Finite Fields

Emin Karayel

March 17, 2025

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, Rabin's test for the irreducibility of polynomials 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 and publications from Ireland and Rosen [3], Rabin [5] 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

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

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.

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 $\bigwedge i.\ i \in A \Longrightarrow g\ i \le f\ i$
  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 $(intro\ finite\text{-}subset[OF\ \text{-}\ assms(1)]\ subsetI,\ force)$
  let $?B = \{i \in A.\ f\ i \neq 0 \lor g\ i \neq 0\}$

  have $b{:}?B = \{i \in A.\ f\ i \neq 0\} \cup \{i \in A.\ g\ i \neq 0\}$
    by $(auto\ simp\ add{:}set\text{-}eq\text{-}iff)$
  have $a{:}finite\ ?B$
    using $assms(1)\ c$ **by** $(subst\ b,\ simp)$
  have $?lhs = sum'\ (\lambda i.\ f\ i - g\ i)\ ?B$
    by $(intro\ sum.mono\text{-}neutral\text{-}cong\text{-}right',\ simp\text{-}all)$

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 $\bigwedge i.\ i \in ?B \Longrightarrow f\ i = 0$
using *sum-nonneg-0*[*OF assms*(*1*) - *a*] **by** *blast*
thus $\bigwedge i.\ i \in A \Longrightarrow f\ i = 0$
by *blast*
qed

lemma *sum′-eq-iff*:
fixes $f :: 'a \Rightarrow nat$
assumes *finite* $\{i \in A.\ f\ i \neq 0\}$
assumes $\bigwedge i.\ i \in A \Longrightarrow f\ i \geq g\ i$
assumes *sum′* $f\ A \leq$ *sum′* $g\ A$
shows $\forall i \in A.\ f\ i = g\ i$
proof −
have $\{i \in A.\ g\ i \neq 0\} \subseteq \{i \in A.\ f\ i \neq 0\}$
using *assms*(*2*) *order-less-le-trans*
by (*intro subsetI*, *auto*)
hence *a*:*finite* $\{i \in A.\ g\ i \neq 0\}$
by (*rule finite-subset*, *intro assms*(*1*))
have $\{i \in A.\ f\ i - g\ i \neq 0\} \subseteq \{i \in A.\ f\ i \neq 0\}$
by (*intro subsetI*, *simp-all*)
hence *b*: *finite* $\{i \in A.\ f\ i - g\ i \neq 0\}$
by (*rule finite-subset*, *intro assms*(*1*))

have *sum'* $(\lambda i.\ f\ i - g\ i)\ A = sum'\ f\ A - sum'\ g\ A$
using *assms(1,2) a* **by** (*subst sum'-subtractf-nat, auto*)
also have ... = *0*
using *assms(3)* **by** *simp*
finally have *sum'* $(\lambda i.\ f\ i - g\ i)\ A = 0$ **by** *simp*
hence $\bigwedge i.\ i \in A \Longrightarrow f\ i - g\ i = 0$
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 $\bigwedge x.\ x \in carrier\ (K\ [X]) \Longrightarrow id\ x \in carrier\ (poly\text{-}ring\ R)$
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 $degree\ f \neq degree\ g$
  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** *ring*) *degree-add*:
  $degree\ (f \oplus_{K[X]} g) \leq max\ (degree\ f)\ (degree\ g)$
  unfolding *univ-poly-add* **by** (*intro poly-add-degree*)

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*:
  $degree\ (\mathbf{1}_{K[X]}) = 0$
  unfolding *univ-poly-one* **by** *simp*

lemma (**in** *domain*) *pow-non-zero*:
  $x \in carrier\ R \Longrightarrow x \neq \mathbf{0} \Longrightarrow x\ [\ulcorner]\ (n :: nat) \neq \mathbf{0}$
  using *integral* **by** (*induction n*, *auto*)

lemma (**in** *domain*) *degree-pow*:
  assumes *subring K R*
  assumes $f \in carrier\ (K[X]) - \{\mathbf{0}_{K[X]}\}$
  shows $degree\ (f\ [\ulcorner]_{K[X]}\ n) = degree\ f * n$
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 $\otimes_{K[X]}$ 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* ($K[X]$) − {$\mathbf{0}_{K\ [X]}$}
proof −
have X_R ∈ *carrier* ($K[X]$)
  using *var-closed*[*OF assms*(*1*)] **by** *simp*
moreover have $X \neq \mathbf{0}_{K\ [X]}$
  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* ($K[X]$) − {$\mathbf{0}_{K\ [X]}$}
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 $X \neq \mathbf{0}_{K\ [X]}$
  unfolding *var-def univ-poly-zero* **by** *simp*
hence X_R [^]$_{K\ [X]}$ $n \neq \mathbf{0}_{K\ [X]}$
  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* $(X_R\ [\widehat{\ }]_{K\ [X]}\ n) = n$
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 $f \in A \rightarrow$ *carrier* $R - \{\mathbf{0}\}$
shows $(\bigotimes i \in A.\ f\ i) \in$ *carrier* $R - \{\mathbf{0}\}$
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*) $= f\ x \otimes$ *finprod R f F*
using *insert* **by** (*subst finprod-insert*, *simp-all add*:*Pi-def*)
also have ... $\in$ *carrier* $R - \{\mathbf{0}\}$
using *integral insert* **by** *auto*
finally show *?case* **by** *simp*
qed

lemma (**in** *domain*) *degree-prod*:
assumes *finite A*
assumes *subring K R*
assumes $f \in A \rightarrow$ *carrier* $(K[X]) - \{\mathbf{0}_{K[X]}\}$
shows *degree* $(\bigotimes_{K[X]} i \in A.\ f\ i) = (\sum i \in A.$ *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 $(f\ x \otimes_{K[X]}$ *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* $(f\ x) + (\sum i \in F.$ *degree* $(f\ i))$
using *insert* **by** (*subst insert*(*3*), *auto*)
also have ... $= (\sum i \in$ *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


lemma (in domain) coeff-a-inv:
  assumes subring K R
  assumes f ∈ carrier (K[X])
  shows coeff (⊖K[X] f) i = ⊖ (coeff f i) (is ?L = ?R)
proof –
  have ?L = coeff (map (a-inv R) f) i
    unfolding univ-poly-a-inv-def'[OF assms(1,2)] by simp
  also have ... = ?R by (induction f) auto
  finally show ?thesis by simp
qed
```

This is a version of geometric sums for commutative rings:

```
lemma (in cring) geom:
  fixes q:: nat
  assumes [simp]: a ∈ carrier R
  shows (a ⊖ 1) ⊗ (⨁i∈{..<q}. a [^] i) = (a [^] q ⊖ 1)
    (is ?lhs = ?rhs)
proof –
  have [simp]: a [^] i ∈ carrier R for i :: nat
    by (intro nat-pow-closed assms)
  have [simp]: ⊖ 1 ⊗ x = ⊖ x if x ∈ carrier R for x
    using l-minus l-one one-closed that by presburger

  let ?cterm = (⨁i∈{1..<q}. a [^] i)

  have ?lhs = a ⊗ (⨁i∈{..<q}. a [^] i)  ⊖ (⨁i∈{..<q}. a [^] i)
```

```
    unfolding a-minus-def by (subst l-distr, simp-all add:Pi-def)
  also have ... = (⨁ i∈{..<q}. a ⊗ a [^] i) ⊖ (⨁ i∈{..<q}. a [^] i)
    by (subst finsum-rdistr, simp-all add:Pi-def)
  also have ... = (⨁ i∈{..<q}. a [^] (Suc i)) ⊖ (⨁ i∈{..<q}. a [^] i)
    by (subst nat-pow-Suc, simp-all add:m-comm)
  also have ... = (⨁ i∈Suc ` {..<q}. a [^] i) ⊖ (⨁ i∈{..<q}. a [^] i)
    by (subst finsum-reindex, simp-all)
  also have ... =
    (⨁ i∈ insert q {1..<q}. a [^] i) ⊖
    (⨁ i∈ insert 0 {1..<q}. a [^] 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=λx y. x ⊖ y] finsum-cong)
        simp-all
  next
    case False
    then show ?thesis by (simp, algebra)
  qed
  also have ... = (a [^] q ⊕ ?cterm) ⊖ (𝟭 ⊕ ?cterm)
    by simp
  also have ... = a [^] q ⊕ ?cterm ⊕ (⊖ 𝟭 ⊕ ⊖ ?cterm)
    unfolding a-minus-def by (subst minus-add, simp-all)
  also have ... = a [^] q ⊕ (?cterm ⊕ (⊖ 𝟭 ⊕ ⊖ ?cterm))
    by (subst a-assoc, simp-all)
  also have ... = a [^] q ⊕ (?cterm ⊕ (⊖ ?cterm ⊕ ⊖ 𝟭))
    by (subst a-comm[where x=⊖ 𝟭], simp-all)
  also have ... = a [^] q ⊕ ((?cterm ⊕ (⊖ ?cterm)) ⊕ ⊖ 𝟭)
    by (subst a-assoc, simp-all)
  also have ... = a [^] q ⊕ (𝟬 ⊕ ⊖ 𝟭)
    by (subst r-neg, simp-all)
  also have ... = a [^] q ⊖ 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.ring-axioms ds.ring-axioms 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* ≠ $\mathbf{0}_S$ *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* $\mathbf{0}_S$
    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*) ≠ $\mathbf{0}_R$
    unfolding *h.hom-zero*[*symmetric*]
    unfolding *the-inv-into-f-f*[*OF h-inj dr.zero-closed*] **by** *simp*
  hence *lead-coeff* (*?hinv x*) ≠ $\mathbf{0}_R$
    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 (λ*y. 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*: $h \in$ *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* $\in$ *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 ... $\subseteq$ *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 $h \in$ *ring-iso R S*
assumes *domain R domain S*
assumes $x \in$ *carrier R* $y \in$ *carrier R*
shows x *divides*$_R$ $y \longleftrightarrow$ (*h x*) *divides*$_S$ (*h y*) (**is** *?lhs* $\longleftrightarrow$ *?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.ring-axioms ds.ring-axioms 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* $\longleftrightarrow$ ($\exists\, c \in$ *carrier R*. $y = x \otimes_R c$)
unfolding *factor-def* **by** *simp*
also have ... $\longleftrightarrow$ ($\exists\, c \in$ *carrier R*. $h\ y = h\ x \otimes_S h\ c$)
using *assms*(*4*,*5*) *inj-onD*[*OF h-inj-on*]
by (*intro bex-cong*, *auto simp flip*:*h.hom-mult*)
also have ... $\longleftrightarrow$ ($\exists\, c \in$ *carrier S*. $h\ y = h\ x \otimes_S c$)
unfolding *h-img*[*symmetric*] **by** *simp*
also have ... $\longleftrightarrow$ *?rhs*

unfolding *factor-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *properfactor-hom*:
assumes $h \in$ *ring-iso R S*
assumes *domain R domain S*
assumes $x \in$ *carrier R* $b \in$ *carrier R*
shows *properfactor R b x* $\longleftrightarrow$ *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 $h \in$ *ring-iso R S*
assumes *domain R domain S*
assumes $x \in$ *carrier R*
shows $x \in$ *Units R* $\longleftrightarrow$ $h\ x \in$ *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.ring-axioms ds.ring-axioms 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* $\mathbf{0}_{poly\text{-}ring\ R} = \mathbf{0}_{poly\text{-}ring\ S}$
by (*simp add*:*univ-poly-zero*)

ultimately have *mh-zero-iff*:
map h f = $\mathbf{0}_{poly\text{-}ring\ S} \longleftrightarrow f = \mathbf{0}_{poly\text{-}ring\ R}$
using *assms*(*4*) **by** (*metis pdr.zero-closed inj-onD*)

have *?lhs* $\longleftrightarrow$ ($f \neq \mathbf{0}_{poly\text{-}ring\ R} \wedge$ *irreducible* (*poly-ring R*) *f*)
unfolding *ring-irreducible-def* **by** *simp*
also have ... $\longleftrightarrow$
($f \neq \mathbf{0}_{poly\text{-}ring\ R} \wedge$ *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* $\neq \mathbf{0}_{poly\text{-}ring\ S} \wedge$ *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 ∃y∈carrier R. x = I +> y
    using that unfolding FactRing-def A-RCOSETS-def' by simp
  then obtain y where y-def: y ∈ 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 ⊗_{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 ⊗ y1)
    using x1-def y1-def
    by (simp add: FactRing-def ii.rcoset-mult-add)
  also have ... = (J <+>_R I) +> x1 ⊗ y1
    using x1-def(1) y1-def(1)
    by (subst a-setmult-rcos-assoc) auto
  also have ... = J +> x1 ⊗ y1
    using a by simp
  also have ... = [mod J:] (J +> x1) ⨂ (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 1R 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 ... = 1R Quot J
    unfolding FactRing-def by simp
  finally show J <+>R 1R Quot I = 1R Quot J
    by simp
qed

lemma (in ring) quot-carr:
  assumes ideal I R
  assumes y ∈ carrier (R Quot I)
  shows y ⊆ carrier R
proof -
  interpret ideal I R using assms(1) by simp
  have y ∈ a-rcosets I
    using assms(2) unfolding FactRing-def by simp
  then obtain v where y-def: y = I +> v v ∈ carrier R
    unfolding A-RCOSETS-def' by auto
  have I +> v ⊆ carrier R
```

```
    using y-def(2) a-r-coset-subset-G a-subset by presburger
  thus y ⊆ carrier R unfolding y-def by simp
qed

lemma (in ring) set-add-zero:
  assumes A ⊆ carrier R
  shows {𝟬} <+>_R A = A
proof –
  have {𝟬} <+>_R A = (⋃x∈A. {𝟬 ⊕ x})
    using assms unfolding set-add-def set-mult-def by simp
  also have ... = (⋃x∈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 ∈ carrier (K[X]) q ∈ 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 ∈ carrier R y ∈ carrier R
  assumes x ≠ 𝟬 y ≠ 𝟬
  assumes x ⊗ y = 𝟬
  shows False
  using integral assms by auto
```

```
lemma (in field) f-comm-group-2:
  assumes x ∈ carrier R
  assumes x ≠ 0
  shows ∃y∈carrier R − {0}. y ⊗ x = 1
proof −
  have x-unit: x ∈ 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 ∈ carrier R b ∈ carrier R
  assumes a divides b
  shows a divides (⊖ b)
proof −
  obtain r1 where r1-def: r1 ∈ carrier R a ⊗ r1 = b
    using assms by (auto simp:factor-def)

  have a ⊗ (⊖ r1) = ⊖ (a ⊗ r1)
    using assms(1) r1-def(1) by algebra
  also have ... = ⊖ b
    using r1-def(2) by simp
  finally have ⊖b = a ⊗ (⊖ r1) by simp
  moreover have ⊖r1 ∈ carrier R
    using r1-def(1) by simp
  ultimately show ?thesis
    by (auto simp:factor-def)
qed

lemma (in domain) div-sum:
  assumes a ∈ carrier R b ∈ carrier R c ∈ carrier R
  assumes a divides b
  assumes a divides c
  shows a divides (b ⊕ c)
proof −
  obtain r1 where r1-def: r1 ∈ carrier R a ⊗ r1 = b
    using assms by (auto simp:factor-def)

  obtain r2 where r2-def: r2 ∈ carrier R a ⊗ r2 = c
    using assms by (auto simp:factor-def)

  have a ⊗ (r1 ⊕ r2) = (a ⊗ r1) ⊕ (a ⊗ r2)
    using assms(1) r1-def(1) r2-def(1) by algebra
  also have ... = b ⊕ c
```

```
    using r1-def(2) r2-def(2) by simp
  finally have b ⊕ c = a ⊗ (r1 ⊕ r2) by simp
  moreover have r1 ⊕ r2 ∈ 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 ∈ carrier R b ∈ carrier R c ∈ carrier R
  assumes a divides b
  shows a divides (b ⊕ c) ⟷ a divides c
proof
  assume a divides (b ⊕ c)
  moreover have a divides (⊖ b)
    using div-neg assms(1,2,4) by simp
  ultimately have a divides ((b ⊕ c) ⊕ (⊖ 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 ⊕ c)
    using assms by (intro div-sum) auto
qed

lemma (in comm-monoid) irreducible-prod-unit:
  assumes f ∈ carrier G  x ∈ Units G
  shows irreducible G f = irreducible G (x ⊗ f) (is ?L = ?R)
proof
  assume ?L
  thus ?R using irreducible-prod-lI assms by auto
next
  have inv x ⊗ (x ⊗ f)  = (inv x  ⊗ x) ⊗ f
    using assms by (intro m-assoc[symmetric]) auto
  also have ... = f using assms by simp
  finally have 0: inv x ⊗ (x ⊗ f) = f by simp
  assume ?R
   hence irreducible G (inv x ⊗ (x ⊗ f) ) using irreducible-prod-lI
assms by blast
  thus ?L using 0 by simp
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* $[^\frown]_G$ *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* ⟶ $x \sim_G y$ ⟶ *x* = *y*) ∧
    (∀ *x* ∈ *carrier G*. *irreducible G x* ⟶ (∃ *y* ∈ *A*. $x \sim_G y$)))

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 ∈ carrier G set as ⊆ carrier G wfactors G as a
  shows factor-mset G a = fmset G as
proof −
  define H where H = {as. wfactors G as a ∧ set as ⊆ carrier G}
  have b:as ∈ H
    using H-def assms by simp

  have c: x ∈ H ⟹ y ∈ H ⟹ 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. ∃ as ∈ 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 ∈ carrier G
  shows ∃ as. factor-mset G a = fmset G as ∧ wfactors G as a ∧
    set as ⊆ carrier G
proof −
  obtain as where as-def: wfactors G as a set as ⊆ 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 ∈ carrier G
  assumes x ∈# factor-mset G a
  obtains y where
    y ∈ 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 ⊆ carrier G
    using factor-mset-aux assms by blast
  hence x ∈# fmset G as
```

using *assms* **by** *simp*
hence $x \in$ *assocs G ‘ set as*
using *assms as-def* **by** (*simp add:fmset-def*)
hence $\exists y.\ y \in$ *set as* $\wedge$ $x =$ *assocs G y*
by *auto*
moreover have $y \in$ *carrier G* $\wedge$ *irreducible G y*
if $y \in$ *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 $a \in$ *carrier G* $b \in$ *carrier G*
shows *factor-mset G* $(a \otimes b) =$ *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 $\subseteq$ *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 $\subseteq$ *carrier G*
using *factor-mset-aux assms*(*2*) **by** *blast*
have $a \otimes b \in$ *carrier G* **using** *assms* **by** *auto*
then obtain *cs* **where** *cs-def*:
factor-mset G $(a \otimes b) =$ *fmset G cs*
wfactors G cs $(a \otimes b)$
set cs $\subseteq$ *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* $\mathbf{1} = \{\#\}$
proof −
have *factor-mset G* $\mathbf{1} =$ *factor-mset G* $(\mathbf{1} \otimes \mathbf{1})$
by *simp*
also have ... $=$ *factor-mset G* $\mathbf{1}$ $+$ *factor-mset G* $\mathbf{1}$
by (*intro factor-mset-mult*, *auto*)
finally show *factor-mset G* $\mathbf{1} = \{\#\}$
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*) + ($\sum$ *a* ∈ *F*. *factor-mset G* (*f a*))
using *insert* **by** *simp*
also have ... = ($\sum$ *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 ($\sum$ *x* ∈ *F*. *f x*) = ($\sum$ *x* ∈ *F*. *image-mset h* (*f x*))
using *assms*
by (*induction F rule*:*finite-induct*, *simp*, *simp*)

lemma *decomp-mset*:
($\sum$ *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 ∈ carrier G set-mset R ⊆ carrier G
  assumes image-mset (assocs G) R = factor-mset G a
  shows a ∼ (⨂x∈set-mset R. x [^] count R x) (is a ∼ ?rhs)
proof –
  have b:irreducible G x if a:x ∈# R for x
  proof –
    have x-carr: x ∈ carrier G
      using a assms(2) by auto
    have assocs G x ∈ assocs G ` set-mset R
      using a by simp
    hence assocs G x ∈# factor-mset G a
      using assms(3) a in-image-mset by metis
    then obtain z where z-def:
      z ∈ carrier G irreducible G z assocs G x = assocs G z
      using factor-mset-set assms(1) by metis
    have z ∼ 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 =
    (∑x∈set-mset R. factor-mset G (x [^] count R x))
    using assms(2) by (subst factor-mset-prod, auto)
  also have ... =
    (∑x∈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 ... = (∑x∈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 ... = (∑x∈set-mset R.
    image-mset (assocs G) (replicate-mset (count R x) x))
    by simp
  also have ... = image-mset (assocs G)
    (∑x∈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 (⨂x∈set-mset R. x [^] count R x) ∈ 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 $x \notin$ *set-mset* A **by** *simp*
  hence $f\ x \notin f$ ‘ *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 $a \in$ *carrier G*
  shows
    finite {d. $d \in R \wedge$ *multiplicity G d a* > *0*}
    $a \sim$ ($\bigotimes d \in$ {d. $d \in R \wedge$ *multiplicity G d a* > *0*}.
        d [^] *multiplicity G d a*) (**is** $a \sim$ *?rhs*)
proof −
  have *r-1*: $R \subseteq$ {x. $x \in$ *carrier G* ∧ *irreducible G x*}
    using *assms*(*1*) **unfolding** *canonical-irreducibles-def* **by** *simp*
  have *r-2*: $\bigwedge x\ y.\ x \in R \Longrightarrow y \in R \Longrightarrow x \sim y \Longrightarrow x = y$
    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' = ($\sum d \in$ {d. $d \in R \wedge$ *multiplicity G d a* > *0*}.
    replicate-mset (*multiplicity G d a*) d)

  have *count* (*factor-mset G a*) (*assocs G x*) > *0*
    if $x \in R$ *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* ‘ {$d \in R$. *0* < *multiplicity G d a*}
    ⊆ *set-mset* (*factor-mset G a*)
    by (*intro image-subsetI*, *simp*)
  hence *a*:*finite* (*assocs G* ‘ {$d \in R$. *0* < *multiplicity G d a*})

using *finite-subset* **by** *auto*

show *finite* $\{d \in R.\ 0 < multiplicity\ G\ d\ a\}$
using *assocs-inj inj-on-subset*[*OF assocs-inj*]
by (*intro finite-imageD*[*OF a*], *simp*)

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

have *set-R′*: *set-mset* $R' = \{d \in R.\ 0 < multiplicity\ G\ d\ a\}$
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* $\exists x'.\ x' \in R \wedge x = assocs\ G\ x'$)
case *True*
then obtain *x′* **where** *x′-def*: $x' \in R$ $x = assocs\ G\ x'$
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*:$x \neq assocs\ G\ z$
if *a1*: $z \in carrier\ G$ **and** *a2*: *irreducible G z* **for** *z*
proof −
obtain *v* **where** *v-def*: $v \in R$ $z \sim v$
using *a1 a2 assms*(*1*)
unfolding *canonical-irreducibles-def* **by** *auto*
hence $z \in assocs\ G\ v$
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 $x \neq assocs\ G\ v$
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 $x \in$ *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*:$x \in$ *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 [^]$_{\textit{mult-of } R}$ (*order R*)
    by (*simp add*:*nat-pow-mult-of*)
  also have ... = x [^]$_{\textit{mult-of } R}$ (*order* (*mult-of R*)+*1*)
    using *carr-non-empty* **unfolding** *order-def*
    by (*intro arg-cong*[**where** *f*=λ*t*. x [^]$_{\textit{mult-of } R}$ *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 $x \in$ *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 = (λy. 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
    define z where ‹z = nat (y mod int n)›
    with n-gt-0 have z-def: ‹int z mod int n = y mod int n› ‹z < n›
      by (simp-all add: z-def nat-less-iff)
    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* = $k\ [\urcorner]_{add\text{-}monoid\ R(|carrier\ :=\ K|)}\ i$
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.ring-axioms* **by** *simp*
show *ring R* **using** *ring-axioms* **by** *simp*
show *int-embed R x* ∈ *carrier R* **if** *x* ∈ *carrier* $\mathcal{Z}$ **for** *x*
using *int-embed-closed* **by** *simp*
show *int-embed R* $(x \otimes_{\mathcal{Z}} y)$ = *int-embed R x* ⊗ *int-embed R y*
if *x* ∈ *carrier* $\mathcal{Z}$ *y* ∈ *carrier* $\mathcal{Z}$ **for** *x y*
using *int-embed-mult* **by** *simp*
show *int-embed R* $(x \oplus_{\mathcal{Z}} y)$ = *int-embed R x* ⊕ *int-embed R y*
if *x* ∈ *carrier* $\mathcal{Z}$ *y* ∈ *carrier* $\mathcal{Z}$ **for** *x y*
using *int-embed-add* **by** *simp*
show *int-embed R* $\mathbf{1}_{\mathcal{Z}} = \mathbf{1}$
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 $y \in UNIV$
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*: $y = u * int\ x + v$ $v \geq 0$ $v < int\ x$
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*) $\neq$ **0** $\longleftrightarrow$ *int-embed R x* $\neq$ **0** **for** *x*
using *int-embed-closed*
by (*cases x* > *0*, *simp*, *simp add*:*int-embed-inv*)

have *int-embed R x* $\neq$ **0** **if** *b*:*x* $\neq$ *0* **for** *x*
proof −
have *nat* (*abs x*) > *0* **using** *b* **by** *simp*
hence *int-embed R* (*nat* (*abs x*)) $\neq$ **0**
using *a* **by** *blast*
hence *int-embed R* (*abs x*) $\neq$ **0** **by** *simp*
thus *?thesis* **using** *c* **by** *simp*
qed
hence *int-embed R n* = **0** $\longleftrightarrow$ *n* = *0*
using *int-embed-zero* **by** *auto*
also have *n* = *0* $\longleftrightarrow$ *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.ring-axioms*
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 $(x \oplus y)\ [\urcorner]\ n =$
$(\bigoplus k \in \{..n\}.\ int\text{-}embed\ R\ (n\ choose\ k) \otimes x\ [\urcorner]\ k \otimes y\ [\urcorner]\ (n-k))$
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* (λ*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*: $card\ A \le n$
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 ⊕ y) [^] n =
  (⨁ A ∈ Pow {..<n}. x [^] (card A) ⊗ y [^] (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=λx. 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 ... = ($\bigoplus$ *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 ... =
($\bigoplus$ *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 ... =
($\bigoplus$ *A* ∈ ($\bigcup$ (*A* ‘ {..*n*})). *x* [^] (*card A*) ⊗ *y* [^] (*n*−*card A*))
using *card-bound* **by** (*intro finsum-cong′*, *auto simp add*:*A-def*)
also have ... =
($\bigoplus$ *k* ∈ {..*n*}. ($\bigoplus$ *A* ∈ *A k*. *x* [^] (*card A*) ⊗ *y* [^] (*n*−*card A*)))
using *fin-A disj-A* **by** (*subst add.finprod-UN-disjoint*, *auto*)
also have ... = ($\bigoplus$ *k* ∈ {..*n*}. ($\bigoplus$ *A* ∈ *A k*. *x* [^] *k* ⊗ *y* [^] (*n*−*k*)))
using *card-A* **by** (*intro finsum-cong′*, *auto*)
also have ... =
($\bigoplus$ *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 ... =
($\bigoplus$ *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 ring-axioms, 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* $\mathcal{Z}$ *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*$_{\mathcal{Z}}$ (*int* (*char R*))
  unfolding *cgenideal-def* **by** *auto*
also have ... = *Idl*$_{\mathcal{Z}}$ {*int* (*char R*)}
  using *int.cgenideal-eq-genideal* **by** *simp*
finally have *a*:*a-kernel* $\mathcal{Z}$ *R* (*int-embed R*) = *Idl*$_{\mathcal{Z}}$ {*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* (‹*pderiv*ı›) **where**
$pderiv_R$ *x* = *ring.normalize R* (
map (λ*i*. *int-embed R i* $\otimes_R$ *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 ∈ carrier (K[X])
  shows coeff (pderiv f) k = int-embed R (Suc k) ⊗ 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 (λi. 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 = 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 = 1_K[X]
  unfolding var-def pderiv-def
  by (simp add:univ-poly-one int-embed-def)

lemma pderiv-zero:
  shows pderiv 0_K[X] = 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 ⊕_K[X] g) = pderiv f ⊕_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 = (λi. 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 ⊕_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] 0K[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 0K[X] ⊖K[X] pderiv f
    by (subst p.l-neg, simp-all)
  also have ... = 0K[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 $f \in$ *carrier* $(K[X])$ $g \in$ *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* $\subseteq$ *carrier R*
using *assms*(*1*,*2*) *univ-poly-carrier*
using *subringE*(*1*)[*OF assms*(*1*)] *polynomial-incl* **by** *blast*
have *b*:*set g* $\subseteq$ *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* = (λi. *int-embed R i*)

have *a*[*simp*]:*?n k* $\in$ *carrier R* **for** *k*
using *int-embed-range*[*OF carrier-is-subring*] **by** *auto*
have *b*[*simp*]:*coeff f k* $\in$ *carrier R* **if** $f \in$ *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*) $\otimes$ *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*) $\otimes$
$(\bigoplus k \in \{..i+1\}.$ *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*) $\otimes$ (*coeff f k* $\otimes$ *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 ... =
    (⨁ k ∈ {..i+1}. ?n k ⊗ coeff f k ⊗ coeff g (i + 1 − k) ⊕
    coeff f k ⊗ (?n (i+1−k) ⊗ coeff g (i + 1 − k)))
    using Pi-def a b m-assoc m-comm
    by (intro finsum-cong' arg-cong2[where f=(⊕)], simp-all)
  also have ... =
    (⨁ k ∈ {..i+1}. ?n k ⊗ coeff f k ⊗ coeff g (i+1−k)) ⊕
    (⨁ k ∈ {..i+1}. coeff f k ⊗ (?n (i+1−k) ⊗ coeff g (i+1−k)))
    by (subst finsum-addf[symmetric], simp-all add:Pi-def)
  also have ... =
    (⨁ k∈insert 0 {1..i+1}. ?n k ⊗ coeff f k ⊗ coeff g (i+1−k)) ⊕
      (⨁ k∈insert (i+1) {..i}. coeff f k ⊗ (?n (i+1−k) ⊗ coeff g
(i+1−k)))
    using subringE(1)[OF assms(1)]
    by (intro arg-cong2[where f=(⊕)] finsum-cong')
      (auto simp:set-eq-iff)
  also have ... =
    (⨁ k ∈ {1..i+1}. ?n k ⊗ coeff f k ⊗ coeff g (i+1−k)) ⊕
    (⨁ k ∈ {..i}. coeff f k ⊗ (?n (i+1−k) ⊗ coeff g (i+1−k)))
    by (subst (1 2) finsum-insert, auto simp add:int-embed-zero)
  also have ... =
    (⨁ k ∈ Suc ‘ {..i}. ?n k ⊗ coeff f (k) ⊗ coeff g (i+1−k)) ⊕
    (⨁ k ∈ {..i}. coeff f k ⊗ (?n (i+1−k) ⊗ coeff g (i+1−k)))
    by (intro arg-cong2[where f=(⊕)] finsum-cong')
      (simp-all add:Pi-def atMost-atLeast0)
  also have ... =
    (⨁ k ∈ {..i}. ?n (k+1) ⊗ coeff f (k+1) ⊗ coeff g (i−k)) ⊕
    (⨁ k ∈ {..i}. coeff f k ⊗ (?n (i+1−k) ⊗ coeff g (i+1−k)))
    by (subst finsum-reindex, auto)
  also have ... =
    (⨁ k ∈ {..i}. coeff (pderiv f) k ⊗ coeff g (i−k)) ⊕
    (⨁ k ∈ {..i}. coeff f k ⊗ 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 ⊗K[X] g) i ⊕ coeff (f ⊗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 $pderiv\ (f\ [^{\frown}]_{K[X]}\ n) =$
$int\text{-}embed\ (K[X])\ n \otimes_{K[X]} f\ [^{\frown}]_{K[X]}\ (n-1) \otimes_{K[X]} pderiv\ f$
(**is** *?lhs* = *?rhs*)
proof −
interpret *p*: *cring* (*K*[*X*])
using *univ-poly-is-cring*[*OF assms*(*2*)] **by** *simp*

let *?n* = λ*n*. *int-embed* (*K*[*X*]) *n*

have [*simp*]: *?n i* ∈ *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 $pderiv\ (f\ [^{\frown}]_{K[X]}\ (m+1)) =$
$?n\ (m+1) \otimes_{K[X]} f\ [^{\frown}]_{K[X]}\ m \otimes_{K[X]} pderiv\ f$
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 $pderiv\ (f\ [^{\frown}]_{K\ [X]}\ (Suc\ m + 1)) =$
$pderiv\ (f\ [^{\frown}]_{K\ [X]}\ (m+1) \otimes_{K[X]} f)$
by *simp*
also have ... =
$pderiv\ (f\ [^{\frown}]_{K\ [X]}\ (m+1)) \otimes_{K[X]} f \oplus_{K[X]}$
$f\ [^{\frown}]_{K\ [X]}\ (m+1) \otimes_{K[X]} pderiv\ f$
using *assms*(*3*) **by** (*subst pderiv-mult*[*OF assms*(*2*)], *auto*)
also have ... =
$(?n\ (m+1) \otimes_{K\ [X]} f\ [^{\frown}]_{K\ [X]}\ m \otimes_{K\ [X]} pderiv\ f) \otimes_{K[X]} f$
$\oplus_{K[X]} f\ [^{\frown}]_{K\ [X]}\ (m+1) \otimes_{K[X]} pderiv\ f$
by (*subst Suc*(*1*), *simp*)
also have
$... = ?n\ (m+1) \otimes_{K[X]} (f\ [^{\frown}]_{K\ [X]}\ (m+1) \otimes_{K[X]} pderiv\ f)$
$\oplus_{K[X]} \mathbf{1}_{K\ [X]} \otimes_{K[X]} (f\ [^{\frown}]_{K\ [X]}\ (m+1) \otimes_{K[X]} pderiv\ f)$
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*) $\oplus_{K[X]}$ $\mathbf{1}_{K\ [X]}$) $\otimes_{K\ [X]}$
(*f* $[\widehat{\ }]_{K\ [X]}$ (*m*+*1*) $\otimes_{K\ [X]}$ *pderiv f*)
using *assms*(*3*) *pderiv-carr*[*OF assms*(*2*)]
by (*subst p.l-distr*[*symmetric*], *simp-all*)
also have ... =
($\mathbf{1}_{K\ [X]}$ $\oplus_{K[X]}$ *?n* (*m*+*1*)) $\otimes_{K\ [X]}$
(*f* $[\widehat{\ }]_{K\ [X]}$ (*m*+*1*) $\otimes_{K\ [X]}$ *pderiv f*)
using *assms*(*3*) *pderiv-carr*[*OF assms*(*2*)]
by (*subst p.a-comm*, *simp-all*)
also have ... = *?n* (*1*+ *Suc m*)
$\otimes_{K\ [X]}$ *f* $[\widehat{\ }]_{K\ [X]}$ (*Suc m*) $\otimes_{K\ [X]}$ *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* $[\widehat{\ }]_{K[X]}$ *n*) =
int-embed (*K*[*X*]) *n* $\otimes_{K[X]}$ *X* $[\widehat{\ }]_{K[X]}$ (*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 $finprod\ R\ f\ A = finprod\ (mult\text{-}of\ R)\ f\ A$
  using *assms* **by** (*induction A rule*:*finite-induct*, *auto*)

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

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

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

definition *monic-irreducible-poly* **where**
  monic-irreducible-poly R f =
    $(monic\text{-}poly\ R\ f \land pirreducible_R\ (carrier\ R)\ f)$

abbreviation $m\text{-}i\text{-}p \equiv monic\text{-}irreducible\text{-}poly$

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** $P \equiv K[X]$

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 $a \in carrier\ (mult\text{-}of\ P)$
  assumes $b \in carrier\ (mult\text{-}of\ P)$
  assumes $c \in carrier\ (mult\text{-}of\ P)$
  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* $\longleftrightarrow a \otimes_P c$ $divides_{mult\text{-}of\ P}$ $b \otimes_P c$
    unfolding *pdivides-def* **using** *p.divides-imp-divides-mult a b*
    by (*meson divides-mult-imp-divides*)
  also have ... $\longleftrightarrow a$ $divides_{mult\text{-}of\ P}$ 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 $x \in carrier\ (mult\text{-}of\ P)$
  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* ≠ [] **by** *auto*

  have *d*: *length g* ≤ *n*

```
    using  assms(3) unfolding n-def by simp

  have lead-coeff (f ⊕P g) = coeff (f ⊕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 1P
proof −
  have 1P ∈ 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* $(f \otimes_P g)$
proof –
have *lead-coeff* $(f \otimes_P g) =$ *lead-coeff f* $\otimes_R$ *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* $(f \otimes_P g) = \mathbf{1}_R$ **by** *simp*
moreover have $(f \otimes_P g) \in$ *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* $(f\ [\char`\^]_P\ (n::nat))$
using *assms monic-poly-one monic-poly-mult*
by (*induction n*, *auto*)

lemma *monic-poly-prod*:
assumes *finite A*
assumes $\bigwedge x.\ x \in A \Longrightarrow$ *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*: $f \in F \to$ *carrier P*
using *insert monic-poly-carr* **by** *simp*
have *b*: $f\ x \in$ *carrier P*
using *insert monic-poly-carr* **by** *simp*
have *monic-poly R* $(f\ x \otimes_P$ *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 $f \sim_{(mult\text{-}of\ P)} g$
shows $f = g$
proof –
obtain *u* **where** *u-def*: $f = g \otimes_P u$ $u \in$ *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*: $y = x \otimes_{(mult\text{-}of\ P)} z$
by (*simp add*:*y-def*)
hence *y-carr*: $y \in carrier\ (mult\text{-}of\ P)$
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* $y = \mathbf{1}_R$
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 $y \sim_{(mult\text{-}of\ P)} x$
by (*intro p.mult-of.associatedI2*[*OF z-unit*] *y-def x-carr*)
hence $x \sim_{(mult\text{-}of\ P)} y$
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*) $\{d.\ monic\text{-}irreducible\text{-}poly\ R\ d\}$
(**is** *canonical-irreducibles* (*mult-of P*) *?S*)
proof −
have *sp-1*:
$?S \subseteq \{x \in carrier\ (mult\text{-}of\ P).\ irreducible\ (mult\text{-}of\ P)\ x\}$
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 $x \in ?S$ $y \in ?S$ $x \sim_{(mult\text{-}of\ P)} y$ **for** *x y*
using *that monic-poly-not-assoc*
by (*simp add*:*monic-irreducible-poly-def*)

have *sp-3*: $\exists y \in ?S.\ x \sim_{(mult\text{-}of\ P)} y$
if $x \in carrier\ (mult\text{-}of\ P)$ *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*:
    $a = (\bigotimes_P d \in \{d.\ monic\text{-}irreducible\text{-}poly\ R\ d \wedge pmult\ d\ a > 0\}.$
      $d\ [\urcorner]_P\ pmult\ d\ a)$ (**is** *?lhs = ?rhs*)
    and *factor-monic-poly-fin*:
      finite $\{d.\ monic\text{-}irreducible\text{-}poly\ R\ d \wedge pmult\ d\ a > 0\}$
proof −
  let $?S = \{d.\ monic\text{-}irreducible\text{-}poly\ R\ d\}$
  let $?T = \{d.\ monic\text{-}irreducible\text{-}poly\ R\ d \wedge pmult\ d\ a > 0\}$
  let *?mip = monic-irreducible-poly R*

  have *sp-4*: $a \in carrier\ (mult\text{-}of\ P)$
    using *assms monic-poly-carr-2*
    unfolding *monic-irreducible-poly-def* **by** *simp*

  have *b-1*: $x \in carrier\ (mult\text{-}of\ P)$ **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*:$x \in carrier\ P$ **if** *?mip x* **for** *x*
    using *that monic-poly-carr*
    unfolding *monic-irreducible-poly-def*
    by *simp*

  have *a-carr*: $a \in carrier\ P - \{\mathbf{0}_P\}$
    using *sp-4* **by** *simp*

  have $?T = \{d.\ ?mip\ d \wedge multiplicity\ (mult\text{-}of\ P)\ d\ a > 0\}$
    by (*simp add*:*pmult-def*)
  also have ... $= \{d \in ?S.\ multiplicity\ (mult\text{-}of\ P)\ d\ a > 0\}$
    using *p.mult-of.multiplicity-gt-0-iff*[*OF b-1 b-2 sp-4*]
    by (*intro order-antisym subsetI*, *auto*)
  finally have *t*:$?T = \{d \in ?S.\ multiplicity\ (mult\text{-}of\ P)\ d\ a > 0\}$
    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*:$x\ [\urcorner]_P\ (n::nat) \in carrier\ (mult\text{-}of\ P)$ **if** *?mip x* **for** *x n*
  proof −
    have *monic-poly R* $(x\ [\urcorner]_P\ n)$
      using *that monic-poly-pow*

unfolding *monic-irreducible-poly-def* **by** *auto*
thus *?thesis*
using *monic-poly-carr-2* **by** *simp*
qed

have *?lhs* $\sim_{(mult\text{-}of\ P)}$
finprod (*mult-of P*)
($\lambda d.\ d$ $[\widehat{\ }]_{(mult\text{-}of\ P)}$ (*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*) ($\lambda d.\ d$ $[\widehat{\ }]_P$ (*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* $\sim_{(mult\text{-}of\ P)}$ *?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′ ($\lambda d.$ *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* − $\{\mathbf{0}_P\}$ **if** *?mip d* **for** *d*
using *that monic-poly-carr-2*
unfolding *monic-irreducible-poly-def* **by** *simp*
have *a*: *d* $[\widehat{\ }]_P$ *n* ∈ *carrier P* − $\{\mathbf{0}_P\}$ **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 ($\bigotimes_P d \in$ {*d. ?mip d* ∧ *pmult d f* > *0*}. *d* $[\widehat{\ }]_P$ *pmult d f*)
using *factor-monic-poly*[*OF assms*(*1*)] **by** *simp*
also have ... =
($\sum i \in$ {*d. ?mip d* ∧ *0* < *pmult d f*}. *degree* (*i* $[\widehat{\ }]_P$ *pmult i f*))
using *a assms*(*1*)

by (*subst degree-prod*[*OF factor-monic-poly-fin*])
(*simp-all add*:*Pi-def*)
also have ... =
($\sum$ *i*∈{*d*. *?mip d* ∧ *0* < *pmult d f*}. *degree i* ∗ *pmult i f*)
using *b degree-pow* **by** (*intro sum.cong*, *auto*)
also have ... =
($\sum$ *d*∈{*d*. *?mip d* ∧ *0* < *pmult d f*}. *pmult d f* ∗ *degree d*)
by (*simp add*:*mult.commute*)
also have ... =
sum′ (λ*d*. *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′* (λ*d*. *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* = [$\mathbf{1}_R$, $\ominus_R$*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 ∈ carrier P − {0_P}
  shows pmult d f ≥ k ⟷ d [^]_P k pdivides f
proof −
  have a:f ∈ carrier (mult-of P)
    using assms(2) by simp
  have b: d ∈ 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 [^]_P k ∈ carrier P using b by simp

  have pmult d f ≥ k ⟷ d [^]_(mult-of P) k divides_(mult-of P) f
    unfolding pmult-def
    by (intro p.mult-of.multiplicity-ge-iff a b c)
  also have ... ⟷ d [^]_P k 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 ∈ carrier P − {0_P}
  shows pmult d f ≥ 1 ⟷ d pdivides f
proof −
  have d ∈ 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* $\in$ *carrier* (*mult-of P*) *g* $\in$ *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* $\in$ *ring-iso R S domain R domain S*
shows *monic-poly S* (*map h f*)
proof −
have *c*: *h* $\in$ *ring-hom R S*
using *assms*(*2*) *ring-iso-def* **by** *auto*
have *e*: *f* $\in$ *carrier* (*poly-ring R*)
using *assms*(*1*) **unfolding** *monic-poly-def* **by** *simp*

have *a:f* $\neq$ []
using *assms*(*1*) **unfolding** *monic-poly-def* **by** *simp*
hence *map h f* $\neq$ [] **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* $\in$ *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 $\in$ *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.ring-axioms 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 ⊆ ((λq. 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 ∈ ((λq. 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 1_P) pdivides (X [^]_P m ⊖_P 1_P) ⟷ 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 1_P) = ((X [^]_P l)[^]_P q) ⊖_P 1_P
    using var-closed
    by (subst p.nat-pow-pow, simp-all add:algebra-simps)
  also have ... =
    (X [^]_P l ⊖_P 1_P) ⊗_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*: $(X\ [\urcorner]_P\ (q{*}l) \ominus_P \mathbf{1}_P) =$
$(X\ [\urcorner]_P\ l \ominus_P \mathbf{1}_P) \otimes_P (\bigoplus{}_P i{\in}\{..{<}q\}.\ (X_R\ [\urcorner]_P\ l)\ [\urcorner]_P\ i)$
by *simp*

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

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

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

have *xn-m-1-deg'*: *degree* $(X\ [\urcorner]_P\ n \ominus_P \mathbf{1}_P) = n$
if $n > 0$ **for** n :: *nat*
proof −
have *degree* $(X\ [\urcorner]_P\ n \ominus_P \mathbf{1}_P) =$ *degree* $(X\ [\urcorner]_P\ n \oplus_P \ominus_P \mathbf{1}_P)$
by (*simp add*:*a-minus-def*)
also have ... = *max* (*degree* $(X\ [\urcorner]_P\ n)$) (*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* $(X\ [\urcorner]_P\ n \ominus_P \mathbf{1}_P) = n$ **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 1P) = degree (0P)
      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 1P) > degree (X_R [^]P r ⊖P 1P)
    using r-range unfolding xn-m-1-deg by simp

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

  have ?lhs ⟷ (X [^]P l ⊖P 1P) pdivides ?x
    by (subst a, simp)
  also have ... ⟷ (X [^]P l ⊖P 1P) pdivides (X [^]P r ⊖P 1P)
    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 1P) pdivides 0P
      unfolding univ-poly-zero
      by (intro pdivides-zero xn-m-1-carr)
    also have ... = (X [^]P r ⊖P 1P)
      by (simp add:a-minus-def True) algebra
    finally show ?thesis using True by simp
  next
    case False
    hence degree (X [^]P r ⊖P 1P) > 0 using xn-m-1-deg by simp
    hence X [^]P r ⊖P 1P ≠ [] by auto
    hence ¬(X [^]P l ⊖P 1P) pdivides (X [^]P r ⊖P 1P)
      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* $\neq$ $\mathbf{0}_P$
proof −
have *degree* (*gauss-poly R n*) $\neq$ *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 $[\widehat{\ }]_P$ n)
by (*intro monic-poly-pow monic-poly-var*)
moreover have $\ominus_P$ $X \in$ *carrier P*
using *var-closed* **by** *simp*
moreover have *degree* ($\ominus_P$ X) $<$ *degree* (X $[\widehat{\ }]_P$ 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 $(x-1) * (\sum i \in \{..<q\}.\ x\widehat{\ }i) = x\widehat{\ }q-1$
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 $(a\ \widehat{\ }\ l - 1)$ *dvd* $(a\ \widehat{\ }\ m - 1) \longleftrightarrow l$ *dvd* m
(**is** *?lhs* $\longleftrightarrow$ *?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*: $r < l$ $r \geq 0$
using *assms* **by** (*auto simp add*:*q-def r-def*)

have $a\ \widehat{\ }(l * q) - 1 = (a\ \widehat{\ }\ l)\ \widehat{\ }\ q - 1$
by (*simp add*: *power-mult*)
also have ... $= (a\widehat{\ }l - 1) * (\sum i \in \{..<q\}.\ (a\widehat{\ }l)\widehat{\ }i)$
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 1P) ⊗P X
    using  gauss-poly-factor a1 b1 by simp
  also have ... ⟷
    (X [^]P (a^n-1) ⊖P 1P) pdivides
    (X [^]P (a^m-1) ⊖P 1P)
    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 λx. 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*) $[\widehat{\ }]_K$ (*order* $R\widehat{\ }n$) = φ(*poly-of-const x*)
if *k-cycle-1*: $x \in$ *carrier R* **for** *x*
proof −
have φ (*poly-of-const x*) $[\widehat{\ }]_K$ (*order* $R\widehat{\ }n$) =
φ (*poly-of-const* (x $[\widehat{\ }]_R$ (*order* $R\widehat{\ }n$)))
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* $\geq$ *1*
if *roots-g1-assms*: *degree d* = *1 monic-irreducible-poly K d* **for** *d*
proof −
obtain *x* **where** *x-def*: $x \in$ *carrier K d* = $[\mathbf{1}_K, \ominus_K x]$
using *f.degree-one-monic-poly roots-g1-assms* **by** *auto*
interpret *x*:*ring-hom-cring poly-ring K K* ($\lambda p.$ *f.eval p x*)
by (*intro fp.eval-cring-hom x-def*)
have *ring.eval K ?g1 x* = $\mathbf{0}_K$
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 $[\mathbf{1}_K, \ominus_K x]$ *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* $\geq$ *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* $R\widehat{\ }n$)
have (φ *X*) $[\widehat{\ }]_K$ (*order* $R\widehat{\ }n$) $\ominus_K$ (φ X_R) =
φ (*gauss-poly R* (*order* $R\widehat{\ }n$))
unfolding *gauss-poly-def a-minus-def* **using** *var-closed*
by (*simp add*: *h.hom-nat-pow*)
also have ... = $\mathbf{0}_K$
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) $[\widehat{\ }]_K$ (*order* $R\widehat{\ }n$) $\ominus_K$ (φ X_R) = $\mathbf{0}_K$
by *simp*
hence *g*:(φ *X*) $[\widehat{\ }]_K$ (*order* $R\widehat{\ }n$) = (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* = [$\mathbf{1}_K$, $\ominus_K$ *y*]
using *f.degree-one-monic-poly roots-g2-assms* **by** *auto*

interpret *x:ring-hom-cring poly-ring K K* (λ*p. 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* = λ*i. 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* (λ*x*. 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* (λ*x*. 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* = $\mathbf{0}_K$

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* $\geq$ *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* ($\lambda x.\ \ominus_K x$) (*carrier K*)
by (*intro inj-onI*, *metis f.minus-minus*)
let *?mip* = *monic-irreducible-poly K*

have *sum′* ($\lambda d.$ $pmult_K$ *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* (($\lambda k.$ [$\mathbf{1}_K$, $\ominus_K$ *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* ($\lambda d.$ *1*) {*d. ?mip d* ∧ *degree d* = *1*}
by *simp*
also have ... = *sum′* ($\lambda d.$ *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′* ($\lambda d.$ *of-bool* (*degree d* = *1*)) {*d. ?mip d*}
by (*intro sum.mono-neutral-cong-left′ subsetI*, *simp-all*)
also have ... ≤ *sum′* ($\lambda d.$ *of-bool* (*degree d* = *1*)) {*d. ?mip d*}
by *simp*
finally have *sum′* ($\lambda d.$ $pmult_K$ *d ?g1* ∗ *degree d*) {*d. ?mip d*}
≤ *sum′* ($\lambda d.$ *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* = [$\mathbf{1}_K$, $\ominus_K$ *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** *auto*
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** *force*
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) $[^\frown]_K$ (*order R*^*degree f*) $\ominus_K$ (φ X_R)
unfolding *gauss-poly-def a-minus-def* **using** *var-closed*
by (*simp add*: *h.hom-nat-pow*)
also have ... = $\mathbf{0}_K$
using *c d* **by** *simp*
finally have φ (*gauss-poly R* (*order R*^*degree f*)) = $\mathbf{0}_K$ **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 $q \sim_{\textit{mult-of } P} p$
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* $divides_{\textit{mult-of } P}$ *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* $divides_{\textit{mult-of } P}$ *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-poly R (order R^n)) = of-bool (degree f dvd n)
proof (cases degree f dvd n)
  case True
  let ?g = gauss-poly R (order R^n)
  have f-carr: f ∈ carrier P f ≠ []
    using assms(2)
    unfolding monic-irreducible-poly-def monic-poly-def
    by auto

  have o2: order R^n > 1
    using finite-field-min-order assms(1) one-less-power by blast
  hence o21: order R^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 ≥ 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 $(f\ [\widehat{\ }]_P\ (2::nat))\ pdivides_R\ ?g$
  using *gauss-poly-carr gauss-poly-not-zero*[*OF o2*]
  by (*subst* (*asm*) *multiplicity-ge-iff*[*OF assms*(*2*)]) *simp-all*
hence $(f\ [\widehat{\ }]_P\ (2::nat))\ divides_{mult\text{-}of\ P}\ ?g$
  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*:
  $h \in carrier\ (mult\text{-}of\ P)$
  $?g = f\ [\widehat{\ }]_P\ (2::nat) \otimes_P h$
  using *dividesD* **by** *auto*
have $\ominus_P \mathbf{1}_P = int\text{-}embed\ P\ (order\ R\ \widehat{\ }\ n)$
  $\otimes_P (X_R\ [\widehat{\ }]_P\ (order\ R\ \widehat{\ }\ n-1)) \ominus_P \mathbf{1}_P$
  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 ... $= pderiv_R\ (X_R\ [\widehat{\ }]_P\ order\ R\ \widehat{\ }\ n) \ominus_P pderiv_R\ X_R$
  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 ... $= pderiv_R\ (f\ [\widehat{\ }]_P\ (2::nat) \otimes_P h)$
  using *h-def*(*2*) **by** *simp*
also have ... $= pderiv_R\ (f\ [\widehat{\ }]_P\ (2::nat)) \otimes_P h$
  $\oplus_P (f\ [\widehat{\ }]_P\ (2::nat)) \otimes_P pderiv_R\ h$
  using *f-carr h-def*
  by (*intro pderiv-mult*, *simp-all*)
also have ... $= int\text{-}embed\ P\ 2 \otimes_P f\ \otimes_P pderiv_R\ f \otimes_P h$
  $\oplus_P f \otimes_P f \otimes_P pderiv_R\ h$
  using *f-carr*
  by (*subst pderiv-pow*, *simp-all add*:*numeral-eq-Suc*)
also have ... $= f \otimes_P (int\text{-}embed\ P\ 2 \otimes_P pderiv_R\ f \otimes_P h)$
  $\oplus_P f \otimes_P (f \otimes_P pderiv_R\ h)$
  using *f-carr pderiv-carr h-def p.int-embed-closed*
  apply (*intro arg-cong2*[**where** $f=(\oplus_P)$])
  by (*subst p.m-comm*, *simp-all add*:*p.m-assoc*)
also have ... $= f \otimes_P$
  $(int\text{-}embed\ P\ 2 \otimes_P pderiv_R\ f \otimes_P h \oplus_P f \otimes_P pderiv_R\ h)$
  using *f-carr pderiv-carr h-def p.int-embed-closed*
  by (*subst p.r-distr*, *simp-all*)
finally have $\ominus_P \mathbf{1}_P = f \otimes_P$
  $(int\text{-}embed\ P\ 2 \otimes_P pderiv_R\ f \otimes_P h \oplus_P f \otimes_P pderiv_R\ h)$
  (**is** $- = f \otimes_P ?q$)
  by *simp*

hence $f\ pdivides_R \ominus_P \mathbf{1}_P$

```
      unfolding factor-def pdivides-def
      using f-carr pderiv-carr h-def p.int-embed-closed
      by auto
    moreover have ⊖P 1P ≠ 0P by simp
    ultimately have  degree f ≤ degree (⊖P 1P)
      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 pmultR 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 pdividesR gauss-poly R (order R^n))
    using div-gauss-poly-iff[OF assms] False by simp
  hence pmultR 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 $?lhs = dirichlet\text{-}prod\ moebius\text{-}mu\ (\lambda x.\ int\ (order\ R)\ \hat{}\ x)\ n$
    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-commutes 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) card-irred-gt-0:
```

assumes *d* > *0*
shows *real*(*order R*)^*d* / (*2*∗*real d*) ≤ *real* (*card* {*f*. *monic-irreducible-poly R f* ∧ *degree f* = *d*})
(**is** *?L* ≤ *?R*)
proof −
consider (*a*) *d* = *1* | (*b*) *d* = *2* | (*c*) *d* > *2* **using** *assms* **by** *linarith*
thus *?thesis*
proof (*cases*)
case *a*
hence *?L* = *real* (*order R*)/*2* **by** *simp*
also have ... ≤ *real* (*order R*) **using** *finite-field-min-order* **by** *simp*
also have ... = *?R* **unfolding** *a card-irred-1* **by** *simp*
finally show *?thesis* **by** *simp*
next
case *b*
hence *?L* = *real* (*order R*^*2*)/*4* + *0* **by** *simp*
also have ... ≤ *real* (*order R*^*2*)/*4* + *real* (*order R*)/*2* ∗ (*real* (*order R*)/*2* − *1*)
using *finite-field-min-order* **by** (*intro add-mono mult-nonneg-nonneg*) *auto*
also have ... = (*real* (*order R*^*2*) − *real* (*order R*))/*2*
by (*simp add*:*algebra-simps power2-eq-square*)
also have ... = *?R* **unfolding** *b card-irred-2* **by** *simp*
finally show *?thesis* **by** *simp*
next
case *c* **thus** *?thesis* **by** (*rule card-irred-gt-2*)
qed
qed

lemma (**in** *finite-field*) *exist-irred*:
assumes *n* > *0*
obtains *f* **where** *monic-irreducible-poly R f degree f* = *n*
proof −
have *0* < *real*(*order R*)^*n* / (*2*∗*real n*)
using *finite-field-min-order assms*
by (*intro divide-pos-pos mult-pos-pos zero-less-power*) *auto*
also have ... ≤ *real* (*card* {*f*. *monic-irreducible-poly R f* ∧ *degree f* = *n*})
(**is** - ≤ *real*(*card ?A*))
by (*intro card-irred-gt-0 assms*)
finally have *0* < *card* {*f*. *monic-irreducible-poly R f* ∧ *degree f* = *n*}
by *auto*
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 $p \equiv char\ R$
  assumes $f \in carrier\ (poly\text{-}ring\ (ZFact\ p))$
  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-iso R) ∈ ring-hom ?P ((char-subring R)[X])
  by (simp add:ring-iso-def)
moreover have (λp. eval p x) ∈ ring-hom ((char-subring R)[X]) R
  using eval-is-hom char-ring-is-subring assms(5) by simp
ultimately have
  (λp. eval p x) ∘ map (char-iso R) ∈ ring-hom ?P R
  using ring-hom-trans by blast
hence a:(λp. eval (map (char-iso R) p) x) ∈ ring-hom ?P R
  by (simp add:comp-def)
interpret h:ring-hom-ring ?P R (λp. eval (map (char-iso R) p) x)
  by (intro ring-hom-ringI2 pzf.ring-axioms a ring-axioms)

let ?h = (λp. 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 (λy. the-elem ((λp. eval (map (char-iso R) p) x) ‘ y))
  ∈ ring-hom (?P Quot ?J) R
  using h.the-elem-hom by simp
moreover have (λy. ?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 (λy. the-elem (?h ‘ y)) ∘ (λy. ?J <+>_?P y)
    ∈ ring-hom (Rupt_(ZFact p) (carrier (ZFact p)) f) R
    using ring-hom-trans by blast
  hence b: (λy. 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 ∈ carrier (Rupt_ZFact p (carrier (ZFact p)) f)
    for y
  proof −
    have y-range: y ⊆ carrier ?P
      using rupt-carr that by simp
    have ?h ‘ y = {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 (λy. the-elem (?h ‘ y)) ∈
    ring-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 ∈ carrier (K[X]) g ∈ carrier (K[X])
  shows f pdivides g ⟷ 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 ⟷ f divides_K[X] g
    using pdivides-iff-shell[OF assms] by simp
  also have ... ⟷ (∃ x ∈ carrier (K[X]). f ⊗_K[X] x = g)
    unfolding pdivides-def factor-def by auto
  also have ... ⟷
    (∃ x ∈ carrier (poly-ring ?S). f ⊗_poly-ring ?S x = g)
    using univ-poly-consistent[OF a] by simp
  also have ... ⟷ f divides_poly-ring ?S g
    unfolding pdivides-def factor-def by auto
  also have ... ⟷ 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 = 0 x ∈ carrier R
proof −
  define τ :: 'a list ⇒ 'a list where τ = 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-ring ?K card K ^ degree f ⊖_poly-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 ^ *n n* > *0*
using *finite-field.finite-field-order*[*OF assms*(*1*)] **by** *auto*
obtain *m* **where** *o2*: *order* F_2 = *char* F_2 ^ *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 ^ *n* = *char* F_2 ^ *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

8 Rabin's test for irreducible polynomials

theory *Rabin-Irreducibility-Test*
imports *Card-Irreducible-Polynomials-Aux*
begin

This section introduces an effective test for irreducibility of polynomials (in finite fields) based on Rabin [5].

definition *pcoprime* :: - ⇒ *'a list* ⇒ *'a list* ⇒ *bool* (‹*pcoprime*ı›)
where *pcoprime*$_R$ *p q* =
(∀ *r* ∈ *carrier* (*poly-ring R*). *r pdivides*$_R$ *p* ∧ *r pdivides*$_R$ *q* ⟶ *degree r* = *0*)

lemma *pcoprimeI*:
assumes ⋀*r*. *r* ∈ *carrier* (*poly-ring R*) ⟹ *r pdivides* $_R$ *p* ⟹ *r pdivides*$_R$ *q* ⟹ *degree r* = *0*
shows *pcoprime*$_R$ *p q*
using *assms* **unfolding** *pcoprime-def* **by** *auto*

context *field*
begin

interpretation *r:polynomial-ring R* (*carrier R*)

unfolding *polynomial-ring-def polynomial-ring-axioms-def*
using *carrier-is-subfield field-axioms* **by** *force*

lemma *pcoprime-one*: $pcoprime_R\ p\ \mathbf{1}_{poly\text{-}ring\ R}$
proof (*rule pcoprimeI*)
fix *r*
assume *r-carr*: $r \in carrier\ (poly\text{-}ring\ R)$
moreover assume $r\ pdivides\ _R\ \mathbf{1}_{poly\text{-}ring\ R}$
moreover have $\mathbf{1}_{poly\text{-}ring\ R} \neq []$ **by** (*simp add*:*univ-poly-one*)
ultimately have $degree\ r \leq degree\ \mathbf{1}_{poly\text{-}ring\ R}$
by (*intro pdivides-imp-degree-le*[*OF carrier-is-subring*] *r-carr*) *auto*
also have ... = *0* **by** (*simp add*:*univ-poly-one*)
finally show *degree r* = *0* **by** *auto*
qed

lemma *pcoprime-left-factor*:
assumes $x \in carrier\ (poly\text{-}ring\ R)$
assumes $y \in carrier\ (poly\text{-}ring\ R)$
assumes $z \in carrier\ (poly\text{-}ring\ R)$
assumes $pcoprime_R\ (x \otimes_{poly\text{-}ring\ R} y)\ z$
shows $pcoprime_R\ x\ z$
proof (*rule pcoprimeI*)
fix *r*
assume *r-carr*: $r \in carrier\ (poly\text{-}ring\ R)$
assume $r\ pdivides\ _R\ x$
hence $r\ pdivides\ _R\ (x \otimes_{poly\text{-}ring\ R} y)$
using *assms*(*1*,*2*) *r-carr r.p.divides-prod-r* **unfolding** *pdivides-def*
by *simp*
moreover assume $r\ pdivides\ _R\ z$
ultimately show *degree r* = *0* **using** *assms*(*4*) *r-carr* **unfolding**
pcoprime-def **by** *simp*
qed

lemma *pcoprime-sym*:
shows *pcoprime x y* = *pcoprime y x*
unfolding *pcoprime-def* **by** *auto*

lemma *pcoprime-left-assoc-cong-aux*:
assumes $x1 \in carrier\ (poly\text{-}ring\ R)$ $x2 \in carrier\ (poly\text{-}ring\ R)$
assumes $x2 \sim_{poly\text{-}ring\ R} x1$
assumes $y \in carrier\ (poly\text{-}ring\ R)$
assumes *pcoprime x1 y*
shows *pcoprime x2 y*
using *assms r.p.divides-cong-r*[*OF - assms*(*3*)] **unfolding** *pcoprime-def*
pdivides-def **by** *simp*

lemma *pcoprime-left-assoc-cong*:
assumes $x1 \in carrier\ (poly\text{-}ring\ R)$ $x2 \in carrier\ (poly\text{-}ring\ R)$
assumes $x1 \sim_{poly\text{-}ring\ R} x2$

assumes *y* ∈ *carrier* (*poly-ring R*)
shows *pcoprime x1 y* = *pcoprime x2 y*
using *assms pcoprime-left-assoc-cong-aux r.p.associated-sym* **by** *metis*

lemma *pcoprime-right-assoc-cong*:
assumes *x1* ∈ *carrier* (*poly-ring R*) *x2* ∈ *carrier* (*poly-ring R*)
assumes $x1 \sim_{poly\text{-}ring\ R} x2$
assumes *y* ∈ *carrier* (*poly-ring R*)
shows *pcoprime y x1* = *pcoprime y x2*
using *assms pcoprime-sym pcoprime-left-assoc-cong* **by** *metis*

lemma *pcoprime-step*:
assumes *f* ∈ *carrier* (*poly-ring R*)
assumes *g* ∈ *carrier* (*poly-ring R*)
shows *pcoprime f g* ⟷ *pcoprime g* (*f pmod g*)
proof −
have *d pdivides f* ⟷ *d pdivides* (*f pmod g*) **if** *d* ∈ *carrier* (*poly-ring R*) *d pdivides g* **for** *d*
proof −
have *d pdivides f* ⟷ *d pdivides* ($g \otimes_{r.P}$ (*f pdiv g*) $\oplus_{r.P}$ (*f pmod g*))
using *pdiv-pmod*[*OF carrier-is-subfield assms*] **by** *simp*
also have ... ⟷ *d pdivides* ((*f pmod g*))
using *that assms long-division-closed*[*OF carrier-is-subfield*] *r.p.divides-prod-r*
unfolding *pdivides-def* **by** (*intro r.p.div-sum-iff*) *simp-all*
finally show *?thesis* **by** *simp*
qed
hence *d pdivides f* ∧ *d pdivides g* ⟷ *d pdivides g* ∧ *d pdivides* (*f pmod g*)
if *d* ∈ *carrier* (*poly-ring R*) **for** *d*
using *that* **by** *auto*
thus *?thesis*
unfolding *pcoprime-def* **by** *auto*
qed

lemma *pcoprime-zero-iff*:
assumes *f* ∈ *carrier* (*poly-ring R*)
shows *pcoprime f* [] ⟷ *length f* = *1*
proof −
consider (*i*) *length f* = *0* | (*ii*) *length f* = *1* | (*iii*) *length f* > *1*
by *linarith*
thus *?thesis*
proof (*cases*)
case *i*
hence *f* = [] **by** *simp*
moreover have *X pdivides* [] **using** *r.pdivides-zero r.var-closed*(*1*) **by** *blast*
moreover have *degree X* = *1* **using** *degree-var* **by** *simp*
ultimately have ¬*pcoprime f* [] **using** *r.var-closed*(*1*) **unfolding**

pcoprime-def **by** *auto*
then show *?thesis* **using** *i* **by** *auto*
next
case *ii*
hence $f \neq []$ *degree f* $= 0$ **by** *auto*
hence *degree d* $= 0$ **if** *d pdivides f d* $\in$ *carrier* (*poly-ring R*) **for** *d*
using *that*(*1*) *pdivides-imp-degree-le*[*OF carrier-is-subring that*(*2*) *assms*] **by** *simp*
hence *pcoprime f* [] **unfolding** *pcoprime-def* **by** *auto*
then show *?thesis* **using** *ii* **by** *simp*
next
case *iii*
have *f pdivides f* **using** *assms* **unfolding** *pdivides-def* **by** *simp*
moreover have *f pdivides* [] **using** *assms r.pdivides-zero* **by** *blast*
moreover have *degree f* > 0 **using** *iii* **by** *simp*
ultimately have $\neg$*pcoprime f* [] **using** *assms* **unfolding** *pcoprime-def* **by** *auto*
then show *?thesis* **using** *iii* **by** *auto*
qed
qed

end

context *finite-field*
begin

interpretation *r*:*polynomial-ring R* (*carrier R*)
unfolding *polynomial-ring-def polynomial-ring-axioms-def*
using *carrier-is-subfield field-axioms* **by** *force*

lemma *exists-irreducible-proper-factor*:
assumes *monic-poly R f degree f* > 0 $\neg$*monic-irreducible-poly R f*
shows $\exists$ *g. monic-irreducible-poly R g* $\wedge$ *g* $pdivides_R$ *f* $\wedge$ *degree g* $<$ *degree f*
proof −
define *S* **where** *S* = {*d. monic-irreducible-poly R d* $\wedge$ $0 <$ *pmult d f*}

have *f-carr*: *f* $\in$ *carrier* (*poly-ring R*) $f \neq \mathbf{0}_{poly\text{-}ring\ R}$
using *assms*(*1*) **unfolding** *monic-poly-def univ-poly-zero* **by** *auto*

have $S \neq \{\}$
proof (*rule ccontr*)
assume *S-empty*: $\neg(S \neq \{\})$
have $f = (\bigotimes_{poly\text{-}ring\ R} d{\in}S.\ d\ [\char`^]_{poly\text{-}ring\ R}\ pmult\ d\ f)$
unfolding *S-def* **by** (*intro factor-monic-poly assms*(*1*))
also have ... $= \mathbf{1}_{poly\text{-}ring\ R}$ **using** *S-empty* **by** *simp*
finally have $f = \mathbf{1}_{poly\text{-}ring\ R}$ **by** *simp*
hence *degree f* $= 0$ **using** *degree-one* **by** *simp*

thus *False* **using** *assms*(*2*) **by** *simp*
qed
then obtain *g* **where** *g-irred*: *monic-irreducible-poly R g* **and** *0* < *pmult g f*
unfolding *S-def* **by** *auto*

hence *1* ≤ *pmult g f* **by** *simp*

hence *g-div*: *g pdivides f* **using** *multiplicity-ge-1-iff-pdivides f-carr g-irred* **by** *blast*

then obtain *h* **where** *f-def*: $f = g \otimes_{poly\text{-}ring\ R} h$ **and** *h-carr*:*h* ∈ *carrier* (*poly-ring R*)
unfolding *pdivides-def* **by** *auto*

have *g-nz*: $g \neq \mathbf{0}_{poly\text{-}ring\ R}$ **and** *h-nz*: $h \neq \mathbf{0}_{poly\text{-}ring\ R}$
and *g-carr*: *g* ∈ *carrier* (*poly-ring R*)
using *f-carr*(*2*) *h-carr g-irred* **unfolding** *f-def monic-irreducible-poly-def monic-poly-def*
by *auto*

have *degree f* = *degree g* + *degree h*
using *g-nz h-nz g-carr h-carr* **unfolding** *f-def* **by** (*intro degree-mult*[*OF r.K-subring*]) *auto*
moreover have *degree h* > *0*
proof (*rule ccontr*)
assume ¬(*degree h* > *0*)
hence *degree h* = *0* **by** *simp*
hence *h* ∈ *Units* (*poly-ring R*)
using *h-carr h-nz* **by** (*simp add*: *carrier-is-subfield univ-poly-units′ univ-poly-zero*)
hence $f \sim_{poly\text{-}ring\ R} g$
unfolding *f-def* **using** *g-carr r.p.associatedI2′* **by** *force*
hence $f \sim_{mult\text{-}of\ (poly\text{-}ring\ R)} g$
using *f-carr g-nz g-carr* **by** (*simp add*: *r.p.assoc-iff-assoc-mult*)
hence *f* = *g*
using *monic-poly-not-assoc assms*(*1*) *g-irred* **unfolding** *monic-irreducible-poly-def* **by** *simp*
hence *monic-irreducible-poly R f*
using *g-irred* **by** *simp*
thus *False*
using *assms*(*3*) **by** *auto*
qed
ultimately have *degree g* < *degree f* **by** *simp*
thus *?thesis* **using** *g-irred g-div* **by** *auto*
qed

theorem *rabin-irreducibility-condition*:
assumes *monic-poly R f degree f* > *0*

defines *N* ≡ {*degree f div p* | *p* . *Factorial-Ring.prime p* ∧ *p dvd degree f*}
shows *monic-irreducible-poly R f* ⟷
(*f pdivides gauss-poly R* (*order R^degree f*) ∧ (∀ *n* ∈ *N*. *pcoprime* (*gauss-poly R* (*order R^n*)) *f*))
(**is** *?L* ⟷ *?R1* ∧ *?R2*)
proof −
have *f-carr*: *f* ∈ *carrier* (*poly-ring R*)
using *assms*(*1*) **unfolding** *monic-poly-def* **by** *blast*

have *?R1* **if** *?L*
using *div-gauss-poly-iff*[**where** *n*=*degree f*] *that assms*(*2*) **by** *simp*
moreover have *False* **if** *cthat*:¬*pcoprime* (*gauss-poly R* (*order R^n*)) *f ?L n* ∈ *N* **for** *n*
proof −
obtain *d* **where** *d-def*:
d pdivides f
d pdivides (*gauss-poly R* (*order R^n*)) *degree d* > *0 d* ∈ *carrier* (*poly-ring R*)
using *cthat*(*1*) **unfolding** *pcoprime-def* **by** *auto*

obtain *p* **where** *p-def*:
n = *degree f div p Factorial-Ring.prime p p dvd degree f*
using *cthat*(*3*) **unfolding** *N-def* **by** *auto*

have *n-gt-0*: *n* > *0*
using *p-def assms*(*2*) **by** (*metis dvd-div-eq-0-iff gr0I*)

have *d* ∉ *Units* (*poly-ring R*)
using *d-def*(*3*,*4*) *univ-poly-units'*[*OF carrier-is-subfield*] **by** *simp*
hence *f pdivides d*
using *cthat*(*2*) *d-def*(*1*,*4*) **unfolding** *monic-irreducible-poly-def ring-irreducible-def*
Divisibility.irreducible-def properfactor-def pdivides-def f-carr **by** *auto*
hence *f pdivides* (*gauss-poly R* (*order R^n*))
using *d-def*(*2*,*4*) *f-carr r.p.divides-trans* **unfolding** *pdivides-def* **by** *metis*
hence *degree f dvd n*
using *n-gt-0 div-gauss-poly-iff*[*OF - cthat*(*2*)] **by** *auto*
thus *False*
using *p-def* **by** (*metis assms*(*2*) *div-less-dividend n-gt-0 nat-dvd-not-less prime-gt-1-nat*)
qed
moreover have *False* **if** *not-l*:¬*?L* **and** *r1*:*?R1* **and** *r2*: *?R2*
proof −
obtain *g* **where** *g-def*: *g pdivides f degree g* < *degree f monic-irreducible-poly R g*
using *r1 not-l exists-irreducible-proper-factor assms*(*1*,*2*) **by** *auto*

```
    have g-carr: g ∈ carrier (poly-ring R) and g-nz: g ≠ 0_poly-ring R
     using g-def(3) unfolding monic-irreducible-poly-def monic-poly-def
by (auto simp:univ-poly-zero)

    have g pdivides gauss-poly R (order R^degree f)
     using g-carr r1 g-def(1) unfolding pdivides-def using r.p.divides-trans
by blast

    hence degree g dvd degree f
      using div-gauss-poly-iff[OF assms(2) g-def(3)] by auto

    then obtain t where deg-f-def:degree f = t * degree g
      by fastforce
    hence t > 1 using g-def(2) by simp
    then obtain p where p-prime: Factorial-Ring.prime p p dvd t
      by (metis order-less-irrefl prime-factor-nat)
    hence p-div-deg-f: p dvd degree f
      unfolding deg-f-def by simp
    define n where n = degree f div p
    have n-in-N: n ∈ N
      unfolding N-def n-def using p-prime(1) p-div-deg-f by auto

    have deg-g-dvd-n: degree g dvd n
      using p-prime(2) unfolding n-def deg-f-def by auto

    have n-gt-0: n > 0
      using p-div-deg-f assms(2) p-prime(1) unfolding n-def
      by (metis dvd-div-eq-0-iff gr0I)

    have deg-g-gt-0: degree g > 0
      using monic-poly-min-degree[OF g-def(3)] by simp

    have 0:g pdivides gauss-poly R (order R^n)
     using deg-g-dvd-n div-gauss-poly-iff[OF n-gt-0 g-def(3)] by simp

    have pcoprime (gauss-poly R (order R^n)) f
      using n-in-N r2 by simp
    thus False
       using 0 g-def(1) g-carr deg-g-gt-0 unfolding pcoprime-def by
simp
  qed
  ultimately show ?thesis
    by auto
qed
```

A more general variant of the previous theorem for non-monic polynomials. The result is from Lemma 1 [5].

theorem *rabin-irreducibility-condition-2*:

assumes *f* ∈ *carrier* (*poly-ring R*) *degree f* > *0*
defines *N* ≡ {*degree f div p* | *p* . *Factorial-Ring.prime p* ∧ *p dvd degree f*}
shows *pirreducible* (*carrier R*) *f* ⟷
(*f pdivides gauss-poly R* (*order R^degree f*) ∧ (∀ *n* ∈ *N*. *pcoprime* (*gauss-poly R* (*order R^n*)) *f*))
(**is** *?L* ⟷ *?R1* ∧ *?R2*)
proof −
define α **where** α = [*inv* (*hd f*)]
let *?g* = (λ*x*. *gauss-poly R* (*order R^x*))
let *?h* = α $\otimes_{poly\text{-}ring\ R}$ *f*

have *f-nz*: *f* ≠ $\mathbf{0}_{poly\text{-}ring\ R}$ **unfolding** *univ-poly-zero* **using** *assms*(*2*) **by** *auto*

hence *hd f* ∈ *carrier R* − {**0**} **using** *assms*(*1*) *lead-coeff-carr* **by** *simp*
hence *inv* (*hd f*) ∈ *carrier R* − {**0**} **using** *field-Units* **by** *auto*
hence α*-unit*: α ∈ *Units* (*poly-ring R*)
unfolding α*-def* **using** *univ-poly-carrier-units* **by** *simp*

have α*-nz*: α ≠ $\mathbf{0}_{poly\text{-}ring\ R}$ **unfolding** *univ-poly-zero* α*-def* **by** *simp*
have *hd ?h* = *hd* α ⊗ *hd f*
using α*-nz f-nz assms*(*1*) α*-unit* **by** (*intro lead-coeff-mult*) *auto*
also have ... = *inv* (*hd f*) ⊗ *hd f* **unfolding** α*-def* **by** *simp*
also have ... = **1** **using** *lead-coeff-carr f-nz assms*(*1*) **by** (*simp add*: *field-Units*)
finally have *hd ?h* = **1** **by** *simp*
moreover have *?h* ≠ []
using α*-nz f-nz univ-poly-zero* **by** (*metis* α*-unit assms*(*1*) *r.p.Units-closed r.p.integral*)
ultimately have *h-monic*: *monic-poly R ?h*
using *r.p.Units-closed*[*OF* α*-unit*] *assms*(*1*) **unfolding** *monic-poly-def* **by** *auto*

have *degree ?h* = *degree* α + *degree f*
using *assms*(*1*) *f-nz* α*-unit* α*-nz* **by** (*intro degree-mult*[*OF carrier-is-subring*]) *auto*
also have ... = *degree f* **unfolding** α*-def* **by** *simp*
finally have *deg-f*: *degree f* = *degree ?h* **by** *simp*

have *hf-cong*: *?h* $\sim_{r.P}$ *f*
using *assms*(*1*) α*-unit* **by** (*simp add*: *r.p.Units-closed r.p.associatedI2 r.p.m-comm*)
hence *0*: *f pdivides ?g* (*degree f*) ⟷ *?h pdivides ?g* (*degree f*)
unfolding *pdivides-def* **using** *r.p.divides-cong-l r.p.associated-sym*
using *r.p.Units-closed*[*OF* α*-unit*] *assms*(*1*) *gauss-poly-carr* **by** *blast*

```
  have 1: pcoprime (?g n) f ⟷ pcoprime (?g n) ?h for n
   using hf-cong r.p.associated-sym r.p.Units-closed[OF α-unit] assms(1)
    by (intro pcoprime-right-assoc-cong gauss-poly-carr) auto

  have ?L ⟷ pirreducible (carrier R) (α ⊗poly-ring R f)
   using α-unit α-nz assms(1) f-nz r.p.integral unfolding ring-irreducible-def
    by (intro arg-cong2[where f=(∧)] r.p.irreducible-prod-unit assms)
auto
  also have ... ⟷ monic-irreducible-poly R (α ⊗poly-ring R f)
    using h-monic unfolding monic-irreducible-poly-def by auto
  also have ... ⟷ ?h pdivides ?g (degree f) ∧ (∀ n ∈ N. pcoprime
(?g n) ?h)
   using assms(2) unfolding N-def deg-f by (intro rabin-irreducibility-condition
h-monic) auto
  also have ... ⟷ f pdivides ?g (degree f) ∧ (∀ n ∈ N. pcoprime (?g
n) f)
    using 0 1 by simp
  finally show ?thesis by simp
qed

end

end
```

9 Executable Structures

```
theory Finite-Fields-Indexed-Algebra-Code
  imports HOL−Algebra.Ring HOL−Algebra.Coset
begin
```

In the following, we introduce records for executable operations for algebraic structures, which can be used for code-generation and evaluation. These are then shown to be equivalent to the (not-necessarily constructive) definitions using `HOL-Algebra`. A more direct approach, i.e., instantiating the structures in the framework with effective operations fails. For example the structure records represent the domain of the algebraic structure as a set, which implies the evaluation of $(\oplus_{residue\text{-}ring\ (10::'c)^{100}})$ requires the construction of $\{0..(10::'a)^{100} - 1\}$. This is technically constructive but very impractical. Moreover, the additive/multiplicative inverse is defined non-constructively using the description operator `THE` in `HOL-Algebra`.

The above could be avoided, if it were possible to introduce code equations conditionally, e.g., for example for $(\ominus_{residue\text{-}ring\ n}\ x)\ y$ (if x y are in the carrier of the structure, but this does not seem to be possible.

Note that, the algebraic structures defined in `HOL-Computational_Algebra` are type-based, which prevents using them in some algorithmic settings. For example, choosing an irreducible polynomial dynamically and performing operations in the factoring ring with respect to it is not possible in the type-based approach.

record *'a idx-ring* =
  idx-pred :: *'a* ⇒ *bool*
  idx-uminus :: *'a* ⇒ *'a*
  idx-plus :: *'a* ⇒ *'a* ⇒ *'a*
  idx-udivide :: *'a* ⇒ *'a*
  idx-mult :: *'a* ⇒ *'a* ⇒ *'a*
  idx-zero :: *'a*
  idx-one :: *'a*

record *'a idx-ring-enum* = *'a idx-ring* +
  idx-size :: *nat*
  idx-enum :: *nat* ⇒ *'a*
  idx-enum-inv :: *'a* ⇒ *nat*

fun *idx-pow* :: (*'a*,*'b*) *idx-ring-scheme* ⇒ *'a* ⇒ *nat* ⇒ *'a* **where**
  idx-pow E x 0 = *idx-one E* |
  idx-pow E x (*Suc n*) = *idx-mult E* (*idx-pow E x n*) *x*

open-bundle *index-algebra-syntax*
begin
notation *idx-zero* (‹0_Cı›)
notation *idx-one* (‹1_Cı›)
notation *idx-plus* (**infixl** ‹$+_C$ı› *65*)
notation *idx-mult* (**infixl** ‹$*_C$ı› *70*)
notation *idx-uminus* (‹$-_C$ı -› [*81*] *80*)
notation *idx-udivide* (‹- $^{-1}{}_C$ı› [*81*] *80*)
notation *idx-pow* (**infixr** ‹$\widehat{}_C$ı› *75*)
end

definition *ring-of* :: (*'a*,*'b*) *idx-ring-scheme* ⇒ *'a ring*
  where *ring-of A* = ⦇
    carrier = {*x*. *idx-pred A x*},
    mult = (λ *x y*. $x *_{C\,A}$ *y*),
    one = $1_{C\,A}$,
    zero = $0_{C\,A}$,
    add = (λ *x y*. $x +_{C\,A}$ *y*) ⦈

definition $ring_C$ **where**
  $ring_C$ *A* = (*ring* (*ring-of A*) ∧ (∀ *x*. *idx-pred A x* ⟶ $-_{C\,A}$ *x* = $\ominus_{ring\text{-}of\ A}$ *x*) ∧
    (∀ *x*. *x* ∈ *Units* (*ring-of A*) ⟶ $x\ ^{-1}{}_{C\,A}$ = $inv_{ring\text{-}of\ A}$ *x*))

lemma *ring-cD-aux*:
$x \mathbin{\widehat{}}_{C\,A}\ n = x\ [\widehat{}]_{\textit{ring-of } A}\ n$
by (*induction n*) (*auto simp*:*ring-of-def*)

lemma *ring-cD*:
assumes $\textit{ring}_C\ A$
shows
$0_{C\,A} = \mathbf{0}_{\textit{ring-of } A}$
$1_{C\,A} = \mathbf{1}_{\textit{ring-of } A}$
$\bigwedge x\ y.\ x *_{C\,A}\ y = x \otimes_{\textit{ring-of } A}\ y$
$\bigwedge x\ y.\ x +_{C\,A}\ y = x \oplus_{\textit{ring-of } A}\ y$
$\bigwedge x.\ x \in \textit{carrier}\ (\textit{ring-of } A) \Longrightarrow\ -_{C\,A}\ x = \ominus_{\textit{ring-of } A}\ x$
$\bigwedge x.\ x \in \textit{Units}\ (\textit{ring-of } A) \Longrightarrow\ x\ ^{-1}{}_{C\,A} = \textit{inv}_{\textit{ring-of } A}\ x$
$\bigwedge x.\ x \mathbin{\widehat{}}_{C\,A}\ n = x\ [\widehat{}]_{\textit{ring-of } A}\ n$
using *assms ring-cD-aux* **unfolding** $\textit{ring}_C$*-def ring-of-def* **by** *auto*

lemma *ring-cI*:
assumes *ring* (*ring-of A*)
assumes $\bigwedge x.\ x \in \textit{carrier}\ (\textit{ring-of } A) \Longrightarrow\ -_{C\,A}\ x = \ominus_{\textit{ring-of } A}\ x$
assumes $\bigwedge x.\ x \in \textit{Units}\ (\textit{ring-of } A) \Longrightarrow\ x^{-1}{}_{C\,A} = \textit{inv}_{\textit{ring-of } A}\ x$
shows $\textit{ring}_C\ A$
proof –
have $x \in \textit{carrier}\ (\textit{ring-of } A) \longleftrightarrow \textit{idx-pred}\ A\ x$ **for** *x* **unfolding** *ring-of-def* **by** *auto*
thus *?thesis* **using** *assms* **unfolding** $\textit{ring}_C$*-def* **by** *auto*
qed

definition $\textit{cring}_C$ **where** $\textit{cring}_C\ A = (\textit{ring}_C\ A \wedge \textit{cring}\ (\textit{ring-of } A))$

lemma *cring-cI*:
assumes *cring* (*ring-of A*)
assumes $\bigwedge x.\ x \in \textit{carrier}\ (\textit{ring-of } A) \Longrightarrow\ -_{C\,A}\ x = \ominus_{\textit{ring-of } A}\ x$
assumes $\bigwedge x.\ x \in \textit{Units}\ (\textit{ring-of } A) \Longrightarrow x^{-1}{}_{C\,A} = \textit{inv}_{\textit{ring-of } A}\ x$
shows $\textit{cring}_C\ A$
unfolding $\textit{cring}_C$*-def* **by** (*intro ring-cI conjI assms cring.axioms*(*1*))

lemma *cring-c-imp-ring*: $\textit{cring}_C\ A \Longrightarrow \textit{ring}_C\ A$
unfolding $\textit{cring}_C$*-def* **by** *simp*

lemmas *cring-cD* = *ring-cD*[*OF cring-c-imp-ring*]

definition $\textit{domain}_C$ **where** $\textit{domain}_C\ A = (\textit{cring}_C\ A \wedge \textit{domain}\ (\textit{ring-of } A))$

lemma *domain-cI*:
assumes *domain* (*ring-of A*)
assumes $\bigwedge x.\ x \in \textit{carrier}\ (\textit{ring-of } A) \Longrightarrow\ -_{C\,A}\ x = \ominus_{\textit{ring-of } A}\ x$
assumes $\bigwedge x.\ x \in \textit{Units}\ (\textit{ring-of } A) \Longrightarrow x^{-1}{}_{C\,A} = \textit{inv}_{\textit{ring-of } A}\ x$

shows *domain*$_C$ *A*
unfolding *domain*$_C$*-def* **by** (*intro conjI cring-cI assms domain.axioms*(*1*))

lemma *domain-c-imp-ring*: *domain*$_C$ *A* $\Longrightarrow$ *ring*$_C$ *A*
unfolding *cring*$_C$*-def domain*$_C$*-def* **by** *simp*

lemmas *domain-cD* = *ring-cD*[*OF domain-c-imp-ring*]

definition *field*$_C$ **where** *field*$_C$ *A* = (*domain*$_C$ *A* $\wedge$ *field* (*ring-of A*))

lemma *field-cI*:
assumes *field* (*ring-of A*)
assumes $\bigwedge x.\ x \in$ *carrier* (*ring-of A*) $\Longrightarrow$ $-_{C\,A}\ x = \ominus_{\textit{ring-of}\ A}\ x$
assumes $\bigwedge x.\ x \in$ *Units* (*ring-of A*) $\Longrightarrow$ $x^{-1}{}_{C\,A} = \textit{inv}_{\textit{ring-of}\ A}\ x$
shows *field*$_C$ *A*
unfolding *field*$_C$*-def* **by** (*intro conjI domain-cI assms field.axioms*(*1*))

lemma *field-c-imp-ring*: *field*$_C$ *A* $\Longrightarrow$ *ring*$_C$ *A*
unfolding *field*$_C$*-def cring*$_C$*-def domain*$_C$*-def* **by** *simp*

lemmas *field-cD* = *ring-cD*[*OF field-c-imp-ring*]

definition *enum*$_C$ **where** *enum*$_C$ *A* = (
finite (*carrier* (*ring-of A*)) $\wedge$
idx-size A = *order* (*ring-of A*) $\wedge$
bij-betw (*idx-enum A*) {..<*order* (*ring-of A*)} (*carrier* (*ring-of A*)) $\wedge$
($\forall$ *x* < *order* (*ring-of A*). *idx-enum-inv A* (*idx-enum A x*) = *x*))

lemma *enum-cI*:
assumes *finite* (*carrier* (*ring-of A*))
assumes *idx-size A* = *order* (*ring-of A*)
assumes *bij-betw* (*idx-enum A*) {..<*order* (*ring-of A*)} (*carrier* (*ring-of A*))
assumes $\bigwedge x.\ x <$ *order* (*ring-of A*) $\Longrightarrow$ *idx-enum-inv A* (*idx-enum A x*) = *x*
shows *enum*$_C$ *A*
using *assms* **unfolding** *enum*$_C$*-def* **by** *auto*

lemma *enum-cD*:
assumes *enum*$_C$ *R*
shows *finite* (*carrier* (*ring-of R*))
and *idx-size R* = *order* (*ring-of R*)
and *bij-betw* (*idx-enum R*) {..<*order* (*ring-of R*)} (*carrier* (*ring-of R*))
and *bij-betw* (*idx-enum-inv R*) (*carrier* (*ring-of R*)) {..<*order* (*ring-of R*)}
and $\bigwedge x.\ x <$ *order* (*ring-of R*) $\Longrightarrow$ *idx-enum-inv R* (*idx-enum R x*) = *x*
and $\bigwedge x.\ x \in$ *carrier* (*ring-of R*) $\Longrightarrow$ *idx-enum R* (*idx-enum-inv R*

x) = x
  using *assms*
proof −
  let *?n* = *order* (*ring-of R*)
  have *a*:*idx-enum-inv R x* = *the-inv-into* {..<*?n*} (*idx-enum R*) *x*
    if *x-carr*: *x* ∈ *carrier* (*ring-of R*) **for** *x*
  proof −
    have *idx-enum R* ' {..<*order* (*ring-of R*)} = *carrier* (*ring-of R*)
      using *assms* **unfolding** *bij-betw-def* $enum_C$*-def* **by** *simp*
     then obtain *y* **where** *y-carr*: *y* ∈ {..< *order* (*ring-of R*)} **and** *x-def*: *x* = *idx-enum R y*
      using *x-carr* **by** *auto*
    have *idx-enum-inv R x* = *y* **using** *assms y-carr* **unfolding** *x-def* $enum_C$*-def* **by** *simp*
    also have ... = *the-inv-into* {..<*?n*} (*idx-enum R*) *x*
      using *assms* **unfolding** *bij-betw-def* $enum_C$*-def* **unfolding** *x-def*
      by (*intro the-inv-into-f-f*[*symmetric*] *y-carr*) *auto*
    finally show *?thesis* **by** *simp*
  qed

  have *bij-betw* (*the-inv-into* {..<*?n*} (*idx-enum R*)) (*carrier* (*ring-of R*)) {..<*?n*}
    using *assms* **unfolding** $enum_C$*-def* **by** (*intro bij-betw-the-inv-into*) *auto*
  thus *bij-betw* (*idx-enum-inv R*) (*carrier* (*ring-of R*)) {..<*order* (*ring-of R*)}
    by (*subst bij-betw-cong*[*OF a*]) *auto*
  show *idx-enum R* (*idx-enum-inv R x*) = *x* **if** *x* ∈ *carrier* (*ring-of R*) **for** *x*
    using *that assms* **unfolding** *a*[*OF that*] $enum_C$*-def bij-betw-def* **by** (*intro f-the-inv-into-f*) *auto*
qed (*use assms* $enum_C$*-def* **in** *auto*)

end

10 Executable Polynomial Rings

theory *Finite-Fields-Poly-Ring-Code*
  imports
    Finite-Fields-Indexed-Algebra-Code
    HOL−Algebra.Polynomials
    Finite-Fields.Card-Irreducible-Polynomials-Aux
begin

fun *o-normalize* :: ('a,'b) *idx-ring-scheme* ⇒ 'a *list* ⇒ 'a *list*
  where
    o-normalize E [] = []
  | *o-normalize E p* = (*if lead-coeff p* ≠ $0_{C\,E}$ *then p else o-normalize E* (*tl p*))

```
fun o-poly-add :: ('a,'b) idx-ring-scheme ⇒ 'a list ⇒ 'a list ⇒ 'a list
where
  o-poly-add E p1 p2 = (
    if length p1 ≥ length p2
      then o-normalize E (map2 (idx-plus E) p1 ((replicate (length p1
− length p2) 0C E ) @ p2))
      else o-poly-add E p2 p1)

fun o-poly-mult :: ('a,'b) idx-ring-scheme ⇒ 'a list ⇒ 'a list ⇒ 'a list
  where
    o-poly-mult E [] p2 = []
  | o-poly-mult E p1 p2 =
       o-poly-add E ((map (idx-mult E (hd p1)) p2) @
      (replicate (degree p1) 0C E )) (o-poly-mult E (tl p1) p2)

definition poly :: ('a,'b) idx-ring-scheme ⇒ 'a list idx-ring
  where poly E = (|
    idx-pred = (λx. (x = [] ∨ hd x ≠ 0C E) ∧ list-all (idx-pred E) x),
    idx-uminus = (λx. map (idx-uminus E) x),
    idx-plus = o-poly-add E,
    idx-udivide = (λx. [idx-udivide E (hd x)]),
    idx-mult = o-poly-mult E,
    idx-zero = [],
    idx-one = [idx-one E] |)

definition poly-var :: ('a,'b) idx-ring-scheme ⇒ 'a list (‹XC1›)
  where poly-var E = [idx-one E, idx-zero E]

lemma poly-var: poly-var R = X ring-of R
  unfolding var-def poly-var-def by (simp add:ring-of-def)

fun poly-eval :: ('a,'b) idx-ring-scheme ⇒ 'a list ⇒ 'a ⇒ 'a
  where poly-eval R fs x = fold (λa b. b *C R x +C R a) fs 0C R



lemma ring-of-poly:
  assumes ringC A
  shows ring-of (poly A) = poly-ring (ring-of A)
proof (intro ring.equality)
 interpret ring ring-of A using assms unfolding ringC-def by auto

  have b: 0 ring-of A = 0C A unfolding ring-of-def by simp
  have c: (⊗ ring-of A) = (*C A) unfolding ring-of-def by simp
  have d: (⊕ ring-of A) = (+C A) unfolding ring-of-def by simp

  have  o-normalize A  x = normalize x for x
    using b by (induction x) simp-all
```

hence *o-poly-add A x y* = *poly-add x y* **if** *length y* ≤ *length x* **for** *x y*
  using *that* **by** (*subst o-poly-add.simps*, *subst poly-add.simps*) (*simp add*: *b d*)
 hence *a*:*o-poly-add A x y* = *poly-add x y* **for** *x y*
  by (*subst o-poly-add.simps*, *subst poly-add.simps*) *simp*

 hence $x \oplus_{ring\text{-}of\ (poly\ A)}\ y = x \oplus_{poly\text{-}ring\ (ring\text{-}of\ A)}\ y$ **for** *x y*
  by (*simp add*:*univ-poly-def poly-def ring-of-def*)

 thus $(\oplus_{ring\text{-}of\ (poly\ A)}) = (\oplus_{poly\text{-}ring\ (ring\text{-}of\ A)})$ **by** (*intro ext*)

 show *carrier* (*ring-of* (*poly A*)) = *carrier* (*poly-ring* (*ring-of A*))
  by (*auto simp add*: *ring-of-def poly-def univ-poly-def polynomial-def list-all-iff*)

 have *o-poly-mult A x y* = *poly-mult x y* **for** *x y*
 proof (*induction x*)
  case *Nil* **then show** *?case* **by** *simp*
 next
  case (*Cons a x*) **then show** *?case*
   by (*subst o-poly-mult.simps*,*subst poly-mult.simps*)
    (*simp add*:*a b c del*:*poly-add.simps o-poly-add.simps*)
 qed
 hence $x \otimes_{ring\text{-}of\ (poly\ A)}\ y = x \otimes_{poly\text{-}ring\ (ring\text{-}of\ A)}\ y$ **for** *x y*
  by (*simp add*: *univ-poly-def poly-def ring-of-def*)
 thus $(\otimes_{ring\text{-}of\ (poly\ A)}) = (\otimes_{poly\text{-}ring\ (ring\text{-}of\ A)})$ **by** (*intro ext*)

qed (*simp-all add*:*ring-of-def poly-def univ-poly-def*)

lemma *poly-eval*:
 assumes $ring_C$ *R*
 assumes *fsc*:*fs* ∈ *carrier* (*ring-of* (*poly R*)) **and** *xc*:*x* ∈ *carrier* (*ring-of R*)
 shows *poly-eval R fs x* = *ring.eval* (*ring-of R*) *fs x*
proof −
 interpret *ring ring-of R* **using** *assms* **unfolding** $ring_C$*-def* **by** *auto*

 have *fs-carr*:*fs* ∈ *carrier* (*poly-ring* (*ring-of R*)) **using** *ring-of-poly*[*OF assms*(*1*)] *fsc* **by** *auto*
 hence *set fs* ⊆ *carrier* (*ring-of R*) **by** (*simp add*: *polynomial-incl univ-poly-carrier*)
 thus *?thesis*
 proof (*induction rule*:*rev-induct*)
  case *Nil* **thus** *?case* **by** *simp* (*simp add*:*ring-of-def*)
 next
  case (*snoc ft fh*)
  have *poly-eval R* (*fh* @ [*ft*]) *x* = *poly-eval R fh x* $*_{C\,R}$ *x* $+_{C\,R}$ *ft*

by *simp*
also have ... = *eval fh* $x *_{C\,R}$ $x +_{C\,R}$ *ft* **using** *snoc* **by** (*subst snoc*) *auto*
also have ... = *eval fh* $x \otimes_{ring\text{-}of\ R}$ $x \oplus_{ring\text{-}of\ R}$ *ft* **by** (*simp add*:*ring-of-def*)
also have ... = *eval* (*fh*@[*ft*]) *x* **using** *snoc* **by** (*intro eval-append-aux*[*symmetric*] *xc*) *auto*
finally show *?case* **by** *auto*
qed
qed

lemma *poly-domain*:
assumes $domain_C$ *A*
shows $domain_C$ (*poly A*)
proof −
interpret *domain ring-of A* **using** *assms* **unfolding** $domain_C$*-def* **by** *auto*

have *a*:$\ominus_{ring\text{-}of\ A}$ $x = -_{C\,A}$ *x* **if** $x \in$ *carrier* (*ring-of A*) **for** *x*
using *that* **by** (*intro domain-cD*[*symmetric*] *assms*)
have $ring_C$ *A*
using *assms* **unfolding** $domain_C$*-def* $cring_C$*-def* **by** *auto*
hence *b*:*ring-of* (*poly A*) = *poly-ring* (*ring-of A*)
by (*subst ring-of-poly*) *auto*

have *c*:*domain* (*ring-of* (*poly A*))
unfolding *b* **by** (*rule univ-poly-is-domain*[*OF carrier-is-subring*])

interpret *d*: *domain poly-ring* (*ring-of A*)
using *c* **unfolding** *b* **by** *simp*

have $-_{C\,poly\ A}$ $x = \ominus_{ring\text{-}of\ (poly\ A)}$ *x* **if** $x \in$ *carrier* (*ring-of* (*poly A*)) **for** *x*
proof −
have $\ominus_{ring\text{-}of\ (poly\ A)}$ *x* = *map* (*a-inv* (*ring-of A*)) *x*
using *that* **unfolding** *b* **by** (*subst univ-poly-a-inv-def'*[*OF carrier-is-subring*]) *auto*
also have ... = *map* ($\lambda r.\ -_{C\,A}$ *r*) *x*
using *that* **unfolding** *b univ-poly-carrier*[*symmetric*] *polynomial-def*
by (*intro map-cong refl a*) *auto*
also have ... = $-_{C\,poly\ A}$ *x*
unfolding *poly-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed
moreover have $x\ ^{-1}{}_{C\,poly\ A}$ = $inv_{ring\text{-}of\ (poly\ A)}$ *x* **if** $x \in$ *Units* (*ring-of* (*poly A*)) **for** *x*
proof −
have $x \in \{[k]\ |k.\ k \in$ *carrier* (*ring-of A*) − $\{\mathbf{0}_{ring\text{-}of\ A}\}\}$

using *that univ-poly-carrier-units-incl* **unfolding** *b* **by** *auto*
then obtain *k* **where** *x-eq*: $k \in carrier\ (ring\text{-}of\ A) - \{\mathbf{0}_{ring\text{-}of\ A}\}$ $x = [k]$ **by** *auto*
have $inv_{ring\text{-}of\ (poly\ A)}\ x \in Units\ (poly\text{-}ring\ (ring\text{-}of\ A))$
using *that* **unfolding** *b* **by** *simp*
hence $inv_{ring\text{-}of\ (poly\ A)}\ x \in \{[k] \mid k.\ k \in carrier\ (ring\text{-}of\ A) - \{\mathbf{0}_{ring\text{-}of\ A}\}\}$
using *that univ-poly-carrier-units-incl* **unfolding** *b* **by** *auto*
then obtain *v* **where** *x-inv-eq*: $v\in carrier\ (ring\text{-}of\ A) - \{\mathbf{0}_{ring\text{-}of\ A}\}$ $inv_{ring\text{-}of\ (poly\ A)}\ x = [v]$ **by** *auto*

have $poly\text{-}mult\ [k]\ [v] = [k] \otimes_{ring\text{-}of\ (poly\ A)} [v]$ **unfolding** *b univ-poly-mult* **by** *simp*
also have ... $= x \otimes_{ring\text{-}of\ (poly\ A)} inv_{ring\text{-}of\ (poly\ A)}\ x$ **using** *x-inv-eq x-eq* **by** *auto*
also have ... $= \mathbf{1}_{ring\text{-}of\ (poly\ A)}$ **using** *that* **unfolding** *b* **by** *simp*
also have ... $= [\mathbf{1}_{ring\text{-}of\ A}]$ **unfolding** *b univ-poly-one* **by** (*simp add*:*ring-of-def*)
finally have $poly\text{-}mult\ [k]\ [v] = [\mathbf{1}_{ring\text{-}of\ A}]$ **by** *simp*
hence $k \otimes_{ring\text{-}of\ A} v \oplus_{ring\text{-}of\ A} \mathbf{0}_{ring\text{-}of\ A} = \mathbf{1}_{ring\text{-}of\ A}$
by (*simp add*:*if-distribR if-distrib*) (*simp cong*:*if-cong*, *metis*)
hence *e*: $k \otimes_{ring\text{-}of\ A} v = \mathbf{1}_{ring\text{-}of\ A}$ **using** *x-eq*(*1*) *x-inv-eq*(*1*) **by** *simp*
hence *f*: $v \otimes_{ring\text{-}of\ A} k = \mathbf{1}_{ring\text{-}of\ A}$ **using** *x-eq*(*1*) *x-inv-eq*(*1*) *m-comm* **by** *simp*
have *g*: $v = inv_{ring\text{-}of\ A}\ k$
using *e x-eq*(*1*) *x-inv-eq*(*1*) **by** (*intro comm-inv-char*[*symmetric*]) *auto*
hence *h*: $k \in Units\ (ring\text{-}of\ A)$ **unfolding** *Units-def* **using** *e f x-eq*(*1*) *x-inv-eq*(*1*) **by** *blast*

have $x\ ^{-1}{}_{C\,poly\ A} = [k]\ ^{-1}{}_{C\,poly\ A}$ **unfolding** *x-eq* **by** *simp*
also have ... $= [k\ ^{-1}{}_{C\,A}]$ **unfolding** *poly-def* **by** *simp*
also have ... $= [v]$
unfolding *g* **by** (*intro domain-cD*[*OF assms*(*1*)] *arg-cong2*[**where** *f*=(#)] *h refl*)
also have ... $= inv_{ring\text{-}of\ (poly\ A)}\ x$ **unfolding** *x-inv-eq* **by** *simp*
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis* **using** *c* **by** (*intro domain-cI*)
qed

function $long\text{-}division_C$ :: ('*a*,'*b*) *idx-ring-scheme* ⇒ '*a list* ⇒ '*a list* ⇒ '*a list* × '*a list*
where $long\text{-}division_C\ F\ f\ g$ = (
if (*length g* = *0* ∨ *length f* < *length g*)
then ([], *f*)
else (

let $k = $ *length f* $-$ *length g*;
  $\alpha = -_{C\,F}$ (*hd f* $*_{C\,F}$ (*hd g*) $^{-1}{}_{C\,F}$);
  $h = [\alpha]\ *_{C\,poly\ F}\ X_{C\,F}\ \widehat{}_{C\,poly\ F}\ k$;
  $f' = f +_{C\,poly\ F} (h *_{C\,poly\ F}\ g)$;
  $f'' =$ *take* (*length f* $-$ *1*) f'
in apfst ($\lambda x.\ x +_{C\,poly\ F} -_{C\,poly\ F}\ h$) (*long-division*$_C$ $F\ f''\ g$)))
by *pat-completeness auto*

lemma *pmod-termination-helper*:
$g \neq [] \Longrightarrow \neg$*length f* $<$ *length g* $\Longrightarrow$ *min x* (*length f* $-$ *1*) $<$ *length f*
by (*metis diff-less length-greater-0-conv list.size*(*3*) *min.strict-coboundedI2*
zero-less-one)

termination by (*relation measure* (λ(-, *f*, -). *length f*)) (*use pmod-termination-helper*
in *auto*)

declare *long-division*$_C$*.simps*[*simp del*]

lemma *long-division-c-length*:
assumes *length g* $>$ *0*
shows *length* (*snd* (*long-division*$_C$ *R f g*)) $<$ *length g*
proof (*induction length f arbitrary*:*f rule*:*nat-less-induct*)
case *1*
have *0*:*length* (*snd* (*long-division*$_C$ *R x g*)) $<$ *length g*
if *length x* $<$ *length f* **for** *x* **using** *1 that* **by** *blast*

show *length* (*snd* (*long-division*$_C$ *R f g*)) $<$ *length g*
proof (*cases length f* $<$ *length g*)
case *True* **then show** *?thesis* **by** (*subst long-division*$_C$*.simps*) *simp*
next
case *False*
hence *length f* $>$ *0* **using** *assms* **by** *auto*
thus *?thesis* **using** *assms* **by** (*subst long-division*$_C$*.simps*)
(*auto intro*!:*0 simp*: *min.commute min.strict-coboundedI1 Let-def*)
qed
qed

context *field*
begin

interpretation *r*:*polynomial-ring R* (*carrier R*)
unfolding *polynomial-ring-def polynomial-ring-axioms-def*
using *carrier-is-subfield field-axioms* **by** *force*

lemma *poly-length-from-coeff*:
assumes $p \in$ *carrier* (*poly-ring R*)
assumes $\bigwedge i.\ i \geq k \Longrightarrow$ *coeff p i* $=$ **0**
shows *length p* $\leq k$

proof (*rule ccontr*)
assume *a*:¬*length p* ≤ *k*
hence *p-nz*: *p* ≠ [] **by** *auto*
have *k* < *length p* **using** *a* **by** *simp*
hence *k* ≤ *length p* − *1* **by** *simp*
hence **0** = *coeff p* (*degree p*) **by** (*intro assms*(*2*)[*symmetric*])
also have ... = *lead-coeff p* **by** (*intro lead-coeff-simp*[*OF p-nz*])
finally have **0** = *lead-coeff p* **by** *simp*
thus *False*
using *p-nz assms*(*1*) **unfolding** *univ-poly-def polynomial-def* **by** *simp*
qed

lemma *poly-add-cancel-len*:
assumes *f* ∈ *carrier* (*poly-ring R*) − {$\mathbf{0}_{poly\text{-}ring\ R}$}
assumes *g* ∈ *carrier* (*poly-ring R*) − {$\mathbf{0}_{poly\text{-}ring\ R}$}
assumes *hd f* = ⊖ *hd g degree f* = *degree g*
shows *length* (*f* $\oplus_{poly\text{-}ring\ R}$ *g*) < *length f*
proof −
have *f-ne*: *f* ≠ [] **using** *assms*(*1*) **unfolding** *univ-poly-zero* **by** *simp*
have *g-ne*: *g* ≠ [] **using** *assms*(*2*) **unfolding** *univ-poly-zero* **by** *simp*

have *coeff f i* = ⊖*coeff g i* **if** *i* ≥ *degree f* **for** *i*
proof (*cases i* = *degree f*)
case *True*
have *coeff f i* = *hd f* **unfolding** *True* **by** (*subst lead-coeff-simp*[*OF f-ne*]) *simp*
also have ... = ⊖*hd g* **using** *assms*(*3*) **by** *simp*
also have ... = ⊖*coeff g i* **unfolding** *True assms*(*4*) **by** (*subst lead-coeff-simp*[*OF g-ne*]) *simp*
finally show *?thesis* **by** *simp*
next
case *False*
hence *i* > *degree f i* > *degree g* **using** *assms*(*4*) *that* **by** *auto*
thus *coeff f i* = ⊖ *coeff g i* **using** *coeff-degree* **by** *simp*
qed
hence *coeff* (*f* $\oplus_{poly\text{-}ring\ R}$ *g*) *i* = **0** **if** *i* ≥ *degree f* **for** *i*
using *assms*(*1*,*2*) *that* **by** (*subst r.coeff-add*) (*auto intro*:*l-neg simp*: *r.coeff-range*)

hence *length* (*f* $\oplus_{poly\text{-}ring\ R}$ *g*) ≤ *length f* − *1*
using *assms*(*1*,*2*) **by** (*intro poly-length-from-coeff*) *auto*
also have ... < *length f* **using** *f-ne* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *pmod-mult-left*:
assumes *f* ∈ *carrier* (*poly-ring R*)
assumes *g* ∈ *carrier* (*poly-ring R*)

assumes $h \in$ *carrier* (*poly-ring R*)
shows ($f \otimes_{poly\text{-}ring\ R}\ g$) *pmod h* = ((*f pmod h*) $\otimes_{poly\text{-}ring\ R}\ g$) *pmod h* (**is** *?L* = *?R*)
proof −
have *h pdivides* ($h \otimes_{poly\text{-}ring\ R}$ (*f pdiv h*)) $\otimes_{poly\text{-}ring\ R}\ g$
using *assms long-division-closed*[*OF carrier-is-subfield*]
by (*simp add*: *dividesI′ pdivides-def r.p.m-assoc*)
hence *0*:($h \otimes_{poly\text{-}ring\ R}$ (*f pdiv h*)) $\otimes_{poly\text{-}ring\ R}\ g$ *pmod h* = $\mathbf{0}_{poly\text{-}ring\ R}$
using *pmod-zero-iff-pdivides*[*OF carrier-is-subfield*] *assms*
long-division-closed[*OF carrier-is-subfield*] *univ-poly-zero*
by (*metis* (*no-types*, *opaque-lifting*) *r.p.m-closed*)

have *?L* = ($h \otimes_{poly\text{-}ring\ R}$ (*f pdiv h*) $\oplus_{poly\text{-}ring\ R}$ (*f pmod h*)) $\otimes_{poly\text{-}ring\ R}\ g$ *pmod h*
using *assms* **by** (*intro arg-cong2*[**where** *f*=($\otimes_{poly\text{-}ring\ R}$)] *arg-cong2*[**where** *f*=(*pmod*)]
pdiv-pmod[*OF carrier-is-subfield*]) *auto*
also have ... = (($h \otimes_{poly\text{-}ring\ R}$ (*f pdiv h*)) $\otimes_{poly\text{-}ring\ R}\ g \oplus_{poly\text{-}ring\ R}$
(*f pmod h*) $\otimes_{poly\text{-}ring\ R}\ g$) *pmod h*
using *assms long-division-closed*[*OF carrier-is-subfield*]
by (*intro r.p.l-distr arg-cong2*[**where** *f*=(*pmod*)]) *auto*
also have ... = (($h \otimes_{poly\text{-}ring\ R}$ (*f pdiv h*)) $\otimes_{poly\text{-}ring\ R}\ g$) *pmod h* $\oplus_{poly\text{-}ring\ R}$
((*f pmod h*) $\otimes_{poly\text{-}ring\ R}\ g$ *pmod h*)
using *assms long-division-closed*[*OF carrier-is-subfield*]
by (*intro long-division-add*[*OF carrier-is-subfield*]) *auto*
also have ... = *?R*
using *assms long-division-closed*[*OF carrier-is-subfield*] **unfolding** *0* **by** *auto*
finally show *?thesis*
by *simp*
qed

lemma *pmod-mult-right*:
assumes $f \in$ *carrier* (*poly-ring R*)
assumes $g \in$ *carrier* (*poly-ring R*)
assumes $h \in$ *carrier* (*poly-ring R*)
shows ($f \otimes_{poly\text{-}ring\ R}\ g$) *pmod h* = ($f \otimes_{poly\text{-}ring\ R}$ (*g pmod h*)) *pmod h* (**is** *?L* = *?R*)
proof −
have *?L* = ($g \otimes_{poly\text{-}ring\ R}\ f$) *pmod h* **using** *assms* **by** *algebra*
also have ... = ((*g pmod h*) $\otimes_{poly\text{-}ring\ R}\ f$) *pmod h* **by** (*intro pmod-mult-left assms*)
also have ... = *?R* **using** *assms long-division-closed*[*OF carrier-is-subfield*] **by** *algebra*
finally show *?thesis* **by** *simp*
qed

lemma *pmod-mult-both*:
assumes $f \in$ *carrier* (*poly-ring R*)
assumes $g \in$ *carrier* (*poly-ring R*)
assumes $h \in$ *carrier* (*poly-ring R*)
shows $(f \otimes_{poly\text{-}ring\ R} g)$ *pmod* $h = ((f$ *pmod* $h) \otimes_{poly\text{-}ring\ R} (g$ *pmod* $h))$ *pmod* h
(**is** *?L* = *?R*)
proof −
have $(f \otimes_{poly\text{-}ring\ R} g)$ *pmod* $h = ((f$ *pmod* $h) \otimes_{poly\text{-}ring\ R} g)$ *pmod* h
by (*intro pmod-mult-left assms*)
also have ... = *?R*
using *assms long-division-closed*[*OF carrier-is-subfield*] **by** (*intro pmod-mult-right*) *auto*
finally show *?thesis* **by** *simp*
qed

lemma *field-Unit-minus-closed*:
assumes $x \in$ *Units R*
shows $\ominus\ x \in$ *Units R*
using *assms mult-of.Units-eq* **by** *auto*

end

lemma *long-division-c*:
assumes $field_C$ *R*
assumes $f \in$ *carrier* (*poly-ring* (*ring-of R*))
assumes $g \in$ *carrier* (*poly-ring* (*ring-of R*))
shows $long\text{-}division_C$ *R f g* = (*ring.pdiv* (*ring-of R*) *f g*, *ring.pmod* (*ring-of R*) *f g*)
proof −
let *?P* = *poly-ring* (*ring-of R*)
let *?result* = $(\lambda f\, r.\ f = snd\ r \oplus_{poly\text{-}ring\ (ring\text{-}of\ R)} (fst\ r \otimes_{poly\text{-}ring\ (ring\text{-}of\ R)} g))$

define *r* **where** *r* = $long\text{-}division_C$ *R f g*

interpret *field ring-of R* **using** *assms*(*1*) **unfolding** $field_C$*-def* **by** *auto*
interpret *d-poly-ring*: *domain poly-ring* (*ring-of R*)
by (*rule univ-poly-is-domain*[*OF carrier-is-subring*])

have *ring-c*: $ring_C$ *R* **using** *assms*(*1*) **unfolding** $field_C$*-def* $domain_C$*-def* $cring_C$*-def* **by** *auto*
have *d-poly*: $domain_C$ (*poly R*) **using** *assms* (*1*) **unfolding** $field_C$*-def* **by** (*intro poly-domain*) *auto*

have *r* = $long\text{-}division_C$ *R f g* $\Longrightarrow$ *?result f r* $\wedge$ {*fst r*, *snd r*} $\subseteq$ *carrier* (*poly-ring* (*ring-of R*))

using *assms*(*2*)
proof (*induction length f arbitrary*: *f r rule*:*nat-less-induct*)
case *1*

have *ind*: $x = snd\ q \oplus_{?P} fst\ q \otimes_{?P} g$ $\{fst\ q,\ snd\ q\} \subseteq carrier$ (*poly-ring* (*ring-of R*))
if $length\ x < length\ f$ $q = long\text{-}division_C\ R\ x\ g$ $x \in carrier$ (*poly-ring* (*ring-of R*))
for *x q* **using** *1*(*1*) *that* **by** *auto*

show *?case*
proof (*cases* $length\ g = 0 \lor length\ f < length\ g$)
case *True*
hence $r = (\mathbf{0}_{poly\text{-}ring\ (ring\text{-}of\ R)}, f)$
unfolding *1*(*2*) *univ-poly-zero* **by** (*subst long-division*$_C$*.simps*) *simp*
then show *?thesis* **using** *assms*(*3*) *1*(*3*) **by** *simp*
next
case *False*
hence $length\ g > 0$ $length\ f \geq length\ g$ **by** *auto*
hence $f \neq []$ $g \neq []$ **by** *auto*
hence *f-carr*: $f \in carrier\ ?P - \{\mathbf{0}_{?P}\}$ **and** *g-carr*: $g \in carrier\ ?P - \{\mathbf{0}_{?P}\}$
using *1*(*3*) *assms*(*3*) *univ-poly-zero* **by** *auto*

define *k* **where** $k = length\ f - length\ g$
define α **where** $\alpha = -_{C\,R}\ (hd\ f *_{C\,R}\ (hd\ g)\ {}^{-1}{}_{C\,R})$
define *h* **where** $h = [\alpha] *_{C\,poly\ R} X_{C\,R}\ \widehat{}_{C\,poly\ R}\ k$
define f' **where** $f' = f +_{C\,poly\ R} (h *_{C\,poly\ R} g)$
define f'' **where** $f'' = take\ (length\ f - 1)\ f'$
obtain *s t* **where** *st-def*: $(s,t) = long\text{-}division_C\ R\ f''\ g$ **by** (*metis surj-pair*)

have $r = apfst\ (\lambda x.\ x +_{C\,poly\ R} -_{C\,poly\ R}\ h)\ (long\text{-}division_C\ R\ f''\ g)$
using *False* **unfolding** *1*(*2*)
by (*subst long-division*$_C$*.simps*) (*simp add*:*Let-def f''-def f'-def h-def α-def k-def*)

hence *r-def*: $r = (s +_{C\,poly\ R} -_{C\,poly\ R}\ h,\ t)$
unfolding *st-def*[*symmetric*] **by** *simp*

have *monic-poly* (*ring-of R*) $(X_{ring\text{-}of\ R}\ [\widehat{}]_{poly\text{-}ring\ (ring\text{-}of\ R)}\ k)$
by (*intro monic-poly-pow monic-poly-var*)
hence [*simp*]: *lead-coeff* $(X_{ring\text{-}of\ R}\ [\widehat{}]_{poly\text{-}ring\ (ring\text{-}of\ R)}\ k) = \mathbf{1}_{ring\text{-}of\ R}$
unfolding *monic-poly-def* **by** *simp*

have *hd-f-unit*: *hd f* ∈ *Units* (*ring-of R*) **and** *hd-g-unit*: *hd g* ∈ *Units* (*ring-of R*)
using *f-carr g-carr lead-coeff-carr field-Units* **by** *auto*
hence *hd-f-carr*: *hd f* ∈ *carrier* (*ring-of R*) **and** *hd-g-carr*: *hd g* ∈ *carrier* (*ring-of R*)
by *auto*

have *k-def'*: $k = degree\ f - degree\ g$ **using** *False* **unfolding** *k-def* **by** *auto*
have *α-def'*: $\alpha = \ominus_{ring\text{-}of\ R} (hd\ f \otimes_{ring\text{-}of\ R} inv_{ring\text{-}of\ R}\ hd\ g)$
unfolding *α-def* **using** *hd-g-unit hd-f-carr field-cD*[*OF assms*(*1*)] **by** *simp*

have *α-unit*: α ∈ *Units* (*ring-of R*) **unfolding** *α-def'* **using** *hd-f-unit hd-g-unit*
by (*intro field-Unit-minus-closed*) *simp*
hence *α-carr*: $\alpha \in carrier\ (ring\text{-}of\ R) - \{\mathbf{0}_{ring\text{-}of\ R}\}$ **unfolding** *field-Units* **by** *simp*
hence *α-poly-carr*: $[\alpha] \in carrier\ (poly\text{-}ring\ (ring\text{-}of\ R)) - \{\mathbf{0}_{poly\text{-}ring\ (ring\text{-}of\ R)}\}$
by (*simp add*: *univ-poly-carrier*[*symmetric*] *univ-poly-zero polynomial-def*)

have *h-def'*: $h = [\alpha] \otimes_{?P} X_{ring\text{-}of\ R} [\urcorner]_{?P}\ k$
unfolding *h-def poly-var domain-cD*[*OF d-poly*] **by** (*simp add*:*ring-of-poly*[*OF ring-c*])
have *f'-def'*: $f' = f \oplus_{?P} (h \otimes_{?P} g)$
unfolding *f'-def domain-cD*[*OF d-poly*] **by** (*simp add*:*ring-of-poly*[*OF ring-c*])

have *h-carr*: $h \in carrier\ (poly\text{-}ring\ (ring\text{-}of\ R)) - \{\mathbf{0}_{poly\text{-}ring\ (ring\text{-}of\ R)}\}$
using *d-poly-ring.mult-of.m-closed α-poly-carr var-pow-carr*[*OF carrier-is-subring*]
unfolding *h-def'* **by** *auto*

have *degree f* = *k* + *degree g* **using** *False* **unfolding** *k-def* **by** *linarith*
also have ... = $degree\ [\alpha] + degree\ (X_{ring\text{-}of\ R} [\urcorner]_{?P}\ k) + degree\ g$
unfolding *var-pow-degree*[*OF carrier-is-subring*] **by** *simp*
also have ... = *degree h* + *degree g* **unfolding** *h-def'*
by (*intro arg-cong2*[**where** *f*=(+)] *degree-mult*[*symmetric*]
carrier-is-subring α-poly-carr var-pow-carr refl)
also have ... = $degree\ (h \otimes_{poly\text{-}ring\ (ring\text{-}of\ R)} g)$
by (*intro degree-mult*[*symmetric*] *carrier-is-subring h-carr g-carr*)
finally have *deg-f*: $degree\ f = degree\ (h \otimes_{poly\text{-}ring\ (ring\text{-}of\ R)} g)$ **by** *simp*

have *f'-carr*: $f' \in$ *carrier* (*poly-ring* (*ring-of R*))
using *f-carr h-carr g-carr* **unfolding** *f'-def'* **by** *auto*

have $hd\ f = \ominus_{ring\text{-}of\ R}\ (\alpha \otimes_{ring\text{-}of\ R}\ lead\text{-}coeff\ g)$
using *hd-g-unit hd-f-carr hd-g-carr α-unit α-carr* **unfolding** *α-def'*
by (*simp add*: *m-assoc l-minus*)
also have ... $= \ominus_{ring\text{-}of\ R}\ (hd\ h \otimes_{ring\text{-}of\ R}\ hd\ g)$
using *hd-f-carr α-carr α-poly-carr var-pow-carr*[*OF carrier-is-subring*] **unfolding** *h-def'*
by (*subst lead-coeff-mult*) (*simp-all add*:*algebra-simps*)
also have ... $= \ominus_{ring\text{-}of\ R}\ hd\ (h \otimes_{poly\text{-}ring\ (ring\text{-}of\ R)}\ g)$
using *h-carr g-carr* **by** (*subst lead-coeff-mult*) *auto*
finally have $hd\ f = \ominus_{ring\text{-}of\ R}\ hd\ (h \otimes_{poly\text{-}ring\ (ring\text{-}of\ R)}\ g)$
by *simp*
hence *len-f'*: *length* f' < *length* f **using** *deg-f h-carr g-carr d-poly-ring.integral*
unfolding *f'-def'* **by** (*intro poly-add-cancel-len f-carr*) *auto*
hence *f''-def'*: $f'' = f'$ **unfolding** *f''-def* **by** *simp*

have {*fst* (*s*,*t*),*snd* (*s*,*t*)} ⊆ *carrier* (*poly-ring* (*ring-of R*))
using *len-f' f''-def' f'-carr* **by** (*intro ind*(*2*)[**where** *x*=*f''*] *st-def*) *auto*
hence *s-carr*: *s* ∈ *carrier ?P* **and** *t-carr*: *t* ∈ *carrier ?P* **by** *auto*

have *r-def'*: $r = (s \ominus_{poly\text{-}ring\ (ring\text{-}of\ R)}\ h,\ t)$
using *h-carr domain-cD*[*OF d-poly*] **unfolding** *r-def a-minus-def*
using *ring-of-poly*[*OF ring-c*,*symmetric*] **by** *simp*

have *r-carr*: {*fst r*, *snd r*} ⊆ *carrier* (*poly-ring* (*ring-of R*))
using *s-carr t-carr h-carr* **unfolding** *r-def'* **by** *auto*
have $f = f'' \ominus_{?P}\ h \otimes_{?P}\ g$
using *h-carr g-carr f-carr* **unfolding** *f''-def' f'-def'* **by** *simp algebra*
also have ... $= (snd\ (s,t) \oplus_{?P}\ fst\ (s,t) \otimes_{?P}\ g) \ominus_{?P}\ h \otimes_{?P}\ g$
using *f'-carr f''-def' len-f'*
by (*intro arg-cong2*[**where** $f=\lambda x\ y.\ x \ominus_{?P}\ y$] *ind*(*1*) *st-def*) *auto*
also have ... $= t \oplus_{?P}\ (s \ominus_{?P}\ h) \otimes_{?P}\ g$
using *s-carr t-carr h-carr g-carr* **by** *simp algebra*
also have ... $= snd\ r \oplus_{poly\text{-}ring\ (ring\text{-}of\ R)}\ fst\ r \otimes_{poly\text{-}ring\ (ring\text{-}of\ R)}\ g$
unfolding *r-def'* **by** *simp*
finally have $f = snd\ r \oplus_{poly\text{-}ring\ (ring\text{-}of\ R)}\ fst\ r \otimes_{poly\text{-}ring\ (ring\text{-}of\ R)}\ g$ **by** *simp*
thus *?thesis* **using** *r-carr* **by** *auto*
qed
qed

hence *result*: *?result f r* {*fst r*, *snd r*} ⊆ *carrier* (*poly-ring* (*ring-of R*))
using *r-def* **by** *auto*
show *?thesis*
proof (*cases g* = [])
case *True* **then show** *?thesis* **by** (*simp add*:*long-division*$_C$*.simps pmod-def pdiv-def*)
next
case *False*
hence *snd r* = [] ∨ *degree* (*snd r*) < *degree g*
using *long-division-c-length* **unfolding** *r-def*
by (*metis One-nat-def Suc-pred length-greater-0-conv not-less-eq*)
moreover have $f = g \otimes_{?P} (fst\ r) \oplus_{poly\text{-}ring\ (ring\text{-}of\ R)} (snd\ r)$
using *result*(*1*,*2*) *assms*(*2*,*3*) **by** *simp algebra*
ultimately have *long-divides f g* (*fst r*, *snd r*)
using *result*(*2*) **unfolding** *long-divides-def* **by** (*auto simp*:*mem-Times-iff*)
hence (*fst r*, *snd r*) = (*pdiv f g*, *pmod f g*)
by (*intro long-divisionI*[*OF carrier-is-subfield*] *False assms*)
then show *?thesis* **unfolding** *r-def* **by** *simp*
qed
qed

definition *pdiv*$_C$ :: (*'a*,*'b*) *idx-ring-scheme* ⇒ *'a list* ⇒ *'a list* ⇒ *'a list* **where**
pdiv$_C$ *R f g* = *fst* (*long-division*$_C$ *R f g*)

lemma *pdiv-c*:
assumes *field*$_C$ *R*
assumes *f* ∈ *carrier* (*poly-ring* (*ring-of R*))
assumes *g* ∈ *carrier* (*poly-ring* (*ring-of R*))
shows *pdiv*$_C$ *R f g* = *ring.pdiv* (*ring-of R*) *f g*
unfolding *pdiv*$_C$*-def long-division-c*[*OF assms*] **by** *simp*

definition *pmod*$_C$ :: (*'a*,*'b*) *idx-ring-scheme* ⇒ *'a list* ⇒ *'a list* ⇒ *'a list* **where**
pmod$_C$ *R f g* = *snd* (*long-division*$_C$ *R f g*)

lemma *pmod-c*:
assumes *field*$_C$ *R*
assumes *f* ∈ *carrier* (*poly-ring* (*ring-of R*))
assumes *g* ∈ *carrier* (*poly-ring* (*ring-of R*))
shows *pmod*$_C$ *R f g* = *ring.pmod* (*ring-of R*) *f g*
unfolding *pmod*$_C$*-def long-division-c*[*OF assms*] **by** *simp*

function *ext-euclidean* ::
(*'a*,*'b*) *idx-ring-scheme* ⇒ *'a list* ⇒ *'a list* ⇒ (*'a list* × *'a list*) × *'a list*
where *ext-euclidean F f g* = (
if f = [] ∨ *g* = [] *then*

((1$_{C\,poly\ F}$, 1$_{C\,poly\ F}$),f +$_{C\,poly\ F}$ g)
else (
let (p,q) = long-division$_C$ F f g;
((u,v),r) = ext-euclidean F g q
*in ((v,u +$_{C\,poly\ F}$ (−$_{C\,poly\ F}$ (p *$_{C\,poly\ F}$ v))),r)))*
by *pat-completeness auto*

termination
apply (*relation measure* (λ(-, -, *f*). *length f*))
subgoal by *simp*
by (*metis case-prod-conv in-measure length-greater-0-conv long-division-c-length prod.sel*(*2*))

lemma (**in** *domain*) *pdivides-self*:
assumes *x* ∈ *carrier* (*poly-ring R*)
shows *x pdivides x*
proof −
interpret *d*:*domain poly-ring R* **by** (*rule univ-poly-is-domain*[*OF carrier-is-subring*])
show *?thesis*
using *assms* **unfolding** *pdivides-def*
by (*intro dividesI*[**where** *c*=$\mathbf{1}_{poly\text{-}ring\ R}$]) *simp-all*
qed

declare *ext-euclidean.simps*[*simp del*]

lemma *ext-euclidean*:
assumes *field$_C$ R*
defines *P* ≡ *poly-ring* (*ring-of R*)
assumes *f* ∈ *carrier* (*poly-ring* (*ring-of R*))
assumes *g* ∈ *carrier* (*poly-ring* (*ring-of R*))
defines *r* ≡ *ext-euclidean R f g*
shows *snd r* = *f* ⊗$_P$ (*fst* (*fst r*)) ⊕$_P$ *g* ⊗$_P$ (*snd* (*fst r*)) (**is** *?T1*)
and *snd r pdivides$_{ring\text{-}of\ R}$ f* (**is** *?T2*) *snd r pdivides$_{ring\text{-}of\ R}$ g* (**is** *?T3*)
and {*snd r*, *fst* (*fst r*), *snd* (*fst r*)} ⊆ *carrier P* (**is** *?T4*)
and *snd r* = [] ⟶ *f* = [] ∧ *g* = [] (**is** *?T5*)
proof −
let *?P*= *poly-ring* (*ring-of R*)

interpret *field ring-of R* **using** *assms*(*1*) **unfolding** *field$_C$-def* **by** *auto*
interpret *d-poly-ring*: *domain poly-ring* (*ring-of R*)
by (*rule univ-poly-is-domain*[*OF carrier-is-subring*])

have *ring-c*: *ring$_C$ R* **using** *assms*(*1*) **unfolding** *field$_C$-def domain$_C$-def cring$_C$-def* **by** *auto*
have *d-poly*: *domain$_C$* (*poly R*) **using** *assms* (*1*) **unfolding** *field$_C$-def*

```
by (intro poly-domain) auto

  have pdiv-zero: x pdivides_ring-of R 0_?P if x ∈ carrier ?P for x
     using that unfolding univ-poly-zero by (intro pdivides-zero[OF
carrier-is-subring])

  have snd r = f ⊗_?P (fst (fst r)) ⊕_?P g ⊗_?P (snd (fst r)) ∧
    snd r pdivides_ring-of R f ∧ snd r pdivides_ring-of R g ∧
    {snd r, fst (fst r), snd (fst r)} ⊆ carrier ?P ∧
    (snd r = [] ⟶ f = [] ∧ g = [])
    if r = ext-euclidean R f g {f,g} ⊆ carrier ?P
    using that
  proof (induction length g arbitrary: f g r rule:nat-less-induct)
    case 1
    have ind:
      snd s = x ⊗_?P fst (fst s) ⊕_?P y ⊗_?P snd (fst s)
      snd s pdivides_ring-of R x snd s pdivides_ring-of R y
      {snd s, fst (fst s), snd (fst s)} ⊆ carrier ?P
      (snd s = [] ⟶ x = [] ∧ y = [])
       if length y < length g s = ext-euclidean R x y {x, y} ⊆ carrier
?P
      for x y s using that 1(1) by metis+
    show ?case
    proof (cases f = [] ∨ g = [])
      case True
      hence r-def: r = ((1_?P, 1_?P), f ⊕_?P g) unfolding 1(2)
      by (simp add:ext-euclidean.simps domain-cD[OF d-poly] ring-of-poly[OF
ring-c])

      consider f = 0_?P | g = 0_?P
        using True unfolding univ-poly-zero by auto
      hence snd r pdivides_ring-of R f ∧ snd r pdivides_ring-of R g
          using 1(3) pdiv-zero pdivides-self unfolding r-def by cases
auto
       moreover have snd r = f ⊗_?P fst (fst r) ⊕_?P g ⊗_?P snd (fst
r)
        using 1(3) unfolding r-def by simp
      moreover have {snd r, fst (fst r), snd (fst r)} ⊆ carrier ?P
        using 1(3) unfolding r-def by auto
      moreover have snd r = [] ⟶ f = [] ∧ g = []
       using 1(3) True unfolding r-def by (auto simp:univ-poly-zero)
      ultimately show ?thesis by (intro conjI) metis+
    next
      case False
      obtain p q where pq-def: (p,q) = long-division_C R f g
        by (metis surj-pair)
      obtain u v s where uvs-def: ((u,v),s) = ext-euclidean R g q
        by (metis surj-pair)
```

have $(p,q) = (pdiv\ f\ g,\ pmod\ f\ g)$
using *1(3)* **unfolding** *pq-def* **by** (*intro long-division-c*[*OF assms*(*1*)]) *auto*
hence *p-def*: $p = pdiv\ f\ g$ **and** *q-def*: $q = pmod\ f\ g$ **by** *auto*
have *p-carr*: $p \in carrier\ ?P$ **and** *q-carr*: $q \in carrier\ ?P$
using *1(3) long-division-closed*[*OF carrier-is-subfield*] **unfolding** *p-def q-def* **by** *auto*

have $length\ g > 0$ **using** *False* **by** *auto*
hence *len-q*: $length\ q < length\ g$ **using** *long-division-c-length pq-def* **by** (*metis snd-conv*)
have *s-eq*: $s = g \otimes_{?P} u \oplus_{?P} q \otimes_{?P} v$
and *s-div-g*: $s\ pdivides_{ring\text{-}of\ R}\ g$
and *s-div-q*: $s\ pdivides_{ring\text{-}of\ R}\ q$
and *suv-carr*: $\{s,u,v\} \subseteq carrier\ ?P$
and *s-zero-iff*: $s = [] \longrightarrow g = [] \wedge q = []$
using *ind*[*OF len-q uvs-def -*] *q-carr 1(3)* **by** *auto*

have $r = ((v,u +_{C\ poly\ R} (-_{C\ poly\ R} (p *_{C\ poly\ R} v))),s)$ **unfolding** *1(2)* **using** *False*
by (*subst ext-euclidean.simps*) (*simp add*: *pq-def*[*symmetric*] *uvs-def*[*symmetric*])
also have ... $= ((v,\ u \ominus_{?P} (p \otimes_{?P} v)),\ s)$ **using** *p-carr suv-carr domain-cD*[*OF d-poly*]
unfolding *a-minus-def ring-of-poly*[*OF ring-c*] **by** (*intro arg-cong2*[**where** *f=Pair*] *refl*) *simp*
finally have *r-def*: $r = ((v,\ u \ominus_{?P} (p \otimes_{?P} v)),\ s)$ **by** *simp*

have $snd\ r = g \otimes_{?P} u \oplus_{?P} q \otimes_{?P} v$ **unfolding** *r-def s-eq* **by** *simp*
also have ... $= g \otimes_{?P} u \oplus_{?P} (f \ominus_{?P} g \otimes_{?P} p) \otimes_{?P} v$
using *1(3) p-carr q-carr suv-carr*
by (*subst pdiv-pmod*[*OF carrier-is-subfield*, *of f g*])
(*simp-all add*:*p-def*[*symmetric*] *q-def*[*symmetric*], *algebra*)
also have ... $= f \otimes_{?P} v \oplus_{?P} g \otimes_{?P} (u \ominus_{?P} ((p \otimes_{?P} v)))$
using *1(3) p-carr q-carr suv-carr* **by** *simp algebra*
finally have *r1*: $snd\ r = f \otimes_{?P} fst\ (fst\ r) \oplus_{?P} g \otimes_{?P} snd\ (fst\ r)$
unfolding *r-def* **by** *simp*
have $pmod\ f\ s = pmod\ (g \otimes_{?P} p \oplus_{?P} q)\ s$ **using** *1(3)*
by (*subst pdiv-pmod*[*OF carrier-is-subfield*, *of f g*])
(*simp-all add*:*p-def*[*symmetric*] *q-def*[*symmetric*])
also have ... $= pmod\ (g \otimes_{?P} p)\ s \oplus_{?P} pmod\ q\ s$
using *1(3) p-carr q-carr suv-carr*
by (*subst long-division-add*[*OF carrier-is-subfield*]) *simp-all*
also have ... $= pmod\ (pmod\ g\ s \otimes_{?P} p)\ s \oplus_{?P} []$
using *1(3) p-carr q-carr suv-carr s-div-q*
by (*intro arg-cong2*[**where** $f=(\oplus_{?P})$] *pmod-mult-left*)
(*simp-all add*: *pmod-zero-iff-pdivides*[*OF carrier-is-subfield*])

also have ... = *pmod* ($\mathbf{0}_{?P}$ $\otimes_{?P}$ *p*) *s* $\oplus_{?P}$ $\mathbf{0}_{?P}$ **unfolding** *univ-poly-zero*
using *1(3) p-carr q-carr suv-carr s-div-g* **by** (*intro arg-cong2*[**where** *f*=($\oplus_{?P}$)]
arg-cong2[**where** *f*=($\otimes_{?P}$)] *arg-cong2*[**where** *f*=*pmod*])
(*simp-all add*: *pmod-zero-iff-pdivides*[*OF carrier-is-subfield*])
also have ... = *pmod* $\mathbf{0}_{?P}$ *s*
using *p-carr suv-carr long-division-closed*[*OF carrier-is-subfield*] **by** *simp*
also have ... = [] **unfolding** *univ-poly-zero*
using *suv-carr long-division-zero*(*2*)[*OF carrier-is-subfield*] **by** *simp*
finally have *pmod f s* = [] **by** *simp*
hence *r2*: *snd r pdivides*$_{ring\text{-}of\ R}$ *f* **using** *suv-carr 1(3)* **unfolding** *r-def*
by (*subst pmod-zero-iff-pdivides*[*OF carrier-is-subfield*,*symmetric*]) *simp-all*
have *r3*: *snd r pdivides*$_{ring\text{-}of\ R}$ *g* **unfolding** *r-def* **using** *s-div-g* **by** *auto*
have *r4*: {*snd r*, *fst* (*fst r*), *snd* (*fst r*)} ⊆ *carrier ?P*
using *suv-carr p-carr* **unfolding** *r-def* **by** *simp-all*
have *r5*: *f* = [] ∧ *g* = [] **if** *snd r* = []
proof −
have *r5-a*: *g* = [] ∧ *q* = [] **using** *that s-zero-iff* **unfolding** *r-def* **by** *simp*
hence *pmod f* [] = [] **unfolding** *q-def* **by** *auto*
hence *f* = [] **using** *pmod-def* **by** *simp*
thus *?thesis* **using** *r5-a* **by** *auto*
qed

show *?thesis* **using** *r1 r2 r3 r4 r5* **by** (*intro conjI*) *metis+*
qed
qed
thus *?T1 ?T2 ?T3 ?T4 ?T5* **using** *assms* **by** *auto*
qed

end

11 Executable Factor Rings

theory *Finite-Fields-Mod-Ring-Code*
imports *Finite-Fields-Indexed-Algebra-Code Ring-Characteristic*
begin

definition *mod-ring* :: *nat* ⇒ *nat idx-ring-enum*
where *mod-ring n* = (|
idx-pred = (λ*x*. *x* < *n*),
idx-uminus = (λ*x*. (*n*−*x*) *mod n*),
idx-plus = (λ*x y*. (*x*+*y*) *mod n*),

idx-udivide = (λ*x*. *nat* (*fst* (*bezout-coefficients* (*int x*) (*int n*)) *mod* (*int n*))),
idx-mult = (λ*x y*. (*x*∗*y*) *mod n*),
idx-zero = *0*,
idx-one = *1*,
idx-size = *n*,
idx-enum = *id*,
idx-enum-inv = *id*
⦈

lemma *zfact-iso-0*:
assumes *n* > *0*
shows *zfact-iso n 0* = $\mathbf{0}_{ZFact\ (int\ n)}$
proof −
let *?I* = *Idl*$_{\mathcal{Z}}$ {*int n*}
have *ideal-I*: *ideal ?I* $\mathcal{Z}$
by (*simp add*: *int.genideal-ideal*)

interpret *i*:*ideal ?I* $\mathcal{Z}$ **using** *ideal-I* **by** *simp*
interpret *s*:*ring-hom-ring* $\mathcal{Z}$ *ZFact* (*int n*) (+>$_{\mathcal{Z}}$) *?I*
using *i.rcos-ring-hom-ring ZFact-def* **by** *auto*

show *?thesis*
by (*simp add*:*zfact-iso-def ZFact-def*)
qed

lemma *zfact-prime-is-field*:
assumes *Factorial-Ring.prime* (*p* :: *nat*)
shows *field* (*ZFact* (*int p*))
using *zfact-prime-is-finite-field*[*OF assms*] *finite-field-def* **by** *auto*

definition *zfact-iso-inv* :: *nat* ⇒ *int set* ⇒ *nat* **where**
zfact-iso-inv p = *the-inv-into* {..<*p*} (*zfact-iso p*)

lemma *zfact-iso-inv-0*:
assumes *n-ge-0*: *n* > *0*
shows *zfact-iso-inv n* $\mathbf{0}_{ZFact\ (int\ n)}$ = *0*
unfolding *zfact-iso-inv-def zfact-iso-0*[*OF n-ge-0*, *symmetric*] **using** *n-ge-0*
by (*rule the-inv-into-f-f*[*OF zfact-iso-inj*], *simp add*:*mod-ring-def*)

lemma *zfact-coset*:
assumes *n-ge-0*: *n* > *0*
assumes *x* ∈ *carrier* (*ZFact* (*int n*))
defines *I* ≡ *Idl*$_{\mathcal{Z}}$ {*int n*}
shows *x* = *I* +>$_{\mathcal{Z}}$ (*int* (*zfact-iso-inv n x*))
proof −
have *x* ∈ *zfact-iso n* ‘ {..<*n*}
using *assms zfact-iso-ran* **by** *simp*

hence *zfact-iso n* (*zfact-iso-inv n x*) = *x*
unfolding *zfact-iso-inv-def* **by** (*intro f-the-inv-into-f zfact-iso-inj*) *auto*
thus *?thesis* **unfolding** *zfact-iso-def I-def* **by** *blast*
qed

lemma *zfact-iso-inv-bij*:
assumes *n* > *0*
shows *bij-betw* (*zfact-iso-inv n*) (*carrier* (*ZFact* (*int n*))) (*carrier* (*ring-of* (*mod-ring n*)))
proof −
have *bij-betw* (*the-inv-into* {..<*n*} (*zfact-iso n*)) (*carrier* (*ZFact* (*int n*))) {..<*n*}
by (*intro bij-betw-the-inv-into zfact-iso-bij*[*OF assms*])
thus *?thesis*
unfolding *zfact-iso-inv-def mod-ring-def ring-of-def lessThan-def* **by** *simp*
qed

lemma *zfact-iso-inv-is-ring-iso*:
fixes *n* :: *nat*
assumes *n-ge-1*: *n* > *1*
shows *zfact-iso-inv n* ∈ *ring-iso* (*ZFact* (*int n*)) (*ring-of* (*mod-ring n*)) (**is** *?f* ∈ -)
proof (*rule ring-iso-memI*)
interpret *r*:*cring* (*ZFact* (*int n*))
using *ZFact-is-cring* **by** *simp*

define *I* **where** $I = Idl_{\mathcal{Z}}$ {*int n*}

have *n-ge-0*: *n* > *0* **using** *n-ge-1* **by** *simp*

interpret *i*:*ideal I* $\mathcal{Z}$
unfolding *I-def* **using** *int.genideal-ideal* **by** *simp*

interpret *s*:*ring-hom-ring* $\mathcal{Z}$ *ZFact* (*int n*) $(+>_{\mathcal{Z}})$ *I*
using *i.rcos-ring-hom-ring ZFact-def I-def* **by** *auto*

show *zfact-iso-inv n x* ∈ *carrier* (*ring-of* (*mod-ring n*)) **if** *x* ∈ *carrier* (*ZFact* (*int n*)) **for** *x*
proof −
have *zfact-iso-inv n x* ∈ {..<*n*}
unfolding *zfact-iso-inv-def* **using** *that zfact-iso-ran*[*OF n-ge-0*]
by (*intro the-inv-into-into zfact-iso-inj n-ge-0*) *auto*
thus *zfact-iso-inv n x* ∈ *carrier* (*ring-of* (*mod-ring n*))
by (*simp add*:*ring-of-def mod-ring-def*)
qed

show $?f\ (x \otimes_{ZFact\ (int\ n)} y) = ?f\ x \otimes_{ring\text{-}of\ (mod\text{-}ring\ n)} ?f\ y$

if *x-carr*: $x \in$ *carrier* (*ZFact* (*int n*)) **and** *y-carr*: $y \in$ *carrier* (*ZFact* (*int n*)) **for** *x y*
proof −
define x' **where** $x' =$ *zfact-iso-inv n x*
define y' **where** $y' =$ *zfact-iso-inv n y*
have $x \otimes_{ZFact\ (int\ n)} y = (I +>_{\mathcal{Z}} (int\ x')) \otimes_{ZFact\ (int\ n)} (I +>_{\mathcal{Z}} (int\ y'))$
unfolding x'*-def* y'*-def*
using *x-carr y-carr zfact-coset*[*OF n-ge-0*] *I-def* **by** *simp*
also have ... $= (I +>_{\mathcal{Z}} (int\ x' * int\ y'))$
by *simp*
also have ... $= (I +>_{\mathcal{Z}} (int\ ((x' * y')\ mod\ n)))$
unfolding *I-def zmod-int* **by** (*rule int-cosetI*[*OF n-ge-0*],*simp*)
also have ... $= (I +>_{\mathcal{Z}} (x' \otimes_{ring\text{-}of\ (mod\text{-}ring\ n)} y'))$
unfolding *ring-of-def mod-ring-def* **by** *simp*
also have ... $=$ *zfact-iso n* $(x' \otimes_{ring\text{-}of\ (mod\text{-}ring\ n)} y')$
unfolding *zfact-iso-def I-def* **by** *simp*
finally have $a{:}x \otimes_{ZFact\ (int\ n)} y =$ *zfact-iso n* $(x' \otimes_{ring\text{-}of\ (mod\text{-}ring\ n)} y')$
by *simp*
have $b{:}x' \otimes_{ring\text{-}of\ (mod\text{-}ring\ n)} y' \in \{..<n\}$
using *mod-ring-def n-ge-0* **by** (*auto simp*:*ring-of-def*)
have *?f* (*zfact-iso n* $(x' \otimes_{ring\text{-}of\ (mod\text{-}ring\ n)} y')) = x' \otimes_{ring\text{-}of\ (mod\text{-}ring\ n)} y'$
unfolding *zfact-iso-inv-def*
by (*rule the-inv-into-f-f*[*OF zfact-iso-inj*[*OF n-ge-0*] *b*])
thus
zfact-iso-inv n $(x \otimes_{ZFact\ (int\ n)} y) =$
zfact-iso-inv n x $\otimes_{ring\text{-}of\ (mod\text{-}ring\ n)}$ *zfact-iso-inv n y*
using *a* x'*-def* y'*-def* **by** *simp*
qed

show *zfact-iso-inv n* $(x \oplus_{ZFact\ (int\ n)} y) =$
zfact-iso-inv n x $\oplus_{ring\text{-}of\ (mod\text{-}ring\ n)}$ *zfact-iso-inv n y*
if *x-carr*: $x \in$ *carrier* (*ZFact* (*int n*)) **and** *y-carr*: $y \in$ *carrier* (*ZFact* (*int n*)) **for** *x y*
proof −
define x' **where** $x' =$ *zfact-iso-inv n x*
define y' **where** $y' =$ *zfact-iso-inv n y*
have $x \oplus_{ZFact\ (int\ n)} y = (I +>_{\mathcal{Z}} (int\ x')) \oplus_{ZFact\ (int\ n)} (I +>_{\mathcal{Z}} (int\ y'))$
unfolding x'*-def* y'*-def*
using *x-carr y-carr zfact-coset*[*OF n-ge-0*] *I-def* **by** *simp*
also have ... $= (I +>_{\mathcal{Z}} (int\ x' + int\ y'))$
by *simp*
also have ... $= (I +>_{\mathcal{Z}} (int\ ((x' + y')\ mod\ n)))$
unfolding *I-def zmod-int* **by** (*rule int-cosetI*[*OF n-ge-0*],*simp*)
also have ... $= (I +>_{\mathcal{Z}} (x' \oplus_{ring\text{-}of\ (mod\text{-}ring\ n)} y'))$

unfolding *mod-ring-def ring-of-def* **by** *simp*
also have ... = *zfact-iso n* $(x' \oplus_{ring\text{-}of\ (mod\text{-}ring\ n)}\ y')$
unfolding *zfact-iso-def I-def* **by** *simp*
finally have *a*:$x \oplus_{ZFact\ (int\ n)}\ y$ = *zfact-iso n* $(x' \oplus_{ring\text{-}of\ (mod\text{-}ring\ n)}\ y')$
by *simp*
have *b*:$x' \oplus_{ring\text{-}of\ (mod\text{-}ring\ n)}\ y' \in \{..<n\}$
using *mod-ring-def n-ge-0* **by** (*auto simp*:*ring-of-def*)
have *?f* (*zfact-iso n* $(x' \oplus_{ring\text{-}of\ (mod\text{-}ring\ n)}\ y'))$ = $x' \oplus_{ring\text{-}of\ (mod\text{-}ring\ n)}\ y'$
unfolding *zfact-iso-inv-def*
by (*rule the-inv-into-f-f*[*OF zfact-iso-inj*[*OF n-ge-0*] *b*])
thus *?f* $(x \oplus_{ZFact\ (int\ n)}\ y)$ = *?f x* $\oplus_{ring\text{-}of\ (mod\text{-}ring\ n)}$ *?f y*
using *a x'-def y'-def* **by** *simp*
qed

have $\mathbf{1}_{ZFact\ (int\ n)}$ = *zfact-iso n* $(\mathbf{1}_{ring\text{-}of\ (mod\text{-}ring\ n)})$
by (*simp add*:*zfact-iso-def ZFact-def I-def*[*symmetric*] *ring-of-def mod-ring-def*)

thus *zfact-iso-inv n* $\mathbf{1}_{ZFact\ (int\ n)} = \mathbf{1}_{ring\text{-}of\ (mod\text{-}ring\ n)}$
unfolding *zfact-iso-inv-def mod-ring-def ring-of-def*
using *the-inv-into-f-f*[*OF zfact-iso-inj*] *n-ge-1* **by** *simp*

show *bij-betw* (*zfact-iso-inv n*) (*carrier* (*ZFact* (*int n*))) (*carrier* (*ring-of* (*mod-ring n*)))
by (*intro zfact-iso-inv-bij n-ge-0*)
qed

lemma *mod-ring-finite*:
finite (*carrier* (*ring-of* (*mod-ring n*)))
by (*simp add*:*mod-ring-def ring-of-def*)

lemma *mod-ring-carr*:
$x \in$ *carrier* (*ring-of* (*mod-ring n*)) $\longleftrightarrow x < n$
by (*simp add*:*mod-ring-def ring-of-def*)

lemma *mod-ring-is-cring*:
assumes *n-ge-1*: $n > 1$
shows *cring* (*ring-of* (*mod-ring n*))
proof −
have *n-ge-0*: $n > 0$ **using** *n-ge-1* **by** *simp*

interpret *cring ZFact* (*int n*)
using *ZFact-is-cring* **by** *simp*

have *cring* ((*ring-of* (*mod-ring n*)) ⦇ *zero* := *zfact-iso-inv n* $\mathbf{0}_{ZFact\ (int\ n)}$ ⦈)

by (*rule ring-iso-imp-img-cring*[*OF zfact-iso-inv-is-ring-iso*[*OF n-ge-1*]])
moreover have
ring-of (*mod-ring n*) (| *zero* := *zfact-iso-inv n* $\mathbf{0}_{ZFact\ (int\ n)}$ |) = *ring-of* (*mod-ring n*)
using *zfact-iso-inv-0*[*OF n-ge-0*] **by** (*simp add*:*mod-ring-def ring-of-def*)
ultimately show *?thesis* **by** *simp*
qed

lemma *zfact-iso-is-ring-iso*:
assumes *n-ge-1*: *n* > *1*
shows *zfact-iso n* ∈ *ring-iso* (*ring-of* (*mod-ring n*)) (*ZFact* (*int n*))
proof −
have *r*:*ring* (*ZFact* (*int n*))
using *ZFact-is-cring cring.axioms*(*1*) **by** *blast*

interpret *s*: *ring* (*ring-of* (*mod-ring n*))
using *mod-ring-is-cring cring.axioms*(*1*) *n-ge-1* **by** *blast*
have *n-ge-0*: *n* > *0* **using** *n-ge-1* **by** *linarith*

have *inv-into* (*carrier* (*ZFact* (*int n*))) (*zfact-iso-inv n*)
∈ *ring-iso* (*ring-of* (*mod-ring n*)) (*ZFact* (*int n*))
using *ring-iso-set-sym*[*OF r zfact-iso-inv-is-ring-iso*[*OF n-ge-1*]] **by** *simp*
moreover have *inv-into* (*carrier* (*ZFact* (*int n*))) (*zfact-iso-inv n*) *x* = *zfact-iso n x*
if *x* ∈ *carrier* (*ring-of* (*mod-ring n*)) **for** *x*
proof −
have *x* ∈ {..<*n*} **using** *that* **by** (*simp add*:*mod-ring-def ring-of-def*)
thus *inv-into* (*carrier* (*ZFact* (*int n*))) (*zfact-iso-inv n*) *x* = *zfact-iso n x*
using *zfact-iso-inv-bij*[*OF n-ge-0*] *zfact-iso-bij*[*OF n-ge-0*] **unfolding** *zfact-iso-inv-def*
by (*intro inv-into-f-eq bij-betw-apply*[*OF zfact-iso-inv-bij*[*OF n-ge-0*]] *the-inv-into-f-f*)
(*auto intro*:*bij-betw-imp-inj-on simp*:*bij-betwE*)
qed

ultimately show *?thesis* **using** *s.ring-iso-restrict* **by** *blast*
qed

If *p* is a prime than *mod-ring p* is a field:

lemma *mod-ring-is-field*:
assumes *Factorial-Ring.prime p*
shows *field* (*ring-of* (*mod-ring p*))
proof −
have *p-ge-0*: *p* > *0* **using** *assms prime-gt-0-nat* **by** *blast*
have *p-ge-1*: *p* > *1* **using** *assms prime-gt-1-nat* **by** *blast*

interpret *field ZFact* (*int p*)

using *zfact-prime-is-field*[*OF assms*] **by** *simp*

have *field* ((*ring-of* (*mod-ring p*)) (| *zero* := *zfact-iso-inv p* $\mathbf{0}_{ZFact\ (int\ p)}$ |))
by (*rule ring-iso-imp-img-field*[*OF zfact-iso-inv-is-ring-iso*[*OF p-ge-1*]])

moreover have
(*ring-of* (*mod-ring p*)) (| *zero* := *zfact-iso-inv p* $\mathbf{0}_{ZFact\ (int\ p)}$ |) = *ring-of* (*mod-ring p*)
using *zfact-iso-inv-0*[*OF p-ge-0*] **by** (*simp add*:*mod-ring-def ring-of-def*)
ultimately show *?thesis* **by** *simp*
qed

lemma *mod-ring-is-ring-c*:
assumes *n* > *1*
shows $cring_C$ (*mod-ring n*)
proof (*intro cring-cI mod-ring-is-cring assms*)
fix *x*
assume *a*:*x* ∈ *carrier* (*ring-of* (*mod-ring n*))
hence *x-le-n*: *x* < *n* **unfolding** *mod-ring-def ring-of-def* **by** *simp*

interpret *cring* (*ring-of* (*mod-ring n*)) **by** (*intro mod-ring-is-cring assms*)

show $-_{C\ mod\text{-}ring\ n}$ *x* = $\ominus_{ring\text{-}of\ (mod\text{-}ring\ n)}$ *x* **using** *x-le-n*
by (*intro minus-equality*[*symmetric*] *a*) (*simp-all add*:*ring-of-def mod-ring-def mod-simps*)
next
fix *x*
assume *a*:*x* ∈ *Units* (*ring-of* (*mod-ring n*))

let *?l* = *fst* (*bezout-coefficients* (*int x*) (*int n*))
let *?r* = *snd* (*bezout-coefficients* (*int x*) (*int n*))

interpret *cring ring-of* (*mod-ring n*) **by** (*intro mod-ring-is-cring assms*)

obtain *y* **where** *x* $\otimes_{ring\text{-}of\ (mod\text{-}ring\ n)}$ *y* = $\mathbf{1}_{ring\text{-}of\ (mod\text{-}ring\ n)}$
using *a* **by** (*meson Units-r-inv-ex*)
hence *x* ∗ *y mod n* = *1* **by** (*simp-all add*:*mod-ring-def ring-of-def*)
hence *gcd x n* = *1* **by** (*metis dvd-triv-left gcd.assoc gcd-1-nat gcd-nat.absorb-iff1 gcd-red-nat*)
hence *0*:*gcd* (*int x*) (*int n*) = *1* **unfolding** *gcd-int-int-eq* **by** *simp*

have *int x* ∗ *?l mod int n* = (*?l* ∗ *int x* + *?r* ∗ *int n*) *mod int n*
using *assms* **by** (*simp add*:*mod-simps algebra-simps*)
also have ... = (*gcd* (*int x*) (*int n*)) *mod int n*
by (*intro arg-cong2*[**where** *f*=(*mod*)] *refl bezout-coefficients*) *simp*
also have ... = *1* **unfolding** *0* **using** *assms* **by** *simp*

finally have *int x ∗ ?l mod int n = 1* **by** *simp*
hence *int x ∗ nat (fst (bezout-coefficients (int x) (int n)) mod int n) mod n = 1*
using *assms* **by** (*simp add:mod-simps*)
hence *x ∗ nat (fst (bezout-coefficients (int x) (int n)) mod int n) mod n = 1*
by (*metis nat-mod-as-int nat-one-as-int of-nat-mult*)
hence $x \otimes_{ring\text{-}of\ (mod\text{-}ring\ n)} x\ ^{-1}{}_{C\, mod\text{-}ring\ n} = \mathbf{1}_{ring\text{-}of\ (mod\text{-}ring\ n)}$
using *assms* **unfolding** *mod-ring-def ring-of-def* **by** *simp*
moreover have *nat (fst (bezout-coefficients (int x) (int n)) mod int n) < n*
using *assms* **by** (*subst nat-less-iff*) *auto*
hence $x\ ^{-1}{}_{C\, mod\text{-}ring\ n} \in carrier\ (ring\text{-}of\ (mod\text{-}ring\ n))$
using *assms* **unfolding** *mod-ring-def ring-of-def* **by** *simp*
moreover have *x ∈ carrier (ring-of (mod-ring n))* **using** *a* **by** *auto*
ultimately show $x\ ^{-1}{}_{C\, mod\text{-}ring\ n} = inv_{ring\text{-}of\ (mod\text{-}ring\ n)}\ x$
by (*intro comm-inv-char*[*symmetric*])
qed

lemma *mod-ring-is-field-c*:
assumes*Factorial-Ring.prime p*
shows $field_C$ (*mod-ring p*)
unfolding $field_C$*-def* $domain_C$*-def*
by (*intro conjI mod-ring-is-ring-c mod-ring-is-field assms prime-gt-1-nat domain.axioms*(*1*) *field.axioms*(*1*))

lemma *mod-ring-is-enum-c*:
shows $enum_C$ (*mod-ring n*)
by (*intro enum-cI*) (*simp-all add:mod-ring-def ring-of-def Coset.order-def lessThan-def*)

end

12 Executable Code for Rabin's Irreducibility Test

theory *Rabin-Irreducibility-Test-Code*
imports
Finite-Fields-Poly-Ring-Code
Finite-Fields-Mod-Ring-Code
Rabin-Irreducibility-Test
begin

fun $pcoprime_C$ :: (*'a, 'b*) *idx-ring-scheme ⇒ 'a list ⇒ 'a list ⇒ bool*
where $pcoprime_C$ *R f g* = (*length* (*snd* (*ext-euclidean R f g*)) = *1*)

declare $pcoprime_C$*.simps*[*simp del*]

```
lemma pcoprime-c:
  assumes field_C R
  assumes f ∈ carrier (poly-ring (ring-of R))
  assumes g ∈ carrier (poly-ring (ring-of R))
  shows pcoprime_C R f g ⟷ pcoprime_ring-of R f g  (is ?L = ?R)
proof (cases f = [] ∧ g = [])
  case True
  interpret field ring-of R
    using assms(1) unfolding field_C-def by simp
  interpret d-poly-ring: domain poly-ring (ring-of R)
    by (rule univ-poly-is-domain[OF carrier-is-subring])

  have ?L = False using True by (simp add: pcoprime_C.simps ext-euclidean.simps
poly-def)
  also have ... ⟷ (length 0_poly-ring (ring-of R) = 1) by (simp
add:univ-poly-zero)
  also have ... ⟷ pcoprime_ring-of R 0_poly-ring (ring-of R) []
    by (subst pcoprime-zero-iff) (simp-all)
  also have ... ⟷ ?R using True by (simp add: univ-poly-zero)
  finally show ?thesis by simp
next
  case False

  let ?P = poly-ring (ring-of R)
  interpret field ring-of R
    using assms(1) unfolding field_C-def by simp
  interpret d-poly-ring: domain poly-ring (ring-of R)
    by (rule univ-poly-is-domain[OF carrier-is-subring])

  obtain s u v where suv-def: ((u,v),s) = ext-euclidean R f g by
(metis surj-pair)

  have s-eq:s = f ⊗_?P u ⊕_?P g ⊗_?P v (is ?T1)
   and s-div-f: s pdivides_ring-of R f and s-div-g: s pdivides_ring-of R g
(is ?T3)
    and suv-carr: {s, u, v} ⊆ carrier ?P
    and s-nz: s ≠ []
   using False suv-def[symmetric] ext-euclidean[OF assms(1,2,3)] by
auto

  have ?L ⟷ length s = 1 using suv-def[symmetric] by (simp
add:pcoprime_C.simps)
  also have ... ⟷ ?R
    unfolding pcoprime-def
  proof (intro iffI impI ballI)
    fix r assume len-s: length s = 1
    assume r-carr:r ∈ carrier ?P
      and r pdivides_ring-of R f ∧ r pdivides_ring-of R g
      hence r-div: pmod f r = 0_?P  pmod g r = 0_?P unfolding
```

univ-poly-zero
using *assms*(*2*,*3*) *pmod-zero-iff-pdivides*[*OF carrier-is-subfield*]
by *auto*

have *pmod s r* = *pmod* (*f* $\otimes_{?P}$ *u*) *r* $\oplus_{?P}$ *pmod* (*g* $\otimes_{?P}$ *v*) *r*
using *r-carr suv-carr assms* **unfolding** *s-eq*
by (*intro long-division-add*[*OF carrier-is-subfield*]) *auto*
also have ... = *pmod* (*pmod f r* $\otimes_{?P}$ *u*) *r* $\oplus_{?P}$ *pmod* (*pmod g r* $\otimes_{?P}$ *v*) *r*
using *r-carr suv-carr assms* **by** (*intro arg-cong2*[**where** *f*=($\oplus_{?P}$)] *pmod-mult-left*) *auto*
also have ... = *pmod* $\mathbf{0}_{?P}$ *r* $\oplus_{?P}$ *pmod* $\mathbf{0}_{?P}$ *r*
using *suv-carr* **unfolding** *r-div* **by** *simp*
also have ... = [] **using** *r-carr* **unfolding** *univ-poly-zero*
by (*simp add*: *long-division-zero*[*OF carrier-is-subfield*] *univ-poly-add*)
finally have *pmod s r* = [] **by** *simp*
hence *r pdivides*$_{ring\text{-}of\ R}$ *s*
using *r-carr suv-carr pmod-zero-iff-pdivides*[*OF carrier-is-subfield*]
by *auto*
hence *degree r* $\leq$ *degree s*
using *s-nz r-carr suv-carr* **by** (*intro pdivides-imp-degree-le*[*OF carrier-is-subring*]) *auto*
thus *degree r* = *0* **using** *len-s* **by** *simp*
next
assume $\forall$ *r*$\in$*carrier ?P*. *r pdivides*$_{ring\text{-}of\ R}$ *f* $\wedge$ *r pdivides*$_{ring\text{-}of\ R}$ *g* $\longrightarrow$ *degree r* = *0*
hence *degree s* = *0* **using** *s-div-f s-div-g suv-carr* **by** *simp*
thus *length s* =*1* **using** *s-nz*
by (*metis diff-is-0-eq diffs0-imp-equal length-0-conv less-one linorder-le-less-linear*)
qed
finally show *?thesis* **by** *simp*
qed

The following is a fast version of *pmod* for polynomials (to a high power) that need to be reduced, this is used for the higher order term of the Gauss polynomial.

fun *pmod-pow*$_C$ :: ('*a*,'*b*) *idx-ring-scheme* $\Rightarrow$ '*a list* $\Rightarrow$ *nat* $\Rightarrow$ '*a list* $\Rightarrow$ '*a list*
where *pmod-pow*$_C$ *F f n g* = (
let r = (*if n* $\geq$ *2 then pmod-pow*$_C$ *F f* (*n div 2*) *g* $\hat{}_{C\,poly\ F}$ *2 else* $1_{C\,poly\ F}$)
in pmod$_C$ *F* (*r* $*_{C\,poly\ F}$ (*f* $\hat{}_{C\,poly\ F}$ (*n mod 2*))) *g*)

declare *pmod-pow*$_C$.*simps*[*simp del*]

lemma *pmod-pow-c*:
assumes *field*$_C$ *R*
assumes *f* $\in$ *carrier* (*poly-ring* (*ring-of R*))

assumes *g* ∈ *carrier* (*poly-ring* (*ring-of R*))
shows *pmod-pow*$_C$ *R f n g* = *ring.pmod* (*ring-of R*) (*f* [^]$_{poly\text{-}ring\ (ring\text{-}of\ R)}$ *n*) *g*
proof (*induction n rule*:*nat-less-induct*)
case (*1 n*)

let *?P* = *poly-ring* (*ring-of R*)
interpret *field ring-of R*
using *assms*(*1*) **unfolding** *field*$_C$*-def* **by** *simp*
interpret *d-poly-ring*: *domain poly-ring* (*ring-of R*)
by (*rule univ-poly-is-domain*[*OF carrier-is-subring*])

have *ring-c*: *ring*$_C$ *R* **using** *assms*(*1*) **unfolding** *field*$_C$*-def domain*$_C$*-def cring*$_C$*-def* **by** *auto*
have *d-poly*: *domain*$_C$ (*poly R*) **using** *assms* (*1*) **unfolding** *field*$_C$*-def*
by (*intro poly-domain*) *auto*

have *ind*: *pmod-pow*$_C$ *R f m g* = *pmod* (*f* [^]$_{?P}$ *m*) *g* **if** *m* < *n* **for** *m*
using *1 that* **by** *auto*

define *r* **where** *r* = (*if n* ≥ *2 then pmod-pow*$_C$ *R f* (*n div 2*) *g* ^$_{C\ poly\ R}$ *2 else 1*$_{C\ poly\ R}$)

have *pmod r g* = *pmod* (*f* [^]$_{?P}$ (*n* − (*n mod 2*))) *g* ∧ *r* ∈ *carrier ?P*
proof (*cases n* ≥ *2*)
case *True*
hence *r* = *pmod-pow*$_C$ *R f* (*n div 2*) *g* [^]$_{?P}$ (*2* :: *nat*)
unfolding *r-def domain-cD*[*OF d-poly*] **by** (*simp add*:*ring-of-poly*[*OF ring-c*])
also have ... = *pmod* (*f* [^]$_{?P}$ (*n div 2*)) *g* [^]$_{?P}$ (*2* :: *nat*)
using *True* **by** (*intro arg-cong2*[**where** *f*=([^]$_{?P}$)] *refl ind*) *auto*
finally have *r-alt*: *r* = *pmod* (*f* [^]$_{?P}$ (*n div 2*)) *g* [^]$_{?P}$ (*2* :: *nat*)
by *simp*

have *pmod r g* = *pmod* (*pmod* (*f* [^]$_{?P}$ (*n div 2*)) *g* ⊗$_{?P}$ *pmod* (*f* [^]$_{?P}$ (*n div 2*)) *g*) *g*
unfolding *r-alt* **using** *assms*(*2*,*3*) *long-division-closed*[*OF carrier-is-subfield*]
by (*simp add*:*numeral-eq-Suc*) *algebra*
also have ... = *pmod* (*f* [^]$_{?P}$ (*n div 2*) ⊗$_{?P}$ *f* [^]$_{?P}$ (*n div 2*)) *g*
using *assms*(*2*,*3*) **by** (*intro pmod-mult-both*[*symmetric*]) *auto*
also have ... = *pmod* (*f* [^]$_{?P}$ ((*n div 2*)+(*n div 2*))) *g*
using *assms*(*2*,*3*) **by** (*subst d-poly-ring.nat-pow-mult*) *auto*
also have ... = *pmod* (*f* [^]$_{?P}$ (*n* − (*n mod 2*))) *g*
by (*intro arg-cong2*[**where** *f*=*pmod*] *refl arg-cong2*[**where** *f*=([^]$_{?P}$)]) *presburger*
finally have *pmod r g* = *pmod* (*f* [^]$_{?P}$ (*n* − (*n mod 2*))) *g*

by *simp*
moreover have $r \in$ *carrier ?P*
using *assms*(*2*,*3*) *long-division-closed*[*OF carrier-is-subfield*] **unfolding** *r-alt* **by** *auto*
ultimately show *?thesis* **by** *auto*
next
case *False*
hence $r = \mathbf{1}_{?P}$
unfolding *r-def* **using** *domain-cD*[*OF d-poly*] *ring-of-poly*[*OF ring-c*] **by** *simp*
also have ... $= f\ [\widehat{\ }]_{?P}\ (0 :: nat)$ **by** *simp*
also have ... $= f\ [\widehat{\ }]_{?P}\ (n - (n\ mod\ 2))$
using *False* **by** (*intro arg-cong2*[**where** f=([$\widehat{\ }$]$_{?P}$)] *refl*) *auto*
finally have $r = f\ [\widehat{\ }]_{?P}\ (n - (n\ mod\ 2))$ **by** *simp*
then show *?thesis* **using** *assms*(*2*) **by** *simp*
qed

hence *r-exp*: *pmod r g* = *pmod* ($f\ [\widehat{\ }]_{?P}\ (n - (n\ mod\ 2))$) *g* **and** *r-carr*: $r \in$ *carrier ?P*
by *auto*

have *pmod-pow*$_C$ *R f n g* = *pmod*$_C$ *R* ($r *_{C\,poly\,R}$ ($f\ \widehat{\ }_{C\,poly\,R}$ (n *mod 2*))) *g*
by (*subst pmod-pow*$_C$*.simps*) (*simp add*:*r-def*[*symmetric*])
also have ... = *pmod*$_C$ *R* ($r \otimes_{?P}$ ($f\ [\widehat{\ }]_{?P}$ (n *mod 2*))) *g*
unfolding *domain-cD*[*OF d-poly*] **by** (*simp add*:*ring-of-poly*[*OF ring-c*])
also have ... = *pmod* ($r \otimes_{?P}$ ($f\ [\widehat{\ }]_{?P}$ (n *mod 2*))) *g*
using *r-carr assms*(*2*,*3*) **by** (*intro pmod-c*[*OF assms*(*1*)]) *auto*
also have ... = *pmod* (*pmod r g* $\otimes_{?P}$ ($f\ [\widehat{\ }]_{?P}$ (n *mod 2*))) *g*
using *r-carr assms*(*2*,*3*) **by** (*intro pmod-mult-left*) *auto*
also have ... = *pmod* ($f\ [\widehat{\ }]_{?P}\ (n - (n\ mod\ 2)) \otimes_{?P}$ ($f\ [\widehat{\ }]_{?P}$ (n *mod 2*))) *g*
using *assms*(*2*,*3*) **unfolding** *r-exp* **by** (*intro pmod-mult-left*[*symmetric*]) *auto*
also have ... = *pmod* ($f\ [\widehat{\ }]_{?P}\ ((n - (n\ mod\ 2)) + (n\ mod\ 2))$) *g*
using *assms*(*2*,*3*) **by** (*intro arg-cong2*[**where** *f*=*pmod*] *refl d-poly-ring.nat-pow-mult*) *auto*
also have ... = *pmod* ($f\ [\widehat{\ }]_{?P}\ n$) *g* **by** *simp*
finally show *pmod-pow*$_C$ *R f n g* = *pmod* ($f\ [\widehat{\ }]_{?P}\ n$) *g* **by** *simp*
qed

The following function checks whether a given polynomial is coprime with the Gauss polynomial $X^n - X$.

definition *pcoprime-with-gauss-poly* :: ($'a$,$'b$) *idx-ring-scheme* $\Rightarrow$ $'a$ *list* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where *pcoprime-with-gauss-poly F p n* =
(*pcoprime*$_C$ *F p* (*pmod-pow*$_C$ *F* $X_{C\,F}$ *n p* $+_{C\,poly\,F}$ ($-_{C\,poly\,F}$ *pmod*$_C$ *F* $X_{C\,F}$ *p*)))

definition *divides-gauss-poly* :: ('a,'b) *idx-ring-scheme* ⇒ 'a *list* ⇒ *nat* ⇒ *bool*
where *divides-gauss-poly F p n* =
(*pmod-pow*$_C$ *F X*$_{C\,F}$ *n p* +$_{C\,poly\,F}$ (−$_{C\,poly\,F}$ *pmod*$_C$ *F X*$_{C\,F}$ *p*) = [])

lemma *mod-gauss-poly*:
assumes *field*$_C$ *R*
assumes *f* ∈ *carrier* (*poly-ring* (*ring-of R*))
shows *pmod-pow*$_C$ *R X*$_{C\,R}$ *n f* +$_{C\,poly\,R}$ (−$_{C\,poly\,R}$ *pmod*$_C$ *R X*$_{C\,R}$ *f*) =
ring.pmod (*ring-of R*) (*gauss-poly* (*ring-of R*) *n*) *f* (**is** *?L* = *?R*)
proof −
interpret *field ring-of R*
using *assms*(*1*) **unfolding** *field*$_C$*-def* **by** *simp*
interpret *d-poly-ring*: *domain poly-ring* (*ring-of R*)
by (*rule univ-poly-is-domain*[*OF carrier-is-subring*])

have *ring-c*: *ring*$_C$ *R* **using** *assms*(*1*) **unfolding** *field*$_C$*-def domain*$_C$*-def cring*$_C$*-def* **by** *auto*
have *d-poly*: *domain*$_C$ (*poly R*) **using** *assms* (*1*) **unfolding** *field*$_C$*-def* **by** (*intro poly-domain*) *auto*
let *?P* = *poly-ring* (*ring-of R*)

have *?L* = *pmod-pow*$_C$ *R X*$_{ring\text{-}of\,R}$ *n f* ⊕$_{?P}$ −$_{C\,poly\,R}$ *pmod*$_C$ *R X*$_{ring\text{-}of\,R}$ *f*
by (*simp add*: *poly-var domain-cD*[*OF d-poly*] *ring-of-poly*[*OF ring-c*])
also have ...= *pmod* (*X*$_{ring\text{-}of\,R}$[^]$_{?P}$ *n*) *f*⊖$_{?P}$ −$_{C\,poly\,R}$ *pmod X*$_{ring\text{-}of\,R}$ *f*
using *assms var-carr*[*OF carrier-is-subring*] **by** (*intro refl arg-cong2*[**where** *f*=(⊕$_{?P}$)]
pmod-pow-c arg-cong[**where** *f*=λ*x*. (−$_{C\,poly\,R}$ *x*)] *pmod-c*) *auto*
also have ... =*pmod* (*X*$_{ring\text{-}of\,R}$[^]$_{?P}$ *n*) *f*⊖$_{?P}$ *pmod X*$_{ring\text{-}of\,R}$ *f*
unfolding *a-minus-def* **using** *assms*(*1*,*2*) *var-carr*[*OF carrier-is-subring*]
ring-of-poly[*OF ring-c*] *long-division-closed*[*OF carrier-is-subfield*]
by (*subst domain-cD*[*OF d-poly*]) *auto*
also have ... = *pmod* (*X*$_{ring\text{-}of\,R}$[^]$_{?P}$ *n*) *f* ⊕$_{?P}$ *pmod* (⊖$_{?P}$ *X*$_{ring\text{-}of\,R}$) *f*
using *assms*(*2*) *var-carr*[*OF carrier-is-subring*]
unfolding *a-minus-def* **by** (*subst long-division-a-inv*[*OF carrier-is-subfield*]) *auto*
also have ... = *pmod* (*gauss-poly* (*ring-of R*) *n*) *f*
using *assms*(*2*) *var-carr*[*OF carrier-is-subring*] *var-pow-carr*[*OF carrier-is-subring*]
unfolding *gauss-poly-def a-minus-def* **by** (*subst long-division-add*[*OF carrier-is-subfield*]) *auto*

finally show *?thesis* **by** *simp*
qed

lemma *pcoprime-with-gauss-poly*:
assumes $field_C$ *R*
assumes $f \in carrier\ (poly\text{-}ring\ (ring\text{-}of\ R))$
shows *pcoprime-with-gauss-poly R f n* ⟷ $pcoprime_{ring\text{-}of\ R}$ (*gauss-poly* (*ring-of R*) *n*) *f*
(**is** *?L* = *?R*)
proof −
interpret *field ring-of R*
using *assms*(*1*) **unfolding** $field_C$*-def* **by** *simp*

have *?L* ⟷ $pcoprime_C$ *R f* (*pmod* (*gauss-poly* (*ring-of R*) *n*) *f*)
unfolding *pcoprime-with-gauss-poly-def* **using** *assms* **by** (*subst mod-gauss-poly*) *auto*
also have ... = $pcoprime_{ring\text{-}of\ R}$ *f* (*pmod* (*gauss-poly* (*ring-of R*) *n*) *f*)
using *assms gauss-poly-carr long-division-closed*[*OF carrier-is-subfield*]
by (*intro pcoprime-c*) *auto*
also have ... = $pcoprime_{ring\text{-}of\ R}$ (*gauss-poly* (*ring-of R*) *n*) *f*
by (*intro pcoprime-step*[*symmetric*] *gauss-poly-carr assms*)
finally show *?thesis* **by** *simp*
qed

lemma *divides-gauss-poly*:
assumes $field_C$ *R*
assumes $f \in carrier\ (poly\text{-}ring\ (ring\text{-}of\ R))$
shows *divides-gauss-poly R f n* ⟷ *f* $pdivides_{ring\text{-}of\ R}$ (*gauss-poly* (*ring-of R*) *n*)
(**is** *?L* = *?R*)
proof −
interpret *field ring-of R*
using *assms*(*1*) **unfolding** $field_C$*-def* **by** *simp*
have *?L* ⟷ (*pmod* (*gauss-poly* (*ring-of R*) *n*) *f* = [])
unfolding *divides-gauss-poly-def* **using** *assms* **by** (*subst mod-gauss-poly*) *auto*
also have ... ⟷ *?R*
using *assms gauss-poly-carr* **by** (*intro pmod-zero-iff-pdivides*[*OF carrier-is-subfield*]) *auto*
finally show *?thesis*
by *simp*
qed

fun *rabin-test-powers* :: (*'a*, *'b*) *idx-ring-enum-scheme* ⇒ *nat* ⇒ *nat list*
where *rabin-test-powers F n* =
map (λ*p*. *idx-size F*^(*n div p*)) (*filter* (λ*p*. *prime p* ∧ *p dvd n*)

[2..<(n+1)])

Given a monic polynomial with coefficients over a finite field returns true, if it is irreducible

fun *rabin-test* :: ('a, 'b) *idx-ring-enum-scheme* ⇒ 'a *list* ⇒ *bool*
  where *rabin-test F f* = (
    if degree f = *0 then*
      False
    else (*if* ¬*divides-gauss-poly F f* (*idx-size F*^*degree f*) *then*
      False
     else (*list-all* (*pcoprime-with-gauss-poly F f*) (*rabin-test-powers F* (*degree f*)))))

declare *rabin-test.simps*[*simp del*]

context
  fixes *R*
  assumes *field-R*: $field_C$ *R*
  assumes *enum-R*: $enum_C$ *R*
begin

interpretation *finite-field* (*ring-of R*)
  using *field-R enum-cD*[*OF enum-R*] **unfolding** $field_C$*-def*
  by (*simp add:finite-field-def finite-field-axioms-def*)

lemma *rabin-test-powers*:
  assumes *n* > *0*
  shows *set* (*rabin-test-powers R n*) =
    {*order* (*ring-of R*)^(*n div p*) | *p* . *Factorial-Ring.prime p* ∧ *p dvd n*}
    (**is** *?L* = *?R*)
proof −
  let *?f* = (λ*x*. *order* (*ring-of R*) ^(*n div x*))

  have *0*:*p* ∈ {*2..n*} **if** *Factorial-Ring.prime p p dvd n* **for** *p*
    using *assms that* **by** (*simp add*: *dvd-imp-le prime-ge-2-nat*)

  have *?L* = *?f* ‘ {*p* ∈ {*2..n*}. *Factorial-Ring.prime p* ∧ *p dvd n*}
    using *enum-cD*[*OF enum-R*] **by** *auto*
  also have ... = *?f* ‘ {*p*. *Factorial-Ring.prime p* ∧ *p dvd n*}
    using *0* **by** (*intro image-cong Collect-cong*) *auto*
  also have ... = *?R*
    by *auto*
  finally show *?thesis* **by** *simp*
qed

lemma *rabin-test*:
  assumes *monic-poly* (*ring-of R*) *f*
  shows *rabin-test R f* ⟷ *monic-irreducible-poly* (*ring-of R*) *f* (**is**

?L = ?R)
proof (*cases degree f = 0*)
case *True*
thus *?thesis* **unfolding** *rabin-test.simps* **using** *monic-poly-min-degree*
by *fastforce*
next
case *False*
define *N* **where** *N* = {*degree f div p* | *p* . *Factorial-Ring.prime p* ∧
p dvd degree f}

have *f-carr*: *f* ∈ *carrier* (*poly-ring* (*ring-of R*))
using *assms*(*1*) **unfolding** *monic-poly-def* **by** *auto*

have *deg-f-gt-0*: *degree f* > *0*
using *False* **by** *auto*
have *rt-powers*: *set* (*rabin-test-powers R* (*degree f*)) = (λ*x*. *order*
(*ring-of R*)^*x*) ' *N*
unfolding *rabin-test-powers*[*OF deg-f-gt-0*] *N-def* **by** *auto*

have *?L* ⟷ *divides-gauss-poly R f* (*idx-size R* ^ *degree f*) ∧
(∀ *n* ∈ *set* (*rabin-test-powers R* (*degree f*)). (*pcoprime-with-gauss-poly*
R f n))
using *False* **by** (*simp add*: *list-all-def rabin-test.simps del*:*rabin-test-powers.simps*)
also have ... ⟷ *f* $pdivides_{ring\text{-}of\ R}$ (*gauss-poly* (*ring-of R*) (*order*
(*ring-of R*) ^ *degree f*))
∧ (∀ *n* ∈ *N*. $pcoprime_{ring\text{-}of\ R}$ (*gauss-poly* (*ring-of R*) (*order* (*ring-of*
R) ^*n*)) *f*)
unfolding *divides-gauss-poly*[*OF field-R f-carr*] *pcoprime-with-gauss-poly*[*OF*
field-R f-carr]
rt-powers enum-cD[*OF enum-R*] **by** *simp*
also have ... ⟷ *?R*
using *False* **unfolding** *N-def* **by** (*intro rabin-irreducibility-condition*[*symmetric*]
assms(*1*)) *auto*
finally show *?thesis* **by** *simp*
qed

end

end

13 Additional results about Bijections and Digit Representations

theory *Finite-Fields-More-Bijections*
imports *HOL−Library.FuncSet Digit-Expansions.Bits-Digits*
begin

lemma *nth-digit-0*:

```
  assumes x < b^k
  shows nth-digit x k b = 0
  using assms unfolding nth-digit-def by auto

lemma nth-digit-bounded':
  assumes b > 0
  shows nth-digit v x b < b
  using assms by (simp add: nth-digit-def)

lemma digit-gen-sum-repr':
  assumes n < b^c
  shows n = (∑ k<c. nth-digit n k b * b ^ k)
proof -
  consider (a) b = 0 c = 0 | (b) b = 0 c > 0 | (c) b = 1 | (d) b>1
by linarith
  thus ?thesis
  proof (cases)
    case a thus ?thesis using assms by simp
  next
    case b thus ?thesis using assms by (simp add: zero-power)
  next
    case c thus ?thesis using assms by (simp add:nth-digit-def)
  next
    case d thus ?thesis by (intro digit-gen-sum-repr assms d)
  qed
qed

lemma
  assumes ⋀x. x ∈ A ⟹ f (g x) = x
  shows ⋀y. y ∈ g ` A ⟹ g (f y) = y
proof -
  show g (f y) = y if 0:y∈ g`A for y
  proof -
    obtain x where x-dom: x ∈ A and y-def: y = g x  using 0 by
auto
    hence g (f y) = g (f (g x)) by simp
     also have ... = g x by (intro arg-cong[where f=g] assms(1)
x-dom)
    also have ... = y unfolding y-def by simp
    finally show ?thesis by simp
  qed
qed

lemma nth-digit-bij:
  bij-betw (λv. (λx∈{..<n}. nth-digit v x b)) {..<b^n} ({..<n} →E
{..<b})
  (is bij-betw ?f ?A ?B)
proof -
  have inj-f: inj-on ?f ?A
```

using *digit-gen-sum-repr′* **by** (*intro inj-on-inverseI*[**where** *g*=(*λx*. ($\sum$ *k*<*n*. *x k* ∗ *b*^*k*))]) *auto*

consider (*a*) *b* = *0* *n*= *0* | (*b*) *b* = *0* *n*>*0* | (*c*) *b* > *0* **by** *linarith*
hence *nth-digit x i b* ∈ {..<*b*} **if** *i* < *n* *x* < *b*^*n* **for** *i x*
proof (*cases*)
case *a* **then show** *?thesis* **using** *that* **by** *auto*
next
case *b* **thus** *?thesis* **using** *that* **by** (*simp add*:*zero-power*)
next
case *c* **thus** *?thesis* **using** *that* **by** (*simp add*:*nth-digit-def*)
qed
hence *?f x* ∈ *?B* **if** *x* ∈ *?A* **for** *x* **using** *that* **unfolding** *restrict-PiE-iff* **by** *auto*
hence *?f* ‘ *?A* = *?B*
using *card-image*[*OF inj-f*] **by** (*intro card-seteq finite-PiE image-subsetI*) (*auto simp*:*card-PiE*)
thus *?thesis* **using** *inj-f* **unfolding** *bij-betw-def* **by** *auto*
qed

lemma *nth-digit-sum*:
assumes ⋀*i*. *i* < *l* ⟹ *f i* < *b*
shows ⋀*k*. *k* < *l* ⟹ *nth-digit* ($\sum$ *i*< *l*. *f i* ∗ *b*^*i*) *k b* = *f k*
and ($\sum$ *i*<*l*. *f i* ∗ *b*^*i*) < *b*^*l*
proof −
define *n* **where** *n* = ($\sum$ *i*< *l*. *f i* ∗ *b*^*i*)

have *restrict f* {..<*l*} ∈ {..<*l*} →$_E$ {..<*b*} **using** *assms*(*1*) **by** *auto*
then obtain *m* **where** *a*:(*λx*∈{..<*l*}. *nth-digit m x b*) = *restrict f* {..<*l*} **and** *b*:*m* ∈ {..<*b*^*l*}
using *bij-betw-imp-surj-on*[*OF nth-digit-bij*[**where** *n*=*l* **and** *b*=*b*]]
by (*metis* (*no-types*, *lifting*) *image-iff*)

have *m* = ($\sum$ *i*< *l*. *nth-digit m i b* ∗ *b*^*i*)
using *b* **by** (*intro digit-gen-sum-repr′*) *auto*
also have ... = ($\sum$ *i*< *l*. *f i* ∗ *b*^*i*)
using *a* **by** (*intro sum.cong arg-cong2*[**where** *f*=(∗)] *refl*) (*metis restrict-apply′*)
also have ... = *n* **unfolding** *n-def* **by** *simp*
finally have *c*:*n* = *m* **by** *simp*
show ($\sum$ *i*<*l*. *f i* ∗ *b*^*i*) < *b*^*l* **unfolding** *n-def*[*symmetric*] *c* **using** *b* **by** *auto*
show *nth-digit* ($\sum$ *i*< *l*. *f i* ∗ *b*^*i*) *k b* = *f k* **if** *k* < *l* **for** *k*
proof −
have *nth-digit* ($\sum$ *i*< *l*. *f i* ∗ *b*^*i*) *k b* = *nth-digit m k b* **unfolding** *n-def*[*symmetric*] *c* **by** *simp*
also have ... = *f k* **using** *a that* **by** (*metis lessThan-iff restrict-apply′*)
finally show *?thesis* **by** *simp*

qed
qed

lemma *bij-betw-reindex*:
assumes *bij-betw f I J*
shows *bij-betw* $(\lambda x.\ \lambda i{\in}I.\ x\ (f\ i))\ (J \to_E S)\ (I \to_E S)$
proof (*rule bij-betwI*[**where** $g{=}(\lambda x.\ \lambda i{\in}J.\ x$ (*the-inv-into I f i*))])
have *0*:*bij-betw* (*the-inv-into I f*) *J I*
using *assms bij-betw-the-inv-into* **by** *auto*

show $(\lambda x.\ \lambda i{\in}I.\ x\ (f\ i)) \in (J \to_E S) \to I \to_E S$
using *bij-betw-apply*[*OF assms*] **by** *auto*
show $(\lambda x.\ \lambda i{\in}J.\ x$ (*the-inv-into I f i*)) $\in (I \to_E S) \to J \to_E S$
using *bij-betw-apply*[*OF 0*] **by** *auto*
show $(\lambda j{\in}J.\ (\lambda i{\in}I.\ x\ (f\ i))$ (*the-inv-into I f j*)) $= x$ **if** $x \in J \to_E S$ **for** x
proof −
have $(\lambda i{\in}I.\ x\ (f\ i))$ (*the-inv-into I f j*) $= x\ j$ **if** $j \in J$ **for** j
using *0 assms f-the-inv-into-f-bij-betw bij-betw-apply that* **by** *fastforce*
thus *?thesis* **using** *PiE-arb*[*OF that*] **by** *auto*
qed
show $(\lambda i{\in}I.\ (\lambda j{\in}J.\ y$ (*the-inv-into I f j*)) $(f\ i)) = y$ **if** $y \in I \to_E S$ **for** y
proof −
have $(\lambda j{\in}J.\ y$ (*the-inv-into I f j*)) $(f\ i) = y\ i$ **if** $i \in I$ **for** i
using *assms 0 that the-inv-into-f-f*[*OF bij-betw-imp-inj-on*[*OF assms*]] *bij-betw-apply* **by** *force*
thus *?thesis* **using** *PiE-arb*[*OF that*] **by** *auto*
qed
qed

lemma *lift-bij-betw*:
assumes *bij-betw f S T*
shows *bij-betw* $(\lambda x.\ \lambda i{\in}I.\ f\ (x\ i))\ (I \to_E S)\ (I \to_E T)$
proof −
let $?g$ = *the-inv-into S f*

have *bij-g*: *bij-betw ?g T S* **using** *bij-betw-the-inv-into*[*OF assms*] **by** *simp*
have *0*:$?g(f\ x){=}x$ **if** $x \in S$ **for** x **by** (*intro the-inv-into-f-f that bij-betw-imp-inj-on*[*OF assms*])
have *1*:$f(?g\ x){=}x$ **if** $x \in T$ **for** x **by** (*intro f-the-inv-into-f-bij-betw*[*OF assms*] *that*)

have $(\lambda i{\in}I.\ f\ (x\ i)) \in I \to_E T$ **if** $x \in (I \to_E S)$ **for** x
using *bij-betw-apply*[*OF assms*] *that* **by** (*auto simp*: *Pi-def*)
moreover have $(\lambda i{\in}I.\ ?g\ (x\ i)) \in I \to_E S$ **if** $x \in (I \to_E T)$ **for** x
using *bij-betw-apply*[*OF bij-g*] *that* **by** (*auto simp*: *Pi-def*)

moreover have $(\lambda i{\in}I.\ ?g\ ((\lambda i{\in}I.\ f\ (x\ i))\ i)) = x$ **if** $x \in (I \rightarrow_E S)$ **for** x
proof −
have $(\lambda i{\in}I.\ ?g\ ((\lambda i{\in}I.\ f\ (x\ i))\ i))\ i = x\ i$ **for** i
using *PiE-mem*[*OF that*] **using** *PiE-arb*[*OF that*] **by** (*cases* $i \in I$) (*simp add*:*0*)+
thus *?thesis* **by** *auto*
qed
moreover have $(\lambda i{\in}I.\ f\ ((\lambda i{\in}I.\ ?g\ (x\ i))\ i)) = x$ **if** $x \in (I \rightarrow_E T)$ **for** x
proof −
have $(\lambda i{\in}I.\ f\ ((\lambda i{\in}I.\ ?g\ (x\ i))\ i))\ i = x\ i$ **for** i
using *PiE-mem*[*OF that*] **using** *PiE-arb*[*OF that*] **by** (*cases* $i \in I$) (*simp add*:*1*)+
thus *?thesis* **by** *auto*
qed
ultimately show *?thesis*
by (*intro bij-betwI*[**where** $g{=}(\lambda x.\ \lambda i{\in}I.\ ?g\ (x\ i))$]) *simp-all*
qed

lemma *lists-bij*:
bij-betw $(\lambda x.\ map\ x\ [\ 0..{<}d]\ )\ (\{..{<}d\} \rightarrow_E S)\ \{x.\ set\ x \subseteq S \wedge length\ x = d\}$
proof (*intro bij-betwI*[**where** $g{=}(\lambda x.\ \lambda i{\in}\{..{<}d\}.\ x\ !\ i)$] *funcsetI CollectI*, *goal-cases*)
case (*1 x*)
hence $x\ `\ \{0..{<}d\} \subseteq S$ **by** (*intro image-subsetI*) *auto*
thus *?case* **by** *simp*
next
case (*2 x*) **thus** *?case* **by** *auto*
next
case (*3 x*)
have *restrict* $((!)\ (map\ x\ [\ 0..{<}d]\ ))\ \{..{<}d\}\ j = x\ j$ **for** j
using *PiE-arb*[*OF 3*] **by** (*cases* $j \in \{..{<}d\}$) *auto*
thus *?case* **by** *auto*
next
case (*4 y*)
have $map\ (restrict\ ((!)\ y)\ \{..{<}d\})\ [\ 0..{<}d\ ] = map\ (((!)\ y))\ [\ 0..{<}d]$ **by** (*intro map-cong*) *auto*
also have ... $= y$ **using** *4 map-nth* **by** *blast*
finally show *?case* **by** *auto*
qed

lemma *bij-betw-prod*: *bij-betw* $(\lambda x.\ (x\ mod\ s,\ x\ div\ s))\ \{..{<}s * t\}\ (\{..{<}(s{::}nat)\} \times \{..{<}t\})$
proof −
have *bij-betw-aux*: $x + s * y < s * t$ **if** $x < s$ $y < t$ **for** $x\ y :: nat$
proof −
have $x + s * y < s + s * y$ **using** *that* **by** *simp*

also have ... = *s* ∗ (*y*+*1*) **by** *simp*
also have ... ≤ *s* ∗ *t* **using** *that* **by** (*intro mult-left-mono*) *auto*
finally show *?thesis* **by** *simp*
qed

show *?thesis*
proof (*cases s* > *0* ∧ *t* > *0*)
case *True*
then show *?thesis* **using** *less-mult-imp-div-less bij-betw-aux*
by (*intro bij-betwI*[**where** *g*=(λ*x*. *fst x* + *s* ∗ *snd x*)]) (*auto simp*:*mult.commute*)
next
case *False* **then show** *?thesis* **by** (*auto simp*:*bij-betw-def*)
qed
qed

end

14 Additional results about PMFs

theory *Finite-Fields-More-PMF*
imports *HOL*−*Probability.Probability-Mass-Function*
begin

lemma *powr-mono-rev*:
fixes *x* :: *real*
assumes *a* ≤ *b* **and** *x* > *0 x* ≤ *1*
shows *x powr b* ≤ *x powr a*
proof −
have *x powr b* = (*1*/*x*) *powr* (−*b*) **using** *assms* **by** (*simp add*: *powr-divide powr-minus-divide*)
also have ... ≤ (*1*/*x*) *powr* (−*a*) **using** *assms* **by** (*intro powr-mono*) *auto*
also have ... = *x powr a* **using** *assms* **by** (*simp add*: *powr-divide powr-minus-divide*)
finally show *?thesis* **by** *simp*
qed

lemma *integral-bind-pmf*:
fixes *f* :: - ⇒ *real*
assumes *bounded* (*f* ' *set-pmf* (*bind-pmf p q*))
shows (∫ *x*. *f x* ∂*bind-pmf p q*) = (∫ *x*. ∫ *y*. *f y* ∂*q x* ∂*p*) (**is** *?L* = *?R*)
proof −
obtain *M* **where** *a*:|*f x*| ≤ *M* **if** *x* ∈ *set-pmf* (*bind-pmf p q*) **for** *x*
using *assms*(*1*) **unfolding** *bounded-iff* **by** *auto*
define *clamp* **where** *clamp x* = (*if* |*x*| > *M then 0 else x*) **for** *x*

obtain *x* **where** *x* ∈ *set-pmf* (*bind-pmf p q*) **using** *set-pmf-not-empty*

by *fast*
hence *M-ge-0*: $M \geq 0$ **using** *a* **by** *fastforce*

have $a:\bigwedge x\ y.\ x \in set\text{-}pmf\ p \Longrightarrow y \in set\text{-}pmf\ (q\ x) \Longrightarrow \neg|f\ y| > M$
using *a* **by** *fastforce*
hence $(\int x.\ f\ x\ \partial bind\text{-}pmf\ p\ q) = (\int x.\ clamp\ (f\ x)\ \partial bind\text{-}pmf\ p\ q)$
unfolding *clamp-def* **by** (*intro integral-cong-AE AE-pmfI*) *auto*
also have ... = $(\int x.\ \int y.\ clamp\ (f\ y)\ \partial q\ x\ \partial p)$ **unfolding** *measure-pmf-bind*
by (*subst integral-bind*[**where** *K=count-space UNIV* **and** *B′=1* **and** *B=M*])
(*simp-all add:measure-subprob clamp-def M-ge-0*)
also have ... = *?R* **unfolding** *clamp-def* **using** *a* **by** (*intro integral-cong-AE AE-pmfI*) *simp-all*
finally show *?thesis* **by** *simp*
qed

lemma *measure-bind-pmf*:
$measure\ (bind\text{-}pmf\ m\ f)\ s = (\int x.\ measure\ (f\ x)\ s\ \partial m)$ (**is** *?L = ?R*)
proof −
have $?L = (\int x.\ indicator\ s\ x\ \partial bind\text{-}pmf\ m\ f)$ **by** *simp*
also have ... = $(\int x.\ (\int y.\ indicator\ s\ y\ \partial f\ x)\ \partial m)$
by (*intro integral-bind-pmf*) (*auto intro*!:*boundedI*)
also have ... = *?R* **by** *simp*
finally show *?thesis* **by** *simp*
qed

end

15 Executable Polynomial Factor Rings

theory *Finite-Fields-Poly-Factor-Ring-Code*
imports
Finite-Fields-Poly-Ring-Code
Rabin-Irreducibility-Test-Code
Finite-Fields-More-Bijections
begin

Enumeration of the polynomials with a given degree:

definition *poly-enum* :: (′*a*,′*b*) *idx-ring-enum-scheme* ⇒ *nat* ⇒ *nat* ⇒ ′*a list*
where *poly-enum R l n* =
dropWhile ((=) $0_{C\,R}$) (*map* (λ*p*. *idx-enum R* (*nth-digit n* (*l*−*1*−*p*) (*idx-size R*))) [*0*..<*l*])

lemma *replicate-drop-while-cancel*:
assumes *k* = *length* (*takeWhile* ((=) *x*) *y*)
shows *replicate k x* @ *dropWhile* ((=) *x*) *y* = *y* (**is** *?L = ?R*)
proof −

have *replicate k x* = *takeWhile* ((=) *x*) *y*
using *assms* **by** (*metis* (*full-types*) *replicate-length-same set-takeWhileD*)
thus *?thesis* **by** *simp*
qed

lemma *arg-cong3*:
assumes $x = u$ $y = v$ $z = w$
shows $f\ x\ y\ z = f\ u\ v\ w$
using *assms* **by** *simp*

lemma *list-all-dropwhile*: *list-all p xs* ⟹ *list-all p* (*dropWhile q xs*)
by (*induction xs*) *auto*

lemma *bij-betw-poly-enum*:
assumes $enum_C$ *R* $ring_C$ *R*
shows *bij-betw* (*poly-enum R l*) {..<*idx-size R*^*l*}
{*xs*. *xs* ∈ *carrier* (*poly-ring* (*ring-of R*)) ∧ *length xs* ≤ *l*}
proof −
let *?b* = *idx-size R*
let *?S0* = {..<*l*} $\rightarrow_E$ {..<*order* (*ring-of R*)}
let *?S1* = {..<*l*} $\rightarrow_E$ {*x*. *idx-pred R x*}
let *?S2* = {*xs*. *list-all* (*idx-pred R*) *xs* ∧ *length xs* = *l*}
let *?S3* = {*xs*. (*xs* = [] ∨ *hd xs* ≠ $0_{C\,R}$) ∧ *list-all* (*idx-pred R*) *xs* ∧ *length xs* ≤ *l*}
let *?S4* = {*xs*. *xs* ∈ *carrier* (*poly-ring* (*ring-of R*)) ∧ *length xs* ≤ *l*}

interpret *ring ring-of R* **using** *assms*(*2*) **unfolding** $ring_C$*-def* **by** *simp*

have *0* < *order* (*ring-of R*) **using** *enum-cD*(*1*)[*OF assms*(*1*)] *order-gt-0-iff-finite* **by** *metis*
also have ... = *?b* **using** *enum-cD*[*OF assms*(*1*)] **by** *auto*
finally have *b-gt-0*: *?b* > *0* **by** *simp*

note *bij0* = *lift-bij-betw*[*OF enum-cD*(*3*)[*OF assms*(*1*)], **where** *I*={..<*l*}]
note *bij1* = *lists-bij*[**where** *d*=*l* **and** *S*={*x*. *idx-pred R x*}]

have *bij-betw* (*dropWhile* ((=) $0_{C\,R}$)) *?S2 ?S3*
proof (*rule bij-betwI*[**where** *g*=λ*xs*. *replicate* (*l* − *length xs*) $0_{C\,R}$ @ *xs*])
have *dropWhile* ((=) $0_{C\,R}$) *xs* ∈ *?S3* **if** *xs* ∈ *?S2* **for** *xs*
proof −
have *dropWhile* ((=) $0_{C\,R}$) *xs* = [] ∨ *hd* (*dropWhile* ((=) $0_{C\,R}$) *xs*) ≠ $0_{C\,R}$
using *hd-dropWhile* **by** (*metis* (*full-types*))
moreover have *length* (*dropWhile* ((=) $0_{C\,R}$) *xs*) ≤ *l*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq length-dropWhile-le that*)
ultimately show *?thesis* **using** *that* **by** (*auto simp*:*list-all-dropwhile*)

qed
thus *dropWhile* ((=) $0_{C\,R}$) ∈ *?S2* → *?S3* **by** *auto*
have *replicate* (*l* − *length xs*) $0_{C\,R}$ @ *xs* ∈ *?S2* **if** *xs* ∈ *?S3* **for** *xs*
proof −
have *idx-pred R* $0_{C\,R}$ **using** *add.one-closed* **by** (*simp add*:*ring-of-def*)
moreover have *length* (*replicate* (*l* − *length xs*) $0_{C\,R}$ @ *xs*) = *l*
using *that* **by** *auto*
ultimately show *?thesis* **using** *that* **by** (*auto simp*:*list-all-iff*)
qed
thus (λ*xs*. *replicate* (*l* − *length xs*) $0_{C\,R}$ @ *xs*) ∈ *?S3* → *?S2* **by**
auto

show *replicate* (*l* − *length* (*dropWhile* ((=) $0_{C\,R}$) *x*)) $0_{C\,R}$ @
dropWhile ((=) $0_{C\,R}$) *x* = *x*
if *x* ∈ *?S2* **for** *x*
proof −
have *length* (*takeWhile* ((=) $0_{C\,R}$) *x*) + *length* (*dropWhile* ((=)
$0_{C\,R}$) *x*) = *length x*
unfolding *length-append*[*symmetric*] **by** *simp*
thus *?thesis* **using** *that* **by** (*intro replicate-drop-while-cancel*)
auto
qed
show *dropWhile* ((=) $0_{C\,R}$) (*replicate* (*l* − *length y*) $0_{C\,R}$ @ *y*) =
y
if *y* ∈ *?S3* **for** *y*
proof −
have *dropWhile* ((=) $0_{C\,R}$) (*replicate* (*l* − *length y*) $0_{C\,R}$ @ *y*)
= *dropWhile* ((=) $0_{C\,R}$) *y*
by (*intro dropWhile-append2*) *simp*
also have ... = *y* **using** *that* **by** (*intro iffD2*[*OF dropWhile-eq-self-iff*])
auto
finally show *?thesis* **by** *simp*
qed
qed
moreover have *?S3* = *?S4*
unfolding *ring-of-poly*[*OF assms*(*2*),*symmetric*] **by** (*simp add*:*ring-of-def*
poly-def)
ultimately have *bij2*: *bij-betw* (*dropWhile* ((=) $0_{C\,R}$)) *?S2 ?S4* **by**
simp

have *bij3*: *bij-betw* (λ*x*. *l*−*1*−*x*) {..<*l*} {..<*l*}
by (*intro bij-betwI*[**where** *g*=λ*x*. *l*−*1*−*x*]) *auto*
note *bij4* = *bij-betw-reindex*[*OF bij3*, **where** *S*={..<*order* (*ring-of*
R)}]
have *bij5*: *bij-betw* (λ*n*. (λ*p*∈{..<*l*}. *nth-digit n p ?b*)) {..<*?b*^*l*} *?S0*
using *nth-digit-bij*[**where** *n*=*l*] *enum-cD*[*OF assms*(*1*)] **by** *simp*
have *bij6*: *bij-betw* (λ*n*. (λ*p*∈{..<*l*}. *nth-digit n* (*l*−*1*−*p*) *?b*)) {..<*?b*^*l*}
?S0
by (*intro iffD2*[*OF arg-cong3*[**where** *f*=*bij-betw*] *bij-betw-trans*[*OF*

bij5 bij4]]) *force+*

have *carrier* (*ring-of R*) = {*x*. *idx-pred R x*} **unfolding** *ring-of-def* **by** *auto*
hence *bij7*: *bij-betw* (λ*n*. (λ*p*∈{..<*l*}. *idx-enum R* (*nth-digit n* (*l*−*1*−*p*) *?b*))) {..<*?b*^*l*} *?S1*
by (*intro iffD2*[*OF arg-cong3*[**where** *f*=*bij-betw*] *bij-betw-trans*[*OF bij6 bij0*]]) *fastforce+*

have *bij8*: *bij-betw* (λ*n*. *map* (λ*p*. *idx-enum R* (*nth-digit n* (*l*−*1*−*p*) *?b*)) [*0*..<*l*]) {..<*?b*^*l*} *?S2*
by (*intro iffD2*[*OF arg-cong3*[**where** *f*=*bij-betw*] *bij-betw-trans*[*OF bij7 bij1*]])
(*auto simp*:*comp-def list-all-iff atLeast0LessThan*[*symmetric*])

thus *bij-betw* (*poly-enum R l*) {..<*idx-size R* ^ *l*} *?S4*
using *bij-betw-trans*[*OF bij8 bij2*] **unfolding** *poly-enum-def comp-def* **by** *simp*
qed

definition *poly-enum-inv* :: (′*a*,′*b*) *idx-ring-enum-scheme* ⇒ *nat* ⇒ ′*a list* ⇒ *nat*
where *poly-enum-inv R l f* =
(*let f*′ = *replicate* (*l* − *length f*) $0_{C R}$ @ *f in*
(∑ *i*<*l*. *idx-enum-inv R* (*f*′ ! (*l* − *1* − *i*)) ∗ *idx-size R* ^*i*))

find-theorems (∑ *i*<*?l*. *?f i* ∗ *?x*^*i*) < *?x*^*?l*

lemma *poly-enum-inv*:
assumes $enum_C$ *R* $ring_C$ *R*
assumes *x* ∈ {*xs*. *xs* ∈ *carrier* (*poly-ring* (*ring-of R*)) ∧ *length xs* ≤ *l*}
shows *the-inv-into* {..<*idx-size R*^*l*} (*poly-enum R l*) *x* = *poly-enum-inv R l x*
proof −
define *f* **where** *f* = *replicate* (*l*− *length x*) $0_{C R}$ @ *x*
let *?b* = *idx-size R*
let *?d* = *dropWhile* ((=) $0_{C R}$)

have *len-f*: *length f* = *l* **using** *assms*(*3*) **unfolding** *f-def* **by** *auto*
note *enum-c* = *enum-cD*[*OF assms*(*1*)]

interpret *ring ring-of R* **using** *assms*(*2*) **unfolding** $ring_C$*-def* **by** *simp*

have *0*: *idx-enum-inv R y* < *?b* **if** *y* ∈ *carrier* (*ring-of R*) **for** *y*
using *bij-betw-imp-surj-on*[*OF enum-c*(*4*)] *enum-c*(*2*) *that* **by** *auto*
have *1*: (*x* = [] ∨ *lead-coeff x* ≠ $0_{C R}$) ∧ *list-all* (*idx-pred R*) *x* ∧

length x $\leq$ *l*
using *assms(3)* **unfolding** *ring-of-poly[OF assms(2),symmetric]* **by** *(simp add:ring-of-def poly-def)*
moreover have $\mathbf{0}_{ring\text{-}of\ R} \in$ *carrier (ring-of R)* **by** *simp*
hence *idx-pred R* $0_{C\,R}$ **unfolding** *ring-of-def* **by** *simp*
ultimately have *2: set f* $\subseteq$ *carrier (ring-of R)*
unfolding *f-def* **by** *(auto simp add:ring-of-def list-all-iff)*

have *poly-enum R l(poly-enum-inv R l x)= poly-enum R l* ($\sum$ *i<l. idx-enum-inv R (f ! (l−1−i))∗?b^i*)
unfolding *poly-enum-inv-def f-def[symmetric]* **by** *simp*
also have ... = *?d (map (λp. idx-enum R (idx-enum-inv R (f ! (l − 1 − (l − 1 − p))))) [0..<l])*
unfolding *poly-enum-def* **using** *2 len-f* **by** *(intro arg-cong[***where** *f=?d]*
*arg-cong[***where** *f=idx-enum R] map-cong refl nth-digit-sum 0) auto*
also have ... =*?d (map (λp. (f ! (l−1 − (l−1−p)))) [0..<l])*
using *2 len-f* **by** *(intro arg-cong[***where** *f=?d] map-cong refl enum-c) auto*
also have ... =*?d (map (λp. (f ! p)) [0..<l])*
by *(intro arg-cong[***where** *f=?d] map-cong) auto*
also have ... = *?d f* **using** *len-f map-nth* **by** *(intro arg-cong[***where** *f=?d]) auto*
also have ... = *?d x* **unfolding** *f-def* **by** *(intro dropWhile-append2) auto*
also have ... = *x* **using** *1* **by** *(intro iffD2[OF dropWhile-eq-self-iff]) auto*
finally have *poly-enum R l (poly-enum-inv R l x) = x* **by** *simp*
moreover have *poly-enum-inv R l x < idx-size R^l*
unfolding *poly-enum-inv-def Let-def f-def[symmetric]* **using** *len-f 2*
by *(intro nth-digit-sum(2) 0) auto*
ultimately show *?thesis*
by *(intro the-inv-into-f-eq bij-betw-imp-inj-on[OF bij-betw-poly-enum[OF assms(1,2)]]) auto*
qed

definition *poly-mod-ring* :: *('a,'b) idx-ring-enum-scheme ⇒ 'a list => 'a list idx-ring-enum*
where *poly-mod-ring R f* = (|
idx-pred = *(λxs. idx-pred (poly R) xs ∧ length xs* $\leq$ *degree f)*,
idx-uminus = *idx-uminus (poly R)*,
idx-plus = *(λx y. pmod*$_C$ *R (x* $+_{C\,poly\ R}$ *y) f)*,
idx-udivide = *(λx. let ((u,v),r) = ext-euclidean R x f in pmod*$_C$ *R* $(r^{-1}{}_{C\,poly\ R} *_{C\,poly\ R}$ *u) f)*,
idx-mult = *(λx y. pmod*$_C$ *R (x* $*_{C\,poly\ R}$ *y) f)*,
idx-zero = $0_{C\,poly\ R}$,
idx-one = $1_{C\,poly\ R}$,

idx-size = *idx-size* R ^ *degree* f,
idx-enum = *poly-enum* R (*degree* f),
idx-enum-inv = *poly-enum-inv* R (*degree* f) ⦈

definition *poly-mod-ring-iso* :: ('a,'b) *idx-ring-enum-scheme* ⇒ 'a *list* ⇒ 'a *list* ⇒ 'a *list set*
where *poly-mod-ring-iso* $R\ f\ x = PIdl_{poly\text{-}ring\ (ring\text{-}of\ R)}\ f +>_{poly\text{-}ring\ (ring\text{-}of\ R)}\ x$

definition *poly-mod-ring-iso-inv* :: ('a,'b) *idx-ring-enum-scheme* ⇒ 'a *list* ⇒ 'a *list set* ⇒ 'a *list*
where *poly-mod-ring-iso-inv* R f =
the-inv-into (*carrier* (*ring-of* (*poly-mod-ring* R f))) (*poly-mod-ring-iso* R f)

context
fixes f
fixes R :: ('a,'b) *idx-ring-enum-scheme*
assumes *field-R*: $field_C$ R
assumes *f-carr*: $f \in$ *carrier* (*poly-ring* (*ring-of* R))
assumes *deg-f*: *degree* $f > 0$
begin

private abbreviation P **where** $P \equiv$ *poly-ring* (*ring-of* R)
private abbreviation I **where** $I \equiv PIdl_{poly\text{-}ring\ (ring\text{-}of\ R)}\ f$

interpretation *field ring-of* R
using *field-R* **unfolding** $field_C$*-def* **by** *auto*

interpretation *d*: *domain* P
by (*intro univ-poly-is-domain carrier-is-subring*)

interpretation *i*: *ideal* I P
using *f-carr* **by** (*intro d.cgenideal-ideal*) *auto*

interpretation *s*: *ring-hom-ring* P P *Quot* I $(+>_P)$ I
using *i.rcos-ring-hom-ring* **by** *auto*

interpretation *cr*: *cring* P *Quot* I
by (*intro i.quotient-is-cring d.cring-axioms*)

lemma *ring-c*: $ring_C$ R
using *field-R* **unfolding** $field_C$*-def* $domain_C$*-def* $cring_C$*-def* **by** *auto*

lemma *d-poly*: $domain_C$ (*poly* R) **using** *field-R* **unfolding** $field_C$*-def*
by (*intro poly-domain*) *auto*

lemma *ideal-mod*:
assumes $y \in$ *carrier* P

shows *I* +>$_P$ (*pmod y f*) = *I* +>$_P$ *y*
proof −
have *f* ∈ *I* **by** (*intro d.cgenideal-self f-carr*)
hence (*f* ⊗$_P$ (*pdiv y f*)) ∈ *I*
using *long-division-closed*[*OF carrier-is-subfield*] *assms f-carr*
by (*intro i.I-r-closed*) (*simp-all*)
hence *y* ∈ *I* +>$_P$ (*pmod y f*)
using *assms f-carr* **unfolding** *a-r-coset-def'*
by (*subst pdiv-pmod*[*OF carrier-is-subfield*, **where** *q*=*f*]) *auto*
thus *?thesis*
by (*intro i.a-repr-independence' assms long-division-closed*[*OF carrier-is-subfield*] *f-carr*)
qed

lemma *poly-mod-ring-carr-1*:
carrier (*ring-of* (*poly-mod-ring R f*)) = {*xs. xs* ∈ *carrier P* ∧ *degree xs* < *degree f*}
(**is** *?L* = *?R*)
proof −
have *?L* = {*xs. xs* ∈ *carrier* (*ring-of* (*poly R*)) ∧ *degree xs* < *degree f*}
using *deg-f* **unfolding** *poly-mod-ring-def ring-of-def* **by** *auto*
also have ... = *?R* **unfolding** *ring-of-poly*[*OF ring-c*] **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *poly-mod-ring-carr*:
assumes *y* ∈ *carrier P*
shows *pmod y f* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
proof −
have *f* ≠ [] **using** *deg-f* **by** *auto*
hence *pmod y f* = [] ∨ *degree* (*pmod y f*) < *degree f*
by (*intro pmod-degree*[*OF carrier-is-subfield*] *assms f-carr*)
hence *degree* (*pmod y f*) < *degree f* **using** *deg-f* **by** *auto*
moreover have *pmod y f* ∈ *carrier P*
using *f-carr assms long-division-closed*[*OF carrier-is-subfield*] **by** *auto*
ultimately show *?thesis* **unfolding** *poly-mod-ring-carr-1* **by** *auto*
qed

lemma *poly-mod-ring-iso-ran*:
poly-mod-ring-iso R f ‘ *carrier* (*ring-of* (*poly-mod-ring R f*)) = *carrier* (*P Quot I*)
proof −
have *poly-mod-ring-iso R f x* ∈ *carrier* (*P Quot I*)
if *x* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*)) **for** *x*
proof −
have *I* ⊆ *carrier P* **by** *auto*
moreover have *x* ∈ *carrier P* **using** *that* **unfolding** *poly-mod-ring-carr-1*

by *auto*
ultimately have *poly-mod-ring-iso R f x* ∈ *a-rcosets*$_P$ *I*
using *that f-carr* **unfolding** *poly-mod-ring-iso-def* **by** (*intro d.a-rcosetsI*) *auto*
thus *?thesis* **unfolding** *FactRing-def* **by** *simp*
qed
moreover have *x* ∈ *poly-mod-ring-iso R f* ‘ *carrier* (*ring-of* (*poly-mod-ring R f*))
if *x* ∈ *carrier* (*P Quot I*) **for** *x*
proof −
have *x* ∈ *a-rcosets*$_P$ *I* **using** *that* **unfolding** *FactRing-def* **by** *auto*
then obtain *y* **where** *y-def*: *x* = *I* +>$_P$ *y y* ∈ *carrier P*
using *that* **unfolding** *A-RCOSETS-def'* **by** *auto*
define *z* **where** *z* = *pmod y f*
have *I* +>$_P$ *z* = *I* +>$_P$ *y* **unfolding** *z-def* **by** (*intro ideal-mod y-def*)
hence *poly-mod-ring-iso R f z* = *x* **unfolding** *poly-mod-ring-iso-def y-def* **by** *simp*
moreover have *z* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
unfolding *z-def* **by** (*intro poly-mod-ring-carr y-def*)
ultimately show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *poly-mod-ring-iso-inj*:
inj-on (*poly-mod-ring-iso R f*) (*carrier* (*ring-of* (*poly-mod-ring R f*)))
proof (*rule inj-onI*)
fix *x y*
assume *x* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
hence *x*:*x* ∈ *carrier P degree x* < *degree f* **unfolding** *poly-mod-ring-carr-1* **by** *auto*
assume *y* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
hence *y*:*y* ∈ *carrier P degree y* < *degree f* **unfolding** *poly-mod-ring-carr-1* **by** *auto*

have *degree* (*x* ⊖$_P$ *y*) ≤ *max* (*degree x*) (*degree* (⊖$_P$*y*))
unfolding *a-minus-def* **by** (*intro degree-add*)
also have ... = *max* (*degree x*) (*degree y*)
unfolding *univ-poly-a-inv-degree*[*OF carrier-is-subring y*(*1*)] **by** *simp*
also have ... < *degree f* **using** *x*(*2*) *y*(*2*) **by** *simp*
finally have *d*:*degree* (*x* ⊖$_P$ *y*) < *degree f* **by** *simp*

assume *poly-mod-ring-iso R f x* = *poly-mod-ring-iso R f y*
hence *I* +>$_P$ *x* = *I* +>$_P$ *y* **unfolding** *poly-mod-ring-iso-def* **by** *simp*
hence *x* ⊖$_P$ *y* ∈ *I* **using** *x y* **by** (*subst d.quotient-eq-iff-same-a-r-cos*[*OF*

i.ideal-axioms]) *auto*
hence *f pdivides*$_{ring\text{-}of\ R}$ ($x \ominus_P y$)
using *f-carr x*(*1*) *y d.m-comm* **unfolding** *cgenideal-def pdivides-def factor-def* **by** *auto*
hence ($x \ominus_P y$) = [] ∨ *degree* ($x \ominus_P y$) ≥ *degree f*
using *x*(*1*) *y*(*1*) *f-carr pdivides-imp-degree-le*[*OF carrier-is-subring*]
by (*meson d.minus-closed*)
hence ($x \ominus_P y$) = $\mathbf{0}_P$ **unfolding** *univ-poly-zero* **using** *d* **by** *simp*
thus $x = y$ **using** *x*(*1*) *y*(*1*) **by** *simp*
qed

lemma *poly-mod-iso-ring-bij*:
bij-betw (*poly-mod-ring-iso R f*) (*carrier* (*ring-of* (*poly-mod-ring R f*))) (*carrier* (*P Quot I*))
using *poly-mod-ring-iso-ran poly-mod-ring-iso-inj* **unfolding** *bij-betw-def*
by *simp*

lemma *poly-mod-iso-ring-bij-2*:
bij-betw (*poly-mod-ring-iso-inv R f*) (*carrier* (*P Quot I*)) (*carrier* (*ring-of* (*poly-mod-ring R f*)))
unfolding *poly-mod-ring-iso-inv-def* **using** *poly-mod-iso-ring-bij bij-betw-the-inv-into*
by *blast*

lemma *poly-mod-ring-iso-inv-1*:
assumes $x \in$ *carrier* (*P Quot I*)
shows *poly-mod-ring-iso R f* (*poly-mod-ring-iso-inv R f x*) = *x*
unfolding *poly-mod-ring-iso-inv-def* **using** *assms poly-mod-iso-ring-bij*
by (*intro f-the-inv-into-f-bij-betw*) *auto*

lemma *poly-mod-ring-iso-inv-2*:
assumes $x \in$ *carrier* (*ring-of* (*poly-mod-ring R f*))
shows *poly-mod-ring-iso-inv R f* (*poly-mod-ring-iso R f x*) = *x*
unfolding *poly-mod-ring-iso-inv-def* **using** *assms*
by (*intro the-inv-into-f-f poly-mod-ring-iso-inj*)

lemma *poly-mod-ring-add*:
assumes $x \in$ *carrier P*
assumes $y \in$ *carrier P*
shows $x \oplus_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)} y$ = *pmod* ($x \oplus_P y$) *f* (**is** *?L* = *?R*)
proof −
have *?L* = $pmod_C$ *R* ($x \oplus_{ring\text{-}of\ (poly\ R)} y$) *f*
unfolding *poly-mod-ring-def ring-of-def* **using** *domain-cD*[*OF d-poly*] **by** *simp*
also have ... = *?R*
using *assms* **unfolding** *ring-of-poly*[*OF ring-c*] **by** (*intro pmod-c*[*OF field-R*] *f-carr*) *auto*
finally show *?thesis*
by *simp*

qed

lemma *poly-mod-ring-zero*: $\mathbf{0}_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)} = \mathbf{0}_P$
proof−
  have $\mathbf{0}_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)} = \mathbf{0}_{ring\text{-}of\ (poly\ R)}$
   using *domain-cD*[*OF d-poly*] **unfolding** *ring-of-def poly-mod-ring-def* **by** *simp*
  also have ... = $\mathbf{0}_P$ **unfolding** *ring-of-poly*[*OF ring-c*] **by** *simp*
  finally show *?thesis* **by** *simp*
qed

lemma *poly-mod-ring-one*: $\mathbf{1}_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)} = \mathbf{1}_P$
proof−
  have $\mathbf{1}_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)} = \mathbf{1}_{ring\text{-}of\ (poly\ R)}$
   using *domain-cD*[*OF d-poly*] **unfolding** *ring-of-def poly-mod-ring-def* **by** *simp*
  also have ... = $\mathbf{1}_P$ **unfolding** *ring-of-poly*[*OF ring-c*] **by** *simp*
  finally show $\mathbf{1}_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)} = \mathbf{1}_P$ **by** *simp*
qed

lemma *poly-mod-ring-mult*:
  assumes $x \in carrier\ P$
  assumes $y \in carrier\ P$
  shows $x \otimes_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)} y = pmod\ (x \otimes_P y)\ f$ (**is** *?L* = *?R*)
proof −
  have $?L = pmod_C\ R\ (x \otimes_{ring\text{-}of\ (poly\ R)} y)\ f$
   unfolding *poly-mod-ring-def ring-of-def* **using** *domain-cD*[*OF d-poly*] **by** *simp*
  also have ... = *?R*
   using *assms* **unfolding** *poly-mod-ring-carr-1 ring-of-poly*[*OF ring-c*]
    by (*intro pmod-c*[*OF field-R*] *f-carr*) *auto*
  finally show *?thesis*
    by *simp*
qed

lemma *poly-mod-ring-iso-inv*:
 poly-mod-ring-iso-inv R f ∈ *ring-iso* (*P Quot I*) (*ring-of* (*poly-mod-ring R f*))
  (**is** *?f* ∈ *ring-iso ?S ?T*)
proof (*rule ring-iso-memI*)
  fix *x* **assume** *x* ∈ *carrier ?S*
 thus *?f x* ∈ *carrier ?T* **using** *bij-betw-apply*[*OF poly-mod-iso-ring-bij-2*] **by** *auto*
next
  fix *x y* **assume** *x*:*x* ∈ *carrier ?S* **and** *y*: *y* ∈ *carrier ?S*
  have *?f x* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
    by (*rule bij-betw-apply*[*OF poly-mod-iso-ring-bij-2 x*])

hence x':*?f x* ∈ *carrier P* **unfolding** *poly-mod-ring-carr-1* **by** *simp*
have *?f y* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
by (*rule bij-betw-apply*[*OF poly-mod-iso-ring-bij-2 y*])
hence y':*?f y* ∈ *carrier P* **unfolding** *poly-mod-ring-carr-1* **by** *simp*

have *0*: *?f x* $\otimes_{?T}$ *?f y* = *pmod* (*?f x* $\otimes_P$ *?f y*) *f*
by (*intro poly-mod-ring-mult* x' y')
also have ... ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
using x' y' **by** (*intro poly-mod-ring-carr*) *auto*
finally have *xy*: *?f x* $\otimes_{?T}$ *?f y* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*)) **by** *simp*

have *?f* (*x* $\otimes_{?S}$ *y*) = *?f* (*poly-mod-ring-iso R f* (*?f x*) $\otimes_{?S}$ *poly-mod-ring-iso R f* (*?f y*))
using *x y* **by** (*simp add*:*poly-mod-ring-iso-inv-1*)
also have ... = *?f* ((*I* +>$_P$ (*?f x*)) $\otimes_{?S}$ (*I* +>$_P$ (*?f y*)))
unfolding *poly-mod-ring-iso-def* **by** *simp*
also have ... = *?f* (*I* +>$_P$ (*?f x* $\otimes_P$ *?f y*))
using x' y' **by** *simp*
also have ... = *?f* (*I* +>$_P$ (*pmod* (*?f x* $\otimes_P$ *?f y*) *f*))
using x' y' **by** (*subst ideal-mod*) *auto*
also have ... = *?f* (*I* +>$_P$ (*?f x* $\otimes_{?T}$ *?f y*))
unfolding *0* **by** *simp*
also have ... = *?f* (*poly-mod-ring-iso R f* (*?f x* $\otimes_{?T}$ *?f y*))
unfolding *poly-mod-ring-iso-def* **by** *simp*
also have ... = *?f x* $\otimes_{?T}$ *?f y*
using *xy* **by** (*intro poly-mod-ring-iso-inv-2*)
finally show *?f* (*x* $\otimes_{?S}$ *y*) = *?f x* $\otimes_{?T}$ *?f y* **by** *simp*
next
fix *x y* **assume** *x*:*x* ∈ *carrier ?S* **and** *y*: *y* ∈ *carrier ?S*
have *?f x* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
by (*rule bij-betw-apply*[*OF poly-mod-iso-ring-bij-2 x*])
hence x':*?f x* ∈ *carrier P* **unfolding** *poly-mod-ring-carr-1* **by** *simp*
have *?f y* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
by (*rule bij-betw-apply*[*OF poly-mod-iso-ring-bij-2 y*])
hence y':*?f y* ∈ *carrier P* **unfolding** *poly-mod-ring-carr-1* **by** *simp*

have *0*: *?f x* $\oplus_{?T}$ *?f y* = *pmod* (*?f x* $\oplus_P$ *?f y*) *f* **by** (*intro poly-mod-ring-add* x' y')
also have ... ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
using x' y' **by** (*intro poly-mod-ring-carr*) *auto*
finally have *xy*: *?f x* $\oplus_{?T}$ *?f y* ∈ *carrier* (*ring-of* (*poly-mod-ring R f*)) **by** *simp*

have *?f* (*x* $\oplus_{?S}$ *y*) = *?f* (*poly-mod-ring-iso R f* (*?f x*) $\oplus_{?S}$ *poly-mod-ring-iso R f* (*?f y*))
using *x y* **by** (*simp add*:*poly-mod-ring-iso-inv-1*)
also have ... = *?f* ((*I* +>$_P$ (*?f x*)) $\oplus_{?S}$ (*I* +>$_P$ (*?f y*)))
unfolding *poly-mod-ring-iso-def* **by** *simp*

also have ... = *?f* (*I* +>$_P$ (*?f x* ⊕$_P$ *?f y*))
using x' y' **by** *simp*
also have ... = *?f* (*I* +>$_P$ (*pmod* (*?f x* ⊕$_P$ *?f y*) *f*))
using x' y' **by** (*subst ideal-mod*) *auto*
also have ... = *?f* (*I* +>$_P$ (*?f x* ⊕$_{?T}$ *?f y*))
unfolding *0* **by** *simp*
also have ... = *?f* (*poly-mod-ring-iso R f* (*?f x* ⊕$_{?T}$ *?f y*))
unfolding *poly-mod-ring-iso-def* **by** *simp*
also have ... = *?f x* ⊕$_{?T}$ *?f y*
using *xy* **by** (*intro poly-mod-ring-iso-inv-2*)
finally show *?f* (*x* ⊕$_{?S}$ *y*) = *?f x* ⊕$_{?T}$ *?f y* **by** *simp*
next
have *poly-mod-ring-iso R f* $\mathbf{1}_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)}$ = (*I* +>$_P$ $\mathbf{1}_P$)
unfolding *poly-mod-ring-one poly-mod-ring-iso-def* **by** *simp*
also have ... = $\mathbf{1}_{P\ Quot\ I}$ **using** *s.hom-one* **by** *simp*
finally have *poly-mod-ring-iso R f* $\mathbf{1}_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)}$ = $\mathbf{1}_{P\ Quot\ I}$ **by** *simp*
moreover have *degree* $\mathbf{1}_P$ < *degree f*
using *deg-f* **unfolding** *univ-poly-one* **by** *simp*
hence $\mathbf{1}_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)}$ ∈ *carrier* (*ring-of* (*poly-mod-ring R f*))
unfolding *poly-mod-ring-one poly-mod-ring-carr-1* **by** *simp*
ultimately show *?f* ($\mathbf{1}_{?S}$) = $\mathbf{1}_{?T}$
unfolding *poly-mod-ring-iso-inv-def* **by** (*intro the-inv-into-f-eq poly-mod-ring-iso-inj*)
next
show *bij-betw ?f* (*carrier ?S*) (*carrier ?T*) **by** (*rule poly-mod-iso-ring-bij-2*)
qed

lemma *cring-poly-mod-ring-1*:
shows *ring-of* (*poly-mod-ring R f*)⦇*zero* := *poly-mod-ring-iso-inv R f* $\mathbf{0}_{P\ Quot\ I}$⦈ =
ring-of (*poly-mod-ring R f*)
and *cring* (*ring-of* (*poly-mod-ring R f*))
proof −
let *?f* = *poly-mod-ring-iso-inv R f*

have *poly-mod-ring-iso R f* $\mathbf{0}_P$ = $\mathbf{0}_{P\ Quot\ PIdl_P\ f}$
unfolding *poly-mod-ring-iso-def* **by** *simp*
moreover have [] ∈ *carrier P* **using** *univ-poly-zero*[**where** *K*=*carrier* (*ring-of R*)] **by** *auto*
ultimately have *?f* $\mathbf{0}_{P\ Quot\ I}$ = $\mathbf{0}_P$
unfolding *univ-poly-zero poly-mod-ring-iso-inv-def* **using** *deg-f*
by (*intro the-inv-into-f-eq bij-betw-imp-inj-on*[*OF poly-mod-iso-ring-bij*])
(*simp-all add*:*add*:*poly-mod-ring-carr-1*)
also have ... = $0_{C\,poly\ R}$ **using** *ring-of-poly*[*OF ring-c*] *domain-cD*[*OF d-poly*] **by** *auto*

finally have *?f* $\mathbf{0}_{P\ Quot\ I}$ = $0_{C\,poly\ R}$ **by** *simp*
thus *ring-of* (*poly-mod-ring R f*)(|*zero* := *?f* $\mathbf{0}_{P\ Quot\ I}$|) = *ring-of* (*poly-mod-ring R f*)
unfolding *ring-of-def poly-mod-ring-def* **by** *auto*
thus *cring* (*ring-of* (*poly-mod-ring R f*))
using *cr.ring-iso-imp-img-cring*[*OF poly-mod-ring-iso-inv*] **by** *simp*
qed

interpretation *cr-p*: *cring* (*ring-of* (*poly-mod-ring R f*))
by (*rule cring-poly-mod-ring-1*)

lemma *cring-c-poly-mod-ring*: $cring_C$ (*poly-mod-ring R f*)
proof −
let *?P* = *ring-of* (*poly-mod-ring R f*)
have $-_{C\,poly\text{-}mod\text{-}ring\ R\ f}\ x = \ominus_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)}\ x$ (**is** *?L* = *?R*)
if $x \in$ *carrier* (*ring-of* (*poly-mod-ring R f*)) **for** *x*
proof (*rule cr-p.minus-equality*[*symmetric*, *OF* - *that*])
have $-_{C\,poly\text{-}mod\text{-}ring\ R\ f}\ x = -_{C\,poly\ R}\ x$ **unfolding** *poly-mod-ring-def* **by** *simp*
also have ... = $\ominus_P\ x$ **using** *that* **unfolding** *poly-mod-ring-carr-1*
by (*subst domain-cD*[*OF d-poly*]) (*simp-all add*:*ring-of-poly*[*OF ring-c*])
finally have *0*:$-_{C\,poly\text{-}mod\text{-}ring\ R\ f}\ x = \ominus_P\ x$ **by** *simp*

have *1*:$\ominus_P\ x \in$ *carrier* (*ring-of* (*poly-mod-ring R f*))
using *that univ-poly-a-inv-degree*[*OF carrier-is-subring*] **unfolding** *poly-mod-ring-carr-1*
by *auto*

have $-_{C\,poly\text{-}mod\text{-}ring\ R\ f}\ x \oplus_{?P} x$ = *pmod* ($\ominus_P\ x \oplus_P x$) *f*
using *that 1* **unfolding** *0 poly-mod-ring-carr-1* **by** (*intro poly-mod-ring-add*) *auto*
also have ... = *pmod* $\mathbf{0}_P$ *f*
using *that* **unfolding** *poly-mod-ring-carr-1* **by** *simp algebra*
also have ... = []
unfolding *univ-poly-zero* **using** *carrier-is-subfield f-carr long-division-zero*(*2*) **by** *presburger*
also have ... = $\mathbf{0}_{?P}$ **by** (*simp add*:*poly-mod-ring-def ring-of-def poly-def*)
finally show $-_{C\,poly\text{-}mod\text{-}ring\ R\ f}\ x \oplus_{?P} x = \mathbf{0}_{?P}$ **by** *simp*

show $-_{C\,poly\text{-}mod\text{-}ring\ R\ f}\ x \in$ *carrier* (*ring-of* (*poly-mod-ring R f*))
unfolding *0* **by** (*rule 1*)
qed
moreover have $x\ ^{-1}{}_{C\,poly\text{-}mod\text{-}ring\ R\ f} = inv_{ring\text{-}of\ (poly\text{-}mod\text{-}ring\ R\ f)}\ x$
if *x-unit*: $x \in$ *Units* (*ring-of* (*poly-mod-ring R f*)) **for** *x*

proof (*rule cr-p.comm-inv-char*[*symmetric*])
show *x-carr*: $x \in$ *carrier* (*ring-of* (*poly-mod-ring R f*))
using *that* **unfolding** *Units-def* **by** *auto*

obtain *y* **where** *y*:$x \otimes_{\textit{ring-of (poly-mod-ring R f)}} y = \mathbf{1}_{\textit{ring-of (poly-mod-ring R f)}}$
and *y-carr*: $y \in$ *carrier* (*ring-of* (*poly-mod-ring R f*))
using *x-unit* **unfolding** *Units-def* **by** *auto*

have *pmod* $(x \otimes_P y)$ $f = x \otimes_{\textit{ring-of (poly-mod-ring R f)}} y$
using *x-carr y-carr* **by** (*intro poly-mod-ring-mult*[*symmetric*])
(*auto simp*:*poly-mod-ring-carr-1*)
also have ... $= \mathbf{1}_P$
unfolding *y poly-mod-ring-one* **by** *simp*
finally have *1*:*pmod* $(x \otimes_P y)$ $f = \mathbf{1}_P$ **by** *simp*

have $\textit{pcoprime}_{\textit{ring-of R}}$ $(x \otimes_P y)$ f = $\textit{pcoprime}_{\textit{ring-of R}}$ f (*pmod*
$(x \otimes_P y)$ f)
using *x-carr y-carr f-carr* **unfolding** *poly-mod-ring-carr-1* **by**
(*intro pcoprime-step*) *auto*
also have ... = $\textit{pcoprime}_{\textit{ring-of R}}$ f $\mathbf{1}_P$ **unfolding** *1* **by** *simp*
also have ... = *True* **using** *pcoprime-one* **by** *simp*
finally have $\textit{pcoprime}_{\textit{ring-of R}}$ $(x \otimes_P y)$ f **by** *simp*
hence $\textit{pcoprime}_{\textit{ring-of R}}$ x f
using *x-carr y-carr f-carr pcoprime-left-factor* **unfolding** *poly-mod-ring-carr-1*
by *blast*
hence *2*:*length* (*snd* (*ext-euclidean R x f*)) = *1*
using *f-carr x-carr pcoprime-c*[*OF field-R*] **unfolding** *poly-mod-ring-carr-1*
$\textit{pcoprime}_C$*.simps*
by *auto*

obtain *u v r* **where** *uvr-def*: ((*u*,*v*),*r*) = *ext-euclidean R x f* **by**
(*metis surj-pair*)

have *x-carr'*: $x \in$ *carrier P* **using** *x-carr* **unfolding** *poly-mod-ring-carr-1*
by *auto*
have *r-eq*:$r = x \otimes_P u \oplus_P f \otimes_P v$ **and** *ruv-carr*: $\{r, u, v\} \subseteq$ *carrier*
P
using *uvr-def*[*symmetric*] *ext-euclidean*[*OF field-R x-carr' f-carr*]
by *auto*

have *length r* = *1* **using** *2 uvr-def*[*symmetric*] **by** *simp*
hence *3*:$r = [\textit{hd } r]$ **by** (*cases r*) *auto*
hence $r \neq \mathbf{0}_P$ **unfolding** *univ-poly-zero* **by** *auto*
hence *hd r* $\in$ *carrier* (*ring-of R*) $-$ $\{\mathbf{0}_{\textit{ring-of R}}\}$
using *ruv-carr* **by** (*intro lead-coeff-carr*) *auto*
hence *r-unit*: $r \in$ *Units P* **using** *3 univ-poly-units*[*OF carrier-is-subfield*]
by *auto*
hence *inv-r-carr*: $\textit{inv}_P$ $r \in$ *carrier P* **by** *simp*

have *0*: $x\ ^{-1}{}_{C\,poly\text{-}mod\text{-}ring\ R\ f} = pmod_C\ R\ (r\ ^{-1}{}_{C\,poly\ R} *_{C\,poly\ R}$ $u)\ f$
by (*simp add*:*poly-mod-ring-def uvr-def*[*symmetric*])
also have ... = $pmod_C\ R\ (inv_P\ r \otimes_P u)\ f$
using *r-unit* **unfolding** *domain-cD*[*OF d-poly*]
by (*subst domain-cD*[*OF d-poly*]) (*simp-all add*:*ring-of-poly*[*OF ring-c*])
also have ... = $pmod\ (inv_P\ r \otimes_P u)\ f$
using *ruv-carr inv-r-carr* **by** (*intro pmod-c*[*OF field-R*] *f-carr*) *simp*
finally have *0*: $x\ ^{-1}{}_{C\,poly\text{-}mod\text{-}ring\ R\ f} = pmod\ (inv_P\ r \otimes_P u)\ f$
by *simp*

show $x\ ^{-1}{}_{C\,poly\text{-}mod\text{-}ring\ R\ f} \in$ *carrier* (*ring-of* (*poly-mod-ring R f*))
using *ruv-carr r-unit* **unfolding** *0* **by** (*intro poly-mod-ring-carr*) *simp*

have *4*: *degree* $\mathbf{1}_P <$ *degree f* **unfolding** *univ-poly-one* **using** *deg-f* **by** *auto*

have *f* $divides_P\ inv_P\ r \otimes_P f \otimes_P v$
using *inv-r-carr ruv-carr f-carr*
by (*intro dividesI*[**where** $c=inv_P\ r \otimes_P v$]) (*simp-all*, *algebra*)
hence *5*: $pmod\ (inv_P\ r \otimes_P f \otimes_P v)\ f = []$
using *f-carr ruv-carr inv-r-carr*
by (*intro iffD2*[*OF pmod-zero-iff-pdivides*[*OF carrier-is-subfield*]]) (*auto simp*:*pdivides-def*)

have $x \otimes_{?P} x\ ^{-1}{}_{C\,poly\text{-}mod\text{-}ring\ R\ f} = pmod\ (x \otimes_P pmod\ (inv_P\ r \otimes_P u)\ f)\ f$
using *ruv-carr inv-r-carr f-carr* **unfolding** *0*
by (*intro poly-mod-ring-mult x-carr′ long-division-closed*[*OF carrier-is-subfield*]) *simp-all*
also have ... = $pmod\ (x \otimes_P (inv_P\ r \otimes_P u))\ f$
using *ruv-carr inv-r-carr f-carr* **by** (*intro pmod-mult-right*[*symmetric*] *x-carr′*) *auto*
also have ... = $pmod\ (inv_P\ r \otimes_P (x \otimes_P u))\ f$
using *x-carr′ ruv-carr inv-r-carr* **by** (*intro arg-cong2*[**where** *f=pmod*] *refl*) (*simp*, *algebra*)
also have ... = $pmod\ (inv_P\ r \otimes_P (r \ominus_P f \otimes_P v))\ f$ **using** *ruv-carr f-carr x-carr′*
by (*intro arg-cong2*[**where** *f=pmod*] *arg-cong2*[**where** $f=(\otimes_P)$] *refl*) (*simp add*:*r-eq*, *algebra*)
also have ... = $pmod\ (inv_P\ r \otimes_P r \ominus_P inv_P\ r \otimes_P f \otimes_P v)\ f$
using *ruv-carr inv-r-carr f-carr* **by** (*intro arg-cong2*[**where** *f=pmod*] *refl*) (*simp*, *algebra*)
also have ... = $pmod\ \mathbf{1}_P\ f \oplus_P pmod\ (\ominus_P\ (inv_P\ r \otimes_P f \otimes_P v))\ f$
using *ruv-carr inv-r-carr f-carr* **unfolding** *d.Units-l-inv*[*OF*

r-unit] *a-minus-def*
by (*intro long-division-add*[*OF carrier-is-subfield*]) *simp-all*
also have ... = $\mathbf{1}_P \ominus_P$ *pmod* (inv_P *r* $\otimes_P$ *f* $\otimes_P$ *v*) *f*
using *ruv-carr f-carr inv-r-carr* **unfolding** *a-minus-def*
by (*intro arg-cong2*[**where** *f*=($\oplus_P$)] *pmod-const*[*OF carrier-is-subfield*]
long-division-a-inv[*OF carrier-is-subfield*] *4*) *simp-all*
also have ... = $\mathbf{1}_P \ominus_P \mathbf{0}_P$ **unfolding** *5 univ-poly-zero* **by** *simp*
also have ... = $\mathbf{1}_{\textit{ring-of (poly-mod-ring R f)}}$ **unfolding** *poly-mod-ring-one*
by *algebra*
finally show $x \otimes_{\textit{ring-of (poly-mod-ring R f)}} x^{-1}{}_{C\,\textit{poly-mod-ring R f}}$
= $\mathbf{1}_{?P}$ **by** *simp*
qed
ultimately show *?thesis* **using** *cring-poly-mod-ring-1* **by** (*intro*
cring-cI)
qed

end

lemma *field-c-poly-mod-ring*:
assumes *field-R*: $\textit{field}_C$ *R*
assumes *monic-irreducible-poly* (*ring-of R*) *f*
shows $\textit{field}_C$ (*poly-mod-ring R f*)
proof −
interpret *field ring-of R* **using** *field-R* **unfolding** $\textit{field}_C$*-def* **by**
auto

have *f-carr*: *f* ∈ *carrier* (*poly-ring* (*ring-of R*))
using *assms*(*2*) *monic-poly-carr* **unfolding** *monic-irreducible-poly-def*
by *auto*

have *deg-f*: *degree f* > *0* **using** *monic-poly-min-degree assms*(*2*) **by**
fastforce

have *f-irred*: $\textit{pirreducible}_{\textit{ring-of R}}$ (*carrier* (*ring-of R*)) *f*
using *assms*(*2*) **unfolding** *monic-irreducible-poly-def* **by** *auto*

interpret *r*:*field poly-ring* (*ring-of R*) *Quot* ($\textit{PIdl}_{\textit{poly-ring (ring-of R)}}$
f)
using *f-irred f-carr iffD2*[*OF rupture-is-field-iff-pirreducible*[*OF*
carrier-is-subfield]]
unfolding *rupture-def* **by** *blast*

have *field* (*ring-of* (*poly-mod-ring R f*))
using *r.ring-iso-imp-img-field*[*OF poly-mod-ring-iso-inv*[*OF field-R*
f-carr deg-f]]
using *cring-poly-mod-ring-1*(*1*)[*OF field-R f-carr deg-f*] **by** *simp*
moreover have $\textit{cring}_C$ (*poly-mod-ring R f*)
by (*rule cring-c-poly-mod-ring*[*OF field-R f-carr deg-f*])

ultimately show *?thesis* **unfolding** *field$_C$-def domain$_C$-def* **using**
field.axioms(1) **by** *blast*
qed

lemma *enum-c-poly-mod-ring*:
  assumes *enum$_C$ R ring$_C$ R*
  shows *enum$_C$ (poly-mod-ring R f)*
proof (*rule enum-cI*)
  let *?l = degree f*
  let *?b = idx-size R*
  let *?S = carrier (ring-of (poly-mod-ring R f))*

  note *bij-0 = bij-betw-poly-enum*[**where** *l=degree f, OF assms(1,2)*]
  have *?S = {xs ∈ carrier (poly-ring (ring-of R)). length xs ≤ ?l}*
    unfolding *ring-of-poly[OF assms(2),symmetric] poly-mod-ring-def*
by (*simp add:ring-of-def*)
  hence *bij-1:bij-betw (poly-enum R (degree f)) {..<idx-size R ^ degree f} ?S*
    using *bij-0* **by** *simp*
  hence *bij-2:bij-betw (idx-enum (poly-mod-ring R f)) {..<idx-size R^degree f} ?S*
    unfolding *poly-mod-ring-def* **by** *simp*

  have *order (ring-of (poly-mod-ring R f)) = card ?S*
    unfolding *Coset.order-def* **by** *simp*
  also have ... = *card {..<idx-size R ^ degree f}* **using** *bij-2* **by** (*metis bij-betw-same-card*)
  finally have *ord-poly-mod-ring*: *order (ring-of (poly-mod-ring R f)) = idx-size R^degree f*
    by *simp*

  show *finite ?S* **using** *bij-2 bij-betw-finite* **by** *blast*
  show *idx-size (poly-mod-ring R f) = order (ring-of (poly-mod-ring R f))*
    unfolding *ord-poly-mod-ring* **by** (*simp add:poly-mod-ring-def*)
  show *bij-betw (idx-enum (poly-mod-ring R f)) {..<order (ring-of (poly-mod-ring R f))} ?S*
    using *bij-2 ord-poly-mod-ring* **by** *auto*
  show *idx-enum-inv (poly-mod-ring R f) (idx-enum (poly-mod-ring R f) x) = x* (**is** *?L* = -)
    if *x < order (ring-of (poly-mod-ring R f))* **for** *x*
  proof −
    have *?L = poly-enum-inv R (degree f) (poly-enum R (degree f) x)*
      unfolding *poly-mod-ring-def* **by** *simp*
    also have ... = *the-inv-into {..<?b ^ ?l} (poly-enum R ?l) (poly-enum R ?l x)*
      using *that ord-poly-mod-ring*
    by (*intro poly-enum-inv[OF assms(1,2),symmetric] bij-betw-apply[OF*

bij-0]) *auto*
  also have ... = *x*
  using *that ord-poly-mod-ring* **by** (*intro the-inv-into-f-f bij-betw-imp-inj-on*[*OF* *bij-0*]) *auto*
  finally show *?thesis* **by** *simp*
 qed
qed

end

16 Algorithms for finding irreducible polynomials

theory *Find-Irreducible-Poly*
 imports
  Finite-Fields-More-PMF
  Finite-Fields-Poly-Factor-Ring-Code
  Rabin-Irreducibility-Test-Code
  Probabilistic-While.While-SPMF
  Card-Irreducible-Polynomials
  Executable-Randomized-Algorithms.Randomized-Algorithm
  HOL−Library.Log-Nat
begin

hide-const (**open**) *Divisibility.prime*
hide-const (**open**) *Finite-Fields-Factorization-Ext.multiplicity*
hide-const (**open**) *Numeral-Type.mod-ring*
hide-const (**open**) *Polynomial.degree*
hide-const (**open**) *Polynomial.order*

Enumeration of the monic polynomials in lexicographic order.

definition *enum-monic-poly* :: ('*a*,'*b*) *idx-ring-enum-scheme* ⇒ *nat* ⇒ *nat* ⇒ '*a list*
 where *enum-monic-poly A d i* = $1_{C\,A}$#[*idx-enum A* (*nth-digit i j* (*idx-size A*)). *j* ← *rev* [*0*..<*d*]]

lemma *enum-monic-poly*:
 assumes $field_C$ *R* $enum_C$ *R*
 shows *bij-betw* (*enum-monic-poly R d*) {..<*order* (*ring-of R*)^*d*}
  {*f*. *monic-poly* (*ring-of R*) *f* ∧ *degree f* = *d*}
proof −
 let *?f* = (λ*x*. $1_{C\,R}$ # *map* (λ*j*. *idx-enum R* (*x j*)) (*rev* [*0*..<*d*]))
 let *?R* = *ring-of R*

 note *select-bij* = *enum-cD*(*3*)[*OF assms*(*2*)]
 note *fin-carr* = *enum-cD*(*1*)[*OF assms*(*2*)]
 note *fo* = *field-cD*[*OF assms*(*1*)]

interpret *finite-field ring-of R*
using *fin-carr assms*(*1*) **unfolding** *finite-field-def finite-field-axioms-def field*$_C$*-def* **by** *auto*

have *1*:*enum-monic-poly R d* = *?f* ∘ (λ*v*. λ*x*∈{..<*d*}. *nth-digit v x* (*order* (*ring-of R*)))
unfolding *enum-monic-poly-def comp-def enum-cD*[*OF assms*(*2*)]
by (*intro ext arg-cong2*[**where** *f*=(#)] *refl map-cong*) *auto*

have *2*:*?f* = (λ*x*. $1_{C\,R}$ # *map x* (*rev* [*0*..<*d*])) ∘ (λ*x*. λ*i*∈{..<*d*}. *idx-enum R* (*x i*))
unfolding *comp-def* **by** *auto*

have *3*: (λ*x*. $\mathbf{1}_{ring\text{-}of\ R}$#*map x* (*rev* [*0*..<*d*])) = (λ*x*. $\mathbf{1}_{ring\text{-}of\ R}$#*x*) ∘*rev*∘ (λ*x*. *map x* [*0*..<*d*])
unfolding *comp-def* **by** (*intro ext*) (*simp add*:*rev-map*)

have *ap-bij*: *bij-betw* ((#) $\mathbf{1}_{?R}$) {*x*. *set x*⊆*carrier ?R*∧*length x*=*d*} {*f*. *monic-poly ?R f* ∧ *degree f*=*d*}
using *list.collapse* **unfolding** *monic-poly-def univ-poly-carrier*[*symmetric*] *polynomial-def*
by (*intro bij-betwI*[**where** *g*=*tl*]) (*fastforce intro*:*in-set-tlD*)+

have *rev-bij*:
bij-betw rev {*x*. *set x* ⊆ *carrier ?R* ∧ *length x* = *d*} {*x*. *set x* ⊆ *carrier ?R* ∧ *length x* = *d*}
by (*intro bij-betwI*[**where** *g*=*rev*]) *auto*

have *bij-betw* (λ*x*. $\mathbf{1}_{?R}$#*map x* (*rev* [*0*..<*d*])) ({..<*d*} $\rightarrow_E$ *carrier ?R*) {*f*. *monic-poly ?R f*∧*degree f*=*d*}
unfolding *3* **by** (*intro bij-betw-trans*[*OF lists-bij*] *bij-betw-trans*[*OF rev-bij*] *ap-bij*)
hence *bij-betw ?f* ({..<*d*} $\rightarrow_E$ {..<*order ?R*}) {*f*. *monic-poly ?R f* ∧ *degree f* = *d*}
unfolding *2* **by** (*intro bij-betw-trans*[*OF lift-bij-betw*[*OF select-bij*]]) (*simp add*:*fo*)
thus *?thesis*
unfolding *1* **by** (*intro bij-betw-trans*[*OF nth-digit-bij*])
qed

abbreviation *tick-spmf* :: (′*a* × *nat*) *spmf* ⇒ (′*a* × *nat*) *spmf*
where *tick-spmf* ≡ *map-spmf* (λ(*x*,*c*). (*x*,*c*+*1*))

Finds an irreducible polynomial in the finite field *mod-ring p* with given degree n:

partial-function (*spmf*) *sample-irreducible-poly* :: *nat* ⇒ *nat* ⇒ (*nat list* × *nat*) *spmf*
where
sample-irreducible-poly p n =

```
do {
  k ← spmf-of-set {..<p^n};
  let poly = enum-monic-poly (mod-ring p) n k;
  if rabin-test (mod-ring p) poly
    then return-spmf (poly,1)
    else tick-spmf (sample-irreducible-poly p n)
}
```

The following is a deterministic version. It returns the lexicographically minimal monic irreducible polynomial. Note that contrary to the randomized algorithm, the run time of the deterministic algorithm may be exponential (w.r.t. to the size of the field and degree of the polynomial).

fun *find-irreducible-poly* :: *nat* ⇒ *nat* ⇒ *nat list*
 where *find-irreducible-poly p n* = (*let f* = *enum-monic-poly* (*mod-ring p*) *n in*
  f (*while* ((λ*k*. ¬*rabin-test* (*mod-ring p*) (*f k*))) (λ*x*. *x* + *1*) *0*))

definition *cost* :: (*'a* × *nat*) *option* ⇒ *enat*
 where *cost x* = (*case x of None* ⇒ ∞ | *Some* (-,*r*) ⇒ *enat r*)

lemma *cost-tick*: *cost* (*map-option* (λ(*x*, *c*). (*x*, *Suc c*)) *c*) = *eSuc* (*cost c*)
 by (*cases c*) (*auto simp*:*cost-def eSuc-enat*)

context
 fixes *n p* :: *nat*
 assumes *p-prime*: *Factorial-Ring.prime p*
 assumes *n-gt-0*: *n* > *0*
begin

private definition *S* **where** *S* = {*f*. *monic-poly* (*ring-of* (*mod-ring p*)) *f* ∧ *degree f* = *n* }
private definition *T* **where** *T* = {*f*. *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) *f* ∧ *degree f* = *n*}

lemmas *field-c* = *mod-ring-is-field-c*[*OF p-prime*]
lemmas *enum-c* = *mod-ring-is-enum-c*[**where** *n*=*p*]

interpretation *finite-field ring-of* (*mod-ring p*)
 unfolding *finite-field-def finite-field-axioms-def*
 by (*intro mod-ring-is-field conjI mod-ring-finite p-prime*)

private lemmas *field-ops* = *field-cD*[*OF field-c*]

private lemma *S-fin*: *finite S*
 unfolding *S-def*
 using *enum-monic-poly*[*OF field-c enum-c*, **where** *d*=*n*]
  bij-betw-finite **by** *auto*

private lemma *T-sub-S*: $T \subseteq S$
  unfolding *S-def T-def monic-irreducible-poly-def* **by** *auto*

private lemma *T-card-gt-0*: *real* (*card T*) > *0*
proof −
  have *0* < *real* (*order* (*ring-of* (*mod-ring p*))) ^ *n* / (*2* ∗ *real n*)
   using *n-gt-0 finite-field-min-order* **by** (*intro divide-pos-pos*) (*simp-all*)
 also have ... ≤ *real* (*card T*) **unfolding** *T-def* **by** (*intro card-irred-gt-0 n-gt-0*)
  finally show *real* (*card T*) > *0* **by** *auto*
qed

private lemma *S-card-gt-0*: *real* (*card S*) > *0*
proof −
  have *0* < *card T* **using** *T-card-gt-0* **by** *simp*
  also have ... ≤ *card S* **by** (*intro card-mono T-sub-S S-fin*)
  finally have *0* < *card S* **by** *simp*
  thus *?thesis* **by** *simp*
qed

private lemma *S-ne*: *S* ≠ {} **using** *S-card-gt-0* **by** *auto*

private lemma *sample-irreducible-poly-step-aux*:
    do {
       k ← *spmf-of-set* {..<*p*^*n*};
       let poly = *enum-monic-poly* (*mod-ring p*) *n k*;
       if rabin-test (*mod-ring p*) *poly then return-spmf* (*poly*,*c*) *else x*
    } =
     do {
       poly ← *spmf-of-set S*;
       if monic-irreducible-poly (*ring-of* (*mod-ring p*)) *poly*
           then return-spmf (*poly*,*c*)
           else x
     }
  (**is** *?L* = *?R*)
proof −
  have *order* (*ring-of* (*mod-ring p*)) = *p*
   unfolding *Finite-Fields-Mod-Ring-Code.mod-ring-def Coset.order-def ring-of-def* **by** *simp*
  hence *0*:*spmf-of-set S* = *map-spmf* (*enum-monic-poly* (*mod-ring p*) *n*) (*spmf-of-set* {..<*p*^*n*})
   using *enum-monic-poly*[*OF field-c enum-c*, **where** *d*=*n*] **unfolding** *bij-betw-def S-def*
     by (*subst map-spmf-of-set-inj-on*) *auto*

  have *?L* =*do* {*f* ← *spmf-of-set S*; *if rabin-test* (*mod-ring p*) *f then return-spmf* (*f*,*c*) *else x*}
    unfolding *0 bind-map-spmf* **by** (*simp add*:*Let-def comp-def*)

also have ... = *?R*
  using *set-spmf-of-set-finite*[*OF S-fin*]
 by (*intro bind-spmf-cong refl if-cong rabin-test field-c enum-c*) (*simp add*:*S-def*)
finally show *?thesis* **by** *simp*
qed

private lemma *sample-irreducible-poly-step*:
sample-irreducible-poly p n =
  do {
   poly ← *spmf-of-set S*;
   if monic-irreducible-poly (*ring-of* (*mod-ring p*)) *poly*
    then return-spmf (*poly*,*1*)
    else tick-spmf (*sample-irreducible-poly p n*)
  }
by (*subst sample-irreducible-poly.simps*) (*simp add*:*sample-irreducible-poly-step-aux*)

private lemma *sample-irreducible-poly-aux-1*:
ord-spmf (=) (*map-spmf fst* (*sample-irreducible-poly p n*)) (*spmf-of-set T*)
proof (*induction rule*:*sample-irreducible-poly.fixp-induct*)
 case *1* **thus** *?case* **by** *simp*
next
 case *2* **thus** *?case* **by** *simp*
next
 case (*3 rec*)
 let *?f* = *monic-irreducible-poly* (*ring-of* (*mod-ring p*))

 have *real* (*card* (*S*∩−{*x*. *?f x*})) = *real* (*card* (*S* − *T*))
  unfolding *S-def T-def* **by** (*intro arg-cong*[**where** *f*=*card*] *arg-cong*[**where** *f*=*of-nat*]) (*auto*)
 also have ... = *real* (*card S* − *card T*)
  by (*intro arg-cong*[**where** *f*=*of-nat*] *card-Diff-subset T-sub-S finite-subset*[*OF T-sub-S S-fin*])
 also have ... = *real* (*card S*) − *card T*
  by (*intro of-nat-diff card-mono S-fin T-sub-S*)
 finally have *0*:*real* (*card* (*S*∩−{*x*. *?f x*})) = *real* (*card S*) − *card T* **by** *simp*

 have *S-card-gt-0*: *real* (*card S*) > *0* **using** *S-ne S-fin* **by** *auto*

 have *do* {*f* ← *spmf-of-set S*;*if ?f f then return-spmf f else spmf-of-set T*} = *spmf-of-set T*
  (**is** *?L* = *?R*)
 proof (*rule spmf-eqI*)
  fix *i*
   have *spmf ?L i* = *spmf* (*pmf-of-set S* ≫=(λ*x*. *if ?f x then return-spmf x else spmf-of-set T*)) *i*
    unfolding *spmf-of-pmf-pmf-of-set*[*OF S-fin S-ne*, *symmetric*]

spmf-of-pmf-def
by (*simp add*:*bind-spmf-def bind-map-pmf*)
also have ... = ($\int$ *x*. (*if ?f x then of-bool* (*x*=*i*) *else spmf* (*spmf-of-set T*) *i*) ∂*pmf-of-set S*)
unfolding *pmf-bind if-distrib if-distribR pmf-return-spmf indicator-def* **by** (*simp cong*:*if-cong*)
also have ... = ($\sum$ *x* $\in$ *S*. (*if ?f x then of-bool* (*x* = *i*) *else spmf* (*spmf-of-set T*) *i*))/*card S*
by (*subst integral-pmf-of-set*[*OF S-ne S-fin*]) *simp*
also have ... = (*of-bool* (*i* $\in$ *T*) + *spmf* (*spmf-of-set T*) *i*$*$*real* (*card* (*S*$\cap$$-${*x*. *?f x*})))/*card S*
using *S-fin S-ne*
by (*subst sum.If-cases*[*OF S-fin*]) (*simp add*:*of-bool-def T-def monic-irreducible-poly-def S-def*)
also have ... = (*of-bool* (*i* $\in$ *T*)$*$(*1* + *real* (*card* (*S*$\cap$$-${*x*. *?f x*}))/*real* (*card T*)))/*card S*
unfolding *spmf-of-set indicator-def* **by** (*simp add*:*algebra-simps*)
also have ... = (*of-bool* (*i* $\in$ *T*)$*$(*real* (*card S*)/*real* (*card T*)))/*card S*
using *T-card-gt-0* **unfolding** *0* **by** (*simp add*:*field-simps*)
also have ... = *of-bool* (*i* $\in$ *T*)/*real* (*card T*)
using *S-card-gt-0* **by** (*simp add*:*field-simps*)
also have ... = *spmf ?R i*
unfolding *spmf-of-set* **by** *simp*
finally show *spmf ?L i* = *spmf ?R i*
by *simp*
qed
hence *ord-spmf* (=)
(*spmf-of-set S* $\gg=$ (λ*x*. *if ?f x then return-spmf x else spmf-of-set T*)) (*spmf-of-set T*)
by *simp*
moreover have *ord-spmf* (=)
(*do* { *poly* $\leftarrow$ *spmf-of-set S*; *if ?f poly then return-spmf poly else map-spmf fst* (*rec p n*)})
(*do* { *poly* $\leftarrow$ *spmf-of-set S*; *if ?f poly then return-spmf poly else spmf-of-set T*})
using *3* **by** (*intro bind-spmf-mono'*) *simp-all*
ultimately have *ord-spmf* (=) (*spmf-of-set S* $\gg=$
(λ*x*. *if ?f x then return-spmf x else map-spmf fst* (*rec p n*))) (*spmf-of-set T*)
using *spmf.leq-trans* **by** *force*
thus *?case* **unfolding** *sample-irreducible-poly-step-aux map-spmf-bind-spmf*
by (*simp add*:*comp-def if-distribR if-distrib spmf.map-comp case-prod-beta cong*:*if-cong*)
qed

lemma *cost-sample-irreducible-poly*:
($\int^+$*x*. *cost x* ∂*sample-irreducible-poly p n*) $\leq$ *2*$*$*real n* (**is** *?L* $\leq$ *?R*)
proof $-$

let *?f* = *monic-irreducible-poly* (*ring-of* (*mod-ring p*))
let *?a* = (λ*t. measure* (*sample-irreducible-poly p n*) {ω. *enat t* < *cost* ω})
let *?b* = (λ*t. measure* (*sample-irreducible-poly p n*) {ω. *enat t* $\geq$ *cost* ω})

define α **where** α = *measure* (*pmf-of-set S*) {*x. ?f x*}
have α*-le-1*: $\alpha \leq 1$ **unfolding** α*-def* **by** *simp*

have *1* / (*2*∗ *real n*) = (*card S* / (*2* ∗ *real n*)) / *card S*
using *S-card-gt-0* **by** (*simp add*:*algebra-simps*)
also have ... = (*real* (*order* (*ring-of* (*mod-ring p*)))^*n* / (*2* ∗ *real n*)) / *card S*
unfolding *S-def bij-betw-same-card*[*OF enum-monic-poly*[*OF field-c enum-c*, **where** *d*=*n*],*symmetric*]
by *simp*
also have ... $\leq$ *card T* / *card S*
unfolding *T-def* **by** (*intro divide-right-mono card-irred-gt-0 n-gt-0*) *auto*
also have ... = α
unfolding α*-def measure-pmf-of-set*[*OF S-ne S-fin*]
by (*intro arg-cong2*[**where** *f*=(/)] *refl arg-cong*[**where** *f*=*of-nat*] *arg-cong*[**where** *f*=*card*])
(*auto simp*: *S-def T-def monic-irreducible-poly-def*)
finally have α*-lb*: *1*/ (*2*∗*real n*) $\leq \alpha$
by *simp*
have *0* < *1*/ (*2*∗*real n*) **using** *n-gt-0* **by** *simp*
also have ... $\leq \alpha$ **using** α*-lb* **by** *simp*
finally have α*-gt-0*: α > *0* **by** *simp*

have *a-step-aux*: *norm* (*a* ∗ *b*) $\leq$ *1* **if** *norm a* $\leq$ *1* *norm b* $\leq$ *1* **for** *a b* :: *real*
using *that* **by** (*simp add*:*abs-mult mult-le-one*)

have *b-eval*: *?b t* = ($\int$ *x*. (*if ?f x then of-bool*(*t* $\geq$ *1*) *else*
measure (*sample-irreducible-poly p n*) {ω. *enat t* $\geq$ *eSuc* (*cost* ω)}) ∂*pmf-of-set S*)
(**is** *?L1* = *?R1*) **for** *t*
proof −
have *?b t* = *measure* (*bind-spmf* (*spmf-of-set S*) (λ*x. if ?f x then return-spmf* (*x*,*1*) *else*
tick-spmf (*sample-irreducible-poly p n*))) {ω. *enat t* $\geq$ *cost* ω}
by (*subst sample-irreducible-poly-step*) *simp*
also have ... = *measure* (*bind-pmf* (*pmf-of-set S*) (λ*x. if ?f x then return-spmf* (*x*,*1*) *else*
tick-spmf (*sample-irreducible-poly p n*))) {ω. *enat t* $\geq$ *cost* ω}
unfolding *spmf-of-pmf-pmf-of-set*[*OF S-fin S-ne*, *symmetric*]
by (*simp add*:*spmf-of-pmf-def bind-map-pmf bind-spmf-def*)
also have ... = ($\int$ *x*. (*if ?f x then of-bool*(*t* $\geq$ *1*) *else*

measure (*tick-spmf* (*sample-irreducible-poly p n*)) {ω. *enat t* ≥ *cost* ω}) ∂*pmf-of-set S*)
unfolding *measure-bind-pmf if-distrib if-distribR emeasure-return-pmf*
by (*simp add*:*indicator-def cost-def comp-def cong*:*if-cong*)
also have ... = *?R1*
unfolding *measure-map-pmf vimage-def*
by (*intro arg-cong2*[**where** *f*=*integral*^{L}] *refl ext if-cong arg-cong2*[**where** *f*=*measure*])
(*auto simp add*:*vimage-def cost-tick eSuc-enat*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

have *b-eval-2*: *?b t* = *1* − (*1*−α)^*t* **for** *t*
proof (*induction t*)
case *0*
have *?b 0* = *0* **unfolding** *b-eval* **by** (*simp add*:*enat-0 cong*:*if-cong*)
thus *?case* **by** *simp*
next
case (*Suc t*)
have *?b* (*Suc t*) = (∫ *x*. (*if ?f x then 1 else ?b t*) ∂*pmf-of-set S*)
unfolding *b-eval*[*of Suc t*]
by (*intro arg-cong2*[**where** *f*=*integral*^{L}] *if-cong arg-cong2*[**where** *f*=*measure*])
(*auto simp add*: *eSuc-enat*[*symmetric*])
also have ... = (∫ *x*. *indicator* {*x*. *?f x*} *x* + *?b t* ∗ *indicator* {*x*. ¬*?f x*} *x* ∂*pmf-of-set S*)
by (*intro Bochner-Integration.integral-cong*) (*auto simp*:*algebra-simps*)
also have ... = (∫ *x*. *indicator* {*x*. *?f x*} *x* ∂*pmf-of-set S*) +
(∫ *x*. *?b t* ∗ *indicator* {*x*. ¬*?f x*} *x* ∂*pmf-of-set S*)
by (*intro Bochner-Integration.integral-add measure-pmf.integrable-const-bound*[**where** *B*=*1*]
AE-pmfI a-step-aux) *auto*
also have ... = α + *?b t* ∗ *measure* (*pmf-of-set S*) {*x*. ¬*?f x*} **unfolding** α*-def* **by** *simp*
also have ... = α + (*1*−α) ∗ *?b t*
unfolding α*-def*
by (*subst measure-pmf.prob-compl*[*symmetric*]) (*auto simp*:*Compl-eq-Diff-UNIV Collect-neg-eq*)
also have ... = *1* − (*1*−α)^*Suc t*
unfolding *Suc* **by** (*simp add*:*algebra-simps*)
finally show *?case* **by** *simp*
qed

hence *a-eval*: *?a t* = (*1*−α)^*t* **for** *t*
proof −
have *?a t* = *1* − *?b t*
by (*simp add*: *measure-pmf.prob-compl*[*symmetric*] *Compl-eq-Diff-UNIV*[*symmetric*]
Collect-neg-eq[*symmetric*] *not-le*)

also have ... = $(1-\alpha)$^t
unfolding *b-eval-2* **by** *simp*
finally show *?thesis* **by** *simp*
qed

have *?L* = ($\sum$ *t. emeasure* (*sample-irreducible-poly p n*) {ω. *enat t* < *cost* ω})
by (*subst nn-integral-enat-function*) *simp-all*
also have ... = ($\sum$ *t. ennreal* (*?a t*))
unfolding *measure-pmf.emeasure-eq-measure* **by** *simp*
also have ... = ($\sum$ *t. ennreal* (($1-\alpha$)^t))
unfolding *a-eval* **by** (*intro arg-cong*[**where** *f*=*suminf*] *ext*) (*simp add*: α-*def ennreal-mult'*)
also have ... = *ennreal* (*1* / ($1-(1-\alpha)$))
using α-*le-1* α-*gt-0*
by (*intro arg-cong2*[**where** *f*=(∗)] *refl suminf-ennreal-eq geometric-sums*) *auto*
also have ... = *ennreal* (*1* / α) **using** α-*le-1* α-*gt-0* **by** *auto*
also have ... ≤ *?R*
using α-*lb n-gt-0* α-*gt-0* **by** (*intro ennreal-leI*) (*simp add*:*field-simps*)
finally show *?thesis* **by** *simp*
qed

private lemma *weight-sample-irreducible-poly*:
weight-spmf (*sample-irreducible-poly p n*) = *1* (**is** *?L* = *?R*)
proof (*rule ccontr*)
assume *?L* ≠ *1*
hence *?L* < *1* **using** *less-eq-real-def weight-spmf-le-1* **by** *blast*
hence (∞::*ennreal*) = ∞ ∗ *ennreal* (*1*−*?L*) **by** *simp*
also have ... = ∞ ∗ *ennreal* (*pmf* (*sample-irreducible-poly p n*) *None*)
unfolding *pmf-None-eq-weight-spmf*[*symmetric*] **by** *simp*
also have ... = ($\int^+$*x*. ∞ ∗ *indicator* {*None*} *x* ∂*sample-irreducible-poly p n*)
by (*simp add*:*emeasure-pmf-single*)
also have ... ≤ ($\int^+$*x. cost x* ∂*sample-irreducible-poly p n*)
unfolding *cost-def* **by** (*intro nn-integral-mono*) (*auto simp*:*indicator-def*)
also have ... ≤ *2*∗*real n* **by** (*intro cost-sample-irreducible-poly*)
finally have (∞::*ennreal*) ≤ *2* ∗ *real n* **by** *simp*
thus *False* **using** *linorder-not-le* **by** *fastforce*
qed

lemma *sample-irreducible-poly-result*:
map-spmf fst (*sample-irreducible-poly p n*) =
spmf-of-set {*f*. *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) *f* ∧ *degree f* = *n*} (**is** *?L* = *?R*)
proof −
have *?L* = *spmf-of-set T* **using** *weight-sample-irreducible-poly*
by (*intro eq-iff-ord-spmf sample-irreducible-poly-aux-1*) (*auto in-*

tro:*weight-spmf-le-1*)
thus *?thesis* **unfolding** *T-def* **by** *simp*
qed

lemma *find-irreducible-poly-result*:
defines *res* ≡ *find-irreducible-poly p n*
shows *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) *res degree res* = *n*
proof −
let *?f* = *enum-monic-poly* (*mod-ring p*) *n*

have *ex*:∃ *k*. *?f k* ∈ *T* ∧ *k* < *order* (*ring-of* (*mod-ring p*))^*n*
proof (*rule ccontr*)
assume ∄ *k*. *?f k* ∈ *T* ∧ *k* < *order* (*ring-of* (*mod-ring p*)) ^ *n*
hence *?f* ‘ {..<*order* (*ring-of* (*mod-ring p*)) ^ *n*} ∩ *T* = {} **by** *auto*
hence *S* ∩ *T* = {}
unfolding *S-def* **using** *bij-betw-imp-surj-on*[*OF enum-monic-poly*[*OF field-c enum-c*]] **by** *auto*
hence *T* = {} **using** *T-sub-S* **by** *auto*
thus *False* **using** *T-card-gt-0* **by** *simp*
qed

then obtain *k* :: *nat* **where** *k-def*: *?f k* ∈ *T* ∀ *j*<*k*. *?f j* ∉ *T*
using *exists-least-iff*[**where** *P*=λ*x*. *?f x* ∈ *T*] **by** *auto*

have *k-ub*: *k* < *order* (*ring-of* (*mod-ring p*))^*n*
using *ex k-def*(*2*) **by** (*meson dual-order.strict-trans1 not-less*)

have *a*: *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) (*?f k*)
using *k-def*(*1*) **unfolding** *T-def* **by** *simp*
have *b*: *monic-poly* (*ring-of* (*mod-ring p*)) (*?f j*) *degree* (*?f j*) = *n* **if** *j* ≤ *k* **for** *j*
proof −
have *j* < *order* (*ring-of* (*mod-ring p*)) ^*n* **using** *k-ub that* **by** *simp*
hence *?f j* ∈ *S* **unfolding** *S-def* **using** *bij-betw-apply*[*OF enum-monic-poly*[*OF field-c enum-c*]] **by** *auto*
thus *monic-poly* (*ring-of* (*mod-ring p*)) (*?f j*) *degree* (*?f j*) = *n* **unfolding** *S-def* **by** *auto*
qed

have *c*: ¬*monic-irreducible-poly* (*ring-of* (*mod-ring p*)) (*?f j*) **if** *j* < *k* **for** *j*
using *b*[*of j*] *that k-def*(*2*) **unfolding** *T-def* **by** *auto*

have *2*: *while* ((λ*k*. ¬*rabin-test* (*mod-ring p*) (*?f k*))) (λ*x*. *x* + *1*) (*k*−*j*) = *k* **if** *j* ≤ *k* **for** *j*
using *that* **proof** (*induction j*)
case *0*

have *rabin-test* (*mod-ring p*) (*?f k*) **by** (*intro iffD2*[*OF rabin-test*] *a b field-c enum-c*) *auto*
thus *?case* **by** (*subst while-unfold*) *simp*
next
case (*Suc j*)
hence ¬*rabin-test* (*mod-ring p*) (*?f* (*k*−*Suc j*))
using *b c* **by** (*subst rabin-test*[*OF field-c enum-c*]) *auto*
moreover have *Suc* (*Suc* (*k* − *Suc j*)) = *Suc* (*k*−*j*) **using** *Suc* **by** *simp*
ultimately show *?case* **using** *Suc*(*1*) **by** (*subst while-unfold*) *simp*
qed

have *3*:*while* ((λ*k*. ¬*rabin-test* (*mod-ring p*) (*?f k*))) (λ*x*. *x* + *1*) *0* = *k*
using *2*[*of k*] **by** *simp*

have *?f k* ∈ *T* **using** *a b* **unfolding** *T-def* **by** *auto*
hence *res* ∈ *T* **unfolding** *res-def find-irreducible-poly.simps Let-def 3* **by** *simp*
thus *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) *res degree res* = *n* **unfolding** *T-def* **by** *auto*
qed

lemma *monic-irred-poly-set-nonempty-finite*:
{*f*. *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) *f* ∧ *degree f* = *n*} ≠ {} (**is** *?R1*)
finite {*f*. *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) *f* ∧ *degree f* = *n*} (**is** *?R2*)
proof −
have *card T* > *0* **using** *T-card-gt-0* **by** *auto*
hence *T* ≠ {} *finite T* **using** *card-ge-0-finite* **by** *auto*
thus *?R1 ?R2* **unfolding** *T-def* **by** *auto*
qed

end

Returns m e such that $n = m^e$, where e is maximal.

definition *split-power* :: *nat* ⇒ *nat* × *nat*
where *split-power n* = (
let e = *last* (*filter* (λ*x*. *is-nth-power-nat x n*) (*1*#[*2*..<*floorlog 2 n*]))
in (*nth-root-nat e n*, *e*))

lemma *split-power-result*:
assumes (*x*,*e*) = *split-power n*
shows *n* = *x*^*e* ⋀*k*. *n* > *1* ⟹ *k*>*e* ⟹ ¬*is-nth-power k n*
proof −
define *es* **where** *es* = *filter* (λ*x*. *is-nth-power-nat x n*) (*1*#[*2*..<*floorlog 2 n*])

define *m* **where** *m* = *max 2* (*floorlog 2 n*)

have *0*: *x* < *m* **if** *that0*: *is-nth-power-nat x n n* > *1* **for** *x*
proof (*rule ccontr*)
assume *a*:¬(*x* < *m*)
obtain *y* **where** *n-def*:*n* = *y*^*x* **using** *that0 is-nth-power-def is-nth-power-nat-def* **by** *auto*
have *y* ≠ *0* **using** *that*(*2*) **unfolding** *n-def*
by (*metis* (*mono-tags*) *nat-power-eq-Suc-0-iff not-less0 power-0-left power-inject-exp*)
moreover have *y* ≠ *1* **using** *that*(*2*) **unfolding** *n-def* **by** *auto*
ultimately have *y-ge-2*: *y* ≥ *2* **by** *simp*
have *n* < *2*^*floorlog 2 n* **using** *that floorlog-bounds* **by** *simp*
also have ... ≤ *2*^*x* **using** *a* **unfolding** *m-def* **by** (*intro power-increasing*) *auto*
also have ... ≤ *y*^*x* **using** *y-ge-2* **by** (*intro power-mono*) *auto*
also have ... = *n* **using** *n-def* **by** *auto*
finally show *False* **by** *simp*
qed

have *1*: *m* = *2* **if** ¬(*n* > *1*)
proof −
have *floorlog 2 n* ≤ *2* **using** *that* **by** (*intro floorlog-leI*) *auto*
thus *?thesis* **unfolding** *m-def* **by** *auto*
qed

have *2*: *n* = *1* **if** *is-nth-power-nat 0 n* **using** *that* **by** (*simp add*: *is-nth-power-nat-code*)

have *set es* = {*x* ∈ *insert 1* {*2*..<*floorlog 2 n*}. *is-nth-power-nat x n*} **unfolding** *es-def* **by** *auto*
also have ... = {*x*. *x* ≠ *0* ∧ *x* < *m* ∧ *is-nth-power-nat x n*} **unfolding** *m-def* **by** *auto*
also have ... = {*x*. *is-nth-power-nat x n* ∧ (*n* > *1* ∨ *x* = *1*)}
using *0 1 2 zero-neq-one* **by** (*intro Collect-cong iffI conjI*) *fastforce*+
finally have *set-es*: *set es* = {*x*. *is-nth-power-nat x n* ∧ (*n* > *1* ∨ *x* = *1*)} **by** *simp*

have *is-nth-power-nat 1 n* **unfolding** *is-nth-power-nat-def* **by** *simp*
hence *es-ne*: *es* ≠ [] **unfolding** *es-def* **by** *auto*

have *sorted*: *sorted es* **unfolding** *es-def* **by** (*intro sorted-wrt-filter*) *simp*

have *e-def*: *e* = *last es* **and** *x-def*: *x* = *nth-root-nat e n*
using *assms* **unfolding** *es-def split-power-def* **by** (*simp-all add*:*Let-def*)

hence *e-in-set-es*: *e* ∈ *set es* **unfolding** *e-def* **using** *es-ne* **by** (*intro* *last-in-set*) *auto*

have *e-max*: $x \leq e$ **if** *that1*:$x \in$ *set es* **for** x
proof −
obtain k **where** $k <$ *length es* $x =$ *es* ! k **using** *that1* **by** (*metis in-set-conv-nth*)
moreover have $e =$ *es* ! (*length es* −*1*) **unfolding** *e-def* **using** *es-ne last-conv-nth* **by** *auto*
ultimately show *?thesis* **using** *sorted-nth-mono*[*OF sorted*] *es-ne* **by** *simp*
qed
have *3*:*is-nth-power-nat e n* ∧ (*1* < *n* ∨ *e* = *1*) **using** *e-in-set-es* **unfolding** *set-es* **by** *simp*
hence $e > 0$ **using** *2 zero-neq-one* **by** *fast*
thus $n = x\hat{\ }e$ **using** *3* **unfolding** *x-def* **using** *nth-root-nat-nth-power*
by (*metis is-nth-power-nat-code nth-root-nat-naive-code power-eq-0-iff*)
show ¬*is-nth-power k n* **if** $n > 1$ $k > e$ **for** k
proof (*rule ccontr*)
assume ¬(¬*is-nth-power k n*)
hence $k \in$ *set es* **using** *that* **unfolding** *set-es is-nth-power-nat-def* **by** *auto*
hence $k \leq e$ **using** *e-max* **by** *auto*
thus *False* **using** *that*(*2*) **by** *auto*
qed
qed

definition *not-perfect-power* :: *nat* ⇒ *bool*
where *not-perfect-power n* = ($n > 1$ ∧ (∀ *x k*. $n = x\ \hat{}\ k \longrightarrow k = 1$))

lemma *is-nth-power-from-multiplicities*:
assumes $n > (0::nat)$
assumes ⋀*p*. *Factorial-Ring.prime p* ⟹ *k dvd* (*multiplicity p n*)
shows *is-nth-power k n*
proof −
have $n = (\prod p \in$ *prime-factors n*. $p\hat{\ }$*multiplicity p n*) **using** *assms*(*1*)
by (*simp add*: *prod-prime-factors*)
also have ... = ($\prod p \in$ *prime-factors n*. $p\hat{\ }$((*multiplicity p n div k*)∗*k*))
by (*intro prod.cong arg-cong2*[**where** *f*=*power*] *dvd-div-mult-self*[*symmetric*] *refl assms*(*2*)) *auto*
also have ... = ($\prod p \in$ *prime-factors n*. $p\hat{\ }$(*multiplicity p n div k*))$\hat{\ }k$
unfolding *power-mult prod-power-distrib*[*symmetric*] **by** *simp*
finally have $n = (\prod p \in$ *prime-factors n*. $p\hat{\ }$(*multiplicity p n div k*))$\hat{\ }k$ **by** *simp*
thus *?thesis* **by** (*intro is-nth-powerI*) *simp*
qed

lemma *power-inj-aux*:

assumes *not-perfect-power a not-perfect-power b*
assumes $n > 0$ $m > n$
assumes $a \,\hat{}\, n = b \,\hat{}\, m$
shows *False*
proof −
define *s* **where** $s = gcd\ n\ m$
define *u* **where** $u = n\ div\ gcd\ n\ m$
define *t* **where** $t = m\ div\ gcd\ n\ m$

have *a-nz*: $a \neq 0$ **and** *b-nz*: $b \neq 0$ **using** *assms*(*1*,*2*) **unfolding** *not-perfect-power-def* **by** *auto*

have $gcd\ n\ m \neq 0$ **using** *assms* (*3*,*4*) **by** *simp*

then obtain *t u* **where** *n-def*: $n = t * s$ **and** *m-def*: $m = u * s$ **and** *cp*: *coprime t u*
using *gcd-coprime-exists* **unfolding** *s-def t-def u-def* **by** *blast*

have *s-gt-0*: $s > 0$ **and** *t-gt-0*: $t > 0$ **and** *u-gt-t*: $u > t$
using *assms*(*3*,*4*) **unfolding** *n-def m-def* **by** *auto*

have $(a \,\hat{}\, t) \,\hat{}\, s = (b \,\hat{}\, u) \,\hat{}\, s$ **using** *assms*(*5*) **unfolding** *n-def m-def power-mult* **by** *simp*
hence *0*: $a\hat{}t = b\hat{}u$ **using** *s-gt-0* **by** (*metis nth-root-nat-nth-power*)

have *u dvd multiplicity p a* **if** *Factorial-Ring.prime p* **for** *p*
proof −
have *prime-elem p* **using** *that* **by** *simp*
hence $t * multiplicity\ p\ a = u * multiplicity\ p\ b$
using *0 a-nz b-nz* **by** (*subst* (*1 2*) *prime-elem-multiplicity-power-distrib*[*symmetric*]) *auto*
hence $u\ dvd\ t * multiplicity\ p\ a$ **by** *simp*
thus *?thesis* **using** *cp coprime-commute coprime-dvd-mult-right-iff* **by** *blast*
qed

hence *is-nth-power u a* **using** *a-nz* **by** (*intro is-nth-power-from-multiplicities*) *auto*
moreover have $u > 1$ **using** *u-gt-t t-gt-0* **by** *auto*
ultimately show *False* **using** *assms*(*1*) **unfolding** *not-perfect-power-def is-nth-power-def* **by** *auto*
qed

Generalization of *prime-power-inj′*

lemma *power-inj*:
assumes *not-perfect-power a not-perfect-power b*
assumes $n > 0$ $m > 0$
assumes $a \,\hat{}\, n = b \,\hat{}\, m$
shows $a = b \wedge n = m$

proof −
consider (*a*) $n < m$ | (*b*) $m < n$ | (*c*) $n = m$ **by** *linarith*
thus *?thesis*
proof (*cases*)
case *a* **thus** *?thesis* **using** *assms power-inj-aux* **by** *auto*
next
case *b* **thus** *?thesis* **using** *assms power-inj-aux*[*OF assms*(*2*,*1*,*4*) *b*] **by** *auto*
next
case *c* **thus** *?thesis* **using** *assms* **by** (*simp add*: *power-eq-iff-eq-base*)
qed
qed

lemma *split-power-base-not-perfect*:
assumes $n > 1$
shows *not-perfect-power* (*fst* (*split-power n*))
proof (*rule ccontr*)
obtain *b e* **where** *be-def*: (*b*,*e*) = *split-power n* **by** (*metis surj-pair*)
have *n-def*:$n = b$ ^ e **and** *e-max*: $\bigwedge k.\ e < k \Longrightarrow \neg$ *is-nth-power k n*
using *assms split-power-result*[*OF be-def*] **by** *auto*

have *e-gt-0*: $e > 0$ **using** *assms* **unfolding** *n-def* **by** (*cases e*) *auto*

assume ¬*not-perfect-power* (*fst* (*split-power n*))
hence ¬*not-perfect-power b* **unfolding** *be-def*[*symmetric*] **by** *simp*
moreover have *b-gt-1*: $b > 1$ **using** *assms* **unfolding** *n-def*
by (*metis less-one nat-neq-iff nat-power-eq-Suc-0-iff power-0-left*)
ultimately obtain *k b′* **where** $k \neq 1$ **and** *b-def*: $b = b'$^k
unfolding *not-perfect-power-def* **by** *auto*
hence *k-gt-1*: $k > 1$ **using** *b-gt-1 nat-neq-iff* **by** *force*
have $n = b'$^($k*e$) **unfolding** *power-mult n-def b-def* **by** *auto*
moreover have $k*e > e$ **using** *k-gt-1 e-gt-0* **by** *simp*
hence ¬*is-nth-power* ($k*e$) *n* **using** *e-max* **by** *auto*
ultimately show *False* **unfolding** *is-nth-power-def* **by** *auto*
qed

lemma *prime-not-perfect*:
assumes *Factorial-Ring.prime p*
shows *not-perfect-power p*
proof −
have *k=1* **if** $p = x$^k **for** *x k* **using** *assms* **unfolding** *that* **by** (*simp add*:*prime-power-iff*)
thus *?thesis* **using** *prime-gt-1-nat*[*OF assms*] **unfolding** *not-perfect-power-def* **by** *auto*
qed

lemma *split-power-prime*:
assumes *Factorial-Ring.prime p* $n > 0$
shows *split-power* (p^n) = (*p*,*n*)

proof −
obtain *x e* **where** *xe*:(*x*,*e*) = *split-power* (*p*^*n*) **by** (*metis surj-pair*)

have *1* < *p*^*1* **using** *prime-gt-1-nat*[*OF assms*(*1*)] **by** *simp*
also have ... ≤ *p*^*n* **using** *assms*(*2*) *prime-gt-0-nat*[*OF assms*(*1*)] **by** (*intro power-increasing*) *auto*
finally have *0*:*p*^*n* > *1* **by** *simp*

have *not-perfect-power x*
using *split-power-base-not-perfect*[*OF 0*] **unfolding** *xe*[*symmetric*] **by** *simp*
moreover have *not-perfect-power p* **by** (*rule prime-not-perfect*[*OF assms*(*1*)])
moreover have *1*:*p*^*n* = *x*^*e* **using** *split-power-result*[*OF xe*] **by** *simp*
moreover have *e* > *0* **using** *0 1* **by** (*cases e*) *auto*
ultimately have *p*=*x* ∧ *n* = *e* **by** (*intro power-inj assms*(*2*))
thus *?thesis* **using** *xe* **by** *simp*
qed

definition *is-prime-power n* = (∃ *p k*. *Factorial-Ring.prime p* ∧ *k* > *0* ∧ *n* = *p*^*k*)

lemma *is-prime-powerI*:
assumes *prime p k* > *0*
shows *is-prime-power* (*p* ^ *k*)
unfolding *is-prime-power-def* **using** *assms* **by** *auto*

definition *GF* **where**
GF n = (
let (*p*,*k*) = *split-power n*;
f = *find-irreducible-poly p k*
in poly-mod-ring (*mod-ring p*) *f*)

definition GF_R **where**
GF_R *n* =
do {
let (*p*,*k*) = *split-power n*;
f ← *sample-irreducible-poly p k*;
return-spmf (*poly-mod-ring* (*mod-ring p*) (*fst f*))
}

lemma *GF-in-GF-R*:
assumes *is-prime-power n*
shows *GF n* ∈ *set-spmf* (GF_R *n*)
proof−
obtain *p k* **where** *n-def*: *n* = *p*^*k* **and** *p-prime*: *prime p* **and** *k-gt-0*: *k* > *0*

using *assms* **unfolding** *is-prime-power-def* **by** *blast*
have *pk-def*: $(p,k) = split\text{-}power\ n$
unfolding *n-def* **using** *split-power-prime*[*OF p-prime k-gt-0*] **by** *auto*
let *?S* = {*f*. *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) *f* ∧ *degree f* = *k*}

have *S-fin*: *finite ?S* **by** (*intro monic-irred-poly-set-nonempty-finite p-prime k-gt-0*)

have *find-irreducible-poly p k* ∈ *?S*
using *find-irreducible-poly-result*[*OF p-prime k-gt-0*] **by** *auto*
also have ... = *set-spmf* (*map-spmf fst* (*sample-irreducible-poly p k*))
unfolding *sample-irreducible-poly-result*[*OF p-prime k-gt-0*] *set-spmf-of-set-finite*[*OF S-fin*]
by *simp*
finally have *0*: *find-irreducible-poly p k* ∈ *set-spmf*(*map-spmf fst* (*sample-irreducible-poly p k*))
by *simp*

have *GF n* = *poly-mod-ring* (*mod-ring p*) (*find-irreducible-poly p k*)
unfolding *GF-def pk-def*[*symmetric*] **by** (*simp del*:*find-irreducible-poly.simps*)
also have ... ∈ *set-spmf* (*map-spmf fst* (*sample-irreducible-poly p k*)) ⤜ (λ*x*. {*poly-mod-ring* (*mod-ring p*) *x*})
using *0* **by** *force*
also have ... = *set-spmf* (GF_R *n*)
unfolding GF_R*-def pk-def*[*symmetric*] **by** (*simp add*:*set-bind-spmf comp-def bind-image*)
finally show *?thesis* **by** *simp*
qed

lemma *galois-field-random-1*:
assumes *is-prime-power n*
shows ⋀ω. ω ∈ *set-spmf* (GF_R *n*) ⟹ $enum_C$ ω ∧ $field_C$ ω ∧ *order* (*ring-of* ω) = *n*
and *lossless-spmf* (GF_R *n*)
proof −
let *?pred* = λω. $enum_C$ ω ∧ $field_C$ ω ∧ *order* (*ring-of* ω) = *n*

obtain *p k* **where** *n-def*: $n = p \hat{} k$ **and** *p-prime*: *prime p* **and** *k-gt-0*: $k > 0$
using *assms* **unfolding** *is-prime-power-def* **by** *blast*
let *?r* = (λ*f*. *poly-mod-ring* (*mod-ring p*) *f*)
let *?S* = {*f*. *monic-irreducible-poly* (*ring-of* (*mod-ring p*)) *f* ∧ *degree f* = *k*}

have *fc*: $field_C$ (*mod-ring p*) **by** (*intro mod-ring-is-field-c p-prime*)
have *ec*: $enum_C$ (*mod-ring p*) **by** (*intro mod-ring-is-enum-c*)

have *S-fin*: *finite ?S* **by** (*intro monic-irred-poly-set-nonempty-finite p-prime k-gt-0*)
have *S-ne*: *?S* ≠ {} **by** (*intro monic-irred-poly-set-nonempty-finite p-prime k-gt-0*)

have *pk-def*: (*p*,*k*) = *split-power n*
unfolding *n-def* **using** *split-power-prime*[*OF p-prime k-gt-0*] **by** *auto*

have *cond*: *?pred* (*?r x*) **if** *x* ∈ *?S* **for** *x*
proof −
have *order* (*ring-of* (*poly-mod-ring* (*mod-ring p*) *x*)) = *idx-size* (*poly-mod-ring* (*mod-ring p*) *x*)
using *enum-cD*[*OF enum-c-poly-mod-ring*[*OF ec field-c-imp-ring*[*OF fc*]]] **by** *simp*
also have ... = *p*^(*degree x*)
by (*simp add*:*poly-mod-ring-def Finite-Fields-Mod-Ring-Code.mod-ring-def*)
also have ... = *n* **unfolding** *n-def* **using** *that* **by** *simp*
finally have *order* (*ring-of* (*poly-mod-ring* (*mod-ring p*) *x*)) = *n* **by** *simp*

thus *?thesis* **using** *that*
by (*intro conjI enum-c-poly-mod-ring field-c-poly-mod-ring ec field-c-imp-ring fc*) *auto*
qed

have GF_R *n* = *bind-spmf* (*map-spmf fst* (*sample-irreducible-poly p k*)) (λ*x*. *return-spmf* (*?r x*))
unfolding GF_R*-def pk-def*[*symmetric*] *map-spmf-conv-bind-spmf* **by** *simp*
also have ... = *spmf-of-set ?S* ⤜ (λ*f*. *return-spmf* ((*?r f*)))
unfolding *sample-irreducible-poly-result*[*OF p-prime k-gt-0*] **by** (*simp*)
also have ... = *pmf-of-set ?S* ⤜ (λ*f*. *return-spmf* (*?r f*))
unfolding *spmf-of-pmf-pmf-of-set*[*OF S-fin S-ne*, *symmetric*] *spmf-of-pmf-def*
by (*simp add*:*bind-spmf-def bind-map-pmf*)
finally have *0*:GF_R *n* = *map-pmf* (*Some* ∘ *?r*) (*pmf-of-set ?S*) **by** (*simp add*:*comp-def map-pmf-def*)

show $enum_C$ ω ∧ $field_C$ ω ∧ *order* (*ring-of* ω) = *n* **if** ω ∈ *set-spmf* (GF_R *n*) **for** ω
proof −
have *Some* ω ∈ *set-pmf* (GF_R *n*) **unfolding** *in-set-spmf*[*symmetric*] **by** (*rule that*)
also have ... = (*Some* ∘ *?r*) ‘ *?S* **unfolding** *0 set-map-pmf set-pmf-of-set*[*OF S-ne S-fin*] **by** *simp*
finally have *Some* ω ∈ (*Some* ∘ *?r*) ‘ *?S* **by** *simp*
hence ω ∈ *?r* ‘ *?S* **by** *auto*

then obtain *x* **where** *x*:$x \in$ *?S* **and** *ω-def*:$\omega =$ *?r x* **by** *auto*
show *?thesis* **unfolding** *ω-def* **by** (*intro cond x*)
qed

have *None* $\notin$ *set-pmf*(*GF*$_R$ *n*) **unfolding** *0 set-map-pmf set-pmf-of-set*[*OF S-ne S-fin*] **by** *auto*
thus *lossless-spmf* (*GF*$_R$ *n*) **using** *lossless-iff-set-pmf-None* **by** *blast*
qed

lemma *galois-field*:
assumes *is-prime-power n*
shows *enum*$_C$ (*GF n*) *field*$_C$ (*GF n*) *order* (*ring-of* (*GF n*)) = *n*
using *galois-field-random-1*(*1*)[*OF assms*(*1*) *GF-in-GF-R*[*OF assms*(*1*)]]
by *auto*

lemma *lossless-imp-spmf-of-pmf*:
assumes *lossless-spmf M*
shows *spmf-of-pmf* (*map-pmf the M*) = *M*
proof −
have *spmf-of-pmf* (*map-pmf the M*) = *map-pmf* (*Some* ∘ *the*) *M*
unfolding *spmf-of-pmf-def* **by** (*simp add*: *pmf.map-comp*)
also have ... = *map-pmf id M*
using *assms* **unfolding** *lossless-iff-set-pmf-None*
by (*intro map-pmf-cong refl*) (*metis id-apply o-apply option.collapse*)
also have ... = *M* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *galois-field-random-2*:
assumes *is-prime-power n*
shows *map-spmf* (λω. *enum*$_C$ ω ∧ *field*$_C$ ω ∧ *order* (*ring-of* ω) = *n*) (*GF*$_R$ *n*) = *return-spmf True*
(**is** *?L* = -)
proof −
have *?L* = *map-spmf* (λω. *True*) (*GF*$_R$ *n*)
using *galois-field-random-1*[*OF assms*] **by** (*intro map-spmf-cong refl*) *auto*
also have ... = *map-pmf* (λω. *Some True*) (*GF*$_R$ *n*)
by (*subst lossless-imp-spmf-of-pmf*[*OF galois-field-random-1*(*2*)[*OF assms*],*symmetric*]) *simp*
also have ... = *return-spmf True* **unfolding** *map-pmf-def* **by** *simp*
finally show *?thesis* **by** *simp*
qed

lemma *bind-galois-field-cong*:
assumes *is-prime-power n*
assumes ⋀ω. *enum*$_C$ ω ⟹ *field*$_C$ ω ⟹ *order* (*ring-of* ω) = *n* ⟹ *f* ω = *g* ω
shows *bind-spmf* (*GF*$_R$ *n*) *f* = *bind-spmf* (*GF*$_R$ *n*) *g*

using *galois-field-random-1* (*1*)[*OF assms*(*1*)]
by (*intro bind-spmf-cong refl assms*(*2*)) *auto*

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.

[5] M. O. Rabin. Probabilistic algorithms in finite fields. *SIAM Journal on Computing*, 9(2):273–280, 1980.