Fundamental Theorem of Finitely Generated Abelian Groups

Joseph Thommes, Manuel Eberl

March 17, 2025

Abstract

This article deals with the formalisation of some group-theoretic results including the fundamental theorem of finitely generated abelian groups characterising the structure of these groups as a uniquely determined product of cyclic groups. Both the invariant factor decomposition and the primary decomposition are covered.

Additional work includes results about the direct product, the internal direct product and more group-theoretic lemmas.

Contents

1 Set Multiplication

```
theory Set_Multiplication
  imports "HOL-Algebra.Multiplicative_Group"
begin
```

This theory/section is of auxiliary nature and is mainly used to establish a connection between the set multiplication and the multiplication of subgroups via the *IDirProd* (although this particular notion is introduced later). However, as in every section of this entry, there are some lemmas that do not have any further usage in this entry, but are of interest just by themselves.

```
lemma (in group) set_mult_union:
  "A <#> (B ∪ C) = (A <#> B) ∪ (A <#> C)"
  ⟨proof⟩
```

```
lemma (in group) set_mult_card_single_el_eq:
  assumes "J ⊆ carrier G" "x ∈ carrier G"
  shows "card (l_coset G x J) = card J" ⟨proof⟩
```

We find an upper bound for the cardinality of a set product.

```
lemma (in group) set_mult_card_le:
  assumes "finite H" "H ⊆ carrier G" "J ⊆ carrier G"
  shows "card (H <#> J) ≤ card H * card J"
  ⟨proof⟩
```

```
lemma (in group) set_mult_finite:
  assumes "finite H" "finite J" "H ⊆ carrier G" "J ⊆ carrier G"
  shows "finite (H <#> J)"
  ⟨proof⟩
```

The next lemma allows us to later to derive that two finite subgroups J and H are complementary if and only if their product has the cardinality $|J| \cdot |H|$.

```
lemma (in group) set_mult_card_eq_impl_empty_inter:
  assumes "finite H" "finite J" "H ⊆ carrier G" "J ⊆ carrier G" "card
(H <#> J) = card H * card J"
```

```
  shows "⋀a b. ⟦a ∈ H; b ∈ H; a ≠ b⟧ ⟹ ((⊗) a ‘ J) ∩ ((⊗) b ‘ J)
= {}"
  ⟨proof⟩
```

```
lemma (in group) set_mult_card_eq_impl_empty_inter':
  assumes "finite H" "finite J" "H ⊆ carrier G" "J ⊆ carrier G" "card
(H <#> J) = card H * card J"
  shows "⋀a b. ⟦a ∈ H; b ∈ H; a ≠ b⟧ ⟹ (l_coset G a J) ∩ (l_coset
G b J) = {}"
  ⟨proof⟩
```

```
lemma (in comm_group) set_mult_comm:
  assumes "H ⊆ carrier G" "J ⊆ carrier G"
  shows "(H <#> J) = (J <#> H)"
  ⟨proof⟩
```

```
lemma (in group) set_mult_one_imp_inc:
  assumes "1 ∈ A" "A ⊆ carrier G" "B ⊆ carrier G"
  shows "B ⊆ (B <#> A)"
⟨proof⟩
```

In all cases, we know that the product of two sets is always contained in the subgroup generated by them.

```
lemma (in group) set_mult_subset_generate:
  assumes "A ⊆ carrier G" "B ⊆ carrier G"
  shows "A <#> B ⊆ generate G (A ∪ B)"
⟨proof⟩
```

In the case of subgroups, the set product is just the subgroup generated by both of the subgroups.

```
lemma (in comm_group) set_mult_eq_generate_subgroup:
  assumes "subgroup H G" "subgroup J G"
  shows "generate G (H ∪ J) = H <#> J" (is "?L = ?R")
⟨proof⟩
```

end

2 Miscellaneous group facts

```
theory Miscellaneous_Groups
  imports Set_Multiplication
begin
```

As the name suggests, this section contains several smaller lemmas about groups.

```
lemma (in subgroup) nat_pow_closed [simp,intro]: "a ∈ H ⟹ pow G a
(n::nat) ∈ H"
  ⟨proof⟩
```

lemma *nat_pow_modify_carrier:* "a [^]$_{G(|carrier := H|)}$ b = a [^]$_G$ (b::nat)"
⟨*proof*⟩

lemma (**in** *group*) *subgroup_card_dvd_group_ord:*
assumes "*subgroup H G*"
shows "*card H dvd order G*"
⟨*proof*⟩

lemma (**in** *group*) *subgroup_card_eq_order:*
assumes "*subgroup H G*"
shows "*card H = order (G(|carrier := H|))*"
⟨*proof*⟩

lemma (**in** *group*) *finite_subgroup_card_neq_0:*
assumes "*subgroup H G*" "*finite H*"
shows "*card H ≠ 0*"
⟨*proof*⟩

lemma (**in** *group*) *subgroup_order_dvd_group_order:*
assumes "*subgroup H G*"
shows "*order (G(|carrier := H|)) dvd order G*"
⟨*proof*⟩

lemma (**in** *group*) *sub_subgroup_dvd_card:*
assumes "*subgroup H G*" "*subgroup J G*" "*J ⊆ H*"
shows "*card J dvd card H*"
⟨*proof*⟩

lemma (**in** *group*) *inter_subgroup_dvd_card:*
assumes "*subgroup H G*" "*subgroup J G*"
shows "*card (H ∩ J) dvd card H*"
⟨*proof*⟩

lemma (**in** *group*) *subgroups_card_coprime_inter_card_one:*
assumes "*subgroup H G*" "*subgroup J G*" "*coprime (card H) (card J)*"
shows "*card (H ∩ J) = 1*"
⟨*proof*⟩

lemma (**in** *group*) *coset_neq_imp_empty_inter:*
assumes "*subgroup H G*" "*a ∈ carrier G*" "*b ∈ carrier G*"
shows "*H #> a ≠ H #> b ⟹ (H #> a) ∩ (H #> b) = {}*"
⟨*proof*⟩

lemma (**in** *comm_group*) *subgroup_is_comm_group:*
assumes "*subgroup H G*"
shows "*comm_group (G(|carrier := H|))*" ⟨*proof*⟩

lemma **(in** *group) pow_int_mod_ord:*
  assumes *[simp]:"a ∈ carrier G" "ord a ≠ 0"*
  shows *"a [^] (n::int) = a [^] (n mod ord a)"*
⟨*proof*⟩

lemma **(in** *group) pow_nat_mod_ord:*
  assumes *[simp]:"a ∈ carrier G" "ord a ≠ 0"*
  shows *"a [^] (n::nat) = a [^] (n mod ord a)"*
⟨*proof*⟩

lemma **(in** *group) ord_min:*
  assumes *"m ≥ 1" "x ∈ carrier G" "x [^] m =* **1***"*
  shows *"ord x ≤ m"*
  ⟨*proof*⟩

lemma **(in** *group) bij_betw_mult_left[intro]:*
  assumes *[simp]: "x ∈ carrier G"*
  shows *"bij_betw (λy. x ⊗ y) (carrier G) (carrier G)"*
  ⟨*proof*⟩

lemma **(in** *subgroup) inv_in_iff:*
  assumes *"x ∈ carrier G" "group G"*
  shows *"inv x ∈ H ⟷ x ∈ H"*
⟨*proof*⟩

lemma **(in** *subgroup) mult_in_cancel_left:*
  assumes *"y ∈ carrier G" "x ∈ H" "group G"*
  shows *"x ⊗ y ∈ H ⟷ y ∈ H"*
⟨*proof*⟩

lemma **(in** *subgroup) mult_in_cancel_right:*
  assumes *"x ∈ carrier G" "y ∈ H" "group G"*
  shows *"x ⊗ y ∈ H ⟷ x ∈ H"*
⟨*proof*⟩

lemma **(in** *group)*
  assumes *"x ∈ carrier G"* **and** *"x [^] n =* **1***"* **and** *"n > 0"*
  shows *ord_le: "ord x ≤ n"* **and** *ord_pos: "ord x > 0"*
⟨*proof*⟩

lemma **(in** *group) ord_conv_Least:*
  assumes *"x ∈ carrier G" "∃n::nat > 0. x [^] n =* **1***"*
  shows *"ord x = (LEAST n::nat. 0 < n ∧ x [^] n =* **1***)"*
⟨*proof*⟩

lemma (**in** *group*) *ord_conv_Gcd:*
  assumes *"x ∈ carrier G"*
  shows *"ord x = Gcd {n. x [^] n = 1}"*
  ⟨*proof*⟩

lemma (**in** *group*) *subgroup_ord_eq:*
  assumes *"subgroup H G" "x ∈ H"*
  shows *"group.ord (G⦇carrier := H⦈) x = ord x"*
  ⟨*proof*⟩

lemma (**in** *group*) *ord_FactGroup:*
  assumes *"subgroup P G" "group (G Mod P)"*
  shows *"order (G Mod P) * card P = order G"*
  ⟨*proof*⟩

lemma (**in** *group*) *one_is_same:*
  assumes *"subgroup H G"*
  shows "$\mathbf{1}_{G⦇carrier\ :=\ H⦈}$ = $\mathbf{1}$"
  ⟨*proof*⟩

lemma (**in** *group*) *kernel_FactGroup:*
  assumes *"P ◁ G"*
  shows *"kernel G (G Mod P) (λx. P #> x) = P"*
⟨*proof*⟩

lemma (**in** *group*) *sub_subgroup_coprime:*
  assumes *"subgroup H G" "subgroup J G" "coprime (card H) (card J)"*
  and *"subgroup sH G" "subgroup sJ G" "sH ⊆ H" "sJ ⊆ J"*
shows *"coprime (card sH) (card sJ)"*
  ⟨*proof*⟩

lemma (**in** *group*) *pow_eq_nat_mod:*
  assumes *"a ∈ carrier G" "a [^] n = a [^] m"*
  shows *"n mod (ord a) = m mod (ord a)"*
⟨*proof*⟩

lemma (**in** *group*) *pow_eq_int_mod:*
  fixes *n m::int*
  assumes *"a ∈ carrier G" "a [^] n = a [^] m"*
  shows *"n mod (ord a) = m mod (ord a)"*
⟨*proof*⟩

end

3 Generated Groups

theory *Generated_Groups_Extend*
  imports *Miscellaneous_Groups*
begin

This section extends the lemmas and facts about *generate*. Starting with a basic fact.

lemma **(in** *group)* *generate_sincl:*
  "A ⊆ generate G A"
  ⟨*proof*⟩

The following lemmas reflect some of the idempotence characteristics of *generate* and have proved useful at several occasions.

lemma **(in** *group)* *generate_idem:*
  assumes *"A ⊆ carrier G"*
  shows *"generate G (generate G A) = generate G A"*
  ⟨*proof*⟩

lemma **(in** *group)* *generate_idem':*
  assumes *"A ⊆ carrier G" "B ⊆ carrier G"*
  shows *"generate G (generate G A ∪ B) = generate G (A ∪ B)"*
⟨*proof*⟩

lemma **(in** *group)* *generate_idem'_right:*
  assumes *"A ⊆ carrier G" "B ⊆ carrier G"*
  shows *"generate G (A ∪ generate G B) = generate G (A ∪ B)"*
  ⟨*proof*⟩

lemma **(in** *group)* *generate_idem_Un:*
  assumes *"A ⊆ carrier G"*
  shows *"generate G (⋃x∈A. generate G {x}) = generate G A"*
⟨*proof*⟩

lemma **(in** *group)* *generate_idem_fUn:*
  assumes *"f A ⊆ carrier G"*
  shows *"generate G (⋃ {generate G {x} |x. x ∈ f A}) = generate G (f A)"*
⟨*proof*⟩

lemma **(in** *group)* *generate_idem_fim_Un:*
  assumes *"⋃(f ‘ A) ⊆ carrier G"*
  shows *"generate G (⋃S ∈ A. generate G (f S)) = generate G (⋃ {generate G {x} |x. x ∈ ⋃ (f ‘ A)})"*
⟨*proof*⟩

The following two rules allow for convenient proving of the equality of two generated sets.

lemma **(in** *group)* *generate_eqI:*
  assumes *"A ⊆ carrier G" "B ⊆ carrier G" "A ⊆ generate G B" "B ⊆ generate G A"*
  shows *"generate G A = generate G B"*
  ⟨*proof*⟩

lemma (**in** *group*) *generate_one_switched_eqI:*
  assumes *"A ⊆ carrier G" "a ∈ A" "B = (A - {a}) ∪ {b}"*
  and *"b ∈ generate G A" "a ∈ generate G B"*
  shows *"generate G A = generate G B"*
⟨*proof*⟩

lemma (**in** *group*) *generate_subset_eqI:*
  assumes *"A ⊆ carrier G" "B ⊆ A" "A - B ⊆ generate G B"*
  shows *"generate G A = generate G B"*
⟨*proof*⟩

Some smaller lemmas about *generate*.

lemma (**in** *group*) *generate_subset_change_eqI:*
  assumes *"A ⊆ carrier G" "B ⊆ carrier G" "C ⊆ carrier G" "generate G A = generate G B"*
  shows *"generate G (A ∪ C) = generate G (B ∪ C)"*
  ⟨*proof*⟩

lemma (**in** *group*) *generate_subgroup_id:*
  assumes *"subgroup H G"*
  shows *"generate G H = H"*
  ⟨*proof*⟩

lemma (**in** *group*) *generate_consistent':*
  assumes *"subgroup H G" "A ⊆ H"*
  shows *"∀x ∈ A. generate G {x} = generate (G⦇carrier := H⦈) {x}"*
  ⟨*proof*⟩

lemma (**in** *group*) *generate_singleton_one:*
  assumes *"generate G {a} = {**1**}"*
  shows *"a = **1**"*
  ⟨*proof*⟩

lemma (**in** *group*) *generate_inv_eq:*
  assumes *"a ∈ carrier G"*
  shows *"generate G {a} = generate G {inv a}"*
  ⟨*proof*⟩

lemma (**in** *group*) *generate_eq_imp_subset:*
  assumes *"generate G A = generate G B"*
  shows *"A ⊆ generate G B"*
  ⟨*proof*⟩

The neutral element does not play a role when generating a subgroup.

lemma (**in** *group*) *generate_one_irrel:*
  *"generate G A = generate G (A ∪ {**1**})"*
⟨*proof*⟩

lemma (**in** *group*) *generate_one_irrel':*

```
  "generate G A = generate G (A - {1})"
  ⟨proof⟩
```

Also, we can express the subgroup generated by a singleton with finite order using just its powers up to its order.

```
lemma (in group) generate_nat_pow:
  assumes "ord a ≠ 0" "a ∈ carrier G"
  shows "generate G {a} = {a [^] k |k. k ∈ {0..ord a - 1}}"
  ⟨proof⟩

lemma (in group) generate_nat_pow':
  assumes "ord a ≠ 0" "a ∈ carrier G"
  shows "generate G {a} = {a [^] k |k. k ∈ {1..ord a}}"
⟨proof⟩

end
```

4 Auxiliary lemmas

```
theory General_Auxiliary
  imports Complex_Main
          "HOL-Algebra.IntRing"
          "HOL.Rings"
begin

lemma inter_imp_subset: "A ∩ B = A ⟹ A ⊆ B"
  ⟨proof⟩

lemma card_inter_eq:
  assumes "finite A" "card (A ∩ B) = card A"
  shows "A ⊆ B"
⟨proof⟩

lemma coprime_eq_empty_prime_inter:
  assumes "(n::nat) ≠ 0" "m ≠ 0"
  shows "coprime n m ⟷ (prime_factors n) ∩ (prime_factors m) = {}"
⟨proof⟩

lemma prime_factors_Prod:
  assumes "finite S" "⋀a. a ∈ S ⟹ f a ≠ 0"
  shows "prime_factors (prod f S) = ⋃(prime_factors ‘ f ‘ S)"
  ⟨proof⟩

lemma lcm_is_Min_multiple_nat:
  assumes "c ≠ 0" "(a::nat) dvd c" "(b::nat) dvd c"
  shows "c ≥ lcm a b"
  ⟨proof⟩

lemma diff_prime_power_imp_coprime:
```

assumes *"p ≠ q" "Factorial_Ring.prime (p::nat)" "Factorial_Ring.prime q"*
shows *"coprime (p ^ (n::nat)) (q ^ m)"*
⟨*proof*⟩

lemma *"finite (prime_factors x)"*
⟨*proof*⟩

lemma *card_ge_1_two_diff:*
assumes *"card A > 1"*
obtains *x y* **where** *"x ∈ A" "y ∈ A" "x ≠ y"*
⟨*proof*⟩

lemma *infinite_two_diff:*
assumes *"infinite A"*
obtains *x y* **where** *"x ∈ A" "y ∈ A" "x ≠ y"*
⟨*proof*⟩

lemma *Inf_le:*
"Inf A ≤ x" **if** *"x ∈ (A::nat set)"* **for** *x*
⟨*proof*⟩

lemma *switch_elem_card_le:*
assumes *"a ∈ A"*
shows *"card (A - {a} ∪ {b}) ≤ card A"*
⟨*proof*⟩

lemma *pairwise_coprime_dvd:*
assumes *"finite A" "pairwise coprime A" "(n::nat) = prod id A" "∀a∈A. a dvd j"*
shows *"n dvd j"*
⟨*proof*⟩

lemma *pairwise_coprime_dvd':*
assumes *"finite A" "⋀i j. ⟦i ∈ A; j ∈ A; i ≠ j⟧ ⟹ coprime (f i) (f j)"*
"(n::nat) = prod f A" "∀a∈A. f a dvd j"
shows *"n dvd j"*
⟨*proof*⟩

lemma *transp_successively_remove1:*
assumes *"transp f" "successively f l"*
shows *"successively f (remove1 a l)"* ⟨*proof*⟩

lemma *exp_one_2pi_iff:*
fixes *x::real* **shows** *"exp (2 * of_real pi * i * x) = 1 ⟷ x ∈ ℤ"*
⟨*proof*⟩

lemma *of_int_divide_in_Ints_iff:*
  assumes *"b ≠ 0"*
  shows *"(of_int a / of_int b :: 'a :: field_char_0) ∈ ℤ ⟷ b dvd a"*
⟨*proof*⟩

lemma *of_nat_divide_in_Ints_iff:*
  assumes *"b ≠ 0"*
  shows *"(of_nat a / of_nat b :: 'a :: field_char_0) ∈ ℤ ⟷ b dvd a"*
  ⟨*proof*⟩

lemma *true_nth_unity_root:*
  fixes *n::nat*
  obtains *x::complex* **where** *"x ^ n = 1" "⋀m. ⟦0<m; m<n⟧ ⟹ x ^ m ≠ 1"*
⟨*proof*⟩

lemma *finite_bij_betwI:*
  assumes *"finite A" "finite B" "inj_on f A" "f ∈ A → B" "card A = card B"*
  shows *"bij_betw f A B"*
⟨*proof*⟩

lemma *powi_mod:*
  "x powi m = x powi (m mod n)" **if** *"x ^ n = 1" "n > 0"* **for** *x::complex* **and** *m::int*
⟨*proof*⟩

lemma *Sigma_insert: "Sigma (insert x A) B = (λy. (x, y)) ` B x ∪ Sigma A B"*
  ⟨*proof*⟩

end

5 Internal direct product

theory *IDirProds*
  imports *Generated_Groups_Extend General_Auxiliary*
begin

5.1 Complementarity

We introduce the notion of complementarity, that plays a central role in the internal direct group product and prove some basic properties about it.

definition (**in** *group*) *complementary :: "'a set ⇒ 'a set ⇒ bool"* **where**

```
  "complementary H1 H2 ⟷ H1 ∩ H2 = {1}"
```

```
lemma (in group) complementary_symm: "complementary A B ⟷ complementary
B A"
  ⟨proof⟩
```

```
lemma (in group) subgroup_carrier_complementary:
  assumes "complementary H J" "subgroup I (G(|carrier := H|))" "subgroup
K (G(|carrier := J|))"
  shows "complementary I K"
⟨proof⟩
```

```
lemma (in group) subgroup_subset_complementary:
  assumes "subgroup H G" "subgroup J G" "subgroup I G"
  and "I ⊆ J" "complementary H J"
shows "complementary H I"
  ⟨proof⟩
```

```
lemma (in group) complementary_subgroup_iff:
  assumes "subgroup H G"
  shows "complementary A B ⟷ group.complementary (G(|carrier := H|))
A B"
⟨proof⟩
```

```
lemma (in group) subgroups_card_coprime_imp_compl:
  assumes "subgroup H G" "subgroup J G" "coprime (card H) (card J)"
  shows "complementary H J" ⟨proof⟩
```

```
lemma (in group) prime_power_complementary_groups:
  assumes "Factorial_Ring.prime p" "Factorial_Ring.prime q" "p ≠ q"
  and "subgroup P G" "card P = p ^ x"
  and "subgroup Q G" "card Q = q ^ y"
  shows "complementary P Q"
⟨proof⟩
```

With the previous work from the theory about set multiplication we can characterize complementarity of two subgroups in abelian groups by the cardinality of their product.

```
lemma (in comm_group) compl_imp_diff_cosets:
  assumes "subgroup H G" "subgroup J G" "finite H" "finite J"
  and "complementary H J"
  shows "⋀a b. ⟦a ∈ J; b ∈ J; a ≠ b⟧ ⟹ (H #> a) ≠ (H #> b)"
⟨proof⟩
```

```
lemma (in comm_group) finite_sub_card_eq_mult_imp_comp:
  assumes "subgroup H G" "subgroup J G" "finite H" "finite J"
  and "card (H <#> J) = (card J * card H)"
  shows "complementary H J"
  ⟨proof⟩
```

lemma (**in** *comm_group*) *finite_sub_comp_imp_card_eq_mult:*
  assumes *"subgroup H G" "subgroup J G" "finite H" "finite J"*
  and *"complementary H J"*
shows *"card (H <#> J) = card J * card H"*
⟨*proof*⟩

lemma (**in** *comm_group*) *finite_sub_comp_iff_card_eq_mult:*
  assumes *"subgroup H G" "subgroup J G" "finite H" "finite J"*
  shows *"card (H <#> J) = card J * card H ⟷ complementary H J"*
  ⟨*proof*⟩

5.2 *IDirProd* - binary internal direct product

We introduce the internal direct product formed by two subgroups (so in its binary form).

definition *IDirProd :: "('a, 'b) monoid_scheme ⇒ 'a set ⇒ 'a set ⇒ 'a set"* **where**
  "IDirProd G Y Z = generate G (Y ∪ Z)"

Some trivial lemmas about the binary internal direct product.

lemma (**in** *group*) *IDirProd_comm:*
  "IDirProd G A B = IDirProd G B A"
  ⟨*proof*⟩

lemma (**in** *group*) *IDirProd_empty_right:*
  assumes *"A ⊆ carrier G"*
  shows *"IDirProd G A {} = generate G A"*
  ⟨*proof*⟩

lemma (**in** *group*) *IDirProd_empty_left:*
  assumes *"A ⊆ carrier G"*
  shows *"IDirProd G {} A = generate G A"*
  ⟨*proof*⟩

lemma (**in** *group*) *IDirProd_one_right:*
  assumes *"A ⊆ carrier G"*
  shows *"IDirProd G A {1} = generate G A"*
  ⟨*proof*⟩

lemma (**in** *group*) *IDirProd_one_left:*
  assumes *"A ⊆ carrier G"*
  shows *"IDirProd G {1} A = generate G A"*
  ⟨*proof*⟩

lemma (**in** *group*) *IDirProd_is_subgroup:*
  assumes *"Y ⊆ carrier G" "Z ⊆ carrier G"*
  shows *"subgroup (IDirProd G Y Z) G"*

⟨proof⟩

Using the theory about set multiplication we can also show the connection of the underlying set in the internal direct product with the set multiplication in the case of an abelian group. Together with the facts about complementarity and the set multiplication we can characterize complementarity by the cardinality of the internal direct product and vice versa.

lemma (**in** *comm_group*) *IDirProd_eq_subgroup_mult:*
  assumes *"subgroup H G" "subgroup J G"*
  shows *"IDirProd G H J = H <#> J"*
  ⟨proof⟩

lemma (**in** *comm_group*) *finite_sub_comp_iff_card_eq_IDirProd:*
  assumes *"subgroup H G" "subgroup J G" "finite H" "finite J"*
  shows *"card (IDirProd G H J) = card J * card H ⟷ complementary H J"*
  ⟨proof⟩

5.3 *IDirProds* - indexed internal direct product

The indexed version of the internal direct product acting on a family of subgroups.

definition *IDirProds :: "('a, 'b) monoid_scheme ⇒ ('c ⇒ 'a set) ⇒ 'c set ⇒ 'a set"* **where**
  "IDirProds G S I = generate G (⋃(S ‘ I))"

Lemmas about the indexed internal direct product.

lemma (**in** *group*) *IDirProds_incl:*
  assumes *"i ∈ I"*
  shows *"S i ⊆ IDirProds G S I"*
  ⟨proof⟩

lemma (**in** *group*) *IDirProds_empty:*
  "IDirProds G S {} = {𝟭}"
  ⟨proof⟩

lemma (**in** *group*) *IDirProds_is_subgroup:*
  assumes *"⋃(S ‘ I) ⊆ (carrier G)"*
  shows *"subgroup (IDirProds G S I) G"*
  ⟨proof⟩

lemma (**in** *group*) *IDirProds_subgroup_id: "subgroup (S i) G ⟹ IDirProds G S {i} = S i"*
  ⟨proof⟩

lemma (**in** *comm_group*) *IDirProds_Un:*
  assumes *"∀i∈A. subgroup (S i) G" "∀j∈B. subgroup (S j) G"*

shows "IDirProds G S (A ∪ B) = IDirProds G S A <#> IDirProds G S B"
⟨proof⟩

lemma (**in** comm_group) IDirProds_finite:
assumes "finite I" "∀i∈I. subgroup (S i) G" "∀i∈I. finite (S i)"
shows "finite (IDirProds G S I)" ⟨proof⟩

lemma (**in** comm_group) IDirProds_compl_imp_compl:
assumes "∀i ∈ I. subgroup (S i) G" **and** "subgroup H G"
assumes "complementary H (IDirProds G S I)" "i ∈ I"
shows "complementary H (S i)"
⟨proof⟩

Using the knowledge about the binary internal direct product, we can - in case that all subgroups in the family have coprime orders - also derive the cardinality of the indexed internal direct product.

lemma (**in** comm_group) IDirProds_card:
assumes "finite I" "∀i∈I. subgroup (S i) G"
"∀i∈I. finite (S i)" "pairwise (λx y. coprime (card (S x)) (card (S y))) I"
shows "card (IDirProds G S I) = (∏i ∈ I. card (S i))" ⟨proof⟩

5.4 Complementary family of subgroups

The notion of a complementary family is introduced. Note that the subgroups are complementary not only to the other subgroups but to the product of the other subgroups.

definition (**in** group) compl_fam :: "('c ⇒ 'a set) ⇒ 'c set ⇒ bool" **where**
"compl_fam S I = (∀i ∈ I. complementary (S i) (IDirProds G S (I - {i})))"

Some lemmas about compl_fam.

lemma (**in** group) compl_fam_empty[simp]: "compl_fam S {}"
⟨proof⟩

lemma (**in** group) compl_fam_cong:
assumes "compl_fam (f ∘ g) A" "inj_on g A"
shows "compl_fam f (g ` A)"
⟨proof⟩

We now connect compl_fam with generate as this will be its main application.

lemma (**in** comm_group) compl_fam_imp_generate_inj:
assumes "gs ⊆ carrier G" "compl_fam (λg. generate G {g}) gs"
shows "inj_on (λg. generate G {g}) gs"
⟨proof⟩

lemma (**in** *comm_group*) *compl_fam_generate_subset:*
  assumes *"compl_fam (λg. generate G {g}) gs"*
    "gs ⊆ carrier G" "A ⊆ gs"
  shows *"compl_fam (λg. generate G {g}) A"*
⟨*proof*⟩

5.5 *is_idirprod*

In order to identify a group as the internal direct product of a family of subgroups, they all have to be normal subgroups, complementary to the product of the rest of the subgroups and generate all of the group - this is captured in the definition of *is_idirprod*.

definition (**in** *group*) *is_idirprod :: "'a set ⇒ ('c ⇒ 'a set) ⇒ 'c set ⇒ bool"* **where**
  "is_idirprod A S I = ((∀i ∈ I. S i ◁ G) ∧ A = IDirProds G S I ∧ compl_fam S I)"

Very basic lemmas about *is_idirprod*.

lemma (**in** *comm_group*) *is_idirprod_subgroup_suffices:*
  assumes *"A = IDirProds G S I" "∀i∈I. subgroup (S i) G" "compl_fam S I"*
  shows *"is_idirprod A S I"*
  ⟨*proof*⟩

lemma (**in** *comm_group*) *is_idirprod_generate:*
  assumes *"A = generate G gs" "gs ⊆ carrier G" "compl_fam (λg. generate G {g}) gs"*
  shows *"is_idirprod A (λg. generate G {g}) gs"*
⟨*proof*⟩

lemma (**in** *comm_group*) *is_idirprod_imp_compl_fam[simp]:*
  assumes *"is_idirprod A S I"*
  shows *"compl_fam S I"*
  ⟨*proof*⟩

lemma (**in** *comm_group*) *is_idirprod_generate_imp_generate[simp]:*
  assumes *"is_idirprod A (λg. generate G {g}) gs"*
  shows *"A = generate G gs"*
⟨*proof*⟩

end

6 Finite Product

theory *Finite_Product_Extend*
  imports *IDirProds*
begin

In this section, some general facts about *finprod* as well as some tailored for the rest of this entry are proven.

It is often needed to split a product in a single factor and the rest. Thus these two lemmas.

lemma **(in** *comm_group)* *finprod_minus:*
  assumes *"a ∈ A" "f ∈ A → carrier G" "finite A"*
  shows *"finprod G f A = f a ⊗ finprod G f (A - {a})"*
⟨*proof*⟩

lemma **(in** *comm_group)* *finprod_minus_symm:*
  assumes *"a ∈ A" "f ∈ A → carrier G" "finite A"*
  shows *"finprod G f A = finprod G f (A - {a}) ⊗ f a"*
⟨*proof*⟩

This makes it very easy to show the following trivial fact.

lemma **(in** *comm_group)* *finprod_singleton:*
  assumes *"f x ∈ carrier G" "finprod G f {x} = a"*
  shows *"f x = a"*
⟨*proof*⟩

The finite product is consistent and closed concerning subgroups.

lemma **(in** *comm_group)* *finprod_subgroup:*
  assumes *"f ∈ S → H" "subgroup H G"*
  shows *"finprod G f S = finprod (G(|carrier := H|)) f S"*
⟨*proof*⟩

lemma **(in** *comm_group)* *finprod_closed_subgroup:*
  assumes *"subgroup H G" "f ∈ A → H"*
  shows *"finprod G f A ∈ H"*
  ⟨*proof*⟩

It also does not matter if we exponentiate all elements taking part in the product or the result of the product.

lemma **(in** *comm_group)* *finprod_exp:*
  assumes *"A ⊆ carrier G" "f ∈ A → carrier G"*
  shows *"(finprod G f A) [^] (k::int) = finprod G ((λa. a [^] k) ∘ f) A"*
  ⟨*proof*⟩

Some lemmas concerning different combinations of functions in the usage of *finprod*.

lemma **(in** *comm_group)* *finprod_cong_split:*
  assumes *"⋀a. a ∈ A ⟹ f a ⊗ g a = h a"*
  and *"f ∈ A → carrier G" "g ∈ A → carrier G" "h ∈ A → carrier G"*
  shows *"finprod G h A = finprod G f A ⊗ finprod G g A"* ⟨*proof*⟩

lemma **(in** *comm_group)* *finprod_comp:*

assumes *"inj_on g A" "(f ∘ g) ‘ A ⊆ carrier G"*
shows *"finprod G f (g ‘ A) = finprod G (f ∘ g) A"*
⟨*proof*⟩

The subgroup generated by a set of generators (in an abelian group) is exactly the set of elements that can be written as a finite product using only powers of these elements.

lemma (**in** *comm_group*) *generate_eq_finprod_PiE_image*:
assumes *"finite gs" "gs ⊆ carrier G"*
shows *"generate G gs = (λx. finprod G x gs) ‘ Pi_E gs (λa. generate G {a})"* (**is** *"?g = ?fp"*)
⟨*proof*⟩

lemma (**in** *comm_group*) *generate_eq_finprod_Pi_image*:
assumes *"finite gs" "gs ⊆ carrier G"*
shows *"generate G gs = (λx. finprod G x gs) ‘ Pi gs (λa. generate G {a})"* (**is** *"?g = ?fp"*)
⟨*proof*⟩

lemma (**in** *comm_group*) *generate_eq_finprod_Pi_int_image*:
assumes *"finite gs" "gs ⊆ carrier G"*
shows *"generate G gs = (λx. finprod G (λg. g [^] x g) gs) ‘ Pi gs (λ_. (UNIV::int set))"*
⟨*proof*⟩

lemma (**in** *comm_group*) *IDirProds_eq_finprod_PiE*:
assumes *"finite I" "⋀i. i ∈ I ⟹ subgroup (S i) G"*
shows *"IDirProds G S I = (λx. finprod G x I) ‘ (Pi_E I S)"* (**is** *"?DP = ?fp"*)
⟨*proof*⟩

lemma (**in** *comm_group*) *IDirProds_eq_finprod_Pi*:
assumes *"finite I" "⋀i. i ∈ I ⟹ subgroup (S i) G"*
shows *"IDirProds G S I = (λx. finprod G x I) ‘ (Pi I S)"* (**is** *"?DP = ?fp"*)
⟨*proof*⟩

If we switch one element from a set of generators, the generated set stays the same if both elements can be generated from the others together with the switched element respectively.

lemma (**in** *comm_group*) *generate_one_switched_exp_eqI*:
assumes *"A ⊆ carrier G" "a ∈ A" "B = (A - {a}) ∪ {b}"*
and *"f ∈ A → (UNIV::int set)" "g ∈ B → (UNIV::int set)"*
and *"a = finprod G (λx. x [^] g x) B" "b = finprod G (λx. x [^] f x) A"*
shows *"generate G A = generate G B"*
⟨*proof*⟩

We can characterize a complementary family of subgroups when the only way to form the neutral element as a product of picked elements from each subgroup is to pick the neutral element from each subgroup.

lemma **(in** *comm_group)* *compl_fam_imp_triv_finprod:*
  assumes *"compl_fam S I" "finite I" "⋀i. i ∈ I ⟹ subgroup (S i) G"*
  and *"finprod G f I =* **1***" "f ∈ Pi I S"*
  shows *"∀i∈I. f i =* **1***"*
⟨*proof*⟩

lemma **(in** *comm_group)* *triv_finprod_imp_compl_fam:*
  assumes *"finite I" "⋀i. i ∈ I ⟹ subgroup (S i) G"*
  and *"∀f ∈ Pi I S. finprod G f I =* **1** *⟶ (∀i∈I. f i =* **1***)"*
  shows *"compl_fam S I"*
⟨*proof*⟩

lemma **(in** *comm_group)* *triv_finprod_iff_compl_fam_Pi:*
  assumes *"finite I" "⋀i. i ∈ I ⟹ subgroup (S i) G"*
  shows *"compl_fam S I ⟷ (∀f ∈ Pi I S. finprod G f I =* **1** *⟶ (∀i∈I. f i =* **1***))"*
  ⟨*proof*⟩

lemma **(in** *comm_group)* *triv_finprod_iff_compl_fam_PiE:*
  assumes *"finite I" "⋀i. i ∈ I ⟹ subgroup (S i) G"*
  shows *"compl_fam S I ⟷ (∀f ∈* Pi_E *I S. finprod G f I =* **1** *⟶ (∀i∈I. f i =* **1***))"*
⟨*proof*⟩

The finite product also distributes when nested.

lemma **(in** *comm_monoid)* *finprod_Sigma:*
  assumes *"finite A" "⋀x. x ∈ A ⟹ finite (B x)"*
  assumes *"⋀x y. x ∈ A ⟹ y ∈ B x ⟹ g x y ∈ carrier G"*
  shows *"(⨂x∈A. ⨂y∈B x. g x y) = (⨂z∈Sigma A B. case z of (x, y) ⇒ g x y)"*
  ⟨*proof*⟩

With the now proven facts, we are able to provide criterias to inductively construct a group that is the internal direct product of a set of generators.

lemma **(in** *comm_group)* *idirprod_generate_ind:*
  assumes *"finite gs" "gs ⊆ carrier G" "g ∈ carrier G"*
    "is_idirprod (generate G gs) (λg. generate G {g}) gs"
    "complementary (generate G {g}) (generate G gs)"
  shows *"is_idirprod (generate G (gs ∪ {g})) (λg. generate G {g}) (gs ∪ {g})"*
⟨*proof*⟩

end

7 Group Homomorphisms

```
theory Group_Hom
  imports Set_Multiplication
begin
```

This section extends the already existing library about group homomorphisms in HOL-Algebra by some useful lemmas. These were mainly inspired by the needs that arised throughout the other proofs.

```
lemma (in group_hom) generate_hom:
  assumes "A ⊆ carrier G"
  shows "h ‘ (generate G A) = generate H (h ‘ A)"
  ⟨proof⟩
```

For two elements with the same image we can find an element in the kernel that maps one of the two elements on the other by multiplication.

```
lemma (in group_hom) kernel_assoc_elem:
  assumes "x ∈ carrier G" "y ∈ carrier G" "h x = h y"
  obtains z where "x = y ⊗_G z" "z ∈ kernel G H h"
⟨proof⟩
```

This can then be used to characterize the pre-image of a set A under homomorphism as a product of A itself with the kernel of the homomorphism.

```
lemma (in group_hom) vimage_eq_set_mult_kern_right:
  assumes "A ⊆ carrier G"
  shows "{x ∈ carrier G. h x ∈ h ‘ A} = A <#> kernel G H h"
⟨proof⟩

lemma (in group_hom) vimage_subset_generate_kern:
  assumes "A ⊆ carrier G"
  shows "{x ∈ carrier G. h x ∈ h ‘ A} ⊆ generate G (A ∪ kernel G H h)"
  ⟨proof⟩
```

The preimage of a subgroup under a homomorphism is also a subgroup.

```
lemma (in group_hom) subgroup_vimage_is_subgroup:
  assumes "subgroup I H"
  shows "subgroup {x ∈ carrier G. h x ∈ I} G" (is "subgroup ?J G")
⟨proof⟩

lemma (in group_hom) iso_kernel:
  assumes "h ∈ iso G H"
  shows "kernel G H h = {𝟭_G}"
  ⟨proof⟩
lemma (in group_hom) induced_group_hom_same_group:
  assumes "subgroup I G"
  shows "group_hom (G ⦇ carrier := I ⦈) H h"
⟨proof⟩
```

The order of an element under a homomorphism divides the order of the element.

```
lemma (in group_hom) hom_ord_dvd_ord:
  assumes "a ∈ carrier G"
  shows "H.ord (h a) dvd G.ord a"
⟨proof⟩
```

In particular, this implies that the image of an element with a finite order also will have a finite order.

```
lemma (in group_hom) finite_ord_stays_finite:
  assumes "a ∈ carrier G" "G.ord a ≠ 0"
  shows "H.ord (h a) ≠ 0"
  ⟨proof⟩
```

For injective homomorphisms, the order stays the same.

```
lemma (in group_hom) inj_imp_ord_eq:
  assumes "a ∈ carrier G" "inj_on h (carrier G)" "G.ord a ≠ 0"
  shows "H.ord (h a) = G.ord a"
⟨proof⟩

lemma (in group_hom) one_in_kernel:
  "1 ∈ kernel G H h"
  ⟨proof⟩

lemma hom_in_carr:
  assumes "f ∈ hom G H"
  shows "⋀x. x ∈ carrier G ⟹ f x ∈ carrier H"
  ⟨proof⟩

lemma iso_in_carr:
  assumes "f ∈ iso G H"
  shows "⋀x. x ∈ carrier G ⟹ f x ∈ carrier H"
  ⟨proof⟩

lemma triv_iso:
  assumes "group G" "group H" "carrier G = {1_G}" "carrier H = {1_H}"
  shows "G ≅ H"
⟨proof⟩
```

The cardinality of the image of a group homomorphism times the cardinality of its kernel is equal to the group order. This is basically another form of Lagrange's theorem.

```
lemma (in group_hom) image_kernel_product: "card (h ` (carrier G)) *
card (kernel G H h) = order G"
⟨proof⟩

end
```

8 Finite and cyclic groups

```
theory Finite_And_Cyclic_Groups
  imports Group_Hom Generated_Groups_Extend General_Auxiliary
begin
```

8.1 Finite groups

We define the notion of finite groups and prove some trivial facts about them.

```
locale finite_group = group +
  assumes fin[simp]: "finite (carrier G)"


lemma (in finite_group) ord_pos:
  assumes "x ∈ carrier G"
  shows   "ord x > 0"
  ⟨proof⟩

lemma (in finite_group) order_gt_0 [simp,intro]: "order G > 0"
  ⟨proof⟩

lemma (in finite_group) finite_ord_conv_Least:
  assumes "x ∈ carrier G"
  shows "ord x = (LEAST n::nat. 0 < n ∧ x [^] n = 𝟭)"
  ⟨proof⟩

lemma (in finite_group) non_trivial_group_ord_gr_1:
  assumes "carrier G ≠ {𝟭}"
  shows "∃e ∈ carrier G. ord e > 1"
⟨proof⟩


lemma (in finite_group) max_order_elem:
  obtains a where "a ∈ carrier G" "∀x ∈ carrier G. ord x ≤ ord a"
⟨proof⟩

lemma (in finite_group) iso_imp_finite:
  assumes "G ≅ H" "group H"
  shows "finite_group H"
⟨proof⟩

lemma (in finite_group) finite_FactGroup:
  assumes "H ◁ G"
  shows "finite_group (G Mod H)"
⟨proof⟩

lemma (in finite_group) bigger_subgroup_is_group:
  assumes "subgroup H G" "card H ≥ order G"
```

```
  shows "H = carrier G"
  ⟨proof⟩
```

All generated subgroups of a finite group are obviously also finite.

```
lemma (in finite_group) finite_generate:
  assumes "A ⊆ carrier G"
  shows "finite (generate G A)"
  ⟨proof⟩
```

We also provide an induction rule for finite groups inspired by Manuel Eberl's AFP entry "Dirichlet L-Functions and Dirichlet's Theorem" and the contained theory "Group_Adjoin". A property that is true for a subgroup generated by some set and stays true when adjoining an element, is also true for the whole group.

```
lemma (in finite_group) generate_induct[consumes 1, case_names base adjoin]:
  assumes "A0 ⊆ carrier G"
  assumes "A0 ⊆ carrier G ⟹ P (G(|carrier := generate G A0|))"
  assumes "⋀a A. ⟦A ⊆ carrier G; a ∈ carrier G - generate G A; A0 ⊆
A;
           P (G(|carrier := generate G A|))⟧ ⟹ P (G(|carrier := generate
G (A ∪ {a})|))"
  shows "P G"
⟨proof⟩
```

8.2 Finite abelian groups

Another trivial locale: the finite abelian group with some trivial facts.

```
locale finite_comm_group = finite_group + comm_group

lemma (in finite_comm_group) iso_imp_finite_comm:
  assumes "G ≅ H" "group H"
  shows "finite_comm_group H"
⟨proof⟩

lemma (in finite_comm_group) finite_comm_FactGroup:
  assumes "subgroup H G"
  shows "finite_comm_group (G Mod H)"
  ⟨proof⟩


lemma (in finite_comm_group) subgroup_imp_finite_comm_group:
  assumes "subgroup H G"
  shows   "finite_comm_group (G(|carrier := H|))"
⟨proof⟩
```

8.3 Cyclic groups

Now, the central notion of a cyclic group is introduced: a group generated by a single element.

locale *cyclic_group = group +*
  fixes *gen :: "'a"*
  assumes *gen_closed[intro, simp]: "gen ∈ carrier G"*
  assumes *generator: "carrier G = generate G {gen}"*

lemma *(***in** *cyclic_group) elem_is_gen_pow:*
  assumes *"x ∈ carrier G"*
  shows *"∃n :: int. x = gen [^] n"*
⟨*proof*⟩

Every cyclic group is commutative/abelian.

sublocale *cyclic_group ⊆ comm_group*
⟨*proof*⟩

Some trivial intro rules for showing that a group is cyclic.

lemma *(***in** *group) cyclic_groupI0:*
  assumes *"a ∈ carrier G" "carrier G = generate G {a}"*
  shows *"cyclic_group G a"*
  ⟨*proof*⟩

lemma *(***in** *group) cyclic_groupI1:*
  assumes *"a ∈ carrier G" "carrier G ⊆ generate G {a}"*
  shows *"cyclic_group G a"*
  ⟨*proof*⟩

lemma *(***in** *group) cyclic_groupI2:*
  assumes *"a ∈ carrier G"*
  shows *"cyclic_group (G⦇carrier := generate G {a}⦈) a"*
⟨*proof*⟩

The order of the generating element is always the same as the group order.

lemma *(***in** *cyclic_group) ord_gen_is_group_order:*
  shows *"ord gen = order G"*
⟨*proof*⟩

In the case of a finite group, it is sufficient to have one element of group order to know that the group is cyclic.

lemma *(***in** *finite_group) element_ord_generates_cyclic:*
  assumes *"a ∈ carrier G" "ord a = order G"*
  shows *"cyclic_group G a"*
⟨*proof*⟩

Another useful fact is that a group of prime order is also cyclic.

lemma *(***in** *group) prime_order_group_is_cyc:*

```
  assumes "Factorial_Ring.prime (order G)"
  obtains g where "cyclic_group G g"
⟨proof⟩
```

What follows is an induction principle for cyclic groups: a predicate is true for all elements of the group if it is true for all elements that can be formed by the generating element by just multiplication and if it also holds under the forming of the inverse (as we by this cover all elements of the group),

```
lemma (in cyclic_group) generator_induct [consumes 1, case_names generate
inv]:
  assumes x: "x ∈ carrier G"
  assumes IH1: "⋀n::nat. P (gen [^] n)"
  assumes IH2: "⋀x. x ∈ carrier G ⟹ P x ⟹ P (inv x)"
  shows   "P x"
⟨proof⟩
```

8.4 Finite cyclic groups

Additionally, the notion of the finite cyclic group is introduced.

```
locale finite_cyclic_group = finite_group + cyclic_group

sublocale finite_cyclic_group ⊆ finite_comm_group
  ⟨proof⟩

lemma (in finite_cyclic_group) ord_gen_gt_zero:
  "ord gen > 0"
  ⟨proof⟩
```

In order to prove something about an element in a finite abelian group, it is possible to show this property for the neutral element or the generating element and inductively for the elements that are formed by multiplying with the generator.

```
lemma (in finite_cyclic_group) generator_induct0 [consumes 1, case_names
one step]:
  assumes x: "x ∈ carrier G"
  assumes IH1: "P 𝟭"
  assumes IH2: "⋀x. ⟦x ∈ carrier G; P x⟧ ⟹ P (x ⊗ gen)"
  shows   "P x"
⟨proof⟩

lemma (in finite_cyclic_group) generator_induct1 [consumes 1, case_names
gen step]:
  assumes x: "x ∈ carrier G"
  assumes IH1: "P gen"
  assumes IH2: "⋀x. ⟦x ∈ carrier G; P x⟧ ⟹ P (x ⊗ gen)"
  shows   "P x"
⟨proof⟩
```

8.5 *get_exp* - discrete logarithm

What now follows is the discrete logarithm for groups. It is used at several times througout this entry and is initially used to show that two cyclic groups of the same order are isomorphic.

```
definition (in group) get_exp where
  "get_exp g = (λa. SOME k::int. a = g [^] k)"
```

For each element with itself as the basis the discrete logarithm indeed does what expected. This is not the strongest possible statement, but sufficient for our needs.

```
lemma (in group) get_exp_self_fulfills:
  assumes "a ∈ carrier G"
  shows "a = a [^] get_exp a a"
⟨proof⟩
```

```
lemma (in group) get_exp_self:
  assumes "a ∈ carrier G"
  shows "get_exp a a mod ord a = (1::int) mod ord a"
  ⟨proof⟩
```

For cyclic groups, the discrete logarithm "works" for every element.

```
lemma (in cyclic_group) get_exp_fulfills:
  assumes "a ∈ carrier G"
  shows "a = gen [^] get_exp gen a"
⟨proof⟩
```

```
lemma (in cyclic_group) get_exp_non_zero:
  assumes"b ∈ carrier G" "b ≠ 1"
  shows "get_exp gen b ≠ 0"
  ⟨proof⟩
```

One well-known logarithmic identity.

```
lemma (in cyclic_group) get_exp_mult_mod:
  assumes "a ∈ carrier G" "b ∈ carrier G"
  shows "get_exp gen (a ⊗ b) mod (ord gen) = (get_exp gen a + get_exp
gen b) mod (ord gen)"
⟨proof⟩
```

We now show that all functions from a group generated by 'a' to a group generated by 'b' that map elements from a^k to b^k in the other group are in fact isomorphisms between these two groups.

```
lemma (in group) iso_cyclic_groups_generate:
  assumes "a ∈ carrier G" "b ∈ carrier H" "group.ord G a = group.ord
H b" "group H"
  shows "{f. ∀k ∈ (UNIV::int set). f (a [^] k) = b [^]_H k}
         ⊆ iso (G⦇carrier := generate G {a}⦈) (H⦇carrier := generate
H {b}⦈)"
```

⟨*proof*⟩

This is then used to derive the isomorphism of two cyclic groups of the same order as a direct consequence.

```
lemma (in cyclic_group) iso_cyclic_groups_same_order:
  assumes "cyclic_group H h" "order G = order H"
  shows "G ≅ H"
⟨proof⟩
```

8.6 Integer modular groups

We show that `integer_mod_group` (written as `Z n`) is in fact a cyclic group. For $n \neq 1$ it is generated by 1 and in the other case by 0.

```
notation integer_mod_group (‹Z›)

lemma Zn_neq1_cyclic_group:
  assumes "n ≠ 1"
  shows "cyclic_group (Z n) 1"
⟨proof⟩

lemma Z1_cyclic_group: "cyclic_group (Z 1) 0"
⟨proof⟩

lemma Zn_cyclic_group:
  obtains x where "cyclic_group (Z n) x"
  ⟨proof⟩
```

Moreover, its order is just n.

```
lemma Zn_order: "order (Z n) = n"
  ⟨proof⟩
```

Consequently, `Z n` is isomorphic to any cyclic group of order n.

```
lemma (in cyclic_group) Zn_iso:
  assumes "order G = n"
  shows "G ≅ Z n"
  ⟨proof⟩

no_notation integer_mod_group (‹Z›)
end
```

9 Direct group product

```
theory DirProds
  imports Finite_Product_Extend Group_Hom Finite_And_Cyclic_Groups
begin

notation integer_mod_group (‹Z›)
```

The direct group product is defined component-wise and provided in an indexed way.

definition *DirProds :: "('a ⇒ ('b, 'c) monoid_scheme) ⇒ 'a set ⇒ ('a ⇒ 'b) monoid"* **where**
  "DirProds G I = ⦇ carrier = Pi_E I (carrier ∘ G),
  monoid.mult = (λx y. restrict (λi. x i $\otimes_{G\ i}$ y i) I),
  one = restrict (λi. $\mathbf{1}_{G\ i}$) I ⦈"

Basic lemmas about *DirProds*.

lemma *DirProds_empty:*
  "carrier (DirProds f {}) = {$\mathbf{1}_{DirProds\ f\ \{\}}$}"
  ⟨*proof*⟩

lemma *DirProds_order:*
  assumes *"finite I"*
  shows *"order (DirProds G I) = prod (order ∘ G) I"*
  ⟨*proof*⟩

lemma *DirProds_in_carrI:*
  assumes *"⋀i. i ∈ I ⟹ x i ∈ carrier (G i)" "⋀i. i ∉ I ⟹ x i = undefined"*
  shows *"x ∈ carrier (DirProds G I)"*
  ⟨*proof*⟩

lemma *comp_in_carr:*
  assumes *"x ∈ carrier (DirProds G I)" "i ∈ I"*
  shows *"x i ∈ carrier (G i)"*
  ⟨*proof*⟩

lemma *comp_mult:*
  assumes *"i ∈ I"*
  shows *"(x $\otimes_{DirProds\ G\ I}$ y) i = (x i $\otimes_{G\ i}$ y i)"*
  ⟨*proof*⟩

lemma *comp_exp_nat:*
  fixes *k::nat*
  assumes *"i ∈ I"*
  shows *"(x $[\hat{}]_{DirProds\ G\ I}$ k) i = x i $[\hat{}]_{G\ i}$ k"*
⟨*proof*⟩

lemma *DirProds_m_closed:*
  assumes *"x ∈ carrier (DirProds G I)" "y ∈ carrier (DirProds G I)" "⋀i. i ∈ I ⟹ group (G i)"*
  shows *"x $\otimes_{DirProds\ G\ I}$ y ∈ carrier (DirProds G I)"*
  ⟨*proof*⟩

lemma *partial_restr:*
  assumes *"a ∈ carrier (DirProds G I)" "J ⊆ I"*
  shows *"restrict a J ∈ carrier (DirProds G J)"*

⟨proof⟩

lemma *eq_parts_imp_eq:*
  assumes *"a ∈ carrier (DirProds G I)" "b ∈ carrier (DirProds G I)"*
"⋀i. i ∈ I ⟹ a i = b i"
  shows *"a = b"*
  ⟨proof⟩

lemma *mult_restr:*
  assumes *"a ∈ carrier (DirProds G I)" "b ∈ carrier (DirProds G I)"*
"J ⊆ I"
  shows *"a $\otimes_{DirProds\ G\ J}$ b = restrict (a $\otimes_{DirProds\ G\ I}$ b) J"*
  ⟨proof⟩

lemma *DirProds_one:*
  assumes *"x ∈ carrier (DirProds G I)"*
  shows *"(∀i ∈ I. x i = $\mathbf{1}_{G\ i}$) ⟷ x = $\mathbf{1}_{DirProds\ G\ I}$"*
  ⟨proof⟩

lemma *DirProds_one':*
  "i∈I ⟹ $\mathbf{1}_{DirProds\ G\ I}$ i = $\mathbf{1}_{G\ i}$"
  ⟨proof⟩

lemma *DirProds_one'':*
  "$\mathbf{1}_{DirProds\ G\ I}$ = restrict (λi. $\mathbf{1}_{G\ i}$) I"
  ⟨proof⟩

lemma *DirProds_mult:*
  "($\otimes_{DirProds\ G\ I}$) = (λx y. restrict (λi. x i $\otimes_{G\ i}$ y i) I)"
  ⟨proof⟩

lemma *DirProds_one_iso: "(λx. x G) ∈ iso (DirProds f {G}) (f G)"*
⟨proof⟩

lemma *DirProds_one_cong: "(DirProds f {G}) ≅ (f G)"*
  ⟨proof⟩

lemma *DirProds_one_iso_sym: "(λx. (λ_∈{G}. x)) ∈ iso (f G) (DirProds f {G})"*
⟨proof⟩

lemma *DirProds_one_cong_sym: "(f G) ≅ (DirProds f {G})"*
  ⟨proof⟩

The direct product is a group iff all factors are groups.

lemma *DirProds_is_group:*
  assumes *"⋀i. i ∈ I ⟹ group (G i)"*
  shows *"group (DirProds G I)"*
⟨proof⟩

lemma *DirProds_obtain_elem_carr:*
  assumes *"group (DirProds G I)" "i ∈ I" "x ∈ carrier (G i)"*
  obtains *k* **where** *"k ∈ carrier (DirProds G I)" "k i = x"*
⟨*proof*⟩

lemma *DirProds_group_imp_groups:*
  assumes *"group (DirProds G I)"* **and** *i: "i ∈ I"*
  shows *"group (G i)"*
⟨*proof*⟩

lemma *DirProds_group_iff: "group (DirProds G I) ⟷ (∀ i∈I. group (G i))"*
  ⟨*proof*⟩

lemma *comp_inv:*
  assumes *"group (DirProds G I)"* **and** *x: "x ∈ carrier (DirProds G I)"* **and** *i: "i ∈ I"*
  shows *"(inv$_{(DirProds\ G\ I)}$ x) i = inv$_{(G\ i)}$ (x i)"*
⟨*proof*⟩

The same is true for abelian groups.

lemma *DirProds_is_comm_group:*
  assumes *"⋀i. i ∈ I ⟹ comm_group (G i)"*
  shows *"comm_group (DirProds G I)"* (**is** *"comm_group ?DP")*
⟨*proof*⟩

lemma *DirProds_comm_group_imp_comm_groups:*
  assumes *"comm_group (DirProds G I)"* **and** *i: "i ∈ I"*
  shows *"comm_group (G i)"*
⟨*proof*⟩

lemma *DirProds_comm_group_iff: "comm_group (DirProds G I) ⟷ (∀ i∈I. comm_group (G i))"*
  ⟨*proof*⟩

And also for finite groups.

lemma *DirProds_is_finite_group:*
  assumes *"⋀i. i∈I ⟹ finite_group (G i)" "finite I"*
  shows *"finite_group (DirProds G I)"*
⟨*proof*⟩

lemma *DirProds_finite_imp_finite_groups:*
  assumes *"finite_group (DirProds G I)" "finite I"*
  shows *"⋀i. i∈I ⟹ finite_group (G i)"*
⟨*proof*⟩

lemma *DirProds_finite_group_iff:*
  assumes *"finite I"*

shows *"finite_group (DirProds G I) ⟷ (∀i∈I. finite_group (G i))"*
⟨*proof*⟩

lemma *DirProds_finite_comm_group_iff:*
assumes *"finite I"*
shows *"finite_comm_group (DirProds G I) ⟷ (∀i∈I. finite_comm_group (G i))"*
⟨*proof*⟩

If a group is an internal direct product of a family of subgroups, it is isomorphic to the direct product of these subgroups.

lemma **(in** *comm_group)* *subgroup_iso_DirProds_IDirProds:*
assumes *"subgroup J G" "is_idirprod J S I" "finite I"*
shows *"(λx. ⨂Gi∈I. x i) ∈ iso (DirProds (λi. G⦇carrier := (S i)⦈) I) (G⦇carrier := J⦈)"*
(is *"?fp ∈ iso ?DP ?J"*)
⟨*proof*⟩

lemma **(in** *comm_group)* *iso_DirProds_IDirProds:*
assumes *"is_idirprod (carrier G) S I" "finite I"*
shows *"(λx. ⨂Gi∈I. x i) ∈ iso (DirProds (λi. G⦇carrier := (S i)⦈) I) G"*
⟨*proof*⟩

lemma **(in** *comm_group)* *cong_DirProds_IDirProds:*
assumes *"is_idirprod (carrier G) S I" "finite I"*
shows *"DirProds (λi. G⦇carrier := (S i)⦈) I ≅ G"*
⟨*proof*⟩

In order to prove the isomorphism between two direct products, the following lemmas provide some criterias.

lemma *DirProds_iso:*
assumes *"bij_betw f I J" "⋀i. i∈I ⟹ Gs i ≅ Hs (f i)"*
"⋀i. i∈I ⟹ group (Gs i)" "⋀j. j∈J ⟹ group (Hs j)"
shows *"DirProds Gs I ≅ DirProds Hs J"*
⟨*proof*⟩

lemma *DirProds_iso1:*
assumes *"⋀i. i∈I ⟹ Gs i ≅ (f ∘ Gs) i" "⋀i. i∈I ⟹ group (Gs i)" "⋀i. i∈I ⟹ group ((f ∘ Gs) i)"*
shows *"DirProds Gs I ≅ DirProds (f ∘ Gs) I"*
⟨*proof*⟩

lemma *DirProds_iso2:*
assumes *"inj_on f A" "group (DirProds g (f ‘ A))"*
shows *"DirProds (g ∘ f) A ≅ DirProds g (f ‘ A)"*
⟨*proof*⟩

The direct group product distributes when nested.

lemma *DirProds_Sigma:*
  "DirProds (λi. DirProds (G i) (J i)) I ≅ DirProds (λ(i,j). G i j) (Sigma I J)" (**is** *"?L ≅ ?R"*)
⟨*proof*⟩

no_notation *integer_mod_group* (<Z>)

end

10 Group relations

theory *Group_Relations*
  imports *Finite_Product_Extend*
begin

We introduce the notion of a relation of a set of elements: a way to express the neutral element by using only powers of said elements. The following predicate describes the set of all the relations that one can construct from a set of elements.

definition (**in** *comm_group*) *relations :: "'a set ⇒ ('a ⇒ int) set"* **where**
  *"relations A = {f. finprod G (λa. a [^] f a) A = **1**} ∩ extensional A"*

Now some basic lemmas about relations.

lemma (**in** *comm_group*) *in_relationsI[intro]:*
  assumes *"finprod G (λa. a [^] f a) A = **1**" "f ∈ extensional A"*
  shows *"f ∈ relations A"*
  ⟨*proof*⟩

lemma (**in** *comm_group*) *triv_rel:*
  "restrict (λ_. 0::int) A ∈ relations A"
⟨*proof*⟩

lemma (**in** *comm_group*) *not_triv_relI:*
  assumes *"a ∈ A" "f a ≠ (0::int)"*
  shows *"f ≠ (λ_∈A. 0::int)"*
  ⟨*proof*⟩

lemma (**in** *comm_group*) *rel_in_carr:*
  assumes *"A ⊆ carrier G" "r ∈ relations A"*
  shows *"(λa. a [^] r a) ∈ A → carrier G"*
  ⟨*proof*⟩

The following lemmas are of importance when proving the fundamental theorem of finitely generated abelian groups in the case that there is just the trivial relation between a set of generators. They all build up to the last lemma that then is actually used in the proof.

lemma (**in** *comm_group*) *relations_zero_imp_pow_not_one:*

assumes "a ∈ A" "∀f∈(relations A). f a = 0"
shows "∀z::int ≠ 0. a [^] z ≠ **1**"
⟨*proof*⟩

lemma (in *comm_group*) *relations_zero_imp_ord_zero:*
assumes "a ∈ A" "∀f∈(relations A). f a = 0"
and "a ∈ carrier G"
shows "ord a = 0"
⟨*proof*⟩

lemma (in *comm_group*) *finprod_relations_triv_harder_better_stronger:*
assumes "A ⊆ carrier G" "relations A = {(λ_∈A. 0::int)}"
shows "∀f ∈ Pi_E A (λa. generate G {a}). finprod G f A = **1** ⟶ (∀a∈A. f a = **1**)"
⟨*proof*⟩

lemma (in *comm_group*) *stronger_PiE_finprod_imp:*
assumes "A ⊆ carrier G" "∀f ∈ Pi_E A (λa. generate G {a}). finprod G f A = **1** ⟶ (∀a∈A. f a = **1**)"
shows "∀f ∈ Pi_E ((λa. generate G {a}) ' A) id.
finprod G f ((λa. generate G {a}) ' A) = **1** ⟶ (∀H∈ (λa. generate G {a}) ' A. f H = **1**)"
⟨*proof*⟩

lemma (in *comm_group*) *finprod_relations_triv:*
assumes "A ⊆ carrier G" "relations A = {(λ_∈A. 0::int)}"
shows "∀f ∈ Pi_E ((λa. generate G {a}) ' A) id.
finprod G f ((λa. generate G {a}) ' A) = **1** ⟶ (∀H∈ (λa. generate G {a}) ' A. f H = **1**)"
⟨*proof*⟩

lemma (in *comm_group*) *ord_zero_strong_imp_rel_triv:*
assumes "A ⊆ carrier G" "∀a ∈ A. ord a = 0"
and "∀f ∈ Pi_E A (λa. generate G {a}). finprod G f A = **1** ⟶ (∀a∈A. f a = **1**)"
shows "relations A = {(λ_∈A. 0::int)}"
⟨*proof*⟩

lemma (in *comm_group*) *compl_fam_iff_relations_triv:*
assumes "finite gs" "gs ⊆ carrier G" "∀g∈gs. ord g = 0"
shows "relations gs = {(λ_∈gs. 0::int)} ⟷ compl_fam (λg. generate G {g}) gs"
⟨*proof*⟩

end

11 Fundamental Theorem of Finitely Generated Abelian Groups

```
theory Finitely_Generated_Abelian_Groups
  imports DirProds Group_Relations
begin

notation integer_mod_group (‹Z›)

locale fin_gen_comm_group = comm_group +
  fixes gen :: "'a set"
  assumes gens_closed: "gen ⊆ carrier G"
  and     fin_gen: "finite gen"
  and     generators: "carrier G = generate G gen"
```

Every finite abelian group is also finitely generated.

```
sublocale finite_comm_group ⊆ fin_gen_comm_group G "carrier G"
  ⟨proof⟩
```

This lemma contains the proof of Kemper from his lecture notes on algebra [1]. However, the proof is not done in the context of a finitely generated group but for a finitely generated subgroup in a commutative group.

```
lemma (in comm_group) ex_idirgen:
  fixes A :: "'a set"
  assumes "finite A" "A ⊆ carrier G"
  shows "∃gs. set gs ⊆ generate G A ∧ distinct gs ∧ is_idirprod (generate
G A) (λg. generate G {g}) (set gs)
           ∧ successively (dvd) (map ord gs) ∧ card (set gs) ≤ card
A"
   (is "?t A")
  ⟨proof⟩

lemma (in comm_group) fundamental_subgr:
  fixes A :: "'a set"
  assumes "finite A" "A ⊆ carrier G"
  obtains gs where
    "set gs ⊆ generate G A" "distinct gs" "is_idirprod (generate G A)
(λg. generate G {g}) (set gs)"
    "successively (dvd) (map ord gs)" "card (set gs) ≤ card A"
  ⟨proof⟩
```

As every group is a subgroup of itself, the theorem follows directly. However, for reasons of convenience and uniqueness (although not completely proved), we strengthen the result by proving that the decomposition can be done without having the trivial factor in the product. We formulate the theorem in various ways: firstly, the invariant factor decomposition.

```
theorem (in fin_gen_comm_group) invariant_factor_decomposition_idirprod:
  obtains gs where
```

"set gs ⊆ carrier G" "distinct gs" "is_idirprod (carrier G) (λg. generate G {g}) (set gs)"
"successively (dvd) (map ord gs)" "card (set gs) ≤ card gen" "**1** ∉ set gs"
⟨*proof*⟩

corollary (in *fin_gen_comm_group*) *invariant_factor_decomposition_dirprod:*
obtains *gs* **where**
"set gs ⊆ carrier G" "distinct gs"
"DirProds (λg. G⦇carrier := generate G {g}⦈) (set gs) ≅ G"
"successively (dvd) (map ord gs)" "card (set gs) ≤ card gen"
"compl_fam (λg. generate G {g}) (set gs)" "**1** ∉ set gs"
⟨*proof*⟩

corollary (in *fin_gen_comm_group*) *invariant_factor_decomposition_dirprod_fam:*
obtains *Hs* **where**
"⋀H. H ∈ set Hs ⟹ subgroup H G" "distinct Hs"
"DirProds (λH. G⦇carrier := H⦈) (set Hs) ≅ G" "successively (dvd) (map card Hs)"
"card (set Hs) ≤ card gen" "compl_fam id (set Hs)" "{**1**} ∉ set Hs"
⟨*proof*⟩

Here, the invariant factor decomposition in its classical form.

corollary (in *fin_gen_comm_group*) *invariant_factor_decomposition_Zn:*
obtains *ns* **where**
"DirProds (λn. Z (ns!n)) {..<length ns} ≅ G" "successively (dvd) ns" "length ns ≤ card gen"
⟨*proof*⟩

As every *integer_mod_group* can be decomposed into a product of prime power groups, we obtain (by using the fact that the direct product does not care about nestedness) the primary decomposition.

lemma *Zn_iso_DirProds_prime_powers:*
assumes "n ≠ 0"
shows "Z n ≅ DirProds (λp. Z (p ^ multiplicity p n)) (prime_factors n)" (**is** "Z n ≅ ?DP")
⟨*proof*⟩

lemma *Zn_iso_DirProds_prime_powers':*
assumes "n ≠ 0"
shows "Z n ≅ DirProds (λp. Z p) ((λp. p ^ multiplicity p n) ` (prime_factors n))" (**is** "Z n ≅ ?DP")
⟨*proof*⟩

corollary (in *fin_gen_comm_group*) *primary_decomposition_Zn:*
obtains *ns* **where**
"DirProds (λn. Z (ns!n)) {..<length ns} ≅ G"
"∀n∈set ns. n = 0 ∨ (∃p k. Factorial_Ring.prime p ∧ k > 0 ∧ n = p ^ k)"

⟨*proof*⟩

As every finite group is also finitely generated, it follows that a finite group can be decomposed in a product of finite cyclic groups.

lemma *(in finite_comm_group) cyclic_product:*
obtains *ns* **where** *"DirProds (λn. Z (ns!n)) {..<length ns} ≅ G" "∀n∈set ns. n≠0"*
⟨*proof*⟩

no_notation *integer_mod_group (‹Z›)*

end

References

[1] G. Kemper. Lecture notes of algebra. https://www.groups.ma.tum.de/fileadmin/w00ccg/algebra/people/kemper/lectureNotes/Algebra.pdf, 04 2020.