Finite Fields

Emin Karayel

June 16, 2022

Abstract

This entry formalizes the classification of the finite fields (also called Galois fields): For each prime power p^n there exists exactly one (up to isomorphisms) finite field of that size and there are no other finite fields. The derivation includes a formalization of the characteristic of rings, the Frobenius endomorphism, formal differentiation for polynomials in HOL-Algebra and Gauss' formula for the number of monic irreducible polynomials over finite fields:

$$\frac{1}{n}\sum_{d|n}\mu(d)p^{n/d}.$$

The proofs are based on the books from Ireland and Rosen [3], as well as, Lidl and Niederreiter [4].

Contents

1 Introduction

The following section starts with preliminary results. Section 3 introduces the characteristic of rings with the Frobenius endomorphism. Whenever it makes sense, the definitions and facts do not assume the finiteness of the fields or rings. For example the characteristic is defined over arbitrary rings (and also fields).

While formal derivatives do exist for type-class based structures in `HOL-Computational_Algebra`, as far as I can tell, they do not exist for the structure based polynomials in `HOL-Algebra`. These are introduced in Section 4.

A cornerstone of the proof is the derivation of Gauss' formula for the number of monic irreducible polynomials over a finite field R in Section 6.2. The proof follows the derivation by Ireland and Rosen [3, §7] closely, with the caveat that it does not assume that R is a simple prime field, but that it is just a finite field. This works by adjusting a proof step with the information that the order of a finite field must be of the form p^n, where p is the characteristic of the field, derived in Section 3. The final step relies on the Möbius inversion theorem formalized by Eberl [2].[1]

With Gauss' formula it is possible to show the existence of the finite fields of order p^n where p is a prime and $n > 0$. During the proof the fact that the polynomial $X^n - X$ splits in a field of order n is also derived, which is necessary for the uniqueness result as well.

The uniqueness proof is inspired by the derivation of the same result in Lidl and Niederreiter [4], but because of the already derived existence proof for irreducible polynomials, it was possible to reduce its complexity.

The classification consists of three theorems:

- *Existence*: For each prime power p^n there exists a finite field of that size. This is shown at the conclusion of Section 6.2.
- *Uniqueness*: Any two finite fields of the same size are isomorphic. This is shown at the conclusion of Section 7.
- *Completeness*: Any finite fields' size must be a prime power. This is shown at the conclusion of Section 3.

[1] Thanks to Katharina Kreuzer for discovering that formalization.

2 Preliminary Results

theory *Finite-Fields-Preliminary-Results*
imports *HOL−Algebra.Polynomial-Divisibility*
begin

2.1 Summation in the discrete topology

The following lemmas transfer the corresponding result from the summation over finite sets to summation over functions which vanish outside of a finite set.

lemma *sum′-subtractf-nat*:
fixes $f :: 'a \Rightarrow nat$
assumes *finite* $\{i \in A.\ f\ i \neq 0\}$
assumes $\bigwedge i.\ i \in A \Longrightarrow g\ i \leq f\ i$
shows $sum'\ (\lambda i.\ f\ i - g\ i)\ A = sum'\ f\ A - sum'\ g\ A$
(**is** *?lhs* = *?rhs*)
proof −
have *c*:*finite* $\{i \in A.\ g\ i \neq 0\}$
using *assms*(*2*)
by (*intro finite-subset*[*OF - assms*(*1*)] *subsetI*, *force*)
let $?B = \{i \in A.\ f\ i \neq 0 \vee g\ i \neq 0\}$

have *b*:$?B = \{i \in A.\ f\ i \neq 0\} \cup \{i \in A.\ g\ i \neq 0\}$
by (*auto simp add*:*set-eq-iff*)
have *a*:*finite ?B*
using *assms*(*1*) *c* **by** (*subst b*, *simp*)
have *?lhs* = $sum'\ (\lambda i.\ f\ i - g\ i)$ *?B*
by (*intro sum.mono-neutral-cong-right′*, *simp-all*)
also have ... = $sum\ (\lambda i.\ f\ i - g\ i)$ *?B*
by (*intro sum.eq-sum a*)
also have ... = *sum f ?B* − *sum g ?B*
using *assms*(*2*) **by** (*subst sum-subtractf-nat*, *auto*)
also have ... = $sum'\ f$ *?B* − $sum'\ g$ *?B*
by (*intro arg-cong2*[**where** *f*=(−)] *sum.eq-sum*[*symmetric*] *a*)
also have ... = *?rhs*
by (*intro arg-cong2*[**where** *f*=(−)] *sum.mono-neutral-cong-left′*)
simp-all
finally show *?thesis*
by *simp*
qed

lemma *sum′-nat-eq-0-iff*:
fixes $f :: 'a \Rightarrow nat$
assumes *finite* $\{i \in A.\ f\ i \neq 0\}$
assumes $sum'\ f\ A = 0$
shows $\bigwedge i.\ i \in A \Longrightarrow f\ i = 0$
proof −
let $?B = \{i \in A.\ f\ i \neq 0\}$

```
  have sum f ?B = sum' f ?B
    by (intro sum.eq-sum[symmetric] assms(1))
  also have ... = sum' f A
    by (intro sum.non-neutral')
  also have ... = 0 using assms(2) by simp
  finally have a:sum f ?B = 0 by simp
  have ⋀i. i ∈ ?B ⟹ f i = 0
    using sum-nonneg-0[OF assms(1) - a] by blast
  thus ⋀i. i ∈ A ⟹ f i = 0
    by blast
qed

lemma sum'-eq-iff:
  fixes f :: 'a ⇒ nat
  assumes finite {i ∈ A. f i ≠ 0}
  assumes ⋀i. i ∈ A ⟹ f i ≥ g i
  assumes sum' f A ≤ sum' g A
  shows ∀ i ∈ A. f i = g i
proof −
  have {i ∈ A. g i ≠ 0} ⊆ {i ∈ A. f i ≠ 0}
    using assms(2) order-less-le-trans
    by (intro subsetI, auto)
  hence a:finite {i ∈ A. g i ≠ 0}
    by (rule finite-subset, intro assms(1))
  have  {i ∈ A. f i − g i ≠ 0} ⊆ {i ∈ A. f i ≠ 0}
    by (intro subsetI, simp-all)
  hence b: finite {i ∈ A. f i − g i ≠ 0}
    by (rule finite-subset, intro assms(1))
  have sum' (λi. f i − g i) A = sum' f A − sum' g A
    using assms(1,2) a by (subst sum'-subtractf-nat, auto)
  also have ... = 0
    using assms(3) by simp
  finally have sum' (λi. f i − g i) A = 0 by simp
  hence ⋀i. i ∈ A ⟹ f i − g i = 0
    using sum'-nat-eq-0-iff[OF b] by simp
  thus ?thesis
    using assms(2) diff-is-0-eq' diffs0-imp-equal by blast
qed
```

2.2 Polynomials

The embedding of the constant polynomials into the polynomials is injective:

```
lemma (in ring) poly-of-const-inj:
  inj poly-of-const
proof −
  have coeff (poly-of-const x) 0 = x for x
    unfolding poly-of-const-def normalize-coeff[symmetric]
```

by *simp*
thus *?thesis* **by** (*metis injI*)
qed

lemma (**in** *domain*) *embed-hom*:
assumes *subring K R*
shows *ring-hom-ring* (*K*[*X*]) (*poly-ring R*) *id*
proof (*rule ring-hom-ringI*)
show *ring* (*K*[*X*])
using *univ-poly-is-ring*[*OF assms*(*1*)] **by** *simp*
show *ring* (*poly-ring R*)
using *univ-poly-is-ring*[*OF carrier-is-subring*] **by** *simp*
have $K \subseteq$ *carrier R*
using *subringE*(*1*)[*OF assms*(*1*)] **by** *simp*
thus $\bigwedge x.\ x \in$ *carrier* (*K* [*X*]) $\Longrightarrow$ *id x* $\in$ *carrier* (*poly-ring R*)
unfolding *univ-poly-carrier*[*symmetric*] *polynomial-def* **by** *auto*
show *id* ($x \otimes_{K\ [X]}\ y$) = *id x* $\otimes_{poly\text{-}ring\ R}$ *id y*
if $x \in$ *carrier* (*K* [*X*]) $y \in$ *carrier* (*K* [*X*]) **for** *x y*
unfolding *univ-poly-mult* **by** *simp*
show *id* ($x \oplus_{K\ [X]}\ y$) = *id x* $\oplus_{poly\text{-}ring\ R}$ *id y*
if $x \in$ *carrier* (*K* [*X*]) $y \in$ *carrier* (*K* [*X*]) **for** *x y*
unfolding *univ-poly-add* **by** *simp*
show *id* $\mathbf{1}_{K\ [X]} = \mathbf{1}_{poly\text{-}ring\ R}$
unfolding *univ-poly-one* **by** *simp*
qed

The following are versions of the properties of the degrees of polynomials, that abstract over the definition of the polynomial ring structure. In the theories *HOL−Algebra.Polynomials* and also *HOL−Algebra.Polynomial-Divisibility* these abstract version are usually indicated with the suffix "shell", consider for example: *domain.pdivides-iff-shell*.

lemma (**in** *ring*) *degree-add-distinct*:
assumes *subring K R*
assumes $f \in$ *carrier* (*K*[*X*]) − {$\mathbf{0}_{K[X]}$}
assumes $g \in$ *carrier* (*K*[*X*]) − {$\mathbf{0}_{K[X]}$}
assumes *degree f* ≠ *degree g*
shows *degree* ($f \oplus_{K[X]}\ g$) = *max* (*degree f*) (*degree g*)
unfolding *univ-poly-add* **using** *assms*(*2*,*3*,*4*)
by (*subst poly-add-degree-eq*[*OF assms*(*1*)])
(*auto simp*:*univ-poly-carrier univ-poly-zero*)

lemma (**in** *domain*) *degree-mult*:
assumes *subring K R*
assumes $f \in$ *carrier* (*K*[*X*]) − {$\mathbf{0}_{K[X]}$}
assumes $g \in$ *carrier* (*K*[*X*]) − {$\mathbf{0}_{K[X]}$}
shows *degree* ($f \otimes_{K[X]}\ g$) = *degree f* + *degree g*

```
  unfolding univ-poly-mult using assms(2,3)
  by (subst poly-mult-degree-eq[OF assms(1)])
    (auto simp:univ-poly-carrier univ-poly-zero)

lemma (in ring) degree-one:
  degree (1_{K[X]}) = 0
  unfolding univ-poly-one by simp

lemma (in domain) pow-non-zero:
  x ∈ carrier R ⟹ x ≠ 0 ⟹ x [^] (n :: nat) ≠ 0
  using integral by (induction n, auto)

lemma (in domain) degree-pow:
  assumes subring K R
  assumes f ∈ carrier (K[X]) − {0_{K[X]}}
  shows degree (f [^]_{K[X]} n) = degree f * n
proof −
  interpret p:domain K[X]
    using univ-poly-is-domain[OF assms(1)] by simp

  show ?thesis
  proof (induction n)
    case 0
    then show ?case by (simp add:univ-poly-one)
  next
    case (Suc n)
    have degree (f [^]_{K [X]} Suc n) = degree (f [^]_{K [X]} n ⊗_{K[X]} f)
      by simp
    also have ... = degree (f [^]_{K [X]} n) + degree f
      using p.pow-non-zero assms(2)
      by (subst degree-mult[OF assms(1)], auto)
    also have ... = degree f * Suc n
      by (subst Suc, simp)
    finally show ?case by simp
  qed
qed

lemma (in ring) degree-var:
  degree (X_R) = 1
  unfolding var-def by simp

lemma (in domain) var-carr:
  fixes n :: nat
  assumes subring K R
  shows X_R ∈ carrier (K[X]) − {0_{K [X]}}
proof −
  have X_R ∈ carrier (K[X])
    using var-closed[OF assms(1)] by simp
```

```
  moreover have X ≠ 𝟬_K [X]
    unfolding var-def univ-poly-zero by simp
  ultimately show ?thesis by simp
qed

lemma (in domain) var-pow-carr:
  fixes n :: nat
  assumes subring K R
  shows X_R [^]_K [X] n ∈ carrier (K[X]) − {𝟬_K [X]}
proof −
  interpret p:domain K[X]
    using univ-poly-is-domain[OF assms(1)] by simp

  have X_R [^]_K [X] n ∈ carrier (K[X])
    using var-pow-closed[OF assms(1)] by simp
  moreover have X ≠ 𝟬_K [X]
    unfolding var-def univ-poly-zero by simp
  hence X_R [^]_K [X] n ≠ 𝟬_K [X]
    using var-closed(1)[OF assms(1)]
    by (intro p.pow-non-zero, auto)
  ultimately show ?thesis by simp
qed

lemma (in domain) var-pow-degree:
  fixes n :: nat
  assumes subring K R
  shows degree (X_R [^]_K [X] n) = n
  using var-carr[OF assms(1)] degree-var
  by (subst degree-pow[OF assms(1)], auto)

lemma (in domain) finprod-non-zero:
  assumes finite A
  assumes f ∈ A → carrier R − {𝟬}
  shows (⨂ i ∈ A. f i) ∈ carrier R − {𝟬}
  using assms
proof (induction A rule:finite-induct)
  case empty
  then show ?case by simp
next
  case (insert x F)
  have finprod R f (insert x F) = f x ⊗ finprod R f F
    using insert by (subst finprod-insert, simp-all add:Pi-def)
  also have ... ∈ carrier R−{𝟬}
    using integral insert by auto
  finally show ?case by simp
qed

lemma (in domain) degree-prod:
```

```
  assumes finite A
  assumes subring K R
  assumes f ∈ A → carrier (K[X]) − {0_K[X]}
  shows degree (⊗_K[X] i ∈ A. f i) = (∑ i ∈ A. degree (f i))
  using assms
proof −
  interpret p:domain K[X]
    using univ-poly-is-domain[OF assms(2)] by simp

  show ?thesis
    using assms(1,3)
  proof (induction A rule: finite-induct)
    case empty
    then show ?case by (simp add:univ-poly-one)
  next
    case (insert x F)
    have degree (finprod (K[X]) f (insert x F)) =
      degree (f x ⊗_K[X] finprod (K[X]) f F)
      using insert by (subst p.finprod-insert, auto)
    also have ... = degree (f x) + degree (finprod (K[X]) f F)
      using insert p.finprod-non-zero[OF insert(1)]
      by (subst degree-mult[OF assms(2)], simp-all)
    also have ... = degree (f x) + (∑ i ∈ F. degree (f i))
      using insert by (subst insert(3), auto)
    also have ... = (∑ i ∈ insert x F. degree (f i))
      using insert by simp
    finally show ?case by simp
  qed
qed

lemma (in ring) coeff-add:
  assumes subring K R
  assumes f ∈ carrier (K[X]) g ∈ carrier (K[X])
  shows coeff (f ⊕_K[X] g) i = coeff f i ⊕_R coeff g i
proof −
  have a:set f ⊆ carrier R
    using assms(1,2) univ-poly-carrier
    using subringE(1)[OF assms(1)] polynomial-incl
    by blast
  have b:set g ⊆ carrier R
    using assms(1,3) univ-poly-carrier
    using subringE(1)[OF assms(1)] polynomial-incl
    by blast
  show ?thesis
    unfolding univ-poly-add poly-add-coeff[OF a b] by simp
qed
```

This is a version of geometric sums for commutative rings:

```
lemma (in cring) geom:
```

fixes q:: *nat*
assumes [*simp*]: $a \in$ *carrier* R
shows $(a \ominus \mathbf{1}) \otimes (\bigoplus i{\in}\{..{<}q\}.\ a\ [\hat{}]\ i) = (a\ [\hat{}]\ q \ominus \mathbf{1})$
(**is** *?lhs = ?rhs*)
proof −
have [*simp*]: $a\ [\hat{}]\ i \in$ *carrier* R **for** i :: *nat*
by (*intro nat-pow-closed assms*)
have [*simp*]: $\ominus \mathbf{1} \otimes x = \ominus x$ **if** $x \in$ *carrier* R **for** x
using *l-minus l-one one-closed that* **by** *presburger*

let *?cterm* = $(\bigoplus i{\in}\{1..{<}q\}.\ a\ [\hat{}]\ i)$

have *?lhs* = $a \otimes (\bigoplus i{\in}\{..{<}q\}.\ a\ [\hat{}]\ i)\ \ominus (\bigoplus i{\in}\{..{<}q\}.\ a\ [\hat{}]\ i)$
unfolding *a-minus-def* **by** (*subst l-distr*, *simp-all add*:*Pi-def*)
also have ... = $(\bigoplus i{\in}\{..{<}q\}.\ a \otimes a\ [\hat{}]\ i) \ominus (\bigoplus i{\in}\{..{<}q\}.\ a\ [\hat{}]\ i)$
by (*subst finsum-rdistr*, *simp-all add*:*Pi-def*)
also have ... = $(\bigoplus i{\in}\{..{<}q\}.\ a\ [\hat{}]\ (Suc\ i)) \ominus (\bigoplus i{\in}\{..{<}q\}.\ a\ [\hat{}]\ i)$
by (*subst nat-pow-Suc*, *simp-all add*:*m-comm*)
also have ... = $(\bigoplus i{\in}Suc\ `\ \{..{<}q\}.\ a\ [\hat{}]\ i) \ominus (\bigoplus i{\in}\{..{<}q\}.\ a\ [\hat{}]\ i)$
by (*subst finsum-reindex*, *simp-all*)
also have ... =
$(\bigoplus i{\in}\ insert\ q\ \{1..{<}q\}.\ a\ [\hat{}]\ i) \ominus$
$(\bigoplus i{\in}\ insert\ 0\ \{1..{<}q\}.\ a\ [\hat{}]\ i)$
proof (*cases* $q > 0$)
case *True*
moreover have $Suc\ `\ \{..{<}q\} = insert\ q\ \{Suc\ 0..{<}q\}$
using *True lessThan-atLeast0* **by** *fastforce*
moreover have $\{..{<}q\} = insert\ 0\ \{Suc\ 0..{<}q\}$
using *True* **by** (*auto simp add*:*set-eq-iff*)
ultimately show *?thesis*
by (*intro arg-cong2*[**where** $f{=}\lambda x\ y.\ x \ominus y$] *finsum-cong*)
simp-all
next
case *False*
then show *?thesis* **by** (*simp*, *algebra*)
qed
also have ... = $(a\ [\hat{}]\ q \oplus ?cterm) \ominus (\mathbf{1} \oplus ?cterm)$
by *simp*
also have ... = $a\ [\hat{}]\ q \oplus ?cterm \oplus (\ominus \mathbf{1} \oplus \ominus ?cterm)$
unfolding *a-minus-def* **by** (*subst minus-add*, *simp-all*)
also have ... = $a\ [\hat{}]\ q \oplus (?cterm \oplus (\ominus \mathbf{1} \oplus \ominus ?cterm))$
by (*subst a-assoc*, *simp-all*)
also have ... = $a\ [\hat{}]\ q \oplus (?cterm \oplus (\ominus ?cterm \oplus \ominus \mathbf{1}))$
by (*subst a-comm*[**where** $x{=}\ominus \mathbf{1}$], *simp-all*)
also have ... = $a\ [\hat{}]\ q \oplus ((?cterm \oplus (\ominus ?cterm)) \oplus \ominus \mathbf{1})$
by (*subst a-assoc*, *simp-all*)
also have ... = $a\ [\hat{}]\ q \oplus (\mathbf{0} \oplus \ominus \mathbf{1})$
by (*subst r-neg*, *simp-all*)
also have ... = $a\ [\hat{}]\ q \ominus \mathbf{1}$

```
    unfolding a-minus-def by simp
  finally show ?thesis by simp
qed

lemma (in domain) rupture-eq-0-iff:
  assumes subfield K R p ∈ carrier (K[X]) q ∈ carrier (K[X])
  shows rupture-surj K p q = 0_(Rupt K p) ⟷ p pdivides q
    (is ?lhs ⟷ ?rhs)
proof −
  interpret h:ring-hom-ring K[X] (Rupt K p) (rupture-surj K p)
    using assms subfieldE by (intro rupture-surj-hom) auto

  have a: q pmod p ∈ (λq. q pmod p) ‘ carrier (K [X])
    using assms(3) by simp
  have 0_K[X] = 0_K[X] pmod p
    using assms(1,2) long-division-zero(2)
    by (simp add:univ-poly-zero)
  hence b: 0_K[X] ∈ (λq. q pmod p) ‘ carrier (K[X])
    by (simp add:image-iff) auto

  have ?lhs ⟷ rupture-surj K p (q pmod p) =
    rupture-surj K p (0_K[X])
    by (subst rupture-surj-composed-with-pmod[OF assms]) simp
  also have ... ⟷ q pmod p = 0_K[X]
    using assms(3)
   by (intro inj-on-eq-iff[OF rupture-surj-inj-on[OF assms(1,2)]] a b)
  also have ... ⟷ ?rhs
    unfolding univ-poly-zero
    by (intro pmod-zero-iff-pdivides[OF assms(1)] assms(2,3))
  finally show ?thesis by simp
qed
```

2.3 Ring Isomorphisms

The following lemma shows that an isomorphism between domains also induces an isomorphism between the corresponding polynomial rings.

```
lemma lift-iso-to-poly-ring:
  assumes h ∈ ring-iso R S domain R domain S
  shows map h ∈ ring-iso (poly-ring R) (poly-ring S)
proof (rule ring-iso-memI)
  interpret dr: domain R using assms(2) by blast
  interpret ds: domain S using assms(3) by blast
  interpret pdr: domain poly-ring R
    using dr.univ-poly-is-domain[OF dr.carrier-is-subring] by simp
  interpret pds: domain poly-ring S
    using ds.univ-poly-is-domain[OF ds.carrier-is-subring] by simp
  interpret h: ring-hom-ring R S h
```

```
  using dr.is-ring ds.is-ring assms(1)
  by (intro ring-hom-ringI2, simp-all add:ring-iso-def)
let ?R = poly-ring R
let ?S = poly-ring S

have h-img: h ‘ (carrier R) = carrier S
  using assms(1) unfolding ring-iso-def bij-betw-def by auto
have h-inj: inj-on h (carrier R)
  using assms(1) unfolding ring-iso-def bij-betw-def by auto
hence h-non-zero-iff:  h x ≠ 0_S
  if x ≠ 0_R x ∈ carrier R for x
  using h.hom-zero dr.zero-closed inj-onD that by metis

have norm-elim: ds.normalize (map h x) = map h x
  if x ∈ carrier (poly-ring R) for x
proof (cases x)
  case Nil then show ?thesis by simp
next
  case (Cons xh xt)
  have xh ∈ carrier R xh ≠ 0_R
    using that unfolding Cons univ-poly-carrier[symmetric]
    unfolding polynomial-def by auto
  hence h xh ≠ 0_S using h-non-zero-iff by simp
  then show ?thesis unfolding Cons by simp
qed

show t-1: map h x ∈ carrier ?S
  if x ∈ carrier ?R for x
  using that hd-in-set h-non-zero-iff hd-map
  unfolding univ-poly-carrier[symmetric] polynomial-def
  by (cases x, auto)

show map h (x ⊗_?R y) = map h x ⊗_?S map h y
  if x ∈ carrier ?R y ∈ carrier ?R for x y
proof −
  have map h (x ⊗_?R y) = ds.normalize (map h (x ⊗_?R y))
    using that by (intro norm-elim[symmetric],simp)
  also have ... = map h x ⊗_?S map h y
   using that unfolding univ-poly-mult univ-poly-carrier[symmetric]

    unfolding polynomial-def
    by (intro h.poly-mult-hom'[of x y] , auto)
  finally show ?thesis by simp
qed

show map h (x ⊕_?R y) = map h x ⊕_?S map h y
  if x ∈ carrier ?R y ∈ carrier ?R for x y
proof −
  have map h (x ⊕_?R y) = ds.normalize (map h (x ⊕_?R y))
```

```
    using that by (intro norm-elim[symmetric],simp)
  also have ... = map h x ⊕_?S map h y
    using that
    unfolding univ-poly-add univ-poly-carrier[symmetric]
    unfolding polynomial-def
    by (intro h.poly-add-hom'[of x y], auto)
  finally show ?thesis by simp
qed

show map h 1_?R = 1_?S
  unfolding univ-poly-one by simp

let ?hinv = map (the-inv-into (carrier R) h)

have map h ∈ carrier ?R → carrier ?S
  using t-1 by simp
moreover have ?hinv x ∈ carrier ?R
  if x ∈ carrier ?S for x
proof (cases x = [])
  case True
  then show ?thesis
    by (simp add:univ-poly-carrier[symmetric] polynomial-def)
next
  case False
  have set-x: set x ⊆ h ` carrier R
    using that h-img unfolding univ-poly-carrier[symmetric]
    unfolding polynomial-def by auto
  have lead-coeff x ≠ 0_S lead-coeff x ∈ carrier S
    using that False unfolding univ-poly-carrier[symmetric]
    unfolding polynomial-def by auto
  hence the-inv-into (carrier R) h (lead-coeff x) ≠
    the-inv-into (carrier R) h 0_S
    using inj-on-the-inv-into[OF h-inj] inj-onD
    using ds.zero-closed h-img by metis
  hence the-inv-into (carrier R) h (lead-coeff x) ≠ 0_R
    unfolding h.hom-zero[symmetric]
    unfolding the-inv-into-f-f[OF h-inj dr.zero-closed] by simp
  hence lead-coeff (?hinv x) ≠ 0_R
    using False by (simp add:hd-map)
  moreover have the-inv-into (carrier R) h ` set x ⊆ carrier R
    using the-inv-into-into[OF h-inj] set-x
    by (intro image-subsetI) auto
  hence set (?hinv x) ⊆ carrier R by simp
  ultimately show ?thesis
    by (simp add:univ-poly-carrier[symmetric] polynomial-def)
qed
moreover have ?hinv (map h x) = x if x ∈ carrier ?R for x
proof −
  have set-x: set x ⊆ carrier R
```

```
    using that unfolding univ-poly-carrier[symmetric]
    unfolding polynomial-def by auto
  have ?hinv (map h x) =
    map (λy. the-inv-into (carrier R) h (h y)) x
    by simp
  also have ... = map id x
    using set-x by (intro map-cong)
      (auto simp add:the-inv-into-f-f[OF h-inj])
  also have ... = x by simp
  finally show ?thesis by simp
 qed
 moreover have map h (?hinv x) = x
  if x ∈ carrier ?S for x
 proof −
  have set-x: set x ⊆ h ‘ carrier R
    using that h-img unfolding univ-poly-carrier[symmetric]
    unfolding polynomial-def by auto
  have map h (?hinv x) =
    map (λy. h (the-inv-into (carrier R) h y)) x
    by simp
  also have ... = map id x
    using set-x by (intro map-cong)
      (auto simp add:f-the-inv-into-f[OF h-inj])
  also have ... = x by simp
  finally show ?thesis by simp
 qed
 ultimately show bij-betw (map h) (carrier ?R) (carrier ?S)
  by (intro bij-betwI[where g=?hinv], auto)
qed

lemma carrier-hom:
 assumes f ∈ carrier (poly-ring R)
 assumes h ∈ ring-iso R S domain R domain S
 shows map h f ∈ carrier (poly-ring S)
proof −
 note poly-iso = lift-iso-to-poly-ring[OF assms(2,3,4)]
 show ?thesis
  using ring-iso-memE(1)[OF poly-iso assms(1)] by simp
qed

lemma carrier-hom′:
 assumes f ∈ carrier (poly-ring R)
 assumes h ∈ ring-hom R S
 assumes domain R domain S
 assumes inj-on h (carrier R)
 shows map h f ∈ carrier (poly-ring S)
proof −
 let ?S = S (| carrier := h ‘ carrier R |)
```

interpret *dr*: *domain R* **using** *assms*(*3*) **by** *blast*
interpret *ds*: *domain S* **using** *assms*(*4*) **by** *blast*
interpret *h1*: *ring-hom-ring R S h*
using *assms*(*2*) *ring-hom-ringI2 dr.ring-axioms*
using *ds.ring-axioms* **by** *blast*
have *subr*: *subring* (*h ‘ carrier R*) *S*
using *h1.img-is-subring*[*OF dr.carrier-is-subring*] **by** *blast*
interpret *h*: *ring-hom-ring* ((*h ‘ carrier R*)[*X*]$_S$) *poly-ring S id*
using *ds.embed-hom*[*OF subr*] **by** *simp*

let *?S* = *S* (| *carrier* := *h ‘ carrier R* |)
have *h* ∈ *ring-hom R ?S*
using *assms*(*2*) **unfolding** *ring-hom-def* **by** *simp*
moreover have *bij-betw h* (*carrier R*) (*carrier ?S*)
using *assms*(*5*) *bij-betw-def* **by** *auto*
ultimately have *h-iso*: *h* ∈ *ring-iso R ?S*
unfolding *ring-iso-def* **by** *simp*

have *dom-S*: *domain ?S*
using *ds.subring-is-domain*[*OF subr*] **by** *simp*

note *poly-iso* = *lift-iso-to-poly-ring*[*OF h-iso assms*(*3*) *dom-S*]
have *map h f* ∈ *carrier* (*poly-ring ?S*)
using *ring-iso-memE*(*1*)[*OF poly-iso assms*(*1*)] **by** *simp*
also have *carrier* (*poly-ring ?S*) =
carrier (*univ-poly S* (*h ‘ carrier R*))
using *ds.univ-poly-consistent*[*OF subr*] **by** *simp*
also have ... ⊆ *carrier* (*poly-ring S*)
using *h.hom-closed* **by** *auto*
finally show *?thesis* **by** *simp*
qed

The following lemmas transfer properties like divisibility, irreducibility etc. between ring isomorphisms.

lemma *divides-hom*:
assumes *h* ∈ *ring-iso R S*
assumes *domain R domain S*
assumes *x* ∈ *carrier R y* ∈ *carrier R*
shows *x divides*$_R$ *y* ⟷ (*h x*) *divides*$_S$ (*h y*) (**is** *?lhs* ⟷ *?rhs*)
proof −
interpret *dr*: *domain R* **using** *assms*(*2*) **by** *blast*
interpret *ds*: *domain S* **using** *assms*(*3*) **by** *blast*
interpret *pdr*: *domain poly-ring R*
using *dr.univ-poly-is-domain*[*OF dr.carrier-is-subring*] **by** *simp*
interpret *pds*: *domain poly-ring S*
using *ds.univ-poly-is-domain*[*OF ds.carrier-is-subring*] **by** *simp*
interpret *h*: *ring-hom-ring R S h*
using *dr.is-ring ds.is-ring assms*(*1*)
by (*intro ring-hom-ringI2*, *simp-all add*:*ring-iso-def*)

```
  have h-inj-on: inj-on h (carrier R)
    using assms(1) unfolding ring-iso-def bij-betw-def by auto
  have h-img: h ‘ (carrier R) = carrier S
    using assms(1) unfolding ring-iso-def bij-betw-def by auto

  have ?lhs ⟷ (∃ c ∈ carrier R. y = x ⊗_R c)
    unfolding factor-def by simp
  also have ... ⟷ (∃ c ∈ carrier R. h y = h x ⊗_S h c)
    using assms(4,5) inj-onD[OF h-inj-on]
    by (intro bex-cong, auto simp flip:h.hom-mult)
  also have ... ⟷ (∃ c ∈ carrier S. h y = h x  ⊗_S c)
    unfolding h-img[symmetric] by simp
  also have ... ⟷ ?rhs
    unfolding factor-def by simp
  finally show ?thesis by simp
qed

lemma properfactor-hom:
  assumes h ∈ ring-iso R S
  assumes domain R domain S
  assumes x ∈ carrier R b ∈ carrier R
  shows properfactor R b x ⟷ properfactor S (h b) (h x)
  using divides-hom[OF assms(1,2,3)] assms(4,5)
  unfolding properfactor-def by simp

lemma Units-hom:
  assumes h ∈ ring-iso R S
  assumes domain R domain S
  assumes x ∈ carrier R
  shows x ∈ Units R ⟷  h x ∈ Units S
proof −

  interpret dr: domain R using assms(2) by blast
  interpret ds: domain S using assms(3) by blast
  interpret pdr: domain poly-ring R
    using dr.univ-poly-is-domain[OF dr.carrier-is-subring] by simp
  interpret pds: domain poly-ring S
    using ds.univ-poly-is-domain[OF ds.carrier-is-subring] by simp
  interpret h: ring-hom-ring R S h
    using dr.is-ring ds.is-ring assms(1)
    by (intro ring-hom-ringI2, simp-all add:ring-iso-def)

  have h-img: h ‘ (carrier R) = carrier S
    using assms(1) unfolding ring-iso-def bij-betw-def by auto

  have h-inj-on: inj-on h (carrier R)
    using assms(1) unfolding ring-iso-def bij-betw-def by auto
```

hence *h-one-iff*: $h\ x = \mathbf{1}_S \longleftrightarrow x = \mathbf{1}_R$ **if** $x \in$ *carrier* R **for** x
using *h.hom-one that* **by** (*metis dr.one-closed inj-onD*)

have $x \in$ *Units* $R \longleftrightarrow$
($\exists\, y \in$*carrier* $R.\ x \otimes_R y = \mathbf{1}_R \wedge y \otimes_R x = \mathbf{1}_R$)
using *assms* **unfolding** *Units-def* **by** *auto*
also have ... $\longleftrightarrow$
($\exists\, y \in$*carrier* $R.\ h\ x \otimes_S h\ y = h\ \mathbf{1}_R \wedge h\ y \otimes_S h\ x = h\ \mathbf{1}_R$)
using *h-one-iff assms* **by** (*intro bex-cong*, *simp-all flip*:*h.hom-mult*)
also have ... $\longleftrightarrow$
($\exists\, y \in$*carrier* $S.\ h\ x \otimes_S y = h\ \mathbf{1}_R \wedge y \otimes_S h\ x = \mathbf{1}_S$)
unfolding *h-img*[*symmetric*] **by** *simp*
also have ... $\longleftrightarrow$ $h\ x \in$ *Units* S
using *assms h.hom-closed* **unfolding** *Units-def* **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *irreducible-hom*:
assumes $h \in$ *ring-iso* $R\ S$
assumes *domain* R *domain* S
assumes $x \in$ *carrier* R
shows *irreducible* $R\ x =$ *irreducible* $S\ (h\ x)$
proof −
have *h-img*: h ‘ (*carrier* R) = *carrier* S
using *assms*(*1*) **unfolding** *ring-iso-def bij-betw-def* **by** *auto*

have *irreducible* $R\ x \longleftrightarrow (x \notin$ *Units* $R \wedge$
($\forall\, b \in$*carrier* $R.$ *properfactor* $R\ b\ x \longrightarrow b \in$ *Units* R))
unfolding *Divisibility.irreducible-def* **by** *simp*
also have ... $\longleftrightarrow (x \notin$ *Units* $R \wedge$
($\forall\, b \in$*carrier* $R.$ *properfactor* $S\ (h\ b)\ (h\ x) \longrightarrow b \in$ *Units* R))
using *properfactor-hom*[*OF assms*(*1*,*2*,*3*)] *assms*(*4*) **by** *simp*
also have ... $\longleftrightarrow (h\ x \notin$ *Units* $S \wedge$
($\forall\, b \in$*carrier* $R.$ *properfactor* $S\ (h\ b)\ (h\ x) \longrightarrow h\ b \in$ *Units* S))
using *assms*(*4*) *Units-hom*[*OF assms*(*1*,*2*,*3*)] **by** *simp*
also have ...$\longleftrightarrow (h\ x \notin$ *Units* $S \wedge$
($\forall\, b \in h$ ‘ *carrier* $R.$ *properfactor* $S\ b\ (h\ x) \longrightarrow b \in$ *Units* S))
by *simp*
also have ... $\longleftrightarrow$ *irreducible* $S\ (h\ x)$
unfolding *h-img Divisibility.irreducible-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *pirreducible-hom*:
assumes $h \in$ *ring-iso* $R\ S$
assumes *domain* R *domain* S
assumes $f \in$ *carrier* (*poly-ring* R)
shows *pirreducible*$_R$ (*carrier* R) $f =$
pirreducible$_S$ (*carrier* S) (*map* $h\ f$)

(**is** *?lhs = ?rhs*)
proof −
note *lift-iso = lift-iso-to-poly-ring*[*OF assms*(*1*,*2*,*3*)]
interpret *dr*: *domain R* **using** *assms*(*2*) **by** *blast*
interpret *ds*: *domain S* **using** *assms*(*3*) **by** *blast*
interpret *pdr*: *domain poly-ring R*
using *dr.univ-poly-is-domain*[*OF dr.carrier-is-subring*] **by** *simp*
interpret *pds*: *domain poly-ring S*
using *ds.univ-poly-is-domain*[*OF ds.carrier-is-subring*] **by** *simp*

have *mh-inj-on*: *inj-on* (*map h*) (*carrier* (*poly-ring R*))
using *lift-iso* **unfolding** *ring-iso-def bij-betw-def* **by** *auto*
moreover have *map h* $\mathbf{0}_{\textit{poly-ring R}} = \mathbf{0}_{\textit{poly-ring S}}$
by (*simp add*:*univ-poly-zero*)
ultimately have *mh-zero-iff*:
map h f = $\mathbf{0}_{\textit{poly-ring S}} \longleftrightarrow$ *f* = $\mathbf{0}_{\textit{poly-ring R}}$
using *assms*(*4*) **by** (*metis pdr.zero-closed inj-onD*)

have *?lhs* $\longleftrightarrow$ (*f* $\neq$ $\mathbf{0}_{\textit{poly-ring R}}$ $\wedge$ *irreducible* (*poly-ring R*) *f*)
unfolding *ring-irreducible-def* **by** *simp*
also have ... $\longleftrightarrow$
(*f* $\neq$ $\mathbf{0}_{\textit{poly-ring R}}$ $\wedge$ *irreducible* (*poly-ring S*) (*map h f*))
using *irreducible-hom*[*OF lift-iso*] *pdr.domain-axioms*
using *assms*(*4*) *pds.domain-axioms* **by** *simp*
also have ... $\longleftrightarrow$
(*map h f* $\neq$ $\mathbf{0}_{\textit{poly-ring S}}$ $\wedge$ *irreducible* (*poly-ring S*) (*map h f*))
using *mh-zero-iff* **by** *simp*
also have ... $\longleftrightarrow$ *?rhs*
unfolding *ring-irreducible-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *ring-hom-cong*:
assumes $\bigwedge x.\ x \in$ *carrier R* $\Longrightarrow$ *f' x* = *f x*
assumes *ring R*
assumes *f* $\in$ *ring-hom R S*
shows *f'* $\in$ *ring-hom R S*
proof −
interpret *ring R* **using** *assms*(*2*) **by** *simp*
show *?thesis*
using *assms*(*1*) *ring-hom-memE*[*OF assms*(*3*)]
by (*intro ring-hom-memI*, *auto*)
qed

The natural homomorphism between factor rings, where one ideal is a subset of the other.

lemma (**in** *ring*) *quot-quot-hom*:
assumes *ideal I R*

assumes *ideal J R*
assumes $I \subseteq J$
shows $(\lambda x.\ (J <+>_R x)) \in$ *ring-hom* $(R\ Quot\ I)\ (R\ Quot\ J)$
proof (*rule ring-hom-memI*)
interpret *ji*: *ideal J R*
using *assms*(*2*) **by** *simp*
interpret *ii*: *ideal I R*
using *assms*(*1*) **by** *simp*

have *a*:$J <+>_R I = J$
using *assms*(*3*) **unfolding** *set-add-def set-mult-def* **by** *auto*

show $J <+>_R x \in$ *carrier* $(R\ Quot\ J)$
if $x \in$ *carrier* $(R\ Quot\ I)$ **for** x
proof −
have $\exists y \in$*carrier R*. $x = I +> y$
using *that* **unfolding** *FactRing-def A-RCOSETS-def'* **by** *simp*
then obtain y **where** *y-def*: $y \in$ *carrier R* $x = I +> y$
by *auto*
have $J <+>_R (I +> y) = (J <+>_R I) +> y$
using *y-def*(*1*) **by** (*subst a-setmult-rcos-assoc*) *auto*
also have ... $= J +> y$ **using** *a* **by** *simp*
finally have $J <+>_R (I +> y) = J +> y$ **by** *simp*
thus *?thesis*
using *y-def* **unfolding** *FactRing-def A-RCOSETS-def'* **by** *auto*
qed

show $J <+>_R x \otimes_{R\ Quot\ I} y =$
$(J <+>_R x) \otimes_{R\ Quot\ J} (J <+>_R y)$
if $x \in$ *carrier* $(R\ Quot\ I)$ $y \in$ *carrier* $(R\ Quot\ I)$
for $x\ y$
proof −
have $\exists x1 \in$*carrier R*. $x = I +> x1$ $\exists y1 \in$*carrier R*. $y = I +> y1$
using *that* **unfolding** *FactRing-def A-RCOSETS-def'* **by** *auto*
then obtain *x1 y1*
where *x1-def*: $x1 \in$ *carrier R* $x = I +> x1$
and *y1-def*: $y1 \in$ *carrier R* $y = I +> y1$
by *auto*
have $J <+>_R x \otimes_{R\ Quot\ I} y = J <+>_R (I +> x1 \otimes y1)$
using *x1-def y1-def*
by (*simp add*: *FactRing-def ii.rcoset-mult-add*)
also have ... $= (J <+>_R I) +> x1 \otimes y1$
using *x1-def*(*1*) *y1-def*(*1*)
by (*subst a-setmult-rcos-assoc*) *auto*
also have ... $= J +> x1 \otimes y1$
using *a* **by** *simp*
also have ... $= [mod\ J:]\ (J +> x1) \bigotimes (J +> y1)$
using *x1-def*(*1*) *y1-def*(*1*) **by** (*subst ji.rcoset-mult-add*, *auto*)
also have ... $=$

```
    [mod J:] ((J <+>_R I) +> x1) ⊗ ((J <+>_R I) +> y1)
    using a by simp
  also have ... =
    [mod J:] (J <+>_R (I +> x1)) ⊗ (J <+>_R (I +> y1))
    using x1-def(1) y1-def(1)
    by (subst (1 2) a-setmult-rcos-assoc) auto
  also have ... = (J <+>_R x) ⊗_R Quot J (J <+>_R y)
    using x1-def y1-def by (simp add: FactRing-def)
  finally show ?thesis by simp
qed

show J <+>_R x ⊕_R Quot I y =
  (J <+>_R x) ⊕_R Quot J (J <+>_R y)
  if x ∈ carrier (R Quot I) y ∈ carrier (R Quot I)
  for x y
proof −
  have ∃x1∈carrier R. x = I +> x1 ∃y1∈carrier R. y = I +> y1
    using that unfolding FactRing-def A-RCOSETS-def' by auto
  then obtain x1 y1
    where x1-def: x1 ∈ carrier R x = I +> x1
      and y1-def: y1 ∈ carrier R y = I +> y1
    by auto
  have J <+>_R x ⊕_R Quot I y =
    J <+>_R ((I +> x1) <+>_R (I +> y1))
    using x1-def y1-def by (simp add:FactRing-def)
  also have ... = J <+>_R (I +> (x1 ⊕ y1))
    using x1-def y1-def ii.a-rcos-sum by simp
  also have ... = (J <+>_R I) +> (x1 ⊕ y1)
    using x1-def y1-def by (subst a-setmult-rcos-assoc) auto
  also have ... = J +> (x1 ⊕ y1)
    using a by simp
  also have ... =
    ((J <+>_R I) +> x1) <+>_R ((J <+>_R I) +> y1)
    using x1-def y1-def ji.a-rcos-sum a by simp
  also have ... =
    J <+>_R (I +> x1) <+>_R (J <+>_R (I +> y1))
    using x1-def y1-def by (subst (1 2) a-setmult-rcos-assoc) auto
  also have ... = (J <+>_R x) ⊕_R Quot J (J <+>_R y)
    using x1-def y1-def by (simp add:FactRing-def)
  finally show ?thesis by simp
qed

have J <+>_R 1_R Quot I = J <+>_R (I +> 1)
  unfolding FactRing-def by simp
also have ... = (J <+>_R I) +> 1
  by (subst a-setmult-rcos-assoc) auto
also have ... = J +> 1 using a by simp
also have ... = 1_R Quot J
  unfolding FactRing-def by simp
```

finally show $J <+>_R \mathbf{1}_{R\ Quot\ I} = \mathbf{1}_{R\ Quot\ J}$
by *simp*
qed

lemma (**in** *ring*) *quot-carr*:
assumes *ideal I R*
assumes $y \in carrier\ (R\ Quot\ I)$
shows $y \subseteq carrier\ R$
proof −
interpret *ideal I R* **using** *assms*(*1*) **by** *simp*
have $y \in a\text{-}rcosets\ I$
using *assms*(*2*) **unfolding** *FactRing-def* **by** *simp*
then obtain *v* **where** *y-def*: $y = I +> v\ v \in carrier\ R$
unfolding *A-RCOSETS-def'* **by** *auto*
have $I +> v \subseteq carrier\ R$
using *y-def*(*2*) *a-r-coset-subset-G a-subset* **by** *presburger*
thus $y \subseteq carrier\ R$ **unfolding** *y-def* **by** *simp*
qed

lemma (**in** *ring*) *set-add-zero*:
assumes $A \subseteq carrier\ R$
shows $\{\mathbf{0}\} <+>_R A = A$
proof −
have $\{\mathbf{0}\} <+>_R A = (\bigcup x{\in}A.\ \{\mathbf{0} \oplus x\})$
using *assms* **unfolding** *set-add-def set-mult-def* **by** *simp*
also have ... $= (\bigcup x{\in}A.\ \{x\})$
using *assms* **by** (*intro arg-cong*[**where** *f*=*Union*] *image-cong*, *auto*)
also have ... $= A$ **by** *simp*
finally show *?thesis* **by** *simp*
qed

Adapted from the proof of *domain.polynomial-rupture*

lemma (**in** *domain*) *rupture-surj-as-eval*:
assumes *subring K R*
assumes $p \in carrier\ (K[X])\ q \in carrier\ (K[X])$
shows *rupture-surj K p q* =
ring.eval (*Rupt K p*) (*map* ((*rupture-surj K p*) ∘ *poly-of-const*) *q*)
(*rupture-surj K p X*)
proof −
let *?surj* = *rupture-surj K p*

interpret *UP*: *domain* $K[X]$
using *univ-poly-is-domain*[*OF assms*(*1*)] **.**
interpret *h*: *ring-hom-ring* $K[X]$ *Rupt K p ?surj*
using *rupture-surj-hom*(*2*)[*OF assms*(*1*,*2*)] **.**

have (*h.S.eval*) (*map* (*?surj* ∘ *poly-of-const*) *q*) (*?surj X*) =
?surj ((*UP.eval*) (*map poly-of-const q*) *X*)
using *h.eval-hom*[*OF UP.carrier-is-subring var-closed*(*1*)[*OF assms*(*1*)]

map-norm-in-poly-ring-carrier[*OF assms*(*1*,*3*)]] **by** *simp*
also have ... = *?surj q*
unfolding *sym*[*OF eval-rewrite*[*OF assms*(*1*,*3*)]] **..**
finally show *?thesis* **by** *simp*
qed

2.4 Divisibility

lemma (**in** *field*) *f-comm-group-1*:
assumes $x \in$ *carrier* R $y \in$ *carrier* R
assumes $x \neq \mathbf{0}$ $y \neq \mathbf{0}$
assumes $x \otimes y = \mathbf{0}$
shows *False*
using *integral assms* **by** *auto*

lemma (**in** *field*) *f-comm-group-2*:
assumes $x \in$ *carrier* R
assumes $x \neq \mathbf{0}$
shows $\exists y \in$ *carrier* $R - \{\mathbf{0}\}.\ y \otimes x = \mathbf{1}$
proof −
have *x-unit*: $x \in$ *Units* R **using** *field-Units assms* **by** *simp*
thus *?thesis* **unfolding** *Units-def* **by** *auto*
qed

sublocale *field* < *mult-of*: *comm-group mult-of R*
rewrites *mult* (*mult-of R*) = *mult R*
and *one* (*mult-of R*) = *one R*
using *f-comm-group-1 f-comm-group-2*
by (*auto intro*!:*comm-groupI m-assoc m-comm*)

lemma (**in** *domain*) *div-neg*:
assumes $a \in$ *carrier* R $b \in$ *carrier* R
assumes *a divides b*
shows *a divides* ($\ominus$ *b*)
proof −
obtain *r1* **where** *r1-def*: *r1* $\in$ *carrier* R $a \otimes$ *r1* = *b*
using *assms* **by** (*auto simp*:*factor-def*)

have $a \otimes (\ominus$ *r1*$) = \ominus (a \otimes$ *r1*$)$
using *assms*(*1*) *r1-def*(*1*) **by** *algebra*
also have ... = $\ominus$ *b*
using *r1-def*(*2*) **by** *simp*
finally have $\ominus b = a \otimes (\ominus$ *r1*$)$ **by** *simp*
moreover have $\ominus$*r1* $\in$ *carrier* R
using *r1-def*(*1*) **by** *simp*
ultimately show *?thesis*
by (*auto simp*:*factor-def*)
qed

lemma (**in** *domain*) *div-sum*:
assumes $a \in carrier\ R\ b \in carrier\ R\ c \in carrier\ R$
assumes *a divides b*
assumes *a divides c*
shows *a divides* $(b \oplus c)$
proof −
obtain *r1* **where** *r1-def*: $r1 \in carrier\ R\ a \otimes r1 = b$
using *assms* **by** (*auto simp*:*factor-def*)

obtain *r2* **where** *r2-def*: $r2 \in carrier\ R\ a \otimes r2 = c$
using *assms* **by** (*auto simp*:*factor-def*)

have $a \otimes (r1 \oplus r2) = (a \otimes r1) \oplus (a \otimes r2)$
using *assms*(*1*) *r1-def*(*1*) *r2-def*(*1*) **by** *algebra*
also have $... = b \oplus c$
using *r1-def*(*2*) *r2-def*(*2*) **by** *simp*
finally have $b \oplus c = a \otimes (r1 \oplus r2)$ **by** *simp*
moreover have $r1 \oplus r2 \in carrier\ R$
using *r1-def*(*1*) *r2-def*(*1*) **by** *simp*
ultimately show *?thesis*
by (*auto simp*:*factor-def*)
qed

lemma (**in** *domain*) *div-sum-iff*:
assumes $a \in carrier\ R\ b \in carrier\ R\ c \in carrier\ R$
assumes *a divides b*
shows *a divides* $(b \oplus c) \longleftrightarrow$ *a divides c*
proof
assume *a divides* $(b \oplus c)$
moreover have *a divides* $(\ominus b)$
using *div-neg assms*(*1*,*2*,*4*) **by** *simp*
ultimately have *a divides* $((b \oplus c) \oplus (\ominus b))$
using *div-sum assms* **by** *simp*
also have $... = c$ **using** *assms*(*1*,*2*,*3*) **by** *algebra*
finally show *a divides c* **by** *simp*
next
assume *a divides c*
thus *a divides* $(b \oplus c)$
using *assms* **by** (*intro div-sum*) *auto*
qed

end

2.5 Factorization

theory *Finite-Fields-Factorization-Ext*
imports *Finite-Fields-Preliminary-Results*
begin

This section contains additional results building on top of the

development in *HOL−Algebra.Divisibility* about factorization in a *factorial-monoid*.

definition *factor-mset* **where** *factor-mset G x* =
(*THE f*. (∃ *as*. *f* = *fmset G as* ∧ *wfactors G as x* ∧ *set as* ⊆ *carrier G*))

In *HOL−Algebra.Divisibility* it is already verified that the multiset representing the factorization of an element of a factorial monoid into irreducible factors is well-defined. With these results it is then possible to define *factor-mset* and show its properties, without referring to a factorization in list form first.

definition *multiplicity* **where**
multiplicity G d g = *Max* {(*n*::*nat*). (*d* [^]$_G$ *n*) *divides*$_G$ *g*}

definition *canonical-irreducibles* **where**
canonical-irreducibles G A = (
A ⊆ {*a*. *a* ∈ *carrier G* ∧ *irreducible G a*} ∧
(∀ *x y*. *x* ∈ *A* ⟶ *y* ∈ *A* ⟶ *x* ∼$_G$ *y* ⟶ *x* = *y*) ∧
(∀ *x* ∈ *carrier G*. *irreducible G x* ⟶ (∃ *y* ∈ *A*. *x* ∼$_G$ *y*)))

A set of irreducible elements that contains exactly one element from each equivalence class of an irreducible element formed by association, is called a set of *canonical-irreducibles*. An example is the set of monic irreducible polynomials as representatives of all irreducible polynomials.

context *factorial-monoid*
begin

lemma *assoc-as-fmset-eq*:
assumes *wfactors G as a*
and *wfactors G bs b*
and *a* ∈ *carrier G*
and *b* ∈ *carrier G*
and *set as* ⊆ *carrier G*
and *set bs* ⊆ *carrier G*
shows *a* ∼ *b* ⟷ (*fmset G as* = *fmset G bs*)
proof −
have *a* ∼ *b* ⟷ (*a divides b* ∧ *b divides a*)
by (*simp add*:*associated-def*)
also have ... ⟷
(*fmset G as* ⊆# *fmset G bs* ∧ *fmset G bs* ⊆# *fmset G as*)
using *divides-as-fmsubset assms* **by** *blast*
also have ... ⟷ (*fmset G as* = *fmset G bs*) **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *factor-mset-aux-1*:

assumes $a \in carrier\ G$ *set as* $\subseteq carrier\ G$ *wfactors* G *as* a
shows *factor-mset* $G\ a = fmset\ G\ as$
proof −
define H **where** $H = \{as.\ wfactors\ G\ as\ a \wedge set\ as \subseteq carrier\ G\}$
have *b*:$as \in H$
using *H-def assms* **by** *simp*

have *c*: $x \in H \Longrightarrow y \in H \Longrightarrow fmset\ G\ x = fmset\ G\ y$ **for** $x\ y$
unfolding *H-def* **using** *assoc-as-fmset-eq*
using *associated-refl assms* **by** *blast*

have *factor-mset* $G\ a = (THE\ f.\ \exists as \in H.\ f = fmset\ G\ as)$
by (*simp add*:*factor-mset-def H-def*, *metis*)

also have ... $= fmset\ G\ as$
using *b c*
by (*intro the1-equality*) *blast*+
finally have *factor-mset* $G\ a = fmset\ G\ as$ **by** *simp*

thus *?thesis*
using *b* **unfolding** *H-def* **by** *auto*
qed

lemma *factor-mset-aux*:
assumes $a \in carrier\ G$
shows $\exists as.\ factor\text{-}mset\ G\ a = fmset\ G\ as \wedge wfactors\ G\ as\ a \wedge$
set as $\subseteq carrier\ G$
proof −
obtain *as* **where** *as-def*: *wfactors* G *as a set as* $\subseteq carrier\ G$
using *wfactors-exist assms* **by** *blast*
thus *?thesis* **using** *factor-mset-aux-1 assms* **by** *blast*
qed

lemma *factor-mset-set*:
assumes $a \in carrier\ G$
assumes $x \in\# factor\text{-}mset\ G\ a$
obtains y **where**
$y \in carrier\ G$
irreducible $G\ y$
assocs $G\ y = x$
proof −
obtain *as* **where** *as-def*:
factor-mset $G\ a = fmset\ G\ as$
wfactors G *as a set as* $\subseteq carrier\ G$
using *factor-mset-aux assms* **by** *blast*
hence $x \in\# fmset\ G\ as$
using *assms* **by** *simp*
hence $x \in assocs\ G$ ‘ *set as*
using *assms as-def* **by** (*simp add*:*fmset-def*)

```
  hence ∃ y. y ∈ set as ∧ x = assocs G y
    by auto
  moreover have y ∈ carrier G ∧ irreducible G y
    if y ∈ set as for y
    using as-def that wfactors-def
    by (simp add: wfactors-def) auto
  ultimately show ?thesis
    using that by blast
qed

lemma factor-mset-mult:
  assumes a ∈ carrier G b ∈ carrier G
  shows factor-mset G (a ⊗ b) = factor-mset G a + factor-mset G b
proof −
  obtain as where as-def:
    factor-mset G a = fmset G as
    wfactors G as a set as ⊆ carrier G
    using factor-mset-aux assms by blast
  obtain bs where bs-def:
    factor-mset G b = fmset G bs
    wfactors G bs b set bs ⊆ carrier G
    using factor-mset-aux assms(2) by blast
  have a ⊗ b ∈ carrier G using assms by auto
  then obtain cs where cs-def:
    factor-mset G (a ⊗ b) = fmset G cs
    wfactors G cs (a ⊗ b)
    set cs ⊆ carrier G
    using factor-mset-aux assms by blast
  have fmset G cs = fmset G as + fmset G bs
    using as-def bs-def cs-def assms
    by (intro  mult-wfactors-fmset[where a=a and b=b]) auto
  thus ?thesis
    using as-def bs-def cs-def by auto
qed

lemma factor-mset-unit: factor-mset G 1 = {#}
proof −
  have factor-mset G 1 = factor-mset G (1 ⊗ 1)
    by simp
  also have ... = factor-mset G 1 + factor-mset G 1
    by (intro factor-mset-mult, auto)
  finally show factor-mset G 1 = {#}
    by simp
qed

lemma factor-mset-irred:
  assumes x ∈ carrier G irreducible G x
  shows factor-mset G x = image-mset (assocs G) {#x#}
proof −
```

```
  have wfactors G [x] x
    using assms by (simp add:wfactors-def)
  hence factor-mset G x = fmset G [x]
    using factor-mset-aux-1 assms by simp
  also have ... = image-mset (assocs G) {#x#}
    by (simp add:fmset-def)
  finally show ?thesis by simp
qed

lemma factor-mset-divides:
  assumes a ∈ carrier G b ∈ carrier G
  shows a divides b ⟷ factor-mset G a ⊆# factor-mset G b
proof −
  obtain as where as-def:
    factor-mset G a = fmset G as
    wfactors G as a set as ⊆ carrier G
    using factor-mset-aux assms by blast
  obtain bs where bs-def:
    factor-mset G b = fmset G bs
    wfactors G bs b set bs ⊆ carrier G
    using factor-mset-aux assms(2) by blast
  hence a divides b ⟷ fmset G as ⊆# fmset G bs
    using as-def bs-def assms
    by (intro divides-as-fmsubset) auto
  also have ... ⟷ factor-mset G a ⊆# factor-mset G b
    using as-def bs-def by simp
  finally show ?thesis by simp
qed

lemma factor-mset-sim:
  assumes a ∈ carrier G b ∈ carrier G
  shows a ∼ b ⟷ factor-mset G a = factor-mset G b
  using factor-mset-divides assms
  by (simp add:associated-def) auto

lemma factor-mset-prod:
  assumes finite A
  assumes f ‘ A ⊆ carrier G
  shows factor-mset G (⨂ a ∈ A. f a) =
    (∑ a ∈ A. factor-mset G (f a))
  using assms
proof (induction A rule:finite-induct)
  case empty
  then show ?case by (simp add:factor-mset-unit)
next
  case (insert x F)
  have factor-mset G (finprod G f (insert x F)) =
    factor-mset G (f x ⊗ finprod G f F)
    using insert by (subst finprod-insert) auto
```

```
  also have ... = factor-mset G (f x) + factor-mset G (finprod G f F)
    using insert by (intro factor-mset-mult finprod-closed) auto
  also have
    ... = factor-mset G (f x) + (∑ a ∈ F. factor-mset G (f a))
    using insert by simp
  also have ... = (∑ a∈insert x F. factor-mset G (f a))
    using insert by simp
  finally show ?case by simp
qed

lemma factor-mset-pow:
  assumes a ∈ carrier G
  shows factor-mset G (a [^] n) = repeat-mset n (factor-mset G a)
proof (induction n)
  case 0
  then show ?case by (simp add:factor-mset-unit)
next
  case (Suc n)
  have factor-mset G (a [^] Suc n) = factor-mset G (a [^] n ⊗ a)
    by simp
  also have ... = factor-mset G (a [^] n) + factor-mset G a
    using assms by (intro factor-mset-mult) auto
  also have ... = repeat-mset n (factor-mset G a) + factor-mset G a
    using Suc by simp
  also have ... = repeat-mset (Suc n) (factor-mset G a)
    by simp
  finally show ?case by simp
qed

lemma image-mset-sum:
  assumes finite F
  shows
    image-mset h (∑ x ∈ F. f x) = (∑ x ∈ F. image-mset h (f x))
  using assms
  by (induction F rule:finite-induct, simp, simp)

lemma decomp-mset:
  (∑ x∈set-mset R. replicate-mset (count R x) x) = R
  by (rule multiset-eqI, simp add:count-sum count-eq-zero-iff)

lemma factor-mset-count:
  assumes a ∈ carrier G d ∈ carrier G irreducible G d
  shows count (factor-mset G a) (assocs G d) = multiplicity G d a
proof −
  have a:
    count (factor-mset G a) (assocs G d) ≥ m ⟷ d [^] m divides a
    (is ?lhs ⟷ ?rhs) for m
  proof −
    have ?lhs ⟷ replicate-mset m (assocs G d) ⊆# factor-mset G a
```

```
    by (simp add:count-le-replicate-mset-subset-eq)
  also have ... ⟷ factor-mset G (d [^] m) ⊆# factor-mset G a
   using assms(2,3) by (simp add:factor-mset-pow factor-mset-irred)
  also have ... ⟷ ?rhs
    using assms(1,2) by (subst factor-mset-divides) auto
  finally show ?thesis by simp
 qed

 define M where M = {(m::nat). d [^] m divides a}

 have M-alt: M = {m. m ≤ count (factor-mset G a) (assocs G d)}
  using a by (simp add:M-def)

 hence Max M = count (factor-mset G a) (assocs G d)
  by (intro Max-eqI, auto)
 thus ?thesis
  unfolding multiplicity-def M-def by auto
qed

lemma multiplicity-ge-iff:
 assumes d ∈ carrier G irreducible G d a ∈ carrier G
 shows multiplicity G d a ≥ k ⟷ d [^] k divides a
  (is ?lhs ⟷ ?rhs)
proof −
 have ?lhs ⟷ count (factor-mset G a) (assocs G d) ≥ k
  using factor-mset-count[OF assms(3,1,2)] by simp
 also have ... ⟷ replicate-mset k (assocs G d) ⊆# factor-mset G a
  by (subst count-le-replicate-mset-subset-eq, simp)
 also have ... ⟷
  repeat-mset k (factor-mset G d) ⊆# factor-mset G a
  by (subst factor-mset-irred[OF assms(1,2)], simp)
 also have ... ⟷ factor-mset G (d [^]_G k) ⊆# factor-mset G a
  by (subst factor-mset-pow[OF assms(1)], simp)
 also have ... ⟷ (d [^] k) divides_G a
  using assms(1) factor-mset-divides[OF - assms(3)] by simp
 finally show ?thesis by simp
qed

lemma multiplicity-gt-0-iff:
 assumes d ∈ carrier G irreducible G d a ∈ carrier G
 shows multiplicity G d a > 0 ⟷ d divides a
 using multiplicity-ge-iff[OF assms(1,2,3), where k=1] assms
 by auto

lemma factor-mset-count-2:
 assumes a ∈ carrier G
 assumes ⋀z. z ∈ carrier G ⟹ irreducible G z ⟹ y ≠ assocs G z
 shows count (factor-mset G a) y = 0
 using factor-mset-set [OF assms(1)] assms(2) by (metis count-inI)
```

lemma *factor-mset-choose*:
assumes $a \in$ *carrier* G *set-mset* $R \subseteq$ *carrier* G
assumes *image-mset* (*assocs* G) $R =$ *factor-mset* G a
shows $a \sim (\bigotimes x \in$ *set-mset* R. $x\ [\widehat{}]$ *count* R $x)$ (**is** $a \sim$ *?rhs*)
proof −
have *b*:*irreducible* G x **if** *a*:$x \in\#$ R **for** x
proof −
have *x-carr*: $x \in$ *carrier* G
using a *assms*(*2*) **by** *auto*
have *assocs* G $x \in$ *assocs* G ‘ *set-mset* R
using a **by** *simp*
hence *assocs* G $x \in\#$ *factor-mset* G a
using *assms*(*3*) a *in-image-mset* **by** *metis*
then obtain z **where** *z-def*:
$z \in$ *carrier* G *irreducible* G z *assocs* G $x =$ *assocs* G z
using *factor-mset-set assms*(*1*) **by** *metis*
have $z \sim x$ **using** *z-def*(*1*,*3*) *assocs-eqD x-carr* **by** *simp*
thus *?thesis* **using** *z-def*(*1*,*2*) *x-carr irreducible-cong* **by** *simp*
qed

have *factor-mset* G *?rhs* =
$(\sum x \in$ *set-mset* R. *factor-mset* G $(x\ [\widehat{}]$ *count* R $x))$
using *assms*(*2*) **by** (*subst factor-mset-prod*, *auto*)
also have ... =
$(\sum x \in$ *set-mset* R. *repeat-mset* (*count* R x) (*factor-mset* G x))
using *assms*(*2*) **by** (*intro sum.cong*, *auto simp add*:*factor-mset-pow*)
also have ... = $(\sum x \in$ *set-mset* R.
repeat-mset (*count* R x) (*image-mset* (*assocs* G) $\{\#x\#\}$))
using *assms*(*2*) b **by** (*intro sum.cong*, *auto simp add*:*factor-mset-irred*)
also have ... = $(\sum x \in$ *set-mset* R.
image-mset (*assocs* G) (*replicate-mset* (*count* R x) x))
by *simp*
also have ... = *image-mset* (*assocs* G)
$(\sum x \in$ *set-mset* R. (*replicate-mset* (*count* R x) x))
by (*simp add*: *image-mset-sum*)
also have ... = *image-mset* (*assocs* G) R
by (*simp add*:*decomp-mset*)
also have ... = *factor-mset* G a
using *assms* **by** *simp*
finally have *factor-mset* G *?rhs* = *factor-mset* G a **by** *simp*
moreover have $(\bigotimes x \in$ *set-mset* R. $x\ [\widehat{}]$ *count* R $x) \in$ *carrier* G
using *assms*(*2*) **by** (*intro finprod-closed*, *auto*)
ultimately show *?thesis*
using *assms*(*1*) **by** (*subst factor-mset-sim*) *auto*
qed

lemma *divides-iff-mult-mono*:
assumes $a \in$ *carrier* G $b \in$ *carrier* G

assumes *canonical-irreducibles G R*
assumes $\bigwedge d.\ d \in R \Longrightarrow$ *multiplicity G d a* $\leq$ *multiplicity G d b*
shows *a divides b*
proof −
have *count* (*factor-mset G a*) *d* $\leq$ *count* (*factor-mset G b*) *d* **for** *d*
proof (*cases* $\exists\, y \in$ *carrier G. irreducible G y* $\wedge$ *d* = *assocs G y*)
case *True*
then obtain *y* **where** *y-def*:
irreducible G y y $\in$ *carrier G d* = *assocs G y*
by *blast*
then obtain *z* **where** *z-def*: *z* $\in$ *R y* $\sim$ *z*
using *assms*(*3*) **unfolding** *canonical-irreducibles-def* **by** *metis*
have *z-more*: *irreducible G z z* $\in$ *carrier G*
using *z-def*(*1*) *assms*(*3*)
unfolding *canonical-irreducibles-def* **by** *auto*
have *y* $\in$ *assocs G z* **using** *z-def*(*2*) *z-more*(*2*) *y-def*(*2*)
by (*simp add*: *closure-ofI2*)
hence *d-def*: *d* = *assocs G z*
using *y-def*(*2*,*3*) *z-more*(*2*) *assocs-repr-independence*
by *blast*
have *count* (*factor-mset G a*) *d* = *multiplicity G z a*
unfolding *d-def*
by (*intro factor-mset-count*[*OF assms*(*1*) *z-more*(*2*,*1*)])
also have ... $\leq$ *multiplicity G z b*
using *assms*(*4*) *z-def*(*1*) **by** *simp*
also have ... = *count* (*factor-mset G b*) *d*
unfolding *d-def*
by (*intro factor-mset-count*[*symmetric*, *OF assms*(*2*) *z-more*(*2*,*1*)])
finally show *?thesis* **by** *simp*
next
case *False*
have *count* (*factor-mset G a*) *d* = *0* **using** *False*
by (*intro factor-mset-count-2*[*OF assms*(*1*)], *simp*)
moreover have *count* (*factor-mset G b*) *d* = *0* **using** *False*
by (*intro factor-mset-count-2*[*OF assms*(*2*)], *simp*)
ultimately show *?thesis* **by** *simp*
qed

hence *factor-mset G a* $\subseteq\#$ *factor-mset G b*
unfolding *subseteq-mset-def* **by** *simp*
thus *?thesis* **using** *factor-mset-divides assms*(*1*,*2*) **by** *simp*
qed

lemma *count-image-mset-inj*:
assumes *inj-on f R x* $\in$ *R set-mset A* $\subseteq$ *R*
shows *count* (*image-mset f A*) (*f x*) = *count A x*
proof (*cases x* $\in\#$ *A*)
case *True*
hence (*f y* = *f x* $\wedge$ *y* $\in\#$ *A*) = (*y* = *x*) **for** *y*

by (*meson assms*(*1*) *assms*(*3*) *inj-onD subsetD*)
hence (*f* −‘ {*f x*} ∩ *set-mset A*) = {*x*}
by (*simp add*:*set-eq-iff*)
thus *?thesis*
by (*subst count-image-mset*, *simp*)
next
case *False*
hence *x* ∉ *set-mset A* **by** *simp*
hence *f x* ∉ *f* ‘ *set-mset A* **using** *assms*
by (*simp add*: *inj-on-image-mem-iff*)
hence *count* (*image-mset f A*) (*f x*) = *0*
by (*simp add*:*count-eq-zero-iff*)
thus *?thesis* **by** (*metis count-inI False*)
qed

Factorization of an element from a *factorial-monoid* using a selection of representatives from each equivalence class formed by (∼).

lemma *split-factors*:
assumes *canonical-irreducibles G R*
assumes *a* ∈ *carrier G*
shows
finite {*d*. *d* ∈ *R* ∧ *multiplicity G d a* > *0*}
a ∼ (⨂ *d*∈{*d*. *d* ∈ *R* ∧ *multiplicity G d a* > *0*}.
d [^] *multiplicity G d a*) (**is** *a* ∼ *?rhs*)
proof −
have *r-1*: *R* ⊆ {*x*. *x* ∈ *carrier G* ∧ *irreducible G x*}
using *assms*(*1*) **unfolding** *canonical-irreducibles-def* **by** *simp*
have *r-2*: ⋀*x y*. *x* ∈ *R* ⟹ *y* ∈ *R* ⟹ *x* ∼ *y* ⟹ *x* = *y*
using *assms*(*1*) **unfolding** *canonical-irreducibles-def* **by** *simp*

have *assocs-inj*: *inj-on* (*assocs G*) *R*
using *r-1 r-2 assocs-eqD* **by** (*intro inj-onI*, *blast*)

define *R*′ **where**
R′ = (∑ *d*∈ {*d*. *d* ∈ *R* ∧ *multiplicity G d a* > *0*}.
replicate-mset (*multiplicity G d a*) *d*)

have *count* (*factor-mset G a*) (*assocs G x*) > *0*
if *x* ∈ *R* *0* < *multiplicity G x a* **for** *x*
using *assms r-1 r-2 that*
by (*subst factor-mset-count*[*OF assms*(*2*)]) *auto*
hence *assocs G* ‘ {*d* ∈ *R*. *0* < *multiplicity G d a*}
⊆ *set-mset* (*factor-mset G a*)
by (*intro image-subsetI*, *simp*)
hence *a*:*finite* (*assocs G* ‘ {*d* ∈ *R*. *0* < *multiplicity G d a*})
using *finite-subset* **by** *auto*

show *finite* {*d* ∈ *R*. *0* < *multiplicity G d a*}

```
  using assocs-inj inj-on-subset[OF assocs-inj]
  by (intro finite-imageD[OF a], simp)

hence count-R':
  count R' d = (if d ∈ R then multiplicity G d a else 0)
  for d
  by (auto simp add:R'-def count-sum)

have set-R': set-mset R' = {d ∈ R. 0 < multiplicity G d a}
  unfolding set-mset-def using count-R' by auto

have count (image-mset (assocs G) R') x =
  count (factor-mset G a) x for x
proof (cases ∃ x'. x' ∈ R ∧ x = assocs G x')
  case True
  then obtain x' where x'-def: x' ∈ R x = assocs G x'
    by blast
  have count (image-mset (assocs G) R') x = count R' x'
    using assocs-inj inj-on-subset[OF assocs-inj] x'-def
    by (subst x'-def(2), subst count-image-mset-inj[OF assocs-inj])
      (auto simp:set-R')
  also have ... = multiplicity G x' a
    using count-R' x'-def by simp
  also have ... = count (factor-mset G a) (assocs G x')
    using x'-def(1) r-1
    by (subst factor-mset-count[OF assms(2)]) auto
  also have ... = count (factor-mset G a) x
    using x'-def(2) by simp
  finally show ?thesis by simp
next
  case False
  have a:x ≠ assocs G z
    if a1: z ∈ carrier G and a2: irreducible G z for z
  proof −
    obtain v where v-def: v ∈ R z ∼ v
      using a1 a2 assms(1)
      unfolding canonical-irreducibles-def by auto
    hence z ∈ assocs G v
      using a1 r-1 v-def(1) by (simp add: closure-ofI2)
    hence assocs G z = assocs G v
      using a1 r-1 v-def(1)  assocs-repr-independence
      by auto
    moreover have x ≠ assocs G v
      using False v-def(1) by simp
    ultimately show ?thesis by simp
  qed

  have count (image-mset (assocs G) R') x = 0
    using False count-R' by (simp add: count-image-mset) auto
```

```
    also have ... = count (factor-mset G a) x
      using a
      by (intro factor-mset-count-2[OF assms(2), symmetric]) auto
    finally show ?thesis by simp
  qed

  hence image-mset (assocs G) R' = factor-mset G a
    by (rule multiset-eqI)

  moreover have set-mset R' ⊆ carrier G
    using r-1 by (auto simp add:set-R')
  ultimately have a ∼ (⨂ x∈set-mset R'. x [^] count R' x)
    using assms(2) by (intro factor-mset-choose, auto)
  also have ... = ?rhs
    using set-R' assms r-1 r-2
    by (intro finprod-cong', auto simp add:count-R')
  finally show a ∼ ?rhs by simp
qed

end

end
```

3 Characteristic of Rings

```
theory Ring-Characteristic
  imports
    Finite-Fields-Factorization-Ext
    HOL−Algebra.IntRing
    HOL−Algebra.Embedded-Algebras
begin

locale finite-field = field +
  assumes finite-carrier: finite (carrier R)
begin

lemma finite-field-min-order:
  order R > 1
proof (rule ccontr)
  assume a:¬(1 < order R)
  have {0_R,1_R} ⊆ carrier R by auto
  hence card {0_R,1_R} ≤ card (carrier R)
    using card-mono finite-carrier by blast
  also have ... ≤ 1 using a by (simp add:order-def)
  finally have card {0_R,1_R} ≤ 1 by blast
  thus False by simp
qed

lemma (in finite-field) order-pow-eq-self:
```

```
  assumes x ∈ carrier R
  shows x [^] (order R) = x
proof (cases x = 0)
  case True
  have order R > 0
    using assms(1) order-gt-0-iff-finite finite-carrier by simp
  then obtain n where n-def:order R = Suc n
    using lessE by blast
  have x [^] (order R) = 0
    unfolding n-def using True by (subst nat-pow-Suc, simp)
  thus ?thesis using True by simp
next
  case False
  have x-carr:x ∈ carrier (mult-of R)
    using False assms by simp

  have carr-non-empty: card (carrier R) > 0
    using order-gt-0-iff-finite finite-carrier
    unfolding order-def by simp
  have x [^] (order R) = x [^]_(mult-of R) (order R)
    by (simp add:nat-pow-mult-of)
  also have ... = x [^]_(mult-of R) (order (mult-of R)+1)
    using carr-non-empty unfolding order-def
    by (intro arg-cong[where f=λt. x [^]_(mult-of R) t]) (simp)
  also have ... = x
    using x-carr
    by (simp add:mult-of.pow-order-eq-1)
  finally show x [^] (order R) = x
    by simp
qed

lemma (in finite-field) order-pow-eq-self':
  assumes x ∈ carrier R
  shows x [^] (order R ^ d) = x
proof (induction d)
  case 0
  then show ?case using assms by simp
next
  case (Suc d)
  have x [^] order R ^ (Suc d) = x [^] (order R ^ d * order R)
    by (simp add:mult.commute)
  also have ... = (x [^] (order R ^ d)) [^] order R
    using assms by (simp add: nat-pow-pow)
  also have ... = (x [^] (order R ^ d))
    using order-pow-eq-self assms by simp
  also have ... = x
    using Suc by simp
  finally show ?case by simp
qed
```

end

lemma *finite-fieldI*:
assumes *field R*
assumes *finite* (*carrier R*)
shows *finite-field R*
using *assms*
unfolding *finite-field-def finite-field-axioms-def*
by *auto*

lemma (**in** *domain*) *finite-domain-units*:
assumes *finite* (*carrier R*)
shows *Units R* = *carrier R* − {**0**} (**is** *?lhs* = *?rhs*)
proof
have *Units R* ⊆ *carrier R* **by** (*simp add*:*Units-def*)
moreover have **0** ∉ *Units R*
by (*meson zero-is-prime*(*1*) *primeE*)
ultimately show *Units R* ⊆ *carrier R* − {**0**} **by** *blast*
next
have *x* ∈ *Units R* **if** *a*: *x* ∈ *carrier R* − {**0**} **for** *x*
proof −
have *x-carr*: *x* ∈ *carrier R* **using** *a* **by** *blast*
define *f* **where** *f* = (λ*y*. *y* ⊗$_R$ *x*)
have *inj-on f* (*carrier R*) **unfolding** *f-def*
by (*rule inj-onI*, *metis DiffD1 DiffD2 a m-rcancel insertI1*)
hence *card* (*carrier R*) = *card* (*f* ‘ *carrier R*)
by (*metis card-image*)
moreover have *f* ‘ *carrier R* ⊆ *carrier R* **unfolding** *f-def*
by (*rule image-subsetI*, *simp add*: *ring.ring-simprules x-carr*)
ultimately have *f* ‘ *carrier R* = *carrier R*
using *card-subset-eq assms* **by** *metis*
moreover have **1**$_R$ ∈ *carrier R* **by** *simp*
ultimately have ∃ *y* ∈ *carrier R*. *f y* = **1**$_R$
by (*metis image-iff*)
then obtain *y*
where *y-carrier*: *y* ∈ *carrier R*
and *y-left-inv*: *y* ⊗$_R$ *x* = **1**$_R$
using *f-def* **by** *blast*
hence *y-right-inv*: *x* ⊗$_R$ *y* = **1**$_R$
by (*metis DiffD1 a cring-simprules*(*14*))
show *x* ∈ *Units R*
using *y-carrier y-left-inv y-right-inv*
by (*metis DiffD1 a divides-one factor-def*)
qed
thus *?rhs* ⊆ *?lhs* **by** *auto*
qed

The following theorem can be found in Lidl and Niederreiter [4,

Theorem 1.31].

theorem *finite-domains-are-fields*:
assumes *domain R*
assumes *finite* (*carrier R*)
shows *finite-field R*
proof −
interpret *domain R* **using** *assms* **by** *auto*
have *Units R* = *carrier R* − {$\mathbf{0}_R$}
using *finite-domain-units*[*OF assms*(*2*)] **by** *simp*
then have *field R*
by (*simp add*: *assms*(*1*) *field.intro field-axioms.intro*)
thus *?thesis*
using *assms*(*2*) *finite-fieldI* **by** *auto*
qed

definition *zfact-iso* :: *nat* ⇒ *nat* ⇒ *int set* **where**
zfact-iso p k = $Idl_{\mathcal{Z}}$ {*int p*} +>$_{\mathcal{Z}}$ (*int k*)

context
fixes *n* :: *nat*
assumes *n-gt-0*: *n* > *0*
begin

private abbreviation *I* **where** *I* ≡ $Idl_{\mathcal{Z}}$ {*int n*}

private lemma *ideal-I*: *ideal I* $\mathcal{Z}$
by (*simp add*: *int.genideal-ideal*)

lemma *int-cosetI*:
assumes *u mod* (*int n*) = *v mod* (*int n*)
shows $Idl_{\mathcal{Z}}$ {*int n*} +>$_{\mathcal{Z}}$ *u* = $Idl_{\mathcal{Z}}$ {*int n*} +>$_{\mathcal{Z}}$ *v*
proof −
have *u* − *v* ∈ *I*
by (*metis Idl-subset-eq-dvd assms int-Idl-subset-ideal mod-eq-dvd-iff*)
thus *?thesis*
using *ideal-I int.quotient-eq-iff-same-a-r-cos* **by** *simp*
qed

lemma *zfact-iso-inj*:
inj-on (*zfact-iso n*) {..<*n*}
proof (*rule inj-onI*)
fix *x y*
assume *a*:*x* ∈ {..<*n*} *y* ∈ {..<*n*}
assume *zfact-iso n x* = *zfact-iso n y*
hence *I* +>$_{\mathcal{Z}}$ (*int x*) = *I* +>$_{\mathcal{Z}}$ (*int y*)
by (*simp add*:*zfact-iso-def*)
hence *int x* − *int y* ∈ *I*
by (*subst int.quotient-eq-iff-same-a-r-cos*[*OF ideal-I*], *auto*)
hence *int x mod int n* = *int y mod int n*

```
    by (meson Idl-subset-eq-dvd int-Idl-subset-ideal mod-eq-dvd-iff)
  thus x = y
    using a by simp
qed

lemma zfact-iso-ran:
  zfact-iso n ` {..<n} = carrier (ZFact (int n))
proof −
  have zfact-iso n ` {..<n} ⊆ carrier (ZFact (int n))
    unfolding zfact-iso-def ZFact-def FactRing-simps
    using int.a-rcosetsI by auto
  moreover have x ∈ zfact-iso n ` {..<n}
    if a:x ∈ carrier (ZFact (int n)) for x
  proof −
    obtain y where y-def: x = I +>_ℤ y
      using a unfolding ZFact-def FactRing-simps by auto
    obtain z where z-def: (int z) mod (int n) = y mod (int n) z < n
      by (metis Euclidean-Division.pos-mod-sign mod-mod-trivial
          nonneg-int-cases of-nat-0-less-iff of-nat-mod n-gt-0
          unique-euclidean-semiring-numeral-class.pos-mod-bound)
    have x = I +>_ℤ y
      by (simp add:y-def)
    also have ... = I +>_ℤ (int z)
      by (intro int-cosetI, simp add:z-def)
    also have ... = zfact-iso n z
      by (simp add:zfact-iso-def)
    finally have x = zfact-iso n z
      by simp
    thus x ∈ zfact-iso n ` {..<n}
      using z-def(2) by blast
  qed
  ultimately show ?thesis by auto
qed

lemma zfact-iso-bij:
  bij-betw (zfact-iso n) {..<n} (carrier (ZFact (int n)))
  using  bij-betw-def zfact-iso-inj zfact-iso-ran by blast

lemma card-zfact-carr: card (carrier (ZFact (int n))) = n
  using bij-betw-same-card[OF zfact-iso-bij] by simp

lemma fin-zfact: finite (carrier (ZFact (int n)))
  using card-zfact-carr n-gt-0 card-ge-0-finite by force

end

lemma zfact-prime-is-finite-field:
  assumes Factorial-Ring.prime p
  shows finite-field (ZFact (int p))
```

proof −
have *p-gt-0*: $p > 0$ **using** *assms*(*1*) *prime-gt-0-nat* **by** *simp*
have *Factorial-Ring.prime* (*int p*)
using *assms* **by** *simp*
moreover have *finite* (*carrier* (*ZFact* (*int p*)))
using *fin-zfact*[*OF p-gt-0*] **by** *simp*
ultimately show *?thesis*
by (*intro finite-domains-are-fields ZFact-prime-is-domain*, *auto*)
qed

definition *int-embed* :: - ⇒ *int* ⇒ - **where**
int-embed R k = *add-pow R k* $\mathbf{1}_R$

lemma (**in** *ring*) *add-pow-consistent*:
fixes *i* :: *int*
assumes *subring K R*
assumes $k \in K$
shows *add-pow R i k* = *add-pow* (*R* (| *carrier* := *K* |)) *i k*
(**is** *?lhs* = *?rhs*)
proof −
have *a*:*subgroup K* (*add-monoid R*)
using *assms*(*1*) *subring.axioms* **by** *auto*
have *add-pow R i k* = k [^]$_{add\text{-}monoid\ R(|carrier := K|)}$ i
using *add.int-pow-consistent*[*OF a assms*(*2*)] **by** *simp*
also have ... = *?rhs*
unfolding *add-pow-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma (**in** *ring*) *int-embed-consistent*:
assumes *subring K R*
shows *int-embed R i* = *int-embed* (*R* (| *carrier* := *K* |)) *i*
proof −
have *a*:$\mathbf{1}$ = $\mathbf{1}_{R\ (|\ carrier := K\ |)}$ **by** *simp*
have *b*:$\mathbf{1}_{R(|carrier := K|)} \in K$
using *assms subringE*(*3*) **by** *auto*
show *?thesis*
unfolding *int-embed-def a* **using** *b add-pow-consistent*[*OF assms*(*1*)]
by *simp*
qed

lemma (**in** *ring*) *int-embed-closed*:
int-embed R k ∈ *carrier R*
unfolding *int-embed-def* **using** *add.int-pow-closed* **by** *simp*

lemma (**in** *ring*) *int-embed-range*:
assumes *subring K R*
shows *int-embed R k* ∈ *K*
proof −

let *?R′ = R* (| *carrier* := *K* |)
interpret *x*:*ring ?R′*
using *subring-is-ring*[*OF assms*] **by** *simp*
have *int-embed R k* = *int-embed ?R′ k*
using *int-embed-consistent*[*OF assms*] **by** *simp*
also have ... ∈ *K*
using *x.int-embed-closed* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma (**in** *ring*) *int-embed-zero*:
int-embed R 0 = $\mathbf{0}_R$
by (*simp add*:*int-embed-def add-pow-def*)

lemma (**in** *ring*) *int-embed-one*:
int-embed R 1 = $\mathbf{1}_R$
by (*simp add*:*int-embed-def*)

lemma (**in** *ring*) *int-embed-add*:
int-embed R (*x*+*y*) = *int-embed R x* $\oplus_R$ *int-embed R y*
by (*simp add*:*int-embed-def add.int-pow-mult*)

lemma (**in** *ring*) *int-embed-inv*:
int-embed R (−*x*) = $\ominus_R$ *int-embed R x* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *int-embed R* (−*x*) ⊕ (*int-embed R x* ⊖ *int-embed R x*)
using *int-embed-closed* **by** *simp*
also have
... = *int-embed R* (−*x*) ⊕ *int-embed R x* ⊕ (⊖ *int-embed R x*)
using *int-embed-closed* **by** (*subst a-minus-def*, *subst a-assoc*, *auto*)
also have ... = *int-embed R* (−*x* +*x*) ⊕ (⊖ *int-embed R x*)
by (*subst int-embed-add*, *simp*)
also have ... = *?rhs*
using *int-embed-closed*
by (*simp add*:*int-embed-zero*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *ring*) *int-embed-diff*:
int-embed R (*x*−*y*) = *int-embed R x* $\ominus_R$ *int-embed R y*
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *int-embed R* (*x* + (−*y*)) **by** *simp*
also have ... = *?rhs*
by (*subst int-embed-add*, *simp add*:*a-minus-def int-embed-inv*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *ring*) *int-embed-mult-aux*:

int-embed R (x∗int y) = int-embed R x ⊗ int-embed R y
proof (*induction y*)
case *0*
then show *?case* **by** (*simp add:int-embed-closed int-embed-zero*)
next
case (*Suc y*)
have *int-embed R (x ∗ int (Suc y)) = int-embed R (x + x ∗ int y)*
by (*simp add:algebra-simps*)
also have *... = int-embed R x ⊕ int-embed R (x ∗ int y)*
by (*subst int-embed-add*, *simp*)
also have
... = int-embed R x ⊗ 1 ⊕ int-embed R x ⊗ int-embed R y
using *int-embed-closed*
by (*subst Suc*, *simp*)
also have *... = int-embed R x ⊗ (int-embed R 1 ⊕ int-embed R y)*
using *int-embed-closed* **by** (*subst r-distr*, *simp-all add:int-embed-one*)
also have *... = int-embed R x ⊗ int-embed R (1+int y)*
by (*subst int-embed-add*, *simp*)
also have *... = int-embed R x ⊗ int-embed R (Suc y)*
by *simp*
finally show *?case* **by** *simp*
qed

lemma (**in** *ring*) *int-embed-mult*:
int-embed R (x∗y) = int-embed R x ⊗$_R$ int-embed R y
proof (*cases y ≥ 0*)
case *True*
then obtain *y′* **where** *y-def*: *y = int y′*
using *nonneg-int-cases* **by** *auto*
have *int-embed R (x ∗ y) = int-embed R (x ∗ int y′)*
unfolding *y-def* **by** *simp*
also have *... = int-embed R x ⊗ int-embed R y′*
by (*subst int-embed-mult-aux*, *simp*)
also have *... = int-embed R x ⊗ int-embed R y*
unfolding *y-def* **by** *simp*
finally show *?thesis* **by** *simp*
next
case *False*
then obtain *y′* **where** *y-def*: *y = − int y′*
by (*meson nle-le nonpos-int-cases*)
have *int-embed R (x ∗ y) = int-embed R (−(x ∗ int y′))*
unfolding *y-def* **by** *simp*
also have *... = ⊖ (int-embed R (x ∗ int y′))*
by (*subst int-embed-inv*, *simp*)
also have *... = ⊖ (int-embed R x ⊗ int-embed R y′)*
by (*subst int-embed-mult-aux*, *simp*)
also have *... = int-embed R x ⊗ ⊖ int-embed R y′*
using *int-embed-closed* **by** *algebra*
also have *... = int-embed R x ⊗ int-embed R (−y′)*

```
    by (subst int-embed-inv, simp)
  also have ... = int-embed R x ⊗ int-embed R y
    unfolding y-def by simp
  finally show ?thesis by simp
qed

lemma (in ring) int-embed-ring-hom:
  ring-hom-ring int-ring R (int-embed R)
proof (rule ring-hom-ringI)
  show ring int-ring using int.is-ring by simp
  show ring R using ring-axioms by simp
  show int-embed R x ∈ carrier R if x ∈ carrier ℤ for x
    using int-embed-closed by simp
  show int-embed R (x⊗ℤy) = int-embed R x ⊗ int-embed R y
    if x ∈ carrier ℤ y ∈ carrier ℤ for x y
    using int-embed-mult by simp
  show int-embed R (x⊕ℤy) = int-embed R x ⊕ int-embed R y
    if x ∈ carrier ℤ y ∈ carrier ℤ for x y
    using int-embed-add by simp
  show int-embed R 1ℤ = 1
    by (simp add:int-embed-one)
qed

abbreviation char-subring where
  char-subring R ≡ int-embed R ‘ UNIV

definition char where
  char R = card (char-subring R)
```

This is a non-standard definition for the characteristic of a ring. Commonly [4, Definition 1.43] it is defined to be the smallest natural number n such that n-times repeated addition of any number is zero. If no such number exists then it is defined to be 0. In the case of rings with unit elements — not that the locale *Ring.ring* requires unit elements — the above definition can be simplified to the number of times the unit elements needs to be repeatedly added to reach 0.

The following three lemmas imply that the definition of the characteristic here coincides with the latter definition.

```
lemma (in ring) char-bound:
  assumes x > 0
  assumes int-embed R (int x) = 0
  shows char R ≤ x char R > 0
proof −
  have char-subring R ⊆ int-embed R ‘ ({0..<int x})
  proof (rule image-subsetI)
    fix y :: int
    assume y ∈ UNIV
```

```
    define u where u = y div (int x)
    define v where v = y mod (int x)
    have int x > 0 using assms by simp
    hence y-exp: y = u * int x + v v ≥ 0 v < int x
      unfolding u-def v-def by simp-all
    have int-embed R y = int-embed R v
      using int-embed-closed unfolding y-exp
      by (simp add:int-embed-mult int-embed-add assms(2))
    also have ... ∈ int-embed R ‘ ({0..<int x})
      using y-exp(2,3) by simp
    finally show int-embed R y ∈ int-embed R ‘ {0..<int x}
      by simp
  qed
  hence a:char-subring R = int-embed R ‘ {0..<int x}
    by auto
  hence char R = card (int-embed R ‘ ({0..<int x}))
    unfolding char-def a by simp
  also have ... ≤ card {0..<int x}
    by (intro card-image-le, simp)
  also have ... = x by simp
  finally show char R ≤ x by simp
  have 1 = card {int-embed R 0} by simp
  also have ... ≤ card (int-embed R ‘ {0..<int x})
    using assms(1) by (intro card-mono finite-imageI, simp-all)
  also have ... = char R
    unfolding char-def a by simp
  finally show char R > 0 by simp
qed

lemma (in ring) embed-char-eq-0:
  int-embed R (int (char R)) = 0
proof (cases finite (char-subring R))
  case True
  interpret h: ring-hom-ring int-ring R (int-embed R)
    using int-embed-ring-hom by simp

  define A where A = {0..int (char R)}
  have card (int-embed R ‘ A) ≤ card (char-subring R)
    by (intro card-mono[OF True] image-subsetI, simp)
  also have ... = char R
    unfolding char-def by simp
  also have ... < card A
    unfolding A-def by simp
  finally have card (int-embed R ‘ A) < card A by simp
  hence ¬inj-on (int-embed R) A
    using pigeonhole by simp
  then obtain x y where xy:
    x ∈ A y ∈ A x ≠ y int-embed R x = int-embed R y
    unfolding inj-on-def by auto
```

```
  define v where v = nat (max x y − min x y)
  have a:int-embed R v = 0
    using xy int-embed-closed
    by (cases x < y, simp-all add:int-embed-diff v-def)
  moreover have v > 0
    using xy by (cases x < y, simp-all add:v-def)
  ultimately have char R ≤ v using char-bound by simp
  moreover have v ≤ char R
    using xy v-def A-def by (cases x < y, simp-all)
  ultimately have char R = v by simp
  then show ?thesis using a by simp
next
  case False
  hence char R = 0
    unfolding char-def by simp
  then show ?thesis by (simp add:int-embed-zero)
qed

lemma (in ring) embed-char-eq-0-iff:
  fixes n :: int
  shows int-embed R n = 0 ⟷ char R dvd n
proof (cases char R > 0)
  case True
  define r where r = n mod char R
  define s where s = n div char R
  have rs: r < char R r ≥ 0 n = r + s * char R
    using True by (simp-all add:r-def s-def)

  have int-embed R n = int-embed R r
    using int-embed-closed unfolding rs(3)
    by (simp add: int-embed-add  int-embed-mult embed-char-eq-0)

  moreover have nat r < char R using rs by simp
  hence int-embed R (nat r) ≠ 0 ∨ nat r = 0
    using True char-bound not-less by blast
  hence int-embed R r ≠ 0 ∨ r = 0
    using rs by simp

  ultimately have int-embed R n = 0 ⟷ r = 0
    using int-embed-zero by auto
  also have r = 0 ⟷ char R dvd n
    using r-def by auto
  finally show ?thesis by simp
next
  case False
  hence char R = 0 by simp
  hence a:x > 0 ⟹ int-embed R (int x) ≠ 0 for x
    using char-bound by auto
```

```
  have c:int-embed R (abs x) ≠ 0 ⟷ int-embed R x ≠ 0 for x
    using int-embed-closed
    by (cases x > 0, simp, simp add:int-embed-inv)

  have int-embed R x ≠ 0 if b:x ≠ 0 for x
  proof −
    have nat (abs x) > 0 using b by simp
    hence int-embed R (nat (abs x)) ≠ 0
      using a by blast
    hence int-embed R (abs x) ≠ 0 by simp
    thus ?thesis using c by simp
  qed
  hence int-embed R n = 0 ⟷ n = 0
    using int-embed-zero by auto
  also have n = 0 ⟷ char R dvd n using False by simp
  finally show ?thesis by simp
qed
```

This result can be found in [4, Theorem 1.44].

```
lemma (in domain) characteristic-is-prime:
  assumes char R > 0
  shows prime (char R)
proof (rule ccontr)
  have ¬(char R = 1)
    using embed-char-eq-0 int-embed-one by auto
  hence ¬(char R dvd 1) using assms(1) by simp
  moreover assume ¬(prime (char R))
  hence ¬(irreducible (char R))
    using irreducible-imp-prime-elem-gcd prime-elem-nat-iff by blast
  ultimately obtain p q where pq-def: p ∗ q = char R p > 1 q > 1
    using assms
    unfolding Factorial-Ring.irreducible-def by auto
  have int-embed R p ⊗ int-embed R q = 0
    using embed-char-eq-0 pq-def
    by (subst int-embed-mult[symmetric]) (metis of-nat-mult)
  hence int-embed R p = 0 ∨ int-embed R q = 0
    using integral int-embed-closed by simp
  hence p∗q ≤ p ∨ p∗q ≤ q
    using char-bound pq-def by auto
  thus False
    using pq-def(2,3) by simp
qed

lemma (in ring) char-ring-is-subring:
  subring (char-subring R) R
proof −
  have subring (int-embed R ‘ carrier int-ring) R
    by (intro ring.carrier-is-subring int.is-ring
        ring-hom-ring.img-is-subring[OF int-embed-ring-hom])
```

thus *?thesis* **by** *simp*
qed

lemma (**in** *cring*) *char-ring-is-subcring*:
subcring (*char-subring R*) *R*
using *subcringI'*[*OF char-ring-is-subring*] **by** *auto*

lemma (**in** *domain*) *char-ring-is-subdomain*:
subdomain (*char-subring R*) *R*
using *subdomainI'*[*OF char-ring-is-subring*] **by** *auto*

lemma *image-set-eqI*:
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \in B$
assumes $\bigwedge x.\ x \in B \Longrightarrow g\ x \in A \wedge f\ (g\ x) = x$
shows $f\ `\ A = B$
using *assms* **by** *force*

This is the binomial expansion theorem for commutative rings.

lemma (**in** *cring*) *binomial-expansion*:
fixes $n :: nat$
assumes [*simp*]: $x \in carrier\ R\ y \in carrier\ R$
shows $(x \oplus y)\ [\hat{}]\ n =$
$(\bigoplus k \in \{..n\}.\ int\text{-}embed\ R\ (n\ choose\ k) \otimes x\ [\hat{}]\ k \otimes y\ [\hat{}]\ (n-k))$
proof −
define A **where** $A = (\lambda k.\ \{A.\ A \subseteq \{..<n\} \wedge card\ A = k\})$

have *fin-A*: *finite* $(A\ i)$ **for** i
unfolding *A-def* **by** *simp*
have *disj-A*: *pairwise* $(\lambda i\ j.\ disjnt\ (A\ i)\ (A\ j))\ \{..n\}$
unfolding *pairwise-def disjnt-def A-def* **by** *auto*
have *card-A*: $B \in A\ i \Longrightarrow card\ B = i$ **if** $i \in \{..n\}$ **for** $i\ B$
unfolding *A-def* **by** *simp*
have *card-A2*: $card\ (A\ i) = (n\ choose\ i)$ **if** $i \in \{..n\}$ **for** i
unfolding *A-def* **using** *n-subsets*[**where** $A=\{..<n\}$] **by** *simp*

have *card-bound*: $card\ A \leq n$
if $A \subseteq \{..<n\}$ **for** $n\ A$
by (*metis card-lessThan finite-lessThan card-mono that*)
have *card-insert*: $card\ (insert\ n\ A) = card\ A + 1$
if $A \subseteq \{..<(n::nat)\}$ **for** $n\ A$
using *finite-subset that* **by** (*subst card-insert-disjoint*, *auto*)

have *embed-distr*: $[m] \cdot y = int\text{-}embed\ R\ (int\ m) \otimes y$
if $y \in carrier\ R$ **for** $m\ y$
unfolding *int-embed-def add-pow-def* **using** *that*
by (*simp add:add-pow-def*[*symmetric*] *int-pow-int add-pow-ldistr*)

have $(x \oplus y)\ [\hat{}]\ n =$
$(\bigoplus A \in Pow\ \{..<n\}.\ x\ [\hat{}]\ (card\ A) \otimes y\ [\hat{}]\ (n-card\ A))$

proof (*induction n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
have *s1*:
insert n ‘ Pow {..<*n*} = {*A*. *A* ⊆ {..<*n*+*1*} ∧ *n* ∈ *A*}
by (*intro image-set-eqI*[**where** *g*=λ*x*. *x* ∩ {..<*n*}], *auto*)
have *s2*:
Pow {..<*n*} = {*A*. *A* ⊆ {..<*n*+*1*} ∧ *n* ∉ *A*}
using *lessThan-Suc* **by** *auto*

have (*x* ⊕ *y*) [^] *Suc n* = (*x* ⊕ *y*) [^] *n* ⊗ (*x* ⊕ *y*) **by** *simp*
also have ... =
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*) ⊗ *y* [^] (*n*−*card A*)) ⊗
(*x* ⊕ *y*)
by (*subst Suc*, *simp*)
also have ... =
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*) ⊗ *y* [^] (*n*−*card A*)) ⊗ *x* ⊕
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*) ⊗ *y* [^] (*n*−*card A*)) ⊗ *y*
by (*subst r-distr*, *auto*)
also have ... =
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*) ⊗ *y* [^] (*n*−*card A*) ⊗ *x*) ⊕
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*) ⊗ *y* [^] (*n*−*card A*) ⊗ *y*)
by (*simp add*:*finsum-ldistr*)
also have ... =
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*+*1*) ⊗ *y* [^] (*n*−*card A*)) ⊕
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*) ⊗ *y* [^] (*n*−*card A*+*1*))
using *m-assoc m-comm*
by (*intro arg-cong2*[**where** *f*=(⊕)] *finsum-cong′*, *auto*)
also have ... =
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card* (*insert n A*))
⊗ *y* [^] (*n*+*1*−*card* (*insert n A*))) ⊕
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*) ⊗ *y* [^] (*n*+*1*−*card A*))
using *finite-subset card-bound card-insert Suc-diff-le*
by (*intro arg-cong2*[**where** *f*=(⊕)] *finsum-cong′*, *simp-all*)
also have ... =
(⨁ *A* ∈ *insert n ‘ Pow* {..<*n*}. *x* [^] (*card A*)
⊗ *y* [^] (*n*+*1*−*card A*)) ⊕
(⨁ *A* ∈ *Pow* {..<*n*}. *x* [^] (*card A*) ⊗ *y* [^] (*n*+*1*−*card A*))
by (*subst finsum-reindex*, *auto simp add*:*inj-on-def*)
also have ... =
(⨁ *A* ∈ {*A*. *A* ⊆ {..<*n*+*1*} ∧ *n* ∈ *A*}.
x [^] (*card A*) ⊗ *y* [^] (*n*+*1*−*card A*)) ⊕
(⨁ *A* ∈ {*A*. *A* ⊆ {..<*n*+*1*} ∧ *n* ∉ *A*}.
x [^] (*card A*) ⊗ *y* [^] (*n*+*1*−*card A*))
by (*intro arg-cong2*[**where** *f*=(⊕)] *finsum-cong′ s1 s2*, *simp-all*)
also have ... = (⨁ *A* ∈
{*A*. *A* ⊆ {..<*n*+*1*} ∧ *n* ∈ *A*} ∪ {*A*. *A* ⊆ {..<*n*+*1*} ∧ *n* ∉ *A*}.

```
      x [^] (card A) ⊗ y [^] (n+1−card A))
    by (subst finsum-Un-disjoint, auto)
  also have ... =
    (⨁ A ∈ Pow {..<n+1}. x [^] (card A) ⊗ y [^] (n+1−card A))
    by (intro finsum-cong', auto)
  finally show ?case by simp
 qed
 also have ... =
  (⨁ A ∈ (⋃ (A ` {..n})). x [^] (card A) ⊗ y [^] (n−card A))
  using card-bound by (intro finsum-cong', auto simp add:A-def)
 also have ... =
  (⨁ k ∈ {..n}. (⨁ A ∈ A k. x [^] (card A) ⊗ y [^] (n−card A)))
  using fin-A disj-A by (subst add.finprod-UN-disjoint, auto)
 also have ... = (⨁ k ∈ {..n}. (⨁ A ∈ A k. x [^] k ⊗ y [^] (n−k)))
  using card-A by (intro finsum-cong', auto)
 also have ... =
  (⨁ k ∈ {..n}. int-embed R (card (A k)) ⊗ x [^] k ⊗ y [^] (n−k))
  using int-embed-closed
  by (subst add.finprod-const, simp-all add:embed-distr m-assoc)
 also have ... =
  (⨁ k ∈ {..n}. int-embed R (n choose k) ⊗ x [^] k ⊗ y [^] (n−k))
  using int-embed-closed card-A2 by (intro finsum-cong', simp-all)
 finally show ?thesis by simp
qed

lemma bin-prime-factor:
 assumes prime p
 assumes k > 0 k < p
 shows p dvd (p choose k)
proof −
 have p dvd fact p
  using assms(1) prime-dvd-fact-iff by auto
 hence p dvd fact k * fact (p − k) * (p choose k)
  using binomial-fact-lemma assms by simp
 hence p dvd fact k ∨ p dvd fact (p−k) ∨ p dvd (p choose k)
  by (simp add: assms(1) prime-dvd-mult-eq-nat)
 thus p dvd (p choose k)
  using assms(1,2,3) prime-dvd-fact-iff by auto
qed

theorem (in domain) freshmans-dream:
 assumes char R > 0
 assumes [simp]: x ∈ carrier R y ∈ carrier R
 shows (x ⊕ y) [^] (char R) = x [^] char R ⊕ y [^] char R
  (is ?lhs = ?rhs)
proof −
 have c:prime (char R)
  using assms(1) characteristic-is-prime by auto
 have a:int-embed R (char R choose i) = 0
```

```
    if i ∈ {..char R} − {0, char R} for i
  proof −
    have i > 0 i < char R using that by auto
    hence char R dvd char R choose i
      using c bin-prime-factor by simp
    thus ?thesis using embed-char-eq-0-iff by simp
  qed

  have ?lhs = (⨁ k ∈ {..char R}. int-embed R (char R choose k)
    ⊗ x [^] k ⊗ y [^] (char R−k))
    using binomial-expansion[OF assms(2,3)] by simp
  also have ... = (⨁ k ∈ {0,char R}.int-embed R (char R choose k)
    ⊗ x [^] k ⊗ y [^] (char R−k))
    using a int-embed-closed
    by (intro add.finprod-mono-neutral-cong-right, simp, simp-all)
  also have ... = ?rhs
   using int-embed-closed assms(1) by (simp add:int-embed-one a-comm)
  finally show ?thesis by simp
qed
```

The following theorem is somtimes called Freshman's dream for obvious reasons, it can be found in Lidl and Niederreiter [4, Theorem 1.46].

```
lemma (in domain) freshmans-dream-ext:
  fixes m
  assumes char R > 0
  assumes [simp]: x ∈ carrier R y ∈ carrier R
  defines n ≡ char R^m
  shows (x ⊕ y) [^] n = x [^] n ⊕ y [^] n
    (is ?lhs = ?rhs)
  unfolding n-def
proof (induction m)
  case 0
  then show ?case by simp
next
  case (Suc m)
  have (x ⊕ y) [^] (char R^(m+1)) =
    (x ⊕ y) [^] (char R^m ∗ char R)
    by (simp add:mult.commute)
  also have ... = ((x ⊕ y) [^] (char R^m)) [^] char R
    using nat-pow-pow by simp
  also have ... = (x [^] (char R^m) ⊕ y [^] (char R^m)) [^] char R
    by (subst Suc, simp)
  also have ... =
    (x [^] (char R^m)) [^] char R ⊕ (y [^] (char R^m)) [^] char R
    by (subst freshmans-dream[OF assms(1), symmetric], simp-all)
  also have ... =
    x [^] (char R^m ∗ char R) ⊕ y [^] (char R^m ∗ char R)
    by (simp add:nat-pow-pow)
```

```
  also have ... = x [^] (char R^Suc m) ⊕ y [^] (char R^Suc m)
    by (simp add:mult.commute)
  finally show ?case by simp
qed
```

The following is a generalized version of the Frobenius homomorphism. The classic version of the theorem is the case where $k = 1$.

```
theorem (in domain) frobenius-hom:
  assumes char R  > 0
  assumes m = char R ^ k
  shows ring-hom-cring R R (λx. x [^] m)
proof −
  have a:(x ⊗ y) [^] m = x [^] m ⊗ y [^] m
    if b:x ∈ carrier R y ∈ carrier R for x y
    using b nat-pow-distrib by simp
  have b:(x ⊕ y) [^] m = x [^] m ⊕ y [^] m
    if b:x ∈ carrier R y ∈ carrier R for x y
    unfolding assms(2) freshmans-dream-ext[OF assms(1) b]
    by simp

  have ring-hom-ring R R (λx. x [^] m)
    by  (intro ring-hom-ringI a b is-ring, simp-all)

  thus ?thesis
    using RingHom.ring-hom-cringI is-cring by blast
qed

lemma (in domain) char-ring-is-subfield:
  assumes char R > 0
  shows subfield (char-subring R) R
proof −
  interpret d:domain R (| carrier := char-subring R |)
    using char-ring-is-subdomain subdomain-is-domain by simp

  have finite (char-subring R)
    using char-def assms by (metis card-ge-0-finite)
  hence Units (R (| carrier := char-subring R |))
    = char-subring R − {0}
    using d.finite-domain-units by simp

  thus ?thesis
    using subfieldI[OF char-ring-is-subcring] by simp
qed

lemma card-lists-length-eq':
  fixes A :: 'a set
  shows card {xs. set xs ⊆ A ∧ length xs = n} = card A ^ n
proof (cases finite A)
```

```
  case True
  then show ?thesis using card-lists-length-eq by auto
next
  case False
  hence inf-A: infinite A by simp
  show ?thesis
  proof (cases n = 0)
    case True
   hence card {xs. set xs ⊆ A ∧ length xs = n} = card {([] :: 'a list)}
      by (intro arg-cong[where f=card], auto simp add:set-eq-iff)
    also have ... = 1 by simp
    also have ... = card A^n using True inf-A by simp
    finally show ?thesis by simp
  next
    case False
    hence inj (replicate n)
      by (meson inj-onI replicate-eq-replicate)
    hence inj-on (replicate n) A using inj-on-subset
      by (metis subset-UNIV)
    hence infinite (replicate n ` A)
      using inf-A finite-image-iff by auto
    moreover have
      replicate n ` A ⊆ {xs. set xs ⊆ A ∧ length xs  = n}
      by (intro image-subsetI, auto)
    ultimately have infinite {xs. set xs ⊆ A ∧ length xs  = n}
      using infinite-super by auto
    hence card {xs. set xs ⊆ A ∧ length xs  = n} = 0 by simp
    then show ?thesis using inf-A False by simp
  qed
qed

lemma (in ring) card-span:
  assumes subfield K R
  assumes independent K w
  assumes set w ⊆ carrier R
  shows card (Span K w) = card K^(length w)
proof −
  define A where A = {x. set x ⊆ K ∧ length x = length w}
  define f where f = (λx. combine x w)

  have x ∈ f ` A if a:x ∈ Span K w for x
  proof −
    obtain y where y ∈ A x = f y
      unfolding A-def f-def
      using unique-decomposition[OF assms(1,2) a] by auto
    thus ?thesis by simp
  qed
  moreover have f x ∈ Span K w if a: x ∈ A for x
    using Span-eq-combine-set[OF assms(1,3)] a
```

```
    unfolding A-def f-def by auto
  ultimately have b:Span K w = f ‘ A by auto

  have False if a: x ∈ A y ∈ A f x = f y x ≠ y for x y
  proof −
    have f x ∈ Span K w using b a by simp
    thus False
      using a unique-decomposition[OF assms(1,2)]
      unfolding f-def A-def by blast
  qed
  hence f-inj: inj-on f A
    unfolding inj-on-def by auto

  have card (Span K w) = card (f ‘ A) using b by simp
  also have ... = card A by (intro card-image f-inj)
  also have ... = card K^length w
    unfolding A-def by (intro card-lists-length-eq′)
  finally show ?thesis by simp
qed

lemma (in ring) finite-carr-imp-char-ge-0:
  assumes finite (carrier R)
  shows char R > 0
proof −
  have char-subring R ⊆ carrier R
    using int-embed-closed by auto
  hence finite (char-subring R)
    using finite-subset assms by auto
  hence card (char-subring R) > 0
    using card-range-greater-zero by simp
  thus char R > 0
    unfolding char-def by simp
qed

lemma (in ring) char-consistent:
  assumes subring H R
  shows char (R (| carrier := H |)) = char R
proof −
  show ?thesis
    using int-embed-consistent[OF assms(1)]
    unfolding char-def by simp
qed

lemma (in ring-hom-ring) char-consistent:
  assumes inj-on h (carrier R)
  shows char R = char S
proof −
  have a:h (int-embed R (int n)) = int-embed S (int n) for n
    using R.int-embed-range[OF R.carrier-is-subring]
```

```
    using R.int-embed-range[OF R.carrier-is-subring]
    using S.int-embed-one R.int-embed-one
    using S.int-embed-zero R.int-embed-zero
    using S.int-embed-add R.int-embed-add
    by (induction n, simp-all)

  have b:h (int-embed R (−(int n))) = int-embed S (−(int n)) for n
    using R.int-embed-range[OF R.carrier-is-subring]
    using S.int-embed-range[OF S.carrier-is-subring] a
    by (simp add:R.int-embed-inv S.int-embed-inv)

  have c:h (int-embed R n) = int-embed S n for n
  proof (cases n ≥ 0)
    case True
    then obtain m where n = int m
      using nonneg-int-cases by auto
    then show ?thesis
      by (simp add:a)
  next
    case False
    hence n ≤ 0 by simp
    then obtain m where n = −int m
      using nonpos-int-cases by auto
    then show ?thesis by (simp add:b)
  qed

  have char S = card (h ‘ char-subring R)
    unfolding char-def image-image c by simp
  also have ... = card (char-subring R)
    using R.int-embed-range[OF R.carrier-is-subring]
    by (intro card-image inj-on-subset[OF assms(1)]) auto
  also have ... = char R unfolding char-def by simp
  finally show ?thesis
    by simp
qed

definition char-iso :: - ⇒ int set ⇒ 'a
  where char-iso R x = the-elem (int-embed R ‘ x)
```

The function *char-iso R* denotes the isomorphism between *ZFact* (*int* (*char R*)) and the characteristic subring.

```
lemma (in ring) char-iso: char-iso R ∈
  ring-iso (ZFact (char R)) (R(|carrier := char-subring R|))
proof −
  interpret h: ring-hom-ring int-ring R int-embed R
    using int-embed-ring-hom by simp

  have a-kernel ℤ R (int-embed R) = {x. int-embed R x = 0}
    unfolding a-kernel-def kernel-def by simp
```

```
  also have ... = {x. char R dvd x}
    using embed-char-eq-0-iff by simp
  also have ... = PIdl_ℤ (int (char R))
    unfolding cgenideal-def by auto
  also have ... = Idl_ℤ {int (char R)}
    using int.cgenideal-eq-genideal by simp
  finally have a:a-kernel ℤ R (int-embed R) = Idl_ℤ {int (char R)}
    by simp
  show ?thesis
    unfolding char-iso-def ZFact-def a[symmetric]
    by (intro h.FactRing-iso-set-aux)
qed
```

The size of a finite field must be a prime power. This can be found in Ireland and Rosen [3, Proposition 7.1.3].

```
theorem (in finite-field) finite-field-order:
  ∃n. order R = char R ^ n ∧ n > 0
proof −
  have a:char R > 0
    using finite-carr-imp-char-ge-0[OF finite-carrier]
    by simp
  let ?CR = char-subring R

  obtain v where v-def: set v = carrier R
    using finite-carrier finite-list by auto
  hence b:set v ⊆ carrier R by auto

  have carrier R = set v using v-def by simp
  also have ... ⊆ Span ?CR v
    using Span-base-incl[OF char-ring-is-subfield[OF a] b] by simp
  finally have carrier R ⊆ Span ?CR v by simp
  moreover have Span ?CR v ⊆ carrier R
    using int-embed-closed v-def by (intro Span-in-carrier, auto)
  ultimately have Span-v: Span ?CR v = carrier R by simp

  obtain w where w-def:
    set w ⊆ carrier R
    independent ?CR w
    Span ?CR v = Span ?CR w
    using b filter-base[OF char-ring-is-subfield[OF a]]
    by metis

  have Span-w: Span ?CR w = carrier R
    using w-def(3) Span-v by simp

  hence order R = card (Span ?CR w) by (simp add:order-def)
  also have ... = card ?CR^length w
    by (intro card-span char-ring-is-subfield[OF a] w-def(1,2))
  finally have c:
```

```
    order R = char R^(length w)
    by (simp add:char-def)
  have length w > 0
    using finite-field-min-order c by auto
  thus ?thesis using c by auto
qed

end
```

4 Formal Derivatives

```
theory Formal-Polynomial-Derivatives
  imports HOL−Algebra.Polynomial-Divisibility Ring-Characteristic
begin

definition pderiv (pderiv₁) where
  pderiv_R x = ring.normalize R (
    map (λi. int-embed R i ⊗_R ring.coeff R x i) (rev [1..<length x]))

context domain
begin

lemma coeff-range:
  assumes subring K R
  assumes f ∈ carrier (K[X])
  shows coeff f i ∈ K
proof −
  have coeff f i ∈ set f ∪ {0}
    using coeff-img(3) by auto
  also have ... ⊆ K ∪ {0}
    using assms(2) univ-poly-carrier polynomial-incl by blast
  also have ... ⊆ K
    using subringE[OF assms(1)] by simp
  finally show ?thesis by simp
qed

lemma pderiv-carr:
  assumes subring K R
  assumes f ∈ carrier (K[X])
  shows pderiv f ∈ carrier (K[X])
proof −
  have int-embed R i ⊗ coeff f i ∈ K for i
    using coeff-range[OF assms] int-embed-range[OF assms(1)]
    using subringE[OF assms(1)] by simp
  hence polynomial K (pderiv f)
    unfolding pderiv-def by (intro normalize-gives-polynomial, auto)
  thus ?thesis
    using univ-poly-carrier by auto
qed
```

lemma *pderiv-coeff*:
  assumes *subring K R*
  assumes $f \in$ *carrier* $(K[X])$
  shows *coeff* (*pderiv f*) *k* = *int-embed R* (*Suc k*) $\otimes$ *coeff f* (*Suc k*)
    (**is** *?lhs* = *?rhs*)
proof (*cases k* + *1* < *length f*)
  case *True*
  define *j* **where** *j* = *length f* − *k* − *2*
  define *d* **where**
    d = *map* (λ*i*. *int-embed R i* ⊗ *coeff f i*) (*rev* [*1*..<*length f*])

  have *a*: *j*+*1* < *length f*
    using *True* **unfolding** *j-def* **by** *simp*
  hence *b*: *j* < *length* [*1*..<*length f*]
    by *simp*
  have *c*: *k* < *length d*
    unfolding *d-def* **using** *True* **by** *simp*
  have *d*: *degree d* − *k* = *j*
    unfolding *d-def j-def* **by** *simp*
  have *e*: *rev* [*Suc 0*..<*length f*] ! *j* = *length f* − *1* − *j*
    using *b* **by** (*subst rev-nth*, *auto*)
  have *f*: *length f* − *j* − *1* = *k*+*1*
    unfolding *j-def* **using** *True* **by** *simp*

  have *coeff* (*pderiv f*) *k* = *coeff* (*normalize d*) *k*
    unfolding *pderiv-def d-def* **by** *simp*
  also have ... = *coeff d k*
    using *normalize-coeff* **by** *simp*
  also have ... = *d* ! *j*
    using *c d* **by** (*subst coeff-nth*, *auto*)
  also have
    ... = *int-embed R* (*length f* − *j* − *1*) ⊗ *coeff f* (*length f* − *j* − *1*)
    using *b e* **unfolding** *d-def* **by** *simp*
  also have ... = *?rhs*
    using *f* **by** *simp*
  finally show *?thesis* **by** *simp*
next
  case *False*
  hence *Suc k* ≥ *length f*
    by *simp*
  hence *a*:*coeff f* (*Suc k*) = **0**
    using *coeff-img* **by** *blast*
  have *b*:*coeff* (*pderiv f*) *k* = **0**
    unfolding *pderiv-def normalize-coeff*[*symmetric*] **using** *False*
    by (*intro coeff-length*, *simp*)
  show *?thesis*
    using *int-embed-range*[*OF carrier-is-subring*] **by** (*simp add*:*a b*)
qed

lemma *pderiv-const*:
assumes *degree x = 0*
shows *pderiv x* = $\mathbf{0}_{K[X]}$
proof (*cases length x = 0*)
case *True*
then show *?thesis* **by** (*simp add*:*univ-poly-zero pderiv-def*)
next
case *False*
hence *length x = 1* **using** *assms* **by** *linarith*
then obtain *y* **where** *x* = [*y*] **by** (*cases x*, *auto*)
then show *?thesis* **by** (*simp add*:*univ-poly-zero pderiv-def*)
qed

lemma *pderiv-var*:
shows *pderiv X* = $\mathbf{1}_{K[X]}$
unfolding *var-def pderiv-def*
by (*simp add*:*univ-poly-one int-embed-def*)

lemma *pderiv-zero*:
shows *pderiv* $\mathbf{0}_{K[X]}$ = $\mathbf{0}_{K[X]}$
unfolding *pderiv-def univ-poly-zero* **by** *simp*

lemma *pderiv-add*:
assumes *subring K R*
assumes [*simp*]: *f* ∈ *carrier* (*K*[*X*]) *g* ∈ *carrier* (*K*[*X*])
shows *pderiv* (*f* $\oplus_{K[X]}$ *g*) = *pderiv f* $\oplus_{K[X]}$ *pderiv g*
(**is** *?lhs* = *?rhs*)
proof −
interpret *p*: *ring* (*K*[*X*])
using *univ-poly-is-ring*[*OF assms*(*1*)] **by** *simp*

let *?n* = (λ*i*. *int-embed R i*)

have *a*[*simp*]:*?n k* ∈ *carrier R* **for** *k*
using *int-embed-range*[*OF carrier-is-subring*] **by** *auto*
have *b*[*simp*]:*coeff f k* ∈ *carrier R* **if** *f* ∈ *carrier* (*K*[*X*]) **for** *k f*
using *coeff-range*[*OF assms*(*1*)] *that*
using *subringE*(*1*)[*OF assms*(*1*)] **by** *auto*

have *coeff ?lhs i* = *coeff ?rhs i* **for** *i*
proof −
have *coeff ?lhs i* = *?n* (*i*+*1*) ⊗ *coeff* (*f* $\oplus_{K\ [X]}$ *g*) (*i*+*1*)
by (*simp add*: *pderiv-coeff*[*OF assms*(*1*)])
also have ... = *?n* (*i*+*1*) ⊗ (*coeff f* (*i*+*1*) ⊕ *coeff g* (*i*+*1*))
by (*subst coeff-add*[*OF assms*], *simp*)
also have ... = *?n* (*i*+*1*) ⊗ *coeff f* (*i*+*1*)
⊕ *int-embed R* (*i*+*1*) ⊗ *coeff g* (*i*+*1*)

```
    by (subst r-distr, simp-all)
  also have ... = coeff (pderiv f) i ⊕ coeff (pderiv g) i
    by (simp add: pderiv-coeff[OF assms(1)])
  also have ... = coeff (pderiv f ⊕_{K [X]} pderiv g) i
    using pderiv-carr[OF assms(1)]
    by (subst coeff-add[OF assms(1)], auto)
  finally show ?thesis by simp
 qed
 hence coeff ?lhs = coeff ?rhs by auto
 thus ?lhs = ?rhs
  using pderiv-carr[OF assms(1)]
  by (subst coeff-iff-polynomial-cond[where K=K])
    (simp-all add:univ-poly-carrier)+
qed

lemma pderiv-inv:
 assumes subring K R
 assumes [simp]: f ∈ carrier (K[X])
 shows pderiv (⊖_{K[X]} f) = ⊖_{K[X]} pderiv f (is ?lhs = ?rhs)
proof −
 interpret p: cring (K[X])
  using univ-poly-is-cring[OF assms(1)] by simp

 have pderiv (⊖_{K[X]} f) = pderiv (⊖_{K[X]} f) ⊕_{K[X]} 0_{K[X]}
  using pderiv-carr[OF assms(1)]
  by (subst p.r-zero, simp-all)
 also have ... = pderiv (⊖_{K[X]} f) ⊕_{K[X]} (pderiv f ⊖_{K[X]} pderiv f)
  using pderiv-carr[OF assms(1)] by simp
 also have ... = pderiv (⊖_{K[X]} f) ⊕_{K[X]} pderiv f ⊖_{K[X]} pderiv f
  using pderiv-carr[OF assms(1)]
  unfolding a-minus-def by (simp add:p.a-assoc)
 also have ... = pderiv (⊖_{K[X]} f ⊕_{K[X]} f) ⊖_{K[X]} pderiv f
  by (subst pderiv-add[OF assms(1)], simp-all)
 also have ... = pderiv 0_{K[X]} ⊖_{K[X]} pderiv f
  by (subst p.l-neg, simp-all)
 also have ... = 0_{K[X]} ⊖_{K[X]} pderiv f
  by (subst pderiv-zero, simp)
 also have ... = ⊖_{K[X]} pderiv f
  unfolding a-minus-def using pderiv-carr[OF assms(1)]
  by (subst p.l-zero, simp-all)
 finally show pderiv (⊖_{K[X]} f) = ⊖_{K[X]} pderiv f
  by simp
qed


lemma coeff-mult:
 assumes subring K R
 assumes f ∈ carrier (K[X]) g ∈ carrier (K[X])
```

shows *coeff* $(f \otimes_{K[X]} g)\ i =$
$(\bigoplus\ k \in \{..i\}.\ (coeff\ f)\ k \otimes (coeff\ g)\ (i - k))$
proof −
have *a*:*set f* ⊆ *carrier R*
using *assms*(*1*,*2*) *univ-poly-carrier*
using *subringE*(*1*)[*OF assms*(*1*)] *polynomial-incl* **by** *blast*
have *b*:*set g* ⊆ *carrier R*
using *assms*(*1*,*3*) *univ-poly-carrier*
using *subringE*(*1*)[*OF assms*(*1*)] *polynomial-incl* **by** *blast*
show *?thesis*
unfolding *univ-poly-mult poly-mult-coeff*[*OF a b*] **by** *simp*
qed

lemma *pderiv-mult*:
assumes *subring K R*
assumes [*simp*]: $f \in carrier\ (K[X])\ g \in carrier\ (K[X])$
shows *pderiv* $(f \otimes_{K[X]} g) =$
$pderiv\ f \otimes_{K[X]} g \oplus_{K[X]} f \otimes_{K[X]} pderiv\ g$
(**is** *?lhs* = *?rhs*)
proof −
interpret *p*: *cring* $(K[X])$
using *univ-poly-is-cring*[*OF assms*(*1*)] **by** *simp*

let *?n* = $(\lambda i.\ int\text{-}embed\ R\ i)$

have *a*[*simp*]:*?n k* ∈ *carrier R* **for** *k*
using *int-embed-range*[*OF carrier-is-subring*] **by** *auto*
have *b*[*simp*]:*coeff f k* ∈ *carrier R* **if** *f* ∈ *carrier* $(K[X])$ **for** *k f*
using *coeff-range*[*OF assms*(*1*)]
using *subringE*(*1*)[*OF assms*(*1*)] *that* **by** *auto*

have *coeff ?lhs i* = *coeff ?rhs i* **for** *i*
proof −
have *coeff ?lhs i* = *?n* (*i*+*1*) ⊗ *coeff* $(f \otimes_{K\ [X]} g)$ (*i*+*1*)
using *assms*(*2*,*3*) **by** (*simp add*: *pderiv-coeff*[*OF assms*(*1*)])
also have ... = *?n* (*i*+*1*) ⊗
$(\bigoplus k \in \{..i+1\}.\ coeff\ f\ k \otimes (coeff\ g\ (i + 1 - k)))$
by (*subst coeff-mult*[*OF assms*], *simp*)
also have ... =
$(\bigoplus k \in \{..i+1\}.\ ?n\ (i+1) \otimes (coeff\ f\ k \otimes coeff\ g\ (i + 1 - k)))$
by (*intro finsum-rdistr*, *simp-all add*:*Pi-def*)
also have ... =
$(\bigoplus k \in \{..i+1\}.\ ?n\ k \otimes (coeff\ f\ k \otimes coeff\ g\ (i + 1 - k)) \oplus$
?n (*i*+*1*−*k*) ⊗ (*coeff f k* ⊗ *coeff g* (*i* + *1* − *k*)))
using *int-embed-add*[*symmetric*] *of-nat-diff*
by (*intro finsum-cong*′)
(*simp-all add*:*l-distr*[*symmetric*] *of-nat-diff*)
also have ... =
$(\bigoplus k \in \{..i+1\}.\ ?n\ k \otimes coeff\ f\ k \otimes coeff\ g\ (i + 1 - k) \oplus$

coeff f k $\otimes$ (*?n* (*i*+1−*k*) $\otimes$ *coeff g* (*i* + 1 − *k*)))
using *Pi-def a b m-assoc m-comm*
by (*intro finsum-cong′ arg-cong2*[**where** *f*=($\oplus$)], *simp-all*)
also have ... =
($\bigoplus k \in \{..i+1\}$. *?n k* $\otimes$ *coeff f k* $\otimes$ *coeff g* (*i*+1−*k*)) $\oplus$
($\bigoplus k \in \{..i+1\}$. *coeff f k* $\otimes$ (*?n* (*i*+1−*k*) $\otimes$ *coeff g* (*i*+1−*k*)))
by (*subst finsum-addf*[*symmetric*], *simp-all add*:*Pi-def*)
also have ... =
($\bigoplus k \in$ *insert 0* $\{1..i+1\}$. *?n k* $\otimes$ *coeff f k* $\otimes$ *coeff g* (*i*+1−*k*)) $\oplus$
($\bigoplus k \in$ *insert* (*i*+1) $\{..i\}$. *coeff f k* $\otimes$ (*?n* (*i*+1−*k*) $\otimes$ *coeff g* (*i*+1−*k*)))
using *subringE*(1)[*OF assms*(1)]
by (*intro arg-cong2*[**where** *f*=($\oplus$)] *finsum-cong′*)
(*auto simp*:*set-eq-iff*)
also have ... =
($\bigoplus k \in \{1..i+1\}$. *?n k* $\otimes$ *coeff f k* $\otimes$ *coeff g* (*i*+1−*k*)) $\oplus$
($\bigoplus k \in \{..i\}$. *coeff f k* $\otimes$ (*?n* (*i*+1−*k*) $\otimes$ *coeff g* (*i*+1−*k*)))
by (*subst* (1 2) *finsum-insert*, *auto simp add*:*int-embed-zero*)
also have ... =
($\bigoplus k \in$ *Suc* ‘ $\{..i\}$. *?n k* $\otimes$ *coeff f* (*k*) $\otimes$ *coeff g* (*i*+1−*k*)) $\oplus$
($\bigoplus k \in \{..i\}$. *coeff f k* $\otimes$ (*?n* (*i*+1−*k*) $\otimes$ *coeff g* (*i*+1−*k*)))
by (*intro arg-cong2*[**where** *f*=($\oplus$)] *finsum-cong′*)
(*simp-all add*:*Pi-def atMost-atLeast0*)
also have ... =
($\bigoplus k \in \{..i\}$. *?n* (*k*+1) $\otimes$ *coeff f* (*k*+1) $\otimes$ *coeff g* (*i*−*k*)) $\oplus$
($\bigoplus k \in \{..i\}$. *coeff f k* $\otimes$ (*?n* (*i*+1−*k*) $\otimes$ *coeff g* (*i*+1−*k*)))
by (*subst finsum-reindex*, *auto*)
also have ... =
($\bigoplus k \in \{..i\}$. *coeff* (*pderiv f*) *k* $\otimes$ *coeff g* (*i*−*k*)) $\oplus$
($\bigoplus k \in \{..i\}$. *coeff f k* $\otimes$ *coeff* (*pderiv g*) (*i*−*k*))
using *Suc-diff-le*
by (*subst* (1 2) *pderiv-coeff*[*OF assms*(1)])
(*auto intro*!: *finsum-cong′*)
also have ... =
coeff (*pderiv f* $\otimes_{K[X]}$ *g*) *i* $\oplus$ *coeff* (*f* $\otimes_{K[X]}$ *pderiv g*) *i*
using *pderiv-carr*[*OF assms*(1)]
by (*subst* (1 2) *coeff-mult*[*OF assms*(1)], *auto*)
also have ... = *coeff ?rhs i*
using *pderiv-carr*[*OF assms*(1)]
by (*subst coeff-add*[*OF assms*(1)], *auto*)
finally show *?thesis* **by** *simp*
qed

hence *coeff ?lhs* = *coeff ?rhs* **by** *auto*
thus *?lhs* = *?rhs*
using *pderiv-carr*[*OF assms*(1)]
by (*subst coeff-iff-polynomial-cond*[**where** *K*=*K*])
(*simp-all add*:*univ-poly-carrier*)
qed

lemma *pderiv-pow*:
assumes $n > (0 :: nat)$
assumes *subring K R*
assumes [*simp*]: $f \in carrier\ (K[X])$
shows $pderiv\ (f\ [\hat{}]_{K[X]}\ n) =$
$int\text{-}embed\ (K[X])\ n \otimes_{K[X]} f\ [\hat{}]_{K[X]}\ (n-1) \otimes_{K[X]} pderiv\ f$
(**is** *?lhs = ?rhs*)
proof −
interpret *p*: *cring* $(K[X])$
using *univ-poly-is-cring*[*OF assms*(*2*)] **by** *simp*

let *?n* $= \lambda n.\ int\text{-}embed\ (K[X])\ n$

have [*simp*]: *?n i* $\in carrier\ (K[X])$ **for** *i*
using *p.int-embed-range*[*OF p.carrier-is-subring*] **by** *simp*

obtain *m* **where** *n-def*: $n = Suc\ m$ **using** *assms*(*1*) *lessE* **by** *blast*
have $pderiv\ (f\ [\hat{}]_{K[X]}\ (m+1)) =$
$?n\ (m+1) \otimes_{K[X]} f\ [\hat{}]_{K[X]}\ m \otimes_{K[X]} pderiv\ f$
proof (*induction m*)
case *0*
then show *?case*
using *pderiv-carr*[*OF assms*(*2*)] *assms*(*3*)
using *p.int-embed-one* **by** *simp*
next
case (*Suc m*)
have $pderiv\ (f\ [\hat{}]_{K\ [X]}\ (Suc\ m\ +\ 1)) =$
$pderiv\ (f\ [\hat{}]_{K\ [X]}\ (m+1) \otimes_{K[X]} f)$
by *simp*
also have ... =
$pderiv\ (f\ [\hat{}]_{K\ [X]}\ (m+1)) \otimes_{K[X]} f \oplus_{K[X]}$
$f\ [\hat{}]_{K\ [X]}\ (m+1) \otimes_{K[X]} pderiv\ f$
using *assms*(*3*) **by** (*subst pderiv-mult*[*OF assms*(*2*)], *auto*)
also have ... =
$(?n\ (m+1) \otimes_{K\ [X]} f\ [\hat{}]_{K\ [X]}\ m \otimes_{K\ [X]} pderiv\ f) \otimes_{K[X]} f$
$\oplus_{K[X]} f\ [\hat{}]_{K\ [X]}\ (m+1) \otimes_{K[X]} pderiv\ f$
by (*subst Suc*(*1*), *simp*)
also have
$... = ?n\ (m+1) \otimes_{K[X]} (f\ [\hat{}]_{K\ [X]}\ (m+1) \otimes_{K[X]} pderiv\ f)$
$\oplus_{K[X]} \mathbf{1}_{K\ [X]} \otimes_{K[X]} (f\ [\hat{}]_{K\ [X]}\ (m+1) \otimes_{K[X]} pderiv\ f)$
using *assms*(*3*) *pderiv-carr*[*OF assms*(*2*)]
apply (*intro arg-cong2*[**where** $f=(\oplus_{K[X]})$])
apply (*simp add*:*p.m-assoc*)
apply (*simp add*:*p.m-comm*)
by *simp*
also have

```
    ... = (?n (m+1) ⊕K[X] 1K [X]) ⊗K [X]
    (f [^]K [X] (m+1) ⊗K [X] pderiv f)
    using assms(3) pderiv-carr[OF assms(2)]
    by (subst p.l-distr[symmetric], simp-all)
  also have ... =
    (1K [X] ⊕K[X] ?n (m+1)) ⊗K [X]
    (f [^]K [X] (m+1) ⊗K [X] pderiv f)
    using assms(3) pderiv-carr[OF assms(2)]
    by (subst p.a-comm, simp-all)
  also have ... = ?n (1+ Suc m)
    ⊗K [X] f [^]K [X] (Suc m) ⊗K [X] pderiv f
    using assms(3) pderiv-carr[OF assms(2)] of-nat-add
    apply (subst (2) of-nat-add, subst p.int-embed-add)
    by (simp add:p.m-assoc p.int-embed-one)
  finally show ?case by simp
 qed
 thus ?thesis using n-def by auto
qed

lemma pderiv-var-pow:
 assumes n > (0::nat)
 assumes subring K R
 shows pderiv (X [^]K[X] n) =
   int-embed (K[X]) n ⊗K[X] X [^]K[X] (n−1)
proof −
 interpret p: cring (K[X])
   using univ-poly-is-cring[OF assms(2)] by simp

 have [simp]: int-embed (K[X]) i ∈ carrier (K[X]) for i
   using p.int-embed-range[OF p.carrier-is-subring] by simp

 show ?thesis
   using var-closed[OF assms(2)]
   using pderiv-var[where K=K] pderiv-carr[OF assms(2)]
   by (subst pderiv-pow[OF assms(1,2)], simp-all)
qed

lemma int-embed-consistent-with-poly-of-const:
 assumes subring K R
 shows int-embed (K[X]) m = poly-of-const (int-embed R m)
proof −
 define K' where K' = R (| carrier := K |)
 interpret p: cring (K[X])
   using univ-poly-is-cring[OF assms] by simp
 interpret d: domain K'
   unfolding K'-def
   using assms(1) subdomainI' subdomain-is-domain by simp
 interpret h: ring-hom-ring  K' K[X] poly-of-const
```

```
    unfolding K'-def
    using canonical-embedding-ring-hom[OF assms(1)] by simp

  define n where n=nat (abs m)

  have a1: int-embed (K[X]) (int n) = poly-of-const (int-embed K' n)
  proof (induction n)
    case 0
    then show ?case by (simp add:d.int-embed-zero p.int-embed-zero)
  next
    case (Suc n)
    then show ?case
      using d.int-embed-closed d.int-embed-add d.int-embed-one
      by (simp add:p.int-embed-add p.int-embed-one)
  qed
  also have ... = poly-of-const (int-embed R n)
    unfolding K'-def using int-embed-consistent[OF assms] by simp
  finally have a:
    int-embed (K[X]) (int n) = poly-of-const (int-embed R (int n))
    by simp

  have int-embed (K[X]) (-(int n)) =
    poly-of-const (int-embed K' (- (int n)))
   using d.int-embed-closed a1 by (simp add: p.int-embed-inv d.int-embed-inv)
  also have ... = poly-of-const (int-embed R (- (int n)))
    unfolding K'-def using int-embed-consistent[OF assms] by simp
  finally have b:
    int-embed (K[X]) (-int n) = poly-of-const (int-embed R (-int n))
    by simp

  show ?thesis
    using a b n-def by (cases m ≥ 0, simp, simp)
qed

end

end
```

5 Factorization into Monic Polynomials

```
theory Monic-Polynomial-Factorization
imports
  Finite-Fields-Factorization-Ext
  Formal-Polynomial-Derivatives
begin

hide-const Factorial-Ring.multiplicity
hide-const Factorial-Ring.irreducible
```

lemma (**in** *domain*) *finprod-mult-of*:
assumes *finite A*
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \in carrier\ (mult\text{-}of\ R)$
shows *finprod R f A* = *finprod* (*mult-of R*) *f A*
using *assms* **by** (*induction A rule*:*finite-induct*, *auto*)

lemma (**in** *ring*) *finite-poly*:
assumes *subring K R*
assumes *finite K*
shows
finite $\{f.\ f \in carrier\ (K[X]) \wedge degree\ f = n\}$ (**is** *finite ?A*)
finite $\{f.\ f \in carrier\ (K[X]) \wedge degree\ f \leq n\}$ (**is** *finite ?B*)
proof −
have *finite* $\{f.\ set\ f \subseteq K \wedge length\ f \leq n + 1\}$ (**is** *finite ?C*)
using *assms*(*2*) *finite-lists-length-le* **by** *auto*
moreover have *?B* ⊆ *?C*
by (*intro subsetI*)
(*auto simp*:*univ-poly-carrier*[*symmetric*] *polynomial-def*)
ultimately show *a*: *finite ?B*
using *finite-subset* **by** *auto*
moreover have *?A* ⊆ *?B*
by (*intro subsetI*, *simp*)
ultimately show *finite ?A*
using *finite-subset* **by** *auto*
qed

definition *pmult* :: - ⇒ *'a list* ⇒ *'a list* ⇒ *nat* (*pmult*ı)
where $pmult_R\ d\ p$ = *multiplicity* (*mult-of* (*poly-ring R*)) *d p*

definition *monic-poly* :: - ⇒ *'a list* ⇒ *bool*
where *monic-poly R f* =
$(f \neq [] \wedge lead\text{-}coeff\ f = \mathbf{1}_R \wedge f \in carrier\ (poly\text{-}ring\ R))$

definition *monic-irreducible-poly* **where**
monic-irreducible-poly R f =
(*monic-poly R f* ∧ $pirreducible_R\ (carrier\ R)\ f$)

abbreviation *m-i-p* ≡ *monic-irreducible-poly*

locale *polynomial-ring* = *field* +
fixes *K*
assumes *polynomial-ring-assms*: *subfield K R*
begin

lemma *K-subring*: *subring K R*
using *polynomial-ring-assms subfieldE*(*1*) **by** *auto*

abbreviation *P* **where** *P* ≡ *K*[*X*]

This locale is used to specialize the following lemmas for a fixed

coefficient ring. It can be introduced in a context as an intepretation to be able to use the following specialized lemmas. Because it is not (and should not) introduced as a sublocale it has no lasting effect for the field locale itself.

lemmas
poly-mult-lead-coeff = *poly-mult-lead-coeff*[*OF K-subring*]
and *degree-add-distinct* = *degree-add-distinct*[*OF K-subring*]
and *coeff-add* = *coeff-add*[*OF K-subring*]
and *var-closed* = *var-closed*[*OF K-subring*]
and *degree-prod* = *degree-prod*[*OF - K-subring*]
and *degree-pow* = *degree-pow*[*OF K-subring*]
and *pirreducible-degree* = *pirreducible-degree*[*OF polynomial-ring-assms*]
and *degree-one-imp-pirreducible* =
degree-one-imp-pirreducible[*OF polynomial-ring-assms*]
and *var-pow-closed* = *var-pow-closed*[*OF K-subring*]
and *var-pow-carr* = *var-pow-carr*[*OF K-subring*]
and *univ-poly-a-inv-degree* = *univ-poly-a-inv-degree*[*OF K-subring*]
and *var-pow-degree* = *var-pow-degree*[*OF K-subring*]
and *pdivides-zero* = *pdivides-zero*[*OF K-subring*]
and *pdivides-imp-degree-le* = *pdivides-imp-degree-le*[*OF K-subring*]
and *var-carr* = *var-carr*[*OF K-subring*]
and *rupture-eq-0-iff* = *rupture-eq-0-iff*[*OF polynomial-ring-assms*]
and *rupture-is-field-iff-pirreducible* =
rupture-is-field-iff-pirreducible[*OF polynomial-ring-assms*]
and *rupture-surj-hom* = *rupture-surj-hom*[*OF K-subring*]
and *canonical-embedding-ring-hom* =
canonical-embedding-ring-hom[*OF K-subring*]
and *rupture-surj-norm-is-hom* = *rupture-surj-norm-is-hom*[*OF K-subring*]
and *rupture-surj-as-eval* = *rupture-surj-as-eval*[*OF K-subring*]
and *eval-cring-hom* = *eval-cring-hom*[*OF K-subring*]
and *coeff-range* = *coeff-range*[*OF K-subring*]
and *finite-poly* = *finite-poly*[*OF K-subring*]
and *int-embed-consistent-with-poly-of-const* =
int-embed-consistent-with-poly-of-const[*OF K-subring*]
and *pderiv-var-pow* = *pderiv-var-pow*[*OF - K-subring*]
and *pderiv-add* = *pderiv-add*[*OF K-subring*]
and *pderiv-inv* = *pderiv-inv*[*OF K-subring*]
and *pderiv-mult* = *pderiv-mult*[*OF K-subring*]
and *pderiv-pow* = *pderiv-pow*[*OF - K-subring*]
and *pderiv-carr* = *pderiv-carr*[*OF K-subring*]

sublocale *p*:*principal-domain poly-ring R*
by (*simp add*: *carrier-is-subfield univ-poly-is-principal*)

end

context *field*
begin

interpretation *polynomial-ring R carrier R*
using *carrier-is-subfield field-axioms*
by (*simp add*:*polynomial-ring-def polynomial-ring-axioms-def*)

lemma *pdivides-mult-r*:
assumes $a \in$ *carrier* (*mult-of P*)
assumes $b \in$ *carrier* (*mult-of P*)
assumes $c \in$ *carrier* (*mult-of P*)
shows $a \otimes_P c$ *pdivides* $b \otimes_P c \longleftrightarrow a$ *pdivides* b
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
have *a*:$b \otimes_P c \in$ *carrier* $P - \{\mathbf{0}_P\}$
using *assms p.mult-of.m-closed* **by** *force*
have *b*:$a \otimes_P c \in$ *carrier P*
using *assms* **by** *simp*
have *c*:$b \in$ *carrier* $P - \{\mathbf{0}_P\}$
using *assms p.mult-of.m-closed* **by** *force*
have *d*:$a \in$ *carrier P* **using** *assms* **by** *simp*
have *?lhs* $\longleftrightarrow a \otimes_P c$ *divides*$_{\textit{mult-of } P}$ $b \otimes_P c$
unfolding *pdivides-def* **using** *p.divides-imp-divides-mult a b*
by (*meson divides-mult-imp-divides*)
also have ... $\longleftrightarrow a$ *divides*$_{\textit{mult-of } P}$ b
using *p.mult-of.divides-mult-r*[*OF assms*] **by** *simp*
also have ... $\longleftrightarrow$ *?rhs*
unfolding *pdivides-def* **using** *p.divides-imp-divides-mult c d*
by (*meson divides-mult-imp-divides*)
finally show *?thesis* **by** *simp*
qed

lemma *lead-coeff-carr*:
assumes $x \in$ *carrier* (*mult-of P*)
shows *lead-coeff x* $\in$ *carrier* $R - \{\mathbf{0}\}$
proof (*cases x*)
case *Nil*
then show *?thesis* **using** *assms* **by** (*simp add*:*univ-poly-zero*)
next
case (*Cons a list*)
hence *a*: *polynomial* (*carrier R*) (*a # list*)
using *assms univ-poly-carrier* **by** *auto*
have *lead-coeff x* = *a*
using *Cons* **by** *simp*
also have $a \in$ *carrier* $R - \{\mathbf{0}\}$
using *lead-coeff-not-zero a* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *lead-coeff-poly-of-const*:
assumes $r \neq \mathbf{0}$
shows *lead-coeff* (*poly-of-const r*) = *r*

using *assms*
by (*simp add*:*poly-of-const-def*)

lemma *lead-coeff-mult*:
assumes $f \in$ *carrier* (*mult-of P*)
assumes $g \in$ *carrier* (*mult-of P*)
shows *lead-coeff* ($f \otimes_P g$) = *lead-coeff f* $\otimes$ *lead-coeff g*
unfolding *univ-poly-mult* **using** *assms*
using *univ-poly-carrier*[**where** *R*=*R* **and** *K*=*carrier R*]
by (*subst poly-mult-lead-coeff*) (*simp-all add*:*univ-poly-zero*)

lemma *monic-poly-carr*:
assumes *monic-poly R f*
shows $f \in$ *carrier P*
using *assms* **unfolding** *monic-poly-def* **by** *simp*

lemma *monic-poly-add-distinct*:
assumes *monic-poly R f*
assumes $g \in$ *carrier P degree g* < *degree f*
shows *monic-poly R* ($f \oplus_P g$)
proof (*cases* $g \neq \mathbf{0}_P$)
case *True*
define *n* **where** *n* = *degree f*
have $f \in$ *carrier P* − {$\mathbf{0}_P$}
using *assms*(*1*) *univ-poly-zero*
unfolding *monic-poly-def* **by** *auto*
hence *degree* ($f \oplus_P g$) = *max* (*degree f*) (*degree g*)
using *assms*(*2*,*3*) *True*
by (*subst degree-add-distinct*, *simp-all*)
also have ... = *degree f*
using *assms*(*3*) **by** *simp*
finally have *b*: *degree* ($f \oplus_P g$) = *n*
unfolding *n-def* **by** *simp*
moreover have *n* > *0*
using *assms*(*3*) **unfolding** *n-def* **by** *simp*
ultimately have *degree* ($f \oplus_P g$) ≠ *degree* ([])
by *simp*
hence *a*:$f \oplus_P g \neq []$ **by** *auto*

have *degree* [] = *0* **by** *simp*
also have ... < *degree f*
using *assms*(*3*) **by** *simp*
finally have *degree f* ≠ *degree* [] **by** *simp*
hence *c*: $f \neq []$ **by** *auto*

have *d*: *length g* ≤ *n*
using *assms*(*3*) **unfolding** *n-def* **by** *simp*

have *lead-coeff* ($f \oplus_P g$) = *coeff* ($f \oplus_P g$) *n*

```
    using a b by (cases f ⊕_P g, auto)
  also have ... = coeff f n ⊕ coeff g n
    using monic-poly-carr assms
    by (subst coeff-add, auto)
  also have ... = lead-coeff f ⊕ coeff g n
    using c unfolding n-def by (cases f, auto)
  also have ... = 1 ⊕ 0
    using assms(1) unfolding monic-poly-def
    unfolding subst coeff-length[OF d] by simp
  also have ... = 1
    by simp
  finally have lead-coeff (f ⊕_P g) = 1 by simp
  moreover have f ⊕_P g ∈ carrier P
    using monic-poly-carr assms by simp
  ultimately show  ?thesis
    using a  unfolding monic-poly-def by auto
next
  case False
  then show ?thesis using assms monic-poly-carr by simp
qed

lemma monic-poly-one: monic-poly R 1_P
proof −
  have 1_P ∈ carrier P
    by simp
  thus ?thesis
    by (simp add:univ-poly-one monic-poly-def)
qed

lemma monic-poly-var: monic-poly R X
proof −
  have X ∈ carrier P
    using var-closed by simp
  thus ?thesis
    by (simp add:var-def monic-poly-def)
qed

lemma monic-poly-carr-2:
  assumes monic-poly R f
  shows f ∈ carrier (mult-of P)
  using assms unfolding monic-poly-def
  by (simp add:univ-poly-zero)

lemma monic-poly-mult:
  assumes monic-poly R f
  assumes monic-poly R g
  shows monic-poly R (f ⊗_P g)
proof −
  have lead-coeff (f ⊗_P g) = lead-coeff f ⊗_R lead-coeff g
```

```
    using assms monic-poly-carr-2
    by (subst lead-coeff-mult) auto
  also have ... = 1
    using assms unfolding monic-poly-def by simp
  finally have lead-coeff (f ⊗_P g) = 1_R by simp
  moreover have (f ⊗_P g) ∈ carrier (mult-of P)
    using monic-poly-carr-2 assms by blast
  ultimately show ?thesis
    by (simp add:monic-poly-def univ-poly-zero)
qed

lemma monic-poly-pow:
  assumes monic-poly R f
  shows monic-poly R (f [^]_P (n::nat))
  using assms monic-poly-one monic-poly-mult
  by (induction n, auto)

lemma monic-poly-prod:
  assumes finite A
  assumes ⋀x. x ∈ A ⟹ monic-poly R (f x)
  shows monic-poly R (finprod P f A)
  using assms
proof (induction A rule:finite-induct)
  case empty
  then show ?case by (simp add:monic-poly-one)
next
  case (insert x F)
  have a: f ∈ F → carrier P
    using insert monic-poly-carr by simp
  have b: f x ∈ carrier P
    using insert monic-poly-carr by simp
  have monic-poly R (f x ⊗_P finprod P f F)
    using insert by (intro monic-poly-mult) auto
  thus ?case
    using insert a b by (subst p.finprod-insert, auto)
qed

lemma monic-poly-not-assoc:
  assumes monic-poly R f
  assumes monic-poly R g
  assumes f ∼_(mult-of P) g
  shows f = g
proof −
  obtain u where u-def: f = g ⊗_P u u ∈ Units (mult-of P)
    using p.mult-of.associatedD2 assms monic-poly-carr-2
    by blast

  hence u ∈ Units P by simp
  then obtain v where v-def: u = [v] v ≠ 0_R v ∈ carrier R
```

```
    using univ-poly-carrier-units by auto

  have 1 = lead-coeff f
    using assms(1) by (simp add:monic-poly-def)
  also have  ... = lead-coeff (g ⊗_P u)
    by (simp add:u-def)
  also have ... = lead-coeff g ⊗ lead-coeff u
    using assms(2) monic-poly-carr-2 v-def u-def(2)
    by (subst lead-coeff-mult, auto simp add:univ-poly-zero)
  also have ... = lead-coeff g ⊗ v
    using v-def by simp
  also have ... = v
    using assms(2) v-def(3) by (simp add:monic-poly-def)
  finally have 1 = v by simp
  hence u = 1_P
    using v-def by (simp add:univ-poly-one)
  thus f = g
    using u-def assms monic-poly-carr by simp
qed

lemma monic-poly-span:
  assumes x ∈ carrier (mult-of P) irreducible (mult-of P) x
  shows ∃ y. monic-irreducible-poly R y ∧ x ∼_(mult-of P) y
proof −
  define z where z = poly-of-const (inv (lead-coeff x))
  define y where y = x ⊗_P z

  have x-carr: x ∈ carrier (mult-of P) using assms by simp

  hence lx-ne-0: lead-coeff x ≠ 0
    and lx-unit: lead-coeff x ∈ Units R
    using lead-coeff-carr[OF x-carr] by (auto simp add:field-Units)
  have lx-inv-ne-0: inv (lead-coeff x) ≠ 0
    using lx-unit
    by (metis Units-closed Units-r-inv r-null zero-not-one)
  have lx-inv-carr: inv (lead-coeff x) ∈ carrier R
    using lx-unit by simp

  have z ∈ carrier P
    using lx-inv-carr poly-of-const-over-carrier
    unfolding z-def by auto
  moreover have z ≠ 0_P
    using lx-inv-ne-0
    by (simp add:z-def poly-of-const-def univ-poly-zero)
  ultimately have z-carr: z ∈ carrier (mult-of P) by simp
  have z-unit: z ∈ Units (mult-of P)
    using lx-inv-ne-0 lx-inv-carr
    by (simp add:univ-poly-carrier-units z-def poly-of-const-def)
  have y-exp: y = x ⊗_(mult-of P) z
```

```
    by (simp add:y-def)
  hence y-carr: y ∈ carrier (mult-of P)
    using x-carr z-carr p.mult-of.m-closed by simp

  have irreducible (mult-of P) y
    unfolding y-def using assms z-unit z-carr
    by (intro p.mult-of.irreducible-prod-rI, auto)
  moreover have lead-coeff y = 1_R
    unfolding y-def using x-carr z-carr lx-inv-ne-0 lx-unit
    by (simp add: lead-coeff-mult z-def lead-coeff-poly-of-const)
  hence monic-poly R y
    using y-carr unfolding monic-poly-def
    by (simp add:univ-poly-zero)
  ultimately have monic-irreducible-poly R y
    using p.irreducible-mult-imp-irreducible y-carr
    by (simp add:monic-irreducible-poly-def ring-irreducible-def)
  moreover have y ∼(mult-of P) x
    by (intro p.mult-of.associatedI2[OF z-unit] y-def x-carr)
  hence x ∼(mult-of P) y
    using x-carr y-carr by (simp add:p.mult-of.associated-sym)
  ultimately show ?thesis by auto
qed

lemma monic-polys-are-canonical-irreducibles:
  canonical-irreducibles (mult-of P) {d. monic-irreducible-poly R d}
  (is canonical-irreducibles (mult-of P) ?S)
proof −
  have sp-1:
    ?S ⊆ {x ∈ carrier (mult-of P). irreducible (mult-of P) x}
    unfolding monic-irreducible-poly-def ring-irreducible-def
    using monic-poly-carr
    by (intro subsetI, simp add: p.irreducible-imp-irreducible-mult)
  have sp-2: x = y
      if x ∈ ?S y ∈ ?S x ∼(mult-of P) y for x y
    using that monic-poly-not-assoc
    by (simp add:monic-irreducible-poly-def)

  have sp-3: ∃ y ∈ ?S. x ∼(mult-of P) y
    if x ∈ carrier (mult-of P) irreducible (mult-of P) x for x
    using that monic-poly-span by simp

  thus ?thesis using sp-1 sp-2 sp-3
    unfolding canonical-irreducibles-def by simp
qed

lemma
  assumes monic-poly R a
  shows factor-monic-poly:
    a = (⊗ P d∈{d. monic-irreducible-poly R d ∧ pmult d a > 0}.
```

```
    d [^]_P pmult d a) (is ?lhs = ?rhs)
   and factor-monic-poly-fin:
    finite {d. monic-irreducible-poly R d ∧ pmult d a > 0}
proof −
 let ?S = {d. monic-irreducible-poly R d}
 let ?T = {d. monic-irreducible-poly R d ∧ pmult d a > 0}
 let ?mip = monic-irreducible-poly R

 have sp-4: a ∈ carrier (mult-of P)
  using assms monic-poly-carr-2
  unfolding monic-irreducible-poly-def by simp

 have b-1: x ∈ carrier (mult-of P) if ?mip x for x
  using that monic-poly-carr-2
  unfolding monic-irreducible-poly-def by simp
 have b-2:irreducible (mult-of P) x if ?mip x for x
  using that
  unfolding monic-irreducible-poly-def ring-irreducible-def
  by (simp add: monic-poly-carr p.irreducible-imp-irreducible-mult)
 have b-3:x ∈ carrier P if ?mip x for x
  using that monic-poly-carr
  unfolding monic-irreducible-poly-def
  by simp

 have a-carr: a ∈ carrier P − {0_P}
  using sp-4 by simp

 have ?T = {d. ?mip d ∧ multiplicity (mult-of P) d a > 0}
  by (simp add:pmult-def)
 also have ... = {d ∈ ?S. multiplicity (mult-of P) d a > 0}
  using p.mult-of.multiplicity-gt-0-iff[OF b-1 b-2 sp-4]
  by (intro order-antisym subsetI, auto)
 finally have t:?T = {d ∈ ?S. multiplicity (mult-of P) d a > 0}
  by simp

 show fin-T: finite ?T
  unfolding t
  using p.mult-of.split-factors(1)
   [OF monic-polys-are-canonical-irreducibles]
  using sp-4 by auto

 have a:x [^]_P (n::nat) ∈ carrier (mult-of P) if ?mip x for x n
 proof −
  have monic-poly R (x [^]_P n)
   using that monic-poly-pow
   unfolding monic-irreducible-poly-def by auto
  thus ?thesis
   using monic-poly-carr-2 by simp
 qed
```

```
  have ?lhs ∼(mult-of P)
    finprod (mult-of P)
      (λd. d [^](mult-of P) (multiplicity (mult-of P) d a)) ?T
    unfolding t
    by (intro p.mult-of.split-factors(2)
        [OF monic-polys-are-canonical-irreducibles sp-4])
  also have ... =
    finprod (mult-of P) (λd. d [^]P (multiplicity (mult-of P) d a)) ?T
    by (simp add:nat-pow-mult-of)
  also have ... = ?rhs
    using fin-T a
    by (subst p.finprod-mult-of, simp-all add:pmult-def)
  finally have ?lhs ∼(mult-of P) ?rhs by simp
  moreover have monic-poly R ?rhs
    using fin-T
    by (intro monic-poly-prod monic-poly-pow)
      (auto simp:monic-irreducible-poly-def)
  ultimately show ?lhs = ?rhs
    using monic-poly-not-assoc assms monic-irreducible-poly-def
    by blast
qed

lemma degree-monic-poly':
  assumes monic-poly R f
  shows
    sum' (λd. pmult d f * degree d) {d. monic-irreducible-poly R d} =
    degree f
proof −
  let ?mip = monic-irreducible-poly R

  have b: d ∈ carrier P − {0P} if ?mip d for d
    using that monic-poly-carr-2
    unfolding monic-irreducible-poly-def by simp
  have a: d [^]P n ∈ carrier P − {0P} if ?mip d for d and n :: nat
    using b that monic-poly-pow
    unfolding monic-irreducible-poly-def
    by (simp add: p.pow-non-zero)

  have degree f =
    degree (⊗P d∈{d. ?mip d ∧ pmult d f > 0}. d [^]P pmult d f)
    using factor-monic-poly[OF assms(1)] by simp
  also have ... =
    (∑ i∈{d. ?mip d ∧ 0 < pmult d f}. degree (i [^]P pmult i f))
    using a assms(1)
    by (subst degree-prod[OF factor-monic-poly-fin])
     (simp-all add:Pi-def)
  also have ... =
    (∑ i∈{d. ?mip d ∧ 0 < pmult d f}. degree i * pmult i f)
```

```
    using b degree-pow by (intro sum.cong, auto)
  also have ... =
    (∑ d∈{d. ?mip d ∧ 0 < pmult d f}. pmult d f * degree d)
    by (simp add:mult.commute)
  also have ... =
    sum' (λd. pmult d f * degree d) {d. ?mip d ∧ 0 < pmult d f}
    using sum.eq-sum factor-monic-poly-fin[OF assms(1)] by simp
  also have ... = sum' (λd. pmult d f * degree d) {d. ?mip d}
    by (intro sum.mono-neutral-cong-left' subsetI, auto)
  finally show ?thesis by simp
qed

lemma monic-poly-min-degree:
  assumes monic-irreducible-poly R f
  shows  degree f ≥ 1
  using assms unfolding monic-irreducible-poly-def monic-poly-def
  by (intro pirreducible-degree) auto

lemma degree-one-monic-poly:
  monic-irreducible-poly R f ∧ degree f = 1 ⟷
  (∃ x ∈ carrier R. f = [1, ⊖x])
proof
  assume monic-irreducible-poly R f ∧ degree f = 1
  hence a:monic-poly R f length f = 2
    unfolding monic-irreducible-poly-def by auto
  then obtain u v where f-def: f = [u,v]
    by (cases f, simp, cases tl f, auto)

  have u = 1 using a unfolding monic-poly-def f-def by simp
  moreover have v ∈ carrier R
    using a unfolding monic-poly-def univ-poly-carrier[symmetric]
    unfolding polynomial-def f-def by simp
  ultimately have f =  [1, ⊖(⊖v)] (⊖v) ∈ carrier R
    using a-inv-closed f-def by auto
  thus (∃ x ∈ carrier R. f = [1_R, ⊖_R x]) by auto
next
  assume (∃ x ∈ carrier R. f = [1, ⊖x])
  then obtain x where f-def: f = [1,⊖x] x ∈ carrier R by auto
  have a:degree f = 1 using f-def(2) unfolding f-def by simp
  have b:f ∈ carrier P
    using f-def(2) unfolding univ-poly-carrier[symmetric]
    unfolding f-def polynomial-def by simp
  have c: pirreducible (carrier R) f
    by (intro degree-one-imp-pirreducible a b)
  have d: lead-coeff f = 1 unfolding f-def by simp
  show monic-irreducible-poly R f ∧ degree f = 1
    using a b c d
    unfolding monic-irreducible-poly-def monic-poly-def
    by auto
```

qed

lemma *multiplicity-ge-iff*:
assumes *monic-irreducible-poly R d*
assumes $f \in$ *carrier* $P - \{\mathbf{0}_P\}$
shows *pmult d f* $\geq k \longleftrightarrow d\ [\widehat{}]_P\ k$ *pdivides f*
proof −
have *a*:$f \in$ *carrier* (*mult-of P*)
using *assms*(*2*) **by** *simp*
have *b*: $d \in$ *carrier* (*mult-of P*)
using *assms*(*1*) *monic-poly-carr-2*
unfolding *monic-irreducible-poly-def* **by** *simp*
have *c*: *irreducible* (*mult-of P*) *d*
using *assms*(*1*) *monic-poly-carr-2*
using *p.irreducible-imp-irreducible-mult*
unfolding *monic-irreducible-poly-def*
unfolding *ring-irreducible-def monic-poly-def*
by *simp*
have *d*: $d\ [\widehat{}]_P\ k \in$ *carrier P* **using** *b* **by** *simp*

have *pmult d f* $\geq k \longleftrightarrow d\ [\widehat{}]_{(\textit{mult-of } P)}\ k\ \textit{divides}_{(\textit{mult-of } P)}\ f$
unfolding *pmult-def*
by (*intro p.mult-of.multiplicity-ge-iff a b c*)
also have ... $\longleftrightarrow d\ [\widehat{}]_P\ k\ \textit{pdivides}_R\ f$
using *p.divides-imp-divides-mult*[*OF d assms*(*2*)]
using *divides-mult-imp-divides*
unfolding *pdivides-def nat-pow-mult-of*
by *auto*
finally show *?thesis* **by** *simp*
qed

lemma *multiplicity-ge-1-iff-pdivides*:
assumes *monic-irreducible-poly R d f* $\in$ *carrier* $P - \{\mathbf{0}_P\}$
shows *pmult d f* $\geq 1 \longleftrightarrow$ *d pdivides f*
proof −
have $d \in$ *carrier P*
using *assms*(*1*) *monic-poly-carr*
unfolding *monic-irreducible-poly-def*
by *simp*
thus *?thesis*
using *multiplicity-ge-iff*[*OF assms*, **where** *k=1*]
by *simp*
qed

lemma *divides-monic-poly*:
assumes *monic-poly R f monic-poly R g*
assumes $\bigwedge d.$ *monic-irreducible-poly R d*
$\Longrightarrow$ *pmult d f* $\leq$ *pmult d g*
shows *f pdivides g*

proof −
have *a*:*f* ∈ *carrier* (*mult-of P*) *g* ∈ *carrier* (*mult-of P*)
using *monic-poly-carr-2 assms*(*1*,*2*) **by** *auto*

have *f* $divides_{(mult\text{-}of\ P)}$ *g*
using *assms*(*3*) **unfolding** *pmult-def*
by (*intro p.mult-of.divides-iff-mult-mono*
[*OF a monic-polys-are-canonical-irreducibles*]) *simp*
thus *?thesis*
unfolding *pdivides-def* **using** *divides-mult-imp-divides* **by** *simp*
qed

end

lemma *monic-poly-hom*:
assumes *monic-poly R f*
assumes *h* ∈ *ring-iso R S domain R domain S*
shows *monic-poly S* (*map h f*)
proof −
have *c*: *h* ∈ *ring-hom R S*
using *assms*(*2*) *ring-iso-def* **by** *auto*
have *e*: *f* ∈ *carrier* (*poly-ring R*)
using *assms*(*1*) **unfolding** *monic-poly-def* **by** *simp*

have *a*:*f* ≠ []
using *assms*(*1*) **unfolding** *monic-poly-def* **by** *simp*
hence *map h f* ≠ [] **by** *simp*
moreover have *lead-coeff f* = $\mathbf{1}_R$
using *assms*(*1*) **unfolding** *monic-poly-def* **by** *simp*
hence *lead-coeff* (*map h f*) = $\mathbf{1}_S$
using *ring-hom-one*[*OF c*] **by** (*simp add*: *hd-map*[*OF a*])
ultimately show *?thesis*
using *carrier-hom*[*OF e assms*(*2*−*4*)]
unfolding *monic-poly-def* **by** *simp*
qed

lemma *monic-irreducible-poly-hom*:
assumes *monic-irreducible-poly R f*
assumes *h* ∈ *ring-iso R S domain R domain S*
shows *monic-irreducible-poly S* (*map h f*)
proof −
have *a*:
$pirreducible_R$ (*carrier R*) *f*
f ∈ *carrier* (*poly-ring R*)
monic-poly R f
using *assms*(*1*)
unfolding *monic-poly-def monic-irreducible-poly-def*
by *auto*

have *pirreducible*$_S$ (*carrier S*) (*map h f*)
using *a pirreducible-hom assms* **by** *auto*
moreover have *monic-poly S* (*map h f*)
using *a monic-poly-hom*[*OF - assms*(*2*,*3*,*4*)] **by** *simp*
ultimately show *?thesis*
unfolding *monic-irreducible-poly-def* **by** *simp*
qed

end

6 Counting Irreducible Polynomials

6.1 The polynomial $X^n - X$

theory *Card-Irreducible-Polynomials-Aux*
imports
HOL−Algebra.Multiplicative-Group
Formal-Polynomial-Derivatives
Monic-Polynomial-Factorization
begin

lemma (**in** *domain*)
assumes *subfield K R*
assumes *f* ∈ *carrier* (*K*[*X*]) *degree f* > *0*
shows *embed-inj*: *inj-on* (*rupture-surj K f* ∘ *poly-of-const*) *K*
and *rupture-order*: *order* (*Rupt K f*) = *card K*^*degree f*
and *rupture-char*: *char* (*Rupt K f*) = *char R*
proof −
interpret *p*: *principal-domain K*[*X*]
using *univ-poly-is-principal*[*OF assms*(*1*)] **by** *simp*

interpret *I*: *ideal PIdl*$_{K[X]}$ *f K*[*X*]
using *p.cgenideal-ideal*[*OF assms*(*2*)] **by** *simp*

interpret *d*: *ring Rupt K f*
unfolding *rupture-def* **using** *I.quotient-is-ring* **by** *simp*

have *e*: *subring K R*
using *assms*(*1*) *subfieldE*(*1*) **by** *auto*

interpret *h*:
ring-hom-ring R (| *carrier* := *K* |)
Rupt K f rupture-surj K f ∘ *poly-of-const*
using *rupture-surj-norm-is-hom*[*OF e assms*(*2*)]
using *ring-hom-ringI2 subring-is-ring d.is-ring e*
by *blast*

have *field* (*R* (|*carrier* := *K*|))
using *assms*(*1*) *subfield-iff*(*2*) **by** *simp*

hence *subfield K* (*R*(|*carrier* := *K*|))
using *ring.subfield-iff*[*OF subring-is-ring*[*OF e*]] **by** *simp*
hence *b*: *subfield* (*rupture-surj K f* ‘ *poly-of-const* ‘ *K*) (*Rupt K f*)
unfolding *image-image comp-def*[*symmetric*]
by (*intro h.img-is-subfield rupture-one-not-zero assms*, *simp*)

have *inj-on poly-of-const K*
using *poly-of-const-inj inj-on-subset* **by** *auto*
moreover have
poly-of-const ‘ *K* ⊆ ((λ*q*. *q pmod f*) ‘ *carrier* (*K* [*X*]))
proof (*rule image-subsetI*)
fix *x* **assume** *x* ∈ *K*
hence *f*:
poly-of-const x ∈ *carrier* (*K*[*X*])
degree (*poly-of-const x*) = *0*
using *poly-of-const-over-subfield*[*OF assms*(*1*)] **by** *auto*
moreover
have *degree* (*poly-of-const x*) < *degree f*
using *f*(*2*) *assms* **by** *simp*
hence *poly-of-const x pmod f* = *poly-of-const x*
by (*intro pmod-const*(*2*)[*OF assms*(*1*)] *f assms*(*2*), *simp*)
ultimately show
poly-of-const x ∈ ((λ*q*. *q pmod f*) ‘ *carrier* (*K* [*X*]))
by *force*
qed
hence *inj-on* (*rupture-surj K f*) (*poly-of-const* ‘ *K*)
using *rupture-surj-inj-on*[*OF assms*(*1*,*2*)] *inj-on-subset* **by** *blast*
ultimately show *d*: *inj-on* (*rupture-surj K f* ∘ *poly-of-const*) *K*
using *comp-inj-on* **by** *auto*

have *a*: *d.dimension* (*degree f*) (*rupture-surj K f* ‘ *poly-of-const* ‘ *K*)

(*carrier* (*Rupt K f*))
using *rupture-dimension*[*OF assms*(*1*−*3*)] **by** *auto*
then obtain *base* **where** *base-def*:
set base ⊆ *carrier* (*Rupt K f*)
d.independent (*rupture-surj K f* ‘ *poly-of-const* ‘ *K*) *base*
length base = *degree f*
d.Span (*rupture-surj K f* ‘ *poly-of-const* ‘ *K*) *base* =
carrier (*Rupt K f*)
using *d.exists-base*[*OF b a*] **by** *auto*
have *order* (*Rupt K f*) =
card (*d.Span* (*rupture-surj K f* ‘ *poly-of-const* ‘ *K*) *base*)
unfolding *order-def base-def*(*4*) **by** *simp*
also have ... =
card (*rupture-surj K f* ‘ *poly-of-const* ‘ *K*) ^ *length base*
using *d.card-span*[*OF b base-def*(*2*,*1*)] **by** *simp*
also have ...
= *card* ((*rupture-surj K f* ∘ *poly-of-const*) ‘ *K*) ^ *degree f*

```
    using base-def(3) image-image unfolding comp-def by metis
  also have ... = card K^degree f
    by (subst card-image[OF d], simp)
  finally show order (Rupt K f) = card K^degree f by simp

  have char (Rupt K f) = char (R (| carrier := K |))
    using h.char-consistent d by simp
  also have ... = char R
    using char-consistent[OF subfieldE(1)[OF assms(1)]] by simp
  finally show char (Rupt K f) = char R by simp
qed

definition gauss-poly where
  gauss-poly K n = X_K [^]poly-ring K (n::nat) ⊖poly-ring K X_K

context field
begin

interpretation polynomial-ring R carrier R
  unfolding polynomial-ring-def polynomial-ring-axioms-def
  using field-axioms carrier-is-subfield by simp
```

The following lemma can be found in Ireland and Rosen [3, §7.1, Lemma 2].

```
lemma gauss-poly-div-gauss-poly-iff-1:
  fixes l m :: nat
  assumes l > 0
  shows (X [^]P l ⊖P 1P) pdivides (X [^]P m ⊖P 1P) ⟷ l dvd m
    (is ?lhs ⟷ ?rhs)
proof −
  define q where q = m div l
  define r where r = m mod l
  have m-def: m = q * l + r and r-range: r < l
    using assms by (auto simp add:q-def r-def)

  have pow-sum-carr:(⨁P i∈{..<q}. (X [^]P l)[^]P i) ∈ carrier P
    using var-pow-closed
    by (intro p.finsum-closed, simp)

  have (X [^]P (q*l) ⊖P 1P) = ((X [^]P l)[^]P q) ⊖P 1P
    using var-closed
    by (subst p.nat-pow-pow, simp-all add:algebra-simps)
  also have ... =
    (X [^]P l ⊖P 1P) ⊗P (⨁P i∈{..<q}. (X [^]P l) [^]P i)
    using var-pow-closed
    by (subst p.geom[symmetric], simp-all)
  finally have pow-sum-fact: (X [^]P (q*l) ⊖P 1P) =
    (X [^]P l ⊖P 1P) ⊗P (⨁P i∈{..<q}. (X_R [^]P l) [^]P i)
    by simp
```

have $(X\ [\widehat{}]_P\ l \ominus_P \mathbf{1}_P)$ *divides*$_P$ $(X\ [\widehat{}]_P\ (q{*}l) \ominus_P \mathbf{1}_P)$
by (*rule dividesI*[*OF pow-sum-carr pow-sum-fact*])

hence *c*:$(X\ [\widehat{}]_P\ l \ominus_P \mathbf{1}_P)$ *divides*$_P$ $X\ [\widehat{}]_P\ r \otimes_P (X\ [\widehat{}]_P\ (q * l) \ominus_P \mathbf{1}_P)$
using *var-pow-closed*
by (*intro p.divides-prod-l*, *auto*)

have $(X\ [\widehat{}]_P\ m \ominus_P \mathbf{1}_P) = X\ [\widehat{}]_P\ (r + q * l) \ominus_P \mathbf{1}_P$
unfolding *m-def* **using** *add.commute* **by** *metis*
also have ... $= (X\ [\widehat{}]_P\ r) \otimes_P (X\ [\widehat{}]_P\ (q{*}l)) \oplus_P (\ominus_P \mathbf{1}_P)$
using *var-closed*
by (*subst p.nat-pow-mult*, *auto simp add*:*a-minus-def*)
also have ... $= ((X\ [\widehat{}]_P\ r) \otimes_P (X\ [\widehat{}]_P\ (q{*}l) \oplus_P (\ominus_P \mathbf{1}_P))$
$\oplus_P (X\ [\widehat{}]_P\ r)) \ominus_P \mathbf{1}_P$
using *var-pow-closed*
by *algebra*
also have ... $= (X\ [\widehat{}]_P\ r) \otimes_P (X\ [\widehat{}]_P\ (q{*}l) \ominus_P \mathbf{1}_P)$
$\oplus_P (X\ [\widehat{}]_P\ r) \ominus_P \mathbf{1}_P$
by *algebra*
also have ... $= (X\ [\widehat{}]_P\ r) \otimes_P (X\ [\widehat{}]_P\ (q{*}l) \ominus_P \mathbf{1}_P)$
$\oplus_P ((X\ [\widehat{}]_P\ r) \ominus_P \mathbf{1}_P)$
unfolding *a-minus-def* **using** *var-pow-closed*
by (*subst p.a-assoc*, *auto*)
finally have *a*:$(X\ [\widehat{}]_P\ m \ominus_P \mathbf{1}_P) =$
$(X\ [\widehat{}]_P\ r) \otimes_P (X\ [\widehat{}]_P\ (q{*}l) \ominus_P \mathbf{1}_P) \oplus_P (X\ [\widehat{}]_P\ r \ominus_P \mathbf{1}_P)$
(**is** - = *?x*)
by *simp*

have *xn-m-1-deg'*: *degree* $(X\ [\widehat{}]_P\ n \ominus_P \mathbf{1}_P) = n$
if $n > 0$ **for** n :: *nat*
proof −
have *degree* $(X\ [\widehat{}]_P\ n \ominus_P \mathbf{1}_P)$ = *degree* $(X\ [\widehat{}]_P\ n \oplus_P \ominus_P \mathbf{1}_P)$
by (*simp add*:*a-minus-def*)
also have ... = *max* (*degree* $(X\ [\widehat{}]_P\ n)$) (*degree* $(\ominus_P \mathbf{1}_P)$)
using *var-pow-closed var-pow-carr var-pow-degree*
using *univ-poly-a-inv-degree degree-one that*
by (*subst degree-add-distinct*, *auto*)
also have ... = n
using *var-pow-degree degree-one univ-poly-a-inv-degree*
by *simp*
finally show *?thesis* **by** *simp*
qed

have *xn-m-1-deg*: *degree* $(X\ [\widehat{}]_P\ n \ominus_P \mathbf{1}_P) = n$ **for** n :: *nat*
proof (*cases* $n > 0$)
case *True*
then show *?thesis* **using** *xn-m-1-deg'* **by** *auto*

```
  next
    case False
    hence n = 0 by simp
    hence degree (X [^]_P n ⊖_P 1_P) = degree (0_P)
      by (intro arg-cong[where f=degree], simp)
    then show ?thesis using False by (simp add:univ-poly-zero)
  qed

  have b: degree (X [^]_P l ⊖_P 1_P) > degree (X_R [^]_P r ⊖_P 1_P)
    using r-range unfolding xn-m-1-deg by simp

  have xn-m-1-carr: X [^]_P n ⊖_P 1_P ∈ carrier P for n :: nat
    unfolding a-minus-def
    by (intro p.a-closed var-pow-closed, simp)

  have ?lhs ⟷ (X [^]_P l ⊖_P 1_P) pdivides ?x
    by (subst a, simp)
  also have ... ⟷ (X [^]_P l ⊖_P 1_P) pdivides (X [^]_P r ⊖_P 1_P)
    unfolding pdivides-def
    by (intro p.div-sum-iff c var-pow-closed
        xn-m-1-carr p.a-closed p.m-closed)
  also have ... ⟷ r = 0
  proof (cases r = 0)
    case True
    have (X [^]_P l ⊖_P 1_P) pdivides 0_P
      unfolding univ-poly-zero
      by (intro pdivides-zero xn-m-1-carr)
    also have ... = (X [^]_P r ⊖_P 1_P)
      by (simp add:a-minus-def True) algebra
    finally show ?thesis using True by simp
  next
    case False
    hence degree (X [^]_P r ⊖_P 1_P) > 0 using xn-m-1-deg by simp
    hence X [^]_P r ⊖_P 1_P ≠ [] by auto
    hence ¬(X [^]_P l ⊖_P 1_P) pdivides (X [^]_P r ⊖_P 1_P)
      using pdivides-imp-degree-le b xn-m-1-carr
      by (metis le-antisym less-or-eq-imp-le nat-neq-iff)
    thus ?thesis using False by simp
  qed
  also have ... ⟷ l dvd m
    unfolding m-def using r-range assms by auto
  finally show ?thesis
    by simp
qed

lemma gauss-poly-factor:
  assumes n > 0
  shows gauss-poly R n = (X [^]_P (n−1) ⊖_P 1_P) ⊗_P X (is - = ?rhs)
proof −
```

```
  have a:1 + (n − 1) = n
    using assms by simp
  have gauss-poly R n = X [^]_P (1+(n−1)) ⊖_P X
    unfolding gauss-poly-def by (subst a, simp)
  also have ... = (X [^]_P (n−1)) ⊗_P X ⊖_P 1_P ⊗_P X
    using var-closed by simp
  also have ... = ?rhs
    unfolding a-minus-def using var-closed l-one
    by (subst p.l-distr, auto, algebra)
  finally show ?thesis by simp
qed

lemma var-neq-zero: X ≠ 0_P
  by (simp add:var-def univ-poly-zero)

lemma var-pow-eq-one-iff: X [^]_P k = 1_P ⟷ k = (0::nat)
proof (cases k=0)
  case True
  then show ?thesis using var-closed(1) by simp
next
  case False
  have degree (X_R [^]_P k) = k
    using var-pow-degree by simp
  also have ... ≠ degree (1_P) using False degree-one by simp
  finally have degree (X_R [^]_P k) ≠ degree 1_P by simp
  then show ?thesis by auto
qed

lemma gauss-poly-carr: gauss-poly R n ∈ carrier P
  using var-closed(1)
  unfolding gauss-poly-def by simp

lemma gauss-poly-degree:
  assumes n > 1
  shows degree (gauss-poly R n) = n
proof −
  have degree (gauss-poly R n) = max n 1
    unfolding gauss-poly-def a-minus-def
    using var-pow-carr var-carr degree-var
    using var-pow-degree univ-poly-a-inv-degree
    using assms by (subst degree-add-distinct, auto)
  also have ... = n using assms by simp
  finally show ?thesis by simp
qed

lemma gauss-poly-not-zero:
  assumes n > 1
  shows gauss-poly R n ≠ 0_P
proof −
```

have *degree* (*gauss-poly R n*) ≠ *degree* ($\mathbf{0}_P$)
using *assms* **by** (*subst gauss-poly-degree*, *simp-all add*:*univ-poly-zero*)
thus *?thesis* **by** *auto*
qed

lemma *gauss-poly-monic*:
assumes *n* > *1*
shows *monic-poly R* (*gauss-poly R n*)
proof −
have *monic-poly R* (*X* [^]$_P$ *n*)
by (*intro monic-poly-pow monic-poly-var*)
moreover have ⊖$_P$ *X* ∈ *carrier P*
using *var-closed* **by** *simp*
moreover have *degree* (⊖$_P$ *X*) < *degree* (*X* [^]$_P$ *n*)
using *assms univ-poly-a-inv-degree var-closed*
using *degree-var*
unfolding *var-pow-degree* **by** (*simp*)
ultimately show *?thesis*
unfolding *gauss-poly-def a-minus-def*
by (*intro monic-poly-add-distinct*, *auto*)
qed

lemma *geom-nat*:
fixes *q* :: *nat*
fixes *x* :: - :: {*comm-ring*,*monoid-mult*}
shows (*x*−*1*) * (∑ *i* ∈ {..<*q*}. *x*^*i*) = *x*^*q*−*1*
by (*induction q*, *auto simp*:*algebra-simps*)

The following lemma can be found in Ireland and Rosen [3, §7.1, Lemma 3].

lemma *gauss-poly-div-gauss-poly-iff-2*:
fixes *a* :: *int*
fixes *l m* :: *nat*
assumes *l* > *0 a* > *1*
shows (*a* ^ *l* − *1*) *dvd* (*a* ^ *m* − *1*) ⟷ *l dvd m*
(**is** *?lhs* ⟷ *?rhs*)
proof −
define *q* **where** *q* = *m div l*
define *r* **where** *r* = *m mod l*
have *m-def*: *m* = *q* * *l* + *r* **and** *r-range*: *r* < *l r* ≥ *0*
using *assms* **by** (*auto simp add*:*q-def r-def*)

have *a* ^ (*l* * *q*) − *1* = (*a* ^ *l*) ^ *q* − *1*
by (*simp add*: *power-mult*)
also have ... = (*a*^*l* − *1*) * (∑ *i* ∈ {..<*q*}. (*a*^*l*)^*i*)
by (*subst geom-nat*[*symmetric*], *simp*)
finally have *a* ^ (*l* * *q*) − *1* = (*a*^*l* − *1*) * (∑ *i* ∈ {..<*q*}. (*a*^*l*)^*i*)
by *simp*
hence *c*:*a* ^ *l* − *1 dvd a*^*r* * (*a* ^ (*q* * *l*) − *1*) **by** (*simp add*:*mult.commute*)

```
  have a ^ m - 1 = a ^ (r + q * l) - 1
    unfolding m-def using add.commute by metis
  also have ... = (a ^ r) * (a ^ (q*l)) -1
    by (simp add: power-add)
  also have ... = ((a ^ r) * (a ^ (q*l) -1)) + (a ^ r) - 1
    by (simp add: right-diff-distrib)
  also have ... = (a ^ r) * (a ^ (q*l) - 1) + ((a ^ r) - 1)
    by simp
  finally have a:
    a ^ m - 1 = (a ^ r) * (a ^ (q*l) - 1) + ((a ^ r) - 1)
    (is - = ?x)
    by simp

  have ?lhs ⟷ (a^l -1) dvd ?x
    by (subst a, simp)
  also have ... ⟷ (a^l -1) dvd (a^r -1)
    using c dvd-add-right-iff by auto
  also have ... ⟷ r = 0
  proof
    assume a ^ l - 1 dvd a ^ r - 1
    hence a ^ l - 1 ≤ a ^ r -1 ∨ r = 0
      using assms r-range zdvd-not-zless by force
    moreover have a ^ r < a^l using assms r-range by simp
    ultimately show r= 0by simp
  next
    assume r = 0
    thus a ^ l - 1 dvd a ^ r - 1 by simp
  qed
  also have ... ⟷ l dvd m
    using r-def by auto
  finally show ?thesis by simp
qed

lemma gauss-poly-div-gauss-poly-iff:
  assumes m > 0 n > 0 a > 1
  shows gauss-poly R (a^n) pdivides_R gauss-poly R (a^m)
    ⟷ n dvd m (is ?lhs=?rhs)
proof -
  have a:a^m > 1 using assms one-less-power by blast
  hence a1: a^m > 0 by linarith
  have b:a^n > 1 using assms one-less-power by blast
  hence b1:a^n > 0 by linarith

  have ?lhs ⟷
    (X [^]_P (a^n-1) ⊖_P 1_P) ⊗_P X pdivides
    (X [^]_P (a^m-1) ⊖_P 1_P) ⊗_P X
    using  gauss-poly-factor a1 b1 by simp
  also have ... ⟷
```

```
    (X [^]_P (a^n−1) ⊖_P 1_P) pdivides
    (X [^]_P (a^m−1) ⊖_P 1_P)
    using var-closed a b var-neq-zero
    by (subst pdivides-mult-r, simp-all add:var-pow-eq-one-iff)
  also have ... ⟷ a^n−1 dvd a^m−1
    using b
    by (subst gauss-poly-div-gauss-poly-iff-1) simp-all
  also have ... ⟷ int (a^n−1) dvd int (a^m−1)
    by (subst of-nat-dvd-iff, simp)
  also have ... ⟷ int a^n−1 dvd int a^m−1
    using a b by (simp add:of-nat-diff)
  also have ... ⟷ n dvd m
    using assms
    by (subst gauss-poly-div-gauss-poly-iff-2) simp-all
  finally show ?thesis by simp
qed

end

context finite-field
begin

interpretation polynomial-ring R carrier R
  unfolding polynomial-ring-def polynomial-ring-axioms-def
  using field-axioms carrier-is-subfield by simp

lemma div-gauss-poly-iff:
  assumes n > 0
  assumes monic-irreducible-poly R f
  shows f pdivides_R gauss-poly R (order R^n) ⟷ degree f dvd n
proof −
  have f-carr: f ∈ carrier P
    using assms(2) unfolding monic-irreducible-poly-def
    unfolding monic-poly-def by simp
  have f-deg: degree f > 0
    using assms(2) monic-poly-min-degree by fastforce

  define K where K = Rupt_R (carrier R) f
  have field-K: field K
    using assms(2) unfolding K-def monic-irreducible-poly-def
    unfolding monic-poly-def
    by (subst rupture-is-field-iff-pirreducible) auto
  have a: order K = order R^degree f
    using rupture-order[OF carrier-is-subfield] f-carr f-deg
    unfolding K-def order-def by simp
  have char-K: char K = char R
    using rupture-char[OF carrier-is-subfield] f-carr f-deg
    unfolding K-def by simp
```

have *card* (*carrier K*) > *0*
  using *a f-deg finite-field-min-order* **unfolding** *order-def* **by** *simp*
hence *d*: *finite* (*carrier K*) **using** *card-ge-0-finite* **by** *auto*
interpret *f*: *finite-field K*
  using *field-K d* **by** (*intro finite-fieldI*, *simp-all*)
interpret *fp*: *polynomial-ring K* (*carrier K*)
  unfolding *polynomial-ring-def polynomial-ring-axioms-def*
  using *f.field-axioms f.carrier-is-subfield* **by** *simp*

define φ **where** φ = *rupture-surj* (*carrier R*) *f*
interpret *h*:*ring-hom-ring P K* φ
  unfolding *K-def* φ*-def* **using** *f-carr rupture-surj-hom* **by** *simp*

have *embed-inj*: *inj-on* (φ ∘ *poly-of-const*) (*carrier R*)
  unfolding φ*-def*
  using *embed-inj*[*OF carrier-is-subfield f-carr f-deg*] **by** *simp*

interpret *r*:*ring-hom-ring R P poly-of-const*
  using *canonical-embedding-ring-hom* **by** *simp*

obtain *rn* **where** *order R* = *char K*^*rn rn* > *0*
  unfolding *char-K* **using** *finite-field-order* **by** *auto*
hence *ord-rn*: *order R* ^*n* = *char K*^(*rn* ∗ *n*) **using** *assms*(*1*)
  by (*simp add*: *power-mult*)

interpret *q*:*ring-hom-cring K K* λ*x*. *x* [^]$_K$ *order R*^*n*
  using *ord-rn*
  by (*intro f.frobenius-hom f.finite-carr-imp-char-ge-0 d*, *simp*)

have *o1*: *order R*^*degree f* > *1*
  using *f-deg finite-field-min-order one-less-power*
  by *blast*
hence *o11*: *order R*^*degree f* > *0* **by** *linarith*
have *o2*: *order R*^*n* > *1*
  using *assms*(*1*) *finite-field-min-order one-less-power*
  by *blast*
hence *o21*: *order R*^*n* > *0* **by** *linarith*
let *?g1* = *gauss-poly K* (*order R*^*degree f*)
let *?g2* = *gauss-poly K* (*order R*^*n*)

have *g1-monic*: *monic-poly K ?g1*
  using *f.gauss-poly-monic*[*OF o1*] **by** *simp*

have *c*:*x* [^]$_K$ (*order R*^*degree f*) = *x* **if** *b*:*x* ∈ *carrier K* **for** *x*
  using *b d order-pow-eq-self*
  unfolding *a*[*symmetric*]
  by (*intro f.order-pow-eq-self*, *auto*)

have *k-cycle*:

```
  φ (poly-of-const x) [^]_K (order R^n) = φ(poly-of-const x)
  if k-cycle-1: x ∈ carrier R for x
proof −
  have φ (poly-of-const x) [^]_K (order R^n) =
    φ (poly-of-const (x [^]_R (order R^n)))
    using k-cycle-1 by (simp add: h.hom-nat-pow r.hom-nat-pow)
  also have ... = φ (poly-of-const x)
    using order-pow-eq-self' k-cycle-1 by simp
  finally show ?thesis by simp
qed

have roots-g1: pmult_K d ?g1 ≥ 1
  if roots-g1-assms: degree d = 1 monic-irreducible-poly K d for d
proof −
  obtain x where x-def: x ∈ carrier K d = [1_K, ⊖_K x]
    using f.degree-one-monic-poly roots-g1-assms by auto
  interpret x:ring-hom-cring poly-ring K K (λp. f.eval p x)
    by (intro fp.eval-cring-hom x-def)
  have ring.eval K ?g1 x = 0_K
    unfolding gauss-poly-def a-minus-def
    using fp.var-closed f.eval-var x-def c
    by (simp, algebra)
  hence f.is-root ?g1 x
    using x-def f.gauss-poly-not-zero[OF o1]
    unfolding f.is-root-def univ-poly-zero by simp
  hence [1_K, ⊖_K x] pdivides_K ?g1
    using f.is-root-imp-pdivides f.gauss-poly-carr by simp
  hence d pdivides_K ?g1 by (simp add:x-def)
  thus pmult_K d ?g1 ≥ 1
    using that f.gauss-poly-not-zero f.gauss-poly-carr o1
    by (subst f.multiplicity-ge-1-iff-pdivides, simp-all)
qed

show ?thesis
proof
  assume f:f pdivides_R gauss-poly R (order R^n)
  have (φ X) [^]_K (order R^n) ⊖_K (φ X_R) =
    φ (gauss-poly R (order R^n))
    unfolding gauss-poly-def a-minus-def using var-closed
    by (simp add: h.hom-nat-pow)
  also have ... = 0_K
    unfolding K-def φ-def using f-carr gauss-poly-carr f
    by (subst rupture-eq-0-iff, simp-all)
  finally have (φ X_R) [^]_K (order R^n) ⊖_K (φ X_R) = 0_K
    by simp
  hence g:(φ X) [^]_K (order R^n) = (φ X)
    using var-closed by simp

  have roots-g2: pmult_K d ?g2 ≥ 1
```

```
  if roots-g2-assms: degree d = 1 monic-irreducible-poly K d for d
proof −
  obtain y where y-def: y ∈ carrier K d = [1_K, ⊖_K y]
    using f.degree-one-monic-poly roots-g2-assms by auto

  interpret x:ring-hom-cring poly-ring K K (λp. f.eval p y)
    by (intro fp.eval-cring-hom y-def)
  obtain x where x-def: x ∈ carrier P y = φ x
    using y-def unfolding φ-def K-def rupture-def
    unfolding FactRing-def A-RCOSETS-def'
    by auto
  let ?τ  = λi. poly-of-const (coeff x i)
  have test: ?τ i ∈ carrier P for i
    by (intro r.hom-closed coeff-range x-def)
  have test-2: coeff x i ∈ carrier R for i
    by (intro coeff-range x-def)

  have x-coeff-carr: i ∈ set x ⟹ i ∈ carrier R  for i
    using x-def(1)
   by (auto simp add:univ-poly-carrier[symmetric] polynomial-def)

  have a:map (φ ∘ poly-of-const) x ∈ carrier (poly-ring K)
    using rupture-surj-norm-is-hom[OF f-carr]
    using domain-axioms f.domain-axioms embed-inj
    by (intro carrier-hom'[OF  x-def(1)])
     (simp-all add:φ-def K-def)

  have (φ x) [^]_K (order R^n) =
    f.eval (map (φ ∘ poly-of-const) x) (φ X) [^]_K (order R^n)
    unfolding φ-def K-def
    by (subst rupture-surj-as-eval[OF f-carr x-def(1)], simp)
  also have ... =
   f.eval (map (λx. φ (poly-of-const x) [^]_K order R ^ n) x) (φ X)
    using a h.hom-closed var-closed(1)
    by (subst q.ring.eval-hom[OF f.carrier-is-subring])
      (simp-all add:comp-def g)
  also have ... = f.eval (map (λx. φ (poly-of-const x)) x) (φ X)
    using k-cycle x-coeff-carr
    by (intro arg-cong2[where f=f.eval] map-cong, simp-all)
  also have ... = (φ x)
    unfolding φ-def K-def
  by (subst rupture-surj-as-eval[OF f-carr x-def(1)], simp add:comp-def)
  finally have φ x [^]_K order R ^ n = φ x by simp

  hence y [^]_K (order R^n) = y using x-def by simp
  hence ring.eval K ?g2 y = 0_K
    unfolding gauss-poly-def a-minus-def
    using fp.var-closed f.eval-var y-def
    by (simp, algebra)
```

```
    hence f.is-root ?g2 y
      using y-def f.gauss-poly-not-zero[OF o2]
      unfolding f.is-root-def univ-poly-zero by simp
    hence d pdivides_K ?g2
      unfolding y-def
      by (intro f.is-root-imp-pdivides f.gauss-poly-carr, simp)
    thus pmult_K d ?g2 ≥ 1
      using that f.gauss-poly-carr f.gauss-poly-not-zero o2
      by (subst f.multiplicity-ge-1-iff-pdivides, auto)
  qed

  have inv-k-inj: inj-on (λx. ⊖_K x) (carrier K)
    by (intro inj-onI, metis f.minus-minus)
  let ?mip = monic-irreducible-poly K

  have sum' (λd. pmult_K d ?g1 * degree d) {d. ?mip d} = degree
?g1
    using f.gauss-poly-monic o1
    by (subst f.degree-monic-poly', simp-all)
  also have ... = order K
    using f.gauss-poly-degree o1 a by simp
  also have ... = card ((λk. [1_K, ⊖_K k]) ` carrier K)
    unfolding order-def using inj-onD[OF inv-k-inj]
    by (intro card-image[symmetric] inj-onI) (simp-all)
  also have ... = card {d. ?mip d ∧ degree d = 1}
    using f.degree-one-monic-poly
    by (intro arg-cong[where f=card], simp add:set-eq-iff image-iff)

  also have ... = sum (λd. 1) {d. ?mip d ∧ degree d = 1}
    by simp
  also have ... = sum' (λd. 1) {d. ?mip d ∧ degree d = 1}
    by (intro sum.eq-sum[symmetric]
        finite-subset[OF - fp.finite-poly(1)[OF d]])
     (auto simp:monic-irreducible-poly-def monic-poly-def)
  also have ... = sum' (λd. of-bool (degree d = 1)) {d. ?mip d}
    by (intro sum.mono-neutral-cong-left' subsetI, simp-all)
  also have ... ≤ sum' (λd. of-bool (degree d = 1)) {d. ?mip d}
    by simp
  finally have sum' (λd. pmult_K d ?g1 * degree d) {d. ?mip d}
    ≤ sum' (λd. of-bool (degree d = 1)) {d. ?mip d}
    by simp
  moreover have
    pmult_K d ?g1 * degree d ≥ of-bool (degree d = 1)
    if v:monic-irreducible-poly K d for d
  proof (cases degree d = 1)
    case True
    then obtain x where x ∈ carrier K d = [1_K, ⊖_K x]
      using f.degree-one-monic-poly v by auto
    hence pmult_K d ?g1 ≥ 1
```

```
    using roots-g1 v by simp
  then show ?thesis using True by simp
 next
  case False
  then show ?thesis by simp
 qed
 moreover have
  finite {d. ?mip d ∧ pmult_K d ?g1 ∗ degree d > 0}
  by (intro finite-subset[OF - f.factor-monic-poly-fin[OF g1-monic]]
      subsetI) simp
 ultimately have v2:
  ∀ d ∈ {d. ?mip d}. pmult_K d ?g1 ∗ degree d =
  of-bool (degree d = 1)
  by (intro sum'-eq-iff, simp-all add:not-le)
 have pmult_K d ?g1 ≤ pmult_K d ?g2 if ?mip d for d
 proof (cases degree d = 1)
  case True
  hence pmult_K d ?g1 = 1 using v2 that by simp
  also have ... ≤ pmult_K d ?g2
   by (intro roots-g2 True that)
  finally show ?thesis by simp
 next
  case False
  hence degree d > 1
   using f.monic-poly-min-degree[OF that] by simp
  hence pmult_K d ?g1 = 0 using v2 that by auto
  then show ?thesis by simp
 qed
 hence ?g1 pdivides_K ?g2
  using o1 o2 f.divides-monic-poly f.gauss-poly-monic by simp
 thus degree f dvd n
  by (subst (asm) f.gauss-poly-div-gauss-poly-iff
      [OF assms(1) f-deg finite-field-min-order], simp)
next
 have d:φ X_R ∈ carrier K
  by (intro h.hom-closed var-closed)

 have φ (gauss-poly R (order R^degree f)) =
  (φ X_R) [^]_K (order R^degree f) ⊖_K (φ X_R)
  unfolding gauss-poly-def a-minus-def using var-closed
  by (simp add: h.hom-nat-pow)
 also have ... = 0_K
  using c d by simp
 finally have φ (gauss-poly R (order R^degree f)) = 0_K by simp
 hence f pdivides_R gauss-poly R (order R^degree f)
  unfolding K-def φ-def using f-carr gauss-poly-carr
  by (subst (asm) rupture-eq-0-iff, simp-all)
 moreover assume degree f dvd n
```

```
    hence gauss-poly R (order R^degree f) pdivides
      (gauss-poly R (order R^n))
      using gauss-poly-div-gauss-poly-iff
        [OF assms(1) f-deg finite-field-min-order]
      by simp
    ultimately show f pdivides_R gauss-poly R (order R^n)
      using f-carr a p.divides-trans unfolding pdivides-def by blast
  qed
qed

lemma gauss-poly-splitted:
  splitted (gauss-poly R (order R))
proof −
  have degree q ≤ 1 if
    q ∈ carrier P
    pirreducible (carrier R) q
    q pdivides gauss-poly R (order R) for q
  proof −
    have q-carr: q ∈ carrier (mult-of P)
      using that unfolding ring-irreducible-def by simp
    moreover have irreducible (mult-of P) q
      using that unfolding ring-irreducible-def
      by (intro p.irreducible-imp-irreducible-mult that, simp-all)
    ultimately obtain p where p-def:
      monic-irreducible-poly R p q ∼_mult-of P p
      using monic-poly-span by auto
    have p-carr: p ∈ carrier P p ≠ []
      using p-def(1)
      unfolding monic-irreducible-poly-def monic-poly-def
      by auto
    moreover have p divides_mult-of P q
      using associatedE[OF p-def(2)] by auto
    hence p pdivides q
      unfolding pdivides-def using divides-mult-imp-divides by simp
    moreover have q pdivides gauss-poly R (order R^1)
      using that by simp
    ultimately have p pdivides gauss-poly R (order R^1)
      unfolding pdivides-def using p.divides-trans by blast
    hence degree p dvd 1
      using  div-gauss-poly-iff[where n=1] p-def(1) by simp
    hence degree p = 1 by simp
    moreover have q divides_mult-of P p
      using associatedE[OF p-def(2)] by auto
    hence q pdivides p
      unfolding pdivides-def using divides-mult-imp-divides by simp
    hence degree q ≤ degree p
      using that p-carr
      by (intro pdivides-imp-degree-le) auto
    ultimately show ?thesis by simp
```

qed

thus *?thesis*
using *gauss-poly-carr*
by (*intro trivial-factors-imp-splitted*, *auto*)
qed

The following lemma, for the case when R is a simple prime field, can be found in Ireland and Rosen [3, §7.1, Theorem 2]. Here the result is verified even for arbitrary finite fields.

lemma *multiplicity-of-factor-of-gauss-poly*:
assumes $n > 0$
assumes *monic-irreducible-poly R f*
shows
$pmult_R\ f\ (gauss\text{-}poly\ R\ (order\ R\hat{}n)) = of\text{-}bool\ (degree\ f\ dvd\ n)$
proof (*cases degree f dvd n*)
case *True*
let *?g* = *gauss-poly R* (*order R*$\hat{}$*n*)
have *f-carr*: $f \in carrier\ P\ f \neq []$
using *assms*(*2*)
unfolding *monic-irreducible-poly-def monic-poly-def*
by *auto*

have *o2*: *order R*$\hat{}$*n* > *1*
using *finite-field-min-order assms*(*1*) *one-less-power* **by** *blast*
hence *o21*: *order R*$\hat{}$*n* > *0* **by** *linarith*

obtain *d* :: *nat* **where** *order-dim*: *order R* = *char R* ^ *d d* > *0*
using *finite-field-order* **by** *blast*
have *d* ∗ *n* > *0* **using** *order-dim assms* **by** *simp*
hence *char-dvd-order*: *int* (*char R*) *dvd int* (*order R* ^ *n*)
unfolding *order-dim*
using *finite-carr-imp-char-ge-0*[*OF finite-carrier*]
by (*simp add*:*power-mult*[*symmetric*])

interpret *h*: *ring-hom-ring R P poly-of-const*
using *canonical-embedding-ring-hom* **by** *simp*

have $f\ pdivides_R\ ?g$
using *True div-gauss-poly-iff*[*OF assms*] **by** *simp*
hence $pmult_R\ f\ ?g \geq 1$
using *multiplicity-ge-1-iff-pdivides*[*OF assms*(*2*)]
using *gauss-poly-carr gauss-poly-not-zero*[*OF o2*]
by *auto*
moreover have $pmult_R\ f\ ?g < 2$
proof (*rule ccontr*)
assume ¬ $pmult_R\ f\ ?g < 2$
hence $pmult_R\ f\ ?g \geq 2$ **by** *simp*
hence $(f\ [\hat{}]_P\ (2::nat))\ pdivides_R\ ?g$

using *gauss-poly-carr gauss-poly-not-zero*[*OF o2*]
by (*subst* (*asm*) *multiplicity-ge-iff*[*OF assms*(*2*)]) *simp-all*
hence (*f* [^]$_P$ (*2*::*nat*)) *divides*$_{mult\text{-}of\ P}$ *?g*
unfolding *pdivides-def*
using *f-carr gauss-poly-not-zero o2 gauss-poly-carr*
by (*intro p.divides-imp-divides-mult*) *simp-all*
then obtain *h* **where** *h-def*:
h ∈ *carrier* (*mult-of P*)
?g = *f* [^]$_P$ (*2*::*nat*) ⊗$_P$ *h*
using *dividesD* **by** *auto*
have ⊖$_P$ **1**$_P$ = *int-embed P* (*order R* ^ *n*)
⊗$_P$ (*X*$_R$ [^]$_P$ (*order R* ^ *n*−*1*)) ⊖$_P$ **1**$_P$
using *var-closed*
apply (*subst int-embed-consistent-with-poly-of-const*)
apply (*subst iffD2*[*OF embed-char-eq-0-iff char-dvd-order*])
by (*simp add*:*a-minus-def*)
also have ... = *pderiv*$_R$ (*X*$_R$ [^]$_P$ *order R* ^ *n*) ⊖$_P$ *pderiv*$_R$ *X*$_R$
using *pderiv-var*
by (*subst pderiv-var-pow*[*OF o21*], *simp*)
also have ... = *pderiv*$_R$ *?g*
unfolding *gauss-poly-def a-minus-def* **using** *var-closed*
by (*subst pderiv-add*, *simp-all add*:*pderiv-inv*)
also have ... = *pderiv*$_R$ (*f* [^]$_P$ (*2*::*nat*) ⊗$_P$ *h*)
using *h-def*(*2*) **by** *simp*
also have ... = *pderiv*$_R$ (*f* [^]$_P$ (*2*::*nat*)) ⊗$_P$ *h*
⊕$_P$ (*f* [^]$_P$ (*2*::*nat*)) ⊗$_P$ *pderiv*$_R$ *h*
using *f-carr h-def*
by (*intro pderiv-mult*, *simp-all*)
also have ... = *int-embed P 2* ⊗$_P$ *f* ⊗$_P$ *pderiv*$_R$ *f* ⊗$_P$ *h*
⊕$_P$ *f* ⊗$_P$ *f* ⊗$_P$ *pderiv*$_R$ *h*
using *f-carr*
by (*subst pderiv-pow*, *simp-all add*:*numeral-eq-Suc*)
also have ... = *f* ⊗$_P$ (*int-embed P 2* ⊗$_P$ *pderiv*$_R$ *f* ⊗$_P$ *h*)
⊕$_P$ *f* ⊗$_P$ (*f* ⊗$_P$ *pderiv*$_R$ *h*)
using *f-carr pderiv-carr h-def p.int-embed-closed*
apply (*intro arg-cong2*[**where** *f*=(⊕$_P$)])
by (*subst p.m-comm*, *simp-all add*:*p.m-assoc*)
also have ... = *f* ⊗$_P$
(*int-embed P 2* ⊗$_P$ *pderiv*$_R$ *f* ⊗$_P$ *h* ⊕$_P$ *f* ⊗$_P$ *pderiv*$_R$ *h*)
using *f-carr pderiv-carr h-def p.int-embed-closed*
by (*subst p.r-distr*, *simp-all*)
finally have ⊖$_P$ **1**$_P$ = *f* ⊗$_P$
(*int-embed P 2* ⊗$_P$ *pderiv*$_R$ *f* ⊗$_P$ *h* ⊕$_P$ *f* ⊗$_P$ *pderiv*$_R$ *h*)
(**is** - = *f* ⊗$_P$ *?q*)
by *simp*

hence *f pdivides*$_R$ ⊖$_P$ **1**$_P$
unfolding *factor-def pdivides-def*
using *f-carr pderiv-carr h-def p.int-embed-closed*

```
    by auto
  moreover have ⊖_P 1_P ≠ 0_P by simp
  ultimately have  degree f ≤ degree (⊖_P 1_P)
    using f-carr
    by (intro pdivides-imp-degree-le, simp-all add:univ-poly-zero)
  also have ... = 0
    by (subst univ-poly-a-inv-degree, simp)
     (simp add:univ-poly-one)
  finally have degree f = 0 by simp

  then show False
    using pirreducible-degree assms(2)
    unfolding monic-irreducible-poly-def monic-poly-def
    by fastforce
 qed
 ultimately have pmult_R f ?g = 1 by simp
 then show ?thesis using True by simp
next
 case False
 have o2: order R^n > 1
  using finite-field-min-order assms(1) one-less-power by blast

 have ¬(f pdivides_R gauss-poly R (order R^n))
  using div-gauss-poly-iff[OF assms] False by simp
 hence pmult_R f (gauss-poly R (order R^n)) = 0
  using multiplicity-ge-1-iff-pdivides[OF assms(2)]
  using gauss-poly-carr gauss-poly-not-zero[OF o2] leI less-one
  by blast
 then show ?thesis using False by simp
qed
```

The following lemma, for the case when R is a simple prime field, can be found in Ireland and Rosen [3, §7.1, Corollary 1]. Here the result is verified even for arbitrary finite fields.

```
lemma card-irred-aux:
 assumes n > 0
 shows order R^n = (∑ d | d dvd n. d *
   card {f. monic-irreducible-poly R f ∧ degree f = d})
 (is ?lhs = ?rhs)
proof −
 let ?G = {f. monic-irreducible-poly R f ∧ degree f dvd n}

 let ?D = {f. monic-irreducible-poly R f}
 have a: finite {d. d dvd n} using finite-divisors-nat assms by simp
 have b: finite {f. monic-irreducible-poly R f ∧ degree f = k} for k
 proof −
  have {f. monic-irreducible-poly R f ∧ degree f = k} ⊆
    {f. f ∈ carrier P ∧ degree f ≤ k}
    unfolding monic-irreducible-poly-def monic-poly-def by auto
```

```
  moreover have finite {f. f ∈ carrier P ∧ degree f ≤ k}
    using finite-poly[OF finite-carrier] by simp
  ultimately show ?thesis using finite-subset by simp
qed

have G-split: ?G =
  ⋃ {{f. monic-irreducible-poly R f ∧ degree f = d} | d. d dvd n}
  by auto
have c: finite ?G
  using a b by (subst G-split, auto)
have d: order R^n > 1
  using assms finite-field-min-order one-less-power by blast

have ?lhs = degree (gauss-poly R (order R^n))
  using d
  by (subst gauss-poly-degree, simp-all)
also have ... =
  sum' (λd. pmult_R d (gauss-poly R (order R^n)) * degree d) ?D
  using d
  by (intro degree-monic-poly'[symmetric] gauss-poly-monic)
also have ... = sum' (λd. of-bool (degree d dvd n) * degree d) ?D
  using multiplicity-of-factor-of-gauss-poly[OF assms]
  by (intro sum.cong', auto)
also have ... = sum' (λd. degree d) ?G
  by (intro sum.mono-neutral-cong-right' subsetI, auto)
also have ... = (∑ d ∈ ?G. degree d)
  using c by (intro sum.eq-sum, simp)
also have ... =
  (∑ f ∈ (⋃ d ∈ {d. d dvd n}.
  {f. monic-irreducible-poly R f ∧ degree f = d}). degree f)
  by (intro sum.cong, auto simp add:set-eq-iff)
also have ... = (∑ d | d dvd n. sum degree
  {f. monic-irreducible-poly R f ∧ degree f = d})
 using a b by (subst sum.UNION-disjoint, auto simp add:set-eq-iff)
also have ... = (∑ d | d dvd n. sum (λ-. d)
  {f. monic-irreducible-poly R f ∧ degree f = d})
  by (intro sum.cong, simp-all)
also have ... = ?rhs
  by (simp add:mult.commute)
finally show ?thesis
  by simp
qed

end

end
```

6.2 Gauss Formula

theory *Card-Irreducible-Polynomials*
imports
Dirichlet-Series.Moebius-Mu
Card-Irreducible-Polynomials-Aux
begin

hide-const *Polynomial.order*

The following theorem is a slightly generalized form of the formula discovered by Gauss for the number of monic irreducible polynomials over a finite field. He originally verified the result for the case when R is a simple prime field. The version of the formula here for the case where R may be an arbitrary finite field can be found in Chebolu and Mináč [1].

theorem (**in** *finite-field*) *card-irred*:
assumes $n > 0$
shows $n * card \{f.\ monic\text{-}irreducible\text{-}poly\ R\ f \wedge degree\ f = n\} =$
$(\sum d \mid d\ dvd\ n.\ moebius\text{-}mu\ d * (order\ R \hat{}\ (n\ div\ d)))$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *dirichlet-prod moebius-mu* ($\lambda x.$ *int* (*order R*) $\hat{}$ *x*) *n*
using *card-irred-aux*
by (*intro moebius-inversion assms*) (*simp flip*:*of-nat-power*)
also have ... = *?rhs*
by (*simp add*:*dirichlet-prod-def*)
finally show *?thesis* **by** *simp*
qed

In the following an explicit analytic lower bound for the cardinality of monic irreducible polynomials is shown, with which existence follows. This part deviates from the classic approach, where existence is verified using a divisibility argument. The reason for the deviation is that an analytic bound can also be used to estimate the runtime of a randomized algorithm selecting an irreducible polynomial, by randomly sampling monic polynomials.

lemma (**in** *finite-field*) *card-irred-1*:
card $\{f.\ monic\text{-}irreducible\text{-}poly\ R\ f \wedge degree\ f = 1\} = order\ R$
proof −
have *int* (*1* $*$ *card* $\{f.\ monic\text{-}irreducible\text{-}poly\ R\ f \wedge degree\ f = 1\}$)
= *int* (*order R*)
by (*subst card-irred*, *auto*)
thus *?thesis* **by** *simp*
qed

lemma (**in** *finite-field*) *card-irred-2*:

```
  real (card {f. monic-irreducible-poly R f ∧ degree f = 2}) =
    (real (order R)^2 − order R) / 2
proof −
  have x dvd 2 ⟹ x = 1 ∨ x = 2 for x :: nat
    using nat-dvd-not-less[where m=2]
    by (metis One-nat-def even-zero gcd-nat.strict-trans2
        less-2-cases nat-neq-iff pos2)
  hence a: {d. d dvd 2} = {1,2::nat}
    by (auto simp add:set-eq-iff)

  have 2*real (card {f. monic-irreducible-poly R f ∧ degree f = 2})
    = of-int (2* card {f. monic-irreducible-poly R f ∧ degree f = 2})
    by simp
  also have ... =
    of-int (∑ d | d dvd 2. moebius-mu d * int (order R) ^ (2 div d))
    by (subst card-irred, auto)
  also have ... = order R^2 − int (order R)
    by (subst a, simp)
  also have ... = real (order R)^2 − order R
    by simp
  finally have
    2 * real (card {f. monic-irreducible-poly R f ∧ degree f = 2}) =
    real (order R)^2 − order R
    by simp
  thus ?thesis by simp
qed

lemma (in finite-field) card-irred-gt-2:
  assumes n > 2
  shows real (order R)^n / (2*real n) ≤
    card {f. monic-irreducible-poly R f ∧ degree f = n}
    (is ?lhs ≤ ?rhs)
proof −
  let ?m = real (order R)
  have a:?m ≥ 2
    using finite-field-min-order by simp

  have b:moebius-mu n ≥ −(1::real) for n :: nat
    using abs-moebius-mu-le[where n=n]
    unfolding abs-le-iff by auto

  have c: n > 0 using assms by simp
  have d: x < n − 1 if d-assms: x dvd n x ≠ n for x :: nat
  proof −
    have x < n
      using d-assms dvd-nat-bounds c by auto
    moreover have ¬(n−1 dvd n) using assms
      by (metis One-nat-def Suc-diff-Suc c diff-zero
          dvd-add-triv-right-iff nat-dvd-1-iff-1
```

```
      nat-neq-iff numeral-2-eq-2 plus-1-eq-Suc)
  hence x ≠ n−1 using d-assms by auto
  ultimately show x < n−1  by simp
 qed

 have ?m^n / 2 = ?m^n − ?m^n/2 by simp
 also have ... ≤ ?m^n − ?m^n/?m^1
  using a by (intro diff-mono divide-left-mono, simp-all)
 also have ... ≤ ?m^n − ?m^(n−1)
  using a c by (subst power-diff, simp-all)
 also have ... ≤ ?m^n − (?m^(n−1) − 1)/1 by simp
 also have ... ≤ ?m^n − (?m^(n−1)−1)/(?m−1)
  using a by (intro diff-left-mono divide-left-mono, simp-all)
 also have ... = ?m^n − (∑ i ∈ {..<n−1}. ?m^i)
  using a by (subst geometric-sum, simp-all)
 also have ... ≤ ?m^n − (∑ i ∈ {k. k dvd n ∧ k ≠ n}. ?m^i)
  using d
  by (intro diff-mono sum-mono2 subsetI, auto simp add:not-less)
 also have ... = ?m^n + (∑ i ∈ {k. k dvd n ∧ k ≠ n}. (−1) * ?m^i)
  by (subst sum-distrib-left[symmetric], simp)
 also have ... ≤ moebius-mu 1 * ?m^n +
  (∑ i ∈ {k. k dvd n ∧ k ≠ n}. moebius-mu (n div i) * ?m^i)
  using b
  by (intro add-mono sum-mono mult-right-mono)
   (simp-all add:not-less)
 also have ... =  (∑ i ∈ insert n {k. k dvd n ∧ k ≠ n}.
  moebius-mu (n div i) * ?m^i)
  using c by (subst sum.insert, auto)
 also have ... = (∑ i ∈ {k. k dvd n}. moebius-mu (n div i) * ?m^i)
  by (intro sum.cong, auto simp add:set-eq-iff)
 also have ... = dirichlet-prod (λi. ?m^i) moebius-mu n
  unfolding dirichlet-prod-def by (intro sum.cong, auto)
 also have ... = dirichlet-prod moebius-mu (λi. ?m^i) n
  using dirichlet-prod-moebius-commute by metis
 also have ... =
  of-int (∑ d | d dvd n. moebius-mu d * order R^(n div d))
  unfolding dirichlet-prod-def by simp
 also have ... = of-int (n *
  card {f. monic-irreducible-poly R f ∧ length f − 1 = n})
  using card-irred[OF c] by simp
 also have ... = n * ?rhs by simp
 finally have ?m^n / 2 ≤ n * ?rhs by simp
 hence ?m ^ n ≤ 2 * n * ?rhs by simp
 hence ?m^n/(2*real n) ≤ ?rhs
  using c by (subst pos-divide-le-eq, simp-all add:algebra-simps)
 thus ?thesis by simp
qed

lemma (in finite-field) exist-irred:
```

```
  assumes n > 0
  obtains f where monic-irreducible-poly R f degree f = n
proof −
  consider (i) n = 1 | (ii) n = 2 | (iii) n>2
    using assms by linarith
  then have
    card {f. monic-irreducible-poly R f ∧ degree f = n} > 0
    (is card ?A > 0)
  proof (cases)
    case i
    hence card ?A = order R
      using card-irred-1 by simp
    also have ... > 0
       using finite-field-min-order by simp
    finally show ?thesis by simp
  next
    case ii
    have 0 < (real (order R) ∗ (real (order R) − 1)) / 2
      using finite-field-min-order by simp
    also have ... = (real (order R)^2 − order R) / 2
      by (simp add:power2-eq-square algebra-simps)
    also have ... = real (card ?A)
      using ii by (subst card-irred-2[symmetric], simp)
    finally have  0 < real (card ?A) by simp
    then show ?thesis by simp
  next
    case iii
    have 0 < real (order R)^n / (2∗real n)
      using finite-field-min-order assms by simp
    also have ... ≤ real (card ?A)
      using iii card-irred-gt-2 by simp
    finally have 0 < real (card ?A) by simp
    then show ?thesis by simp
  qed
  hence ?A ≠ {}
    by (metis card.empty nless-le)
  then obtain f where monic-irreducible-poly R f degree f = n
    by auto
  thus ?thesis using that by simp
qed

theorem existence:
  assumes n > 0
  assumes Factorial-Ring.prime p
  shows ∃(F:: int set list set ring). finite-field F ∧ order F = p^n
proof −
  interpret zf: finite-field ZFact (int p)
    using zfact-prime-is-finite-field assms by simp
```

interpret *zfp*: *polynomial-ring ZFact p carrier* (*ZFact p*)
unfolding *polynomial-ring-def polynomial-ring-axioms-def*
using *zf.field-axioms zf.carrier-is-subfield* **by** *simp*

have *p-gt-0*: $p > 0$ **using** *prime-gt-0-nat assms*(*2*) **by** *simp*

obtain *f* **where** *f-def*:
monic-irreducible-poly (*ZFact* (*int p*)) *f*
degree f = *n*
using *zf.exist-irred assms* **by** *auto*

let *?F* = $Rupt_{(ZFact\ p)}$ (*carrier* (*ZFact p*)) *f*
have *f* ∈ *carrier* (*poly-ring* (*ZFact* (*int p*)))
using *f-def*(*1*) *zf.monic-poly-carr*
unfolding *monic-irreducible-poly-def*
by *simp*
moreover have *degree f* > *0*
using *assms*(*1*) *f-def* **by** *simp*
ultimately have *order ?F* = *card* (*carrier* (*ZFact p*))^*degree f*
by (*intro zf.rupture-order*[*OF zf.carrier-is-subfield*]) *auto*
hence *a*:*order ?F* = *p*^*n*
unfolding *f-def*(*2*) *card-zfact-carr*[*OF p-gt-0*] **by** *simp*

have *field ?F*
using *f-def*(*1*) *zf.monic-poly-carr monic-irreducible-poly-def*
by (*subst zfp.rupture-is-field-iff-pirreducible*) *auto*
moreover have *order ?F* > *0*
unfolding *a* **using** *assms*(*1*,*2*) *p-gt-0* **by** *simp*
ultimately have *b*:*finite-field ?F*
using *card-ge-0-finite*
by (*intro finite-fieldI*, *auto simp add*:*Coset.order-def*)

show *?thesis*
using *a b*
by (*intro exI*[**where** *x*=*?F*], *simp*)
qed

end

7 Isomorphism between Finite Fields

theory *Finite-Fields-Isomorphic*
imports
Card-Irreducible-Polynomials
begin

lemma (**in** *finite-field*) *eval-on-root-is-iso*:
defines *p* ≡ *char R*
assumes *f* ∈ *carrier* (*poly-ring* (*ZFact p*))

assumes $pirreducible_{(ZFact\ p)}\ (carrier\ (ZFact\ p))\ f$
assumes $order\ R = p \hat{} degree\ f$
assumes $x \in carrier\ R$
assumes $eval\ (map\ (char\text{-}iso\ R)\ f)\ x = \mathbf{0}$
shows $ring\text{-}hom\text{-}ring\ (Rupt_{(ZFact\ p)}\ (carrier\ (ZFact\ p))\ f)\ R$
$(\lambda g.\ the\text{-}elem\ ((\lambda g'.\ eval\ (map\ (char\text{-}iso\ R)\ g')\ x)\ `\ g))$
proof −
let $?P = poly\text{-}ring\ (ZFact\ p)$

have *char-pos*: $char\ R > 0$
using *finite-carr-imp-char-ge-0*[*OF finite-carrier*] **by** *simp*

have *p-prime*: *Factorial-Ring.prime p*
unfolding *p-def*
using *characteristic-is-prime*[*OF char-pos*] **by** *simp*

interpret *zf*: *finite-field ZFact p*
using *zfact-prime-is-finite-field p-prime* **by** *simp*
interpret *pzf*: *principal-domain poly-ring* (*ZFact p*)
using *zf.univ-poly-is-principal*[*OF zf.carrier-is-subfield*] **by** *simp*

interpret *i*: *ideal* (*PIdl*$_{?P}$ *f*) *?P*
by (*intro pzf.cgenideal-ideal assms*(*2*))
have *rupt-carr*: $y \subseteq carrier\ (poly\text{-}ring\ (ZFact\ p))$
if $y \in carrier\ (Rupt_{ZFact\ p}\ (carrier\ (ZFact\ p))\ f)$ **for** *y*
using *that pzf.quot-carr i.ideal-axioms* **by** (*simp add*:*rupture-def*)

have *rupt-is-ring*: $ring\ (Rupt_{ZFact\ p}\ (carrier\ (ZFact\ p))\ f)$
unfolding *rupture-def* **by** (*intro i.quotient-is-ring*)

have $map\ (char\text{-}iso\ R) \in$
$ring\text{-}iso\ ?P\ (poly\text{-}ring\ (R(\!|carrier := char\text{-}subring\ R|\!)))$
using *lift-iso-to-poly-ring*[*OF char-iso*] *zf.domain-axioms*
using *char-ring-is-subdomain subdomain-is-domain*
by (*simp add*:*p-def*)
moreover have $(char\text{-}subring\ R)[X] =$
$poly\text{-}ring\ (R\ (\!|carrier := char\text{-}subring\ R|\!))$
using *univ-poly-consistent*[*OF char-ring-is-subring*] **by** *simp*
ultimately have
$map\ (char\text{-}iso\ R) \in ring\text{-}hom\ ?P\ ((char\text{-}subring\ R)[X])$
by (*simp add*:*ring-iso-def*)
moreover have $(\lambda p.\ eval\ p\ x) \in ring\text{-}hom\ ((char\text{-}subring\ R)[X])\ R$
using *eval-is-hom char-ring-is-subring assms*(*5*) **by** *simp*
ultimately have
$(\lambda p.\ eval\ p\ x) \circ map\ (char\text{-}iso\ R) \in ring\text{-}hom\ ?P\ R$
using *ring-hom-trans* **by** *blast*
hence *a*:$(\lambda p.\ eval\ (map\ (char\text{-}iso\ R)\ p)\ x) \in ring\text{-}hom\ ?P\ R$
by (*simp add*:*comp-def*)
interpret *h*:*ring-hom-ring ?P R* $(\lambda p.\ eval\ (map\ (char\text{-}iso\ R)\ p)\ x)$

by (*intro ring-hom-ringI2 pzf.is-ring a ring-axioms*)

let *?h* = (λ*p*. *eval* (*map* (*char-iso R*) *p*) *x*)
let *?J* = *a-kernel* (*poly-ring* (*ZFact* (*int p*))) *R ?h*

have *?h* ‘ *a-kernel* (*poly-ring* (*ZFact* (*int p*))) *R ?h* ⊆ {**0**}
by *auto*
moreover have
0$_{?P}$ ∈ *a-kernel* (*poly-ring* (*ZFact* (*int p*))) *R ?h*
?h **0**$_{?P}$ = **0**
unfolding *a-kernel-def'* **by** *simp-all*
hence {**0**} ⊆ *?h* ‘ *a-kernel* (*poly-ring* (*ZFact* (*int p*))) *R ?h*
by *simp*
ultimately have *c*:
?h ‘ *a-kernel* (*poly-ring* (*ZFact* (*int p*))) *R ?h* = {**0**}
by *auto*

have *d*: *PIdl*$_{?P}$ *f* ⊆ *a-kernel ?P R ?h*
proof (*rule subsetI*)
fix *y* **assume** *y* ∈ *PIdl*$_{?P}$ *f*
then obtain *y'* **where** *y'-def*: *y'* ∈ *carrier ?P y* = *y'* ⊗$_{?P}$ *f*
unfolding *cgenideal-def* **by** *auto*
have *?h y* = *?h* (*y'* ⊗$_{?P}$ *f*) **by** (*simp add*:*y'-def*)
also have ... = *?h y'* ⊗ *?h f*
using *y'-def assms*(*2*) **by** *simp*
also have ... = *?h y'* ⊗ **0**
using *assms*(*6*) **by** *simp*
also have ... = **0**
using *y'-def* **by** *simp*
finally have *?h y* = **0** **by** *simp*
moreover have *y* ∈ *carrier ?P* **using** *y'-def assms*(*2*) **by** *simp*
ultimately show *y* ∈ *a-kernel ?P R ?h*
unfolding *a-kernel-def kernel-def* **by** *simp*
qed

have (λ*y*. *the-elem* ((λ*p*. *eval* (*map* (*char-iso R*) *p*) *x*) ‘ *y*))
∈ *ring-hom* (*?P Quot ?J*) *R*
using *h.the-elem-hom* **by** *simp*
moreover have (λ*y*. *?J* <+>$_{?P}$ *y*)
∈ *ring-hom* (*Rupt*$_{(ZFact\ p)}$ (*carrier* (*ZFact p*)) *f*) (*?P Quot ?J*)
unfolding *rupture-def* **using** *h.kernel-is-ideal d assms*(*2*)
by (*intro pzf.quot-quot-hom pzf.cgenideal-ideal*) *auto*
ultimately have (λ*y*. *the-elem* (*?h* ‘ *y*)) ∘ (λ*y*. *?J* <+>$_{?P}$ *y*)
∈ *ring-hom* (*Rupt*$_{(ZFact\ p)}$ (*carrier* (*ZFact p*)) *f*) *R*
using *ring-hom-trans* **by** *blast*
hence *b*: (λ*y*. *the-elem* (*?h* ‘ (*?J* <+>$_{?P}$ *y*))) ∈
ring-hom (*Rupt*$_{(ZFact\ p)}$ (*carrier* (*ZFact p*)) *f*) *R*
by (*simp add*:*comp-def*)
have *?h* ‘ *y* = *?h* ‘ (*?J* <+>$_{?P}$ *y*)

if $y \in carrier\ (Rupt_{ZFact\ p}\ (carrier\ (ZFact\ p))\ f)$
for y
proof −
have *y-range*: $y \subseteq carrier\ ?P$
using *rupt-carr that* **by** *simp*
have $?h\ `\ y = \{\mathbf{0}\} <+>_R\ ?h\ `\ y$
using *y-range h.hom-closed* **by** (*subst set-add-zero*, *auto*)
also have $... = ?h\ `\ ?J <+>_R\ ?h\ `\ y$
by (*subst c*, *simp*)
also have $... = ?h\ `\ (?J <+>_{?P}\ y)$
by (*subst set-add-hom*[*OF a - y-range*], *subst a-kernel-def'*) *auto*
finally show *?thesis* **by** *simp*
qed
hence $(\lambda y.\ the\text{-}elem\ (?h\ `\ y)) \in$
$ring\text{-}hom\ (Rupt_{(ZFact\ p)})\ (carrier\ (ZFact\ p))\ f)\ R$
by (*intro ring-hom-cong*[*OF - rupt-is-ring b*]) *simp*
thus *?thesis*
by (*intro ring-hom-ringI2 rupt-is-ring ring-axioms*, *simp*)
qed

lemma (**in** *domain*) *pdivides-consistent*:
assumes *subfield K R f* $\in$ *carrier* $(K[X])$ *g* $\in$ *carrier* $(K[X])$
shows *f pdivides g* $\longleftrightarrow$ *f pdivides*$_{R\ (\!|\ carrier := K\ |\!)}$ *g*
proof −
have *a*:*subring K R*
using *assms*(*1*) *subfieldE*(*1*) **by** *auto*
let $?S = R\ (\!|\ carrier := K\ |\!)$
have *f pdivides g* $\longleftrightarrow$ *f divides*$_{K[X]}$ *g*
using *pdivides-iff-shell*[*OF assms*] **by** *simp*
also have $... \longleftrightarrow (\exists x \in carrier\ (K[X]).\ f \otimes_{K[X]}\ x = g)$
unfolding *pdivides-def factor-def* **by** *auto*
also have $... \longleftrightarrow$
$(\exists x \in carrier\ (poly\text{-}ring\ ?S).\ f \otimes_{poly\text{-}ring\ ?S}\ x = g)$
using *univ-poly-consistent*[*OF a*] **by** *simp*
also have $... \longleftrightarrow$ *f divides*$_{poly\text{-}ring\ ?S}$ *g*
unfolding *pdivides-def factor-def* **by** *auto*
also have $... \longleftrightarrow$ *f pdivides*$_{?S}$ *g*
unfolding *pdivides-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma (**in** *finite-field*) *find-root*:
assumes *subfield K R*
assumes *monic-irreducible-poly* $(R\ (\!|\ carrier := K\ |\!))\ f$
assumes *order R* = *card K*^*degree f*
obtains *x* **where** *eval f x* = $\mathbf{0}$ *x* $\in$ *carrier R*
proof −
define $\tau :: 'a\ list \Rightarrow 'a\ list$ **where** $\tau = id$
let $?K = R\ (\!|\ carrier := K\ |\!)$

have *finite K*
using *assms*(*1*) **by** (*intro finite-subset*[*OF - finite-carrier*], *simp*)
hence *fin-K*: *finite* (*carrier* (*?K*))
by *simp*
interpret *f*: *finite-field ?K*
using *assms*(*1*) *subfield-iff fin-K finite-fieldI* **by** *blast*
have *b*:*subring K R*
using *assms*(*1*) *subfieldE*(*1*) **by** *blast*
interpret *e*: *ring-hom-ring* (*K*[*X*]) (*poly-ring R*) *τ*
using *embed-hom*[*OF b*] **by** (*simp add*:*τ-def*)

have *a*: *card K^degree f* > *1*
using *assms*(*3*) *finite-field-min-order* **by** *simp*
have *f* ∈ *carrier* (*poly-ring ?K*)
using *f.monic-poly-carr assms*(*2*)
unfolding *monic-irreducible-poly-def* **by** *simp*
hence *f-carr-2*: *f* ∈ *carrier* (*K*[*X*])
using *univ-poly-consistent*[*OF b*] **by** *simp*
have *f-carr*: *f* ∈ *carrier* (*poly-ring R*)
using *e.hom-closed*[*OF f-carr-2*] **unfolding** *τ-def* **by** *simp*

have *gp-carr*: *gauss-poly ?K* (*order ?K^degree f*) ∈ *carrier* (*K*[*X*])
using *f.gauss-poly-carr univ-poly-consistent*[*OF b*] **by** *simp*

have *gauss-poly ?K* (*order ?K^degree f*) =
gauss-poly ?K (*card K^degree f*)
by (*simp add*:*Coset.order-def*)
also have ... =
$X_{?K}$ [^]$_{poly\text{-}ring\ ?K}$ *card K ^ degree f* ⊖$_{poly\text{-}ring\ ?K}$ $X_{?K}$
unfolding *gauss-poly-def* **by** *simp*
also have ... = X_R [^]$_{K[X]}$ *card K ^ degree f* ⊖$_{K[X]}$ X_R
unfolding *var-def* **using** *univ-poly-consistent*[*OF b*] **by** *simp*
also have ... = *τ* (X_R [^]$_{K[X]}$ *card K ^ degree f* ⊖$_{K[X]}$ X_R)
unfolding *τ-def* **by** *simp*
also have ... = *gauss-poly R* (*card K^degree f*)
unfolding *gauss-poly-def a-minus-def* **using** *var-closed*[*OF b*]
by (*simp add*:*e.hom-nat-pow*, *simp add*:*τ-def*)
finally have *gp-consistent*: *gauss-poly ?K* (*order ?K^degree f*) =
gauss-poly R (*card K^degree f*)
by *simp*

have *deg-f*: *degree f* > *0*
using *f.monic-poly-min-degree*[*OF assms*(*2*)] **by** *simp*

have *splitted f*
proof (*cases degree f* > *1*)
case *True*

have *f pdivides*$_{?K}$ *gauss-poly ?K* (*order ?K^degree f*)

```
    using f.div-gauss-poly-iff[OF deg-f assms(2)] by simp
  hence f pdivides gauss-poly ?K (order ?K^degree f)
    using pdivides-consistent[OF assms(1)] f-carr-2 gp-carr by simp
  hence f pdivides gauss-poly R (card K^degree f)
    using gp-consistent by simp
  moreover have splitted (gauss-poly R (card K^degree f))
   unfolding assms(3)[symmetric] using gauss-poly-splitted by simp
  moreover have gauss-poly R (card K^degree f) ≠ []
    using gauss-poly-not-zero a by (simp add: univ-poly-zero)
  ultimately show splitted f
    using pdivides-imp-splitted f-carr gauss-poly-carr by auto
 next
  case False
  hence degree f = 1 using deg-f by simp
  thus ?thesis using f-carr degree-one-imp-splitted by auto
 qed
 hence size (roots f) > 0
  using deg-f unfolding splitted-def by simp
 then obtain x where x-def: x ∈ carrier R is-root f x
  using roots-mem-iff-is-root[OF f-carr]
  by (metis f-carr nonempty-has-size not-empty-rootsE)
 have eval f x = 0
  using x-def is-root-def by blast
 thus ?thesis using x-def using that by simp
qed

lemma (in finite-field) find-iso-from-zfact:
 defines p ≡ int (char R)
 assumes monic-irreducible-poly (ZFact p) f
 assumes order R = char R^degree f
 shows ∃φ. φ ∈ ring-iso (Rupt_(ZFact p) (carrier (ZFact p)) f) R
proof −
 have char-pos: char R > 0
  using finite-carr-imp-char-ge-0[OF finite-carrier] by simp

 interpret zf: finite-field ZFact p
  unfolding p-def using zfact-prime-is-finite-field
  using characteristic-is-prime[OF char-pos] by simp

 interpret zfp: polynomial-ring ZFact p carrier (ZFact p)
  unfolding polynomial-ring-def polynomial-ring-axioms-def
  using zf.field-axioms zf.carrier-is-subfield by simp

 let ?f' = map (char-iso R) f
 let ?F = Rupt_(ZFact p) (carrier (ZFact p)) f

 have domain (R(|carrier := char-subring R|))
  using char-ring-is-subdomain subdomain-is-domain by simp
```

hence *monic-irreducible-poly* (*R* (| *carrier* := *char-subring R* |)) *?f′*
using *char-iso p-def zf.domain-axioms*
by (*intro monic-irreducible-poly-hom*[*OF assms*(*2*)]) *auto*
moreover have *order R* = *card* (*char-subring R*)^*degree ?f′*
using *assms*(*3*) **unfolding** *char-def* **by** *simp*
ultimately obtain *x* **where** *x-def*: *eval ?f′ x* = **0** *x* ∈ *carrier R*
using *find-root*[*OF char-ring-is-subfield*[*OF char-pos*]] **by** *blast*
let *?φ* = (λ*g*. *the-elem* ((λ*g′*. *eval* (*map* (*char-iso R*) *g′*) *x*) ‘ *g*))
interpret *r*: *ring-hom-ring ?F R ?φ*
using *assms*(*2*,*3*)
unfolding *monic-irreducible-poly-def monic-poly-def p-def*
by (*intro eval-on-root-is-iso x-def*, *auto*)
have *a*:*?φ* ∈ *ring-hom ?F R*
using *r.homh* **by** *auto*

have *field* (*Rupt*$_{ZFact\ p}$ (*carrier* (*ZFact p*)) *f*)
using *assms*(*2*)
unfolding *monic-irreducible-poly-def monic-poly-def*
by (*subst zfp.rupture-is-field-iff-pirreducible*, *simp-all*)
hence *b*:*inj-on ?φ* (*carrier ?F*)
using *non-trivial-field-hom-is-inj*[*OF a - field-axioms*] **by** *simp*

have *card* (*?φ* ‘ *carrier ?F*) = *order ?F*
using *card-image*[*OF b*] **unfolding** *Coset.order-def* **by** *simp*
also have ... = *card* (*carrier* (*ZFact p*))^*degree f*
using *assms*(*2*) *zf.monic-poly-min-degree*[*OF assms*(*2*)]
unfolding *monic-irreducible-poly-def monic-poly-def*
by (*intro zf.rupture-order*[*OF zf.carrier-is-subfield*]) *auto*
also have ... = *char R* ^*degree f*
unfolding *p-def* **by** (*subst card-zfact-carr*[*OF char-pos*], *simp*)
also have ... = *card* (*carrier R*)
using *assms*(*3*) **unfolding** *Coset.order-def* **by** *simp*
finally have *card* (*?φ* ‘ *carrier ?F*) = *card* (*carrier R*) **by** *simp*
moreover have *?φ* ‘ *carrier ?F* ⊆ *carrier R*
by (*intro image-subsetI*, *simp*)
ultimately have *?φ* ‘ *carrier ?F* = *carrier R*
by (*intro card-seteq finite-carrier*, *auto*)
hence *bij-betw ?φ* (*carrier ?F*) (*carrier R*)
using *b bij-betw-imageI* **by** *auto*

thus *?thesis*
unfolding *ring-iso-def* **using** *a b* **by** *auto*
qed

theorem *uniqueness*:
assumes *finite-field* F_1
assumes *finite-field* F_2
assumes *order* F_1 = *order* F_2
shows $F_1 \simeq F_2$

proof −
obtain n **where** *o1*: *order* F_1 = *char* $F_1\hat{}n$ $n > 0$
using *finite-field.finite-field-order*[*OF assms*(*1*)] **by** *auto*
obtain m **where** *o2*: *order* F_2 = *char* $F_2\hat{}m$ $m > 0$
using *finite-field.finite-field-order*[*OF assms*(*2*)] **by** *auto*

interpret *f1*: *finite-field* F_1 **using** *assms*(*1*) **by** *simp*
interpret *f2*: *finite-field* F_2 **using** *assms*(*2*) **by** *simp*

have *char-pos*: *char* $F_1 > 0$ *char* $F_2 > 0$
using *f1.finite-carrier f1.finite-carr-imp-char-ge-0*
using *f2.finite-carrier f2.finite-carr-imp-char-ge-0* **by** *auto*
hence *char-prime*:
Factorial-Ring.prime (*char* F_1)
Factorial-Ring.prime (*char* F_2)
using *f1.characteristic-is-prime f2.characteristic-is-prime*
by *auto*

have *char* $F_1\hat{}n$ = *char* $F_2\hat{}m$
using *o1 o2 assms*(*3*) **by** *simp*
hence *eq*: $n = m$ *char* F_1 = *char* F_2
using *char-prime char-pos o1*(*2*) *o2*(*2*) *prime-power-inj'* **by** *auto*

obtain p **where** *p-def*: p = *char* F_1 p = *char* F_2
using *eq* **by** *simp*

have *p-prime*: *Factorial-Ring.prime p*
unfolding *p-def*(*1*)
using *f1.characteristic-is-prime char-pos* **by** *simp*

interpret *zf*: *finite-field ZFact* (*int p*)
using *zfact-prime-is-finite-field p-prime o1*(*2*)
using *prime-nat-int-transfer* **by** *blast*

interpret *zfp*: *polynomial-ring ZFact p carrier* (*ZFact p*)
unfolding *polynomial-ring-def polynomial-ring-axioms-def*
using *zf.field-axioms zf.carrier-is-subfield* **by** *simp*

obtain f **where** *f-def*:
monic-irreducible-poly (*ZFact* (*int p*)) f *degree* $f = n$
using *zf.exist-irred o1*(*2*) **by** *auto*

let *?*F_0 = $Rupt_{(ZFact\ p)}$ (*carrier* (*ZFact p*)) f

obtain φ_1 **where** φ_1*-def*: $\varphi_1 \in$ *ring-iso ?*F_0 F_1
using *f1.find-iso-from-zfact f-def o1*
unfolding *p-def* **by** *auto*

obtain φ_2 **where** φ_2*-def*: $\varphi_2 \in$ *ring-iso ?*F_0 F_2

using *f2.find-iso-from-zfact f-def o2*
unfolding *p-def*(*2*) *eq*(*1*) **by** *auto*

have *?F*$_0 \simeq F_1$ **using** φ_1*-def is-ring-iso-def* **by** *auto*
moreover have *?F*$_0 \simeq F_2$ **using** φ_2*-def is-ring-iso-def* **by** *auto*
moreover have *field ?F*$_0$
using *f-def*(*1*) *zf.monic-poly-carr monic-irreducible-poly-def*
by (*subst zfp.rupture-is-field-iff-pirreducible*) *auto*
hence *ring ?F*$_0$ **using** *field.is-ring* **by** *auto*
ultimately show *?thesis*
using *ring-iso-trans ring-iso-sym* **by** *blast*
qed

end

References

[1] S. K. Chebolu and J. Mináč. Counting irreducible polynomials over finite fields using the inclusion-exclusion principle. *Mathematics Magazine*, 84:369 – 371, 2010.

[2] M. Eberl. Dirichlet series. *Archive of Formal Proofs*, Oct. 2017. https://isa-afp.org/entries/Dirichlet_Series.html, Formal proof development.

[3] K. Ireland and M. Rosen. *A classical introduction to modern number theory*, volume 84 of *Graduate texts in mathematics*. Springer, 1982.

[4] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, USA, 1986.