Perfect Fields

Manuel Eberl, Katharina Kreuzer

March 17, 2025

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 $\rightarrow_E$ UNIV) (span B)"*
⟨*proof*⟩

lemma **(in** *vector_space)* *card_span:*
  assumes *[simp]: "independent B" "finite B"*
  shows *"card (span B) = CARD('a) ^ card B"*
⟨*proof*⟩

lemma **(in** *zero_neq_one)* *CARD_neq_1: "CARD('a) ≠ Suc 0"*
⟨*proof*⟩

theorem *CARD_finite_field_is_CHAR_power: "∃n>0. CARD('a :: finite_field) = CHAR('a) ^ n"*
⟨*proof*⟩

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*⟩

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"*
  ⟨*proof*⟩

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"*
  ⟨*proof*⟩

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"*
⟨*proof*⟩

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"*
⟨*proof*⟩

lemma *(in semiring_prime_char) frob_0 [simp]: "frob (0 :: 'a) = 0"*
⟨*proof*⟩

lemma *(in semiring_1) frob_1 [simp]: "frob 1 = 1"*
⟨*proof*⟩

lemma *(in comm_semiring_1) frob_mult: "frob (x * y) = frob x * frob (y :: 'a)"*
⟨*proof*⟩

lemma *(in comm_semiring_1)*
frob_add: "prime CHAR('a) ⟹ frob (x + y :: 'a) = frob x + frob (y :: 'a)"
⟨*proof*⟩

lemma *(in comm_ring_1) frob_uminus: "prime CHAR('a) ⟹ frob (-x :: 'a) = -frob x"*
⟨*proof*⟩

lemma *(in comm_ring_prime_char) frob_diff:*
"prime CHAR('a) ⟹ frob (x - y :: 'a) = frob x - frob (y :: 'a)"
⟨*proof*⟩

interpretation *frob_sr: semiring_hom "frob :: 'a :: {comm_semiring_prime_char} ⇒ 'a"*
⟨*proof*⟩

interpretation *frob: ring_hom "frob :: 'a :: {comm_ring_prime_char} ⇒ 'a"*
⟨*proof*⟩

interpretation *frob: field_hom "frob :: 'a :: {field_prime_char} ⇒ 'a"*
⟨*proof*⟩

lemma *frob_mod_ring' [simp]: "(x :: 'a :: prime_card mod_ring) ^ CARD('a) = x"*
⟨*proof*⟩

lemma *frob_mod_ring [simp]: "frob (x :: 'a :: prime_card mod_ring) = x"*
⟨*proof*⟩

context *semiring_1_no_zero_divisors*
begin

lemma *frob_eq_0D:*
"frob (x :: 'a) = 0 ⟹ x = 0"
⟨*proof*⟩

lemma *frob_eq_0_iff [simp]:*
"frob (x :: 'a) = 0 ⟷ x = 0 ∧ CHAR('a) > 0"
⟨*proof*⟩

end

context *idom_prime_char*
begin

lemma *inj_frob: "inj (frob :: 'a ⇒ 'a)"*
⟨*proof*⟩

lemma *frob_eq_frob_iff [simp]:*
"frob (x :: 'a) = frob y ⟷ x = y"
⟨*proof*⟩

lemma *frob_eq_1_iff [simp]: "frob (x :: 'a) = 1 ⟷ x = 1"*
⟨*proof*⟩

lemma *inv_frob_frob [simp]: "inv_frob (frob (x :: 'a)) = x"*
⟨*proof*⟩

lemma *frob_inv_frob [simp]:*
assumes *"x ∈ range frob"*
shows *"frob (inv_frob x) = (x :: 'a)"*
⟨*proof*⟩

lemma *inv_frob_eqI: "frob y = x ⟹ inv_frob x = y"*
⟨*proof*⟩

lemma *inv_frob_eq_0_iff [simp]: "inv_frob (x :: 'a) = 0 ⟷ x = 0"*
⟨*proof*⟩

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"*
  ⟨*proof*⟩

lemma *inv_frob_eq_iff [simp]: "inv_frob (x :: 'a) = y ⟷ frob y = x"*
  ⟨*proof*⟩

end

context *alg_closed_field*
begin

lemma *alg_closed_surj_frob:*
  assumes *"CHAR('a) > 0"*
  shows *"surj (frob :: 'a ⇒ 'a)"*
⟨*proof*⟩

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*⟩

end

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

⟨proof⟩

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
⟨proof⟩

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⟩

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)))"*
⟨proof⟩

lemma *inv_frob_poly_0 [simp]: "inv_frob_poly 0 = 0"*
⟨proof⟩

lemma *inv_frob_poly_1 [simp]: "inv_frob_poly 1 = 1"*
⟨proof⟩

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*⟩

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*⟩

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)"*
⟨*proof*⟩

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*⟩

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*⟩

lemma *inv_frob_poly_power': "inv_frob_poly (p ^ CHAR('a) :: 'a poly) = p"*
⟨*proof*⟩

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⟩

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⟩

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"
  ⟨proof⟩

lemma take_every_1 [simp]: "take_every (Suc 0) xs = xs"
  ⟨proof⟩

lemma int_length_take_every: "n > 0 ⟹ int (length (take_every n xs))
= ceiling (length xs / n)"
⟨proof⟩

lemma length_take_every:
  "n > 0 ⟹ length (take_every n xs) = nat (ceiling (length xs / n))"
  ⟨proof⟩

lemma take_every_nth [simp]:
  "n > 0 ⟹ i < length (take_every n xs) ⟹ take_every n xs ! i = xs
! (n * i)"
⟨proof⟩

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⟩
```

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⟩
```

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
  ⟨proof⟩
end

context surj_frob
begin
subclass perfect_field
  ⟨proof⟩
end

context alg_closed_field
begin
subclass perfect_field
  ⟨proof⟩
end

theorem irreducible_imp_pderiv_nonzero:
  assumes "irreducible (p :: 'a :: perfect_field poly)"
  shows   "pderiv p ≠ 0"
⟨proof⟩

corollary irreducible_imp_separable:
  assumes "irreducible (p :: 'a :: perfect_field poly)"
  shows   "coprime p (pderiv p)"
⟨proof⟩
```

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⟩

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⟩

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].