Perfect Fields

Manuel Eberl, Katharina Kreuzer

April 18, 2024

Abstract

This entry provides a type class for *perfect fields*. A perfect field K can be characterized by one of the following equivalent conditions [2]:

1. Any irreducible polynomial p is separable, i.e. $\gcd(p, p') = 1$, or, equivalently, $p' \neq 0$.
2. Either $\mathrm{char}(K) = 0$ or $\mathrm{char}(K) = p > 0$ and the Frobenius endomorphism $x \mapsto x^p$ is surjective (i.e. every element of K has a p-th root).

We define perfect fields using the second characterization and show the equivalence to the first characterization. The implication "2 $\Rightarrow$ 1" is relatively straightforward using the injectivity of the Frobenius homomorphism.

Examples for perfect fields are [2]:

- any field of characteristic 0 (e.g. $\mathbb{R}$ and $\mathbb{C}$)
- any finite field (i.e. $\mathbb{F}_q$ for $q = p^n$, $n > 0$ and p prime)
- any algebraically closed field (for example the formal Puiseux series over finite fields)

Contents

1 Perfect Fields

```
theory Perfect_Fields
imports
  "HOL-Computational_Algebra.Computational_Algebra"
  "Berlekamp_Zassenhaus.Finite_Field"
begin

lemma (in vector_space) bij_betw_representation:
  assumes [simp]: "independent B" "finite B"
  shows   "bij_betw (λv. ∑b∈B. scale (v b) b) (B →E UNIV) (span B)"
proof (rule bij_betwI)
  show "(λv. ∑b∈B. v b *s b) ∈ (B →E UNIV) → local.span B"
    (is "?f ∈ _")
    by (auto intro: span_sum span_scale span_base)
  show "(λx. restrict (representation B x) B) ∈ local.span B → B →E
UNIV"
    (is "?g ∈ _") by auto
  show "?g (?f v) = v" if "v ∈ B →E UNIV" for v
  proof
    fix b :: 'b
    show "?g (?f v) b = v b"
    proof (cases "b ∈ B")
      case b: True
      have "?g (?f v) b = (∑i∈B. local.representation B (v i *s i) b)"
        using b by (subst representation_sum) (auto intro: span_scale
span_base)
      also have "... = (∑i∈B. v i * local.representation B i b)"
        by (intro sum.cong) (auto simp: representation_scale span_base)
      also have "... = (∑i∈{b}. v i * local.representation B i b)"
        by (intro sum.mono_neutral_right) (auto simp: representation_basis
b)
      also have "... = v b"
        by (simp add: representation_basis b)
      finally show "?g (?f v) b = v b" .
    qed (use that in auto)
  qed
  show "?f (?g v) = v" if "v ∈ span B" for v
    using that by (simp add: sum_representation_eq)
qed

lemma (in vector_space) card_span:
  assumes [simp]: "independent B" "finite B"
  shows   "card (span B) = CARD('a) ^ card B"
proof -
  have "card (B →E (UNIV :: 'a set)) = card (span B)"
    by (rule bij_betw_same_card, rule bij_betw_representation) fact+
  thus ?thesis
    by (simp add: card_PiE dim_span_eq_card_independent)
```

```
qed

lemma (in zero_neq_one) CARD_neq_1: "CARD('a) ≠ Suc 0"
proof
  assume "CARD('a) = Suc 0"
  have "{0, 1} ⊆ (UNIV :: 'a set)"
    by simp
  also have "is_singleton (UNIV :: 'a set)"
    by (simp add: is_singleton_altdef ‹CARD('a) = _›)
  then obtain x :: 'a where "UNIV = {x}"
    by (elim is_singletonE)
  finally have "0 = (1 :: 'a)"
    by blast
  thus False
    using zero_neq_one by contradiction
qed

theorem CARD_finite_field_is_CHAR_power: "∃n>0. CARD('a :: finite_field)
= CHAR('a) ^ n"
proof -
  define s :: "'a ring_char mod_ring ⇒ 'a ⇒ 'a" where
    "s = (λx y. of_int (to_int_mod_ring x) * y)"
  interpret vector_space s
    by unfold_locales (auto simp: s_def algebra_simps to_int_mod_ring_add
to_int_mod_ring_mult)
  obtain B where B: "independent B" "span B = UNIV"
    by (rule basis_exists[of UNIV]) auto
  have [simp]: "finite B"
    by simp
  have "card (span B) = CHAR('a) ^ card B"
    using B by (subst card_span) auto
  hence *: "CARD('a) = CHAR('a) ^ card B"
    using B by simp
  from * have "card B ≠ 0"
    by (auto simp: B(2) CARD_neq_1)
  with * show ?thesis
    by blast
qed
```

1.1 The Freshman's Dream in rings of non-zero characteristic

```
lemma (in comm_semiring_1) freshmans_dream:
  fixes x y :: 'a and n :: nat
  assumes "prime CHAR('a)"
  assumes n_def: "n = CHAR('a)"
  shows   "(x + y) ^ n = x ^ n + y ^ n"
proof -
  interpret comm_semiring_prime_char
    by standard (auto intro!: exI[of _ "CHAR('a)"] assms)
```

```
  have "n > 0"
    unfolding n_def by simp
  have "(x + y) ^ n = (∑k≤n. of_nat (n choose k) * x ^ k * y ^ (n -
k))"
    by (rule binomial_ring)
  also have "... = (∑k∈{0,n}. of_nat (n choose k) * x ^ k * y ^ (n -
k))"
  proof (intro sum.mono_neutral_right ballI)
    fix k assume "k ∈ {..n} - {0, n}"
    hence k: "k > 0" "k < n"
      by auto
    have "CHAR('a) dvd (n choose k)"
      unfolding n_def
      by (rule dvd_choose_prime) (use k in ‹auto simp: n_def›)
    hence "of_nat (n choose k) = (0 :: 'a)"
      using of_nat_eq_0_iff_char_dvd by blast
    thus "of_nat (n choose k) * x ^ k * y ^ (n - k) = 0"
      by simp
  qed auto
  finally show ?thesis
    using ‹n > 0› by (simp add: add_ac)
qed

lemma (in comm_semiring_1) freshmans_dream':
  assumes [simp]: "prime CHAR('a)" and "m = CHAR('a) ^ n"
  shows "(x + y :: 'a) ^ m = x ^ m + y ^ m"
  unfolding assms(2)
proof (induction n)
  case (Suc n)
  have "(x + y) ^ (CHAR('a) ^ n * CHAR('a)) = ((x + y) ^ (CHAR('a) ^ n))
^ CHAR('a)"
    by (rule power_mult)
  thus ?case
    by (simp add: Suc.IH freshmans_dream Groups.mult_ac flip: power_mult)
qed auto

lemma (in comm_semiring_1) freshmans_dream_sum:
  fixes f :: "'b ⇒ 'a"
  assumes "prime CHAR('a)" and "n = CHAR('a)"
  shows "sum f A ^ n = sum (λi. f i ^ n) A"
  using assms
  by (induct A rule: infinite_finite_induct)
     (auto simp add: power_0_left freshmans_dream)

lemma (in comm_semiring_1) freshmans_dream_sum':
  fixes f :: "'b ⇒ 'a"
  assumes "prime CHAR('a)" "m = CHAR('a) ^ n"
  shows   "sum f A ^ m = sum (λi. f i ^ m) A"
  using assms
```

```
  by (induction A rule: infinite_finite_induct)
     (auto simp: freshmans_dream' power_0_left)
```

1.2 The Frobenius endomorphism

```
definition (in semiring_1) frob :: "'a ⇒ 'a" where
  "frob x = x ^ CHAR('a)"

definition (in semiring_1) inv_frob :: "'a ⇒ 'a" where
  "inv_frob x = (if x ∈ {0, 1} then x else if x ∈ range frob then inv_into
UNIV frob x else x)"

lemma (in semiring_1) inv_frob_0 [simp]: "inv_frob 0 = 0"
  and inv_frob_1 [simp]: "inv_frob 1 = 1"
  by (simp_all add: inv_frob_def)

lemma (in semiring_prime_char) frob_0 [simp]: "frob (0 :: 'a) = 0"
  by (simp add: frob_def power_0_left)

lemma (in semiring_1) frob_1 [simp]: "frob 1 = 1"
  by (simp add: frob_def)

lemma (in comm_semiring_1) frob_mult: "frob (x * y) = frob x * frob (y
:: 'a)"
  by (simp add: frob_def power_mult_distrib)

lemma (in comm_semiring_1)
  frob_add: "prime CHAR('a) ⟹ frob (x + y :: 'a) = frob x + frob (y
:: 'a)"
  by (simp add: frob_def freshmans_dream)

lemma (in comm_ring_1) frob_uminus: "prime CHAR('a) ⟹ frob (-x :: 'a)
= -frob x"
proof -
  assume "prime CHAR('a)"
  hence "frob (-x) + frob x = 0"
    by (subst frob_add [symmetric]) (auto simp: frob_def power_0_left)
  thus ?thesis
    by (simp add: add_eq_0_iff)
qed

lemma (in comm_ring_prime_char) frob_diff:
  "prime CHAR('a) ⟹ frob (x - y :: 'a) = frob x - frob (y :: 'a)"
  using frob_add[of x "-y"] by (simp add: frob_uminus)

interpretation frob_sr: semiring_hom "frob :: 'a :: {comm_semiring_prime_char}
⇒ 'a"
  by standard (auto simp: frob_add frob_mult)
```

```
interpretation frob: ring_hom "frob :: 'a :: {comm_ring_prime_char} ⇒
'a"
  by standard auto

interpretation frob: field_hom "frob :: 'a :: {field_prime_char} ⇒ 'a"
  by standard auto

lemma frob_mod_ring' [simp]: "(x :: 'a :: prime_card mod_ring) ^ CARD('a)
= x"
  by (metis CARD_mod_ring finite_field_power_card_eq_same)

lemma frob_mod_ring [simp]: "frob (x :: 'a :: prime_card mod_ring) =
x"
  by (simp add: frob_def)

context semiring_1_no_zero_divisors
begin

lemma frob_eq_0D:
  "frob (x :: 'a) = 0 ⟹ x = 0"
  by (auto simp: frob_def)

lemma frob_eq_0_iff [simp]:
  "frob (x :: 'a) = 0 ⟷ x = 0 ∧ CHAR('a) > 0"
  by (auto simp: frob_def)

end


context idom_prime_char
begin

lemma inj_frob: "inj (frob :: 'a ⇒ 'a)"
proof
  fix x y :: 'a
  assume "frob x = frob y"
  hence "frob (x - y) = 0"
    by (simp add: frob_diff del: frob_eq_0_iff)
  thus "x = y"
    by simp
qed

lemma frob_eq_frob_iff [simp]:
  "frob (x :: 'a) = frob y ⟷ x = y"
  using inj_frob by (auto simp: inj_def)

lemma frob_eq_1_iff [simp]: "frob (x :: 'a) = 1 ⟷ x = 1"
  using frob_eq_frob_iff by fastforce
```

```
lemma inv_frob_frob [simp]: "inv_frob (frob (x :: 'a)) = x"
  by (simp add: inj_frob inv_frob_def)

lemma frob_inv_frob [simp]:
  assumes "x ∈ range frob"
  shows   "frob (inv_frob x) = (x :: 'a)"
  using assms by (auto simp: inj_frob inv_frob_def)

lemma inv_frob_eqI: "frob y = x ⟹ inv_frob x = y"
  using inv_frob_frob local.frob_def by force

lemma inv_frob_eq_0_iff [simp]: "inv_frob (x :: 'a) = 0 ⟷ x = 0"
  using inj_frob by (auto simp: inv_frob_def split: if_splits)

end



class surj_frob = field_prime_char +
  assumes surj_frob [simp]: "surj (frob :: 'a ⇒ 'a)"
begin

lemma in_range_frob [simp, intro]: "(x :: 'a) ∈ range frob"
  using surj_frob by blast

lemma inv_frob_eq_iff [simp]: "inv_frob (x :: 'a) = y ⟷ frob y = x"
  using frob_inv_frob inv_frob_frob by blast

end


context alg_closed_field
begin

lemma alg_closed_surj_frob:
  assumes "CHAR('a) > 0"
  shows   "surj (frob :: 'a ⇒ 'a)"
proof -
  show "surj (frob :: 'a ⇒ 'a)"
  proof safe
    fix x :: 'a
    obtain y where "y ^ CHAR('a) = x"
      using nth_root_exists CHAR_pos assms by blast
    hence "frob y = x"
      using CHAR_pos by (simp add: frob_def)
    thus "x ∈ range frob"
      by (metis rangeI)
  qed auto
```

```
qed

end
```

The following type class describes a field with a surjective Frobenius endomorphism that is effectively computable. This includes all finite fields.

```
class inv_frob = surj_frob +
  fixes inv_frob_code :: "'a ⇒ 'a"
  assumes inv_frob_code: "inv_frob x = inv_frob_code x"

lemmas [code] = inv_frob_code


context finite_field
begin

subclass surj_frob
proof
  show "surj (frob :: 'a ⇒ 'a)"
    using inj_frob finite_UNIV by (simp add: finite_UNIV_inj_surj)
qed

end


lemma inv_frob_mod_ring [simp]: "inv_frob (x :: 'a :: prime_card mod_ring)
= x"
  by (auto simp: frob_def)

instantiation mod_ring :: (prime_card) inv_frob
begin

definition inv_frob_code_mod_ring :: "'a mod_ring ⇒ 'a mod_ring" where
  "inv_frob_code_mod_ring x = x"

instance
  by standard (auto simp: inv_frob_code_mod_ring_def)

end
```

1.3 Inverting the Frobenius endomorphism on polynomials

If *K* is a field of prime characteristic *p* with a surjective Frobenius endomorphism, every polynomial *P* with *P' = 0* has a *p*-th root.

To see that, let $\phi(a) = a^p$ denote the Frobenius endomorphism of *K* and its extension to *K[X]*.

If `P' = 0` for some `P ∈ K[X]`, then `P` must be of the form

$$P = a_0 + a_p x^p + a_{2p} x^{2p} + \ldots + a_{kp} x^{kp} \ .$$

If we now set

$$Q := \phi^{-1}(a_0) + \phi^{-1}(a_p)x + \phi^{-1}(a_{2p})x^2 + \ldots + \phi^{-1}(a_{kp})x^k$$

we get $\phi(Q) = P$, i.e. Q is the p-th root of $P(x)$.

```
lift_definition inv_frob_poly :: "'a :: field poly ⇒ 'a poly" is
  "λp i. if CHAR('a) = 0 then p i else inv_frob (p (i * CHAR('a)) :: 'a)"
proof goal_cases
  case (1 f)
  show ?case
  proof (cases "CHAR('a) > 0")
    case True
    from 1 obtain N where N: "f i = 0" if "i ≥ N" for i
      using cofinite_eq_sequentially eventually_sequentially by auto
    have "inv_frob (f (i * CHAR('a))) = 0" if "i ≥ N" for i
    proof -
      have "f (i * CHAR('a)) = 0"
      proof (rule N)
        show "N ≤ i * CHAR('a)"
          using that True
          by (metis One_nat_def Suc_leI le_trans mult.right_neutral mult_le_mono2)
      qed
      thus "inv_frob (f (i * CHAR('a))) = 0"
        by (auto simp: power_0_left)
    qed
    thus ?thesis using True
      unfolding cofinite_eq_sequentially eventually_sequentially by auto
  qed (use 1 in auto)
qed

lemma coeff_inv_frob_poly [simp]:
  fixes p :: "'a :: field poly"
  assumes "CHAR('a) > 0"
  shows "poly.coeff (inv_frob_poly p) i = inv_frob (poly.coeff p (i *
CHAR('a)))"
  using assms by transfer auto

lemma inv_frob_poly_0 [simp]: "inv_frob_poly 0 = 0"
  by transfer (auto simp: fun_eq_iff power_0_left)

lemma inv_frob_poly_1 [simp]: "inv_frob_poly 1 = 1"
  by transfer (auto simp: fun_eq_iff power_0_left)

lemma degree_inv_frob_poly_le:
  fixes p :: "'a :: field poly"
```

```
  assumes "CHAR('a) > 0"
  shows "Polynomial.degree (inv_frob_poly p) ≤ Polynomial.degree p div
CHAR('a)"
proof (intro degree_le allI impI)
  fix i assume "Polynomial.degree p div CHAR('a) < i"
  hence "i * CHAR('a) > Polynomial.degree p"
    using assms div_less_iff_less_mult by blast
  thus "Polynomial.coeff (inv_frob_poly p) i = 0"
    by (simp add: coeff_eq_0 power_0_left assms)
qed

context
  assumes "SORT_CONSTRAINT('a :: comm_ring_1)"
  assumes prime_char: "prime CHAR('a)"
begin

lemma poly_power_prime_char_as_sum_of_monoms:
  fixes h :: "'a poly"
  shows "h ^ CHAR('a) = (∑i≤Polynomial.degree h. Polynomial.monom (Polynomial.coeff
h i ^ CHAR('a)) (CHAR('a)*i))"
proof -
  have "h ^ CHAR('a) = (∑i≤Polynomial.degree h. Polynomial.monom (Polynomial.coeff
h i) i) ^ CHAR('a)"
    by (simp add: poly_as_sum_of_monoms)
  also have "... = (∑i≤Polynomial.degree h. (Polynomial.monom (Polynomial.coeff
h i) i) ^ CHAR('a))"
    by (simp add: freshmans_dream_sum prime_char)
  also have "... = (∑i≤Polynomial.degree h. Polynomial.monom (Polynomial.coeff
h i ^ CHAR('a)) (CHAR('a)*i))"
  proof (rule sum.cong, rule)
    fix x assume x: "x ∈ {..Polynomial.degree h}"
    show "Polynomial.monom (Polynomial.coeff h x) x ^ CHAR('a) = Polynomial.monom
(Polynomial.coeff h x ^ CHAR('a)) (CHAR('a) * x)"
      by (unfold poly_eq_iff, auto simp add: monom_power)
  qed
  finally show ?thesis .
qed

lemma coeff_of_prime_char_power [simp]:
  fixes y :: "'a poly"
  shows "poly.coeff (y ^ CHAR('a)) (i * CHAR('a)) = poly.coeff y i ^ CHAR('a)"
  using prime_char
  by (subst poly_power_prime_char_as_sum_of_monoms, subst Polynomial.coeff_sum)
     (auto intro: le_degree simp: power_0_left)

lemma coeff_of_prime_char_power':
  fixes y :: "'a poly"
  shows "poly.coeff (y ^ CHAR('a)) i =
          (if CHAR('a) dvd i then poly.coeff y (i div CHAR('a)) ^ CHAR('a)
```

```
else 0)"
proof -
  have "poly.coeff (y ^ CHAR('a)) i =
          (∑j≤Polynomial.degree y. Polynomial.coeff (Polynomial.monom
(Polynomial.coeff y j ^ CHAR('a)) (CHAR('a) * j)) i)"
    by (subst poly_power_prime_char_as_sum_of_monoms, subst Polynomial.coeff_sum)
auto
  also have "... = (∑j∈(if CHAR('a) dvd i ∧ i div CHAR('a) ≤ Polynomial.degree
y then {i div CHAR('a)} else {}).
                     Polynomial.coeff (Polynomial.monom (Polynomial.coeff
y j ^ CHAR('a)) (CHAR('a) * j)) i)"
    by (intro sum.mono_neutral_right) (use prime_char in auto)
  also have "... = (if CHAR('a) dvd i then poly.coeff y (i div CHAR('a))
^ CHAR('a) else 0)"
  proof (cases "CHAR('a) dvd i ∧ i div CHAR('a) > Polynomial.degree y")
    case True
    hence "Polynomial.coeff y (i div CHAR('a)) ^ CHAR('a) = 0"
      using prime_char by (simp add: coeff_eq_0 zero_power power_0_left)
    thus ?thesis
      by auto
  qed auto
  finally show ?thesis .
qed

end


context
  assumes "SORT_CONSTRAINT('a :: field)"
  assumes pos_char: "CHAR('a) > 0"
begin

interpretation field_prime_char "(/)" inverse "(*)" "1 :: 'a" "(+)" 0 "(-)"
uminus
  rewrites "semiring_1.frob 1 (*) (+) (0 :: 'a) = frob" and
           "semiring_1.inv_frob 1 (*) (+) (0 :: 'a) = inv_frob" and
           "semiring_1.semiring_char 1 (+) 0 TYPE('a) = CHAR('a)"
proof unfold_locales
  have *: "class.semiring_1 (1 :: 'a) (*) (+) 0" ..
  have [simp]: "semiring_1.of_nat (1 :: 'a) (+) 0 = of_nat"
    by (auto simp: of_nat_def semiring_1.of_nat_def[OF *])
  thus "∃n>0. semiring_1.of_nat (1 :: 'a) (+) 0 n = 0"
    by (intro exI[of _ "CHAR('a)"]) (use pos_char in auto)
  show "semiring_1.semiring_char 1 (+) 0 TYPE('a) = CHAR('a)"
    by (simp add: fun_eq_iff semiring_char_def semiring_1.semiring_char_def[OF
*])
  show [simp]: "semiring_1.frob (1 :: 'a) (*) (+) 0 = frob"
    by (simp add: frob_def semiring_1.frob_def[OF *] fun_eq_iff
                 power.power_def power_def semiring_char_def semiring_1.semiring_char_def[
```

```
*])
  show "semiring_1.inv_frob (1 :: 'a) (*) (+) 0 = inv_frob"
    by (simp add: inv_frob_def semiring_1.inv_frob_def[OF *] fun_eq_iff)
qed

lemma inv_frob_poly_power': "inv_frob_poly (p ^ CHAR('a) :: 'a poly)
= p"
  using prime_CHAR_semidom[OF pos_char] pos_char
  by (auto simp: poly_eq_iff simp flip: frob_def)

lemma inv_frob_poly_power:
  fixes p :: "'a poly"
  assumes "is_nth_power CHAR('a) p" and "n = CHAR('a)"
  shows   "inv_frob_poly p ^ CHAR('a) = p"
proof -
  from assms(1) obtain q where q: "p = q ^ CHAR('a)"
    by (elim is_nth_powerE)
  thus ?thesis using assms
    by (simp add: q inv_frob_poly_power')
qed

theorem pderiv_eq_0_imp_nth_power:
  assumes "pderiv (p :: 'a poly) = 0"
  assumes [simp]: "surj (frob :: 'a ⇒ 'a)"
  shows   "is_nth_power CHAR('a) p"
proof -
  have *: "poly.coeff p n = 0" if n: "¬CHAR('a) dvd n" for n
  proof (cases "n = 0")
    case False
    have "poly.coeff (pderiv p) (n - 1) = of_nat n * poly.coeff p n"
      using False by (auto simp: coeff_pderiv)
    with assms and n show "poly.coeff p n = 0"
      by (auto simp: of_nat_eq_0_iff_char_dvd)
  qed (use that in auto)

  have **: "inv_frob_poly p ^ CHAR('a) = p"
  proof (rule poly_eqI)
    fix n :: nat
    show "poly.coeff (inv_frob_poly p ^ CHAR('a)) n = poly.coeff p n"
      using * CHAR_dvd_CARD[where ?'a = 'a]
      by (subst coeff_of_prime_char_power')
         (auto simp: poly_eq_iff frob_def [symmetric]
                     coeff_of_prime_char_power'[where ?'a = 'a] simp
flip: power_mult)
  qed

  show ?thesis
    by (subst **[symmetric]) auto
qed
```

end

1.4 Code generation

We now also make this notion of "taking the p-th root of a polynomial" executable. For this, we need an auxiliary function that takes a list $[x_0, \ldots, x_m]$ and returns the list of every n-th element, i.e. it throws away all elements except those x_i where i is a multiple of n.

```
fun take_every :: "nat ⇒ 'a list ⇒ 'a list" where
  "take_every _ [] = []"
| "take_every n (x # xs) = x # take_every n (drop (n - 1) xs)"

lemma take_every_0 [simp]: "take_every 0 xs = xs"
  by (induction xs) auto

lemma take_every_1 [simp]: "take_every (Suc 0) xs = xs"
  by (induction xs) auto

lemma int_length_take_every: "n > 0 ⟹ int (length (take_every n xs))
= ceiling (length xs / n)"
proof (induction n xs rule: take_every.induct)
  case (2 n x xs)
  show ?case
  proof (cases "Suc (length xs) ≥ n")
    case True
    thus ?thesis using 2
      by (auto simp: dvd_imp_le of_nat_diff diff_divide_distrib split:
if_splits)
  next
    case False
    hence "⌈(1 + real (length xs)) / real n⌉ = 1"
      by (intro ceiling_unique) auto
    thus ?thesis using False
      by auto
  qed
qed auto

lemma length_take_every:
  "n > 0 ⟹ length (take_every n xs) = nat (ceiling (length xs / n))"
  using int_length_take_every[of n xs] by simp

lemma take_every_nth [simp]:
  "n > 0 ⟹ i < length (take_every n xs) ⟹ take_every n xs ! i = xs
! (n * i)"
proof (induction n xs arbitrary: i rule: take_every.induct)
  case (2 n x xs i)
  show ?case
```

```
  proof (cases i)
    case (Suc j)
    have "n - Suc 0 ≤ length xs"
      using Suc "2.prems" nat_le_linear by force
    hence "drop (n - Suc 0) xs ! (n * j) = xs ! (n - 1 + n * j)"
      using Suc by (subst nth_drop) auto
    also have "n - 1 + n * j = n + n * j - 1"
      using ‹n > 0› by linarith
    finally show ?thesis
      using "2.IH"[of j] "2.prems" Suc by simp
  qed auto
qed auto

lemma coeffs_eq_strip_whileI:
  assumes "⋀i. i < length xs ⟹ Polynomial.coeff p i = xs ! i"
  assumes "p ≠ 0 ⟹ length xs > Polynomial.degree p"
  shows   "Polynomial.coeffs p = strip_while ((=) 0) xs"
proof (rule coeffs_eqI)
  fix n :: nat
  show "Polynomial.coeff p n = nth_default 0 (strip_while ((=) 0) xs)
n"
    using assms
    by (metis coeff_0 coeff_Poly_eq coeffs_Poly le_degree nth_default_coeffs_eq

      nth_default_eq_dflt_iff nth_default_nth order_le_less_trans)
qed auto
```

This implements the code equation for *inv_frob_poly*.

```
lemma inv_frob_poly_code [code]:
  "Polynomial.coeffs (inv_frob_poly (p :: 'a :: field_prime_char poly))
=
     (if CHAR('a) = 0 then Polynomial.coeffs p else
        map inv_frob (strip_while ((=) 0) (take_every CHAR('a) (Polynomial.coeffs
p))))"
    (is "_ = If _ _ ?rhs")
proof (cases "CHAR('a) = 0 ∨ p = 0")
  case False
  from False have "p ≠ 0"
    by auto
  have "Polynomial.coeffs (inv_frob_poly p) =
          strip_while ((=) 0) (map inv_frob (take_every CHAR('a) (Polynomial.coeffs
p)))"
  proof (rule coeffs_eq_strip_whileI)
    fix i assume i: "i < length (map inv_frob (take_every CHAR('a) (Polynomial.coeffs
p)))"
    show "Polynomial.coeff (inv_frob_poly p) i = map inv_frob (take_every
CHAR('a) (Polynomial.coeffs p)) ! i"
    proof -
      have "i < length (take_every CHAR('a) (Polynomial.coeffs p))"
```

```
        using i by simp
      also have "length (take_every CHAR('a) (Polynomial.coeffs p)) =
                   nat ⌈(Polynomial.degree p + 1) / real CHAR('a)⌉"
        using False CHAR_pos[where ?'a = 'a]
        by (simp add: length_take_every length_coeffs)
      finally have "i < real (Polynomial.degree p + 1) / real CHAR('a)"
        by linarith
      hence "real i * real CHAR('a) < real (Polynomial.degree p + 1)"
        using False CHAR_pos[where ?'a = 'a] by (simp add: field_simps)
      hence "i * CHAR('a) ≤ Polynomial.degree p"
        unfolding of_nat_mult [symmetric] by linarith
      hence "Polynomial.coeffs p ! (i * CHAR('a)) = Polynomial.coeff p
(i * CHAR('a))"
        using False by (intro coeffs_nth) (auto simp: length_take_every)
      thus ?thesis using False i CHAR_pos[where ?'a = 'a]
        by (auto simp: nth_default_def mult.commute)
    qed
  next
    assume nz: "inv_frob_poly p ≠ 0"
    have "Polynomial.degree (inv_frob_poly p) ≤ Polynomial.degree p div
CHAR('a)"
      by (rule degree_inv_frob_poly_le) (fact CHAR_pos)
    also have "... < nat ⌈(real (Polynomial.degree p) + 1) / real CHAR('a)⌉"
      using CHAR_pos[where ?'a = 'a]
      by (metis div_less_iff_less_mult linorder_not_le nat_le_real_less
of_nat_0_less_iff
              of_nat_ceiling of_nat_mult pos_less_divide_eq)
    also have "... = length (take_every CHAR('a) (Polynomial.coeffs p))"
      using CHAR_pos[where ?'a = 'a] ‹p ≠ 0› by (simp add: length_take_every
length_coeffs add_ac)
    finally show "length (map inv_frob (take_every CHAR('a) (Polynomial.coeffs
p))) > Polynomial.degree (inv_frob_poly p)"
      by simp_all
  qed
  also have "strip_while ((=) 0) (map inv_frob (take_every CHAR('a) (Polynomial.coeffs
p))) =
             map inv_frob (strip_while ((=) 0 ∘ inv_frob) (take_every
CHAR('a) (Polynomial.coeffs p)))"
    by (rule strip_while_map)
  also have "(=) 0 ∘ inv_frob = (=) (0 :: 'a)"
    by (auto simp: fun_eq_iff)
  finally show ?thesis
    using False by metis
qed auto
```

1.5 Perfect fields

We now introduce perfect fields. The textbook definition of a perfect field is that every irreducible polynomial is separable, i.e. if a polynomial P has no non-trivial divisors then $\gcd(P, P') = 0$.

For technical reasons, this is somewhat difficult to express in Isabelle/HOL's typeclass system. We therefore use the following much simpler equivalent definition (and prove equivalence later): a field is perfect if it either has characteristic 0 or its Frobenius endomorphism is surjective.

```
class perfect_field = field +
  assumes perfect_field: "CHAR('a) = 0 ∨ surj (frob :: 'a ⇒ 'a)"

context field_char_0
begin
subclass perfect_field
  by standard auto
end

context surj_frob
begin
subclass perfect_field
  by standard auto
end

context alg_closed_field
begin
subclass perfect_field
  by standard (use alg_closed_surj_frob in auto)
end

theorem irreducible_imp_pderiv_nonzero:
  assumes "irreducible (p :: 'a :: perfect_field poly)"
  shows   "pderiv p ≠ 0"
proof (cases "CHAR('a) = 0")
  case True
  interpret A: semiring_1 "1 :: 'a" "(*)" "(+)" "0 :: 'a" ..
  have *: "class.semiring_1 (1 :: 'a) (*) (+) 0" ..
  interpret A: field_char_0 "(/)" inverse "(*)" "1 :: 'a" "(+)" 0 "(-)"
uminus
  proof
    have "inj (of_nat :: nat ⇒ 'a)"
      by (auto simp: inj_on_def of_nat_eq_iff_cong_CHAR True)
    also have "of_nat = semiring_1.of_nat (1 :: 'a) (+) 0"
      by (simp add: of_nat_def [abs_def] semiring_1.of_nat_def [OF *,
abs_def])
    finally show "inj ..." .
  qed

  show ?thesis
  proof
    assume "pderiv p = 0"
    hence **: "poly.coeff p (Suc n) = 0" for n
```

```
      by (auto simp: poly_eq_iff coeff_pderiv of_nat_eq_0_iff_char_dvd
True simp del: of_nat_Suc)
    have "poly.coeff p n = 0" if "n > 0" for n
      using **[of "n - 1"] that by (cases n) auto
    hence "Polynomial.degree p = 0"
      by force
    thus False
      using assms by force
  qed

next
  case False
  hence [simp]: "surj (frob :: 'a ⇒ 'a)"
    by (meson perfect_field)

  interpret A: field_prime_char "(/)" inverse "(*)" "1 :: 'a" "(+)" 0 "(-)"
uminus
  proof
    have *: "class.semiring_1 1 (*) (+) (0 :: 'a)" ..
    have "semiring_1.of_nat 1 (+) (0 :: 'a) = of_nat"
      by (simp add: fun_eq_iff of_nat_def semiring_1.of_nat_def[OF *])
    thus "∃n>0. semiring_1.of_nat 1 (+) 0 n = (0 :: 'a)"
      by (intro exI[of _ "CHAR('a)"]) (use False in auto)
  qed

  show ?thesis
  proof
    assume "pderiv p = 0"
    hence "is_nth_power CHAR('a) p"
      using pderiv_eq_0_imp_nth_power[of p] surj_frob False by simp
    then obtain q where "p = q ^ CHAR('a)"
      by (elim is_nth_powerE)
    with assms show False
      by auto
  qed
qed

corollary irreducible_imp_separable:
  assumes "irreducible (p :: 'a :: perfect_field poly)"
  shows   "coprime p (pderiv p)"
proof (rule coprimeI)
  fix q assume q: "q dvd p" "q dvd pderiv p"
  have "¬p dvd q"
  proof
    assume "p dvd q"
    hence "p dvd pderiv p"
      using q dvd_trans by blast
    hence "Polynomial.degree p ≤ Polynomial.degree (pderiv p)"
      by (rule dvd_imp_degree_le) (use assms irreducible_imp_pderiv_nonzero
```

```
in auto)
    also have "... ≤ Polynomial.degree p - 1"
      using degree_pderiv_le by auto
    finally have "Polynomial.degree p = 0"
      by simp
    with assms show False
      using irreducible_imp_pderiv_nonzero is_unit_iff_degree by blast
  qed
  with ‹q dvd p› show "is_unit q"
    using assms comm_semiring_1_class.irreducibleD' by blast
qed

end
```

1.6 Alternative definition of perfect fields

```
theory Perfect_Field_Altdef
imports
  "HOL-Algebra.Algebraic_Closure_Type"
  Perfect_Fields
begin
```

In the following, we will show that our definition of perfect fields is equivalent to the usual textbook one (for example [1]). That is: a field in which every irreducible polynomial is separable (or, equivalently, has non-zero derivative) either has characteristic 0 or a surjective Frobenius endomorphism.

The proof works like this:

Let's call our field *K* with prime characteristic *p*. Suppose there were some *c* ∈ *K* that is not a *p*-th root. The polynomial $P := X^p - c$ in $K[X]$ clearly has a zero derivative and is therefore not separable. By our assumption, it must then have a monic non-trivial factor $Q \in K[X]$.

Let *L* be some field extension of *K* where *c* does have a *p*-th root α (in our case, we choose *L* to be the algebraic closure of *K*).

Clearly, *Q* is also a non-trivial factor of *P* in *L*. However, we also have *P = X^p - c = X^p - α^p = (X - α)^p*, so we must have $Q = (X - \alpha)^m$ for some *0 ≤ m < p* since *X - α* is prime.

However, the coefficient of X^{m-1} in $(X - \alpha)^m$ is *-mα*, and since *Q ∈ K[X]* we must have *-mα ∈ K* and therefore *α ∈ K*.

```
theorem perfect_field_alt:
  assumes "⋀p :: 'a :: field_gcd poly. Factorial_Ring.irreducible p ⟹
pderiv p ≠ 0"
  shows   "CHAR('a) = 0 ∨ surj (frob :: 'a ⇒ 'a)"
proof (cases "CHAR('a) = 0")
  case False
  let ?p = "CHAR('a)"
  from False have "Factorial_Ring.prime ?p"
```

```
    by (simp add: prime_CHAR_semidom)
  hence "?p > 1"
    using prime_gt_1_nat by blast
  note p = ‹Factorial_Ring.prime ?p› ‹?p > 1›

  interpret to_ac: map_poly_inj_comm_ring_hom "to_ac :: 'a ⇒ 'a alg_closure"
    by unfold_locales auto

  have "surj (frob :: 'a ⇒ 'a)"
  proof safe
    fix c :: 'a
    obtain α :: "'a alg_closure" where α: "α ^ ?p = to_ac c"
      using p nth_root_exists[of ?p "to_ac c"] by auto
    define P where "P = Polynomial.monom 1 ?p + [:-c:]"
    define P' where "P' = map_poly to_ac P"
    have deg: "Polynomial.degree P = ?p"
      unfolding P_def using p by (subst degree_add_eq_left) (auto simp:
degree_monom_eq)

    have "[:-α, 1:] ^ ?p = ([:0, 1:] + [:-α:]) ^ ?p"
      by (simp add: one_pCons)
    also have "... = [:0, 1:] ^ ?p - [:α^?p:]"
      using p by (subst freshmans_dream) (auto simp: poly_const_pow minus_power_prime_CHAR)
    also have "α ^ ?p = to_ac c"
      by (simp add: α)
    also have "[:0, 1:] ^ CHAR('a) - [:to_ac c:] = P'"
      by (simp add: P_def P'_def to_ac.hom_add to_ac.hom_power
                to_ac.base.map_poly_pCons_hom monom_altdef)
    finally have eq: "P' = [:-α, 1:] ^ ?p" ..

    have "¬is_unit P" "P ≠ 0"
      using deg p by auto
    then obtain Q where Q: "Factorial_Ring.prime Q" "Q dvd P"
      by (metis prime_divisor_exists)
    have "monic Q"
      using unit_factor_prime[OF Q(1)] by (auto simp: unit_factor_poly_def
one_pCons)

    from Q(2) have "map_poly to_ac Q dvd P'"
      by (auto simp: P'_def)
    hence "map_poly to_ac Q dvd [:-α, 1:] ^ ?p"
      by (simp add: ‹P' = [:-α, 1:] ^ ?p›)
    moreover have "Factorial_Ring.prime_elem [:-α, 1:]"
      by (intro prime_elem_linear_field_poly) auto
    hence "Factorial_Ring.prime [:-α, 1:]"
      unfolding Factorial_Ring.prime_def by (auto simp: normalize_monic)
    ultimately obtain m where "m ≤ ?p" "normalize (map_poly to_ac Q)
= [:-α, 1:] ^ m"
      using divides_primepow by blast
```

```
    hence "map_poly to_ac Q = [:-α, 1:] ^ m"
      using ‹monic Q› by (subst (asm) normalize_monic) auto
    moreover from this have "m > 0"
      using Q by (intro Nat.gr0I) auto
    moreover have "m ≠ ?p"
    proof
      assume "m = ?p"
      hence "Q = P"
        using ‹map_poly to_ac Q = [:-α, 1:] ^ m› eq
        by (simp add: P'_def to_ac.injectivity)
      with Q have "Factorial_Ring.irreducible P"
        using idom_class.prime_elem_imp_irreducible by blast
      with assms have "pderiv P ≠ 0"
        by blast
      thus False
        by (auto simp: P_def pderiv_add pderiv_monom of_nat_eq_0_iff_char_dvd)
    qed
    ultimately have m: "m ∈ {0<..<?p}" "map_poly to_ac Q = [:-α, 1:]
^ m"
      using ‹m ≤ ?p› by auto

    from m(1) have "¬?p dvd m"
      using p by auto
    have "poly.coeff ([:-α, 1:] ^ m) (m - 1) = - of_nat (m choose (m -
1)) * α"
      using m(1) by (subst coeff_linear_poly_power) auto
    also have "m choose (m - 1) = m"
      using ‹0 < m› by (subst binomial_symmetric) auto
    also have "[:-α, 1:] ^ m = map_poly to_ac Q"
      using m(2) ..
    also have "poly.coeff ... (m - 1) = to_ac (poly.coeff Q (m - 1))"
      by simp
    finally have "α = to_ac (-poly.coeff Q (m - 1) / of_nat m)"
      using m(1) p ‹¬?p dvd m› by (auto simp: field_simps of_nat_eq_0_iff_char_dvd)
    hence "(- poly.coeff Q (m - 1) / of_nat m) ^ ?p = c"
      using α by (metis to_ac.base.eq_iff to_ac.base.hom_power)
    thus "c ∈ range frob"
      unfolding frob_def by blast
  qed auto
  thus ?thesis ..
qed auto

corollary perfect_field_alt':
  assumes "⋀p :: 'a :: field_gcd poly. Factorial_Ring.irreducible p ⟹
Rings.coprime p (pderiv p)"
  shows   "CHAR('a) = 0 ∨ surj (frob :: 'a ⇒ 'a)"
proof (rule perfect_field_alt)
  fix p :: "'a poly"
  assume p: "Factorial_Ring.irreducible p"
```

```
  with assms[OF p] show "pderiv p ≠ 0"
    by auto
qed

end
```

References

[1] K. Conrad. Perfect fields. Online at https://kconrad.math.uconn.edu/blurbs/galoistheory/perfect.pdf, 2021. Course notes, University of Connecticut.

[2] Wikipedia contributors. Perfect field — Wikipedia, the free encyclopedia, 2023. [Online; accessed 3-November-2023].