The Hahn-Banach Theorem for Real Vector Spaces

Gertrud Bauer
http://www.in.tum.de/~bauerg/

October 31, 1999

Abstract

The Hahn-Banach Theorem is one of the most fundamental results in functional analysis. We present a fully formal proof of two versions of the theorem, one for general linear spaces and another for normed spaces. This development is based on simply-typed classical set-theory, as provided by Isabelle/HOL.

Contents

1 Preface

This is a fully formal proof of the Hahn-Banach Theorem. It closely follows the informal presentation given in the textbook [1, §36]. Another formal proof of the same theorem has been done in Mizar [3]. A general overview of the relevance and history of the Hahn-Banach Theorem is given in [2].

The document is structured as follows. The first part contains definitions of basic notions of linear algebra: vector spaces, subspaces, normed spaces, continuous linearforms, norm of functions and an order on functions by domain extension. The second part contains some lemmas about the supremum (w.r.t. the function order) and extension of non-maximal functions. With these preliminaries, the main proof of the theorem (in its two versions) is conducted in the third part.

Part I
Basic Notions

2 Bounds

theory *Bounds = Main + Real:*

A supremum[1] of an ordered set B w. r. t. A is defined as a least upper bound of B, which lies in A.

If a supremum exists, then $Sup\,A\,B$ is equal to the supremum.

```
constdefs
  is_Sup :: "('a::order) set => 'a set => 'a => bool"
  "is_Sup A B x == isLub A B x"

  Sup :: "('a::order) set => 'a set => 'a"
  "Sup A B == Eps (is_Sup A B)"
```

The supremum of B is less than any upper bound of B.

```
lemma sup_le_ub: "isUb A B y ==> is_Sup A B s ==> s <= y"
  by (unfold is_Sup_def, rule isLub_le_isUb)
```

The supremum B is an upper bound for B.

```
lemma sup_ub: "y:B ==> is_Sup A B s ==> y <= s"
  by (unfold is_Sup_def, rule isLubD2)
```

The supremum of a non-empty set B is greater than a lower bound of B.

```
lemma sup_ub1:
  "[| ALL y:B. a <= y; is_Sup A B s; x:B |] ==> a <= s"
proof -
  assume "ALL y:B. a <= y" "is_Sup A B s" "x:B"
  have "a <= x" by (rule bspec)
  also have "x <= s" by (rule sup_ub)
  finally show "a <= s" .
qed

end
```

[1] The definition of the supremum is based on one in http://isabelle.in.tum.de/library/HOL/HOL-Real/Lubs.html

3 Auxiliary theorems

theory *Aux = Real + Zorn:*

Some existing theorems are declared as extra introduction or elimination rules, respectively.

lemmas *[intro!!] = isLub_isUb*
lemmas *[intro!!] = chainD*
lemmas *chainE2 = chainD2 [elimify]*

Lemmas about sets.

lemma *Int_singletonD: "[| A Int B = {v}; x:A; x:B |] ==> x = v"*
  by *(fast elim: equalityE)*

lemma *set_less_imp_diff_not_empty: "H < E ==> EX x0:E. x0 ~: H"*
 by *(force simp add: psubset_eq)*

Some lemmas about orders.

lemma *lt_imp_not_eq: "x < (y::'a::order) ==> x ~= y"*
  by *(rule order_less_le[RS iffD1, RS conjunct2])*

lemma *le_noteq_imp_less:*
  "[| x <= (r::'a::order); x ~= r |] ==> x < r"
proof -
  assume *"x <= (r::'a::order)"* **and** *ne:"x ~= r"*
  hence *"x < r | x = r"* **by** *(simp add: order_le_less)*
  with *ne* **show** *?thesis* **by** *simp*
qed

Some lemmas about linear orders.

theorem *linorder_linear_split:*
"[| x < a ==> Q; x = a ==> Q; a < (x::'a::linorder) ==> Q |] ==> Q"
  by *(rule linorder_less_linear [of x a, elimify]) force+*

lemma *le_max1: "x <= max x (y::'a::linorder)"*
  by *(simp add: le_max_iff_disj[of x x y])*

lemma *le_max2: "y <= max x (y::'a::linorder)"*
  by *(simp add: le_max_iff_disj[of y x y])*

Some lemmas for the reals.

lemma *real_add_minus_eq: "x - y = 0r ==> x = y"*
proof -
  assume *"x - y = 0r"*
  have *"x + - y = 0r"* **by** *(simp!)*
  hence *"x = - (- y)"* **by** *(rule real_add_minus_eq_minus)*

```
  also have "... = y" by simp
  finally show "?thesis" .
qed

lemma rabs_minus_one: "rabs (- 1r) = 1r"
proof -
  have "-1r < 0r"
    by (rule real_minus_zero_less_iff[RS iffD1], simp,
        rule real_zero_less_one)
  hence "rabs (- 1r) = - (- 1r)"
    by (rule rabs_minus_eqI2)
  also have "... = 1r" by simp
  finally show ?thesis .
qed

lemma real_mult_le_le_mono2:
  "[| 0r <= z; x <= y |] ==> x * z <= y * z"
proof -
  assume "0r <= z" "x <= y"
  hence "x < y | x = y" by (force simp add: order_le_less)
  thus ?thesis
  proof (elim disjE)
    assume "x < y" show ?thesis by (rule real_mult_le_less_mono1)
  next
    assume "x = y" thus ?thesis by simp
  qed
qed

lemma real_mult_less_le_anti:
  "[| z < 0r; x <= y |] ==> z * y <= z * x"
proof -
  assume "z < 0r" "x <= y"
  hence "0r < - z" by simp
  hence "0r <= - z" by (rule real_less_imp_le)
  hence "(- z) * x <= (- z) * y"
    by (rule real_mult_le_le_mono1)
  hence  "- (z * x) <= - (z * y)"
    by (simp only: real_minus_mult_eq1)
  thus ?thesis by simp
qed

lemma real_mult_less_le_mono:
  "[| 0r < z; x <= y |] ==> z * x <= z * y"
proof -
  assume "0r < z" "x <= y"
  have "0r <= z" by (rule real_less_imp_le)
  thus ?thesis by (rule real_mult_le_le_mono1)
qed
```

```
lemma real_mult_diff_distrib:
  "a * (- x - (y::real)) = - a * x - a * y"
proof -
  have "- x - y = - x + - y" by simp
  also have "a * ... = a * - x + a * - y"
    by (simp only: real_add_mult_distrib2)
  also have "... = - a * x - a * y"
    by (simp add: real_minus_mult_eq2 [RS sym] real_minus_mult_eq1)
  finally show ?thesis .
qed

lemma real_mult_diff_distrib2: "a * (x - (y::real)) = a * x - a * y"
proof -
  have "x - y = x + - y" by simp
  also have "a * ... = a * x + a * - y"
    by (simp only: real_add_mult_distrib2)
  also have "... = a * x - a * y"
    by (simp add: real_minus_mult_eq2 [RS sym] real_minus_mult_eq1)
  finally show ?thesis .
qed

lemma real_minus_le: "- (x::real) <= y ==> - y <= x"
  by simp

lemma real_diff_ineq_swap:
  "(d::real) - b <= c + a ==> - a - b <= c - d"
  by simp

end
```

4 Vector spaces

```
theory VectorSpace = Bounds + Aux:
```

4.1 Signature

For the definition of real vector spaces a type α of the sort $\{plus, minus\}$ is considered, on which a real scalar multiplication $\odot$, and a zero element $\mathbf{0}$ is defined.

```
consts
  prod  :: "[real, 'a] => 'a"                    (infixr "<*>" 70)
  zero  :: 'a                                    ("<0>")

syntax (symbols)
  prod  :: "[real, 'a] => 'a"                    (infixr "⊙" 70)
  zero  :: 'a                                    ("0")
```

4.2 Vector space laws

A *vector space* is a non-empty set V of elements from α with the following vector space laws: The set V is closed under addition and scalar multiplication, addition is associative and commutative; $-x$ is the inverse of x w. r. t. addition and $\mathbf{0}$ is the neutral element of addition. Addition and multiplication are distributive; scalar multiplication is associative and the real number 1 is the neutral element of scalar multiplication.

constdefs

```
  is_vectorspace :: "('a::{plus,minus}) set => bool"
  "is_vectorspace V == V ~= {}
   & (ALL x:V. ALL y:V. ALL z:V. ALL a b.
        x + y : V
      & a <*> x : V
      & (x + y) + z = x + (y + z)
      & x + y = y + x
      & x - x = <0>
      & <0> + x = x
      & a <*> (x + y) = a <*> x + a <*> y
      & (a + b) <*> x = a <*> x + b <*> x
      & (a * b) <*> x = a <*> b <*> x
      & 1r <*> x = x
      & - x = (- 1r) <*> x
      & x - y = x + - y)"
```

The corresponding introduction rule is:

lemma *vsI [intro]:*

```
  "[| <0>:V;
  ALL x:V. ALL y:V. x + y : V;
  ALL x:V. ALL a. a <*> x : V;
  ALL x:V. ALL y:V. ALL z:V. (x + y) + z = x + (y + z);
  ALL x:V. ALL y:V. x + y = y + x;
  ALL x:V. x - x = <0>;
  ALL x:V. <0> + x = x;
  ALL x:V. ALL y:V. ALL a. a <*> (x + y) = a <*> x + a <*> y;
  ALL x:V. ALL a b. (a + b) <*> x = a <*> x + b <*> x;
  ALL x:V. ALL a b. (a * b) <*> x = a <*> b <*> x;
  ALL x:V. 1r <*> x = x;
  ALL x:V. - x = (- 1r) <*> x;
  ALL x:V. ALL y:V. x - y = x + - y |] ==> is_vectorspace V"
```

proof *(unfold is_vectorspace_def, intro conjI ballI allI)*

```
  fix x y z
  assume "x:V" "y:V" "z:V"
    "ALL x:V. ALL y:V. ALL z:V. x + y + z = x + (y + z)"
  thus "x + y + z =  x + (y + z)" by (elim bspec[elimify])
```

qed *force+*

The corresponding destruction rules are:

lemma *negate_eq1:*
```
  "[| is_vectorspace V; x:V |] ==> - x = (- 1r) <*> x"
```
by *(unfold is_vectorspace_def) simp*

lemma *diff_eq1:*
```
  "[| is_vectorspace V; x:V; y:V |] ==> x - y = x + - y"
```
by *(unfold is_vectorspace_def) simp*

lemma *negate_eq2:*
```
  "[| is_vectorspace V; x:V |] ==> (- 1r) <*> x = - x"
```
by *(unfold is_vectorspace_def) simp*

lemma *diff_eq2:*
```
  "[| is_vectorspace V; x:V; y:V |] ==> x + - y = x - y"
```
by *(unfold is_vectorspace_def) simp*

lemma *vs_not_empty [intro !!]: "is_vectorspace V ==> (V ~= {})"*
by *(unfold is_vectorspace_def) simp*

lemma *vs_add_closed [simp, intro!!]:*
```
  "[| is_vectorspace V; x:V; y:V |] ==> x + y : V"
```
by *(unfold is_vectorspace_def) simp*

lemma *vs_mult_closed [simp, intro!!]:*
```
  "[| is_vectorspace V; x:V |] ==> a <*> x : V"
```
by *(unfold is_vectorspace_def) simp*

lemma *vs_diff_closed [simp, intro!!]:*
```
 "[| is_vectorspace V; x:V; y:V |] ==> x - y : V"
```
by *(simp add: diff_eq1 negate_eq1)*

lemma *vs_neg_closed [simp, intro!!]:*
```
  "[| is_vectorspace V; x:V |] ==> - x : V"
```
by *(simp add: negate_eq1)*

lemma *vs_add_assoc [simp]:*
```
  "[| is_vectorspace V; x:V; y:V; z:V |]
   ==> (x + y) + z = x + (y + z)"
```
by *(unfold is_vectorspace_def) fast*

lemma *vs_add_commute [simp]:*
```
  "[| is_vectorspace V; x:V; y:V |] ==> y + x = x + y"
```
by *(unfold is_vectorspace_def) simp*

lemma *vs_add_left_commute [simp]:*
```
  "[| is_vectorspace V; x:V; y:V; z:V |]
  ==> x + (y + z) = y + (x + z)"
```
proof -
```
  assume "is_vectorspace V" "x:V" "y:V" "z:V"
```

```
  hence "x + (y + z) = (x + y) + z"
    by (simp only: vs_add_assoc)
  also have "... = (y + x) + z" by (simp! only: vs_add_commute)
  also have "... = y + (x + z)" by (simp! only: vs_add_assoc)
  finally show ?thesis .
qed

theorems vs_add_ac = vs_add_assoc vs_add_commute vs_add_left_commute

lemma vs_diff_self [simp]:
  "[| is_vectorspace V; x:V |] ==>  x - x = <0>"
  by (unfold is_vectorspace_def) simp
```

The existence of the zero element of a vector space follows from the non-emptiness of carrier set.

```
lemma zero_in_vs [simp, intro]: "is_vectorspace V ==> <0>:V"
proof -
  assume "is_vectorspace V"
  have "V ~= {}" ..
  hence "EX x. x:V" by force
  thus ?thesis
  proof
    fix x assume "x:V"
    have "<0> = x - x" by (simp!)
    also have "... : V" by (simp! only: vs_diff_closed)
    finally show ?thesis .
  qed
qed

lemma vs_add_zero_left [simp]:
  "[| is_vectorspace V; x:V |] ==>  <0> + x = x"
  by (unfold is_vectorspace_def) simp

lemma vs_add_zero_right [simp]:
  "[| is_vectorspace V; x:V |] ==>  x + <0> = x"
proof -
  assume "is_vectorspace V" "x:V"
  hence "x + <0> = <0> + x" by simp
  also have "... = x" by (simp!)
  finally show ?thesis .
qed

lemma vs_add_mult_distrib1:
  "[| is_vectorspace V; x:V; y:V |]
  ==> a <*> (x + y) = a <*> x + a <*> y"
  by (unfold is_vectorspace_def) simp

lemma vs_add_mult_distrib2:
  "[| is_vectorspace V; x:V |]
```

```
  ==> (a + b) <*> x = a <*> x + b <*> x"
  by (unfold is_vectorspace_def) simp

lemma vs_mult_assoc:
  "[| is_vectorspace V; x:V |] ==> (a * b) <*> x = a <*> (b <*> x)"
  by (unfold is_vectorspace_def) simp

lemma vs_mult_assoc2 [simp]:
 "[| is_vectorspace V; x:V |] ==> a <*> b <*> x = (a * b) <*> x"
  by (simp only: vs_mult_assoc)

lemma vs_mult_1 [simp]:
  "[| is_vectorspace V; x:V |] ==> 1r <*> x = x"
  by (unfold is_vectorspace_def) simp

lemma vs_diff_mult_distrib1:
  "[| is_vectorspace V; x:V; y:V |]
  ==> a <*> (x - y) = a <*> x - a <*> y"
  by (simp add: diff_eq1 negate_eq1 vs_add_mult_distrib1)

lemma vs_diff_mult_distrib2:
  "[| is_vectorspace V; x:V |]
  ==> (a - b) <*> x = a <*> x - (b <*> x)"
proof -
  assume "is_vectorspace V" "x:V"
  have " (a - b) <*> x = (a + - b ) <*> x"
    by (unfold real_diff_def, simp)
  also have "... = a <*> x + (- b) <*> x"
    by (rule vs_add_mult_distrib2)
  also have "... = a <*> x + - (b <*> x)"
    by (simp! add: negate_eq1)
  also have "... = a <*> x - (b <*> x)"
    by (simp! add: diff_eq1)
  finally show ?thesis .
qed
```

Further derived laws:

```
lemma vs_mult_zero_left [simp]:
  "[| is_vectorspace V; x:V |] ==> 0r <*> x = <0>"
proof -
  assume "is_vectorspace V" "x:V"
  have  "0r <*> x = (1r - 1r) <*> x" by (simp only: real_diff_self)
  also have "... = (1r + - 1r) <*> x" by simp
  also have "... =  1r <*> x + (- 1r) <*> x"
    by (rule vs_add_mult_distrib2)
  also have "... = x + (- 1r) <*> x" by (simp!)
  also have "... = x + - x" by (simp! add: negate_eq2)
  also have "... = x - x" by (simp! add: diff_eq2)
  also have "... = <0>" by (simp!)
```

```
  finally show ?thesis .
qed

lemma vs_mult_zero_right [simp]:
  "[| is_vectorspace (V:: 'a::{plus, minus} set) |]
  ==> a <*> <0> = (<0>::'a)"
proof -
  assume "is_vectorspace V"
  have "a <*> <0> = a <*> (<0> - (<0>::'a))" by (simp!)
  also have "... =  a <*> <0> - a <*> <0>"
     by (rule vs_diff_mult_distrib1) (simp!)+
  also have "... = <0>" by (simp!)
  finally show ?thesis .
qed

lemma vs_minus_mult_cancel [simp]:
  "[| is_vectorspace V; x:V |] ==> (- a) <*> - x = a <*> x"
  by (simp add: negate_eq1)

lemma vs_add_minus_left_eq_diff:
  "[| is_vectorspace V; x:V; y:V |] ==> - x + y = y - x"
proof -
  assume "is_vectorspace V" "x:V" "y:V"
  have "- x + y = y + - x"
    by (simp! add: vs_add_commute [RS sym, of V "- x"])
  also have "... = y - x" by (simp! add: diff_eq1)
  finally show ?thesis .
qed

lemma vs_add_minus [simp]:
  "[| is_vectorspace V; x:V |] ==> x + - x = <0>"
  by (simp! add: diff_eq2)

lemma vs_add_minus_left [simp]:
  "[| is_vectorspace V; x:V |] ==> - x + x = <0>"
  by (simp! add: diff_eq2)

lemma vs_minus_minus [simp]:
  "[| is_vectorspace V; x:V |] ==> - (- x) = x"
  by (simp add: negate_eq1)

lemma vs_minus_zero [simp]:
  "is_vectorspace (V::'a::{minus, plus} set) ==> - (<0>::'a) = <0>"
  by (simp add: negate_eq1)

lemma vs_minus_zero_iff [simp]:
  "[| is_vectorspace V; x:V |] ==> (- x = <0>) = (x = <0>)"
  (concl is "?L = ?R")
proof -
```

```
  assume "is_vectorspace V" "x:V"
  show "?L = ?R"
  proof
    have "x = - (- x)" by (rule vs_minus_minus [RS sym])
    also assume ?L
    also have "- ... = <0>" by (rule vs_minus_zero)
    finally show ?R .
  qed (simp!)
qed

lemma vs_add_minus_cancel [simp]:
  "[| is_vectorspace V; x:V; y:V |] ==> x + (- x + y) = y"
  by (simp add: vs_add_assoc [RS sym] del: vs_add_commute)

lemma vs_minus_add_cancel [simp]:
  "[| is_vectorspace V; x:V; y:V |] ==> - x + (x + y) = y"
  by (simp add: vs_add_assoc [RS sym] del: vs_add_commute)

lemma vs_minus_add_distrib [simp]:
  "[| is_vectorspace V; x:V; y:V |]
  ==> - (x + y) = - x + - y"
  by (simp add: negate_eq1 vs_add_mult_distrib1)

lemma vs_diff_zero [simp]:
  "[| is_vectorspace V; x:V |] ==> x - <0> = x"
  by (simp add: diff_eq1)

lemma vs_diff_zero_right [simp]:
  "[| is_vectorspace V; x:V |] ==> <0> - x = - x"
  by (simp add:diff_eq1)

lemma vs_add_left_cancel:
  "[| is_vectorspace V; x:V; y:V; z:V |]
   ==> (x + y = x + z) = (y = z)"
  (concl is "?L = ?R")
proof
  assume "is_vectorspace V" "x:V" "y:V" "z:V"
  have "y = <0> + y" by (simp!)
  also have "... = - x + x + y" by (simp!)
  also have "... = - x + (x + y)"
    by (simp! only: vs_add_assoc vs_neg_closed)
  also assume ?L
  also have "- x + ... = - x + x + z"
    by (rule vs_add_assoc [RS sym]) (simp!)+
  also have "... = z" by (simp!)
  finally show ?R .
qed force

lemma vs_add_right_cancel:
```

```
  "[| is_vectorspace V; x:V; y:V; z:V |]
  ==> (y + x = z + x) = (y = z)"
  by (simp only: vs_add_commute vs_add_left_cancel)

lemma vs_add_assoc_cong:
  "[| is_vectorspace V; x:V; y:V; x':V; y':V; z:V |]
  ==> x + y = x' + y' ==> x + (y + z) = x' + (y' + z)"
  by (simp only: vs_add_assoc [RS sym])

lemma vs_mult_left_commute:
  "[| is_vectorspace V; x:V; y:V; z:V |]
  ==> x <*> y <*> z = y <*> x <*> z"
  by (simp add: real_mult_commute)

lemma vs_mult_zero_uniq :
  "[| is_vectorspace V; x:V; a <*> x = <0>; x ~= <0> |] ==> a = 0r"
proof (rule classical)
  assume "is_vectorspace V" "x:V" "a <*> x = <0>" "x ~= <0>"
  assume "a ~= 0r"
  have "x = (rinv a * a) <*> x" by (simp!)
  also have "... = rinv a <*> (a <*> x)" by (rule vs_mult_assoc)
  also have "... = rinv a <*> <0>" by (simp!)
  also have "... = <0>" by (simp!)
  finally have "x = <0>" .
  thus "a = 0r" by contradiction
qed

lemma vs_mult_left_cancel:
  "[| is_vectorspace V; x:V; y:V; a ~= 0r |] ==>
  (a <*> x = a <*> y) = (x = y)"
  (concl is "?L = ?R")
proof
  assume "is_vectorspace V" "x:V" "y:V" "a ~= 0r"
  have "x = 1r <*> x" by (simp!)
  also have "... = (rinv a * a) <*> x" by (simp!)
  also have "... = rinv a <*> (a <*> x)"
    by (simp! only: vs_mult_assoc)
  also assume ?L
  also have "rinv a <*> ... = y" by (simp!)
  finally show ?R .
qed simp

lemma vs_mult_right_cancel:
  "[| is_vectorspace V; x:V; x ~= <0> |]
  ==> (a <*> x = b <*> x) = (a = b)" (concl is "?L = ?R")
proof
  assume "is_vectorspace V" "x:V" "x ~= <0>"
  have "(a - b) <*> x = a <*> x - b <*> x"
    by (simp! add: vs_diff_mult_distrib2)
```

```
  also assume ?L hence "a <*> x - b <*> x = <0>" by (simp!)
  finally have "(a - b) <*> x = <0>" .
  hence "a - b = 0r" by (simp! add: vs_mult_zero_uniq)
  thus "a = b" by (rule real_add_minus_eq)
qed simp

lemma vs_eq_diff_eq:
  "[| is_vectorspace V; x:V; y:V; z:V |] ==>
  (x = z - y) = (x + y = z)"
  (concl is "?L = ?R" )
proof -
  assume vs: "is_vectorspace V" "x:V" "y:V" "z:V"
  show "?L = ?R"
  proof
    assume ?L
    hence "x + y = z - y + y" by simp
    also have "... = z + - y + y" by (simp! add: diff_eq1)
    also have "... = z + (- y + y)"
      by (rule vs_add_assoc) (simp!)+
    also from vs have "... = z + <0>"
      by (simp only: vs_add_minus_left)
    also from vs have "... = z" by (simp only: vs_add_zero_right)
    finally show ?R .
  next
    assume ?R
    hence "z - y = (x + y) - y" by simp
    also from vs have "... = x + y + - y"
      by (simp add: diff_eq1)
    also have "... = x + (y + - y)"
      by (rule vs_add_assoc) (simp!)+
    also have "... = x" by (simp!)
    finally show ?L by (rule sym)
  qed
qed

lemma vs_add_minus_eq_minus:
  "[| is_vectorspace V; x:V; y:V; x + y = <0> |] ==> x = - y"
proof -
  assume "is_vectorspace V" "x:V" "y:V"
  have "x = (- y + y) + x" by (simp!)
  also have "... = - y + (x + y)" by (simp!)
  also assume "x + y = <0>"
  also have "- y + <0> = - y" by (simp!)
  finally show "x = - y" .
qed

lemma vs_add_minus_eq:
  "[| is_vectorspace V; x:V; y:V; x - y = <0> |] ==> x = y"
proof -
```

```
  assume "is_vectorspace V" "x:V" "y:V" "x - y = <0>"
  assume "x - y = <0>"
  hence e: "x + - y = <0>" by (simp! add: diff_eq1)
  with _ _ _ have "x = - (- y)"
    by (rule vs_add_minus_eq_minus) (simp!)+
  thus "x = y" by (simp!)
qed

lemma vs_add_diff_swap:
  "[| is_vectorspace V; a:V; b:V; c:V; d:V; a + b = c + d |]
  ==> a - c = d - b"
proof -
  assume vs: "is_vectorspace V" "a:V" "b:V" "c:V" "d:V"
    and eq: "a + b = c + d"
  have "- c + (a + b) = - c + (c + d)"
    by (simp! add: vs_add_left_cancel)
  also have "... = d" by (rule vs_minus_add_cancel)
  finally have eq: "- c + (a + b) = d" .
  from vs have "a - c = (- c + (a + b)) + - b"
    by (simp add: vs_add_ac diff_eq1)
  also from eq have "...  = d + - b"
    by (simp! add: vs_add_right_cancel)
  also have "... = d - b" by (simp! add : diff_eq2)
  finally show "a - c = d - b" .
qed

lemma vs_add_cancel_21:
  "[| is_vectorspace V; x:V; y:V; z:V; u:V |]
  ==> (x + (y + z) = y + u) = ((x + z) = u)"
  (concl is "?L = ?R")
proof -
  assume "is_vectorspace V" "x:V" "y:V""z:V" "u:V"
  show "?L = ?R"
  proof
    have "x + z = - y + y + (x + z)" by (simp!)
    also have "... = - y + (y + (x + z))"
      by (rule vs_add_assoc) (simp!)+
    also have "y + (x + z) = x + (y + z)" by (simp!)
    also assume ?L
    also have "- y + (y + u) = u" by (simp!)
    finally show ?R .
  qed (simp! only: vs_add_left_commute [of V x])
qed

lemma vs_add_cancel_end:
  "[| is_vectorspace V;  x:V; y:V; z:V |]
  ==> (x + (y + z) = y) = (x = - z)"
  (concl is "?L = ?R" )
proof -
```

```
  assume "is_vectorspace V" "x:V" "y:V" "z:V"
  show "?L = ?R"
  proof
    assume l: ?L
    have "x + z = <0>"
    proof (rule vs_add_left_cancel [RS iffD1])
      have "y + (x + z) = x + (y + z)" by (simp!)
      also note l
      also have "y = y + <0>" by (simp!)
      finally show "y + (x + z) = y + <0>" .
    qed (simp!)+
    thus "x = - z" by (simp! add: vs_add_minus_eq_minus)
  next
    assume r: ?R
    hence "x + (y + z) = - z + (y + z)" by simp
    also have "... = y + (- z + z)"
      by (simp! only: vs_add_left_commute)
    also have "... = y"  by (simp!)
    finally show ?L .
  qed
qed

end
```

5 Subspaces

```
theory Subspace = VectorSpace:
```

5.1 Definition

A non-empty subset U of a vector space V is a *subspace* of V, iff U is closed under addition and scalar multiplication.

```
constdefs
  is_subspace ::  "['a::{minus, plus} set, 'a set] => bool"
  "is_subspace U V == U ~= {} & U <= V
     & (ALL x:U. ALL y:U. ALL a. x + y : U & a <*> x : U)"

lemma subspaceI [intro]:
  "[| <0> : U; U <= V; ALL x:U. ALL y:U. (x + y : U);
  ALL x:U. ALL a. a <*> x : U |]
  ==> is_subspace U V"
proof (unfold is_subspace_def, intro conjI)
  assume "<0> : U" thus "U ~= {}" by fast
qed (simp+)

lemma subspace_not_empty [intro!!]: "is_subspace U V ==> U ~= {}"
  by (unfold is_subspace_def) simp
```

```
lemma subspace_subset [intro !!]: "is_subspace U V ==> U <= V"
  by (unfold is_subspace_def) simp

lemma subspace_subsetD [simp, intro!!]:
  "[| is_subspace U V; x:U |] ==> x:V"
  by (unfold is_subspace_def) force

lemma subspace_add_closed [simp, intro!!]:
  "[| is_subspace U V; x:U; y:U |] ==> x + y : U"
  by (unfold is_subspace_def) simp

lemma subspace_mult_closed [simp, intro!!]:
  "[| is_subspace U V; x:U |] ==> a <*> x : U"
  by (unfold is_subspace_def) simp

lemma subspace_diff_closed [simp, intro!!]:
  "[| is_subspace U V; is_vectorspace V; x:U; y:U |]
  ==> x - y : U"
  by (simp! add: diff_eq1 negate_eq1)
```

Similar as for linear spaces, the existence of the zero element in every subspace follows from the non-emptiness of the carrier set and by vector space laws.

```
lemma zero_in_subspace [intro !!]:
  "[| is_subspace U V; is_vectorspace V |] ==> <0> : U"
proof -
  assume "is_subspace U V" and v: "is_vectorspace V"
  have "U ~= {}" ..
  hence "EX x. x:U" by force
  thus ?thesis
  proof
    fix x assume u: "x:U"
    hence "x:V" by (simp!)
    with v have "<0> = x - x" by (simp!)
    also have "... : U" by (rule subspace_diff_closed)
    finally show ?thesis .
  qed
qed

lemma subspace_neg_closed [simp, intro!!]:
  "[| is_subspace U V; is_vectorspace V; x:U |] ==> - x : U"
  by (simp add: negate_eq1)
```

Further derived laws: every subspace is a vector space.

```
lemma subspace_vs [intro!!]:
  "[| is_subspace U V; is_vectorspace V |] ==> is_vectorspace U"
proof -
```

```
  assume "is_subspace U V" "is_vectorspace V"
  show ?thesis
  proof
    show "<0> : U" ..
    show "ALL x:U. ALL a. a <*> x : U" by (simp!)
    show "ALL x:U. ALL y:U. x + y : U" by (simp!)
    show "ALL x:U. - x = -1r <*> x" by (simp! add: negate_eq1)
    show "ALL x:U. ALL y:U. x - y =  x + - y"
      by (simp! add: diff_eq1)
  qed (simp! add: vs_add_mult_distrib1 vs_add_mult_distrib2)+
qed
```

The subspace relation is reflexive.

```
lemma subspace_refl [intro]: "is_vectorspace V ==> is_subspace V V"
proof
  assume "is_vectorspace V"
  show "<0> : V" ..
  show "V <= V" ..
  show "ALL x:V. ALL y:V. x + y : V" by (simp!)
  show "ALL x:V. ALL a. a <*> x : V" by (simp!)
qed
```

The subspace relation is transitive.

```
lemma subspace_trans:
  "[| is_subspace U V; is_vectorspace V; is_subspace V W |]
  ==> is_subspace U W"
proof
  assume "is_subspace U V" "is_subspace V W" "is_vectorspace V"
  show "<0> : U" ..

  have "U <= V" ..
  also have "V <= W" ..
  finally show "U <= W" .

  show "ALL x:U. ALL y:U. x + y : U"
  proof (intro ballI)
    fix x y assume "x:U" "y:U"
    show "x + y : U" by (simp!)
  qed

  show "ALL x:U. ALL a. a <*> x : U"
  proof (intro ballI allI)
    fix x a assume "x:U"
    show "a <*> x : U" by (simp!)
  qed
qed
```

5.2 Linear closure

The *linear closure* of a vector x is the set of all scalar multiples of x.

```
constdefs
  lin :: "'a => 'a set"
  "lin x == {a <*> x | a. True}"

lemma linD: "x : lin v = (EX a::real. x = a <*> v)"
  by (unfold lin_def) fast

lemma linI [intro!!]: "a <*> x0 : lin x0"
  by (unfold lin_def) fast
```

Every vector is contained in its linear closure.

```
lemma x_lin_x: "[| is_vectorspace V; x:V |] ==> x : lin x"
proof (unfold lin_def, intro CollectI exI conjI)
  assume "is_vectorspace V" "x:V"
  show "x = 1r <*> x" by (simp!)
qed simp
```

Any linear closure is a subspace.

```
lemma lin_subspace [intro!!]:
  "[| is_vectorspace V; x:V |] ==> is_subspace (lin x) V"
proof
  assume "is_vectorspace V" "x:V"
  show "<0> : lin x"
  proof (unfold lin_def, intro CollectI exI conjI)
    show "<0> = 0r <*> x" by (simp!)
  qed simp

  show "lin x <= V"
  proof (unfold lin_def, intro subsetI, elim CollectE exE conjE)
    fix xa a assume "xa = a <*> x"
    show "xa:V" by (simp!)
  qed

  show "ALL x1 : lin x. ALL x2 : lin x. x1 + x2 : lin x"
  proof (intro ballI)
    fix x1 x2 assume "x1 : lin x" "x2 : lin x"
    thus "x1 + x2 : lin x"
    proof (unfold lin_def, elim CollectE exE conjE,
      intro CollectI exI conjI)
      fix a1 a2 assume "x1 = a1 <*> x" "x2 = a2 <*> x"
      show "x1 + x2 = (a1 + a2) <*> x"
        by (simp! add: vs_add_mult_distrib2)
    qed simp
  qed
```

```
  show "ALL xa:lin x. ALL a. a <*> xa : lin x"
  proof (intro ballI allI)
    fix x1 a assume "x1 : lin x"
    thus "a <*> x1 : lin x"
    proof (unfold lin_def, elim CollectE exE conjE,
      intro CollectI exI conjI)
      fix a1 assume "x1 = a1 <*> x"
      show "a <*> x1 = (a * a1) <*> x" by (simp!)
    qed simp
  qed
qed
```

Any linear closure is a vector space.

```
lemma lin_vs [intro!!]:
  "[| is_vectorspace V; x:V |] ==> is_vectorspace (lin x)"
proof (rule subspace_vs)
  assume "is_vectorspace V" "x:V"
  show "is_subspace (lin x) V" ..
qed
```

5.3 Sum of two vectorspaces

The *sum* of two vectorspaces U and V is the set of all sums of elements from U and V.

```
instance set :: (plus) plus by intro_classes

defs vs_sum_def:
  "U + V == {u + v | u v. u:U & v:V}"

lemma vs_sumD:
  "x: U + V = (EX u:U. EX v:V. x = u + v)"
    by (unfold vs_sum_def) fast

lemmas vs_sumE = vs_sumD [RS iffD1, elimify]

lemma vs_sumI [intro!!]:
  "[| x:U; y:V; t= x + y |] ==> t : U + V"
  by (unfold vs_sum_def) fast
```

U is a subspace of $U + V$.

```
lemma subspace_vs_sum1 [intro!!]:
  "[| is_vectorspace U; is_vectorspace V |]
  ==> is_subspace U (U + V)"
proof
  assume "is_vectorspace U" "is_vectorspace V"
  show "<0> : U" ..
  show "U <= U + V"
  proof (intro subsetI vs_sumI)
```

```
  fix x assume "x:U"
    show "x = x + <0>" by (simp!)
    show "<0> : V" by (simp!)
  qed
  show "ALL x:U. ALL y:U. x + y : U"
  proof (intro ballI)
    fix x y assume "x:U" "y:U" show "x + y : U" by (simp!)
  qed
  show "ALL x:U. ALL a. a <*> x : U"
  proof (intro ballI allI)
    fix x a assume "x:U" show "a <*> x : U" by (simp!)
  qed
qed
```

The sum of two subspaces is again a subspace.

```
lemma vs_sum_subspace [intro!!]:
  "[| is_subspace U E; is_subspace V E; is_vectorspace E |]
  ==> is_subspace (U + V) E"
proof
  assume "is_subspace U E" "is_subspace V E" "is_vectorspace E"
  show "<0> : U + V"
  proof (intro vs_sumI)
    show "<0> : U" ..
    show "<0> : V" ..
    show "(<0>::'a) = <0> + <0>" by (simp!)
  qed

  show "U + V <= E"
  proof (intro subsetI, elim vs_sumE bexE)
    fix x u v assume "u : U" "v : V" "x = u + v"
    show "x:E" by (simp!)
  qed

  show "ALL x: U + V. ALL y: U + V. x + y : U + V"
  proof (intro ballI)
    fix x y assume "x : U + V" "y : U + V"
    thus "x + y : U + V"
    proof (elim vs_sumE bexE, intro vs_sumI)
      fix ux vx uy vy
      assume "ux : U" "vx : V" "x = ux + vx"
        and "uy : U" "vy : V" "y = uy + vy"
      show "x + y = (ux + uy) + (vx + vy)" by (simp!)
    qed (simp!)+
  qed

  show "ALL x : U + V. ALL a. a <*> x : U + V"
  proof (intro ballI allI)
    fix x a assume "x : U + V"
    thus "a <*> x : U + V"
```

```
    proof (elim vs_sumE bexE, intro vs_sumI)
      fix a x u v assume "u : U" "v : V" "x = u + v"
      show "a <*> x = (a <*> u) + (a <*> v)"
        by (simp! add: vs_add_mult_distrib1)
    qed (simp!)+
  qed
qed
```

The sum of two subspaces is a vectorspace.

```
lemma vs_sum_vs [intro!!]:
  "[| is_subspace U E; is_subspace V E; is_vectorspace E |]
  ==> is_vectorspace (U + V)"
proof (rule subspace_vs)
  assume "is_subspace U E" "is_subspace V E" "is_vectorspace E"
  show "is_subspace (U + V) E" ..
qed
```

5.4 Direct sums

The sum of U and V is called *direct*, iff the zero element is the only common element of U and V. For every element x of the direct sum of U and V the decomposition in $x = u + v$ with $u \in U$ and $v \in V$ is unique.

```
lemma decomp:
  "[| is_vectorspace E; is_subspace U E; is_subspace V E;
  U Int V = {<0>}; u1:U; u2:U; v1:V; v2:V; u1 + v1 = u2 + v2 |]
  ==> u1 = u2 & v1 = v2"
proof
  assume "is_vectorspace E" "is_subspace U E" "is_subspace V E"
    "U Int V = {<0>}" "u1:U" "u2:U" "v1:V" "v2:V"
    "u1 + v1 = u2 + v2"
  have eq: "u1 - u2 = v2 - v1" by (simp! add: vs_add_diff_swap)
  have u: "u1 - u2 : U" by (simp!)
  with eq have v': "v2 - v1 : U" by simp
  have v: "v2 - v1 : V" by (simp!)
  with eq have u': "u1 - u2 : V" by simp

  show "u1 = u2"
  proof (rule vs_add_minus_eq)
    show "u1 - u2 = <0>" by (rule Int_singletonD [OF _ u u'])
    show "u1 : E" ..
    show "u2 : E" ..
  qed

  show "v1 = v2"
  proof (rule vs_add_minus_eq [RS sym])
    show "v2 - v1 = <0>" by (rule Int_singletonD [OF _ v' v])
    show "v1 : E" ..
    show "v2 : E" ..
```

```
  qed
qed
```

An application of the previous lemma will be used in the proof of the Hahn-Banach Theorem (see page 55): for any element $y + a \odot x_0$ of the direct sum of a vectorspace H and the linear closure of x_0 the components $y \in H$ and a are uniquely determined.

```
lemma decomp_H0:
  "[| is_vectorspace E; is_subspace H E; y1 : H; y2 : H;
  x0 ~: H; x0 : E; x0 ~= <0>; y1 + a1 <*> x0 = y2 + a2 <*> x0 |]
  ==> y1 = y2 & a1 = a2"
proof
  assume "is_vectorspace E" and h: "is_subspace H E"
     and "y1 : H" "y2 : H" "x0 ~: H" "x0 : E" "x0 ~= <0>"
         "y1 + a1 <*> x0 = y2 + a2 <*> x0"

  have c: "y1 = y2 & a1 <*> x0 = a2 <*> x0"
  proof (rule decomp)
    show "a1 <*> x0 : lin x0" ..
    show "a2 <*> x0 : lin x0" ..
    show "H Int (lin x0) = {<0>}"
    proof
      show "H Int lin x0 <= {<0>}"
      proof (intro subsetI, elim IntE, rule singleton_iff[RS iffD2])
        fix x assume "x:H" "x : lin x0"
        thus "x = <0>"
        proof (unfold lin_def, elim CollectE exE conjE)
          fix a assume "x = a <*> x0"
          show ?thesis
          proof (rule case_split)
            assume "a = 0r" show ?thesis by (simp!)
          next
            assume "a ~= 0r"
            from h have "rinv a <*> a <*> x0 : H"
              by (rule subspace_mult_closed) (simp!)
            also have "rinv a <*> a <*> x0 = x0" by (simp!)
            finally have "x0 : H" .
            thus ?thesis by contradiction
          qed
       qed
      qed
      show "{<0>} <= H Int lin x0"
      proof (intro subsetI, elim singletonE, intro IntI,
        (simp only:)+)
        show "<0>:H" ..
        from lin_vs show "<0> : lin x0" ..
      qed
    qed
    show "is_subspace (lin x0) E" ..
```

```
  qed

  from c show "y1 = y2" by simp

  show  "a1 = a2"
  proof (rule vs_mult_right_cancel [RS iffD1])
    from c show "a1 <*> x0 = a2 <*> x0" by simp
  qed
qed
```

Since for any element $y + a \odot x_0$ of the direct sum of a vectorspace H and the linear closure of x_0 the components $y \in H$ and a are unique, it follows from $y \in H$ that $a = 0$.

```
lemma decomp_H0_H:
  "[| is_vectorspace E; is_subspace H E; t:H; x0 ~: H; x0:E;
  x0 ~= <0> |]
  ==> (SOME (y, a). t = y + a <*> x0 & y : H) = (t, 0r)"
proof (rule, unfold split_paired_all)
  assume "is_vectorspace E" "is_subspace H E" "t:H" "x0 ~: H" "x0:E"
    "x0 ~= <0>"
  have h: "is_vectorspace H" ..
  fix y a presume t1: "t = y + a <*> x0" and "y:H"
  have "y = t & a = 0r"
    by (rule decomp_H0) (assumption | (simp!))+
  thus "(y, a) = (t, 0r)" by (simp!)
qed (simp!)+
```

The components $y \in H$ and a in $y + a \odot x_0$ are unique, so the function h_0 defined by $h_0(y + a \odot x_0) = hy + a \cdot \xi$ is definite.

```
lemma h0_definite:
  "[| h0 == (λx. let (y, a) = SOME (y, a). (x = y + a <*> x0 & y:H)
                in (h y) + a * xi);
  x = y + a <*> x0; is_vectorspace E; is_subspace H E;
  y:H; x0 ~: H; x0:E; x0 ~= <0> |]
  ==> h0 x = h y + a * xi"
proof -
  assume
    "h0 == (λx. let (y, a) = SOME (y, a). (x = y + a <*> x0 & y:H)
              in (h y) + a * xi)"
    "x = y + a <*> x0" "is_vectorspace E" "is_subspace H E"
    "y:H" "x0 ~: H" "x0:E" "x0 ~= <0>"
  have "x : H + (lin x0)"
    by (simp! add: vs_sum_def lin_def) force+
  have "EX! xa. ((λ(y, a). x = y + a <*> x0 & y:H) xa)"
  proof
    show "EX xa. ((λ(y, a). x = y + a <*> x0 & y:H) xa)"
      by (force!)
  next
    fix xa ya
```

```
    assume "(λ(y,a). x = y + a <*> x0 & y : H) xa"
           "(λ(y,a). x = y + a <*> x0 & y : H) ya"
    show "xa = ya"
    proof -
      show "fst xa = fst ya & snd xa = snd ya ==> xa = ya"
        by (rule Pair_fst_snd_eq [RS iffD2])
      have x: "x = fst xa + snd xa <*> x0 & fst xa : H"
        by (force!)
      have y: "x = fst ya + snd ya <*> x0 & fst ya : H"
        by (force!)
      from x y show "fst xa = fst ya & snd xa = snd ya"
        by (elim conjE) (rule decomp_H0, (simp!)+)
    qed
  qed
  hence eq: "(SOME (y, a). x = y + a <*> x0 & y:H) = (y, a)"
    by (rule select1_equality) (force!)
  thus "h0 x = h y + a * xi" by (simp! add: Let_def)
qed

end
```

6 Normed vector spaces

```
theory NormedSpace = Subspace:
```

6.1 Quasinorms

A *seminorm* $\|\cdot\|$ is a function on a real vector space into the reals that has the following properties: It is positive definite, absolute homogenous and subadditive.

```
constdefs
  is_seminorm :: "['a::{plus, minus} set, 'a => real] => bool"
  "is_seminorm V norm == ALL x: V. ALL y:V. ALL a.
        0r <= norm x
      & norm (a <*> x) = (rabs a) * (norm x)
      & norm (x + y) <= norm x + norm y"

lemma is_seminormI [intro]:
  "[| !! x y a. [| x:V; y:V|] ==> 0r <= norm x;
  !! x a. x:V ==> norm (a <*> x) = (rabs a) * (norm x);
  !! x y. [|x:V; y:V |] ==> norm (x + y) <= norm x + norm y |]
  ==> is_seminorm V norm"
  by (unfold is_seminorm_def, force)

lemma seminorm_ge_zero [intro!!]:
  "[| is_seminorm V norm; x:V |] ==> 0r <= norm x"
  by (unfold is_seminorm_def, force)
```

```
lemma seminorm_rabs_homogenous:
  "[| is_seminorm V norm; x:V |]
  ==> norm (a <*> x) = (rabs a) * (norm x)"
  by (unfold is_seminorm_def, force)

lemma seminorm_subadditive:
  "[| is_seminorm V norm; x:V; y:V |]
  ==> norm (x + y) <= norm x + norm y"
  by (unfold is_seminorm_def, force)

lemma seminorm_diff_subadditive:
  "[| is_seminorm V norm; x:V; y:V; is_vectorspace V |]
  ==> norm (x - y) <= norm x + norm y"
proof -
  assume "is_seminorm V norm" "x:V" "y:V" "is_vectorspace V"
  have "norm (x - y) = norm (x + - 1r <*> y)"
    by (simp! add: diff_eq2 negate_eq2)
  also have "... <= norm x + norm  (- 1r <*> y)"
    by (simp! add: seminorm_subadditive)
  also have "norm (- 1r <*> y) = rabs (- 1r) * norm y"
    by (rule seminorm_rabs_homogenous)
  also have "rabs (- 1r) = 1r" by (rule rabs_minus_one)
  finally show "norm (x - y) <= norm x + norm y" by simp
qed

lemma seminorm_minus:
  "[| is_seminorm V norm; x:V; is_vectorspace V |]
  ==> norm (- x) = norm x"
proof -
  assume "is_seminorm V norm" "x:V" "is_vectorspace V"
  have "norm (- x) = norm (- 1r <*> x)" by (simp! only: negate_eq1)
  also have "... = rabs (- 1r) * norm x"
    by (rule seminorm_rabs_homogenous)
  also have "rabs (- 1r) = 1r" by (rule rabs_minus_one)
  finally show "norm (- x) = norm x" by simp
qed
```

6.2 Norms

A *norm* $\|\cdot\|$ is a seminorm that maps only the **0** vector to 0.

```
constdefs
  is_norm :: "['a::{minus, plus} set, 'a => real] => bool"
  "is_norm V norm == ALL x: V.  is_seminorm V norm
      & (norm x = 0r) = (x = <0>)"

lemma is_normI [intro]:
  "ALL x: V.  is_seminorm V norm  & (norm x = 0r) = (x = <0>)
```

```
  ==> is_norm V norm" by (simp only: is_norm_def)

lemma norm_is_seminorm [intro!!]:
  "[| is_norm V norm; x:V |] ==> is_seminorm V norm"
  by (unfold is_norm_def, force)

lemma norm_zero_iff:
  "[| is_norm V norm; x:V |] ==> (norm x = 0r) = (x = <0>)"
  by (unfold is_norm_def, force)

lemma norm_ge_zero [intro!!]:
  "[|is_norm V norm; x:V |] ==> 0r <= norm x"
  by (unfold is_norm_def is_seminorm_def, force)
```

6.3 Normed vector spaces

A vector space together with a norm is called a *normed space*.

```
constdefs
  is_normed_vectorspace ::
  "['a::{plus, minus} set, 'a => real] => bool"
  "is_normed_vectorspace V norm ==
      is_vectorspace V &
      is_norm V norm"

lemma normed_vsI [intro]:
  "[| is_vectorspace V; is_norm V norm |]
  ==> is_normed_vectorspace V norm"
  by (unfold is_normed_vectorspace_def) blast

lemma normed_vs_vs [intro!!]:
  "is_normed_vectorspace V norm ==> is_vectorspace V"
  by (unfold is_normed_vectorspace_def) force

lemma normed_vs_norm [intro!!]:
  "is_normed_vectorspace V norm ==> is_norm V norm"
  by (unfold is_normed_vectorspace_def, elim conjE)

lemma normed_vs_norm_ge_zero [intro!!]:
  "[| is_normed_vectorspace V norm; x:V |] ==> 0r <= norm x"
  by (unfold is_normed_vectorspace_def, rule, elim conjE)

lemma normed_vs_norm_gt_zero [intro!!]:
  "[| is_normed_vectorspace V norm; x:V; x ~= <0> |] ==> 0r < norm x"
proof (unfold is_normed_vectorspace_def, elim conjE)
  assume "x : V" "x ~= <0>" "is_vectorspace V" "is_norm V norm"
  have "0r <= norm x" ..
  also have "0r ~= norm x"
  proof
```

```
    presume "norm x = 0r"
    also have "?this = (x = <0>)" by (rule norm_zero_iff)
    finally have "x = <0>" .
    thus "False" by contradiction
  qed (rule sym)
  finally show "0r < norm x" .
qed

lemma normed_vs_norm_rabs_homogenous [intro!!]:
  "[| is_normed_vectorspace V norm; x:V |]
  ==> norm (a <*> x) = (rabs a) * (norm x)"
  by (rule seminorm_rabs_homogenous, rule norm_is_seminorm,
      rule normed_vs_norm)

lemma normed_vs_norm_subadditive [intro!!]:
  "[| is_normed_vectorspace V norm; x:V; y:V |]
  ==> norm (x + y) <= norm x + norm y"
  by (rule seminorm_subadditive, rule norm_is_seminorm,
     rule normed_vs_norm)
```

Any subspace of a normed vector space is again a normed vectorspace.

```
lemma subspace_normed_vs [intro!!]:
  "[| is_subspace F E; is_vectorspace E;
  is_normed_vectorspace E norm |] ==> is_normed_vectorspace F norm"
proof (rule normed_vsI)
  assume "is_subspace F E" "is_vectorspace E"
         "is_normed_vectorspace E norm"
  show "is_vectorspace F" ..
  show "is_norm F norm"
  proof (intro is_normI ballI conjI)
    show "is_seminorm F norm"
    proof
      fix x y a presume "x : E"
      show "0r <= norm x" ..
      show "norm (a <*> x) = rabs a * norm x" ..
      presume "y : E"
      show "norm (x + y) <= norm x + norm y" ..
    qed (simp!)+

    fix x assume "x : F"
    show "(norm x = 0r) = (x = <0>)"
    proof (rule norm_zero_iff)
      show "is_norm E norm" ..
    qed (simp!)
  qed
qed

end
```

7 Linearforms

theory *Linearform = VectorSpace:*

A *linear form* is a function on a vector space into the reals that is additive and multiplicative.

constdefs
```
  is_linearform :: "['a::{minus, plus} set, 'a => real] => bool"
  "is_linearform V f ==
      (ALL x: V. ALL y: V. f (x + y) = f x + f y) &
      (ALL x: V. ALL a. f (a <*> x) = a * (f x))"
```

lemma *is_linearformI [intro]:*
```
  "[| !! x y. [| x : V; y : V |] ==> f (x + y) = f x + f y;
    !! x c. x : V ==> f (c <*> x) = c * f x |]
 ==> is_linearform V f"
 by (unfold is_linearform_def) force
```

lemma *linearform_add [intro!!]:*
```
  "[| is_linearform V f; x:V; y:V |] ==> f (x + y) = f x + f y"
  by (unfold is_linearform_def) fast
```

lemma *linearform_mult [intro!!]:*
```
  "[| is_linearform V f; x:V |] ==>  f (a <*> x) = a * (f x)"
  by (unfold is_linearform_def) fast
```

lemma *linearform_neg [intro!!]:*
```
  "[|  is_vectorspace V; is_linearform V f; x:V|]
  ==> f (- x) = - f x"
proof -
  assume "is_linearform V f" "is_vectorspace V" "x:V"
  have "f (- x) = f ((- 1r) <*> x)" by (simp! add: negate_eq1)
  also have "... = (- 1r) * (f x)" by (rule linearform_mult)
  also have "... = - (f x)" by (simp!)
  finally show ?thesis .
qed
```

lemma *linearform_diff [intro!!]:*
```
  "[| is_vectorspace V; is_linearform V f; x:V; y:V |]
  ==> f (x - y) = f x - f y"
proof -
  assume "is_vectorspace V" "is_linearform V f" "x:V" "y:V"
  have "f (x - y) = f (x + - y)" by (simp! only: diff_eq1)
  also have "... = f x + f (- y)"
    by (rule linearform_add) (simp!)+
  also have "f (- y) = - f y" by (rule linearform_neg)
  finally show "f (x - y) = f x - f y" by (simp!)
qed
```

Every linear form yields 0 for the **0** vector.

```
lemma linearform_zero [intro!!, simp]:
  "[| is_vectorspace V; is_linearform V f |] ==> f <0> = 0r"
proof -
  assume "is_vectorspace V" "is_linearform V f"
  have "f <0> = f (<0> - <0>)" by (simp!)
  also have "... = f <0> - f <0>"
    by (rule linearform_diff) (simp!)+
  also have "... = 0r" by simp
  finally show "f <0> = 0r" .
qed

end
```

8 An order on functions

```
theory FunctionOrder = Subspace + Linearform:
```

8.1 The graph of a function

We define the *graph* of a (real) function f with domain F as the set

$$\{(x, f\,x).\, x : F\}$$

So we are modeling partial functions by specifying the domain and the mapping function. We use the term "function" also for its graph.

```
types 'a graph = "('a * real) set"

constdefs
  graph :: "['a set, 'a => real] => 'a graph "
  "graph F f == {(x, f x) | x. x:F}"

lemma graphI [intro!!]: "x:F ==> (x, f x) : graph F f"
  by (unfold graph_def, intro CollectI exI) force

lemma graphI2 [intro!!]: "x:F ==> EX t: (graph F f). t = (x, f x)"
  by (unfold graph_def, force)

lemma graphD1 [intro!!]: "(x, y): graph F f ==> x:F"
  by (unfold graph_def, elim CollectE exE) force

lemma graphD2 [intro!!]: "(x, y): graph H h ==> y = h x"
  by (unfold graph_def, elim CollectE exE) force
```

8.2 Functions ordered by domain extension

A function h' is an extension of h, iff the graph of h is a subset of the graph of h'.

```
lemma graph_extI:
  "[| !! x. x: H ==> h x = h' x; H <= H'|]
  ==> graph H h <= graph H' h'"
  by (unfold graph_def, force)

lemma graph_extD1 [intro!!]:
  "[| graph H h <= graph H' h'; x:H |] ==> h x = h' x"
  by (unfold graph_def, force)

lemma graph_extD2 [intro!!]:
  "[| graph H h <= graph H' h' |] ==> H <= H'"
  by (unfold graph_def, force)
```

8.3 Domain and function of a graph

The inverse functions to *graph* are *domain* and *funct*.

```
constdefs
  domain :: "'a graph => 'a set"
  "domain g == {x. EX y. (x, y):g}"

  funct :: "'a graph => ('a => real)"
  "funct g == λx. (SOME y. (x, y):g)"
```

The following lemma states that g is the graph of a function if the relation induced by g is unique.

```
lemma graph_domain_funct:
  "(!!x y z. (x, y):g ==> (x, z):g ==> z = y)
  ==> graph (domain g) (funct g) = g"
proof (unfold domain_def funct_def graph_def, auto)
  fix a b assume "(a, b) : g"
  show "(a, SOME y. (a, y) : g) : g" by (rule selectI2)
  show "EX y. (a, y) : g" ..
  assume uniq: "!!x y z. (x, y):g ==> (x, z):g ==> z = y"
  show "b = (SOME y. (a, y) : g)"
  proof (rule select_equality [RS sym])
    fix y assume "(a, y):g" show "y = b" by (rule uniq)
  qed
qed
```

8.4 Norm-preserving extensions of a function

Given a linear form f on the space F and a seminorm p on E. The set of all linear extensions of f, to superspaces H of F, which are bounded by p,

is defined as follows.

```
constdefs
  norm_pres_extensions ::
    "['a::{minus, plus} set, 'a  => real, 'a set, 'a => real]
    => 'a graph set"
    "norm_pres_extensions E p F f
    == {g. EX H h. graph H h = g
                & is_linearform H h
                & is_subspace H E
                & is_subspace F H
                & graph F f <= graph H h
                & (ALL x:H. h x <= p x)}"

lemma norm_pres_extension_D:
  "g : norm_pres_extensions E p F f
  ==> EX H h. graph H h = g
            & is_linearform H h
            & is_subspace H E
            & is_subspace F H
            & graph F f <= graph H h
            & (ALL x:H. h x <= p x)"
  by (unfold norm_pres_extensions_def) force

lemma norm_pres_extensionI2 [intro]:
  "[| is_linearform H h; is_subspace H E; is_subspace F H;
  graph F f <= graph H h; ALL x:H. h x <= p x |]
  ==> (graph H h : norm_pres_extensions E p F f)"
 by (unfold norm_pres_extensions_def) force

lemma norm_pres_extensionI [intro]:
  "EX H h. graph H h = g
         & is_linearform H h
         & is_subspace H E
         & is_subspace F H
         & graph F f <= graph H h
         & (ALL x:H. h x <= p x)
  ==> g: norm_pres_extensions E p F f"
  by (unfold norm_pres_extensions_def) force

end
```

9 The norm of a function

theory *FunctionNorm = NormedSpace + FunctionOrder:*

9.1 Continuous linear forms

A linear form f on a normed vector space $(V, \|\cdot\|)$ is *continuous*, iff it is bounded, i. e.

$$\exists c \in R.\ \forall x \in V.\ |f\,x| \leq c \cdot \|x\|$$

In our application no other functions than linear forms are considered, so we can define continuous linear forms as bounded linear forms:

```
constdefs
  is_continuous ::
  "['a::{minus, plus} set, 'a => real, 'a => real] => bool"
  "is_continuous V norm f ==
    is_linearform V f & (EX c. ALL x:V. rabs (f x) <= c * norm x)"

lemma continuousI [intro]:
  "[| is_linearform V f; !! x. x:V ==> rabs (f x) <= c * norm x |]
  ==> is_continuous V norm f"
proof (unfold is_continuous_def, intro exI conjI ballI)
  assume r: "!! x. x:V ==> rabs (f x) <= c * norm x"
  fix x assume "x:V" show "rabs (f x) <= c * norm x" by (rule r)
qed

lemma continuous_linearform [intro!!]:
  "is_continuous V norm f ==> is_linearform V f"
  by (unfold is_continuous_def) force

lemma continuous_bounded [intro!!]:
  "is_continuous V norm f
  ==> EX c. ALL x:V. rabs (f x) <= c * norm x"
  by (unfold is_continuous_def) force
```

9.2 The norm of a linear form

The least real number c for which holds

$$\forall x \in V.\ |f\,x| \leq c \cdot \|x\|$$

is called the *norm* of f.

For non-trivial vector spaces $V \neq \{\mathbf{0}\}$ the norm can be defined as

$$\|f\| = \sup_{x \neq \mathbf{0}} \frac{|f\,x|}{\|x\|}$$

For the case $V = \{\mathbf{0}\}$ the supremum would be taken from an empty set. Since $\mathbb{R}$ is unbounded, there would be no supremum. To avoid this situation it must be guaranteed that there is an element in this set. This element must be ≥ 0 so that *function-norm* has the norm properties. Furthermore it does

not have to change the norm in all other cases, so it must be 0, as all other elements of are ≥ 0.
Thus we define the set B the supremum is taken from as

$$\{0\} \cup \left\{ \frac{|f\,x|}{\|x\|}.x \neq \mathbf{0} \ \wedge \ x \in F \right\}$$

constdefs

```
  B :: "[ 'a set, 'a => real, 'a => real ] => real set"
  "B V norm f ==
  {0r} Un {rabs (f x) * rinv (norm x) | x. x ~= <0> & x:V}"
```

n is the function norm of f, iff n is the supremum of B.

constdefs

```
  is_function_norm ::
  " ['a set, 'a => real, 'a => real] => real => bool"
  "is_function_norm V norm f fn == is_Sup UNIV (B V norm f) fn"
```

function-norm is equal to the supremum of B, if the supremum exists. Otherwise it is undefined.

constdefs

```
  function_norm :: " ['a set, 'a => real, 'a => real] => real"
  "function_norm V norm f == Sup UNIV (B V norm f)"
```

lemma `B_not_empty: "0r : B V norm f"`
by `(unfold B_def, force)`

The following lemma states that every continuous linear form on a normed space $(V, \|\cdot\|)$ has a function norm.

lemma `ex_fnorm [intro!!]:`

```
  "[| is_normed_vectorspace V norm; is_continuous V norm f|]
     ==> is_function_norm V norm f (function_norm V norm f)"
```

proof `(unfold function_norm_def is_function_norm_def`

```
  is_continuous_def Sup_def, elim conjE, rule selectI2EX)
  assume "is_normed_vectorspace V norm"
  assume "is_linearform V f"
  and e: "EX c. ALL x:V. rabs (f x) <= c * norm x"
```

The existence of the supremum is shown using the completeness of the reals. Completeness means, that every non-empty bounded set of reals has a supremum.

```
  show  "EX a. is_Sup UNIV (B V norm f) a"
  proof (unfold is_Sup_def, rule reals_complete)
```

First we have to show that B is non-empty:

```
    show "EX X. X : B V norm f"
    proof (intro exI)
      show "0r : (B V norm f)" by (unfold B_def, force)
    qed
```

Then we have to show that B is bounded:

```
from e show "EX Y. isUb UNIV (B V norm f) Y"
proof
```

We know that f is bounded by some value c.

```
  fix c assume a: "ALL x:V. rabs (f x) <= c * norm x"
  def b == "max c 0r"

  show "?thesis"
  proof (intro exI isUbI setleI ballI, unfold B_def,
    elim UnE CollectE exE conjE singletonE)
```

To proof the thesis, we have to show that there is some constant b, such that $y \leq b$ for all $y \in B$. Due to the definition of B there are two cases for $y \in B$. If $y = 0$ then $y \leq idtmax\, c0$:

```
    fix y assume "y = 0r"
    show "y <= b" by (simp! add: le_max2)
```

The second case is $y = |f\, x|/\|x\|$ for some $x \in V$ with $x \neq \mathbf{0}$.

```
  next
    fix x y
    assume "x:V" "x ~= <0>"
```

The thesis follows by a short calculation using the fact that f is bounded.

```
    assume "y = rabs (f x) * rinv (norm x)"
    also have "... <= c * norm x * rinv (norm x)"
    proof (rule real_mult_le_le_mono2)
      show "0r <= rinv (norm x)"
        by (rule real_less_imp_le, rule real_rinv_gt_zero,
            rule normed_vs_norm_gt_zero)
      from a show "rabs (f x) <= c * norm x" ..
    qed
    also have "... = c * (norm x * rinv (norm x))"
      by (rule real_mult_assoc)
    also have "(norm x * rinv (norm x)) = 1r"
    proof (rule real_mult_inv_right)
      show nz: "norm x ~= 0r"
        by (rule not_sym, rule lt_imp_not_eq,
          rule normed_vs_norm_gt_zero)
    qed
    also have "c * ... <= b " by (simp! add: le_max1)
    finally show "y <= b" .
  qed simp
 qed
qed
qed
```

The norm of a continuous function is always ≥ 0.

```
lemma fnorm_ge_zero [intro!!]:
  "[| is_continuous V norm f; is_normed_vectorspace V norm|]
   ==> 0r <= function_norm V norm f"
proof -
  assume c: "is_continuous V norm f"
     and n: "is_normed_vectorspace V norm"
```

The function norm is defined as the supremum of B. So it is ≥ 0 if all elements in B are ≥ 0, provided the supremum exists and B is not empty.

```
  show ?thesis
  proof (unfold function_norm_def, rule sup_ub1)
    show "ALL x:(B V norm f). 0r <= x"
    proof (intro ballI, unfold B_def,
           elim UnE singletonE CollectE exE conjE)
      fix x r
      assume "x : V" "x ~= <0>"
        and r: "r = rabs (f x) * rinv (norm x)"

      have ge: "0r <= rabs (f x)" by (simp! only: rabs_ge_zero)
      have "0r <= rinv (norm x)"
        by (rule real_less_imp_le, rule real_rinv_gt_zero, rule)
      with ge show "0r <= r"
       by (simp only: r, rule real_le_mult_order)
    qed (simp!)
```

Since f is continuous the function norm exists:

```
    have "is_function_norm V norm f (function_norm V norm f)" ..
    thus "is_Sup UNIV (B V norm f) (Sup UNIV (B V norm f))"
      by (unfold is_function_norm_def function_norm_def)
```

B is non-empty by construction:

```
    show "0r : B V norm f" by (rule B_not_empty)
  qed
qed
```

The fundamental property of function norms is:

$$|f\,x| \leq \|f\| \cdot \|x\|$$

```
lemma norm_fx_le_norm_f_norm_x:
  "[| is_normed_vectorspace V norm; x:V; is_continuous V norm f |]
    ==> rabs (f x) <= function_norm V norm f * norm x"
proof -
  assume "is_normed_vectorspace V norm" "x:V"
  and c: "is_continuous V norm f"
  have v: "is_vectorspace V" ..
  assume "x:V"
```

The proof is by case analysis on x.

```
  show ?thesis
  proof (rule case_split)
```

For the case $x = \mathbf{0}$ the thesis follows from the linearity of f: for every linear function holds $f\,\mathbf{0} = 0$.

```
    assume "x = <0>"
    have "rabs (f x) = rabs (f <0>)" by (simp!)
    also from v continuous_linearform have "f <0> = 0r" ..
    also note rabs_zero
    also have "0r <= function_norm V norm f * norm x"
    proof (rule real_le_mult_order)
      show "0r <= function_norm V norm f" ..
      show "0r <= norm x" ..
    qed
    finally
    show "rabs (f x) <= function_norm V norm f * norm x" .

  next
    assume "x ~= <0>"
    have n: "0r <= norm x" ..
    have nz: "norm x ~= 0r"
    proof (rule lt_imp_not_eq [RS not_sym])
      show "0r < norm x" ..
    qed
```

For the case $x \neq \mathbf{0}$ we derive the following fact from the definition of the function norm:

```
    have l: "rabs (f x) * rinv (norm x) <= function_norm V norm f"
    proof (unfold function_norm_def, rule sup_ub)
      from ex_fnorm [OF _ c]
      show "is_Sup UNIV (B V norm f) (Sup UNIV (B V norm f))"
         by (simp! add: is_function_norm_def function_norm_def)
      show "rabs (f x) * rinv (norm x) : B V norm f"
        by (unfold B_def, intro UnI2 CollectI exI [of _ x]
            conjI, simp)
    qed
```

The thesis now follows by a short calculation:

```
    have "rabs (f x) = rabs (f x) * 1r" by (simp!)
    also from nz have "1r = rinv (norm x) * norm x"
      by (rule real_mult_inv_left [RS sym])
    also
    have "rabs (f x) * ... = rabs (f x) * rinv (norm x) * norm x"
      by (simp! add: real_mult_assoc [of "rabs (f x)"])
    also have "... <= function_norm V norm f * norm x"
      by (rule real_mult_le_le_mono2 [OF n l])
    finally
    show "rabs (f x) <= function_norm V norm f * norm x" .
  qed
qed
```

The function norm is the least positive real number for which the following inequation holds:

$$|f\,x| \leq c \cdot \|x\|$$

```
lemma fnorm_le_ub:
  "[| is_normed_vectorspace V norm; is_continuous V norm f;
     ALL x:V. rabs (f x) <= c * norm x; 0r <= c |]
  ==> function_norm V norm f <= c"
proof (unfold function_norm_def)
  assume "is_normed_vectorspace V norm"
  assume c: "is_continuous V norm f"
  assume fb: "ALL x:V. rabs (f x) <= c * norm x"
         and "0r <= c"
```

Suppose the inequation holds for some $c \geq 0$. If c is an upper bound of B, then c is greater than the function norm since the function norm is the least upper bound.

```
  show "Sup UNIV (B V norm f) <= c"
  proof (rule sup_le_ub)
    from ex_fnorm [OF _ c]
    show "is_Sup UNIV (B V norm f) (Sup UNIV (B V norm f))"
      by (simp! add: is_function_norm_def function_norm_def)
```

c is an upper bound of B, i. e. every $y \in B$ is less than c.

```
    show "isUb UNIV (B V norm f) c"
    proof (intro isUbI setleI ballI)
      fix y assume "y: B V norm f"
      thus le: "y <= c"
      proof (unfold B_def, elim UnE CollectE exE conjE singletonE)
```

The first case for $y \in B$ is $y = 0$.

```
        assume "y = 0r"
        show "y <= c" by (force!)
```

The second case is $y = |f\,x|/\|x\|$ for some $x \in V$ with $x \neq \mathbf{0}$.

```
      next
        fix x
        assume "x : V" "x ~= <0>"

        have lz: "0r < norm x"
          by (simp! add: normed_vs_norm_gt_zero)

        have nz: "norm x ~= 0r"
        proof (rule not_sym)
          from lz show "0r ~= norm x"
            by (simp! add: order_less_imp_not_eq)
        qed

        from lz have "0r < rinv (norm x)"
          by (simp! add: real_rinv_gt_zero)
```

```
      hence rinv_gez: "0r <= rinv (norm x)"
        by (rule real_less_imp_le)

      assume "y = rabs (f x) * rinv (norm x)"
      also from rinv_gez have "... <= c * norm x * rinv (norm x)"
        proof (rule real_mult_le_le_mono2)
          show "rabs (f x) <= c * norm x" by (rule bspec)
        qed
      also have "... <= c" by (simp add: nz real_mult_assoc)
      finally show ?thesis .
    qed
  qed force
 qed
qed

end
```

10 Zorn's Lemma

```
theory ZornLemma = Aux + Zorn:
```

Zorn's Lemmas states: if every linear ordered subset of an ordered set S has an upper bound in S, then there exists a maximal element in S. In our application, S is a set of sets ordered by set inclusion. Since the union of a chain of sets is an upper bound for all elements of the chain, the conditions of Zorn's lemma can be modified: if S is non-empty, it suffices to show that for every non-empty chain c in S the union of c also lies in S.

```
theorem Zorn's_Lemma:
  "a:S ==> (!!c. c: chain S ==> EX x. x:c ==> Union c : S)
  ==>  EX y: S. ALL z: S. y <= z --> y = z"
proof (rule Zorn_Lemma2)[2]  assume aS: "a:S"
  assume r: "!!c. c: chain S ==> EX x. x:c ==> Union c : S"
  show "ALL c:chain S. EX y:S. ALL z:c. z <= y"
  proof
    fix c assume "c:chain S"
    show "EX y:S. ALL z:c. z <= y"
    proof (rule case_split)
```

If c is an empty chain, then every element in S is an upper bound of c.

```
      assume "c={}"
      with aS show ?thesis by fast
```

If c is non-empty, then $\bigcup c$ is an upper bound of c, lying in S.

```
    next
      assume c: "c~={}"
```

[2] See http://isabelle.in.tum.de/library/HOL/HOL-Real/Zorn.html

```
      show ?thesis
      proof
        show "ALL z:c. z <= Union c" by fast
        show "Union c : S"
        proof (rule r)
          from c show "EX x. x:c" by fast
        qed
      qed
    qed
  qed
qed

end
```

Part II
Lemmas for the Proof

11 The supremum w.r.t. the function order

theory *HahnBanachSupLemmas = FunctionNorm + ZornLemma:*

This section contains some lemmas that will be used in the proof of the Hahn-Banach Theorem. In this section the following context is presumed. Let E be a real vector space with a seminorm q on E. F is a subspace of E and f a linear form on F. We consider a chain c of norm-preserving extensions of f, such that $\bigcup c = \mathit{graph}\, H\, h$. We will show some properties about the limit function h, i.e. the supremum of the chain c.

Let c be a chain of norm-preserving extensions of the function f and let $\mathit{graph}\, H\, h$ be the supremum of c. Every element in H is member of one of the elements of the chain.

```
lemma some_H'h't:
  "[| M = norm_pres_extensions E p F f; c: chain M;
  graph H h = Union c; x:H |]
   ==> EX H' h'. graph H' h' : c & (x, h x) : graph H' h'
       & is_linearform H' h' & is_subspace H' E
       & is_subspace F H' & graph F f <= graph H' h'
       & (ALL x:H'. h' x <= p x)"
proof -
  assume m: "M = norm_pres_extensions E p F f" and "c: chain M"
     and u: "graph H h = Union c" "x:H"

  have h: "(x, h x) : graph H h" ..
  with u have "(x, h x) : Union c" by simp
  hence ex1: "EX g:c. (x, h x) : g"
    by (simp only: Union_iff)
  thus ?thesis
  proof (elim bexE)
    fix g assume g: "g:c" "(x, h x) : g"
    have "c <= M" by (rule chainD2)
    hence "g : M" ..
    hence "g : norm_pres_extensions E p F f" by (simp only: m)
    hence "EX H' h'. graph H' h' = g
                  & is_linearform H' h'
                  & is_subspace H' E
                  & is_subspace F H'
                  & graph F f <= graph H' h'
                  & (ALL x:H'. h' x <= p x)"
```

```
      by (rule norm_pres_extension_D)
    thus ?thesis
    proof (elim exE conjE)
      fix H' h'
      assume "graph H' h' = g" "is_linearform H' h'"
        "is_subspace H' E" "is_subspace F H'"
        "graph F f <= graph H' h'" "ALL x:H'. h' x <= p x"
      show ?thesis
      proof (intro exI conjI)
        show "graph H' h' : c" by (simp!)
        show "(x, h x) : graph H' h'" by (simp!)
      qed
    qed
  qed
qed
```

Let c be a chain of norm-preserving extensions of the function f and let *graph H h* be the supremum of c. Every element in the domain H of the supremum function is member of the domain H' of some function h', such that h extends h'.

```
lemma some_H'h':
  "[| M = norm_pres_extensions E p F f; c: chain M;
  graph H h = Union c; x:H |]
  ==> EX H' h'. x:H' & graph H' h' <= graph H h
      & is_linearform H' h' & is_subspace H' E & is_subspace F H'
      & graph F f <= graph H' h' & (ALL x:H'. h' x <= p x)"
proof -
  assume "M = norm_pres_extensions E p F f" and cM: "c: chain M"
     and u: "graph H h = Union c" "x:H"

  have "EX H' h'. graph H' h' : c & (x, h x) : graph H' h'
       & is_linearform H' h' & is_subspace H' E
       & is_subspace F H' & graph F f <= graph H' h'
       & (ALL x:H'. h' x <= p x)"
    by (rule some_H'h't)
  thus ?thesis
  proof (elim exE conjE)
    fix H' h' assume "(x, h x) : graph H' h'" "graph H' h' : c"
      "is_linearform H' h'" "is_subspace H' E" "is_subspace F H'"
      "graph F f <= graph H' h'" "ALL x:H'. h' x <= p x"
    show ?thesis
    proof (intro exI conjI)
      show "x:H'" by (rule graphD1)
      from cM u show "graph H' h' <= graph H h"
        by (simp! only: chain_ball_Union_upper)
    qed
  qed
qed
```

Any two elements x and y in the domain H of the supremum function h are both in the domain H' of some function h', such that h extends h'.

```
lemma some_H'h'2:
  "[| M = norm_pres_extensions E p F f; c: chain M;
  graph H h = Union c;  x:H; y:H |]
  ==> EX H' h'. x:H' & y:H' & graph H' h' <= graph H h
      & is_linearform H' h' & is_subspace H' E & is_subspace F H'
      & graph F f <= graph H' h' & (ALL x:H'. h' x <= p x)"
proof -
  assume "M = norm_pres_extensions E p F f" "c: chain M"
         "graph H h = Union c" "x:H" "y:H"
```

x is in the domain H' of some function h', such that h extends h'.

```
  have e1: "EX H' h'. graph H' h' : c & (x, h x) : graph H' h'
       & is_linearform H' h' & is_subspace H' E
       & is_subspace F H' & graph F f <= graph H' h'
       & (ALL x:H'. h' x <= p x)"
    by (rule some_H'h't)
```

y is in the domain H'' of some function h'', such that h extends h''.

```
  have e2: "EX H'' h''. graph H'' h'' : c & (y, h y) : graph H'' h''
       & is_linearform H'' h'' & is_subspace H'' E
       & is_subspace F H'' & graph F f <= graph H'' h''
       & (ALL x:H''. h'' x <= p x)"
    by (rule some_H'h't)

  from e1 e2 show ?thesis
  proof (elim exE conjE)
    fix H' h' assume "(y, h y): graph H' h'" "graph H' h' : c"
      "is_linearform H' h'" "is_subspace H' E" "is_subspace F H'"
      "graph F f <= graph H' h'" "ALL x:H'. h' x <= p x"

    fix H'' h'' assume "(x, h x): graph H'' h''" "graph H'' h'' : c"
      "is_linearform H'' h''" "is_subspace H'' E" "is_subspace F H''"
      "graph F f <= graph H'' h''" "ALL x:H''. h'' x <= p x"
```

Since both h' and h'' are elements of the chain, h'' is an extension of h' or vice versa. Thus both x and y are contained in the greater one.

```
    have "graph H'' h'' <= graph H' h' | graph H' h' <= graph H'' h''"
      (is "?case1 | ?case2")
      by (rule chainD)
    thus ?thesis
    proof
      assume ?case1
      show ?thesis
      proof (intro exI conjI)
        have "(x, h x) : graph H'' h''" .
        also have "... <= graph H' h'" .
        finally have xh: "(x, h x): graph H' h'" .
```

```
      thus x: "x:H'" ..
      show y: "y:H'" ..
      show "graph H' h' <= graph H h"
        by (simp! only: chain_ball_Union_upper)
    qed
  next
    assume ?case2
    show ?thesis
    proof (intro exI conjI)
      show x: "x:H''" ..
      have "(y, h y) : graph H' h'" by (simp!)
      also have "... <= graph H'' h''" .
      finally have yh: "(y, h y): graph H'' h''" .
      thus y: "y:H''" ..
      show "graph H'' h'' <= graph H h"
        by (simp! only: chain_ball_Union_upper)
    qed
  qed
qed
```

The relation induced by the graph of the supremum of a chain c is definite, i. e. it is the graph of a function.

```
lemma sup_definite:
  "[| M == norm_pres_extensions E p F f; c : chain M;
  (x, y) : Union c; (x, z) : Union c |] ==> z = y"
proof -
  assume "c:chain M" "M == norm_pres_extensions E p F f"
    "(x, y) : Union c" "(x, z) : Union c"
  thus ?thesis
  proof (elim UnionE chainE2)
```

Since both $(x, y) \in \bigcup c$ and $(x, z) \in \bigcup c$ they are members of some graphs G_1 and G_2, resp., such that both G_1 and G_2 are members of c.

```
    fix G1 G2 assume
      "(x, y) : G1" "G1 : c" "(x, z) : G2" "G2 : c" "c <= M"

    have "G1 : M" ..
    hence e1: "EX H1 h1. graph H1 h1 = G1"
      by (force! dest: norm_pres_extension_D)
    have "G2 : M" ..
    hence e2: "EX H2 h2. graph H2 h2 = G2"
      by (force! dest: norm_pres_extension_D)
    from e1 e2 show ?thesis
    proof (elim exE)
      fix H1 h1 H2 h2
      assume "graph H1 h1 = G1" "graph H2 h2 = G2"
```

G_1 is contained in G_2 or vice versa, since both G_1 and G_2 are members of c.

```
      have "G1 <= G2 | G2 <= G1" (is "?case1 | ?case2") ..
      thus ?thesis
      proof
        assume ?case1
        have "(x, y) : graph H2 h2" by (force!)
        hence "y = h2 x" ..
        also have "(x, z) : graph H2 h2" by (simp!)
        hence "z = h2 x" ..
        finally show ?thesis .
      next
        assume ?case2
        have "(x, y) : graph H1 h1" by (simp!)
        hence "y = h1 x" ..
        also have "(x, z) : graph H1 h1" by (force!)
        hence "z = h1 x" ..
        finally show ?thesis .
      qed
    qed
  qed
qed
```

The limit function h is linear. Every element x in the domain of h is in the domain of a function h' in the chain of norm preserving extensions. Furthermore, h is an extension of h' so the function values of x are identical for h' and h. Finally, the function h' is linear by construction of M.

```
lemma sup_lf:
  "[| M = norm_pres_extensions E p F f; c: chain M;
  graph H h = Union c |] ==> is_linearform H h"
proof -
  assume "M = norm_pres_extensions E p F f" "c: chain M"
         "graph H h = Union c"

  show "is_linearform H h"
  proof
    fix x y assume "x : H" "y : H"
    have "EX H' h'. x:H' & y:H' & graph H' h' <= graph H h
            & is_linearform H' h' & is_subspace H' E
            & is_subspace F H' & graph F f <= graph H' h'
            & (ALL x:H'. h' x <= p x)"
      by (rule some_H'h'2)
```

We have to show that h is additive.

```
    thus "h (x + y) = h x + h y"
    proof (elim exE conjE)
      fix H' h' assume "x:H'" "y:H'"
        and b: "graph H' h' <= graph H h"
        and "is_linearform H' h'" "is_subspace H' E"
      have "h' (x + y) = h' x + h' y"
        by (rule linearform_add)
```

```
      also have "h' x = h x" ..
      also have "h' y = h y" ..
      also have "x + y : H'" ..
      with b have "h' (x + y) = h (x + y)" ..
      finally show ?thesis .
    qed
  next
    fix a x assume "x : H"
    have "EX H' h'. x:H' & graph H' h' <= graph H h
            & is_linearform H' h' & is_subspace H' E
            & is_subspace F H' & graph F f <= graph H' h'
            & (ALL x:H'. h' x <= p x)"
      by (rule some_H'h')
```

We have to show that h is multiplicative.

```
    thus "h (a <*> x) = a * h x"
    proof (elim exE conjE)
      fix H' h' assume "x:H'"
        and b: "graph H' h' <= graph H h"
        and "is_linearform H' h'" "is_subspace H' E"
      have "h' (a <*> x) = a * h' x"
        by (rule linearform_mult)
      also have "h' x = h x" ..
      also have "a <*> x : H'" ..
      with b have "h' (a <*> x) = h (a <*> x)" ..
      finally show ?thesis .
    qed
  qed
qed
```

The limit of a non-empty chain of norm preserving extensions of f is an extension of f, since every element of the chain is an extension of f and the supremum is an extension for every element of the chain.

```
lemma sup_ext:
  "[| M = norm_pres_extensions E p F f; c: chain M; EX x. x:c;
  graph H h = Union c |] ==> graph F f <= graph H h"
proof -
  assume "M = norm_pres_extensions E p F f" "c: chain M"
         "graph H h = Union c"
  assume "EX x. x:c"
  thus ?thesis
  proof
    fix x assume "x:c"
    have "c <= M" by (rule chainD2)
    hence "x:M" ..
    hence "x : norm_pres_extensions E p F f" by (simp!)

    hence "EX G g. graph G g = x
            & is_linearform G g
```

```
            & is_subspace G E
            & is_subspace F G
            & graph F f <= graph G g
            & (ALL x:G. g x <= p x)"
      by (simp! add: norm_pres_extension_D)

    thus ?thesis
    proof (elim exE conjE)
      fix G g assume "graph F f <= graph G g"
      also assume "graph G g = x"
      also have "... : c" .
      hence "x <= Union c" by fast
      also have [RS sym]: "graph H h = Union c" .
      finally show ?thesis .
    qed
  qed
qed
```

The domain H of the limit function is a superspace of F, since F is a subset of H. The existence of the **0** element in F and the closure properties follow from the fact that F is a vectorspace.

```
lemma sup_supF:
  "[| M = norm_pres_extensions E p F f; c: chain M; EX x. x:c;
  graph H h = Union c; is_subspace F E; is_vectorspace E |]
  ==> is_subspace F H"
proof -
  assume "M = norm_pres_extensions E p F f" "c: chain M" "EX x. x:c"
    "graph H h = Union c" "is_subspace F E" "is_vectorspace E"

  show ?thesis
  proof
    show "<0> : F" ..
    show "F <= H"
    proof (rule graph_extD2)
      show "graph F f <= graph H h"
        by (rule sup_ext)
    qed
    show "ALL x:F. ALL y:F. x + y : F"
    proof (intro ballI)
      fix x y assume "x:F" "y:F"
      show "x + y : F" by (simp!)
    qed
    show "ALL x:F. ALL a. a <*> x : F"
    proof (intro ballI allI)
      fix x a assume "x:F"
      show "a <*> x : F" by (simp!)
    qed
  qed
qed
```

The domain H of the limit function is a subspace of E.

```
lemma sup_subE:
  "[| M = norm_pres_extensions E p F f; c: chain M; EX x. x:c;
  graph H h = Union c; is_subspace F E; is_vectorspace E |]
  ==> is_subspace H E"
proof -
  assume "M = norm_pres_extensions E p F f" "c: chain M" "EX x. x:c"
    "graph H h = Union c" "is_subspace F E" "is_vectorspace E"
  show ?thesis
  proof
```

The **0** element is in H, as F is a subset of H:

```
    have "<0> : F" ..
    also have "is_subspace F H" by (rule sup_supF)
    hence "F <= H" ..
    finally show "<0> : H" .
```

H is a subset of E:

```
    show "H <= E"
    proof
      fix x assume "x:H"
      have "EX H' h'. x:H' & graph H' h' <= graph H h
              & is_linearform H' h' & is_subspace H' E
              & is_subspace F H' & graph F f <= graph H' h'
              & (ALL x:H'. h' x <= p x)"
        by (rule some_H'h')
      thus "x:E"
      proof (elim exE conjE)
        fix H' h' assume "x:H'" "is_subspace H' E"
        have "H' <= E" ..
        thus "x:E" ..
      qed
    qed
```

H is closed under addition:

```
    show "ALL x:H. ALL y:H. x + y : H"
    proof (intro ballI)
      fix x y assume "x:H" "y:H"
      have "EX H' h'. x:H' & y:H' & graph H' h' <= graph H h
              & is_linearform H' h' & is_subspace H' E
              & is_subspace F H' & graph F f <= graph H' h'
              & (ALL x:H'. h' x <= p x)"
        by (rule some_H'h'2)
      thus "x + y : H"
      proof (elim exE conjE)
        fix H' h'
        assume "x:H'" "y:H'" "is_subspace H' E"
          "graph H' h' <= graph H h"
        have "x + y : H'" ..
```

```
      also have "H' <= H" ..
      finally show ?thesis .
    qed
  qed
```

H is closed under scalar multiplication:

```
  show "ALL x:H. ALL a. a <*> x : H"
  proof (intro ballI allI)
    fix x a assume "x:H"
    have "EX H' h'. x:H' & graph H' h' <= graph H h
            & is_linearform H' h' & is_subspace H' E
            & is_subspace F H' & graph F f <= graph H' h'
            & (ALL x:H'. h' x <= p x)"
      by (rule some_H'h')
    thus "a <*> x : H"
    proof (elim exE conjE)
      fix H' h'
      assume "x:H'" "is_subspace H' E" "graph H' h' <= graph H h"
      have "a <*> x : H'" ..
      also have "H' <= H" ..
      finally show ?thesis .
    qed
  qed
qed
```

The limit function is bounded by the norm p as well, since all elements in the chain are bounded by p.

```
lemma sup_norm_pres:
  "[| M = norm_pres_extensions E p F f; c: chain M;
  graph H h = Union c |] ==> ALL x:H. h x <= p x"
proof
  assume "M = norm_pres_extensions E p F f" "c: chain M"
         "graph H h = Union c"
  fix x assume "x:H"
  have "EX H' h'. x:H' & graph H' h' <= graph H h
          & is_linearform H' h' & is_subspace H' E & is_subspace F H'
          & graph F f <= graph H' h' & (ALL x:H'. h' x <= p x)"
    by (rule some_H'h')
  thus "h x <= p x"
  proof (elim exE conjE)
    fix H' h'
    assume "x: H'" "graph H' h' <= graph H h"
      and a: "ALL x: H'. h' x <= p x"
    have [RS sym]: "h' x = h x" ..
    also from a have "h' x <= p x " ..
    finally show ?thesis .
  qed
qed
```

The following lemma is a property of linearforms on real vector spaces. It will be used for the lemma *rabs-HahnBanach* (see page 64). For real vector spaces the following inequations are equivalent:

$$\forall x \in H. |hx| \leq px \quad \text{and}$$
$$\forall x \in H. hx \leq px$$

```
lemma rabs_ineq_iff:
  "[| is_subspace H E; is_vectorspace E; is_seminorm E p;
  is_linearform H h |]
  ==> (ALL x:H. rabs (h x) <= p x) = (ALL x:H. h x <= p x)"
  (concl is "?L = ?R")
proof -
  assume "is_subspace H E" "is_vectorspace E" "is_seminorm E p"
         "is_linearform H h"
  have h: "is_vectorspace H" ..
  show ?thesis
  proof
    assume l: ?L
    show ?R
    proof
      fix x assume x: "x:H"
      have "h x <= rabs (h x)" by (rule rabs_ge_self)
      also from l have "... <= p x" ..
      finally show "h x <= p x" .
    qed
  next
    assume r: ?R
    show ?L
    proof
      fix x assume "x:H"
      show "!! a b. [| - a <= b; b <= a |] ==> rabs b <= a"
        by arith
      show "- p x <= h x"
      proof (rule real_minus_le)
        from h have "- h x = h (- x)"
          by (rule linearform_neg [RS sym])
        also from r have "... <= p (- x)" by (simp!)
        also have "... = p x"
          by (rule seminorm_minus, rule subspace_subsetD)
        finally show "- h x <= p x" .
      qed
      from r show "h x <= p x" ..
    qed
  qed
qed


end
```

12 Extending non-maximal functions

theory *HahnBanachExtLemmas = FunctionNorm:*

In this section the following context is presumed. Let E be a real vector space with a seminorm q on E. F is a subspace of E and f a linear function on F. We consider a subspace H of E that is a superspace of F and a linear form h on H. H is a not equal to E and x_0 is an element in $E \backslash H$. H is extended to the direct sum $H_0 = H + \mathit{lin}\, x_0$, so for any $x \in H_0$ the decomposition of $x = y + a \odot x$ with $y \in H$ is unique. h_0 is defined on H_0 by $h_0\, x = h\, y + a \cdot \xi$ for a certain ξ.

Subsequently we show some properties of this extension h_0 of h.

This lemma will be used to show the existence of a linear extension of f (see page 61). It is a consequence of the completeness of $\mathbb{R}$. To show

$$\exists \xi.\ \forall y \in F.\ a\, y \leq \xi \wedge \xi \leq b\, y$$

it suffices to show that

$$\forall u \in F.\ \forall v \in F.\ a\, u \leq b\, v$$

```
lemma ex_xi:
  "[| is_vectorspace F; !! u v. [| u:F; v:F |] ==> a u <= b v |]
  ==> EX (xi::real). ALL y:F. a y <= xi & xi <= b y"
proof -
  assume vs: "is_vectorspace F"
  assume r: "(!! u v. [| u:F; v:F |] ==> a u <= (b v::real))"
```

From the completeness of the reals follows: The set $S = \{a u \in F\}$ has a supremum, if it is non-empty and has an upper bound.

```
  let ?S = "{a u :: real | u. u:F}"

  have "EX xi. isLub UNIV ?S xi"
  proof (rule reals_complete)
```

The set S is non-empty, since $a\,\mathbf{0} \in S$:

```
    from vs have "a <0> : ?S" by force
    thus "EX X. X : ?S" ..
```

$b\,\mathbf{0}$ is an upper bound of S:

```
    show "EX Y. isUb UNIV ?S Y"
    proof
      show "isUb UNIV ?S (b <0>)"
      proof (intro isUbI setleI ballI)
        show "b <0> : UNIV" ..
      next
```

Every element $y \in S$ is less than $b\mathbf{0}$:

```
        fix y assume y: "y : ?S"
        from y have "EX u:F. y = a u" by fast
        thus "y <= b <0>"
        proof
          fix u assume "u:F"
          assume "y = a u"
          also have "a u <= b <0>" by (rule r) (simp!)+
          finally show ?thesis .
        qed
      qed
    qed
  qed

  thus "EX xi. ALL y:F. a y <= xi & xi <= b y"
  proof (elim exE)
    fix xi assume "isLub UNIV ?S xi"
    show ?thesis
    proof (intro exI conjI ballI)
```

For all $y \in F$ holds $a\,y \leq \xi$:

```
      fix y assume y: "y:F"
      show "a y <= xi"
      proof (rule isUbD)
        show "isUb UNIV ?S xi" ..
      qed (force!)
    next
```

For all $y \in F$ holds $\xi \leq b\,y$:

```
      fix y assume "y:F"
      show "xi <= b y"
      proof (intro isLub_le_isUb isUbI setleI)
        show "b y : UNIV" ..
        show "ALL ya : ?S. ya <= b y"
        proof
          fix au assume au: "au : ?S "
          hence "EX u:F. au = a u" by fast
          thus "au <= b y"
          proof
            fix u assume "u:F" assume "au = a u"
            also have "... <= b y" by (rule r)
            finally show ?thesis .
          qed
        qed
      qed
    qed
  qed
qed
```

The function h_0 is defined as a $h_0\, x = h\, y + a \cdot \xi$ where $x = y + a \prod \xi$ is a linear extension of h to H_0.

```
lemma h0_lf:
  "[| h0 == (λx. let (y, a) = SOME (y, a). x = y + a <*> x0 & y:H
                in h y + a * xi);
  H0 == H + lin x0; is_subspace H E; is_linearform H h; x0 ~: H;
  x0 : E; x0 ~= <0>; is_vectorspace E |]
  ==> is_linearform H0 h0"
proof -
  assume h0_def:
    "h0 == (λx. let (y, a) = SOME (y, a). x = y + a <*> x0 & y:H
              in h y + a * xi)"
    and H0_def: "H0 == H + lin x0"
    and vs: "is_subspace H E" "is_linearform H h" "x0 ~: H"
      "x0 ~= <0>" "x0 : E" "is_vectorspace E"

  have h0: "is_vectorspace H0"
  proof (unfold H0_def, rule vs_sum_vs)
    show "is_subspace (lin x0) E" ..
  qed

  show ?thesis
  proof
    fix x1 x2 assume x1: "x1 : H0" and x2: "x2 : H0"
```

We now have to show that h_0 is additive, i. e. $h_0\,(x_1 + x_2) = h_0\, x_1 + h_0\, x_2$ for $x_1, x_2 \in H$.

```
    have x1x2: "x1 + x2 : H0"
      by (rule vs_add_closed, rule h0)
    from x1
    have ex_x1: "EX y1 a1. x1 = y1 + a1 <*> x0  & y1 : H"
      by (unfold H0_def vs_sum_def lin_def) fast
    from x2
    have ex_x2: "EX y2 a2. x2 = y2 + a2 <*> x0 & y2 : H"
      by (unfold H0_def vs_sum_def lin_def) fast
    from x1x2
    have ex_x1x2: "EX y a. x1 + x2 = y + a <*> x0 & y : H"
      by (unfold H0_def vs_sum_def lin_def) fast

    from ex_x1 ex_x2 ex_x1x2
    show "h0 (x1 + x2) = h0 x1 + h0 x2"
    proof (elim exE conjE)
      fix y1 y2 y a1 a2 a
      assume y1: "x1 = y1 + a1 <*> x0"     and y1': "y1 : H"
         and y2: "x2 = y2 + a2 <*> x0"     and y2': "y2 : H"
         and y: "x1 + x2 = y + a <*> x0"   and y':  "y  : H"

      have ya: "y1 + y2 = y & a1 + a2 = a"
      proof (rule decomp_H0)
```

```
    show "y1 + y2 + (a1 + a2) <*> x0 = y + a <*> x0"
      by (simp! add: vs_add_mult_distrib2 [of E])
    show "y1 + y2 : H" ..
  qed

  have "h0 (x1 + x2) = h y + a * xi"
    by (rule h0_definite)
  also have "... = h (y1 + y2) + (a1 + a2) * xi"
    by (simp add: ya)
  also from vs y1' y2'
  have "... = h y1 + h y2 + a1 * xi + a2 * xi"
    by (simp add: linearform_add [of H]
                  real_add_mult_distrib)
  also have "... = (h y1 + a1 * xi) + (h y2 + a2 * xi)"
    by simp
  also have "h y1 + a1 * xi = h0 x1"
    by (rule h0_definite [RS sym])
  also have "h y2 + a2 * xi = h0 x2"
    by (rule h0_definite [RS sym])
  finally show ?thesis .
qed
```

We further have to show that h_0 is multiplicative, i. e. $h_0\,(c \odot x_1) = c \cdot h_0\,x_1$ for $x \in H$ and $c \in \mathbb{R}$.

```
next
  fix c x1 assume x1: "x1 : H0"
  have ax1: "c <*> x1 : H0"
    by (rule vs_mult_closed, rule h0)
  from x1 have ex_x: "!! x. x: H0
                      ==> EX y a. x = y + a <*> x0 & y : H"
    by (unfold H0_def vs_sum_def lin_def) fast

  from x1 have ex_x1: "EX y1 a1. x1 = y1 + a1 <*> x0 & y1 : H"
    by (unfold H0_def vs_sum_def lin_def) fast
  with ex_x [of "c <*> x1", OF ax1]
  show "h0 (c <*> x1) = c * (h0 x1)"
  proof (elim exE conjE)
    fix y1 y a1 a
    assume y1: "x1 = y1 + a1 <*> x0"       and y1': "y1 : H"
      and y: "c <*> x1 = y  + a  <*> x0"   and y':  "y  : H"

    have ya: "c <*> y1 = y & c * a1 = a"
    proof (rule decomp_H0)
      show "c <*> y1 + (c * a1) <*> x0 = y + a <*> x0"
        by (simp! add: add: vs_add_mult_distrib1)
      show "c <*> y1 : H" ..
    qed

    have "h0 (c <*> x1) = h y + a * xi"
```

```
      by (rule h0_definite)
    also have "... = h (c <*> y1) + (c * a1) * xi"
      by (simp add: ya)
    also from vs y1' have "... = c * h y1 + c * a1 * xi"
      by (simp add: linearform_mult [of H])
    also from vs y1' have "... = c * (h y1 + a1 * xi)"
      by (simp add: real_add_mult_distrib2 real_mult_assoc)
    also have "h y1 + a1 * xi = h0 x1"
      by (rule h0_definite [RS sym])
    finally show ?thesis .
  qed
 qed
qed
```

The linear extension h_0 of h is bounded by the seminorm p.

```
lemma h0_norm_pres:
 "[| h0 == (λx. let (y, a) = SOME (y, a). x = y + a <*> x0 & y:H
              in h y + a * xi);
 H0 == H + lin x0; x0 ~: H; x0 : E; x0 ~= <0>; is_vectorspace E;
 is_subspace H E; is_seminorm E p; is_linearform H h;
 ALL y:H. h y <= p y; (ALL y:H. - p (y + x0) - h y <= xi)
 & (ALL y:H. xi <= p (y + x0) - h y) |]
  ==> ALL x:H0. h0 x <= p x"
proof
 assume h0_def:
   "h0 == (λx. let (y, a) = SOME (y, a). x = y + a <*> x0 & y:H
            in (h y) + a * xi)"
   and H0_def: "H0 == H + lin x0"
   and vs: "x0 ~: H" "x0 : E" "x0 ~= <0>" "is_vectorspace E"
           "is_subspace H E" "is_seminorm E p" "is_linearform H h"
   and a: "ALL y:H. h y <= p y"
 presume a1: "ALL ya:H. - p (ya + x0) - h ya <= xi"
 presume a2: "ALL ya:H. xi <= p (ya + x0) - h ya"
 fix x assume "x : H0"
 have ex_x:
   "!! x. x : H0 ==> EX y a. x = y + a <*> x0 & y : H"
   by (unfold H0_def vs_sum_def lin_def) fast
 have "EX y a. x = y + a <*> x0 & y : H"
   by (rule ex_x)
 thus "h0 x <= p x"
 proof (elim exE conjE)
   fix y a assume x: "x = y + a <*> x0" and y: "y : H"
   have "h0 x = h y + a * xi"
     by (rule h0_definite)
```

Now we show $h\,y + a \cdot \xi \leq p(y + a \odot x_0)$ by case analysis on a.

```
   also have "... <= p (y + a <*> x0)"
   proof (rule linorder_linear_split)
```

```
  assume z: "a = 0r"
  with vs y a show ?thesis by simp
```

In the case $a < 0$, we use a_1 with ya taken as y/a:

```
next
  assume lz: "a < 0r" hence nz: "a ~= 0r" by simp
  from a1
  have "- p (rinv a <*> y + x0) - h (rinv a <*> y) <= xi"
    by (rule bspec) (simp!)
```

The thesis for this case now follows by a short calculation.

```
  hence "a * xi
        <= a * (- p (rinv a <*> y + x0) - h (rinv a <*> y))"
    by (rule real_mult_less_le_anti [OF lz])
  also have "... = - a * (p (rinv a <*> y + x0))
                   - a * (h (rinv a <*> y))"
    by (rule real_mult_diff_distrib)
  also from lz vs y have "- a * (p (rinv a <*> y + x0))
                           = p (a <*> (rinv a <*> y + x0))"
    by (simp add: seminorm_rabs_homogenous rabs_minus_eqI2)
  also from nz vs y have "... = p (y + a <*> x0)"
    by (simp add: vs_add_mult_distrib1)
  also from nz vs y have "a * (h (rinv a <*> y)) =  h y"
    by (simp add: linearform_mult [RS sym])
  finally have "a * xi <= p (y + a <*> x0) - h y" .

  hence "h y + a * xi <= h y + p (y + a <*> x0) - h y"
    by (simp add: real_add_left_cancel_le)
  thus ?thesis by simp
```

In the case $a > 0$, we use a_2 with ya taken as y/a:

```
next
  assume gz: "0r < a" hence nz: "a ~= 0r" by simp
  from a2
  have "xi <= p (rinv a <*> y + x0) - h (rinv a <*> y)"
    by (rule bspec) (simp!)
```

The thesis for this case follows by a short calculation:

```
  with gz have "a * xi
        <= a * (p (rinv a <*> y + x0) - h (rinv a <*> y))"
    by (rule real_mult_less_le_mono)
  also have "... = a * p (rinv a <*> y + x0)
                   - a * h (rinv a <*> y)"
    by (rule real_mult_diff_distrib2)
  also from gz vs y
  have "a * p (rinv a <*> y + x0)
       = p (a <*> (rinv a <*> y + x0))"
    by (simp add: seminorm_rabs_homogenous rabs_eqI2)
```

```
      also from nz vs y
      have "... = p (y + a <*> x0)"
        by (simp add: vs_add_mult_distrib1)
      also from nz vs y have "a * h (rinv a <*> y) = h y"
        by (simp add: linearform_mult [RS sym])
      finally have "a * xi <= p (y + a <*> x0) - h y" .

      hence "h y + a * xi <= h y + (p (y + a <*> x0) - h y)"
        by (simp add: real_add_left_cancel_le)
      thus ?thesis by simp
    qed
    also from x have "... = p x" by simp
    finally show ?thesis .
  qed
qed blast+

end
```

Part III
The Main Proof

13 The Hahn-Banach Theorem

```
theory HahnBanach
     = HahnBanachSupLemmas + HahnBanachExtLemmas + ZornLemma:
```

We present the proof of two different versions of the Hahn-Banach Theorem, closely following [1, §36].

13.1 The case of general linear spaces

Let f be a linear form f defined on a subspace F of a norm vector space E. If f is bounded by some seminorm q on E, it can be extended to a function on E in a norm-preserving way.

```
theorem HahnBanach:
  "[| is_vectorspace E; is_subspace F E; is_seminorm E p;
  is_linearform F f; ALL x:F. f x <= p x |]
  ==> EX h. is_linearform E h
          & (ALL x:F. h x = f x)
          & (ALL x:E. h x <= p x)"
proof -
  assume "is_vectorspace E" "is_subspace F E" "is_seminorm E p"
         "is_linearform F f" "ALL x:F. f x <= p x"
```

We define M to be the set of all linear extensions of f to superspaces of F, which are bounded by p.

```
  def M == "norm_pres_extensions E p F f"
```

We show that M is non-empty:

```
  have aM: "graph F f : norm_pres_extensions E p F f"
  proof (rule norm_pres_extensionI2)
    have "is_vectorspace F" ..
    thus "is_subspace F F" ..
  qed (blast!)+
```

13.1.1 Existence of a limit function the norm-preserving extensions

For every non-empty chain of norm-preserving extensions the union of all functions in the chain is again a norm-preserving extension. (The union is an upper bound

for all elements. It is even the least upper bound, because every upper bound of M is also an upper bound for $\bigcup c$.)

```
{
  fix c assume "c:chain M" "EX x. x:c"
  have "Union c : M"

  proof (unfold M_def, rule norm_pres_extensionI)
    show "EX (H::'a set) h::'a => real. graph H h = Union c
            & is_linearform H h
            & is_subspace H E
            & is_subspace F H
            & graph F f <= graph H h
            & (ALL x::'a:H. h x <= p x)"
    proof (intro exI conjI)
      let ?H = "domain (Union c)"
      let ?h = "funct (Union c)"

      show a: "graph ?H ?h = Union c"
      proof (rule graph_domain_funct)
        fix x y z assume "(x, y) : Union c" "(x, z) : Union c"
        show "z = y" by (rule sup_definite)
      qed

      show "is_linearform ?H ?h"
        by (simp! add: sup_lf a)
      show "is_subspace ?H E"
        by (rule sup_subE, rule a) (simp!)+
      show "is_subspace F ?H"
        by (rule sup_supF, rule a) (simp!)+
      show "graph F f <= graph ?H ?h"
        by (rule sup_ext, rule a) (simp!)+
      show "ALL x::'a:?H. ?h x <= p x"
        by (rule sup_norm_pres, rule a) (simp!)+
    qed
  qed
}
```

According to Zorn's Lemma there is a maximal norm-preserving extension $g \in M$.

```
  with aM have bex_g: "EX g:M. ALL x:M. g <= x --> g = x"
    by (simp! add: Zorn's_Lemma)

  thus ?thesis
  proof
    fix g assume g: "g:M" "ALL x:M. g <= x --> g = x"

    have ex_Hh:
      "EX H h. graph H h = g
           & is_linearform H h
           & is_subspace H E
```

```
        & is_subspace F H
        & graph F f <= graph H h
        & (ALL x:H. h x <= p x) "
    by (simp! add: norm_pres_extension_D)
  thus ?thesis
  proof (elim exE conjE, intro exI)
    fix H h
    assume "graph H h = g" "is_linearform (H::'a set) h"
           "is_subspace H E" "is_subspace F H"
      and h_ext: "graph F f <= graph H h"
      and h_bound: "ALL x:H. h x <= p x"

    have h: "is_vectorspace H" ..
    have f: "is_vectorspace F" ..
```

13.1.2 The domain of the limit function

```
have eq: "H = E"
proof (rule classical)
```

Assume that the domain of the supremum is not E,

```
  assume "H ~= E"
  have "H <= E" ..
  hence "H < E" ..
```

then there exists an element x_0 in E H:

```
  hence "EX x0:E. x0~:H"
    by (rule set_less_imp_diff_not_empty)
```

We get that h can be extended in a norm-preserving way to some H_0.

```
  hence "EX H0 h0. g <= graph H0 h0 & g ~= graph H0 h0
                & graph H0 h0 : M"
  proof
    fix x0 assume "x0:E" "x0~:H"

    have x0: "x0 ~= <0>"
    proof (rule classical)
      presume "x0 = <0>"
      with h have "x0:H" by simp
      thus ?thesis by contradiction
    qed blast
```

Define H_0 as the (direct) sum of H and the linear closure of x_0.

```
    def H0 == "H + lin x0"

    from h have xi: "EX xi. ALL y:H. - p (y + x0) - h y <= xi
                                  & xi <= p (y + x0) - h y"
```

```
proof (rule ex_xi)
  fix u v assume "u:H" "v:H"
  from h have "h v - h u = h (v - u)"
     by (simp! add: linearform_diff)
  also from h_bound have "... <= p (v - u)"
    by (simp!)
  also have "v - u = x0 + - x0 + v + - u"
    by (simp! add: diff_eq1)
  also have "... = v + x0 + - (u + x0)"
    by (simp!)
  also have "... = (v + x0) - (u + x0)"
    by (simp! add: diff_eq1)
  also have "p ... <= p (v + x0) + p (u + x0)"
     by (rule seminorm_diff_subadditive) (simp!)+
  finally have "h v - h u <= p (v + x0) + p (u + x0)" .

  thus "- p (u + x0) - h u <= p (v + x0) - h v"
    by (rule real_diff_ineq_swap)
qed
hence "EX h0. g <= graph H0 h0 & g ~= graph H0 h0
          & graph H0 h0 : M"
proof (elim exE, intro exI conjI)
  fix xi
  assume a: "ALL y:H. - p (y + x0) - h y <= xi
                   & xi <= p (y + x0) - h y"
```

Define h_0 as the canonical linear extension of h on H_0:

```
  def h0 ==
      "λx. let (y,a) = SOME (y, a). x = y + a <*> x0 & y:H
           in (h y) + a * xi"
```

We get that the graph of h_0 extend that of h.

```
  have  "graph H h <= graph H0 h0"
  proof (rule graph_extI)
    fix t assume "t:H"
    have "(SOME (y, a). t = y + a <*> x0 & y : H) = (t,0r)"
      by (rule decomp_H0_H, rule x0)
    thus "h t = h0 t" by (simp! add: Let_def)
  next
    show "H <= H0"
    proof (rule subspace_subset)
      show "is_subspace H H0"
      proof (unfold H0_def, rule subspace_vs_sum1)
        show "is_vectorspace H" ..
        show "is_vectorspace (lin x0)" ..
      qed
    qed
  qed
  thus "g <= graph H0 h0" by (simp!)
```

Apparently h_0 is not equal to h.

```
have "graph H h ~= graph H0 h0"
proof
  assume e: "graph H h = graph H0 h0"
  have "x0 : H0"
  proof (unfold H0_def, rule vs_sumI)
    show "x0 = <0> + x0" by (simp!)
    from h show "<0> : H" ..
    show "x0 : lin x0" by (rule x_lin_x)
  qed
  hence "(x0, h0 x0) : graph H0 h0" ..
  with e have "(x0, h0 x0) : graph H h" by simp
  hence "x0 : H" ..
  thus False by contradiction
qed
thus "g ~= graph H0 h0" by (simp!)
```

Furthermore h_0 is a norm-preserving extension of f.

```
have "graph H0 h0 : norm_pres_extensions E p F f"
proof (rule norm_pres_extensionI2)
  show "is_linearform H0 h0"
    by (rule h0_lf, rule x0) (simp!)+
  show "is_subspace H0 E"
    by (unfold H0_def, rule vs_sum_subspace,
       rule lin_subspace)

  have "is_subspace F H" .
  also from h lin_vs
  have [fold H0_def]: "is_subspace H (H + lin x0)" ..
  finally (subspace_trans [OF _ h])
  show f_h0: "is_subspace F H0" .

  show "graph F f <= graph H0 h0"
  proof (rule graph_extI)
    fix x assume "x:F"
    have "f x = h x" ..
    also have " ... = h x + 0r * xi" by simp
    also have "... = (let (y,a) = (x, 0r) in h y + a * xi)"
      by (simp add: Let_def)
    also have
      "(x, 0r) = (SOME (y, a). x = y + a <*> x0 & y : H)"
      by (rule decomp_H0_H [RS sym], rule x0) (simp!)+
    also have
      "(let (y,a) = (SOME (y,a). x = y + a <*> x0 & y : H)
        in h y + a * xi)
       = h0 x" by (simp!)
    finally show "f x = h0 x" .
  next
    from f_h0 show "F <= H0" ..
```

```
      qed

      show "ALL x:H0. h0 x <= p x"
        by (rule h0_norm_pres, rule x0) (assumption | (simp!))+
    qed
    thus "graph H0 h0 : M" by (simp!)
  qed
  thus ?thesis ..
qed
```

We have shown that h can still be extended to some h_0, in contradiction to the assumption that h is a maximal element.

```
  hence "EX x:M. g <= x & g ~= x"
    by (elim exE conjE, intro bexI conjI)
  hence "~ (ALL x:M. g <= x --> g = x)" by simp
  thus ?thesis by contradiction
qed
```

It follows $H = E$, and the thesis can be shown.

```
show "is_linearform E h & (ALL x:F. h x = f x)
     & (ALL x:E. h x <= p x)"
proof (intro conjI)
  from eq show "is_linearform E h" by (simp!)
  show "ALL x:F. h x = f x"
  proof (intro ballI, rule sym)
    fix x assume "x:F" show "f x = h x " ..
  qed
  from eq show "ALL x:E. h x <= p x" by (force!)
qed

qed
qed
qed
```

13.2 An alternative formulation of the theorem

The following alternative formulation of the Hahn-Banach Theorem uses the fact that for real numbers the following inequations are equivalent:[3]

$$\forall x \in H. |hx| \leq px \quad \text{and}$$
$$\forall x \in H. hx \leq px$$

```
theorem rabs_HahnBanach:
  "[| is_vectorspace E; is_subspace F E; is_seminorm E p;
  is_linearform F f; ALL x:F. rabs (f x) <= p x |]
  ==> EX g. is_linearform E g
```

[3] This was shown in lemma *rabs-ineq-iff* (see page 51).

```
          & (ALL x:F. g x = f x)
          & (ALL x:E. rabs (g x) <= p x)"
proof -
  assume e: "is_vectorspace E" "is_subspace F E" "is_seminorm E p"
           "is_linearform F f"  "ALL x:F. rabs (f x) <= p x"
  have "ALL x:F. f x <= p x"
    by (rule rabs_ineq_iff [RS iffD1])
  hence "EX g. is_linearform E g & (ALL x:F. g x = f x)
             & (ALL x:E. g x <= p x)"
    by (simp! only: HahnBanach)
  thus ?thesis
  proof (elim exE conjE)
    fix g assume "is_linearform E g" "ALL x:F. g x = f x"
                 "ALL x:E. g x <= p x"
    show ?thesis
    proof (intro exI conjI)
      from e show "ALL x:E. rabs (g x) <= p x"
        by (simp! add: rabs_ineq_iff [OF subspace_refl])
    qed
  qed
qed
```

13.3 The Hahn-Banach Theorem for normed spaces

Every continuous linear form f on a subspace F of a norm space E, can be extended to a continuous linear form g on E such that $\|f\| = \|g\|$.

```
theorem norm_HahnBanach:
  "[| is_normed_vectorspace E norm; is_subspace F E;
  is_linearform F f; is_continuous F norm f |]
  ==> EX g. is_linearform E g
         & is_continuous E norm g
         & (ALL x:F. g x = f x)
         & function_norm E norm g = function_norm F norm f"
proof -
  assume e_norm: "is_normed_vectorspace E norm"
  assume f: "is_subspace F E" "is_linearform F f"
  assume f_cont: "is_continuous F norm f"
  have e: "is_vectorspace E" ..
  with _ have f_norm: "is_normed_vectorspace F norm" ..
```

We define the function p on E as follows:

$$p\,x = \|f\| \cdot \|x\|$$

```
  def p == "λx. function_norm F norm f * norm x"
```

p is a seminorm on E:

```
  have q: "is_seminorm E p"
```

```
proof
  fix x y a assume "x:E" "y:E"
```

p is positive definite:

```
  show "0r <= p x"
  proof (unfold p_def, rule real_le_mult_order)
    from _ f_norm show "0r <= function_norm F norm f" ..
    show "0r <= norm x" ..
  qed
```

p is absolutely homogenous:

```
  show "p (a <*> x) = rabs a * p x"
  proof -
    have "p (a <*> x) = function_norm F norm f * norm (a <*> x)"
      by (simp!)
    also have "norm (a <*> x) = rabs a * norm x"
      by (rule normed_vs_norm_rabs_homogenous)
    also have "function_norm F norm f * (rabs a * norm x)
      = rabs a * (function_norm F norm f * norm x)"
      by (simp! only: real_mult_left_commute)
    also have "... = rabs a * p x" by (simp!)
    finally show ?thesis .
  qed
```

Furthermore, p is subadditive:

```
  show "p (x + y) <= p x + p y"
  proof -
    have "p (x + y) = function_norm F norm f * norm (x + y)"
      by (simp!)
    also
    have "... <= function_norm F norm f * (norm x + norm y)"
    proof (rule real_mult_le_le_mono1)
      from _ f_norm show "0r <= function_norm F norm f" ..
      show "norm (x + y) <= norm x + norm y" ..
    qed
    also have "... = function_norm F norm f * norm x
                     + function_norm F norm f * norm y"
      by (simp! only: real_add_mult_distrib2)
    finally show ?thesis by (simp!)
  qed
qed
```

f is bounded by p.

```
  have "ALL x:F. rabs (f x) <= p x"
  proof
    fix x assume "x:F"
     from f_norm show "rabs (f x) <= p x"
       by (simp! add: norm_fx_le_norm_f_norm_x)
  qed
```

Using the fact that p is a seminorm and f is bounded by q we can apply the Hahn-Banach Theorem for real vector spaces. So f can be extended in a norm-preserving way to some function g on the whole vector space E.

```
with e f q
have "EX g. is_linearform E g & (ALL x:F. g x = f x)
          & (ALL x:E. rabs (g x) <= p x)"
  by (simp! add: rabs_HahnBanach)

thus ?thesis
proof (elim exE conjE)
  fix g
  assume "is_linearform E g" and a: "ALL x:F. g x = f x"
     and b: "ALL x:E. rabs (g x) <= p x"

  show "EX g. is_linearform E g
          & is_continuous E norm g
          & (ALL x:F. g x = f x)
          & function_norm E norm g = function_norm F norm f"
  proof (intro exI conjI)
```

We futhermore have to show that g is also continuous:

```
    show g_cont: "is_continuous E norm g"
    proof
      fix x assume "x:E"
      from b [RS bspec, OF this]
      show "rabs (g x) <= function_norm F norm f * norm x"
        by (unfold p_def)
    qed
```

To complete the proof, we show that $\|g\| = \|f\|$.

```
    show "function_norm E norm g = function_norm F norm f"
      (is "?L = ?R")
    proof (rule order_antisym)
```

First we show $\|g\| \leq \|f\|$. The function norm $\|g\|$ is defined as the smallest $c \in \mathbb{R}$ such that

$$\forall x \in E.\ |g\,x| \leq c \cdot \|x\|$$

Furthermore holds

$$\forall x \in E.\ |g\,x| \leq \|f\| \cdot \|x\|$$

```
      have "ALL x:E. rabs (g x) <= function_norm F norm f * norm x"
      proof
        fix x assume "x:E"
        show "rabs (g x) <= function_norm F norm f * norm x"
          by (simp!)
      qed

      with _ g_cont show "?L <= ?R"
      proof (rule fnorm_le_ub)
```

```
      from f_cont f_norm show "0r <= function_norm F norm f" ..
    qed
```

The other direction is achieved by a similar argument.

```
    have "ALL x:F. rabs (f x) <= function_norm E norm g * norm x"
    proof
      fix x assume "x : F"
      from a have "g x = f x" ..
      hence "rabs (f x) = rabs (g x)" by simp
      also from _ _ g_cont
      have "... <= function_norm E norm g * norm x"
        by (rule norm_fx_le_norm_f_norm_x) (simp!)+
      finally
      show "rabs (f x) <= function_norm E norm g * norm x" .
    qed

    with f_norm f_cont show "?R <= ?L"
    proof (rule fnorm_le_ub)
      from g_cont show "0r <= function_norm E norm g" ..
    qed
  qed
 qed
qed
qed

end
```

References

[1] H. Heuser. *Funktionalanalysis: Theorie und Anwendung*. Teubner, 1986.

[2] L. Narici and E. Beckenstein. The Hahn-Banach Theorem: The life and times. In *Topology Atlas*. York University, Toronto, Ontario, Canada, 1996. http://at.yorku.ca/topology/preprint.htm and http://at.yorku.ca/p/a/a/a/16.htm.

[3] B. Nowak and A. Trybulec. Hahn-Banach theorem. *Journal of Formalized Mathematics*, 5, 1993. http://mizar.uwb.edu.pl/JFM/Vol5/hahnban.html.