *CFinite-def*)

lemma *CFinite-hcmod-iff*: (*x* ∈ *CFinite*) = (*hcmod x* ∈ *HFinite*)
by (*simp add*: *CFinite-def HFinite-def*)

lemma *CFinite-number-of* [*simp*]: *number-of w* ∈ *CFinite*
by (*rule SComplex-number-of* [*THEN SComplex-subset-CFinite* [*THEN subsetD*]])

lemma *CFinite-bounded*: [|*x* ∈ *CFinite*; *y* ≤ *hcmod x*; *0* ≤ *y* |] ==> *y*: *HFinite*
by (*auto intro*: *HFinite-bounded simp add*: *CFinite-hcmod-iff*)

37.3 The Complex Infinitesimals form a Subring

lemma *CInfinitesimal-zero* [*iff*]: *0* ∈ *CInfinitesimal*
by (*simp add*: *CInfinitesimal-def*)

lemma *hcomplex-sum-of-halves*: $x/(2::hcomplex) + x/(2::hcomplex) = x$
by *auto*

lemma *CInfinitesimal-hcmod-iff*:
$(z \in CInfinitesimal) = (hcmod\ z \in Infinitesimal)$
by (*simp add*: *CInfinitesimal-def Infinitesimal-def*)

lemma *one-not-CInfinitesimal* [*simp*]: $1 \notin CInfinitesimal$
by (*simp add*: *CInfinitesimal-hcmod-iff*)

lemma *CInfinitesimal-add*:
$[|\ x \in CInfinitesimal;\ y \in CInfinitesimal\ |] ==> (x+y) \in CInfinitesimal$
apply (*auto simp add*: *CInfinitesimal-hcmod-iff*)
apply (*rule hrabs-le-Infinitesimal*)
apply (*rule-tac* $y = hcmod\ y$ **in** *Infinitesimal-add*, *auto*)
done

lemma *CInfinitesimal-minus-iff* [*simp*]:
$(-x:CInfinitesimal) = (x:CInfinitesimal)$
by (*simp add*: *CInfinitesimal-def*)

lemma *CInfinitesimal-diff*:
$[|\ x \in CInfinitesimal;\ \ y \in CInfinitesimal\ |] ==> x-y \in CInfinitesimal$
by (*simp add*: *diff-minus CInfinitesimal-add*)

lemma *CInfinitesimal-mult*:
$[|\ x \in CInfinitesimal;\ y \in CInfinitesimal\ |] ==> x * y \in CInfinitesimal$
by (*auto intro*: *Infinitesimal-mult simp add*: *CInfinitesimal-hcmod-iff hcmod-mult*)

lemma *CInfinitesimal-CFinite-mult*:
$[|\ x \in CInfinitesimal;\ y \in CFinite\ |] ==> (x * y) \in CInfinitesimal$
by (*auto intro*: *Infinitesimal-HFinite-mult simp add*: *CInfinitesimal-hcmod-iff CFinite-hcmod-iff hcmod-mult*)

lemma *CInfinitesimal-CFinite-mult2*:
$[|\ x \in CInfinitesimal;\ y \in CFinite\ |] ==> (y * x) \in CInfinitesimal$
by (*auto dest*: *CInfinitesimal-CFinite-mult simp add*: *mult-commute*)

lemma *CInfinite-hcmod-iff*: $(z \in CInfinite) = (hcmod\ z \in HInfinite)$
by (*simp add*: *CInfinite-def HInfinite-def*)

lemma *CInfinite-inverse-CInfinitesimal*:
$x \in CInfinite ==> inverse\ x \in CInfinitesimal$
by (*auto intro*: *HInfinite-inverse-Infinitesimal simp add*: *CInfinitesimal-hcmod-iff CInfinite-hcmod-iff hcmod-hcomplex-inverse*)

lemma *CInfinite-mult*: $[|x \in CInfinite;\ y \in CInfinite|] ==> (x*y)$: *CInfinite*
by (*auto intro*: *HInfinite-mult simp add*: *CInfinite-hcmod-iff hcmod-mult*)

lemma *CInfinite-minus-iff* [*simp*]: $(-x \in CInfinite) = (x \in CInfinite)$
by (*simp add*: *CInfinite-def*)

lemma *CFinite-sum-squares*:
$[|a \in CFinite;\ b \in CFinite;\ c \in CFinite|]$
$\Longrightarrow a*a + b*b + c*c \in CFinite$
by (*auto intro*: *CFinite-mult CFinite-add*)

lemma *not-CInfinitesimal-not-zero*: $x \notin CInfinitesimal \Longrightarrow x \neq 0$
by *auto*

lemma *not-CInfinitesimal-not-zero2*: $x \in CFinite - CInfinitesimal \Longrightarrow x \neq 0$
by *auto*

lemma *CFinite-diff-CInfinitesimal-hcmod*:
$x \in CFinite - CInfinitesimal \Longrightarrow hcmod\ x \in HFinite - Infinitesimal$
by (*simp add*: *CFinite-hcmod-iff CInfinitesimal-hcmod-iff*)

lemma *hcmod-less-CInfinitesimal*:
$[|\ e \in CInfinitesimal;\ hcmod\ x < hcmod\ e\ |] \Longrightarrow x \in CInfinitesimal$
by (*auto intro*: *hrabs-less-Infinitesimal simp add*: *CInfinitesimal-hcmod-iff*)

lemma *hcmod-le-CInfinitesimal*:
$[|\ e \in CInfinitesimal;\ hcmod\ x \leq hcmod\ e\ |] \Longrightarrow x \in CInfinitesimal$
by (*auto intro*: *hrabs-le-Infinitesimal simp add*: *CInfinitesimal-hcmod-iff*)

lemma *CInfinitesimal-interval*:
$[|\ e \in CInfinitesimal;$
$e' \in CInfinitesimal;$
$hcmod\ e' < hcmod\ x\ ;\ hcmod\ x < hcmod\ e$
$|] \Longrightarrow x \in CInfinitesimal$
by (*auto intro*: *Infinitesimal-interval simp add*: *CInfinitesimal-hcmod-iff*)

lemma *CInfinitesimal-interval2*:
$[|\ e \in CInfinitesimal;$
$e' \in CInfinitesimal;$
$hcmod\ e' \leq hcmod\ x\ ;\ hcmod\ x \leq hcmod\ e$
$|] \Longrightarrow x \in CInfinitesimal$
by (*auto intro*: *Infinitesimal-interval2 simp add*: *CInfinitesimal-hcmod-iff*)

lemma *not-CInfinitesimal-mult*:
$[|\ x \notin CInfinitesimal;\ \ y \notin CInfinitesimal|] \Longrightarrow (x*y) \notin CInfinitesimal$
apply (*auto simp add*: *CInfinitesimal-hcmod-iff hcmod-mult*)
apply (*drule not-Infinitesimal-mult*, *auto*)
done

lemma *CInfinitesimal-mult-disj*:
$x*y \in CInfinitesimal \Longrightarrow x \in CInfinitesimal \mid y \in CInfinitesimal$

by (*auto dest*: *Infinitesimal-mult-disj simp add*: *CInfinitesimal-hcmod-iff hcmod-mult*)

lemma *CFinite-CInfinitesimal-diff-mult*:
[| *x* ∈ *CFinite* − *CInfinitesimal*; *y* ∈ *CFinite* − *CInfinitesimal* |]
==> *x*∗*y* ∈ *CFinite* − *CInfinitesimal*
by (*blast dest*: *CFinite-mult not-CInfinitesimal-mult*)

lemma *CInfinitesimal-subset-CFinite*: *CInfinitesimal* ≤ *CFinite*
by (*auto intro*: *Infinitesimal-subset-HFinite* [*THEN subsetD*]
simp add: *CInfinitesimal-hcmod-iff CFinite-hcmod-iff*)

lemma *CInfinitesimal-hcomplex-of-complex-mult*:
x ∈ *CInfinitesimal* ==> *x* ∗ *hcomplex-of-complex r* ∈ *CInfinitesimal*
by (*auto intro*!: *Infinitesimal-HFinite-mult simp add*: *CInfinitesimal-hcmod-iff hcmod-mult*)

lemma *CInfinitesimal-hcomplex-of-complex-mult2*:
x ∈ *CInfinitesimal* ==> *hcomplex-of-complex r* ∗ *x* ∈ *CInfinitesimal*
by (*auto intro*!: *Infinitesimal-HFinite-mult2 simp add*: *CInfinitesimal-hcmod-iff hcmod-mult*)

37.4 The "Infinitely Close" Relation

lemma *mem-cinfmal-iff*: *x*:*CInfinitesimal* = (*x* @*c*= *0*)
by (*simp add*: *CInfinitesimal-hcmod-iff capprox-def*)

lemma *capprox-minus-iff*: (*x* @*c*= *y*) = (*x* + −*y* @*c*= *0*)
by (*simp add*: *capprox-def diff-minus*)

lemma *capprox-minus-iff2*: (*x* @*c*= *y*) = (−*y* + *x* @*c*= *0*)
by (*simp add*: *capprox-def diff-minus add-commute*)

lemma *capprox-refl* [*simp*]: *x* @*c*= *x*
by (*simp add*: *capprox-def*)

lemma *capprox-sym*: *x* @*c*= *y* ==> *y* @*c*= *x*
by (*simp add*: *capprox-def CInfinitesimal-def hcmod-diff-commute*)

lemma *capprox-trans*: [| *x* @*c*= *y*; *y* @*c*= *z* |] ==> *x* @*c*= *z*
apply (*simp add*: *capprox-def*)
apply (*drule CInfinitesimal-add*, *assumption*)
apply (*simp add*: *diff-minus*)
done

lemma *capprox-trans2*: [| *r* @*c*= *x*; *s* @*c*= *x* |] ==> *r* @*c*= *s*
by (*blast intro*: *capprox-sym capprox-trans*)

lemma *capprox-trans3*: [| *x* @*c*= *r*; *x* @*c*= *s*|] ==> *r* @*c*= *s*
by (*blast intro*: *capprox-sym capprox-trans*)

lemma *number-of-capprox-reorient* [*simp*]:
(*number-of w* @*c*= *x*) = (*x* @*c*= *number-of w*)
by (*blast intro*: *capprox-sym*)

lemma *CInfinitesimal-capprox-minus*: (*x*−*y* ∈ *CInfinitesimal*) = (*x* @*c*= *y*)
by (*simp add*: *diff-minus capprox-minus-iff* [*symmetric*] *mem-cinfmal-iff*)

lemma *capprox-monad-iff*: (*x* @*c*= *y*) = (*cmonad*(*x*)=*cmonad*(*y*))
by (*auto simp add*: *cmonad-def dest*: *capprox-sym elim*!: *capprox-trans equalityCE*)

lemma *Infinitesimal-capprox*:
[| *x* ∈ *CInfinitesimal*; *y* ∈ *CInfinitesimal* |] ==> *x* @*c*= *y*
apply (*simp add*: *mem-cinfmal-iff*)
apply (*blast intro*: *capprox-trans capprox-sym*)
done

lemma *capprox-add*: [| *a* @*c*= *b*; *c* @*c*= *d* |] ==> *a*+*c* @*c*= *b*+*d*
apply (*simp add*: *capprox-def diff-minus*)
apply (*rule minus-add-distrib* [*THEN ssubst*])
apply (*rule add-assoc* [*THEN ssubst*])
apply (*rule-tac b1* = *c* **in** *add-left-commute* [*THEN subst*])
apply (*rule add-assoc* [*THEN subst*])
apply (*blast intro*: *CInfinitesimal-add*)
done

lemma *capprox-minus*: *a* @*c*= *b* ==> −*a* @*c*= −*b*
apply (*rule capprox-minus-iff* [*THEN iffD2*, *THEN capprox-sym*])
apply (*drule capprox-minus-iff* [*THEN iffD1*])
apply (*simp add*: *add-commute*)
done

lemma *capprox-minus2*: −*a* @*c*= −*b* ==> *a* @*c*= *b*
by (*auto dest*: *capprox-minus*)

lemma *capprox-minus-cancel* [*simp*]: (−*a* @*c*= −*b*) = (*a* @*c*= *b*)
by (*blast intro*: *capprox-minus capprox-minus2*)

lemma *capprox-add-minus*: [| *a* @*c*= *b*; *c* @*c*= *d* |] ==> *a* + −*c* @*c*= *b* + −*d*
by (*blast intro*!: *capprox-add capprox-minus*)

lemma *capprox-mult1*:
[| *a* @*c*= *b*; *c* ∈ *CFinite*|] ==> *a*∗*c* @*c*= *b*∗*c*
apply (*simp add*: *capprox-def diff-minus*)
apply (*simp only*: *CInfinitesimal-CFinite-mult minus-mult-left left-distrib* [*symmetric*])
done

lemma *capprox-mult2*: [|*a* @*c*= *b*; *c* ∈ *CFinite*|] ==> *c*∗*a* @*c*= *c*∗*b*
by (*simp add*: *capprox-mult1 mult-commute*)

lemma *capprox-mult-subst*:
[|*u* @*c*= *v*∗*x*; *x* @*c*= *y*; *v* ∈ *CFinite*|] ==> *u* @*c*= *v*∗*y*
by (*blast intro*: *capprox-mult2 capprox-trans*)

lemma *capprox-mult-subst2*:
[| *u* @*c*= *x*∗*v*; *x* @*c*= *y*; *v* ∈ *CFinite* |] ==> *u* @*c*= *y*∗*v*
by (*blast intro*: *capprox-mult1 capprox-trans*)

lemma *capprox-mult-subst-SComplex*:
[| *u* @*c*= *x*∗*hcomplex-of-complex v*; *x* @*c*= *y* |]
==> *u* @*c*= *y*∗*hcomplex-of-complex v*
by (*auto intro*: *capprox-mult-subst2*)

lemma *capprox-eq-imp*: *a* = *b* ==> *a* @*c*= *b*
by (*simp add*: *capprox-def*)

lemma *CInfinitesimal-minus-capprox*: *x* ∈ *CInfinitesimal* ==> −*x* @*c*= *x*
by (*blast intro*: *CInfinitesimal-minus-iff* [*THEN iffD2*] *mem-cinfmal-iff* [*THEN iffD1*] *capprox-trans2*)

lemma *bex-CInfinitesimal-iff*: (∃ *y* ∈ *CInfinitesimal*. *x* − *z* = *y*) = (*x* @*c*= *z*)
by (*unfold capprox-def*, *blast*)

lemma *bex-CInfinitesimal-iff2*: (∃ *y* ∈ *CInfinitesimal*. *x* = *z* + *y*) = (*x* @*c*= *z*)
by (*simp add*: *bex-CInfinitesimal-iff* [*symmetric*], *force*)

lemma *CInfinitesimal-add-capprox*:
[| *y* ∈ *CInfinitesimal*; *x* + *y* = *z* |] ==> *x* @*c*= *z*
apply (*rule bex-CInfinitesimal-iff* [*THEN iffD1*])
apply (*drule CInfinitesimal-minus-iff* [*THEN iffD2*])
apply (*simp add*: *eq-commute compare-rls*)
done

lemma *CInfinitesimal-add-capprox-self*: *y* ∈ *CInfinitesimal* ==> *x* @*c*= *x* + *y*
apply (*rule bex-CInfinitesimal-iff* [*THEN iffD1*])
apply (*drule CInfinitesimal-minus-iff* [*THEN iffD2*])
apply (*simp add*: *eq-commute compare-rls*)
done

lemma *CInfinitesimal-add-capprox-self2*: *y* ∈ *CInfinitesimal* ==> *x* @*c*= *y* + *x*
by (*auto dest*: *CInfinitesimal-add-capprox-self simp add*: *add-commute*)

lemma *CInfinitesimal-add-minus-capprox-self*:
y ∈ *CInfinitesimal* ==> *x* @*c*= *x* + −*y*
by (*blast intro*!: *CInfinitesimal-add-capprox-self CInfinitesimal-minus-iff* [*THEN iffD2*])

lemma *CInfinitesimal-add-cancel*:
[| *y* ∈ *CInfinitesimal*; *x*+*y* @*c*= *z*|] ==> *x* @*c*= *z*

apply (*drule-tac x = x* **in** *CInfinitesimal-add-capprox-self* [*THEN capprox-sym*])
apply (*erule capprox-trans3* [*THEN capprox-sym*], *assumption*)
done

lemma *CInfinitesimal-add-right-cancel*:
[| *y* ∈ *CInfinitesimal*; *x* @*c*= *z* + *y*|] ==> *x* @*c*= *z*
apply (*drule-tac x = z* **in** *CInfinitesimal-add-capprox-self2* [*THEN capprox-sym*])
apply (*erule capprox-trans3* [*THEN capprox-sym*])
apply (*simp add*: *add-commute*)
apply (*erule capprox-sym*)
done

lemma *capprox-add-left-cancel*: *d* + *b* @*c*= *d* + *c* ==> *b* @*c*= *c*
apply (*drule capprox-minus-iff* [*THEN iffD1*])
apply (*simp add*: *minus-add-distrib capprox-minus-iff* [*symmetric*] *add-ac*)
done

lemma *capprox-add-right-cancel*: *b* + *d* @*c*= *c* + *d* ==> *b* @*c*= *c*
apply (*rule capprox-add-left-cancel*)
apply (*simp add*: *add-commute*)
done

lemma *capprox-add-mono1*: *b* @*c*= *c* ==> *d* + *b* @*c*= *d* + *c*
apply (*rule capprox-minus-iff* [*THEN iffD2*])
apply (*simp add*: *capprox-minus-iff* [*symmetric*] *add-ac*)
done

lemma *capprox-add-mono2*: *b* @*c*= *c* ==> *b* + *a* @*c*= *c* + *a*
apply (*simp* (*no-asm-simp*) *add*: *add-commute capprox-add-mono1*)
done

lemma *capprox-add-left-iff* [*iff*]: (*a* + *b* @*c*= *a* + *c*) = (*b* @*c*= *c*)
by (*fast elim*: *capprox-add-left-cancel capprox-add-mono1*)

lemma *capprox-add-right-iff* [*iff*]: (*b* + *a* @*c*= *c* + *a*) = (*b* @*c*= *c*)
by (*simp add*: *add-commute*)

lemma *capprox-CFinite*: [| *x* ∈ *CFinite*; *x* @*c*= *y* |] ==> *y* ∈ *CFinite*
apply (*drule bex-CInfinitesimal-iff2* [*THEN iffD2*], *safe*)
apply (*drule CInfinitesimal-subset-CFinite* [*THEN subsetD*, *THEN CFinite-minus-iff*
[*THEN iffD2*]])
apply (*drule CFinite-add*)
apply (*assumption*, *auto*)
done

lemma *capprox-hcomplex-of-complex-CFinite*:
x @*c*= *hcomplex-of-complex D* ==> *x* ∈ *CFinite*
by (*rule capprox-sym* [*THEN* [*2*] *capprox-CFinite*], *auto*)

lemma *capprox-mult-CFinite*:
[|*a* @*c*= *b*; *c* @*c*= *d*; *b* ∈ *CFinite*; *d* ∈ *CFinite*|] ==> *a*∗*c* @*c*= *b*∗*d*
apply (*rule capprox-trans*)
apply (*rule-tac* [*2*] *capprox-mult2*)
apply (*rule capprox-mult1*)
prefer *2* **apply** (*blast intro*: *capprox-CFinite capprox-sym*, *auto*)
done

lemma *capprox-mult-hcomplex-of-complex*:
[|*a* @*c*= *hcomplex-of-complex b*; *c* @*c*= *hcomplex-of-complex d* |]
==> *a*∗*c* @*c*= *hcomplex-of-complex b* ∗ *hcomplex-of-complex d*
apply (*blast intro*!: *capprox-mult-CFinite capprox-hcomplex-of-complex-CFinite CFinite-hcomplex-of-complex*)
done

lemma *capprox-SComplex-mult-cancel-zero*:
[| *a* ∈ *SComplex*; *a* ≠ *0*; *a*∗*x* @*c*= *0* |] ==> *x* @*c*= *0*
apply (*drule SComplex-inverse* [*THEN SComplex-subset-CFinite* [*THEN subsetD*]])
apply (*auto dest*: *capprox-mult2 simp add*: *mult-assoc* [*symmetric*])
done

lemma *capprox-mult-SComplex1*: [| *a* ∈ *SComplex*; *x* @*c*= *0* |] ==> *x*∗*a* @*c*= *0*
by (*auto dest*: *SComplex-subset-CFinite* [*THEN subsetD*] *capprox-mult1*)

lemma *capprox-mult-SComplex2*: [| *a* ∈ *SComplex*; *x* @*c*= *0* |] ==> *a*∗*x* @*c*= *0*
by (*auto dest*: *SComplex-subset-CFinite* [*THEN subsetD*] *capprox-mult2*)

lemma *capprox-mult-SComplex-zero-cancel-iff* [*simp*]:
[|*a* ∈ *SComplex*; *a* ≠ *0* |] ==> (*a*∗*x* @*c*= *0*) = (*x* @*c*= *0*)
by (*blast intro*: *capprox-SComplex-mult-cancel-zero capprox-mult-SComplex2*)

lemma *capprox-SComplex-mult-cancel*:
[| *a* ∈ *SComplex*; *a* ≠ *0*; *a*∗ *w* @*c*= *a*∗*z* |] ==> *w* @*c*= *z*
apply (*drule SComplex-inverse* [*THEN SComplex-subset-CFinite* [*THEN subsetD*]])
apply (*auto dest*: *capprox-mult2 simp add*: *mult-assoc* [*symmetric*])
done

lemma *capprox-SComplex-mult-cancel-iff1* [*simp*]:
[| *a* ∈ *SComplex*; *a* ≠ *0*|] ==> (*a*∗ *w* @*c*= *a*∗*z*) = (*w* @*c*= *z*)
by (*auto intro*!: *capprox-mult2 SComplex-subset-CFinite* [*THEN subsetD*]
intro: *capprox-SComplex-mult-cancel*)

lemma *capprox-hcmod-approx-zero*: (*x* @*c*= *y*) = (*hcmod* (*y* − *x*) @= *0*)
apply (*rule capprox-minus-iff* [*THEN ssubst*])
apply (*simp add*: *capprox-def CInfinitesimal-hcmod-iff mem-infmal-iff diff-minus*
[*symmetric*] *hcmod-diff-commute*)
done

lemma *capprox-approx-zero-iff*: (*x* @*c*= *0*) = (*hcmod x* @= *0*)
by (*simp add*: *capprox-hcmod-approx-zero*)

lemma *capprox-minus-zero-cancel-iff* [*simp*]: $(-x \ @c= \ 0) = (x \ @c= \ 0)$
by (*simp add*: *capprox-hcmod-approx-zero*)

lemma *Infinitesimal-hcmod-add-diff*:
$u \ @c= \ 0 \Longrightarrow hcmod(x + u) - hcmod \ x \in Infinitesimal$
apply (*rule-tac* $e = hcmod \ u$ **and** $e' = - \ hcmod \ u$ **in** *Infinitesimal-interval2*)
apply (*auto dest*: *capprox-approx-zero-iff* [*THEN iffD1*]
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*: $u \ @c= \ 0 \Longrightarrow hcmod(x + u) \ @= \ hcmod \ x$
apply (*rule approx-minus-iff* [*THEN iffD2*])
apply (*auto intro*: *Infinitesimal-hcmod-add-diff simp add*: *mem-infmal-iff* [*symmetric*]
diff-minus [*symmetric*])
done

lemma *capprox-hcmod-approx*: $x \ @c= \ y \Longrightarrow hcmod \ x \ @= \ hcmod \ y$
by (*auto intro*: *approx-hcmod-add-hcmod*
dest!: *bex-CInfinitesimal-iff2* [*THEN iffD2*]
simp add: *mem-cinfmal-iff*)

37.5 Zero is the Only Infinitesimal Complex Number

lemma *CInfinitesimal-less-SComplex*:
$[\!| x \in SComplex; \ y \in CInfinitesimal; \ 0 < hcmod \ x \ |\!] \Longrightarrow hcmod \ y < hcmod \ x$
by (*auto intro*!: *Infinitesimal-less-SReal SComplex-hcmod-SReal simp add*: *CInfinitesimal-hcmod-iff*)

lemma *SComplex-Int-CInfinitesimal-zero*: $SComplex \ Int \ CInfinitesimal = \{0\}$
apply (*auto simp add*: *SComplex-def CInfinitesimal-hcmod-iff*)
apply (*cut-tac* $r = r$ **in** *SReal-hcmod-hcomplex-of-complex*)
apply (*drule-tac* $A = Reals$ **in** *IntI*, *assumption*)
apply (*subgoal-tac hcmod* (*hcomplex-of-complex r*) = *0*)
apply *simp*
apply (*simp add*: *SReal-Int-Infinitesimal-zero*)
done

lemma *SComplex-CInfinitesimal-zero*:
$[\!| x \in SComplex; \ x \in CInfinitesimal |\!] \Longrightarrow x = 0$
by (*cut-tac SComplex-Int-CInfinitesimal-zero*, *blast*)

lemma *SComplex-CFinite-diff-CInfinitesimal*:
$[\!| x \in SComplex; \ x \neq 0 \ |\!] \Longrightarrow x \in CFinite - CInfinitesimal$
by (*auto dest*: *SComplex-CInfinitesimal-zero SComplex-subset-CFinite* [*THEN subsetD*])

lemma *hcomplex-of-complex-CFinite-diff-CInfinitesimal*:

hcomplex-of-complex x ≠ 0
==> hcomplex-of-complex x ∈ CFinite − CInfinitesimal
by (*rule SComplex-CFinite-diff-CInfinitesimal*, *auto*)

lemma *hcomplex-of-complex-CInfinitesimal-iff-0* [*iff*]:
(hcomplex-of-complex x ∈ CInfinitesimal) = (x=0)
apply (*auto*)
apply (*rule ccontr*)
apply (*rule hcomplex-of-complex-CFinite-diff-CInfinitesimal* [*THEN DiffD2*], *auto*)
done

lemma *number-of-not-CInfinitesimal* [*simp*]:
number-of w ≠ (0::hcomplex) ==> number-of w ∉ CInfinitesimal
by (*fast dest*: *SComplex-number-of* [*THEN SComplex-CInfinitesimal-zero*])

lemma *capprox-SComplex-not-zero*:
[| y ∈ SComplex; x @c= y; y≠ 0 |] ==> x ≠ 0
by (*auto dest*: *SComplex-CInfinitesimal-zero capprox-sym* [*THEN mem-cinfmal-iff* [*THEN iffD2*]])

lemma *CFinite-diff-CInfinitesimal-capprox*:
[| x @c= y; y ∈ CFinite − CInfinitesimal |]
==> x ∈ CFinite − CInfinitesimal
apply (*auto intro*: *capprox-sym* [*THEN* [*2*] *capprox-CFinite*]
simp add: *mem-cinfmal-iff*)
apply (*drule capprox-trans3*, *assumption*)
apply (*blast dest*: *capprox-sym*)
done

lemma *CInfinitesimal-ratio*:
[| y ≠ 0; y ∈ CInfinitesimal; x/y ∈ CFinite |] ==> x ∈ CInfinitesimal
apply (*drule CInfinitesimal-CFinite-mult2*, *assumption*)
apply (*simp add*: *divide-inverse mult-assoc*)
done

lemma *SComplex-capprox-iff*:
[|x ∈ SComplex; y ∈ SComplex|] ==> (x @c= y) = (x = y)
apply *auto*
apply (*simp add*: *capprox-def*)
apply (*subgoal-tac x−y = 0*, *simp*)
apply (*rule SComplex-CInfinitesimal-zero*)
apply (*simp add*: *SComplex-diff*, *assumption*)
done

lemma *number-of-capprox-iff* [*simp*]:
(number-of v @c= number-of w) = (number-of v = (number-of w :: hcomplex))
by (*rule SComplex-capprox-iff*, *auto*)

lemma *number-of-CInfinitesimal-iff* [*simp*]:

(*number-of w* ∈ *CInfinitesimal*) = (*number-of w* = (*0*::*hcomplex*))
apply (*rule iffI*)
apply (*fast dest*: *SComplex-number-of* [*THEN SComplex-CInfinitesimal-zero*])
apply (*simp* (*no-asm-simp*))
done

lemma *hcomplex-of-complex-approx-iff* [*simp*]:
(*hcomplex-of-complex k* @*c*= *hcomplex-of-complex m*) = (*k* = *m*)
apply *auto*
apply (*rule inj-hcomplex-of-complex* [*THEN injD*])
apply (*rule SComplex-capprox-iff* [*THEN iffD1*], *auto*)
done

lemma *hcomplex-of-complex-capprox-number-of-iff* [*simp*]:
(*hcomplex-of-complex k* @*c*= *number-of w*) = (*k* = *number-of w*)
by (*subst hcomplex-of-complex-approx-iff* [*symmetric*], *auto*)

lemma *capprox-unique-complex*:
[| *r* ∈ *SComplex*; *s* ∈ *SComplex*; *r* @*c*= *x*; *s* @*c*= *x*|] ==> *r* = *s*
by (*blast intro*: *SComplex-capprox-iff* [*THEN iffD1*] *capprox-trans2*)

lemma *hcomplex-capproxD1*:
star-n X @*c*= *star-n Y*
==> *star-n* (%*n*. *Re*(*X n*)) @= *star-n* (%*n*. *Re*(*Y n*))
apply (*auto simp add*: *approx-FreeUltrafilterNat-iff*)
apply (*drule capprox-minus-iff* [*THEN iffD1*])
apply (*auto simp add*: *star-n-minus star-n-add mem-cinfmal-iff* [*symmetric*] *CInfinitesimal-hcmod-iff hcmod Infinitesimal-FreeUltrafilterNat-iff2*)
apply (*drule-tac x* = *m* **in** *spec*, *ultra*)
apply (*rename-tac Z x*)
apply (*case-tac X x*)
apply (*case-tac Y x*)
apply (*auto simp add*: *complex-minus complex-add complex-mod*
simp del: *realpow-Suc*)
apply (*rule-tac y*=*abs*(*Z x*) **in** *order-le-less-trans*)
apply (*drule-tac t* = *Z x* **in** *sym*)
apply (*auto simp del*: *realpow-Suc*)
done

lemma *hcomplex-capproxD2*:
star-n X @*c*= *star-n Y*
==> *star-n* (%*n*. *Im*(*X n*)) @= *star-n* (%*n*. *Im*(*Y n*))
apply (*auto simp add*: *approx-FreeUltrafilterNat-iff*)
apply (*drule capprox-minus-iff* [*THEN iffD1*])
apply (*auto simp add*: *star-n-minus star-n-add mem-cinfmal-iff* [*symmetric*] *CInfinitesimal-hcmod-iff hcmod Infinitesimal-FreeUltrafilterNat-iff2*)
apply (*drule-tac x* = *m* **in** *spec*, *ultra*)
apply (*rename-tac Z x*)

apply (*case-tac X x*)
apply (*case-tac Y x*)
apply (*auto simp add*: *complex-minus complex-add complex-mod simp del*: *realpow-Suc*)
apply (*rule-tac y=abs(Z x)* **in** *order-le-less-trans*)
apply (*drule-tac t = Z x* **in** *sym*)
apply (*auto simp del*: *realpow-Suc*)
done

lemma *hcomplex-capproxI*:
[| *star-n* (%*n*. *Re*(*X n*)) @= *star-n* (%*n*. *Re*(*Y n*));
star-n (%*n*. *Im*(*X n*)) @= *star-n* (%*n*. *Im*(*Y n*))
|] ==> *star-n X* @*c*= *star-n Y*
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*drule approx-minus-iff* [*THEN iffD1*])
apply (*rule capprox-minus-iff* [*THEN iffD2*])
apply (*auto simp add*: *mem-cinfmal-iff* [*symmetric*] *mem-infmal-iff* [*symmetric*]
star-n-add star-n-minus CInfinitesimal-hcmod-iff hcmod Infinitesimal-FreeUltrafilterNat-iff)
apply (*rule bexI* [*OF - Rep-star-star-n*], *auto*)
apply (*drule-tac x = u/2* **in** *spec*)
apply (*drule-tac x = u/2* **in** *spec*, *auto*, *ultra*)
apply (*drule sym*, *drule sym*)
apply (*case-tac X x*)
apply (*case-tac Y x*)
apply (*auto simp add*: *complex-minus complex-add complex-mod snd-conv fst-conv*
numeral-2-eq-2)
apply (*rename-tac a b c d*)
apply (*subgoal-tac sqrt* (*abs* (*a + - c*) ^ *2* + *abs* (*b + - d*) ^ *2*) < *u*)
apply (*rule-tac* [*2*] *lemma-sqrt-hcomplex-capprox*, *auto*)
apply (*simp add*: *power2-eq-square*)
done

lemma *capprox-approx-iff*:
(*star-n X* @*c*= *star-n Y*) =
(*star-n* (%*n*. *Re*(*X n*)) @= *star-n* (%*n*. *Re*(*Y n*)) &
star-n (%*n*. *Im*(*X n*)) @= *star-n* (%*n*. *Im*(*Y n*)))
apply (*blast intro*: *hcomplex-capproxI hcomplex-capproxD1 hcomplex-capproxD2*)
done

lemma *hcomplex-of-hypreal-capprox-iff* [*simp*]:
(*hcomplex-of-hypreal x* @*c*= *hcomplex-of-hypreal z*) = (*x* @= *z*)
apply (*cases x*, *cases z*)
apply (*simp add*: *hcomplex-of-hypreal capprox-approx-iff*)
done

lemma *CFinite-HFinite-Re*:
star-n X ∈ *CFinite*
==> *star-n* (%*n*. *Re*(*X n*)) ∈ *HFinite*
apply (*auto simp add*: *CFinite-hcmod-iff hcmod HFinite-FreeUltrafilterNat-iff*)
apply (*rule bexI* [*OF - Rep-star-star-n*])

apply (*rule-tac x = u* **in** *exI*, *ultra*)
apply (*drule sym*, *case-tac X x*)
apply (*auto simp add*: *complex-mod numeral-2-eq-2 simp del*: *realpow-Suc*)
apply (*rule ccontr*, *drule linorder-not-less* [*THEN iffD1*])
apply (*drule order-less-le-trans*, *assumption*)
apply (*drule real-sqrt-ge-abs1* [*THEN* [*2*] *order-less-le-trans*])
apply (*auto simp add*: *numeral-2-eq-2* [*symmetric*])
done

lemma *CFinite-HFinite-Im*:
star-n X ∈ CFinite
==> star-n (%n. Im(X n)) ∈ HFinite
apply (*auto simp add*: *CFinite-hcmod-iff hcmod HFinite-FreeUltrafilterNat-iff*)
apply (*rule bexI* [*OF - Rep-star-star-n*])
apply (*rule-tac x = u* **in** *exI*, *ultra*)
apply (*drule sym*, *case-tac X x*)
apply (*auto simp add*: *complex-mod simp del*: *realpow-Suc*)
apply (*rule ccontr*, *drule linorder-not-less* [*THEN iffD1*])
apply (*drule order-less-le-trans*, *assumption*)
apply (*drule real-sqrt-ge-abs2* [*THEN* [*2*] *order-less-le-trans*], *auto*)
done

lemma *HFinite-Re-Im-CFinite*:
[| *star-n (%n. Re(X n)) ∈ HFinite*;
star-n (%n. Im(X n)) ∈ HFinite
|] *==> star-n X ∈ CFinite*
apply (*auto simp add*: *CFinite-hcmod-iff hcmod HFinite-FreeUltrafilterNat-iff*)
apply (*rename-tac Y Z u v*)
apply (*rule bexI* [*OF - Rep-star-star-n*])
apply (*rule-tac x = 2* (u + v)* **in** *exI*)
apply *ultra*
apply (*drule sym*, *case-tac X x*)
apply (*auto simp add*: *complex-mod numeral-2-eq-2 simp del*: *realpow-Suc*)
apply (*subgoal-tac 0 < u*)
prefer *2* **apply** *arith*
apply (*subgoal-tac 0 < v*)
prefer *2* **apply** *arith*
apply (*subgoal-tac sqrt (abs (Y x) ^ 2 + abs (Z x) ^ 2) < 2*u + 2*v*)
apply (*rule-tac* [*2*] *lemma-sqrt-hcomplex-capprox*, *auto*)
apply (*simp add*: *power2-eq-square*)
done

lemma *CFinite-HFinite-iff*:
(*star-n X ∈ CFinite*) =
(*star-n (%n. Re(X n)) ∈ HFinite &*
star-n (%n. Im(X n)) ∈ HFinite)
by (*blast intro*: *HFinite-Re-Im-CFinite CFinite-HFinite-Im CFinite-HFinite-Re*)

lemma *SComplex-Re-SReal*:

star-n X ∈ SComplex
==> star-n (%n. Re(X n)) ∈ Reals
apply (*auto simp add*: *SComplex-def SReal-def star-of-def star-n-eq-iff*)
apply (*rule-tac x = Re r* **in** *exI*, *ultra*)
done

lemma *SComplex-Im-SReal*:
star-n X ∈ SComplex
==> star-n (%n. Im(X n)) ∈ Reals
apply (*auto simp add*: *SComplex-def SReal-def star-of-def star-n-eq-iff*)
apply (*rule-tac x = Im r* **in** *exI*, *ultra*)
done

lemma *Reals-Re-Im-SComplex*:
[| *star-n (%n. Re(X n)) ∈ Reals*;
star-n (%n. Im(X n)) ∈ Reals
|] *==> star-n X ∈ SComplex*
apply (*auto simp add*: *SComplex-def SReal-def star-of-def star-n-eq-iff*)
apply (*rule-tac x = Complex r ra* **in** *exI*, *ultra*)
done

lemma *SComplex-SReal-iff*:
(*star-n X ∈ SComplex*) =
(*star-n (%n. Re(X n)) ∈ Reals* &
star-n (%n. Im(X n)) ∈ Reals)
by (*blast intro*: *SComplex-Re-SReal SComplex-Im-SReal Reals-Re-Im-SComplex*)

lemma *CInfinitesimal-Infinitesimal-iff*:
(*star-n X ∈ CInfinitesimal*) =
(*star-n (%n. Re(X n)) ∈ Infinitesimal* &
star-n (%n. Im(X n)) ∈ Infinitesimal)
by (*simp add*: *mem-cinfmal-iff mem-infmal-iff star-n-zero-num capprox-approx-iff*)

lemma *eq-Abs-star-EX*:
(∃ *t*. *P t*) = (∃ *X*. *P* (*star-n X*))
by (*rule ex-star-eq*)

lemma *eq-Abs-star-Bex*:
(∃ *t* ∈ *A*. *P t*) = (∃ *X*. *star-n X ∈ A* & *P* (*star-n X*))
by (*simp add*: *Bex-def ex-star-eq*)

lemma *stc-part-Ex*: *x*:*CFinite ==>* ∃ *t* ∈ *SComplex*. *x* @*c*= *t*
apply (*cases x*)
apply (*auto simp add*: *CFinite-HFinite-iff eq-Abs-star-Bex SComplex-SReal-iff capprox-approx-iff*)
apply (*drule st-part-Ex*, *safe*)+
apply (*rule-tac x = t* **in** *star-cases*)
apply (*rule-tac x = ta* **in** *star-cases*, *auto*)
apply (*rule-tac x = %n. Complex* (*Xa n*) (*Xb n*) **in** *exI*)

apply *auto*
done

lemma *stc-part-Ex1*: *x*:*CFinite* ==> *EX*! *t*. *t* ∈ *SComplex* & *x* @c= *t*
apply (*drule stc-part-Ex*, *safe*)
apply (*drule-tac* [*2*] *capprox-sym*, *drule-tac* [*2*] *capprox-sym*, *drule-tac* [*2*] *capprox-sym*)
apply (*auto intro*!: *capprox-unique-complex*)
done

lemma *CFinite-Int-CInfinite-empty*: *CFinite Int CInfinite* = {}
by (*simp add*: *CFinite-def CInfinite-def*, *auto*)

lemma *CFinite-not-CInfinite*: *x* ∈ *CFinite* ==> *x* ∉ *CInfinite*
by (*insert CFinite-Int-CInfinite-empty*, *blast*)

Not sure this is a good idea!

declare *CFinite-Int-CInfinite-empty* [*simp*]

lemma *not-CFinite-CInfinite*: *x*∉ *CFinite* ==> *x* ∈ *CInfinite*
by (*auto intro*: *not-HFinite-HInfinite simp add*: *CFinite-hcmod-iff CInfinite-hcmod-iff*)

lemma *CInfinite-CFinite-disj*: *x* ∈ *CInfinite* | *x* ∈ *CFinite*
by (*blast intro*: *not-CFinite-CInfinite*)

lemma *CInfinite-CFinite-iff*: (*x* ∈ *CInfinite*) = (*x* ∉ *CFinite*)
by (*blast dest*: *CFinite-not-CInfinite not-CFinite-CInfinite*)

lemma *CFinite-CInfinite-iff*: (*x* ∈ *CFinite*) = (*x* ∉ *CInfinite*)
by (*simp add*: *CInfinite-CFinite-iff*)

lemma *CInfinite-diff-CFinite-CInfinitesimal-disj*:
x ∉ *CInfinitesimal* ==> *x* ∈ *CInfinite* | *x* ∈ *CFinite* − *CInfinitesimal*
by (*fast intro*: *not-CFinite-CInfinite*)

lemma *CFinite-inverse*:
[| *x* ∈ *CFinite*; *x* ∉ *CInfinitesimal* |] ==> *inverse x* ∈ *CFinite*
apply (*cut-tac x* = *inverse x* **in** *CInfinite-CFinite-disj*)
apply (*auto dest*!: *CInfinite-inverse-CInfinitesimal*)
done

lemma *CFinite-inverse2*: *x* ∈ *CFinite* − *CInfinitesimal* ==> *inverse x* ∈ *CFinite*
by (*blast intro*: *CFinite-inverse*)

lemma *CInfinitesimal-inverse-CFinite*:
x ∉ *CInfinitesimal* ==> *inverse*(*x*) ∈ *CFinite*
apply (*drule CInfinite-diff-CFinite-CInfinitesimal-disj*)
apply (*blast intro*: *CFinite-inverse CInfinite-inverse-CInfinitesimal CInfinitesimal-subset-CFinite* [*THEN subsetD*])
done

lemma *CFinite-not-CInfinitesimal-inverse*:
  $x \in CFinite - CInfinitesimal \Longrightarrow inverse\ x \in CFinite - CInfinitesimal$
apply (*auto intro*: *CInfinitesimal-inverse-CFinite*)
apply (*drule CInfinitesimal-CFinite-mult2*, *assumption*)
apply (*simp add*: *not-CInfinitesimal-not-zero*)
done

lemma *capprox-inverse*:
  $[|\ x\ @c=\ y;\ y \in\ CFinite - CInfinitesimal\ |] \Longrightarrow inverse\ x\ @c=\ inverse\ y$
apply (*frule CFinite-diff-CInfinitesimal-capprox*, *assumption*)
apply (*frule not-CInfinitesimal-not-zero2*)
apply (*frule-tac x = x* **in** *not-CInfinitesimal-not-zero2*)
apply (*drule CFinite-inverse2*)+
apply (*drule capprox-mult2*, *assumption*, *auto*)
apply (*drule-tac c = inverse x* **in** *capprox-mult1*, *assumption*)
apply (*auto intro*: *capprox-sym simp add*: *mult-assoc*)
done

lemmas *hcomplex-of-complex-capprox-inverse =*
  hcomplex-of-complex-CFinite-diff-CInfinitesimal [*THEN* [*2*] *capprox-inverse*]

lemma *inverse-add-CInfinitesimal-capprox*:
  $[|\ x \in CFinite - CInfinitesimal;$
    $h \in CInfinitesimal\ |] \Longrightarrow inverse(x + h)\ @c=\ inverse\ x$
by (*auto intro*: *capprox-inverse capprox-sym CInfinitesimal-add-capprox-self*)

lemma *inverse-add-CInfinitesimal-capprox2*:
  $[|\ x \in CFinite - CInfinitesimal;$
    $h \in CInfinitesimal\ |] \Longrightarrow inverse(h + x)\ @c=\ inverse\ x$
apply (*rule add-commute* [*THEN subst*])
apply (*blast intro*: *inverse-add-CInfinitesimal-capprox*)
done

lemma *inverse-add-CInfinitesimal-approx-CInfinitesimal*:
  $[|\ x \in CFinite - CInfinitesimal;$
    $h \in CInfinitesimal\ |] \Longrightarrow inverse(x + h) - inverse\ x\ @c=\ h$
apply (*rule capprox-trans2*)
apply (*auto intro*: *inverse-add-CInfinitesimal-capprox*
    simp add: *mem-cinfmal-iff diff-minus capprox-minus-iff* [*symmetric*])
done

lemma *CInfinitesimal-square-iff* [*iff*]:
  $(x*x \in CInfinitesimal) = (x \in CInfinitesimal)$
by (*simp add*: *CInfinitesimal-hcmod-iff hcmod-mult*)

lemma *capprox-CFinite-mult-cancel*:
  $[|\ a \in CFinite-CInfinitesimal;\ a*w\ @c=\ a*z\ |] \Longrightarrow w\ @c=\ z$

apply *safe*
apply (*frule CFinite-inverse*, *assumption*)
apply (*drule not-CInfinitesimal-not-zero*)
apply (*auto dest*: *capprox-mult2 simp add*: *mult-assoc* [*symmetric*])
done

lemma *capprox-CFinite-mult-cancel-iff1*:
$a \in CFinite - CInfinitesimal \Longrightarrow (a * w \ @c= a * z) = (w \ @c= z)$
by (*auto intro*: *capprox-mult2 capprox-CFinite-mult-cancel*)

37.6 Theorems About Monads

lemma *capprox-cmonad-iff*: $(x \ @c= y) = (cmonad(x) = cmonad(y))$
apply (*simp add*: *cmonad-def*)
apply (*auto dest*: *capprox-sym elim*!: *capprox-trans equalityCE*)
done

lemma *CInfinitesimal-cmonad-eq*:
$e \in CInfinitesimal \Longrightarrow cmonad\ (x+e) = cmonad\ x$
by (*fast intro*!: *CInfinitesimal-add-capprox-self* [*THEN capprox-sym*] *capprox-cmonad-iff* [*THEN iffD1*])

lemma *mem-cmonad-iff*: $(u \in cmonad\ x) = (-u \in cmonad\ (-x))$
by (*simp add*: *cmonad-def*)

lemma *CInfinitesimal-cmonad-zero-iff*: $(x{:}CInfinitesimal) = (x \in cmonad\ 0)$
by (*auto intro*: *capprox-sym simp add*: *mem-cinfmal-iff cmonad-def*)

lemma *cmonad-zero-minus-iff*: $(x \in cmonad\ 0) = (-x \in cmonad\ 0)$
by (*simp add*: *CInfinitesimal-cmonad-zero-iff* [*symmetric*])

lemma *cmonad-zero-hcmod-iff*: $(x \in cmonad\ 0) = (hcmod\ x{:}monad\ 0)$
by (*simp add*: *CInfinitesimal-cmonad-zero-iff* [*symmetric*] *CInfinitesimal-hcmod-iff* *Infinitesimal-monad-zero-iff* [*symmetric*])

lemma *mem-cmonad-self* [*simp*]: $x \in cmonad\ x$
by (*simp add*: *cmonad-def*)

37.7 Theorems About Standard Part

lemma *stc-capprox-self*: $x \in CFinite \Longrightarrow stc\ x \ @c= x$
apply (*simp add*: *stc-def*)
apply (*frule stc-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *capprox-sym*)
done

lemma *stc-SComplex*: $x \in CFinite \Longrightarrow stc\ x \in SComplex$
apply (*simp add*: *stc-def*)
apply (*frule stc-part-Ex*, *safe*)

apply (*rule someI2*)
apply (*auto intro*: *capprox-sym*)
done

lemma *stc-CFinite*: $x \in CFinite \Longrightarrow stc\ x \in CFinite$
by (*erule stc-SComplex* [*THEN SComplex-subset-CFinite* [*THEN subsetD*]])

lemma *stc-SComplex-eq* [*simp*]: $x \in SComplex \Longrightarrow stc\ x = x$
apply (*simp add*: *stc-def*)
apply (*rule some-equality*)
apply (*auto intro*: *SComplex-subset-CFinite* [*THEN subsetD*])
apply (*blast dest*: *SComplex-capprox-iff* [*THEN iffD1*])
done

lemma *stc-hcomplex-of-complex*:
$stc\ (hcomplex\text{-}of\text{-}complex\ x) = hcomplex\text{-}of\text{-}complex\ x$
by *auto*

lemma *stc-eq-capprox*:
$[|\ x \in CFinite;\ y \in CFinite;\ stc\ x = stc\ y\ |] \Longrightarrow x\ @c=\ y$
by (*auto dest*!: *stc-capprox-self elim*!: *capprox-trans3*)

lemma *capprox-stc-eq*:
$[|\ x \in CFinite;\ y \in CFinite;\ x\ @c=\ y\ |] \Longrightarrow stc\ x = stc\ y$
by (*blast intro*: *capprox-trans capprox-trans2 SComplex-capprox-iff* [*THEN iffD1*]
dest: *stc-capprox-self stc-SComplex*)

lemma *stc-eq-capprox-iff*:
$[|\ x \in CFinite;\ y \in CFinite|] \Longrightarrow (x\ @c=\ y) = (stc\ x = stc\ y)$
by (*blast intro*: *capprox-stc-eq stc-eq-capprox*)

lemma *stc-CInfinitesimal-add-SComplex*:
$[|\ x \in SComplex;\ e \in CInfinitesimal\ |] \Longrightarrow stc(x + e) = x$
apply (*frule stc-SComplex-eq* [*THEN subst*])
prefer *2* **apply** *assumption*
apply (*frule SComplex-subset-CFinite* [*THEN subsetD*])
apply (*frule CInfinitesimal-subset-CFinite* [*THEN subsetD*])
apply (*drule stc-SComplex-eq*)
apply (*rule capprox-stc-eq*)
apply (*auto intro*: *CFinite-add simp add*: *CInfinitesimal-add-capprox-self* [*THEN capprox-sym*])
done

lemma *stc-CInfinitesimal-add-SComplex2*:
$[|\ x \in SComplex;\ e \in CInfinitesimal\ |] \Longrightarrow stc(e + x) = x$
apply (*rule add-commute* [*THEN subst*])
apply (*blast intro*!: *stc-CInfinitesimal-add-SComplex*)
done

lemma *CFinite-stc-CInfinitesimal-add*:
$x \in CFinite \Longrightarrow \exists e \in CInfinitesimal.\ x = stc(x) + e$
by (*blast dest*!: *stc-capprox-self* [*THEN capprox-sym*] *bex-CInfinitesimal-iff2* [*THEN iffD2*])

lemma *stc-add*:
$[\!| x \in CFinite; y \in CFinite |\!] \Longrightarrow stc\ (x + y) = stc(x) + stc(y)$
apply (*frule CFinite-stc-CInfinitesimal-add*)
apply (*frule-tac x = y* **in** *CFinite-stc-CInfinitesimal-add*, *safe*)
apply (*subgoal-tac stc* $(x + y) = stc\ ((stc\ x + e) + (stc\ y + ea))$)
apply (*drule-tac* [*2*] *sym*, *drule-tac* [*2*] *sym*)
prefer *2* **apply** *simp*
apply (*simp* (*no-asm-simp*) *add*: *add-ac*)
apply (*drule stc-SComplex*)+
apply (*drule SComplex-add*, *assumption*)
apply (*drule CInfinitesimal-add*, *assumption*)
apply (*rule add-assoc* [*THEN subst*])
apply (*blast intro*!: *stc-CInfinitesimal-add-SComplex2*)
done

lemma *stc-number-of* [*simp*]: *stc* (*number-of w*) = *number-of w*
by (*rule SComplex-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*: $y \in CFinite \Longrightarrow stc(-y) = -stc(y)$
apply (*frule CFinite-minus-iff* [*THEN iffD2*])
apply (*rule hcomplex-add-minus-eq-minus*)
apply (*drule stc-add* [*symmetric*], *assumption*)
apply (*simp add*: *add-commute*)
done

lemma *stc-diff*:
$[\!| x \in CFinite; y \in CFinite |\!] \Longrightarrow stc\ (x-y) = stc(x) - stc(y)$
apply (*simp add*: *diff-minus*)
apply (*frule-tac y1 = y* **in** *stc-minus* [*symmetric*])
apply (*drule-tac x1 = y* **in** *CFinite-minus-iff* [*THEN iffD2*])
apply (*auto intro*: *stc-add*)
done

lemma *lemma-stc-mult*:
$[\!| x \in CFinite; y \in CFinite;$
$e \in CInfinitesimal;$
$ea: CInfinitesimal |\!]$
$\Longrightarrow e*y + x*ea + e*ea: CInfinitesimal$

apply (*frule-tac* $x = e$ **and** $y = y$ **in** *CInfinitesimal-CFinite-mult*)
apply (*frule-tac* [*2*] $x = ea$ **and** $y = x$ **in** *CInfinitesimal-CFinite-mult*)
apply (*drule-tac* [*3*] *CInfinitesimal-mult*)
apply (*auto intro*: *CInfinitesimal-add simp add*: *add-ac mult-ac*)
done

lemma *stc-mult*:
[| $x \in$ *CFinite*; $y \in$ *CFinite* |]
==> $stc\ (x * y) = stc(x) * stc(y)$
apply (*frule CFinite-stc-CInfinitesimal-add*)
apply (*frule-tac* $x = y$ **in** *CFinite-stc-CInfinitesimal-add*, *safe*)
apply (*subgoal-tac* $stc\ (x * y) = stc\ ((stc\ x + e) * (stc\ y + ea))$)
apply (*drule-tac* [*2*] *sym*, *drule-tac* [*2*] *sym*)
prefer *2* **apply** *simp*
apply (*erule-tac* $V = x = stc\ x + e$ **in** *thin-rl*)
apply (*erule-tac* $V = y = stc\ y + ea$ **in** *thin-rl*)
apply (*simp add*: *left-distrib right-distrib*)
apply (*drule stc-SComplex*)+
apply (*simp* (*no-asm-use*) *add*: *add-assoc*)
apply (*rule stc-CInfinitesimal-add-SComplex*)
apply (*blast intro*!: *SComplex-mult*)
apply (*drule SComplex-subset-CFinite* [*THEN subsetD*])+
apply (*rule add-assoc* [*THEN subst*])
apply (*blast intro*!: *lemma-stc-mult*)
done

lemma *stc-CInfinitesimal*: $x \in$ *CInfinitesimal* ==> $stc\ x = 0$
apply (*rule stc-zero* [*THEN subst*])
apply (*rule capprox-stc-eq*)
apply (*auto intro*: *CInfinitesimal-subset-CFinite* [*THEN subsetD*]
simp add: *mem-cinfmal-iff* [*symmetric*])
done

lemma *stc-not-CInfinitesimal*: $stc(x) \neq 0$ ==> $x \notin$ *CInfinitesimal*
by (*fast intro*: *stc-CInfinitesimal*)

lemma *stc-inverse*:
[| $x \in$ *CFinite*; $stc\ x \neq 0$ |]
==> $stc(inverse\ x) = inverse\ (stc\ x)$
apply (*rule-tac c1* = $stc\ x$ **in** *hcomplex-mult-left-cancel* [*THEN iffD1*])
apply (*auto simp add*: *stc-mult* [*symmetric*] *stc-not-CInfinitesimal CFinite-inverse*)
apply (*subst right-inverse*, *auto*)
done

lemma *stc-divide* [*simp*]:
[| $x \in$ *CFinite*; $y \in$ *CFinite*; $stc\ y \neq 0$ |]
==> $stc(x/y) = (stc\ x)\ /\ (stc\ y)$
by (*simp add*: *divide-inverse stc-mult stc-not-CInfinitesimal CFinite-inverse stc-inverse*)

lemma *stc-idempotent* [*simp*]: $x \in CFinite \Longrightarrow stc(stc(x)) = stc(x)$
by (*blast intro*: *stc-CFinite stc-capprox-self capprox-stc-eq*)

lemma *CFinite-HFinite-hcomplex-of-hypreal*:
$z \in HFinite \Longrightarrow hcomplex\text{-}of\text{-}hypreal\ z \in CFinite$
apply (*cases z*)
apply (*simp add*: *hcomplex-of-hypreal CFinite-HFinite-iff star-n-zero-num* [*symmetric*])
done

lemma *SComplex-SReal-hcomplex-of-hypreal*:
$x \in Reals \Longrightarrow hcomplex\text{-}of\text{-}hypreal\ x \in SComplex$
by (*auto simp add*: *SReal-def SComplex-def hcomplex-of-hypreal-def*)

lemma *stc-hcomplex-of-hypreal*:
$z \in HFinite \Longrightarrow stc(hcomplex\text{-}of\text{-}hypreal\ z) = hcomplex\text{-}of\text{-}hypreal\ (st\ z)$
apply (*simp add*: *st-def stc-def*)
apply (*frule st-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *approx-sym*)
apply (*drule CFinite-HFinite-hcomplex-of-hypreal*)
apply (*frule stc-part-Ex*, *safe*)
apply (*rule someI2*)
apply (*auto intro*: *capprox-sym intro*!: *capprox-unique-complex dest*: *SComplex-SReal-hcomplex-of-hypreal*)
done

lemma *CInfinitesimal-hcnj-iff* [*simp*]:
$(hcnj\ z \in CInfinitesimal) = (z \in CInfinitesimal)$
by (*simp add*: *CInfinitesimal-hcmod-iff*)

lemma *CInfinite-HInfinite-iff*:
$(star\text{-}n\ X \in CInfinite) =$
$(star\text{-}n\ (\%n.\ Re(X\ n)) \in HInfinite\ |$
$star\text{-}n\ (\%n.\ Im(X\ n)) \in HInfinite)$
by (*simp add*: *CInfinite-CFinite-iff HInfinite-HFinite-iff CFinite-HFinite-iff*)

These theorems should probably be deleted

lemma *hcomplex-split-CInfinitesimal-iff*:
$(hcomplex\text{-}of\text{-}hypreal\ x + iii * hcomplex\text{-}of\text{-}hypreal\ y \in CInfinitesimal) =$
$(x \in Infinitesimal\ \&\ y \in Infinitesimal)$
apply (*cases x*, *cases y*)
apply (*simp add*: *iii-def star-of-def star-n-add star-n-mult hcomplex-of-hypreal CInfinitesimal-Infinitesimal-iff*)
done

lemma *hcomplex-split-CFinite-iff*:
$(hcomplex\text{-}of\text{-}hypreal\ x + iii * hcomplex\text{-}of\text{-}hypreal\ y \in CFinite) =$
$(x \in HFinite\ \&\ y \in HFinite)$
apply (*cases x*, *cases y*)

apply (*simp add*: *iii-def star-of-def star-n-add star-n-mult hcomplex-of-hypreal CFinite-HFinite-iff*)
done

lemma *hcomplex-split-SComplex-iff*:
(*hcomplex-of-hypreal x + iii * hcomplex-of-hypreal y ∈ SComplex*) =
(*x ∈ Reals & y ∈ Reals*)
apply (*cases x*, *cases y*)
apply (*simp add*: *iii-def star-of-def star-n-add star-n-mult hcomplex-of-hypreal SComplex-SReal-iff*)
done

lemma *hcomplex-split-CInfinite-iff*:
(*hcomplex-of-hypreal x + iii * hcomplex-of-hypreal y ∈ CInfinite*) =
(*x ∈ HInfinite | y ∈ HInfinite*)
apply (*cases x*, *cases y*)
apply (*simp add*: *iii-def star-of-def star-n-add star-n-mult hcomplex-of-hypreal CInfinite-HInfinite-iff*)
done

lemma *hcomplex-split-capprox-iff*:
(*hcomplex-of-hypreal x + iii * hcomplex-of-hypreal y @c=*
*hcomplex-of-hypreal x′ + iii * hcomplex-of-hypreal y′*) =
(*x @= x′ & y @= y′*)
apply (*cases x*, *cases y*, *cases x′*, *cases y′*)
apply (*simp add*: *iii-def star-of-def star-n-add star-n-mult hcomplex-of-hypreal capprox-approx-iff*)
done

lemma *complex-seq-to-hcomplex-CInfinitesimal*:
∀ *n*. *cmod* (*X n − x*) < *inverse* (*real* (*Suc n*)) ==>
star-n X − hcomplex-of-complex x ∈ CInfinitesimal
apply (*simp add*: *star-n-diff CInfinitesimal-hcmod-iff star-of-def Infinitesimal-FreeUltrafilterNat-iff hcmod*)
apply (*rule bexI*, *auto*)
apply (*auto dest*: *FreeUltrafilterNat-inverse-real-of-posnat FreeUltrafilterNat-all FreeUltrafilterNat-Int intro*: *order-less-trans FreeUltrafilterNat-subset*)
done

lemma *CInfinitesimal-hcomplex-of-hypreal-epsilon* [*simp*]:
hcomplex-of-hypreal epsilon ∈ CInfinitesimal
by (*simp add*: *CInfinitesimal-hcmod-iff*)

lemma *hcomplex-of-complex-approx-zero-iff* [*simp*]:
(*hcomplex-of-complex z @c= 0*) = (*z = 0*)
by (*simp add*: *star-of-zero* [*symmetric*] *del*: *star-of-zero*)

lemma *hcomplex-of-complex-approx-zero-iff2* [*simp*]:
(*0 @c= hcomplex-of-complex z*) = (*z = 0*)
by (*simp add*: *star-of-zero* [*symmetric*] *del*: *star-of-zero*)

ML

⟪
val SComplex-add = thm SComplex-add;
val SComplex-mult = thm SComplex-mult;
val SComplex-inverse = thm SComplex-inverse;
val SComplex-divide = thm SComplex-divide;
val SComplex-minus = thm SComplex-minus;
val SComplex-minus-iff = thm SComplex-minus-iff;
val SComplex-diff = thm SComplex-diff;
val SComplex-add-cancel = thm SComplex-add-cancel;
val SReal-hcmod-hcomplex-of-complex = thm SReal-hcmod-hcomplex-of-complex;
val SReal-hcmod-number-of = thm SReal-hcmod-number-of;
val SReal-hcmod-SComplex = thm SReal-hcmod-SComplex;
val SComplex-hcomplex-of-complex = thm SComplex-hcomplex-of-complex;
val SComplex-number-of = thm SComplex-number-of;
val SComplex-divide-number-of = thm SComplex-divide-number-of;
val SComplex-UNIV-complex = thm SComplex-UNIV-complex;
val SComplex-iff = thm SComplex-iff;
val hcomplex-of-complex-image = thm hcomplex-of-complex-image;
val inv-hcomplex-of-complex-image = thm inv-hcomplex-of-complex-image;
val SComplex-hcomplex-of-complex-image = thm SComplex-hcomplex-of-complex-image;
val SComplex-SReal-dense = thm SComplex-SReal-dense;
val SComplex-hcmod-SReal = thm SComplex-hcmod-SReal;
val SComplex-zero = thm SComplex-zero;
val SComplex-one = thm SComplex-one;
val CFinite-add = thm CFinite-add;
val CFinite-mult = thm CFinite-mult;
val CFinite-minus-iff = thm CFinite-minus-iff;
val SComplex-subset-CFinite = thm SComplex-subset-CFinite;
val HFinite-hcmod-hcomplex-of-complex = thm HFinite-hcmod-hcomplex-of-complex;
val CFinite-hcomplex-of-complex = thm CFinite-hcomplex-of-complex;
val CFiniteD = thm CFiniteD;
val CFinite-hcmod-iff = thm CFinite-hcmod-iff;
val CFinite-number-of = thm CFinite-number-of;
val CFinite-bounded = thm CFinite-bounded;
val CInfinitesimal-zero = thm CInfinitesimal-zero;
val hcomplex-sum-of-halves = thm hcomplex-sum-of-halves;
val CInfinitesimal-hcmod-iff = thm CInfinitesimal-hcmod-iff;
val one-not-CInfinitesimal = thm one-not-CInfinitesimal;
val CInfinitesimal-add = thm CInfinitesimal-add;
val CInfinitesimal-minus-iff = thm CInfinitesimal-minus-iff;
val CInfinitesimal-diff = thm CInfinitesimal-diff;
val CInfinitesimal-mult = thm CInfinitesimal-mult;
val CInfinitesimal-CFinite-mult = thm CInfinitesimal-CFinite-mult;
val CInfinitesimal-CFinite-mult2 = thm CInfinitesimal-CFinite-mult2;
val CInfinite-hcmod-iff = thm CInfinite-hcmod-iff;
val CInfinite-inverse-CInfinitesimal = thm CInfinite-inverse-CInfinitesimal;
val CInfinite-mult = thm CInfinite-mult;
val CInfinite-minus-iff = thm CInfinite-minus-iff;
val CFinite-sum-squares = thm CFinite-sum-squares;

val not-CInfinitesimal-not-zero = thm not-CInfinitesimal-not-zero;
val not-CInfinitesimal-not-zero2 = thm not-CInfinitesimal-not-zero2;
val CFinite-diff-CInfinitesimal-hcmod = thm CFinite-diff-CInfinitesimal-hcmod;
val hcmod-less-CInfinitesimal = thm hcmod-less-CInfinitesimal;
val hcmod-le-CInfinitesimal = thm hcmod-le-CInfinitesimal;
val CInfinitesimal-interval = thm CInfinitesimal-interval;
val CInfinitesimal-interval2 = thm CInfinitesimal-interval2;
val not-CInfinitesimal-mult = thm not-CInfinitesimal-mult;
val CInfinitesimal-mult-disj = thm CInfinitesimal-mult-disj;
val CFinite-CInfinitesimal-diff-mult = thm CFinite-CInfinitesimal-diff-mult;
val CInfinitesimal-subset-CFinite = thm CInfinitesimal-subset-CFinite;
val CInfinitesimal-hcomplex-of-complex-mult = thm CInfinitesimal-hcomplex-of-complex-mult;
val CInfinitesimal-hcomplex-of-complex-mult2 = thm CInfinitesimal-hcomplex-of-complex-mult2;
val mem-cinfmal-iff = thm mem-cinfmal-iff;
val capprox-minus-iff = thm capprox-minus-iff;
val capprox-minus-iff2 = thm capprox-minus-iff2;
val capprox-refl = thm capprox-refl;
val capprox-sym = thm capprox-sym;
val capprox-trans = thm capprox-trans;
val capprox-trans2 = thm capprox-trans2;
val capprox-trans3 = thm capprox-trans3;
val number-of-capprox-reorient = thm number-of-capprox-reorient;
val CInfinitesimal-capprox-minus = thm CInfinitesimal-capprox-minus;
val capprox-monad-iff = thm capprox-monad-iff;
val Infinitesimal-capprox = thm Infinitesimal-capprox;
val capprox-add = thm capprox-add;
val capprox-minus = thm capprox-minus;
val capprox-minus2 = thm capprox-minus2;
val capprox-minus-cancel = thm capprox-minus-cancel;
val capprox-add-minus = thm capprox-add-minus;
val capprox-mult1 = thm capprox-mult1;
val capprox-mult2 = thm capprox-mult2;
val capprox-mult-subst = thm capprox-mult-subst;
val capprox-mult-subst2 = thm capprox-mult-subst2;
val capprox-mult-subst-SComplex = thm capprox-mult-subst-SComplex;
val capprox-eq-imp = thm capprox-eq-imp;
val CInfinitesimal-minus-capprox = thm CInfinitesimal-minus-capprox;
val bex-CInfinitesimal-iff = thm bex-CInfinitesimal-iff;
val bex-CInfinitesimal-iff2 = thm bex-CInfinitesimal-iff2;
val CInfinitesimal-add-capprox = thm CInfinitesimal-add-capprox;
val CInfinitesimal-add-capprox-self = thm CInfinitesimal-add-capprox-self;
val CInfinitesimal-add-capprox-self2 = thm CInfinitesimal-add-capprox-self2;
val CInfinitesimal-add-minus-capprox-self = thm CInfinitesimal-add-minus-capprox-self;
val CInfinitesimal-add-cancel = thm CInfinitesimal-add-cancel;
val CInfinitesimal-add-right-cancel = thm CInfinitesimal-add-right-cancel;
val capprox-add-left-cancel = thm capprox-add-left-cancel;
val capprox-add-right-cancel = thm capprox-add-right-cancel;
val capprox-add-mono1 = thm capprox-add-mono1;
val capprox-add-mono2 = thm capprox-add-mono2;

val capprox-add-left-iff = thm capprox-add-left-iff;
val capprox-add-right-iff = thm capprox-add-right-iff;
val capprox-CFinite = thm capprox-CFinite;
val capprox-hcomplex-of-complex-CFinite = thm capprox-hcomplex-of-complex-CFinite;
val capprox-mult-CFinite = thm capprox-mult-CFinite;
val capprox-mult-hcomplex-of-complex = thm capprox-mult-hcomplex-of-complex;
val capprox-SComplex-mult-cancel-zero = thm capprox-SComplex-mult-cancel-zero;
val capprox-mult-SComplex1 = thm capprox-mult-SComplex1;
val capprox-mult-SComplex2 = thm capprox-mult-SComplex2;
val capprox-mult-SComplex-zero-cancel-iff = thm capprox-mult-SComplex-zero-cancel-iff;
val capprox-SComplex-mult-cancel = thm capprox-SComplex-mult-cancel;
val capprox-SComplex-mult-cancel-iff1 = thm capprox-SComplex-mult-cancel-iff1;
val capprox-hcmod-approx-zero = thm capprox-hcmod-approx-zero;
val capprox-approx-zero-iff = thm capprox-approx-zero-iff;
val capprox-minus-zero-cancel-iff = thm capprox-minus-zero-cancel-iff;
val Infinitesimal-hcmod-add-diff = thm Infinitesimal-hcmod-add-diff;
val approx-hcmod-add-hcmod = thm approx-hcmod-add-hcmod;
val capprox-hcmod-approx = thm capprox-hcmod-approx;
val CInfinitesimal-less-SComplex = thm CInfinitesimal-less-SComplex;
val SComplex-Int-CInfinitesimal-zero = thm SComplex-Int-CInfinitesimal-zero;
val SComplex-CInfinitesimal-zero = thm SComplex-CInfinitesimal-zero;
val SComplex-CFinite-diff-CInfinitesimal = thm SComplex-CFinite-diff-CInfinitesimal;
val hcomplex-of-complex-CFinite-diff-CInfinitesimal = thm hcomplex-of-complex-CFinite-diff-CInfinitesimal;
val hcomplex-of-complex-CInfinitesimal-iff-0 = thm hcomplex-of-complex-CInfinitesimal-iff-0;
val number-of-not-CInfinitesimal = thm number-of-not-CInfinitesimal;
val capprox-SComplex-not-zero = thm capprox-SComplex-not-zero;
val CFinite-diff-CInfinitesimal-capprox = thm CFinite-diff-CInfinitesimal-capprox;
val CInfinitesimal-ratio = thm CInfinitesimal-ratio;
val SComplex-capprox-iff = thm SComplex-capprox-iff;
val number-of-capprox-iff = thm number-of-capprox-iff;
val number-of-CInfinitesimal-iff = thm number-of-CInfinitesimal-iff;
val hcomplex-of-complex-approx-iff = thm hcomplex-of-complex-approx-iff;
val hcomplex-of-complex-capprox-number-of-iff = thm hcomplex-of-complex-capprox-number-of-iff;
val capprox-unique-complex = thm capprox-unique-complex;
val hcomplex-capproxD1 = thm hcomplex-capproxD1;
val hcomplex-capproxD2 = thm hcomplex-capproxD2;
val hcomplex-capproxI = thm hcomplex-capproxI;
val capprox-approx-iff = thm capprox-approx-iff;
val hcomplex-of-hypreal-capprox-iff = thm hcomplex-of-hypreal-capprox-iff;
val CFinite-HFinite-Re = thm CFinite-HFinite-Re;
val CFinite-HFinite-Im = thm CFinite-HFinite-Im;
val HFinite-Re-Im-CFinite = thm HFinite-Re-Im-CFinite;
val CFinite-HFinite-iff = thm CFinite-HFinite-iff;
val SComplex-Re-SReal = thm SComplex-Re-SReal;
val SComplex-Im-SReal = thm SComplex-Im-SReal;
val Reals-Re-Im-SComplex = thm Reals-Re-Im-SComplex;
val SComplex-SReal-iff = thm SComplex-SReal-iff;
val CInfinitesimal-Infinitesimal-iff = thm CInfinitesimal-Infinitesimal-iff;
val eq-Abs-star-Bex = thm eq-Abs-star-Bex;

val stc-part-Ex = thm stc-part-Ex;
val stc-part-Ex1 = thm stc-part-Ex1;
val CFinite-Int-CInfinite-empty = thm CFinite-Int-CInfinite-empty;
val CFinite-not-CInfinite = thm CFinite-not-CInfinite;
val not-CFinite-CInfinite = thm not-CFinite-CInfinite;
val CInfinite-CFinite-disj = thm CInfinite-CFinite-disj;
val CInfinite-CFinite-iff = thm CInfinite-CFinite-iff;
val CFinite-CInfinite-iff = thm CFinite-CInfinite-iff;
val CInfinite-diff-CFinite-CInfinitesimal-disj = thm CInfinite-diff-CFinite-CInfinitesimal-disj;
val CFinite-inverse = thm CFinite-inverse;
val CFinite-inverse2 = thm CFinite-inverse2;
val CInfinitesimal-inverse-CFinite = thm CInfinitesimal-inverse-CFinite;
val CFinite-not-CInfinitesimal-inverse = thm CFinite-not-CInfinitesimal-inverse;
val capprox-inverse = thm capprox-inverse;
val hcomplex-of-complex-capprox-inverse = thms hcomplex-of-complex-capprox-inverse;
val inverse-add-CInfinitesimal-capprox = thm inverse-add-CInfinitesimal-capprox;
val inverse-add-CInfinitesimal-capprox2 = thm inverse-add-CInfinitesimal-capprox2;
val inverse-add-CInfinitesimal-approx-CInfinitesimal = thm inverse-add-CInfinitesimal-approx-CInfinitesimal;
val CInfinitesimal-square-iff = thm CInfinitesimal-square-iff;
val capprox-CFinite-mult-cancel = thm capprox-CFinite-mult-cancel;
val capprox-CFinite-mult-cancel-iff1 = thm capprox-CFinite-mult-cancel-iff1;
val capprox-cmonad-iff = thm capprox-cmonad-iff;
val CInfinitesimal-cmonad-eq = thm CInfinitesimal-cmonad-eq;
val mem-cmonad-iff = thm mem-cmonad-iff;
val CInfinitesimal-cmonad-zero-iff = thm CInfinitesimal-cmonad-zero-iff;
val cmonad-zero-minus-iff = thm cmonad-zero-minus-iff;
val cmonad-zero-hcmod-iff = thm cmonad-zero-hcmod-iff;
val mem-cmonad-self = thm mem-cmonad-self;
val stc-capprox-self = thm stc-capprox-self;
val stc-SComplex = thm stc-SComplex;
val stc-CFinite = thm stc-CFinite;
val stc-SComplex-eq = thm stc-SComplex-eq;
val stc-hcomplex-of-complex = thm stc-hcomplex-of-complex;
val stc-eq-capprox = thm stc-eq-capprox;
val capprox-stc-eq = thm capprox-stc-eq;
val stc-eq-capprox-iff = thm stc-eq-capprox-iff;
val stc-CInfinitesimal-add-SComplex = thm stc-CInfinitesimal-add-SComplex;
val stc-CInfinitesimal-add-SComplex2 = thm stc-CInfinitesimal-add-SComplex2;
val CFinite-stc-CInfinitesimal-add = thm CFinite-stc-CInfinitesimal-add;
val stc-add = thm stc-add;
val stc-number-of = thm stc-number-of;
val stc-zero = thm stc-zero;
val stc-one = thm stc-one;
val stc-minus = thm stc-minus;
val stc-diff = thm stc-diff;
val lemma-stc-mult = thm lemma-stc-mult;
val stc-mult = thm stc-mult;
val stc-CInfinitesimal = thm stc-CInfinitesimal;
val stc-not-CInfinitesimal = thm stc-not-CInfinitesimal;

val stc-inverse = thm stc-inverse;
val stc-divide = thm stc-divide;
val stc-idempotent = thm stc-idempotent;
val CFinite-HFinite-hcomplex-of-hypreal = thm CFinite-HFinite-hcomplex-of-hypreal;
val SComplex-SReal-hcomplex-of-hypreal = thm SComplex-SReal-hcomplex-of-hypreal;
val stc-hcomplex-of-hypreal = thm stc-hcomplex-of-hypreal;
val CInfinitesimal-hcnj-iff = thm CInfinitesimal-hcnj-iff;
val CInfinite-HInfinite-iff = thm CInfinite-HInfinite-iff;
val hcomplex-split-CInfinitesimal-iff = thm hcomplex-split-CInfinitesimal-iff;
val hcomplex-split-CFinite-iff = thm hcomplex-split-CFinite-iff;
val hcomplex-split-SComplex-iff = thm hcomplex-split-SComplex-iff;
val hcomplex-split-CInfinite-iff = thm hcomplex-split-CInfinite-iff;
val hcomplex-split-capprox-iff = thm hcomplex-split-capprox-iff;
val complex-seq-to-hcomplex-CInfinitesimal = thm complex-seq-to-hcomplex-CInfinitesimal;
val CInfinitesimal-hcomplex-of-hypreal-epsilon = thm CInfinitesimal-hcomplex-of-hypreal-epsilon;
val hcomplex-of-complex-approx-zero-iff = thm hcomplex-of-complex-approx-zero-iff;
val hcomplex-of-complex-approx-zero-iff2 = thm hcomplex-of-complex-approx-zero-iff2;
⟫

end

38 CStar: Star-transforms in NSA, Extending Sets of Complex Numbers and Complex Functions

theory *CStar*
imports *NSCA*
begin

38.1 Properties of the *-Transform Applied to Sets of Reals

lemma *STARC-SComplex-subset*: $SComplex \subseteq *s* (UNIV :: complex\ set)$
by *simp*

lemma *STARC-hcomplex-of-complex-Int*:
$*s* X\ Int\ SComplex = hcomplex\text{-}of\text{-}complex\ `\ X$
by (*auto simp add*: *SComplex-def STAR-mem-iff*)

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*

38.2 Theorems about Nonstandard Extensions of Functions

lemma *cstarfun-if-eq*:
$w \neq hcomplex\text{-}of\text{-}complex\ x$
$\Longrightarrow (*f* (\lambda z.\ if\ z = x\ then\ a\ else\ g\ z))\ w = (*f* g)\ w$
apply (*cases w*)

apply (*simp add*: *star-of-def starfun star-n-eq-iff*, *ultra*)
done

lemma *starfun-capprox*:
($*f*$ *f*) (*hcomplex-of-complex a*) @*c*= *hcomplex-of-complex* (*f a*)
by *auto*

lemma *starfunC-hcpow*: ($*f*$ (%*z*. *z* ^ *n*)) *Z* = *Z hcpow hypnat-of-nat n*
apply (*cases Z*)
apply (*simp add*: *hcpow starfun hypnat-of-nat-eq*)
done

lemma *starfun-mult-CFinite-capprox*:
[| ($*f*$ *f*) *y* @*c*= *l*; ($*f*$ *g*) *y* @*c*= *m*; *l*: *CFinite*; *m*: *CFinite* |]
==> ($*f*$ (%*x*. *f x* * *g x*)) *y* @*c*= *l* * *m*
apply (*drule capprox-mult-CFinite*, *assumption*+)
apply (*auto intro*: *capprox-sym* [*THEN* [*2*] *capprox-CFinite*])
done

lemma *starfun-add-capprox*:
[| ($*f*$ *f*) *y* @*c*= *l*; ($*f*$ *g*) *y* @*c*= *m* |]
==> ($*f*$ (%*x*. *f x* + *g x*)) *y* @*c*= *l* + *m*
by (*auto intro*: *capprox-add*)

lemma *starfunCR-cmod*: $*f*$ *cmod* = *hcmod*
apply (*rule ext*)
apply (*rule-tac x* = *x* **in** *star-cases*)
apply (*simp add*: *starfun hcmod*)
done

38.3 Internal Functions - Some Redundancy With *f* Now

lemma *starfun-n-diff*:
($*fn*$ *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-diff*)
done

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*)))
apply (*cases x*, *cases z*)
apply (*auto simp add*: *starfun hIm hRe complex-Re-Im-cancel-iff star-n-eq-iff*, *ultra*+)
done

lemma *starfunC-approx-Re-Im-iff*:
(($*f*$ *f*) *x* @*c*= *z*) = ((($*f*$ (%*x*. *Re*(*f x*))) *x* @= *hRe* (*z*)) &
(($*f*$ (%*x*. *Im*(*f x*))) *x* @= *hIm* (*z*)))
apply (*cases x*, *cases z*)
apply (*simp add*: *starfun hIm hRe capprox-approx-iff*)
done

lemma *starfunC-Idfun-capprox*:
x @*c*= *hcomplex-of-complex a* ==> ($*f*$ (%*x*. *x*)) *x* @*c*= *hcomplex-of-complex a*
by *simp*

ML
⟪
val STARC-SComplex-subset = thm STARC-SComplex-subset;
val STARC-hcomplex-of-complex-Int = thm STARC-hcomplex-of-complex-Int;
val lemma-not-hcomplexA = thm lemma-not-hcomplexA;
val starfun-capprox = thm starfun-capprox;
val starfunC-hcpow = thm starfunC-hcpow;
val starfun-mult-CFinite-capprox = thm starfun-mult-CFinite-capprox;
val starfun-add-capprox = thm starfun-add-capprox;
val starfunCR-cmod = thm starfunCR-cmod;
val starfun-inverse-inverse = thm starfun-inverse-inverse;
val starfun-n-diff = thm starfun-n-diff;
val starfunC-eq-Re-Im-iff = thm starfunC-eq-Re-Im-iff;
val starfunC-approx-Re-Im-iff = thm starfunC-approx-Re-Im-iff;
val starfunC-Idfun-capprox = thm starfunC-Idfun-capprox;
⟫

end

39 CSeries: Finite Summation and Infinite Series for Complex Numbers

theory *CSeries*
imports *CStar*
begin

consts *sumc* :: [*nat*,*nat*,(*nat*=>*complex*)] => *complex*
primrec
sumc-0: *sumc m 0 f* = *0*
sumc-Suc: *sumc m* (*Suc n*) *f* = (*if n* < *m then 0 else sumc m n f* + *f*(*n*))

lemma *sumc-Suc-zero* [*simp*]: *sumc* (*Suc n*) *n f* = *0*

by (*induct n*, *auto*)

lemma *sumc-eq-bounds* [*simp*]: *sumc m m f = 0*
by (*induct m*, *auto*)

lemma *sumc-Suc-eq* [*simp*]: *sumc m (Suc m) f = f(m)*
by *auto*

lemma *sumc-add-lbound-zero* [*simp*]: *sumc (m+k) k f = 0*
by (*induct k*, *auto*)

lemma *sumc-add*: *sumc m n f + sumc m n g = sumc m n (%n. f n + g n)*
apply (*induct n*)
apply (*auto simp add*: *add-ac*)
done

lemma *sumc-mult*: *r * sumc m n f = sumc m n (%n. r * f n)*
apply (*induct n*, *auto*)
apply (*auto simp add*: *right-distrib*)
done

lemma *sumc-split-add* [*rule-format*]:
n < p --> sumc 0 n f + sumc n p f = sumc 0 p f
apply (*induct p*)
apply (*auto dest*!: *leI dest*: *le-anti-sym*)
done

lemma *sumc-split-add-minus*:
n < p ==> sumc 0 p f + - sumc 0 n f = sumc n p f
apply (*drule-tac f1 = f* **in** *sumc-split-add* [*symmetric*])
apply (*simp add*: *add-ac*)
done

lemma *sumc-cmod*: *cmod(sumc m n f) ≤ (∑ i=m..<n. cmod(f i))*
apply (*induct n*)
apply (*auto intro*: *complex-mod-triangle-ineq* [*THEN order-trans*])
done

lemma *sumc-fun-eq* [*rule-format* (*no-asm*)]:
(∀ r. m ≤ r & r < n --> f r = g r) --> sumc m n f = sumc m n g
by (*induct n*, *auto*)

lemma *sumc-const* [*simp*]: *sumc 0 n (%i. r) = complex-of-real (real n) * r*
apply (*induct n*)
apply (*auto simp add*: *left-distrib real-of-nat-Suc*)
done

lemma *sumc-add-mult-const*:
*sumc 0 n f + -(complex-of-real(real n) * r) = sumc 0 n (%i. f i + -r)*

by (*simp add*: *sumc-add* [*symmetric*])

lemma *sumc-diff-mult-const*:
  sumc 0 n f − (complex-of-real(real n)∗r) = sumc 0 n (%i. f i − r)
by (*simp add*: *diff-minus sumc-add-mult-const*)

lemma *sumc-less-bounds-zero* [*rule-format*]: *n < m −−> sumc m n f = 0*
by (*induct n*, *auto*)

lemma *sumc-minus*: *sumc m n (%i. − f i) = − sumc m n f*
by (*induct n*, *auto*)

lemma *sumc-shift-bounds*: *sumc (m+k) (n+k) f = sumc m n (%i. f(i + k))*
by (*induct n*, *auto*)

lemma *sumc-minus-one-complexpow-zero* [*simp*]:
  sumc 0 (2∗n) (%i. (−1) ^ Suc i) = 0
by (*induct n*, *auto*)

lemma *sumc-interval-const* [*rule-format* (*no-asm*)]:
  (∀ n. m ≤ Suc n −−> f n = r) & m ≤ na
   −−> sumc m na f = (complex-of-real(real (na − m)) ∗ r)
apply (*induct na*)
apply (*auto simp add*: *Suc-diff-le real-of-nat-Suc left-distrib*)
done

lemma *sumc-interval-const2* [*rule-format* (*no-asm*)]:
  (∀ n. m ≤ n −−> f n = r) & m ≤ na
   −−> sumc m na f = (complex-of-real(real (na − m)) ∗ r)
apply (*induct na*)
apply (*auto simp add*: *left-distrib Suc-diff-le real-of-nat-Suc*)
done

lemma *sumr-cmod-ge-zero* [*iff*]: *0 ≤ (∑ n=m..<n::nat. cmod (f n))*
by (*induct n*, *auto simp add*: *add-increasing*)

lemma *rabs-sumc-cmod-cancel* [*simp*]:
  abs (∑ n=m..<n::nat. cmod (f n)) = (∑ n=m..<n. cmod (f n))
by (*simp add*: *abs-if linorder-not-less*)

lemma *sumc-one-lb-complexpow-zero* [*simp*]: *sumc 1 n (%n. f(n) ∗ 0 ^ n) = 0*
apply (*induct n*)
apply (*case-tac* [*2*] *n*, *auto*)
done

lemma *sumc-diff*: *sumc m n f − sumc m n g = sumc m n (%n. f n − g n)*
by (*simp add*: *diff-minus sumc-add* [*symmetric*] *sumc-minus*)

lemma *sumc-subst* [*rule-format* (*no-asm*)]:
(∀ *p*. (*m* ≤ *p* & *p* < *m* + *n* −−> (*f p* = *g p*))) −−> *sumc m n f* = *sumc m n g*
by (*induct n*, *auto*)

lemma *sumc-group* [*simp*]:
sumc 0 n (%*m*. *sumc* (*m* ∗ *k*) (*m*∗*k* + *k*) *f*) = *sumc 0* (*n* ∗ *k*) *f*
apply (*subgoal-tac k* = *0* | *0* < *k*, *auto*)
apply (*induct n*)
apply (*auto simp add*: *sumc-split-add add-commute*)
done

ML
⟪
val sumc-Suc-zero = *thm sumc-Suc-zero*;
val sumc-eq-bounds = *thm sumc-eq-bounds*;
val sumc-Suc-eq = *thm sumc-Suc-eq*;
val sumc-add-lbound-zero = *thm sumc-add-lbound-zero*;
val sumc-add = *thm sumc-add*;
val sumc-mult = *thm sumc-mult*;
val sumc-split-add = *thm sumc-split-add*;
val sumc-split-add-minus = *thm sumc-split-add-minus*;
val sumc-cmod = *thm sumc-cmod*;
val sumc-fun-eq = *thm sumc-fun-eq*;
val sumc-const = *thm sumc-const*;
val sumc-add-mult-const = *thm sumc-add-mult-const*;
val sumc-diff-mult-const = *thm sumc-diff-mult-const*;
val sumc-less-bounds-zero = *thm sumc-less-bounds-zero*;
val sumc-minus = *thm sumc-minus*;
val sumc-shift-bounds = *thm sumc-shift-bounds*;
val sumc-minus-one-complexpow-zero = *thm sumc-minus-one-complexpow-zero*;
val sumc-interval-const = *thm sumc-interval-const*;
val sumc-interval-const2 = *thm sumc-interval-const2*;
val sumr-cmod-ge-zero = *thm sumr-cmod-ge-zero*;
val rabs-sumc-cmod-cancel = *thm rabs-sumc-cmod-cancel*;
val sumc-one-lb-complexpow-zero = *thm sumc-one-lb-complexpow-zero*;
val sumc-diff = *thm sumc-diff*;
val sumc-subst = *thm sumc-subst*;
val sumc-group = *thm sumc-group*;
⟫

end

40 CLim: Limits, Continuity and Differentiation for Complex Functions

theory *CLim*
imports *CSeries*
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

constdefs

CLIM :: [*complex*=>*complex*,*complex*,*complex*] => *bool*
(((-)/ -- (-)/ --*C*> (-)) [60, 0, 60] 60)
$f\ --\ a\ --C>\ L ==$
$\forall r.\ 0 < r\ -->$
$(\exists s.\ 0 < s\ \&\ (\forall x.\ (x \neq a\ \&\ (cmod(x - a) < s)$
$-->\ cmod(f\ x - L) < r)))$

NSCLIM :: [*complex*=>*complex*,*complex*,*complex*] => *bool*
(((-)/ -- (-)/ --*NSC*> (-)) [60, 0, 60] 60)
$f\ --\ a\ --NSC>\ L == (\forall x.\ (x \neq hcomplex\text{-}of\text{-}complex\ a\ \&$
$x\ @c=\ hcomplex\text{-}of\text{-}complex\ a$
$-->\ (\ *f*\ f)\ x\ @c=\ hcomplex\text{-}of\text{-}complex\ L))$

CRLIM :: [*complex*=>*real*,*complex*,*real*] => *bool*
(((-)/ -- (-)/ --*CR*> (-)) [60, 0, 60] 60)
$f\ --\ a\ --CR>\ L ==$

$$\forall r.\ 0 < r \longrightarrow (\exists s.\ 0 < s\ \&\ (\forall x.\ (x \neq a\ \&\ (cmod(x - a) < s) \longrightarrow abs(f\ x - L) < r)))$$

$$NSCRLIM :: [complex{=}{>}real, complex, real] => bool$$
$$(((-)/\ --\ (-)/\ --NSCR{>}\ (-))\ [60,\ 0,\ 60]\ 60)$$
$$f\ --\ a\ --NSCR{>}\ L == (\forall x.\ (x \neq hcomplex\text{-}of\text{-}complex\ a\ \&\ x\ @c{=}\ hcomplex\text{-}of\text{-}complex\ a \longrightarrow (\ *f*\ f)\ x\ @{=}\ hypreal\text{-}of\text{-}real\ L))$$

$$isContc :: [complex{=}{>}complex, complex] => bool$$
$$isContc\ f\ a == (f\ --\ a\ --C{>}\ (f\ a))$$

$$isNSContc :: [complex{=}{>}complex, complex] => bool$$
$$isNSContc\ f\ a == (\forall y.\ y\ @c{=}\ hcomplex\text{-}of\text{-}complex\ a \longrightarrow (\ *f*\ f)\ y\ @c{=}\ hcomplex\text{-}of\text{-}complex\ (f\ a))$$

$$isContCR :: [complex{=}{>}real, complex] => bool$$
$$isContCR\ f\ a == (f\ --\ a\ --CR{>}\ (f\ a))$$

$$isNSContCR :: [complex{=}{>}real, complex] => bool$$
$$isNSContCR\ f\ a == (\forall y.\ y\ @c{=}\ hcomplex\text{-}of\text{-}complex\ a \longrightarrow (\ *f*\ f)\ y\ @{=}\ hypreal\text{-}of\text{-}real\ (f\ a))$$

$$cderiv:: [complex{=}{>}complex, complex, complex] => bool$$
$$((CDERIV\ (-)/\ (-)/\ :{>}\ (-))\ [60,\ 0,\ 60]\ 60)$$
$$CDERIV\ f\ x :> D == ((\%h.\ (f(x + h) - f(x))/h)\ --\ 0\ --C{>}\ D)$$

$$nscderiv :: [complex{=}{>}complex, complex, complex] => bool$$
$$((NSCDERIV\ (-)/\ (-)/\ :{>}\ (-))\ [60,\ 0,\ 60]\ 60)$$
$$NSCDERIV\ f\ x :> D == (\forall h \in CInfinitesimal - \{0\}.\ ((\ *f*\ f)(hcomplex\text{-}of\text{-}complex\ x + h) - hcomplex\text{-}of\text{-}complex\ (f\ x))/h\ @c{=}\ hcomplex\text{-}of\text{-}complex\ D)$$

$$cdifferentiable :: [complex{=}{>}complex, complex] => bool$$
$$(\textbf{infixl}\ cdifferentiable\ 60)$$
$$f\ cdifferentiable\ x == (\exists D.\ CDERIV\ f\ x :> D)$$

$$NSCdifferentiable :: [complex{=}{>}complex, complex] => bool$$
$$(\textbf{infixl}\ NSCdifferentiable\ 60)$$
$$f\ NSCdifferentiable\ x == (\exists D.\ NSCDERIV\ f\ x :> D)$$

$$isUContc :: (complex{=}{>}complex) => bool$$

$isUContc\ f == (\forall r.\ 0 < r \longrightarrow$
$(\exists s.\ 0 < s\ \&\ (\forall x\ y.\ cmod(x - y) < s$
$\longrightarrow cmod(f\ x - f\ y) < r)))$

$isNSUContc :: (complex \Rightarrow complex) \Rightarrow bool$
$isNSUContc\ f == (\forall x\ y.\ x\ @c=\ y \longrightarrow (\ *f*\ f)\ x\ @c=\ (\ *f*\ f)\ y)$

40.1 Limit of Complex to Complex Function

lemma *NSCLIM-NSCRLIM-Re*: $f\ --\ a\ --NSC>\ L \Longrightarrow (\%x.\ Re(f\ x))\ --\ a\ --NSCR>\ Re(L)$
by (*simp add*: *NSCLIM-def NSCRLIM-def starfunC-approx-Re-Im-iff hRe-hcomplex-of-complex*)

lemma *NSCLIM-NSCRLIM-Im*: $f\ --\ a\ --NSC>\ L \Longrightarrow (\%x.\ Im(f\ x))\ --\ a\ --NSCR>\ Im(L)$
by (*simp add*: *NSCLIM-def NSCRLIM-def starfunC-approx-Re-Im-iff hIm-hcomplex-of-complex*)

lemma *CLIM-NSCLIM*:
$f\ --\ x\ --C>\ L \Longrightarrow f\ --\ x\ --NSC>\ L$
apply (*simp add*: *CLIM-def NSCLIM-def capprox-def*, *auto*)
apply (*rule-tac* $x = xa$ **in** *star-cases*)
apply (*auto simp add*: *starfun star-n-diff star-of-def star-n-eq-iff CInfinitesimal-hcmod-iff hcmod Infinitesimal-FreeUltrafilterNat-iff*)
apply (*rule bexI* [*OF - Rep-star-star-n*], *safe*)
apply (*drule-tac* $x = u$ **in** *spec*, *auto*)
apply (*drule-tac* $x = s$ **in** *spec*, *auto*, *ultra*)
apply (*drule sym*, *auto*)
done

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

lemma *lemma-CLIM*:
$\forall s.\ 0 < s \longrightarrow (\exists xa.\ \ xa \neq x\ \&$
$cmod\ (xa - x) < s\ \ \&\ r \leq cmod\ (f\ xa - L))$
$\Longrightarrow \forall (n::nat).\ \exists xa.\ \ xa \neq x\ \&$
$cmod(xa - x) < inverse(real(Suc\ n))\ \&\ r \leq cmod(f\ xa - L)$
apply *clarify*
apply (*cut-tac* $n1 = n$ **in** *real-of-nat-Suc-gt-zero* [*THEN positive-imp-inverse-positive*], *auto*)
done

lemma *lemma-skolemize-CLIM2*:

$\forall s.\ 0 < s \longrightarrow (\exists xa.\ \ xa \neq x\ \&$
$\quad cmod\ (xa - x) < s\ \ \&\ r \leq cmod\ (f\ xa - L))$
$\Longrightarrow \exists X.\ \forall (n::nat).\ X\ n \neq x\ \&$
$\quad cmod(X\ n - x) < inverse(real(Suc\ n))\ \&\ r \leq cmod(f\ (X\ n) - L)$
apply (*drule lemma-CLIM*)
apply (*drule choice*, *blast*)
done

lemma *lemma-csimp*:
$\forall n.\ X\ n \neq x\ \&$
$cmod\ (X\ n - x) < inverse\ (real(Suc\ n))\ \&$
$r \leq cmod\ (f\ (X\ n) - L) \Longrightarrow$
$\forall n.\ cmod\ (X\ n - x) < inverse\ (real(Suc\ n))$
by *auto*

lemma *NSCLIM-CLIM*:
$f\ -\!-\ x\ -\!-NSC\!>\ L \Longrightarrow f\ -\!-\ x\ -\!-C\!>\ L$
apply (*simp add*: *CLIM-def NSCLIM-def*)
apply (*rule ccontr*)
apply (*auto simp add*: *eq-Abs-star-ALL starfun*
CInfinitesimal-capprox-minus [*symmetric*] *star-n-diff*
CInfinitesimal-hcmod-iff star-of-def star-n-eq-iff
Infinitesimal-FreeUltrafilterNat-iff hcmod)
apply (*simp add*: *linorder-not-less*)
apply (*drule lemma-skolemize-CLIM2*, *safe*)
apply (*drule-tac x* = *X* **in** *spec*, *auto*)
apply (*drule lemma-csimp* [*THEN complex-seq-to-hcomplex-CInfinitesimal*])
apply (*simp add*: *CInfinitesimal-hcmod-iff star-of-def*
Infinitesimal-FreeUltrafilterNat-iff star-n-diff hcmod, *blast*)
apply (*drule-tac x* = *r* **in** *spec*, *clarify*)
apply (*drule FreeUltrafilterNat-all*, *ultra*, *arith*)
done

First key result

theorem *CLIM-NSCLIM-iff*: $(f\ -\!-\ x\ -\!-C\!>\ L) = (f\ -\!-\ x\ -\!-NSC\!>\ L)$
by (*blast intro*: *CLIM-NSCLIM NSCLIM-CLIM*)

40.2 Limit of Complex to Real Function

lemma *CRLIM-NSCRLIM*: $f\ -\!-\ x\ -\!-CR\!>\ L \Longrightarrow f\ -\!-\ x\ -\!-NSCR\!>\ L$
apply (*simp add*: *CRLIM-def NSCRLIM-def capprox-def*, *auto*)
apply (*rule-tac x* = *xa* **in** *star-cases*)
apply (*auto simp add*: *starfun star-n-diff*
CInfinitesimal-hcmod-iff hcmod
Infinitesimal-FreeUltrafilterNat-iff
star-of-def star-n-eq-iff
Infinitesimal-approx-minus [*symmetric*])
apply (*rule bexI* [*OF - Rep-star-star-n*], *safe*)
apply (*drule-tac x* = *u* **in** *spec*, *auto*)

apply (*drule-tac* $x = s$ **in** *spec*, *auto*, *ultra*)
apply (*drule sym*, *auto*)
done

lemma *lemma-CRLIM*:
$\forall s.\ 0 < s \longrightarrow (\exists xa.\ xa \neq x\ \&$
$cmod\ (xa - x) < s\ \&\ r \leq abs\ (f\ xa - L))$
$\Longrightarrow \forall (n::nat).\ \exists xa.\ xa \neq x\ \&$
$cmod(xa - x) < inverse(real(Suc\ n))\ \&\ r \leq abs\ (f\ xa - L)$
apply *clarify*
apply (*cut-tac n1* $= n$ **in** *real-of-nat-Suc-gt-zero* [*THEN positive-imp-inverse-positive*], *auto*)
done

lemma *lemma-skolemize-CRLIM2*:
$\forall s.\ 0 < s \longrightarrow (\exists xa.\ xa \neq x\ \&$
$cmod\ (xa - x) < s\ \&\ r \leq abs\ (f\ xa - L))$
$\Longrightarrow \exists X.\ \forall (n::nat).\ X\ n \neq x\ \&$
$cmod(X\ n - x) < inverse(real(Suc\ n))\ \&\ r \leq abs\ (f\ (X\ n) - L)$
apply (*drule lemma-CRLIM*)
apply (*drule choice*, *blast*)
done

lemma *lemma-crsimp*:
$\forall n.\ X\ n \neq x\ \&$
$cmod\ (X\ n - x) < inverse\ (real(Suc\ n))\ \&$
$r \leq abs\ (f\ (X\ n) - L) \Longrightarrow$
$\forall n.\ cmod\ (X\ n - x) < inverse\ (real(Suc\ n))$
by *auto*

lemma *NSCRLIM-CRLIM*: $f -- x --NSCR> L \Longrightarrow f -- x --CR> L$
apply (*simp add*: *CRLIM-def NSCRLIM-def capprox-def*)
apply (*rule ccontr*)
apply (*auto simp add*: *eq-Abs-star-ALL starfun star-n-diff*
CInfinitesimal-hcmod-iff
hcmod Infinitesimal-approx-minus [*symmetric*]
star-of-def star-n-eq-iff
Infinitesimal-FreeUltrafilterNat-iff)
apply (*simp add*: *linorder-not-less*)
apply (*drule lemma-skolemize-CRLIM2*, *safe*)
apply (*drule-tac* $x = X$ **in** *spec*, *auto*)
apply (*drule lemma-crsimp* [*THEN complex-seq-to-hcomplex-CInfinitesimal*])
apply (*simp add*: *CInfinitesimal-hcmod-iff star-of-def*
Infinitesimal-FreeUltrafilterNat-iff star-n-diff hcmod)
apply (*auto simp add*: *star-of-def star-n-diff*)
apply (*drule-tac* $x = r$ **in** *spec*, *clarify*)
apply (*drule FreeUltrafilterNat-all*, *ultra*)
done

Second key result

theorem *CRLIM-NSCRLIM-iff*: $(f \mathrel{--} x \mathrel{--CR>} L) = (f \mathrel{--} x \mathrel{--NSCR>} L)$
by (*blast intro*: *CRLIM-NSCRLIM NSCRLIM-CRLIM*)

lemma *CLIM-CRLIM-Re*: $f \mathrel{--} a \mathrel{--C>} L \Longrightarrow (\%x.\ Re(f\,x)) \mathrel{--} a \mathrel{--CR>} Re(L)$
by (*auto dest*: *NSCLIM-NSCRLIM-Re simp add*: *CLIM-NSCLIM-iff CRLIM-NSCRLIM-iff* [*symmetric*])

lemma *CLIM-CRLIM-Im*: $f \mathrel{--} a \mathrel{--C>} L \Longrightarrow (\%x.\ Im(f\,x)) \mathrel{--} a \mathrel{--CR>} Im(L)$
by (*auto dest*: *NSCLIM-NSCRLIM-Im simp add*: *CLIM-NSCLIM-iff CRLIM-NSCRLIM-iff* [*symmetric*])

lemma *CLIM-cnj*: $f \mathrel{--} a \mathrel{--C>} L \Longrightarrow (\%x.\ cnj\ (f\,x)) \mathrel{--} a \mathrel{--C>} cnj\ L$
by (*simp add*: *CLIM-def complex-cnj-diff* [*symmetric*])

lemma *CLIM-cnj-iff*: $((\%x.\ cnj\ (f\,x)) \mathrel{--} a \mathrel{--C>} cnj\ L) = (f \mathrel{--} a \mathrel{--C>} L)$
by (*simp add*: *CLIM-def complex-cnj-diff* [*symmetric*])

lemma *NSCLIM-add*:
$[\!|\ f \mathrel{--} x \mathrel{--NSC>} l;\ g \mathrel{--} x \mathrel{--NSC>} m\ |\!]$
$\Longrightarrow (\%x.\ f(x) + g(x)) \mathrel{--} x \mathrel{--NSC>} (l + m)$
by (*auto simp*: *NSCLIM-def intro*!: *capprox-add*)

lemma *CLIM-add*:
$[\!|\ f \mathrel{--} x \mathrel{--C>} l;\ g \mathrel{--} x \mathrel{--C>} m\ |\!]$
$\Longrightarrow (\%x.\ f(x) + g(x)) \mathrel{--} x \mathrel{--C>} (l + m)$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-add*)

lemma *NSCLIM-mult*:
$[\!|\ f \mathrel{--} x \mathrel{--NSC>} l;\ g \mathrel{--} x \mathrel{--NSC>} m\ |\!]$
$\Longrightarrow (\%x.\ f(x) * g(x)) \mathrel{--} x \mathrel{--NSC>} (l * m)$
by (*auto simp add*: *NSCLIM-def intro*!: *capprox-mult-CFinite*)

lemma *CLIM-mult*:
$[\!|\ f \mathrel{--} x \mathrel{--C>} l;\ g \mathrel{--} x \mathrel{--C>} m\ |\!]$
$\Longrightarrow (\%x.\ f(x) * g(x)) \mathrel{--} x \mathrel{--C>} (l * m)$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-mult*)

lemma *NSCLIM-const* [*simp*]: $(\%x.\ k) \mathrel{--} x \mathrel{--NSC>} k$
by (*simp add*: *NSCLIM-def*)

lemma *CLIM-const* [*simp*]: $(\%x.\ k)$ $--$ x $--C>$ k
by (*simp add*: *CLIM-def*)

lemma *NSCLIM-minus*: f $--$ a $--NSC>$ L $==>$ $(\%x.\ -f(x))$ $--$ a $--NSC>$ $-L$
by (*simp add*: *NSCLIM-def*)

lemma *CLIM-minus*: f $--$ a $--C>$ L $==>$ $(\%x.\ -f(x))$ $--$ a $--C>$ $-L$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-minus*)

lemma *NSCLIM-diff*:
$[|\ f$ $--$ x $--NSC>$ $l;$ g $--$ x $--NSC>$ $m\ |]$
$==>$ $(\%x.\ f(x) - g(x))$ $--$ x $--NSC>$ $(l - m)$
by (*simp add*: *diff-minus NSCLIM-add NSCLIM-minus*)

lemma *CLIM-diff*:
$[|\ f$ $--$ x $--C>$ $l;$ g $--$ x $--C>$ $m\ |]$
$==>$ $(\%x.\ f(x) - g(x))$ $--$ x $--C>$ $(l - m)$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-diff*)

lemma *NSCLIM-inverse*:
$[|\ f$ $--$ a $--NSC>$ $L;$ $L \neq 0\ |]$
$==>$ $(\%x.\ inverse(f(x)))$ $--$ a $--NSC>$ $(inverse\ L)$
apply (*simp add*: *NSCLIM-def*, *clarify*)
apply (*drule spec*)
apply (*force simp add*: *hcomplex-of-complex-capprox-inverse*)
done

lemma *CLIM-inverse*:
$[|\ f$ $--$ a $--C>$ $L;$ $L \neq 0\ |]$
$==>$ $(\%x.\ inverse(f(x)))$ $--$ a $--C>$ $(inverse\ L)$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-inverse*)

lemma *NSCLIM-zero*: f $--$ a $--NSC>$ l $==>$ $(\%x.\ f(x) - l)$ $--$ a $--NSC>$ 0
apply (*simp add*: *diff-minus*)
apply (*rule-tac a1* $= l$ **in** *right-minus* [*THEN subst*])
apply (*rule NSCLIM-add*, *auto*)
done

lemma *CLIM-zero*: f $--$ a $--C>$ l $==>$ $(\%x.\ f(x) - l)$ $--$ a $--C>$ 0
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-zero*)

lemma *NSCLIM-zero-cancel*: $(\%x.\ f(x) - l)$ $--$ x $--NSC>$ 0 $==>$ f $--$ x $--NSC>$ l
by (*drule-tac* $g = \%x.\ l$ **and** $m = l$ **in** *NSCLIM-add*, *auto*)

lemma *CLIM-zero-cancel*: $(\%x.\ f(x) - l)$ $--$ x $--C>$ 0 $==>$ f $--$ x $--C>$ l
by (*drule-tac* $g = \%x.\ l$ **and** $m = l$ **in** *CLIM-add*, *auto*)

lemma *NSCLIM-not-zero*: $k \neq 0$ $==>$ $\sim$ $((\%x.\ k)$ $--$ x $--NSC>$ $0)$
apply (*auto simp del*: *star-of-zero simp add*: *NSCLIM-def*)
apply (*rule-tac* x = *hcomplex-of-complex* x + *hcomplex-of-hypreal epsilon* **in** *exI*)
apply (*auto intro*: *CInfinitesimal-add-capprox-self* [*THEN capprox-sym*]
simp *del*: *star-of-zero*)
done

lemmas *NSCLIM-not-zeroE* = *NSCLIM-not-zero* [*THEN notE*, *standard*]

lemma *CLIM-not-zero*: $k \neq 0$ $==>$ $\sim$ $((\%x.\ k)$ $--$ x $--C>$ $0)$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-not-zero*)

lemma *NSCLIM-const-eq*: $(\%x.\ k)$ $--$ x $--NSC>$ L $==>$ $k = L$
apply (*rule ccontr*)
apply (*drule NSCLIM-zero*)
apply (*rule NSCLIM-not-zeroE* [*of* $k-L$], *auto*)
done

lemma *CLIM-const-eq*: $(\%x.\ k)$ $--$ x $--C>$ L $==>$ $k = L$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-const-eq*)

lemma *NSCLIM-unique*: $[|\ f$ $--$ x $--NSC>$ L; f $--$ x $--NSC>$ $M\ |]$ $==>$ $L = M$
apply (*drule NSCLIM-minus*)
apply (*drule NSCLIM-add*, *assumption*)
apply (*auto dest*!: *NSCLIM-const-eq* [*symmetric*])
done

lemma *CLIM-unique*: $[|\ f$ $--$ x $--C>$ L; f $--$ x $--C>$ $M\ |]$ $==>$ $L = M$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-unique*)

lemma *NSCLIM-mult-zero*:
[| f −− x −−*NSC*> *0*; g −− x −−*NSC*> *0* |] ==> (%x. $f(x)$∗$g(x)$) −− x −−*NSC*> *0*
by (*drule NSCLIM-mult*, *auto*)

lemma *CLIM-mult-zero*:
[| f −− x −−*C*> *0*; g −− x −−*C*> *0* |] ==> (%x. $f(x)$∗$g(x)$) −− x −−*C*> *0*
by (*drule CLIM-mult*, *auto*)

lemma *NSCLIM-self*: (%x. x) −− a −−*NSC*> a
by (*auto simp add*: *NSCLIM-def intro*: *starfunC-Idfun-capprox*)

lemma *CLIM-self*: (%x. x) −− a −−*C*> a
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-self*)

lemma *NSCLIM-NSCRLIM-iff*:
(f −− x −−*NSC*> L) = ((%y. *cmod*(f y − L)) −− x −−*NSCR*> *0*)
apply (*auto simp add*: *NSCLIM-def NSCRLIM-def CInfinitesimal-capprox-minus* [*symmetric*] *CInfinitesimal-hcmod-iff*)
apply (*auto dest*!: *spec*)
apply (*rule-tac* [!] x = *xa* **in** *star-cases*)
apply (*auto simp add*: *star-n-diff starfun hcmod mem-infmal-iff star-of-def*)
done

lemma *CLIM-CRLIM-iff*: (f −− x −−*C*> L) = ((%y. *cmod*(f y − L)) −− x −−*CR*> *0*)
by (*simp add*: *CLIM-def CRLIM-def*)

lemma *NSCLIM-NSCRLIM-iff2*:
(f −− x −−*NSC*> L) = ((%y. *cmod*(f y − L)) −− x −−*NSCR*> *0*)
by (*simp add*: *CRLIM-NSCRLIM-iff* [*symmetric*] *CLIM-CRLIM-iff CLIM-NSCLIM-iff* [*symmetric*])

lemma *NSCLIM-NSCRLIM-Re-Im-iff*:
(f −− a −−*NSC*> L) = ((%x. *Re*(f x)) −− a −−*NSCR*> *Re*(L) &
(%x. *Im*(f x)) −− a −−*NSCR*> *Im*(L))
apply (*auto intro*: *NSCLIM-NSCRLIM-Re NSCLIM-NSCRLIM-Im*)
apply (*auto simp add*: *NSCLIM-def NSCRLIM-def*)
apply (*auto dest*!: *spec*)
apply (*rule-tac* x = x **in** *star-cases*)
apply (*simp add*: *capprox-approx-iff starfun star-of-def*)

done

lemma *CLIM-CRLIM-Re-Im-iff*:
(*f* −− *a* −−*C*> *L*) = ((%*x*. *Re*(*f x*)) −− *a* −−*CR*> *Re*(*L*) &
(%*x*. *Im*(*f x*)) −− *a* −−*CR*> *Im*(*L*))
by (*simp add*: *CLIM-NSCLIM-iff CRLIM-NSCRLIM-iff NSCLIM-NSCRLIM-Re-Im-iff*)

40.3 Continuity

lemma *isNSContcD*:
[| *isNSContc f a*; *y* @*c*= *hcomplex-of-complex a* |]
==> (∗*f*∗ *f*) *y* @*c*= *hcomplex-of-complex* (*f a*)
by (*simp add*: *isNSContc-def*)

lemma *isNSContc-NSCLIM*: *isNSContc f a* ==> *f* −− *a* −−*NSC*> (*f a*)
by (*simp add*: *isNSContc-def NSCLIM-def*)

lemma *NSCLIM-isNSContc*:
f −− *a* −−*NSC*> (*f a*) ==> *isNSContc f a*
apply (*simp add*: *isNSContc-def NSCLIM-def*, *auto*)
apply (*case-tac y* = *hcomplex-of-complex a*, *auto*)
done

Nonstandard continuity can be defined using NS Limit in similar fashion to standard definition of continuity

lemma *isNSContc-NSCLIM-iff*: (*isNSContc f a*) = (*f* −− *a* −−*NSC*> (*f a*))
by (*blast intro*: *isNSContc-NSCLIM NSCLIM-isNSContc*)

lemma *isNSContc-CLIM-iff*: (*isNSContc f a*) = (*f* −− *a* −−*C*> (*f a*))
by (*simp add*: *CLIM-NSCLIM-iff isNSContc-NSCLIM-iff*)

lemma *isNSContc-isContc-iff*: (*isNSContc f a*) = (*isContc f a*)
by (*simp add*: *isContc-def isNSContc-CLIM-iff*)

lemma *isContc-isNSContc*: *isContc f a* ==> *isNSContc f a*
by (*erule isNSContc-isContc-iff* [*THEN iffD2*])

lemma *isNSContc-isContc*: *isNSContc f a* ==> *isContc f a*
by (*erule isNSContc-isContc-iff* [*THEN iffD1*])

Alternative definition of continuity

lemma *NSCLIM-h-iff*: (*f* −− *a* −−*NSC*> *L*) = ((%*h*. *f*(*a* + *h*)) −− *0* −−*NSC*> *L*)
apply (*simp add*: *NSCLIM-def*, *auto*)
apply (*drule-tac x* = *hcomplex-of-complex a* + *x* **in** *spec*)
apply (*drule-tac* [*2*] *x* = − *hcomplex-of-complex a* + *x* **in** *spec*, *safe*, *simp*)
apply (*rule mem-cinfmal-iff* [*THEN iffD2*, *THEN CInfinitesimal-add-capprox-self* [*THEN capprox-sym*]])

apply (*rule-tac* [*4*] *capprox-minus-iff2* [*THEN iffD1*])
prefer *3* **apply** (*simp add*: *compare-rls add-commute*)
apply (*rule-tac* [*2*] $x = x$ **in** *star-cases*)
apply (*rule-tac* [*4*] $x = x$ **in** *star-cases*)
apply (*auto simp add*: *starfun star-n-minus star-n-add star-of-def*)
done

lemma *NSCLIM-isContc-iff*:
$(f \mathrel{--} a \mathrel{--NSC>} f\ a) = ((\%h.\ f(a + h)) \mathrel{--} 0 \mathrel{--NSC>} f\ a)$
by (*rule NSCLIM-h-iff*)

lemma *CLIM-isContc-iff*: $(f \mathrel{--} a \mathrel{--C>} f\ a) = ((\%h.\ f(a + h)) \mathrel{--} 0 \mathrel{--C>} f(a))$
by (*simp add*: *CLIM-NSCLIM-iff NSCLIM-isContc-iff*)

lemma *isContc-iff*: $(isContc\ f\ x) = ((\%h.\ f(x + h)) \mathrel{--} 0 \mathrel{--C>} f(x))$
by (*simp add*: *isContc-def CLIM-isContc-iff*)

lemma *isContc-add*:
$[|\ isContc\ f\ a;\ isContc\ g\ a\ |] ==> isContc\ (\%x.\ f(x) + g(x))\ a$
by (*auto intro*: *capprox-add simp add*: *isNSContc-isContc-iff* [*symmetric*] *isNSContc-def*)

lemma *isContc-mult*:
$[|\ isContc\ f\ a;\ isContc\ g\ a\ |] ==> isContc\ (\%x.\ f(x) * g(x))\ a$
by (*auto intro*!: *starfun-mult-CFinite-capprox*
[*simplified starfun-mult* [*symmetric*]]
simp add: *isNSContc-isContc-iff* [*symmetric*] *isNSContc-def*)

lemma *isContc-o*: $[|\ isContc\ f\ a;\ isContc\ g\ (f\ a)\ |] ==> isContc\ (g\ o\ f)\ a$
by (*simp add*: *isNSContc-isContc-iff* [*symmetric*] *isNSContc-def starfun-o* [*symmetric*])

lemma *isContc-o2*:
$[|\ isContc\ f\ a;\ isContc\ g\ (f\ a)\ |] ==> isContc\ (\%x.\ g\ (f\ x))\ a$
by (*auto dest*: *isContc-o simp add*: *o-def*)

lemma *isNSContc-minus*: $isNSContc\ f\ a ==> isNSContc\ (\%x.\ -\ f\ x)\ a$
by (*simp add*: *isNSContc-def*)

lemma *isContc-minus*: $isContc\ f\ a ==> isContc\ (\%x.\ -\ f\ x)\ a$
by (*simp add*: *isNSContc-isContc-iff* [*symmetric*] *isNSContc-minus*)

lemma *isContc-inverse*:
$[|\ isContc\ f\ x;\ f\ x \neq 0\ |] ==> isContc\ (\%x.\ inverse\ (f\ x))\ x$
by (*simp add*: *isContc-def CLIM-inverse*)

lemma *isNSContc-inverse*:
$[|\ isNSContc\ f\ x;\ f\ x \neq 0\ |] ==> isNSContc\ (\%x.\ inverse\ (f\ x))\ x$
by (*auto intro*: *isContc-inverse simp add*: *isNSContc-isContc-iff*)

lemma *isContc-diff*:
[| *isContc f a*; *isContc g a* |] ==> *isContc* (%*x*. *f*(*x*) − *g*(*x*)) *a*
apply (*simp add*: *diff-minus*)
apply (*auto intro*: *isContc-add isContc-minus*)
done

lemma *isContc-const* [*simp*]: *isContc* (%*x*. *k*) *a*
by (*simp add*: *isContc-def*)

lemma *isNSContc-const* [*simp*]: *isNSContc* (%*x*. *k*) *a*
by (*simp add*: *isNSContc-def*)

40.4 Functions from Complex to Reals

lemma *isNSContCRD*:
[| *isNSContCR f a*; *y* @*c*= *hcomplex-of-complex a* |]
==> (∗*f*∗ *f*) *y* @= *hypreal-of-real* (*f a*)
by (*simp add*: *isNSContCR-def*)

lemma *isNSContCR-NSCRLIM*: *isNSContCR f a* ==> *f* −− *a* −−*NSCR*> (*f a*)
by (*simp add*: *isNSContCR-def NSCRLIM-def*)

lemma *NSCRLIM-isNSContCR*: *f* −− *a* −−*NSCR*> (*f a*) ==> *isNSContCR f a*
apply (*auto simp add*: *isNSContCR-def NSCRLIM-def*)
apply (*case-tac y* = *hcomplex-of-complex a*, *auto*)
done

lemma *isNSContCR-NSCRLIM-iff*: (*isNSContCR f a*) = (*f* −− *a* −−*NSCR*> (*f a*))
by (*blast intro*: *isNSContCR-NSCRLIM NSCRLIM-isNSContCR*)

lemma *isNSContCR-CRLIM-iff*: (*isNSContCR f a*) = (*f* −− *a* −−*CR*> (*f a*))
by (*simp add*: *CRLIM-NSCRLIM-iff isNSContCR-NSCRLIM-iff*)

lemma *isNSContCR-isContCR-iff*: (*isNSContCR f a*) = (*isContCR f a*)
by (*simp add*: *isContCR-def isNSContCR-CRLIM-iff*)

lemma *isContCR-isNSContCR*: *isContCR f a* ==> *isNSContCR f a*
by (*erule isNSContCR-isContCR-iff* [*THEN iffD2*])

lemma *isNSContCR-isContCR*: *isNSContCR f a* ==> *isContCR f a*
by (*erule isNSContCR-isContCR-iff* [*THEN iffD1*])

lemma *isNSContCR-cmod* [*simp*]: *isNSContCR cmod* (*a*)
by (*auto intro*: *capprox-hcmod-approx*
simp add: *starfunCR-cmod hcmod-hcomplex-of-complex* [*symmetric*]

isNSContCR-def)

lemma *isContCR-cmod* [*simp*]: *isContCR cmod* (*a*)
by (*simp add*: *isNSContCR-isContCR-iff* [*symmetric*])

lemma *isContc-isContCR-Re*: *isContc f a ==> isContCR* (%*x*. *Re* (*f x*)) *a*
by (*simp add*: *isContc-def isContCR-def CLIM-CRLIM-Re*)

lemma *isContc-isContCR-Im*: *isContc f a ==> isContCR* (%*x*. *Im* (*f x*)) *a*
by (*simp add*: *isContc-def isContCR-def CLIM-CRLIM-Im*)

40.5 Derivatives

lemma *CDERIV-iff*: (*CDERIV f x :> D*) = ((%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−*C*> *D*)
by (*simp add*: *cderiv-def*)

lemma *CDERIV-NSC-iff*:
(*CDERIV f x :> D*) = ((%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−*NSC*> *D*)
by (*simp add*: *cderiv-def CLIM-NSCLIM-iff*)

lemma *CDERIVD*: *CDERIV f x :> D ==>* (%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−*C*> *D*
by (*simp add*: *cderiv-def*)

lemma *NSC-DERIVD*: *CDERIV f x :> D ==>* (%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−*NSC*> *D*
by (*simp add*: *cderiv-def CLIM-NSCLIM-iff*)

Uniqueness

lemma *CDERIV-unique*: [| *CDERIV f x :> D*; *CDERIV f x :> E* |] ==> *D* = *E*
by (*simp add*: *cderiv-def CLIM-unique*)

lemma *NSCDeriv-unique*: [| *NSCDERIV f x :> D*; *NSCDERIV f x :> E* |] ==> *D* = *E*
apply (*simp add*: *nscderiv-def*)
apply (*auto dest*!: *bspec* [**where** *x* = *hcomplex-of-hypreal epsilon*]
intro!: *inj-hcomplex-of-complex* [*THEN injD*] *dest*: *capprox-trans3*)
done

40.6 Differentiability

lemma *CDERIV-CLIM-iff*:
((%*h*. (*f*(*a* + *h*) − *f*(*a*))/*h*) −− *0* −−*C*> *D*) =
((%*x*. (*f*(*x*) − *f*(*a*)) / (*x* − *a*)) −− *a* −−*C*> *D*)
apply (*simp add*: *CLIM-def*)
apply (*rule-tac f*=*All* **in** *arg-cong*)
apply (*rule ext*)

apply (*rule imp-cong*)
apply (*rule refl*)
apply (*rule-tac f=Ex* **in** *arg-cong*)
apply (*rule ext*)
apply (*rule conj-cong*)
apply (*rule refl*)
apply (*rule trans*)
apply (*rule all-shift* [**where** *a=a*], *simp*)
done

lemma *CDERIV-iff2*:
(*CDERIV f x :> D*) = ((%*z*. (*f*(*z*) − *f*(*x*)) / (*z* − *x*)) −− *x* −−*C*> *D*)
by (*simp add*: *cderiv-def CDERIV-CLIM-iff*)

40.7 Equivalence of NS and Standard Differentiation

lemma *NSCDERIV-NSCLIM-iff*:
(*NSCDERIV f x :> D*) = ((%*h*. (*f*(*x* + *h*) − *f*(*x*))/*h*) −− *0* −−*NSC*> *D*)
apply (*simp add*: *nscderiv-def NSCLIM-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-cinfmal-iff starfun-lambda-cancel*)
done

lemma *NSCDERIV-NSCLIM-iff2*:
(*NSCDERIV f x :> D*) = ((%*z*. (*f*(*z*) − *f*(*x*)) / (*z* − *x*)) −− *x* −−*NSC*> *D*)
by (*simp add*: *NSCDERIV-NSCLIM-iff CDERIV-CLIM-iff CLIM-NSCLIM-iff* [*symmetric*])

lemma *NSCDERIV-iff2*:
(*NSCDERIV f x :> D*) =
(∀ *xa*. *xa* ≠ *hcomplex-of-complex x* & *xa* @*c*= *hcomplex-of-complex x* −−>
(∗*f*∗ (%*z*. (*f z* − *f x*) / (*z* − *x*))) *xa* @*c*= *hcomplex-of-complex D*)
by (*simp add*: *NSCDERIV-NSCLIM-iff2 NSCLIM-def*)

lemma *NSCDERIV-CDERIV-iff*: (*NSCDERIV f x :> D*) = (*CDERIV f x :> D*)
by (*simp add*: *cderiv-def NSCDERIV-NSCLIM-iff CLIM-NSCLIM-iff*)

lemma *NSCDERIV-isNSContc*: *NSCDERIV f x :> D ==> isNSContc f x*
apply (*auto simp add*: *nscderiv-def isNSContc-NSCLIM-iff NSCLIM-def diff-minus*)
apply (*drule capprox-minus-iff* [*THEN iffD1*])
apply (*subgoal-tac xa + − (hcomplex-of-complex x) ≠ 0*)
prefer *2* **apply** (*simp add*: *compare-rls*)
apply (*drule-tac x = − hcomplex-of-complex x + xa* **in** *bspec*)
prefer *2* **apply** (*simp add*: *add-assoc* [*symmetric*])
apply (*auto simp add*: *mem-cinfmal-iff* [*symmetric*] *add-commute*)
apply (*drule-tac c = xa + − hcomplex-of-complex x* **in** *capprox-mult1*)
apply (*auto intro*: *CInfinitesimal-subset-CFinite* [*THEN subsetD*]

simp add: *mult-assoc*)
apply (*drule-tac x3 = D* **in**
CFinite-hcomplex-of-complex [*THEN* [*2*] *CInfinitesimal-CFinite-mult*,
THEN mem-cinfmal-iff [*THEN iffD1*]])
apply (*blast intro*: *capprox-trans mult-commute* [*THEN subst*] *capprox-minus-iff* [*THEN iffD2*])
done

lemma *CDERIV-isContc*: *CDERIV f x :> D ==> isContc f x*
by (*simp add*: *NSCDERIV-CDERIV-iff* [*symmetric*] *isNSContc-isContc-iff* [*symmetric*] *NSCDERIV-isNSContc*)

Differentiation rules for combinations of functions follow by clear, straightforard algebraic manipulations

lemma *NSCDERIV-const* [*simp*]: (*NSCDERIV* (%*x*. *k*) *x* :> *0*)
by (*simp add*: *NSCDERIV-NSCLIM-iff*)

lemma *CDERIV-const* [*simp*]: (*CDERIV* (%*x*. *k*) *x* :> *0*)
by (*simp add*: *NSCDERIV-CDERIV-iff* [*symmetric*])

lemma *NSCDERIV-add*:
[| *NSCDERIV f x* :> *Da*; *NSCDERIV g x* :> *Db* |]
==> *NSCDERIV* (%*x*. *f x* + *g x*) *x* :> *Da* + *Db*
apply (*simp add*: *NSCDERIV-NSCLIM-iff NSCLIM-def*, *clarify*)
apply (*auto dest*!: *spec simp add*: *add-divide-distrib diff-minus*)
apply (*drule-tac b = hcomplex-of-complex Da* **and** *d = hcomplex-of-complex Db*
in *capprox-add*)
apply (*auto simp add*: *add-ac*)
done

lemma *CDERIV-add*:
[| *CDERIV f x* :> *Da*; *CDERIV g x* :> *Db* |]
==> *CDERIV* (%*x*. *f x* + *g x*) *x* :> *Da* + *Db*
by (*simp add*: *NSCDERIV-add NSCDERIV-CDERIV-iff* [*symmetric*])

40.8 Lemmas for Multiplication

lemma *lemma-nscderiv1*: ((*a*::*hcomplex*)∗*b*) − (*c*∗*d*) = (*b*∗(*a* − *c*)) + (*c*∗(*b* − *d*))
by (*simp add*: *right-diff-distrib*)

lemma *lemma-nscderiv2*:
[| (*x* + *y*) / *z* = *hcomplex-of-complex D* + *yb*; *z* ≠ *0*;
z : *CInfinitesimal*; *yb* : *CInfinitesimal* |]
==> *x* + *y* @*c*= *0*
apply (*frule-tac c1 = z* **in** *hcomplex-mult-right-cancel* [*THEN iffD2*], *assumption*)
apply (*erule-tac V* = (*x* + *y*) / *z* = *hcomplex-of-complex D* + *yb* **in** *thin-rl*)
apply (*auto intro*!: *CInfinitesimal-CFinite-mult2 CFinite-add*
simp add: *mem-cinfmal-iff* [*symmetric*])

apply (*erule CInfinitesimal-subset-CFinite* [*THEN subsetD*])
done

lemma *NSCDERIV-mult*:
[| *NSCDERIV f x :> Da*; *NSCDERIV g x :> Db* |]
==> *NSCDERIV* (%*x*. *f x* * *g x*) *x :>* (*Da* * *g*(*x*)) + (*Db* * *f*(*x*))
apply (*simp add*: *NSCDERIV-NSCLIM-iff NSCLIM-def*, *clarify*)
apply (*auto dest*!: *spec*
simp add: *starfun-lambda-cancel lemma-nscderiv1*)
apply (*simp* (*no-asm*) *add*: *add-divide-distrib*)
apply (*drule bex-CInfinitesimal-iff2* [*THEN iffD2*])+
apply (*auto simp del*: *times-divide-eq-right simp add*: *times-divide-eq-right* [*symmetric*])
apply (*simp add*: *diff-minus*)
apply (*drule-tac D = Db* **in** *lemma-nscderiv2*)
apply (*drule-tac* [*4*]
capprox-minus-iff [*THEN iffD2*, *THEN bex-CInfinitesimal-iff2* [*THEN iffD2*]])
apply (*auto intro*!: *capprox-add-mono1 simp add*: *left-distrib right-distrib mult-commute add-assoc*)
apply (*rule-tac b1 = hcomplex-of-complex Db * hcomplex-of-complex* (*f x*) **in** *add-commute* [*THEN subst*])
apply (*auto intro*!: *CInfinitesimal-add-capprox-self2* [*THEN capprox-sym*]
CInfinitesimal-add CInfinitesimal-mult
CInfinitesimal-hcomplex-of-complex-mult
CInfinitesimal-hcomplex-of-complex-mult2
simp add: *add-assoc* [*symmetric*])
done

lemma *CDERIV-mult*:
[| *CDERIV f x :> Da*; *CDERIV g x :> Db* |]
==> *CDERIV* (%*x*. *f x* * *g x*) *x :>* (*Da* * *g*(*x*)) + (*Db* * *f*(*x*))
by (*simp add*: *NSCDERIV-mult NSCDERIV-CDERIV-iff* [*symmetric*])

lemma *NSCDERIV-cmult*: *NSCDERIV f x :> D ==> NSCDERIV* (%*x*. *c* * *f x*) *x :> c*D*
apply (*simp add*: *times-divide-eq-right* [*symmetric*] *NSCDERIV-NSCLIM-iff*
minus-mult-right right-distrib [*symmetric*] *diff-minus*
del: *times-divide-eq-right minus-mult-right* [*symmetric*])
apply (*erule NSCLIM-const* [*THEN NSCLIM-mult*])
done

lemma *CDERIV-cmult*: *CDERIV f x :> D ==> CDERIV* (%*x*. *c* * *f x*) *x :> c*D*
by (*simp add*: *NSCDERIV-cmult NSCDERIV-CDERIV-iff* [*symmetric*])

lemma *NSCDERIV-minus*: *NSCDERIV f x :> D ==> NSCDERIV* (%*x*. −(*f x*)) *x :> −D*
apply (*simp add*: *NSCDERIV-NSCLIM-iff diff-minus*)
apply (*rule-tac t = f x* **in** *minus-minus* [*THEN subst*])

apply (*simp* (*no-asm-simp*) *add*: *minus-add-distrib* [*symmetric*]
del: *minus-add-distrib minus-minus*)
apply (*erule NSCLIM-minus*)
done

lemma *CDERIV-minus*: *CDERIV f x :> D ==> CDERIV* (%*x*. −(*f x*)) *x :>* −*D*
by (*simp add*: *NSCDERIV-minus NSCDERIV-CDERIV-iff* [*symmetric*])

lemma *NSCDERIV-add-minus*:
[| *NSCDERIV f x :> Da*; *NSCDERIV g x :> Db* |]
==> *NSCDERIV* (%*x*. *f x* + −*g x*) *x :> Da* + −*Db*
by (*blast dest*: *NSCDERIV-add NSCDERIV-minus*)

lemma *CDERIV-add-minus*:
[| *CDERIV f x :> Da*; *CDERIV g x :> Db* |]
==> *CDERIV* (%*x*. *f x* + −*g x*) *x :> Da* + −*Db*
by (*blast dest*: *CDERIV-add CDERIV-minus*)

lemma *NSCDERIV-diff*:
[| *NSCDERIV f x :> Da*; *NSCDERIV g x :> Db* |]
==> *NSCDERIV* (%*x*. *f x* − *g x*) *x :> Da* − *Db*
by (*simp add*: *diff-minus NSCDERIV-add-minus*)

lemma *CDERIV-diff*:
[| *CDERIV f x :> Da*; *CDERIV g x :> Db* |]
==> *CDERIV* (%*x*. *f x* − *g x*) *x :> Da* − *Db*
by (*simp add*: *diff-minus CDERIV-add-minus*)

40.9 Chain Rule

lemma *NSCDERIV-zero*:
[| *NSCDERIV g x :> D*;
(∗*f*∗ *g*) (*hcomplex-of-complex*(*x*) + *xa*) = *hcomplex-of-complex*(*g x*);
xa : *CInfinitesimal*; *xa* ≠ *0*
|] ==> *D* = *0*
apply (*simp add*: *nscderiv-def*)
apply (*drule bspec*, *auto*)
done

lemma *NSCDERIV-capprox*:
[| *NSCDERIV f x :> D*; *h*: *CInfinitesimal*; *h* ≠ *0* |]
==> (∗*f*∗ *f*) (*hcomplex-of-complex*(*x*) + *h*) − *hcomplex-of-complex*(*f x*) @*c*= *0*
apply (*simp add*: *nscderiv-def mem-cinfmal-iff* [*symmetric*])
apply (*rule CInfinitesimal-ratio*)
apply (*rule-tac* [*3*] *capprox-hcomplex-of-complex-CFinite*, *auto*)
done

lemma *NSCDERIVD1*:
[| *NSCDERIV f (g x) :> Da*;
(**f* g*) (*hcomplex-of-complex*(*x*) + *xa*) ≠ *hcomplex-of-complex (g x)*;
(**f* g*) (*hcomplex-of-complex*(*x*) + *xa*) @*c*= *hcomplex-of-complex (g x)*|]
==> ((**f* f*) ((**f* g*) (*hcomplex-of-complex*(*x*) + *xa*))
− *hcomplex-of-complex (f (g x))*) /
((**f* g*) (*hcomplex-of-complex*(*x*) + *xa*) − *hcomplex-of-complex (g x)*)
@*c*= *hcomplex-of-complex (Da)*
by (*simp add*: *NSCDERIV-NSCLIM-iff2 NSCLIM-def*)

lemma *NSCDERIVD2*:
[| *NSCDERIV g x :> Db*; *xa*: *CInfinitesimal*; *xa* ≠ *0* |]
==> ((**f* g*) (*hcomplex-of-complex x* + *xa*) − *hcomplex-of-complex*(*g x*)) / *xa*
@*c*= *hcomplex-of-complex (Db)*
by (*simp add*: *NSCDERIV-NSCLIM-iff NSCLIM-def mem-cinfmal-iff starfun-lambda-cancel*)

lemma *lemma-complex-chain*: (*z*::*hcomplex*) ≠ *0* ==> *x*y* = (*x*inverse*(*z*))*(*z*y*)
by *auto*

Chain rule

theorem *NSCDERIV-chain*:
[| *NSCDERIV f (g x) :> Da*; *NSCDERIV g x :> Db* |]
==> *NSCDERIV (f o g) x :> Da * Db*
apply (*simp* (*no-asm-simp*) *add*: *NSCDERIV-NSCLIM-iff NSCLIM-def mem-cinfmal-iff* [*symmetric*])
apply *safe*
apply (*frule-tac f* = *g* **in** *NSCDERIV-capprox*)
apply (*auto simp add*: *starfun-lambda-cancel2 starfun-o* [*symmetric*])
apply (*case-tac* (**f* g*) (*hcomplex-of-complex* (*x*) + *xa*) = *hcomplex-of-complex* (*g x*))
apply (*drule-tac g* = *g* **in** *NSCDERIV-zero*)
apply (*auto simp add*: *divide-inverse*)

apply (*rule-tac z1* = (∗*f*∗ *g*) (*hcomplex-of-complex* (*x*) + *xa*) − *hcomplex-of-complex* (*g x*) **and** *y1* = *inverse xa* **in** *lemma-complex-chain* [*THEN ssubst*])
apply (*simp* (*no-asm-simp*))
apply (*rule capprox-mult-hcomplex-of-complex*)
apply (*auto intro*!: *NSCDERIVD1 intro*: *capprox-minus-iff* [*THEN iffD2*]
simp add: *diff-minus* [*symmetric*] *divide-inverse* [*symmetric*])
apply (*blast intro*: *NSCDERIVD2*)
done

standard version

lemma *CDERIV-chain*:
[| *CDERIV f* (*g x*) :> *Da*; *CDERIV g x* :> *Db* |]
==> *CDERIV* (*f o g*) *x* :> *Da* ∗ *Db*
by (*simp add*: *NSCDERIV-CDERIV-iff* [*symmetric*] *NSCDERIV-chain*)

lemma *CDERIV-chain2*:
[| *CDERIV f* (*g x*) :> *Da*; *CDERIV g x* :> *Db* |]
==> *CDERIV* (%*x*. *f* (*g x*)) *x* :> *Da* ∗ *Db*
by (*auto dest*: *CDERIV-chain simp add*: *o-def*)

40.10 Differentiation of Natural Number Powers

lemma *NSCDERIV-Id* [*simp*]: *NSCDERIV* (%*x*. *x*) *x* :> *1*
by (*simp add*: *NSCDERIV-NSCLIM-iff NSCLIM-def divide-self del*: *divide-self-if*)

lemma *CDERIV-Id* [*simp*]: *CDERIV* (%*x*. *x*) *x* :> *1*
by (*simp add*: *NSCDERIV-CDERIV-iff* [*symmetric*])

lemmas *isContc-Id* = *CDERIV-Id* [*THEN CDERIV-isContc*, *standard*]

derivative of linear multiplication

lemma *CDERIV-cmult-Id* [*simp*]: *CDERIV* (*op* ∗ *c*) *x* :> *c*
by (*cut-tac c* = *c* **and** *x* = *x* **in** *CDERIV-Id* [*THEN CDERIV-cmult*], *simp*)

lemma *NSCDERIV-cmult-Id* [*simp*]: *NSCDERIV* (*op* ∗ *c*) *x* :> *c*
by (*simp add*: *NSCDERIV-CDERIV-iff*)

lemma *CDERIV-pow* [*simp*]:
CDERIV (%*x*. *x* ^ *n*) *x* :> (*complex-of-real* (*real n*)) ∗ (*x* ^ (*n* − *Suc 0*))
apply (*induct-tac n*)
apply (*drule-tac* [*2*] *CDERIV-Id* [*THEN CDERIV-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*:
NSCDERIV (%*x*. *x* ^ *n*) *x* :> *complex-of-real* (*real n*) ∗ (*x* ^ (*n* − *1*))

by (*simp add*: *NSCDERIV-CDERIV-iff*)

lemma *lemma-CDERIV-subst*:
[|*CDERIV f x :> D*; *D = E*|] ==> *CDERIV f x :> E*
by *auto*

lemma *CInfinitesimal-add-not-zero*:
[| *h*: *CInfinitesimal*; *x ≠ 0* |] ==> *hcomplex-of-complex x + h ≠ 0*
apply *clarify*
apply (*drule equals-zero-I*, *auto*)
done

Can't relax the premise $x \neq (0::'a)$: it isn't continuous at zero

lemma *NSCDERIV-inverse*:
x ≠ 0 ==> *NSCDERIV* (%*x*. *inverse*(*x*)) *x* :> (− (*inverse x ^ 2*))
apply (*simp add*: *nscderiv-def Ball-def*, *clarify*)
apply (*frule CInfinitesimal-add-not-zero* [**where** *x*=*x*])
apply (*auto simp add*: *starfun-inverse-inverse diff-minus*
simp del: *minus-mult-left* [*symmetric*] *minus-mult-right* [*symmetric*])
apply (*simp add*: *add-commute numeral-2-eq-2 inverse-add*
inverse-mult-distrib [*symmetric*] *inverse-minus-eq* [*symmetric*]
add-ac mult-ac
del: *inverse-minus-eq inverse-mult-distrib*
minus-mult-right [*symmetric*] *minus-mult-left* [*symmetric*])
apply (*simp only*: *mult-assoc* [*symmetric*] *right-distrib*)
apply (*rule-tac y* = *inverse* (− *hcomplex-of-complex x* ∗ *hcomplex-of-complex x*)
in *capprox-trans*)
apply (*rule inverse-add-CInfinitesimal-capprox2*)
apply (*auto dest*!: *hcomplex-of-complex-CFinite-diff-CInfinitesimal*
intro: *CFinite-mult*
simp add: *inverse-minus-eq* [*symmetric*])
apply (*rule CInfinitesimal-CFinite-mult2*, *auto*)
done

lemma *CDERIV-inverse*:
x ≠ 0 ==> *CDERIV* (%*x*. *inverse*(*x*)) *x* :> (−(*inverse x ^ 2*))
by (*simp add*: *NSCDERIV-inverse NSCDERIV-CDERIV-iff* [*symmetric*]
del: *complexpow-Suc*)

40.11 Derivative of Reciprocals (Function *inverse*)

lemma *CDERIV-inverse-fun*:
[| *CDERIV f x :> d*; *f*(*x*) ≠ *0* |]
==> *CDERIV* (%*x*. *inverse*(*f x*)) *x* :> (− (*d* ∗ *inverse*(*f*(*x*) ^ *2*)))
apply (*rule mult-commute* [*THEN subst*])
apply (*simp add*: *minus-mult-left power-inverse*
del: *complexpow-Suc minus-mult-left* [*symmetric*])
apply (*fold o-def*)

apply (*blast intro*!: *CDERIV-chain CDERIV-inverse*)
done

lemma *NSCDERIV-inverse-fun*:
[| *NSCDERIV f x :> d*; $f(x) \neq 0$ |]
==> *NSCDERIV* (%*x*. *inverse*(*f x*)) *x* :> (− (*d* ∗ *inverse*($f(x)$ ^ *2*)))
by (*simp add*: *NSCDERIV-CDERIV-iff CDERIV-inverse-fun del*: *complexpow-Suc*)

40.12 Derivative of Quotient

lemma *CDERIV-quotient*:
[| *CDERIV f x :> d*; *CDERIV g x :> e*; $g(x) \neq 0$ |]
==> *CDERIV* (%*y*. *f*(*y*) / (*g y*)) *x* :> (*d*∗*g*(*x*) − (*e*∗*f*(*x*))) / (*g*(*x*) ^ *2*)
apply (*simp add*: *diff-minus*)
apply (*drule-tac f* = *g* **in** *CDERIV-inverse-fun*)
apply (*drule-tac* [*2*] *CDERIV-mult*, *assumption*+)
apply (*simp add*: *divide-inverse right-distrib power-inverse minus-mult-left*
mult-ac
del: *minus-mult-right* [*symmetric*] *minus-mult-left* [*symmetric*]
complexpow-Suc)
done

lemma *NSCDERIV-quotient*:
[| *NSCDERIV f x :> d*; *NSCDERIV g x :> e*; $g(x) \neq 0$ |]
==> *NSCDERIV* (%*y*. *f*(*y*) / (*g y*)) *x* :> (*d*∗*g*(*x*) − (*e*∗*f*(*x*))) / (*g*(*x*) ^ *2*)
by (*simp add*: *NSCDERIV-CDERIV-iff CDERIV-quotient del*: *complexpow-Suc*)

40.13 Caratheodory Formulation of Derivative at a Point: Standard Proof

lemma *CLIM-equal*:
[| ∀ *x*. *x* ≠ *a* −−> (*f x* = *g x*) |] ==> (*f* −− *a* −−*C*> *l*) = (*g* −− *a* −−*C*> *l*)
by (*simp add*: *CLIM-def complex-add-minus-iff*)

lemma *CLIM-trans*:
[| (%*x*. *f*(*x*) + −*g*(*x*)) −− *a* −−*C*> *0*; *g* −− *a* −−*C*> *l* |] ==> *f* −− *a* −−*C*> *l*
apply (*drule CLIM-add*, *assumption*)
apply (*simp add*: *complex-add-assoc*)
done

lemma *CARAT-CDERIV*:
(*CDERIV f x :> l*) =
(∃ *g*. (∀ *z*. *f z* − *f x* = *g z* ∗ (*z* − *x*)) & *isContc g x* & *g x* = *l*)
apply *safe*
apply (*rule-tac x* = %*z*. *if z*=*x then l else* (*f* (*z*) − *f* (*x*)) / (*z*−*x*) **in** *exI*)
apply (*auto simp add*: *mult-assoc isContc-iff CDERIV-iff*)
apply (*rule-tac* [!] *CLIM-equal* [*THEN iffD1*], *auto*)

done

lemma *CARAT-NSCDERIV*:
NSCDERIV f x :> l ==>
∃ g. (∀ z. f z − f x = g z ∗ (z − x)) & isNSContc g x & g x = l
by (*simp add*: *NSCDERIV-CDERIV-iff isNSContc-isContc-iff CARAT-CDERIV*)

lemma *CARAT-CDERIVD*:
(∀ z. f z − f x = g z ∗ (z − x)) & isNSContc g x & g x = l
==> NSCDERIV f x :> l
by (*auto simp add*: *NSCDERIV-iff2 isNSContc-def cstarfun-if-eq*)

ML
⟪
val complex-add-minus-iff = thm complex-add-minus-iff;
val complex-add-eq-0-iff = thm complex-add-eq-0-iff;
val NSCLIM-NSCRLIM-Re = thm NSCLIM-NSCRLIM-Re;
val NSCLIM-NSCRLIM-Im = thm NSCLIM-NSCRLIM-Im;
val CLIM-NSCLIM = thm CLIM-NSCLIM;
val eq-Abs-star-ALL = thm eq-Abs-star-ALL;
val lemma-CLIM = thm lemma-CLIM;
val lemma-skolemize-CLIM2 = thm lemma-skolemize-CLIM2;
val lemma-csimp = thm lemma-csimp;
val NSCLIM-CLIM = thm NSCLIM-CLIM;
val CLIM-NSCLIM-iff = thm CLIM-NSCLIM-iff;
val CRLIM-NSCRLIM = thm CRLIM-NSCRLIM;
val lemma-CRLIM = thm lemma-CRLIM;
val lemma-skolemize-CRLIM2 = thm lemma-skolemize-CRLIM2;
val lemma-crsimp = thm lemma-crsimp;
val NSCRLIM-CRLIM = thm NSCRLIM-CRLIM;
val CRLIM-NSCRLIM-iff = thm CRLIM-NSCRLIM-iff;
val CLIM-CRLIM-Re = thm CLIM-CRLIM-Re;
val CLIM-CRLIM-Im = thm CLIM-CRLIM-Im;
val CLIM-cnj = thm CLIM-cnj;
val CLIM-cnj-iff = thm CLIM-cnj-iff;
val NSCLIM-add = thm NSCLIM-add;
val CLIM-add = thm CLIM-add;
val NSCLIM-mult = thm NSCLIM-mult;
val CLIM-mult = thm CLIM-mult;
val NSCLIM-const = thm NSCLIM-const;
val CLIM-const = thm CLIM-const;
val NSCLIM-minus = thm NSCLIM-minus;
val CLIM-minus = thm CLIM-minus;
val NSCLIM-diff = thm NSCLIM-diff;
val CLIM-diff = thm CLIM-diff;
val NSCLIM-inverse = thm NSCLIM-inverse;
val CLIM-inverse = thm CLIM-inverse;

val NSCLIM-zero = thm NSCLIM-zero;
val CLIM-zero = thm CLIM-zero;
val NSCLIM-zero-cancel = thm NSCLIM-zero-cancel;
val CLIM-zero-cancel = thm CLIM-zero-cancel;
val NSCLIM-not-zero = thm NSCLIM-not-zero;
val NSCLIM-not-zeroE = thms NSCLIM-not-zeroE;
val CLIM-not-zero = thm CLIM-not-zero;
val NSCLIM-const-eq = thm NSCLIM-const-eq;
val CLIM-const-eq = thm CLIM-const-eq;
val NSCLIM-unique = thm NSCLIM-unique;
val CLIM-unique = thm CLIM-unique;
val NSCLIM-mult-zero = thm NSCLIM-mult-zero;
val CLIM-mult-zero = thm CLIM-mult-zero;
val NSCLIM-self = thm NSCLIM-self;
val CLIM-self = thm CLIM-self;
val NSCLIM-NSCRLIM-iff = thm NSCLIM-NSCRLIM-iff;
val CLIM-CRLIM-iff = thm CLIM-CRLIM-iff;
val NSCLIM-NSCRLIM-iff2 = thm NSCLIM-NSCRLIM-iff2;
val NSCLIM-NSCRLIM-Re-Im-iff = thm NSCLIM-NSCRLIM-Re-Im-iff;
val CLIM-CRLIM-Re-Im-iff = thm CLIM-CRLIM-Re-Im-iff;
val isNSContcD = thm isNSContcD;
val isNSContc-NSCLIM = thm isNSContc-NSCLIM;
val NSCLIM-isNSContc = thm NSCLIM-isNSContc;
val isNSContc-NSCLIM-iff = thm isNSContc-NSCLIM-iff;
val isNSContc-CLIM-iff = thm isNSContc-CLIM-iff;
val isNSContc-isContc-iff = thm isNSContc-isContc-iff;
val isContc-isNSContc = thm isContc-isNSContc;
val isNSContc-isContc = thm isNSContc-isContc;
val NSCLIM-h-iff = thm NSCLIM-h-iff;
val NSCLIM-isContc-iff = thm NSCLIM-isContc-iff;
val CLIM-isContc-iff = thm CLIM-isContc-iff;
val isContc-iff = thm isContc-iff;
val isContc-add = thm isContc-add;
val isContc-mult = thm isContc-mult;
val isContc-o = thm isContc-o;
val isContc-o2 = thm isContc-o2;
val isNSContc-minus = thm isNSContc-minus;
val isContc-minus = thm isContc-minus;
val isContc-inverse = thm isContc-inverse;
val isNSContc-inverse = thm isNSContc-inverse;
val isContc-diff = thm isContc-diff;
val isContc-const = thm isContc-const;
val isNSContc-const = thm isNSContc-const;
val isNSContCRD = thm isNSContCRD;
val isNSContCR-NSCRLIM = thm isNSContCR-NSCRLIM;
val NSCRLIM-isNSContCR = thm NSCRLIM-isNSContCR;
val isNSContCR-NSCRLIM-iff = thm isNSContCR-NSCRLIM-iff;
val isNSContCR-CRLIM-iff = thm isNSContCR-CRLIM-iff;
val isNSContCR-isContCR-iff = thm isNSContCR-isContCR-iff;

val isContCR-isNSContCR = thm isContCR-isNSContCR;
val isNSContCR-isContCR = thm isNSContCR-isContCR;
val isNSContCR-cmod = thm isNSContCR-cmod;
val isContCR-cmod = thm isContCR-cmod;
val isContc-isContCR-Re = thm isContc-isContCR-Re;
val isContc-isContCR-Im = thm isContc-isContCR-Im;
val CDERIV-iff = thm CDERIV-iff;
val CDERIV-NSC-iff = thm CDERIV-NSC-iff;
val CDERIVD = thm CDERIVD;
val NSC-DERIVD = thm NSC-DERIVD;
val CDERIV-unique = thm CDERIV-unique;
val NSCDeriv-unique = thm NSCDeriv-unique;
val CDERIV-CLIM-iff = thm CDERIV-CLIM-iff;
val CDERIV-iff2 = thm CDERIV-iff2;
val NSCDERIV-NSCLIM-iff = thm NSCDERIV-NSCLIM-iff;
val NSCDERIV-NSCLIM-iff2 = thm NSCDERIV-NSCLIM-iff2;
val NSCDERIV-iff2 = thm NSCDERIV-iff2;
val NSCDERIV-CDERIV-iff = thm NSCDERIV-CDERIV-iff;
val NSCDERIV-isNSContc = thm NSCDERIV-isNSContc;
val CDERIV-isContc = thm CDERIV-isContc;
val NSCDERIV-const = thm NSCDERIV-const;
val CDERIV-const = thm CDERIV-const;
val NSCDERIV-add = thm NSCDERIV-add;
val CDERIV-add = thm CDERIV-add;
val lemma-nscderiv1 = thm lemma-nscderiv1;
val lemma-nscderiv2 = thm lemma-nscderiv2;
val NSCDERIV-mult = thm NSCDERIV-mult;
val CDERIV-mult = thm CDERIV-mult;
val NSCDERIV-cmult = thm NSCDERIV-cmult;
val CDERIV-cmult = thm CDERIV-cmult;
val NSCDERIV-minus = thm NSCDERIV-minus;
val CDERIV-minus = thm CDERIV-minus;
val NSCDERIV-add-minus = thm NSCDERIV-add-minus;
val CDERIV-add-minus = thm CDERIV-add-minus;
val NSCDERIV-diff = thm NSCDERIV-diff;
val CDERIV-diff = thm CDERIV-diff;
val NSCDERIV-zero = thm NSCDERIV-zero;
val NSCDERIV-capprox = thm NSCDERIV-capprox;
val NSCDERIVD1 = thm NSCDERIVD1;
val NSCDERIVD2 = thm NSCDERIVD2;
val lemma-complex-chain = thm lemma-complex-chain;
val NSCDERIV-chain = thm NSCDERIV-chain;
val CDERIV-chain = thm CDERIV-chain;
val CDERIV-chain2 = thm CDERIV-chain2;
val NSCDERIV-Id = thm NSCDERIV-Id;
val CDERIV-Id = thm CDERIV-Id;
val isContc-Id = thms isContc-Id;
val CDERIV-cmult-Id = thm CDERIV-cmult-Id;
val NSCDERIV-cmult-Id = thm NSCDERIV-cmult-Id;

val CDERIV-pow = thm CDERIV-pow;
val NSCDERIV-pow = thm NSCDERIV-pow;
val lemma-CDERIV-subst = thm lemma-CDERIV-subst;
val CInfinitesimal-add-not-zero = thm CInfinitesimal-add-not-zero;
val NSCDERIV-inverse = thm NSCDERIV-inverse;
val CDERIV-inverse = thm CDERIV-inverse;
val CDERIV-inverse-fun = thm CDERIV-inverse-fun;
val NSCDERIV-inverse-fun = thm NSCDERIV-inverse-fun;
val lemma-complex-mult-inverse-squared = thm lemma-complex-mult-inverse-squared;
val CDERIV-quotient = thm CDERIV-quotient;
val NSCDERIV-quotient = thm NSCDERIV-quotient;
val CLIM-equal = thm CLIM-equal;
val CLIM-trans = thm CLIM-trans;
val CARAT-CDERIV = thm CARAT-CDERIV;
val CARAT-NSCDERIV = thm CARAT-NSCDERIV;
val CARAT-CDERIVD = thm CARAT-CDERIVD;
⟩⟩

end

41 Complex-Main: Comprehensive Complex Theory

theory *Complex-Main*
imports *CLim*
begin

end