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)"
  unfolding set_mult_def by auto

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" unfolding l_coset_def
proof -
  have "card ((⊗) x ` J) = card J"
    using inj_on_cmult[of x] card_image[of "(⊗) x" J] assms inj_on_subset[of
"(⊗) x" "carrier G" J]
    by blast
  moreover have "(⋃y∈J. {x ⊗ y}) = (⊗) x ` J" using image_def[of "(⊗)
x" J] by blast
  ultimately show "card (⋃h∈J. {x ⊗ h}) = card J" by presburger
qed
```

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"
  using assms
proof (induction "card H" arbitrary: H)
  case 0
  then have "H = {}" by force
```

```
  then show ?case using set_mult_def[of G H J] by simp
next
  case (Suc n)
  then obtain a where a_def: "a ∈ H" by fastforce
  then have c_n: "card (H - {a}) = n" using Suc by force
  then have "card ((H - {a}) <#> J) ≤ card (H - {a}) * card J" using
Suc by blast
  moreover have "card ({a} <#> J) = card J"
    using Suc(4, 5) a_def set_mult_card_single_el_eq[of J a] l_coset_eq_set_mult[of
G a J] by auto
  moreover have "H <#> J = (H - {a} <#> J) ∪ ({a} <#> J)" using set_mult_def[of
G _ J] a_def by auto
  moreover have "card (H - {a}) * card J + card J = Suc n * card J" us-
ing c_n mult_Suc by presburger
  ultimately show ?case using card_Un_le[of "H - {a} <#> J" "{a} <#>
J"] c_n ‹Suc n = card H› by auto
qed

lemma (in group) set_mult_finite:
  assumes "finite H" "finite J" "H ⊆ carrier G" "J ⊆ carrier G"
  shows "finite (H <#> J)"
  using assms set_mult_def[of G H J] by auto
```

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)
= {}"
  using assms
proof (induction H rule: finite_induct)
  case empty
  then show ?case by fast
next
  case step: (insert x H)
  from step have x_c: "x ∈ carrier G" by simp
  from step have H_c: "H ⊆ carrier G" by simp
  from set_mult_card_single_el_eq[of J x] have card_x: "card ({x} <#>
J) = card J"
    using ‹J ⊆ carrier G› x_c l_coset_eq_set_mult by metis
  moreover have ins: "(insert x H) <#> J = (H <#> J) ∪ ({x} <#> J)"
    using set_mult_def[of G _ J] by auto
  ultimately have "card (H <#> J) ≥ card H * card J"
    using card_Un_le[of "H <#> J" "{x} <#> J"] ‹card (insert x H <#> J)
= card (insert x H) * card J›
    by (simp add: step.hyps(1) step.hyps(2))
  then have card_eq:"card (H <#> J) = card H * card J"
```

```
    using set_mult_card_le[of H J] step H_c by linarith
  then have ih: "⋀a b. ⟦a ∈ H; b ∈ H; a ≠ b⟧ ⟹ ((⊗) a ‘ J) ∩ ((⊗)
b ‘ J) = {}"
    using step H_c by presburger

  have "card (insert x H) * card J = card H * card J + card J" using ‹x
∉ H› using step by simp
  then have "({x} <#> J) ∩ (H <#> J) = {}"
    using card_eq card_x ins card_Un_Int[of "H <#> J" "{x} <#> J"] step
set_mult_finite by auto
  then have "⋀a. a ∈ H ⟹ (⋃y∈J. {a ⊗ y}) ∩ (⋃y∈J. {x ⊗ y}) =
{}"
    using set_mult_def[of G _ J] by blast
  then have "⋀a b. ⟦a ∈ (insert x H); b ∈ (insert x H); a ≠ b⟧ ⟹
((⊗) a ‘ J) ∩ ((⊗) b ‘ J) = {}"
    using ‹x ∉ H› ih by blast
  then show ?case using step by presburger
qed

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) = {}"
  unfolding l_coset_def
  using set_mult_card_eq_impl_empty_inter image_def[of "(⊗) _" J] assms
by blast

lemma (in comm_group) set_mult_comm:
  assumes "H ⊆ carrier G" "J ⊆ carrier G"
  shows "(H <#> J) = (J <#> H)"
  unfolding set_mult_def
proof -
  have 1: "⋀a b. ⟦a ∈ carrier G; b ∈ carrier G⟧ ⟹ {a ⊗ b} = {b ⊗
a}" using m_comm by simp
  then have "⋀a b.⟦a ∈ H; b ∈ J⟧ ⟹ {a ⊗ b} = {b ⊗ a}" using assms
by auto
  moreover have "⋀a b.⟦b ∈ H; a ∈ J⟧ ⟹ {a ⊗ b} = {b ⊗ a}" using
assms 1 by auto
  ultimately show "(⋃h∈H. ⋃k∈J. {h ⊗ k}) = (⋃k∈J. ⋃h∈H. {k ⊗ h})"
by fast
qed

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

```
  thus "x ∈ (B <#> A)" using assms unfolding set_mult_def by force
qed
```

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
  fix x
  assume "x ∈ A <#> B"
  then obtain a b where ab: "a ∈ A" "b ∈ B" "x = a ⊗ b" unfolding set_mult_def
by blast
  then have "a ∈ generate G (A ∪ B)" "b ∈ generate G (A ∪ B)"
    using generate.incl[of _ "A ∪ B" G] by simp+
  thus "x ∈ generate G (A ∪ B)" using ab generate.eng by metis
qed
```

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
  show "?L ⊆ ?R"
  proof
    fix x
    assume "x ∈ ?L"
    then show "x ∈ ?R"
    proof(induction rule: generate.induct)
      case one
      have "1 ⊗ 1 = 1" using nat_pow_one[of 2] by simp
      thus ?case
        using assms subgroup.one_closed[OF assms(1)]
              subgroup.one_closed[OF assms(2)] set_mult_def[of G H J]
by fastforce
    next
      case (incl x)
      have H1: "1 ∈ H" using assms subgroup.one_closed by auto
      have J1: "1 ∈ J" using assms subgroup.one_closed by auto
      have lx: "x ⊗ 1 = x" using r_one[of x] incl subgroup.subset assms
by blast
      have rx: "1 ⊗ x = x" using l_one[of x] incl subgroup.subset assms
by blast
      show ?case
      proof (cases "x ∈ H")
        case True
        then show ?thesis using set_mult_def J1 lx by fastforce
      next
```

```
        case False
        then show ?thesis using set_mult_def H1 rx incl by fastforce
      qed
    next
      case (inv h)
      then have inv_in:"(inv h) ∈ H ∪ J" (is "?iv ∈ H ∪ J")
        using assms subgroup.m_inv_closed[of _ G h] by (cases "h ∈ H";
blast)
      have H1: "1 ∈ H" using assms subgroup.one_closed by auto
      have J1: "1 ∈ J" using assms subgroup.one_closed by auto
      have lx: "?iv ⊗ 1 = ?iv" using r_one[of "?iv"] subgroup.subset
inv_in assms by blast
      have rx: "1 ⊗ ?iv = ?iv" using l_one[of "?iv"] incl subgroup.subset
inv_in assms by blast
      show ?case
      proof (cases "?iv ∈ H")
        case True
        then show ?thesis using set_mult_def[of G H J] J1 lx by fastforce
      next
        case False
        then show ?thesis using set_mult_def[of G H J] H1 rx inv_in by
fastforce
      qed
    next
      case (eng h g)
      from eng(3) obtain a b where aH: "a ∈ H" and bJ: "b ∈ J" and
h_def: "h = a ⊗ b"
        using set_mult_def[of G H J] by fast
      have a_carr: "a ∈ carrier G" by (metis subgroup.mem_carrier assms(1)
aH)
      have b_carr: "b ∈ carrier G" by (metis subgroup.mem_carrier assms(2)
bJ)
      from eng(4) obtain c d where cH: "c ∈ H" and dJ: "d ∈ J" and
g_def: "g = c ⊗ d"
        using set_mult_def[of G H J] by fast
      have c_carr: "c ∈ carrier G" by (metis subgroup.mem_carrier assms(1)
cH)
      have d_carr: "d ∈ carrier G" by (metis subgroup.mem_carrier assms(2)
dJ)
      then have "h ⊗ g = (a ⊗ c) ⊗ (b ⊗ d)"
        using a_carr b_carr c_carr d_carr g_def h_def m_assoc m_comm by
force
      moreover have "a ⊗ c ∈ H" using assms(1) aH cH subgroup.m_closed
by fast
      moreover have "b ⊗ d ∈ J" using assms(2) bJ dJ subgroup.m_closed
by fast
      ultimately show ?case using set_mult_def by fast
    qed
  qed
```

```
next
  show "?R ⊆ ?L" using set_mult_subset_generate[of H J] subgroup.subset
assms by blast
qed

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"
  by (induction n) (auto simp: nat_pow_def)


lemma nat_pow_modify_carrier: "a [^]G⦇carrier := H⦈ b = a [^]G (b::nat)"
  by (simp add: nat_pow_def)

lemma (in group) subgroup_card_dvd_group_ord:
  assumes "subgroup H G"
  shows "card H dvd order G"
  using Coset.group.lagrange[of G H] assms group_axioms by (metis dvd_triv_right)

lemma (in group) subgroup_card_eq_order:
  assumes "subgroup H G"
  shows "card H = order (G⦇carrier := H⦈)"
  unfolding order_def by simp

lemma (in group) finite_subgroup_card_neq_0:
  assumes "subgroup H G" "finite H"
  shows "card H ≠ 0"
  using subgroup_nonempty assms by auto

lemma (in group) subgroup_order_dvd_group_order:
  assumes "subgroup H G"
  shows "order (G⦇carrier := H⦈) dvd order G"
  by (metis subgroup_card_dvd_group_ord[of H] assms subgroup_card_eq_order)

lemma (in group) sub_subgroup_dvd_card:
  assumes "subgroup H G" "subgroup J G" "J ⊆ H"
  shows "card J dvd card H"
  by (metis subgroup_incl[of J H] subgroup_card_eq_order[of H]
           group.subgroup_card_dvd_group_ord[of "(G⦇carrier := H⦈)" J]
```

```
assms
            subgroup.subgroup_is_group[of H G] group_axioms)

lemma (in group) inter_subgroup_dvd_card:
  assumes "subgroup H G" "subgroup J G"
  shows "card (H ∩ J) dvd card H"
  using subgroups_Inter_pair[of H J] assms sub_subgroup_dvd_card[of H
"H ∩ J"] by blast

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 -
  from assms inter_subgroup_dvd_card have "is_unit (card (H ∩ J))" un-
folding coprime_def
    by (metis assms(3) coprime_common_divisor inf_commute)
  then show ?thesis by simp
qed

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) = {}"
  by (metis Int_emptyI assms repr_independence)

lemma (in comm_group) subgroup_is_comm_group:
  assumes "subgroup H G"
  shows "comm_group (G⦇carrier := H⦈)" unfolding comm_group_def
proof
  interpret H: subgroup H G by fact
  interpret H: submonoid H G using H.subgroup_is_submonoid .
  show "Group.group (G⦇carrier := H⦈)" by blast
  show "comm_monoid (G⦇carrier := H⦈)" using submonoid_is_comm_monoid
H.submonoid_axioms by blast
qed

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 -
  obtain q r where d: "q = n div ord a" "r = n mod ord a" "n = q * ord
a + r"
    using mod_div_decomp by blast
  hence "a [^] n = (a [^] int (ord a)) [^] q ⊗ a [^] r"
    using assms(1) int_pow_mult int_pow_pow
    by (metis mult_of_nat_commute)
  also have "... = 𝟭 [^] q ⊗ a [^] r"
    by (simp add: int_pow_int)
  also have "... = a [^] r" by simp
  finally show ?thesis using d(2) by blast
```

```
qed

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 -
  obtain q r where d: "q = n div ord a" "r = n mod ord a" "n = q * ord
a + r"
    using mod_div_decomp by blast
  hence "a [^] n = (a [^] ord a) [^] q ⊗ a [^] r"
    using assms(1) nat_pow_mult nat_pow_pow by presburger
  also have "... = 1 [^] q ⊗ a [^] r" by auto
  also have "... = a [^] r" by simp
  finally show ?thesis using d(2) by blast
qed

lemma (in group) ord_min:
  assumes "m ≥ 1" "x ∈ carrier G" "x [^] m = 1"
  shows   "ord x ≤ m"
  using assms pow_eq_id by auto


lemma (in group) bij_betw_mult_left[intro]:
  assumes [simp]: "x ∈ carrier G"
  shows "bij_betw (λy. x ⊗ y) (carrier G) (carrier G)"
  by (intro bij_betwI[where ?g = "λy. inv x ⊗ y"])
     (auto simp: m_assoc [symmetric])


lemma (in subgroup) inv_in_iff:
  assumes "x ∈ carrier G" "group G"
  shows   "inv x ∈ H ⟷ x ∈ H"
proof safe
  assume "inv x ∈ H"
  hence "inv (inv x) ∈ H" by blast
  also have "inv (inv x) = x"
    by (intro group.inv_inv) (use assms in auto)
  finally show "x ∈ H" .
qed auto


lemma (in subgroup) mult_in_cancel_left:
  assumes "y ∈ carrier G" "x ∈ H" "group G"
  shows   "x ⊗ y ∈ H ⟷ y ∈ H"
proof safe
  assume "x ⊗ y ∈ H"
  hence "inv x ⊗ (x ⊗ y) ∈ H"
    using assms by blast
  also have "inv x ⊗ (x ⊗ y) = y"
```

```
    using assms by (simp add: ‹x ⊗ y ∈ H› group.inv_solve_left')
  finally show "y ∈ H" .
qed (use assms in auto)


lemma (in subgroup) mult_in_cancel_right:
  assumes "x ∈ carrier G" "y ∈ H" "group G"
  shows   "x ⊗ y ∈ H ⟷ x ∈ H"
proof safe
  assume "x ⊗ y ∈ H"
  hence "(x ⊗ y) ⊗ inv y ∈ H"
    using assms by blast
  also have "(x ⊗ y) ⊗ inv y = x"
    using assms by (simp add: ‹x ⊗ y ∈ H› group.inv_solve_right')
  finally show "x ∈ H" .
qed (use assms in auto)

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 -
  have "ord x dvd n"
    using pow_eq_id[of x n] assms by auto
  thus "ord x ≤ n" "ord x > 0"
    using assms by (auto intro: dvd_imp_le)
qed

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 (rule antisym)
  show "ord x ≤ (LEAST n::nat. 0 < n ∧ x [^] n = 1)"
    using assms LeastI_ex[OF assms(2)] by (intro ord_le) auto
  show "ord x ≥ (LEAST n::nat. 0 < n ∧ x [^] n = 1)"
    using assms by (intro Least_le) (auto intro: pow_ord_eq_1 ord_pos)
qed

lemma (in group) ord_conv_Gcd:
  assumes "x ∈ carrier G"
  shows   "ord x = Gcd {n. x [^] n = 1}"
  by (rule sym, rule Gcd_eqI) (use assms in ‹auto simp: pow_eq_id›)

lemma (in group) subgroup_ord_eq:
  assumes "subgroup H G" "x ∈ H"
  shows "group.ord (G⦇carrier := H⦈) x = ord x"
  using nat_pow_consistent ord_def group.ord_def[of "(G⦇carrier := H⦈)"
x]
        subgroup.subgroup_is_group[of H G] assms by simp
```

```
lemma (in group) ord_FactGroup:
  assumes "subgroup P G" "group (G Mod P)"
  shows "order (G Mod P) * card P = order G"
  using lagrange[of P] FactGroup_def[of G P] assms order_def[of "(G Mod
P)"] by fastforce

lemma (in group) one_is_same:
  assumes "subgroup H G"
  shows "1G(|carrier := H|) = 1"
  by simp

lemma (in group) kernel_FactGroup:
  assumes "P ◁ G"
  shows "kernel G (G Mod P) (λx. P #> x) = P"
proof(rule equalityI; rule subsetI)
  fix x
  assume "x ∈ kernel G (G Mod P) ((#>) P)"
  then have "P #> x = 1G Mod P" "x ∈ carrier G" unfolding kernel_def
by simp+
  with coset_join1[of P x] show "x ∈ P" using assms unfolding normal_def
by simp
next
  fix x
  assume x:"x ∈ P"
  then have xc: "x ∈ carrier G" using assms subgroup.subset unfolding
normal_def by fast
  from x have "P #> x = P" using assms
    by (simp add: normal_imp_subgroup subgroup.rcos_const)
  thus "x ∈ kernel G (G Mod P) ((#>) P)" unfolding kernel_def using xc
by simp
qed

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)"
  using assms by (meson coprime_divisors sub_subgroup_dvd_card)

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 -
  from assms have "a [^] (n - m) = 1" using pow_eq_div2 by blast
  hence "ord a dvd n - m" using assms(1) pow_eq_id by blast
  thus ?thesis
    by (metis assms mod_eq_dvd_iff_nat nat_le_linear pow_eq_div2 pow_eq_id)
qed

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 -
  from assms have "a [^] (n - m) = 1" using int_pow_closed int_pow_diff
r_inv by presburger
  hence "ord a dvd n - m" using assms(1) int_pow_eq_id by blast
  thus ?thesis by (meson mod_eq_dvd_iff)
qed

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"
  using generate.incl by fast
```

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"
  using assms generateI group.generate_is_subgroup by blast

lemma (in group) generate_idem':
  assumes "A ⊆ carrier G" "B ⊆ carrier G"
  shows "generate G (generate G A ∪ B) = generate G (A ∪ B)"
proof
  show "generate G (generate G A ∪ B) ⊆ generate G (A ∪ B)"
  proof -
    have "generate G A ∪ B ⊆ generate G (A ∪ B)"
    proof -
      have "generate G A ⊆ generate G (A ∪ B)" using mono_generate by
simp
      moreover have "B ⊆ generate G (A ∪ B)" by (simp add: generate.incl
subset_iff)
      ultimately show ?thesis by simp
    qed
    then have "generate G (generate G A ∪ B) ⊆ generate G (generate
G (A ∪ B))"
      using mono_generate by auto
```

```
    with generate_idem[of "A ∪ B"] show ?thesis using assms by simp
  qed
  show "generate G (A ∪ B) ⊆ generate G (generate G A ∪ B)"
  proof -
    have "A ⊆ generate G A" using generate.incl by fast
    thus ?thesis using mono_generate[of "A ∪ B" "generate G A ∪ B"] by
blast
  qed
qed

lemma (in group) generate_idem'_right:
  assumes "A ⊆ carrier G" "B ⊆ carrier G"
  shows "generate G (A ∪ generate G B) = generate G (A ∪ B)"
  using generate_idem'[OF assms(2) assms(1)] by (simp add: sup_commute)

lemma (in group) generate_idem_Un:
  assumes "A ⊆ carrier G"
  shows "generate G (⋃x∈A. generate G {x}) = generate G A"
proof
  have "A ⊆ (⋃x∈A. generate G {x})" using generate.incl by force
  thus "generate G A ⊆ generate G (⋃x∈A. generate G {x})" using mono_generate
by presburger
  have "⋀x. x ∈ A ⟹ generate G {x} ⊆ generate G A" using mono_generate
by auto
  hence "(⋃x∈A. generate G {x}) ⊆ generate G A" by blast
  thus "generate G (⋃x∈A. generate G {x}) ⊆ generate G A"
    using generate_idem[OF assms] mono_generate by blast
qed

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
  have "f A ⊆ ⋃ {generate G {x} |x. x ∈ f A}"
  proof
    fix x
    assume x: "x ∈ f A"
    have "x ∈ generate G {x}" using generate.incl by fast
    thus "x ∈ ⋃ {generate G {x} |x. x ∈ f A}" using x by blast
  qed
  thus "generate G (f A) ⊆ generate G (⋃ {generate G {x} |x. x ∈ f A})"
using mono_generate by auto
  have "⋀x. x ∈ f A ⟹ generate G {x} ⊆ generate G (f A)" using mono_generate
by simp
  hence "(⋃ {generate G {x} |x. x ∈ f A}) ⊆ generate G (f A)" by blast
  with mono_generate[OF this] show "generate G (⋃ {generate G {x} |x.
x ∈ f A}) ⊆ generate G (f A)"
    using generate_idem[OF assms] by simp
```

```
qed

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

  have "⋀S. S ∈ A ⟹ generate G (f S) = generate G (⋃ {generate G
{x} |x. x ∈ f S})"
    using generate_idem_fUn[of f] assms by blast
  then have "generate G (⋃S ∈ A. generate G (f S))
           = generate G (⋃S ∈ A. generate G (⋃ {generate G {x} |x. x
∈ f S}))" by simp

  have "⋃ {generate G {x} |x. x ∈ ⋃ (f ‘ A)} ⊆ (⋃S∈A. generate G
(f S))"
  proof
    fix x
    assume x: "x ∈ ⋃ {generate G {x} |x. x ∈ ⋃ (f ‘ A)}"
    then obtain a where a: "x ∈ generate G {a}" "a ∈ ⋃ (f ‘ A)" by
blast
    then obtain M where M: "a ∈ f M" "M ∈ A" by blast
    then have "generate G {a} ⊆ generate G (f M)"
      using generate.incl[OF M(1), of G] mono_generate[of "{a}" "generate
G (f M)"]
            generate_idem assms by auto
    then have "x ∈ generate G (f M)" using a by blast
    thus "x ∈ (⋃S∈A. generate G (f S))" using M by blast
  qed
  thus "generate G (⋃ {generate G {x} |x. x ∈ ⋃ (f ‘ A)}) ⊆ generate
G (⋃S∈A. generate G (f S))"
    using mono_generate by simp
  have a: "generate G (⋃S∈A. generate G (f S)) ⊆ generate G (⋃ (f ‘
A))"
  proof -
    have "⋀S. S ∈ A ⟹ generate G (f S) ⊆ generate G (⋃ (f ‘ A))"
      using mono_generate[of _ "⋃ (f ‘ A)"] by blast
    then have "(⋃S∈A. generate G (f S)) ⊆ generate G (⋃ (f ‘ A))" by
blast
    then have "generate G (⋃S∈A. generate G (f S)) ⊆ generate G (generate
G (⋃ (f ‘ A)))"
      using mono_generate by meson
    thus "generate G (⋃S∈A. generate G (f S)) ⊆  generate G (⋃ (f ‘
A))"
      using generate_idem assms by blast
  qed
  have "⋃ {generate G {x} |x. x ∈ ⋃ (f ‘ A)} = (⋃x∈⋃ (f ‘ A). generate
G {x})" by blast
```

```
  with generate_idem_Un[OF assms]
  have "generate G (⋃ {generate G {x} |x. x ∈ ⋃ (f ` A)}) = generate
G (⋃ (f ` A))" by simp
  with a show "generate G (⋃S∈A. generate G (f S))
          ⊆ generate G (⋃ {generate G {x} |x. x ∈ ⋃ (f ` A)})" by
blast
qed
```

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"
  using assms generate_idem by (metis generate_idem' inf_sup_aci(5) sup.absorb2)

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(intro generate_eqI)
  show "A ⊆ carrier G" by fact
  show "B ⊆ carrier G" using assms generate_incl by blast
  show "A ⊆ generate G B" using assms generate_sincl[of B] by blast
  show "B ⊆ generate G A" using assms generate_sincl[of A] by blast
qed

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
  show "generate G B ⊆ generate G A" by (intro mono_generate, fact)
  show "generate G A ⊆ generate G B"
  proof(subst generate_idem[of "B", symmetric])
    show "generate G A ⊆ generate G (generate G B)"
      by (intro mono_generate, use assms generate_sincl[of B] in auto)
  qed (use assms in blast)
qed
```

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)"
  by (metis assms generate_idem')

lemma (in group) generate_subgroup_id:
  assumes "subgroup H G"
  shows "generate G H = H"
```

```
  using assms generateI by auto

lemma (in group) generate_consistent':
  assumes "subgroup H G" "A ⊆ H"
  shows "∀x ∈ A. generate G {x} = generate (G⦇carrier := H⦈) {x}"
  using generate_consistent assms by auto

lemma (in group) generate_singleton_one:
  assumes "generate G {a} = {𝟭}"
  shows "a = 𝟭"
  using generate.incl[of a "{a}" G] assms by auto

lemma (in group) generate_inv_eq:
  assumes "a ∈ carrier G"
  shows "generate G {a} = generate G {inv a}"
  by (intro generate_eqI;
      use assms generate.inv[of a] generate.inv[of "inv a" "{inv a}" G]
inv_inv[OF assms] in auto)

lemma (in group) generate_eq_imp_subset:
  assumes "generate G A = generate G B"
  shows "A ⊆ generate G B"
  using generate.incl assms by fast
```

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
  show "generate G A ⊆ generate G (A ∪ {𝟭})" by (intro mono_generate,
blast)
  show "generate G (A ∪ {𝟭}) ⊆ generate G A"
  proof(rule subsetI)
    show "x ∈ generate G A" if "x ∈ generate G (A ∪ {𝟭})" for x us-
ing that
      by (induction rule: generate.induct;
          use generate.one generate.incl generate.inv generate.eng in
auto)
  qed
qed

lemma (in group) generate_one_irrel':
  "generate G A = generate G (A - {𝟭})"
  using generate_one_irrel by (metis Un_Diff_cancel2)
```

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}}"
```

```
  using assms generate_pow_nat ord_elems_inf_carrier by auto

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 -
  have "{a [^] k |k. k ∈ {1..ord a}} = {a [^] k |k. k ∈ {0..ord a - 1}}"
  proof -
    have "a [^] k ∈ {a [^] k |k. k ∈ {0..ord a - 1}}" if "k ∈ {1..ord
a}" for k
      using that pow_nat_mod_ord[OF assms(2, 1), of "ord a"] assms by
(cases "k = ord a"; force)
    moreover have "a [^] k ∈ {a [^] k |k. k ∈ {1..ord a}}" if "k ∈ {0..ord
a - 1}" for k
    proof(cases "k = 0")
      case True
      hence "a [^] k = a [^] ord a" using pow_ord_eq_1[OF assms(2)] by
auto
      moreover have "ord a ∈ {1..ord a}"
        using assms unfolding atLeastAtMost_def atLeast_def atMost_def
by auto
      ultimately show ?thesis by blast
    next
      case False
      then show ?thesis using that by auto
    qed
    ultimately show ?thesis by blast
  qed
  with generate_nat_pow[OF assms] show ?thesis by simp
qed

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"
  by blast

lemma card_inter_eq:
  assumes "finite A" "card (A ∩ B) = card A"
  shows "A ⊆ B"
proof -
  have "A ∩ B ⊆ A" by blast
```

```
  with assms have "A ∩ B = A" using card_subset_eq by blast
  thus ?thesis by blast
qed

lemma coprime_eq_empty_prime_inter:
  assumes "(n::nat) ≠ 0" "m ≠ 0"
  shows "coprime n m ⟷ (prime_factors n) ∩ (prime_factors m) = {}"
proof
  show "coprime n m ⟹ prime_factors n ∩ prime_factors m = {}"
  proof (rule ccontr)
    assume cp: "coprime n m"
    assume pf: "prime_factors n ∩ prime_factors m ≠ {}"
    then obtain p where p: "p ∈ prime_factors n" "p ∈ prime_factors
m" by blast
    then have p_dvd: "p dvd n" "p dvd m" by blast+
    moreover have "¬is_unit p" using p using not_prime_unit by blast
    ultimately show "False" using cp unfolding coprime_def by simp
  qed
  assume assm: "prime_factors n ∩ prime_factors m = {}"
  show "coprime n m" unfolding coprime_def
  proof
    fix c
    show "c dvd n ⟶ c dvd m ⟶ is_unit c"
    proof(rule; rule)
      assume c: "c dvd n" "c dvd m"
      then have "prime_factors c ⊆ prime_factors n" "prime_factors c
⊆ prime_factors m"
        using assms dvd_prime_factors by blast+
      then have "prime_factors c = {}" using assm by blast
      thus "is_unit c" using assms c
        by (metis dvd_0_left_iff prime_factorization_empty_iff set_mset_eq_empty_iff)
    qed
  qed
qed

lemma prime_factors_Prod:
  assumes "finite S" "⋀a. a ∈ S ⟹ f a ≠ 0"
  shows "prime_factors (prod f S) = ⋃(prime_factors ` f ` S)"
  using assms
proof(induction S rule: finite_induct)
  case empty
  then show ?case by simp
next
  case i: (insert x F)
  from i have x: "f x ≠ 0" by blast
  from i have F: "prod f F ≠ 0" by simp
  from i have "prime_factors(prod f F) = ⋃ (prime_factors ` f ` F)"
by blast
  moreover have "prod f (insert x F) = (prod f F) * f x" using i mult.commute
```

```
by force
  ultimately
  have "prime_factors (prod f (insert x F)) = (⋃(prime_factors ‘ f ‘
F)) ∪ prime_factors (f x)"
    using prime_factors_product[OF F x] by argo
  thus ?case by force
qed

lemma lcm_is_Min_multiple_nat:
  assumes "c ≠ 0" "(a::nat) dvd c" "(b::nat) dvd c"
  shows "c ≥ lcm a b"
  using lcm_least[of a c b] assms by fastforce

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)"
  using assms
  by (metis power_0 power_one_right prime_dvd_power prime_imp_power_coprime_nat
            prime_nat_iff prime_power_inj'')

lemma "finite (prime_factors x)"
  using finite_set_mset by blast

lemma card_ge_1_two_diff:
  assumes "card A > 1"
  obtains x y where "x ∈ A" "y ∈ A" "x ≠ y"
proof -
  have fA: "finite A" using assms by (metis card.infinite not_one_less_zero)
  from assms obtain x where x: "x ∈ A" by fastforce
  with assms fA have "card (A - {x}) > 0" by simp
  then obtain y where y: "y ∈ (A - {x})" by (metis card_gt_0_iff ex_in_conv)
  thus ?thesis using that[of x y] x by blast
qed

lemma infinite_two_diff:
  assumes "infinite A"
  obtains x y where "x ∈ A" "y ∈ A" "x ≠ y"
proof -
  from assms obtain x where x: "x ∈ A" by fastforce
  from assms have "infinite (A - {x})" by simp
  then obtain y where y: "y ∈ (A - {x})"
    by (metis ex_in_conv finite.emptyI)
  show ?thesis using that[of x y] using x y by blast
qed

lemma Inf_le:
  "Inf A ≤ x" if "x ∈ (A::nat set)" for x
proof (cases "A = {}")
```

```
  case True
  then show ?thesis using that by simp
next
  case False
  hence "Inf A ≤ Inf {x}" using that by (simp add: cInf_lower)
  also have "... = x" by simp
  finally show "Inf A ≤ x" by blast
qed

lemma switch_elem_card_le:
  assumes "a ∈ A"
  shows "card (A - {a} ∪ {b}) ≤ card A"
  using assms
  by (metis Diff_insert_absorb Set.set_insert Un_commute card.infinite
card_insert_disjoint
            card_mono finite_insert insert_is_Un insert_subset order_refl)

lemma pairwise_coprime_dvd:
  assumes "finite A" "pairwise coprime A" "(n::nat) = prod id A" "∀a∈A.
a dvd j"
  shows "n dvd j"
  using assms
proof (induction A arbitrary: n)
  case i: (insert x F)
  have "prod id F dvd j" "x dvd j" using i unfolding pairwise_def by
auto
  moreover have "coprime (prod id F) x"
    by (metis i(2, 4) id_apply pairwise_insert prod_coprime_left)
  ultimately show ?case using i(1, 2, 5) by (simp add: coprime_commute
divides_mult)
qed simp

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"
  using assms
proof (induction A arbitrary: n)
  case i: (insert x F)
  have "prod f F dvd j" "f x dvd j" using i unfolding pairwise_def by
auto
  moreover have "coprime (prod f F) (f x)" by(intro prod_coprime_left,
use i in blast)
  ultimately show ?case using i by (simp add: coprime_commute divides_mult)
qed simp

lemma transp_successively_remove1:
  assumes "transp f" "successively f l"
```

```
  shows "successively f (remove1 a l)" using assms(2)
proof(induction l rule: induct_list012)
  case (3 x y zs)
  from 3(3)[unfolded successively.simps] have fs: "f x y" "successively
f (y # zs)" by auto
  moreover from this(2) successively.simps have s: "successively f zs"
by(cases zs, auto)
  ultimately have s2: "successively f (remove1 a zs)" "successively f
(remove1 a (y # zs))"
    using 3 by auto
  consider (x) "x = a" | (y) "y = a ∧ x ≠ a" | (zs) "a ≠ x ∧ a ≠ y"
by blast
  thus ?case
  proof (cases)
    case x
    then show ?thesis using 3 by simp
  next
    case y
    then show ?thesis
    proof (cases zs)
      case Nil
      then show ?thesis using fs by simp
    next
      case (Cons a list)
      hence "f y a" using fs by simp
      hence "f x a" using fs(1) assms(1)[unfolded transp_def] by blast
      then show ?thesis using Cons y s by auto
    qed
  next
    case zs
    then show ?thesis using s2 fs by auto
  qed
qed auto


lemma exp_one_2pi_iff:
  fixes x::real shows "exp (2 * of_real pi * i * x) = 1 ⟷ x ∈ ℤ"
proof -
  have c: "cis (2 * x * pi) = 1 ⟷ x ∈ ℤ"
    by (auto simp: complex_eq_iff sin_times_pi_eq_0 cos_one_2pi_int, meson
Ints_cases)
  have "exp (2 * of_real pi * i * x) = exp (i * complex_of_real (2 * x
* pi))"
  proof -
    have "2 * of_real pi * i * x = i * complex_of_real (2 * x * pi)" by
simp
    thus ?thesis by argo
  qed
  also from cis_conv_exp have "... = cis (2 * x * pi)" by simp
```

```
  finally show ?thesis using c by simp
qed


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
  assume *: "(of_int a / of_int b :: 'a :: field_char_0) ∈ ℤ"
  from * obtain n where "of_int a / of_int b = (of_int n :: 'a)"
    by (elim Ints_cases)
  hence "of_int (b * n) = (of_int a :: 'a)"
    using assms by (subst of_int_mult) (auto simp: field_simps)
  hence "b * n = a"
    by (subst (asm) of_int_eq_iff)
  thus "b dvd a" by auto
qed auto


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"
  using of_int_divide_in_Ints_iff[of "int b" "int a"] assms by simp

lemma true_nth_unity_root:
  fixes n::nat
  obtains x::complex where "x ^ n = 1" "⋀m. ⟦0<m; m<n⟧ ⟹ x ^ m ≠ 1"
proof(cases "n = 0")
  case False
  show ?thesis
  proof (rule that)
    show "cis (2 * pi / n) ^ n = 1"
      by (simp add: DeMoivre)
  next
    fix m assume m: "m > 0" "m < n"
    have "cis (2 * pi / n) ^ m = cis (2 * pi * m / n)"
      by (simp add: DeMoivre algebra_simps)
    also have "... = 1 ⟷ real m / real n ∈ ℤ"
      using exp_one_2pi_iff[of "m / n"] by (simp add: cis_conv_exp algebra_simps)
    also have "... ⟷ n dvd m"
      using m by (subst of_nat_divide_in_Ints_iff) auto
    also have "¬n dvd m"
      using m by auto
    finally show "cis (2 * pi / real n) ^ m ≠ 1" .
  qed
qed simp
```

```
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 (intro bij_betw_imageI)
  show "inj_on f A" by fact
  show "f ` A = B"
  proof -
    have "card (f ` A) = card B" using assms by (simp add: card_image)
    moreover have "f ` A ⊆ B" using assms by blast
    ultimately show ?thesis using assms by (meson card_subset_eq)
  qed
qed

lemma powi_mod:
  "x powi m = x powi (m mod n)" if "x ^ n = 1" "n > 0" for x::complex and m::int
proof -
  have xnz: "x ≠ 0" using that by (metis zero_neq_one zero_power)
  obtain k::int where k: "m = k*n + (m mod n)" using div_mod_decomp_int by blast
  have "x powi m = x powi (k*n) * x powi (m mod n)" by (subst k, intro power_int_add, use xnz in auto)
  moreover have "x powi (k*n) = 1" using that
    by (metis mult.commute power_int_1_left power_int_mult power_int_of_nat)
  ultimately show ?thesis by force
qed


lemma Sigma_insert: "Sigma (insert x A) B = (λy. (x, y)) ` B x ∪ Sigma A B"
  by auto

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"
  unfolding complementary_def by blast

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 -
  have "1 ∈ I" "1 ∈ K" using subgroup.one_closed assms by fastforce+
  moreover have "I ∩ K ⊆ H ∩ J" using subgroup.subset assms by force
  ultimately show ?thesis using assms unfolding complementary_def by
blast
qed

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"
  by (intro subgroup_carrier_complementary[OF assms(5), of H I] subgroup_incl,
use assms in auto)

lemma (in group) complementary_subgroup_iff:
  assumes "subgroup H G"
  shows "complementary A B ⟷ group.complementary (G(|carrier := H|))
A B"
proof -
  interpret H: group "G(|carrier := H|)" using subgroup.subgroup_is_group
assms by blast
  have "1_G = 1_G(|carrier := H|)" by simp
  then show ?thesis unfolding complementary_def H.complementary_def by
simp
qed

lemma (in group) subgroups_card_coprime_imp_compl:
  assumes "subgroup H G" "subgroup J G" "coprime (card H) (card J)"
  shows "complementary H J" unfolding complementary_def
proof -
  interpret JH: subgroup "(H ∩ J)" G using assms subgroups_Inter_pair
by blast
  from subgroups_card_coprime_inter_card_one[OF assms] show "H ∩ J =
{1}" using JH.one_closed
    by (metis card_1_singletonE singletonD)
qed

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 -
  from assms have "coprime (card P) (card Q)"
    by (metis coprime_power_right_iff primes_coprime coprime_def)
  then show ?thesis using subgroups_card_coprime_imp_compl assms complementary_def
by blast
qed
```

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 (rule ccontr; safe)
  fix a b
  assume ab: "a ∈ J" "b ∈ J" "a ≠ b"
  then have [simp]: "a ∈ carrier G" "b ∈ carrier G" using assms subgroup.subset
by auto
  assume "H #> a = H #> b"
  then have "a ⊗ inv b ∈ H" using assms(1, 2) ab
    by (metis comm_group_axioms comm_group_def rcos_self
              subgroup.mem_carrier subgroup.rcos_module_imp)
  moreover have "a ⊗ inv b ∈ J"
    by (rule subgroup.m_closed[OF assms(2) ab(1) subgroup.m_inv_closed[OF
assms(2) ab(2)]])
  moreover have "a ⊗ inv b ≠ 1" using ab inv_equality by fastforce
  ultimately have "H ∩ J ≠ {1}" by blast
  thus False using assms(5) unfolding complementary_def by blast
qed

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"
  unfolding complementary_def
proof (rule ccontr)
  assume "H ∩ J ≠ {1}"
  have "1 ∈ H" using subgroup.one_closed assms(1) by blast
  moreover have "1 ∈ J" using subgroup.one_closed assms(2) by blast
  ultimately have "1 ∈ (H ∩ J)" by blast

  then obtain a where a_def: "a ∈ (H ∩ J) ∧ a ≠ 1" using ‹H ∩ J ≠
{1}› by blast
  then have aH: "a ∈ H" by blast
  then have a_inv_H: "inv a ∈ H ∧ inv a ≠ 1" using assms(1)
    by (meson a_def inv_eq_1_iff subgroup.mem_carrier subgroupE(3))
  from a_def have aJ: "a ∈ J" by blast
```

```
  then have a_inv_J: "inv a ∈ J ∧ inv a ≠ 1" using assms(2)
    by (meson a_def inv_eq_1_iff subgroup.mem_carrier subgroupE(3))
  from a_def have a_c: "a ∈ carrier G" using subgroup.subset[of J G]
assms(2) by blast

  from set_mult_card_eq_impl_empty_inter'[of H J]
  have empty: "⋀a b. ⟦a ∈ H; b ∈ H; a ≠ b⟧ ⟹ (l_coset G a J) ∩ (l_coset
G b J) = {}"
    using assms subgroup.subset[of _ G] by simp

  have "1 ∈ 1 <# J" using ‹1 ∈ J› unfolding l_coset_def by force
  moreover have "1 ∈ a <# J" using a_inv_J aJ a_c assms ‹1 ∈ J› coset_join3
by blast
  ultimately have "(l_coset G 1 J) ∩ (l_coset G a J) ≠ {}" by blast

  then show "False" using empty[of "1" a] a_def aH ‹1 ∈ H› by blast
qed

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 -
  have carr: "H ⊆ carrier G" "J ⊆ carrier G" using assms subgroup.subset
by auto

  from coset_neq_imp_empty_inter[OF assms(1)] compl_imp_diff_cosets[OF
assms(1,2)]
  have em_inter: "⋀a b. ⟦a ∈ J; b ∈ J; a ≠ b⟧ ⟹ (H #> a) ∩ (H #>
b) = {}"
    by (meson assms subgroup.mem_carrier)

  have "card (⋃a∈J. (H #> a)) = card J * card H" using assms(4) carr(2)
em_inter
  proof (induction J rule: finite_induct)
    case empty
    then show ?case by auto
  next
    case i: (insert x F)
    then have cF:"card (⋃ ((#>) H ` F)) = card F * card H" by blast
    have xc[simp]: "x ∈ carrier G" using i(4) by simp
    let ?J = "insert x F"
    from i(2, 4, 5) have em:"(H #> x) ∩ (⋃y∈F. (H #> y)) = {}" by auto
    have "finite (H #> x)"
      by (meson carr(1) rcosetsI rcosets_finite assms(3) xc)
    moreover have "finite (H <#> F)" using set_mult_finite[OF assms(3)
i(1) carr(1)] i(4) by blast
    moreover have "H <#> F = (⋃a∈F. (H #> a))"
      unfolding set_mult_def using r_coset_def[of G H] by auto
```

```
    ultimately have "card(H #> x) + card(⋃y∈F. (H #> y))
                    = card((H #> x) ∪ (⋃y∈F. (H #> y))) + card((H #>
x) ∩ (⋃y∈F. (H #> y)))"
      using card_Un_Int by auto
    then have "card(H #> x) + card(⋃y∈F. (H #> y)) = card((H #> x) ∪
(⋃y∈F. (H #> y)))"
      using i(5) em by simp
    moreover have "card (H #> x) = card H"
      using card_rcosets_equal[of _ H] rcosetsI[of H] carr(1) xc by metis
    moreover have "card (insert x F) * card H = card F * card H + card
H"
      by (simp add: i)
    ultimately show ?case using cF by simp
  qed
  moreover have "H <#> J = (⋃a∈J. (H #> a))"
    unfolding set_mult_def using r_coset_def[of G H] by auto
  ultimately show "card (H <#> J) = card J * card H" by argo
qed

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"
  using finite_sub_comp_imp_card_eq_mult[OF assms] finite_sub_card_eq_mult_imp_comp[OF
assms]
  by blast
```

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"
  unfolding IDirProd_def by (simp add: sup_commute)

lemma (in group) IDirProd_empty_right:
  assumes "A ⊆ carrier G"
  shows "IDirProd G A {} = generate G A"
  unfolding IDirProd_def by simp

lemma (in group) IDirProd_empty_left:
  assumes "A ⊆ carrier G"
  shows "IDirProd G {} A = generate G A"
  unfolding IDirProd_def by simp
```

```
lemma (in group) IDirProd_one_right:
  assumes "A ⊆ carrier G"
  shows "IDirProd G A {1} = generate G A"
  unfolding IDirProd_def
proof
  interpret sA: subgroup "(generate G A)" G using assms generate_is_subgroup
by simp
  interpret sAone: subgroup "(generate G (A ∪ {1}))" G using assms generate_is_subgroup
by simp
  show "generate G (A ∪ {1}) ⊆ generate G A"
    using generate_subgroup_incl[of "A ∪ {1}" "generate G A"]
          generate.incl assms sA.one_closed sA.subgroup_axioms by fast
  show "generate G A ⊆ generate G (A ∪ {1})"
    using mono_generate[of A "A ∪ {1}"] by blast
qed

lemma (in group) IDirProd_one_left:
  assumes "A ⊆ carrier G"
  shows "IDirProd G {1} A = generate G A"
  using IDirProd_one_right[of A] assms unfolding IDirProd_def by force

lemma (in group) IDirProd_is_subgroup:
  assumes "Y ⊆ carrier G" "Z ⊆ carrier G"
  shows "subgroup (IDirProd G Y Z) G"
  unfolding IDirProd_def using generate_is_subgroup[of "Y ∪ Z"] assms
by simp
```

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"
  unfolding IDirProd_def
  by (rule set_mult_eq_generate_subgroup[OF assms])

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"
  using finite_sub_comp_iff_card_eq_mult IDirProd_eq_subgroup_mult assms
by presburger
```

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"
  unfolding IDirProds_def using assms generate.incl[of _ "⋃(S ` I)" G]
by blast

lemma (in group) IDirProds_empty:
  "IDirProds G S {} = {1}"
  unfolding IDirProds_def using generate_empty by simp

lemma (in group) IDirProds_is_subgroup:
  assumes "⋃(S ` I) ⊆ (carrier G)"
  shows "subgroup (IDirProds G S I) G"
  unfolding IDirProds_def using generate_is_subgroup[of "⋃(S ` I)"] assms
by auto

lemma (in group) IDirProds_subgroup_id: "subgroup (S i) G ⟹ IDirProds
G S {i} = S i"
  by (simp add: generate_subgroup_id IDirProds_def)

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 -
  have subset: "⋃ (S ` A) ⊆ carrier G" "⋃ (S ` B) ⊆ carrier G"
    using subgroup.subset assms(1, 2) by blast+
  have "IDirProds G S A <#> IDirProds G S B = IDirProd G (IDirProds G
S A) (IDirProds G S B)"
    using assms by (intro IDirProd_eq_subgroup_mult [symmetric] IDirProds_is_subgroup
subset)
  also have "... = generate G (⋃ (S ` A) ∪ IDirProds G S B)"
    unfolding IDirProds_def IDirProd_def by (intro generate_idem' generate_incl
subset)
  also have "... = generate G (⋃(S ` A) ∪ ⋃(S ` B))"
    unfolding IDirProds_def IDirProd_def
    by (intro generate_idem'_right generate_incl subset)
  also have "⋃(S ` A) ∪ ⋃(S ` B) = ⋃(S ` (A ∪ B))"
    by blast
  also have "generate G ... = IDirProds G S (A ∪ B)"
```

```
    unfolding IDirProds_def ..
  finally show ?thesis ..
qed

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)" using assms
proof (induction I rule: finite_induct)
  case empty
  thus ?case using IDirProds_empty[of S] by simp
next
  case i: (insert x I)
  interpret Sx: subgroup "S x" G using i by blast
  have cx: "(S x) ⊆ carrier G" by force
  have cI: "⋃(S ` I) ⊆ carrier G" using i subgroup.subset by blast
  interpret subgroup "IDirProds G S I" G using IDirProds_is_subgroup[OF
cI] .
  have cIP: "(IDirProds G S I) ⊆ carrier G" by force
  from i have f: "finite (S x)" "finite (IDirProds G S I)" "finite {x}"
by blast+
  from IDirProds_Un[of "{x}" S I]
  have "IDirProds G S ({x} ∪ I) = IDirProds G S {x} <#> IDirProds G S
I" using i by blast
  also have "... = S x <#> IDirProds G S I"
    using IDirProds_subgroup_id[of S x] Sx.subgroup_axioms by force
  also have "finite (...)" using set_mult_finite[OF f(1, 2) cx cIP] .
  finally show ?case unfolding insert_def by simp
qed

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 -
  have "S i ⊆ IDirProds G S I" using assms IDirProds_incl by fast
  then have "H ∩ (S i) ⊆ H ∩ IDirProds G S I" by blast
  moreover have "1 ∈ H ∩ (S i)" using subgroup.one_closed assms by
auto
  ultimately show "complementary H (S i)" using assms(3) unfolding complementary_def
by blast
qed
```

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))" using assms
proof (induction I rule: finite_induct)
  case empty
  then show ?case using IDirProds_empty[of S] by simp
next
  case i: (insert x I)
  have sx: "subgroup (S x) G" using i(4) by blast
  have cx: "(S x) ⊆ carrier G" using subgroup.subset[OF sx] .
  have fx: "finite (S x)" using i by blast
  have cI: "⋃(S ` I) ⊆ carrier G" using subgroup.subset[of _ G] i(4)
by blast
  from generate_is_subgroup[OF this] have sIP: "subgroup (IDirProds G
S I) G"
    unfolding IDirProds_def .
  then have cIP: "(IDirProds G S I) ⊆ carrier G" using subgroup.subset
by blast
  have fIP: "finite (IDirProds G S I)" using IDirProds_finite[OF i(1)]
i by blast

  from i have ih: "card (IDirProds G S I) = (∏i∈I. card (S i))" un-
folding pairwise_def by blast
  hence cop: "coprime (card (IDirProds G S I)) (card (S x))"
  proof -
    have cFI0: "card (IDirProds G S I) ≠ 0" using finite_subgroup_card_neq_0[OF
sIP fIP] .
    moreover have cx0: "card (S x) ≠ 0" using finite_subgroup_card_neq_0[OF
sx fx] .
    moreover have  "prime_factors (card (IDirProds G S I)) ∩ prime_factors
(card (S x)) = {}"
    proof (rule ccontr)
      have n0: "⋀i. i ∈ I ⟹ card (S i) ≠ 0" using finite_subgroup_card_neq_0
i(4, 5) by blast
      assume "prime_factors (card (IDirProds G S I)) ∩ prime_factors
(card (S x)) ≠ {}"
      moreover have "prime_factors (card (IDirProds G S I)) = ⋃(prime_factors
` (card ∘ S) ` I)"
        using n0 prime_factors_Prod[OF i(1), of "card ∘ S"] by (subst
ih; simp)
      moreover have "⋀i. i ∈ I ⟹ prime_factors (card (S i)) ∩ prime_factors
(card (S x)) = {}"
      proof -
        fix i
        assume ind: "i ∈ I"
        then have coPx: "coprime (card (S i)) (card (S x))"
          using i(2, 6) unfolding pairwise_def by auto
        have "card (S i) ≠ 0" using n0 ind by blast
        from coprime_eq_empty_prime_inter[OF this cx0]
        show "prime_factors (card (S i)) ∩ prime_factors (card (S x))
= {}" using coPx by blast
```

```
      qed
      ultimately show "False" by auto
    qed
    ultimately show ?thesis using coprime_eq_empty_prime_inter by blast
  qed
  have "card (IDirProds G S (insert x I)) = card (S x) * card (IDirProds
G S I)"
  proof -
    from finite_sub_comp_iff_card_eq_IDirProd[OF sIP sx fIP fx]
         subgroups_card_coprime_imp_compl[OF sIP sx cop]
    have "card (IDirProd G (IDirProds G S I) (S x)) = card (S x) * card
(IDirProds G S I)" by blast
    moreover have "generate G (⋃ (S ` insert x I)) = generate G (generate
G (⋃ (S ` I)) ∪ S x)"
      by (simp add: Un_commute cI cx generate_idem'_right)
    ultimately show ?thesis unfolding IDirProds_def IDirProd_def by argo
  qed
  also have "... = card (S x) * prod (card ∘ S) I" using ih by simp
  also have "... = prod (card ∘ S) ({x} ∪ I)" using i.hyps by auto
  finally show ?case by simp
qed
```

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 {}"
  unfolding compl_fam_def by simp

lemma (in group) compl_fam_cong:
  assumes "compl_fam (f ∘ g) A" "inj_on g A"
  shows "compl_fam f (g ` A)"
proof -
  have "((f ∘ g) ` (A - {i})) =  (f ` (g ` A - {g i}))" if "i ∈ A" for
i
    using assms that unfolding inj_on_def comp_def by blast
  thus ?thesis using assms unfolding compl_fam_def IDirProds_def complementary_def
by simp
qed
```

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(rule, rule ccontr)
  fix x y
  assume xy: "x ∈ gs" "y ∈ gs" "x ≠ y"
  have gen: "generate G (⋃g∈gs - {y}. generate G {g}) = generate G (gs
- {y})"
    by (intro generate_idem_Un, use assms in blast)
  assume g: "generate G {x} = generate G {y}"
  with xy have "generate G {y} ⊆ generate G (gs - {y})" using mono_generate[of
"{x}" "gs - {y}"] by auto
  with xy have gyo: "generate G {y} = {1}" using assms(2) generate.one
gen
    unfolding compl_fam_def complementary_def IDirProds_def by blast
  hence yo: "y = 1" using generate_singleton_one by simp
  from gyo g generate_singleton_one have xo: "x = 1" by simp
  from xy yo xo show False by blast
qed

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(unfold compl_fam_def complementary_def IDirProds_def, subst generate_idem_Un)
  show "⋀i. A - {i} ⊆ carrier G" using assms by blast
  have "generate G {i} ∩ generate G (A - {i}) = {1}" if "i ∈ A" for i
  proof -
    have "1 ∈ generate G {i} ∩ generate G (A - {i})" using generate.one
by blast
    moreover have "generate G (A - {i}) ⊆ generate G (gs - {i})"
      by (intro mono_generate, use assms in fast)
    moreover have "generate G {i} ∩ generate G (gs - {i}) = {1}"
      using assms that generate_idem_Un[of "gs - {i}"]
      unfolding compl_fam_def IDirProds_def complementary_def by blast
    ultimately show ?thesis by blast
  qed
  thus "∀i∈A. generate G {i} ∩ generate G (A - {i}) = {1}" by auto
qed
```

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"
  unfolding is_idirprod_def using assms subgroup_imp_normal by blast

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(intro is_idirprod_subgroup_suffices)
  show "A = IDirProds G (λg. generate G {g}) gs"
    using assms generate_idem_Un[OF assms(2)] unfolding IDirProds_def
by argo
  show "∀i∈gs. subgroup (generate G {i}) G" using assms generate_is_subgroup
by auto
  show "compl_fam (λg. generate G {g}) gs" by fact
qed

lemma (in comm_group) is_idirprod_imp_compl_fam[simp]:
  assumes "is_idirprod A S I"
  shows "compl_fam S I"
  using assms unfolding is_idirprod_def by blast

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 -
  have "gs ⊆ carrier G"
  proof
    show "g ∈ carrier G" if "g ∈ gs" for g
    proof -
      interpret g: subgroup "generate G {g}" G
        using assms that normal_imp_subgroup unfolding is_idirprod_def
by blast
      show ?thesis using g.subset generate.incl by fast
    qed
  qed
  thus ?thesis using assms generate_idem_Un unfolding is_idirprod_def
IDirProds_def by presburger
qed

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 -
  from assms have "A = insert a (A - {a})" by blast
  then have "finprod G f A = finprod G f (insert a (A - {a}))" by simp
  also have "... = f a ⊗ finprod G f (A - {a})" using assms by (intro
finprod_insert, auto)
  finally show ?thesis .
qed

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 -
  from assms have "A = insert a (A - {a})" by blast
  then have "finprod G f A = finprod G f (insert a (A - {a}))" by simp
  also have "... = f a ⊗ finprod G f (A - {a})" using assms by (intro
finprod_insert, auto)
  also have "... = finprod G f (A - {a}) ⊗ f a"
    by (intro m_comm, use assms in blast, intro finprod_closed, use assms
in blast)
  finally show ?thesis .
qed
```

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 -
  have "finprod G f {x} = f x ⊗ finprod G f {}" using finprod_minus[of
x "{x}" f] assms by auto
  thus ?thesis using assms by simp
qed
```

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 (cases "finite S")
  case True
  interpret H: comm_group "G⦇carrier := H⦈" using subgroup_is_comm_group[OF
assms(2)] .
  show ?thesis using True assms
  proof (induction S rule: finite_induct)
    case empty
    then show ?case using finprod_empty H.finprod_empty by simp
  next
    case i: (insert x F)
    then have "finprod G f F = finprod (G⦇carrier := H⦈) f F" by blast
    moreover have "finprod G f (insert x F) = f x ⊗ finprod G f F"
    proof(intro finprod_insert[OF i(1, 2), of f])
      show "f ∈ F → carrier G" "f x ∈ carrier G" using i(4) subgroup.subset[OF
i(5)] by blast+
    qed
    ultimately have "finprod G f (insert x F) = f x ⊗G⦇carrier := H⦈ finprod
(G⦇carrier := H⦈) f F"
      by auto
    moreover have "finprod (G⦇carrier := H⦈) f (insert x F) = ... "
    proof(intro H.finprod_insert[OF i(1, 2)])
      show "f ∈ F → carrier (G⦇carrier := H⦈)" "f x ∈ carrier (G⦇carrier
:= H⦈)" using i(4) by auto
    qed
    ultimately show ?case by simp
  qed
next
  case False
  then show ?thesis unfolding finprod_def by simp
qed

lemma (in comm_group) finprod_closed_subgroup:
  assumes "subgroup H G" "f ∈ A → H"
  shows "finprod G f A ∈ H"
  using assms(2)
proof (induct A rule: infinite_finite_induct)
case (infinite A)
then show ?case using subgroup.one_closed[OF assms(1)] by auto
next
  case empty
  then show ?case using subgroup.one_closed[OF assms(1)] by auto
next
  case i: (insert x F)
  from finprod_insert[OF i(1, 2), of f] i have fi: "finprod G f (insert
x F) = f x ⊗ finprod G f F"
    using subgroup.subset[OF assms(1)] by blast
  from i have "finprod G f F ∈ H" "f x ∈ H" by blast+
  with fi show ?case using subgroup.m_closed[OF assms(1)] by presburger
```

```
qed
```

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"
  using assms
proof(induction A rule: infinite_finite_induct)
  case i: (insert x F)
  hence ih: "finprod G f F [^] k = finprod G ((λa. a [^] k) ∘ f) F" by
blast
  have fpc: "finprod G f F ∈ carrier G" by (intro finprod_closed, use
i in auto)
  have fxc: "f x ∈ carrier G" using i by auto
  have "finprod G f (insert x F) = f x ⊗ finprod G f F" by (intro finprod_insert,
use i in auto)
  hence "finprod G f (insert x F) [^] k = (f x ⊗ finprod G f F) [^] k"
by simp
  also have "... = f x [^] k ⊗ finprod G f F [^] k" using fpc fxc int_pow_distrib
by blast
  also have "... = ((λa. a [^] k) ∘ f) x ⊗ finprod G ((λa. a [^] k) ∘
f) F" using ih by simp
  also have "... = finprod G ((λa. a [^] k) ∘ f) (insert x F)"
    by (intro finprod_insert[symmetric], use i in auto)
  finally show ?case .
qed auto
```

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" using assms
proof(induct A rule: infinite_finite_induct)
  case (infinite A)
  then show ?case by simp
next
  case empty
  then show ?case by simp
next
  case i: (insert x F)
  then have iH: "finprod G h F = finprod G f F ⊗ finprod G g F" by fast
  have f: "finprod G f (insert x F) = f x ⊗ finprod G f F"
    by (intro finprod_insert[OF i(1, 2), of f]; use i(5) in simp)
  have g: "finprod G g (insert x F) = g x ⊗ finprod G g F"
    by (intro finprod_insert[OF i(1, 2), of g]; use i(6) in simp)
```

```
  have h: "finprod G h (insert x F) = h x ⊗ finprod G h F"
    by (intro finprod_insert[OF i(1, 2), of h]; use i(7) in simp)
  also have "... = h x ⊗ (finprod G f F ⊗ finprod G g F)" using iH by
argo
  also have "... = f x ⊗ g x ⊗ (finprod G f F ⊗ finprod G g F)" us-
ing i(4) by simp
  also have "... = f x ⊗ finprod G f F ⊗ (g x ⊗ finprod G g F)" us-
ing m_comm m_assoc i(5-7) by simp
  also have "... = finprod G f (insert x F) ⊗ finprod G g (insert x F)"
using f g by argo
  finally show ?case .
qed

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"
  using finprod_reindex[OF _ assms(1), of f] using assms(2) unfolding
comp_def by blast
```

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
  show "?g ⊆ ?fp"
  proof
    fix x
    assume x: "x ∈ ?g"
    thus "x ∈ ?fp"
    proof (induction rule: generate.induct)
      case one
      show ?case
      proof
        let ?r = "restrict (λ_. 1) gs"
        show "?r ∈ (Π_E a∈gs. generate G {a})" using generate.one by
auto
        show "1 = finprod G ?r gs" by(intro finprod_one_eqI[symmetric],
simp)
      qed
    next
      case g: (incl g)
      show ?case
      proof
        let ?r = "restrict ((λ_. 1)(g := g)) gs"
        show "?r ∈ (Π_E a∈gs. generate G {a})" using generate.one generate.incl[of
g "{g}" G]
```

```
        by fastforce
      show "g = finprod G ?r gs"
      proof -
        have "finprod G ?r gs = ?r g ⊗ finprod G ?r (gs - {g})"
          by (intro finprod_minus, use assms g in auto)
        moreover have "?r g = g" using g by simp
        moreover have "finprod G ?r (gs - {g}) = 1" by(rule finprod_one_eqI;
use g in simp)
        ultimately show ?thesis using assms g by auto
      qed
    qed
  next
    case g: (inv g)
    show ?case
    proof
      let ?r = "restrict ((λ_. 1)(g := inv g)) gs"
      show "?r ∈ (Π_E a∈gs. generate G {a})" using generate.one generate.inv[of
g "{g}" G]
        by fastforce
      show "inv g = finprod G ?r gs"
      proof -
        have "finprod G ?r gs = ?r g ⊗ finprod G ?r (gs - {g})"
          by (intro finprod_minus, use assms g in auto)
        moreover have "?r g = inv g" using g by simp
        moreover have "finprod G ?r (gs - {g}) = 1" by(rule finprod_one_eqI;
use g in simp)
        ultimately show ?thesis using assms g by auto
      qed
    qed
  next
    case gh: (eng g h)
    from gh obtain i where i: "i ∈ (Π_E a∈gs. generate G {a})" "g
= finprod G i gs" by blast
    from gh obtain j where j: "j ∈ (Π_E a∈gs. generate G {a})" "h
= finprod G j gs" by blast
    from i j have "g ⊗ h = finprod G i gs ⊗ finprod G j gs" by blast
    also have "... = finprod G (λa. i a ⊗ j a) gs"
    proof(intro finprod_multf[symmetric]; rule)
      fix x
      assume x: "x ∈ gs"
      have "i x ∈ generate G {x}" "j x ∈ generate G {x}"using i(1)
j(1) x by blast+
      thus "i x ∈ carrier G" "j x ∈ carrier G" using generate_incl[of
"{x}"] x assms(2) by blast+
    qed
    also have "... = finprod G (restrict (λa. i a ⊗ j a) gs) gs"
    proof(intro finprod_cong)
      have ip: "i g ∈ generate G {g}" if "g ∈ gs" for g using i that
by auto
```

```
        have jp: "j g ∈ generate G {g}" if "g ∈ gs" for g using j that
by auto
        have "i g ⊗ j g ∈ generate G {g}" if "g ∈ gs" for g
          using generate.eng[OF ip[OF that] jp[OF that]] .
        thus "((λa. i a ⊗ j a) ∈ gs → carrier G) = True" using generate_incl
assms(2) by blast
      qed auto
      finally have "g ⊗ h = finprod G (restrict (λa. i a ⊗ j a) gs) gs"
.
      moreover have "(restrict (λa. i a ⊗ j a) gs) ∈ (Π_E a∈gs. generate
G {a})"
      proof -
        have ip: "i g ∈ generate G {g}" if "g ∈ gs" for g using i that
by auto
        have jp: "j g ∈ generate G {g}" if "g ∈ gs" for g using j that
by auto
        have "i g ⊗ j g ∈ generate G {g}" if "g ∈ gs" for g
          using generate.eng[OF ip[OF that] jp[OF that]] .
        thus ?thesis by auto
      qed
      ultimately show ?case using i j by blast
    qed
  qed
  show "?fp ⊆ ?g"
  proof
    fix x
    assume x: "x ∈ ?fp"
    then obtain f where f: "f ∈ (Pi_E gs (λa. generate G {a}))" "x =
finprod G f gs" by blast
    have sg: "subgroup ?g G" by(intro generate_is_subgroup, fact)
    have "finprod G f gs ∈ ?g"
    proof(intro finprod_closed_subgroup[OF sg])
      have "f g ∈ generate G gs" if "g ∈ gs" for g
      proof -
        have "f g ∈ generate G {g}" using f(1) that by auto
        moreover have "generate G {g} ⊆ generate G gs" by(intro mono_generate,
use that in simp)
        ultimately show ?thesis by fast
      qed
      thus "f ∈ gs → generate G gs" by simp
    qed
    thus "x ∈ ?g" using f by blast
  qed
qed

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 -
  have "(λx. finprod G x gs) ` Pi_E gs (λa. generate G {a})
      = (λx. finprod G x gs) ` Pi gs (λa. generate G {a})"
  proof
    have "Pi_E gs (λa. generate G {a}) ⊆ Pi gs (λa. generate G {a})" by
blast
    thus "(λx. finprod G x gs) ` Pi_E gs (λa. generate G {a})
        ⊆ (λx. finprod G x gs) ` Pi gs (λa. generate G {a})" by blast
    show "(λx. finprod G x gs) ` Pi gs (λa. generate G {a})
        ⊆ (λx. finprod G x gs) ` Pi_E gs (λa. generate G {a})"
    proof
      fix x
      assume x: "x ∈ (λx. finprod G x gs) ` Pi gs (λa. generate G {a})"
      then obtain f where f: "x = finprod G f gs" "f ∈ Pi gs (λa. generate
G {a})" by blast
      moreover have "finprod G f gs = finprod G (restrict f gs) gs"
      proof(intro finprod_cong)
        have "f g ∈ carrier G" if "g ∈ gs" for g
          using that f(2) mono_generate[of "{g}" gs] generate_incl[OF
assms(2)] by fast
        thus "(f ∈ gs → carrier G) = True" by blast
      qed auto
      moreover have "restrict f gs ∈ Pi_E gs (λa. generate G {a})" us-
ing f(2) by simp
      ultimately show "x ∈ (λx. finprod G x gs) ` Pi_E gs (λa. generate
G {a})" by blast
    qed
  qed
  with generate_eq_finprod_PiE_image[OF assms] show ?thesis by auto
qed

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 -
  from generate_eq_finprod_Pi_image[OF assms]
  have "generate G gs = (λx. finprod G x gs) ` (Π a∈gs. generate G {a})"
.
  also have "... = (λx. finprod G (λg. g [^] x g) gs) ` Pi gs (λ_. (UNIV::int
set))"
  proof(rule; rule)
    fix x
    assume x: "x ∈ (λx. finprod G x gs) ` (Π a∈gs. generate G {a})"
    then obtain f where f: "f ∈ (Π a∈gs. generate G {a})" "x = finprod
G f gs" by blast
    hence "∃k::int. f a = a [^] k" if "a ∈ gs" for a using generate_pow[of
a] that assms(2) by blast
    hence "∃(h::'a ⇒ int). ∀a∈gs. f a = a [^] h a" by meson
```

```
    then obtain h where h: "∀a∈gs. f a = a [^] h a" "h ∈ gs → (UNIV
:: int set)" by auto
    have "finprod G (λg. g [^] h g) gs = finprod G f gs"
      by (intro finprod_cong, use int_pow_closed h assms(2) in auto)
    with f have "x = finprod G (λg. g [^] h g) gs" by argo
    with h(2) show "x ∈ (λx. finprod G (λg. g [^] x g) gs) ` (gs →
(UNIV::int set))" by auto
  next
    fix x
    assume x: "x ∈ (λx. finprod G (λg. g [^] x g) gs) ` (gs → (UNIV::int
set))"
    then obtain h where h: "x = finprod G (λg. g [^] h g) gs" "h ∈ gs
→ (UNIV :: int set)" by blast
    hence "∃k∈generate G {a}. a [^] h a = k" if "a ∈ gs" for a
      using generate_pow[of a] that assms(2) by blast
    then obtain f where f: "∀a∈gs. a [^] h a = f a" "f ∈ (Π a∈gs.
generate G {a})" by fast
    have "finprod G f gs = finprod G (λg. g [^] h g) gs"
    proof(intro finprod_cong)
      have "f a ∈ carrier G" if "a ∈ gs" for a
        using generate_incl[of "{a}"] assms(2) that f(2) by fast
      thus "(f ∈ gs → carrier G) = True" by blast
    qed (use f in auto)
    with h have "x = finprod G f gs" by argo
    with f(2) show "x ∈ (λx. finprod G x gs) ` (Π a∈gs. generate G {a})"
by blast
  qed
  finally show ?thesis .
qed


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
  show "?fp ⊆ ?DP"
  proof
    fix x
    assume x: "x ∈ ?fp"
    then obtain f where f: "f ∈ (Pi_E I S)" "x = finprod G f I" by blast
    have sDP: "subgroup ?DP G"
      by (intro IDirProds_is_subgroup; use subgroup.subset[OF assms(2)]
in blast)
    have "finprod G f I ∈ ?DP"
    proof(intro finprod_closed_subgroup[OF sDP])
      have "f i ∈ IDirProds G S I" if "i ∈ I" for i
      proof
        show "f i ∈ (S i)" using f(1) that by auto
```

```
        show "(S i) ⊆ IDirProds G S I" by (intro IDirProds_incl[OF that])
      qed
      thus "f ∈ I → IDirProds G S I" by simp
    qed
    thus "x ∈ ?DP" using f by blast
  qed
  show "?DP ⊆ ?fp"
  proof(unfold IDirProds_def; rule subsetI)
    fix x
    assume x: "x ∈ generate G (⋃(S ` I))"
    thus "x ∈ ?fp" using assms
    proof (induction rule: generate.induct)
      case one
      define g where g: "g = (λx. if x ∈ I then 1 else undefined)"
      then have "g ∈ Pi_E I S"
        using subgroup.one_closed[OF one(2)] by auto
      moreover have "finprod G g I = 1" by (intro finprod_one_eqI; use
g in simp)
      ultimately show ?case unfolding image_def by (auto; metis)
    next
      case i: (incl h)
      from i obtain j where j: "j ∈ I" "h ∈ (S j)" by blast
      define hf where "hf = (λx. (if x ∈ I then 1 else undefined))(j
:= h)"
      with j have "hf ∈ Pi_E I S"
        using subgroup.one_closed[OF i(3)] by force
      moreover have "finprod G hf I = h"
      proof -
        have "finprod G hf I = hf j ⊗ finprod G hf (I - {j})"
          by (intro finprod_minus, use assms hf_def subgroup.subset[OF
i(3)[OF j(1)]] j in auto)
        moreover have "hf j = h" using hf_def by simp
        moreover have "finprod G hf (I - {j}) = 1" by (rule finprod_one_eqI;
use hf_def in simp)
        ultimately show ?thesis using subgroup.subset[OF i(3)[OF j(1)]]
j(2) by auto
      qed
      ultimately show ?case unfolding image_def by (auto; metis)
    next
      case i: (inv h)
      from i obtain j where j: "j ∈ I" "h ∈ (S j)" by blast
      have ih: "inv h ∈ (S j)" using subgroup.m_inv_closed[OF i(3)[OF
j(1)] j(2)] .
      define hf where "hf = (λx. (if x ∈ I then 1 else undefined))(j
:= inv h)"
      with j ih have "hf ∈ Pi_E I S"
        using subgroup.one_closed[OF i(3)] by force
      moreover have "finprod G hf I = inv h"
      proof -
```

```
        have "finprod G hf I = hf j ⊗ finprod G hf (I - {j})"
          by (intro finprod_minus, use assms hf_def subgroup.subset[OF
i(3)[OF j(1)]] j in auto)
        moreover have "hf j = inv h" using hf_def by simp
        moreover have "finprod G hf (I - {j}) = 1" by (rule finprod_one_eqI;
use hf_def in simp)
        ultimately show ?thesis using subgroup.subset[OF i(3)[OF j(1)]]
j(2) by auto
      qed
      ultimately show ?case unfolding image_def by (auto; metis)
    next
      case e: (eng a b)
      from e obtain f where f: "f ∈ Pi_E I S" "a = finprod G f I" by
blast
      from e obtain g where g: "g ∈ Pi_E I S" "b = finprod G g I" by
blast
      from f g have "a ⊗ b = finprod G f I ⊗ finprod G g I" by blast
      also have "... = finprod G (λa. f a ⊗ g a) I"
      proof(intro finprod_multf[symmetric])
        have "⋃(S ` I) ⊆ carrier G" using subgroup.subset[OF e(6)] by
blast
        thus "f ∈ I → carrier G" "g ∈ I → carrier G"
          using f(1) g(1) unfolding PiE_def Pi_def by auto
      qed
      also have "... = finprod G (restrict (λa. f a ⊗ g a) I) I"
      proof(intro finprod_cong)
        show "I = I" by simp
        show "⋀i. i ∈ I =simp=> f i ⊗ g i = (λa∈I. f a ⊗ g a) i" by
simp
        have fp: "f i ∈ (S i)" if "i ∈ I" for i using f that by auto
        have gp: "g i ∈ (S i)" if "i ∈ I" for i using g that by auto
        have "f i ⊗ g i ∈ (S i)" if "i ∈ I" for i
          using subgroup.m_closed[OF e(6)[OF that] fp[OF that] gp[OF that]]
.
        thus "((λa. f a ⊗ g a) ∈ I → carrier G) = True" using subgroup.subset[OF
e(6)] by auto
      qed
      finally have "a ⊗ b = finprod G (restrict (λa. f a ⊗ g a) I) I"
.
      moreover have "(restrict (λa. f a ⊗ g a) I) ∈ Pi_E I S"
      proof -
        have fp: "f i ∈ (S i)" if "i ∈ I" for i using f that by auto
        have gp: "g i ∈ (S i)" if "i ∈ I" for i using g that by auto
        have "f i ⊗ g i ∈ (S i)" if "i ∈ I" for i
          using subgroup.m_closed[OF e(6)[OF that] fp[OF that] gp[OF that]]
.
        thus ?thesis by auto
      qed
      ultimately show ?case using f g by blast
```

```
    qed
  qed
qed

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 -
  have "(λx. finprod G x I) ` (Pi I S) = (λx. finprod G x I) ` (Pi_E I
S)"
  proof
    have "Pi_E I S ⊆ Pi I S" by blast
    thus "(λx. finprod G x I) ` Pi_E I S ⊆ (λx. finprod G x I) ` Pi I
S" by blast
    show "(λx. finprod G x I) ` Pi I S ⊆ (λx. finprod G x I) ` Pi_E I
S"
    proof
      fix x
      assume x: "x ∈ (λx. finprod G x I) ` Pi I S"
      then obtain f where f: "x = finprod G f I" "f ∈ Pi I S" by blast
      moreover have "finprod G f I = finprod G (restrict f I) I"
        by (intro finprod_cong; use f(2) subgroup.subset[OF assms(2)]
in fastforce)
      moreover have "restrict f I ∈ Pi_E I S" using f(2) by simp
      ultimately show "x ∈ (λx. finprod G x I) ` Pi_E I S" by blast
    qed
  qed
  with IDirProds_eq_finprod_PiE[OF assms] show ?thesis by auto
qed
```

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(intro generate_one_switched_eqI[OF assms(1, 2, 3)]; cases "finite
A")
  case True
  hence fB: "finite B" using assms(3) by blast
  have cB: "B ⊆ carrier G"
  proof -
    have "b ∈ carrier G"
      by (subst assms(7), intro finprod_closed, use assms(1, 4) int_pow_closed
in fast)
```

```
    thus ?thesis using assms(1, 3) by blast
  qed
  show "a ∈ generate G B"
  proof(subst generate_eq_finprod_Pi_image[OF fB cB], rule)
    show "a = finprod G (λx. x [^] g x) B" by fact
    have "x [^] g x ∈ generate G {x}" if "x ∈ B" for x using generate_pow[of
x] cB that by blast
    thus "(λx. x [^] g x) ∈ (Π a∈B. generate G {a})" unfolding Pi_def
by blast
  qed
  show "b ∈ generate G A"
  proof(subst generate_eq_finprod_Pi_image[OF True assms(1)], rule)
    show "b = finprod G (λx. x [^] f x) A" by fact
    have "x [^] f x ∈ generate G {x}" if "x ∈ A" for x
      using generate_pow[of x] assms(1) that by blast
    thus "(λx. x [^] f x) ∈ (Π a∈A. generate G {a})" unfolding Pi_def
by blast
  qed
next
  case False
  hence b: "b = 1" using assms(7) unfolding finprod_def by simp
  from False assms(3) have "infinite B" by simp
  hence a: "a = 1" using assms(6) unfolding finprod_def by simp
  show "a ∈ generate G B" using generate.one a by blast
  show "b ∈ generate G A" using generate.one b by blast
qed
```

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 (rule ccontr; clarify)
  from assms(5) have f: "f i ∈ (S i)" if "i ∈ I" for i using that by
fastforce
  fix i
  assume i: "i ∈ I"
  have si: "subgroup (S i) G" using assms(3)[OF i] .
  consider (triv) "(S i) = {1}" | (not_triv) "(S i) ≠ {1}" by blast
  thus "f i = 1"
  proof (cases)
    case triv
    then show ?thesis using f[OF i] by blast
  next
    case not_triv
    show ?thesis
    proof (rule ccontr)
```

```
      have fc: "f i ∈ carrier G" using f[OF i] subgroup.subset[OF si]
by blast
      assume no: "f i ≠ 1"
      have fH: "f i ∈ (S i)" using f[OF i] .
      from subgroup.m_inv_closed[OF si this] have ifi: "inv (f i) ∈ (S
i)" .
      moreover have "inv (f i) ≠ 1" using no fc by simp
      moreover have "inv (f i) = finprod G f (I - {i})"
      proof -
        have "1 = finprod G f I" using assms(4) by simp
        also have "... = finprod G f (insert i (I - {i}))"
        proof -
          have "I = insert i (I - {i})" using i by fast
          thus ?thesis by simp
        qed
        also have "... = f i ⊗ finprod G f (I - {i})"
        proof(intro finprod_insert)
          show "finite (I - {i})" using assms(2) by blast
          show "i ∉ I - {i}" by blast
          show "f ∈ I - {i} → carrier G" using assms(3) f subgroup.subset
by blast
          show "f i ∈ carrier G" by fact
        qed
        finally have o: "1 = f i ⊗ finprod G f (I - {i})" .
        show ?thesis
        proof(intro inv_equality)
          show "f i ∈ carrier G" by fact
          show "finprod G f (I - {i}) ∈ carrier G"
            by (intro finprod_closed; use assms(3) f subgroup.subset in
blast)
          from m_comm[OF this fc] o show "finprod G f (I - {i}) ⊗ f
i = 1" by simp
        qed
      qed
      moreover have "finprod G f (I - {i}) ∈ IDirProds G S (I - {i})"
      proof (intro finprod_closed_subgroup IDirProds_is_subgroup)
        show "⋃ (S ` (I - {i})) ⊆ carrier G" using assms(3) subgroup.subset
by auto
        have "f j ∈ (IDirProds G S (I - {i}))" if "j ∈ (I - {i})" for
j
          using IDirProds_incl[OF that] f that by blast
        thus "f ∈ I - {i} → IDirProds G S (I - {i})" by blast
      qed
      ultimately have "¬complementary (S i) (IDirProds G S (I - {i}))"
        unfolding complementary_def by auto
      thus False using assms(1) i unfolding compl_fam_def by blast
    qed
  qed
qed
```

```
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 (unfold compl_fam_def; rule)
  fix k
  assume k: "k ∈ I"
  let ?DP = "IDirProds G S (I - {k})"
  show "complementary (S k) ?DP"
  proof (rule ccontr; unfold complementary_def)
    have sk: "subgroup (S k) G" using assms(2)[OF k] .
    have sDP: "subgroup ?DP G"
      by (intro IDirProds_is_subgroup; use subgroup.subset[OF assms(2)]
in blast)
    assume a: "(S k) ∩ IDirProds G S (I - {k}) ≠ {1}"
    then obtain x where x: "x ∈ (S k)" "x ∈ IDirProds G S (I - {k})"
"x ≠ 1"
      using subgroup.one_closed sk sDP by blast
    then have "x ∈ (λx. finprod G x (I - {k})) ` (Pi (I - {k}) S)"
      using IDirProds_eq_finprod_Pi[of "(I - {k})"] assms(1, 2) by blast
    then obtain ht where ht: "finprod G ht (I - {k}) = x" "ht ∈ Pi (I
- {k}) S" by blast
    define h where h: "h = (ht(k := inv x))"
    then have hPi: "h ∈ Pi I S" using ht subgroup.m_inv_closed[OF assms(2)[OF
k] x(1)] by auto
    have "finprod G h (I - {k}) = x"
    proof (subst ht(1)[symmetric], intro finprod_cong)
      show "I - {k} = I - {k}" by simp
      show "(h ∈ I - {k} → carrier G) = True" using h ht(2) subgroup.subset[OF
assms(2)]
        unfolding Pi_def id_def by auto
      show "⋀i. i ∈ I - {k} =simp=> h i = ht i" using ht(2) h by simp
    qed
    moreover have "finprod G h I = h k ⊗ finprod G h (I - {k})"
      by (intro finprod_minus; use k assms hPi subgroup.subset[OF assms(2)]
Pi_def in blast)
    ultimately have "finprod G h I = inv x ⊗ x" using h by simp
    then have "finprod G h I = 1" using subgroup.subset[OF sk] x(1) by
auto
    moreover have "h k ≠ 1" using h x(3) subgroup.subset[OF sk] x(1)
by force
    ultimately show False using assms(3) k hPi by blast
  qed
qed

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))"
  using compl_fam_imp_triv_finprod triv_finprod_imp_compl_fam assms by
blast

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
  show "compl_fam S I ⟹ ∀f∈Pi_E I S. finprod G f I = 1 ⟶ (∀i∈I.
f i = 1)"
    using triv_finprod_iff_compl_fam_Pi[OF assms] by auto
  have "∀f∈Pi_E I S. finprod G f I = 1 ⟶ (∀i∈I. f i = 1)
    ⟹ ∀f∈Pi I S. finprod G f I = 1 ⟶ (∀i∈I. f i = 1)"
  proof(rule+)
    fix f i
    assume f: "f ∈ Pi I S" "finprod G f I = 1" and i: "i ∈ I"
    assume allf: "∀f∈Pi_E I S. finprod G f I = 1 ⟶ (∀i∈I. f i = 1)"
    have "f i = restrict f I i" using i by simp
    moreover have "finprod G (restrict f I) I = finprod G f I"
      using f subgroup.subset[OF assms(2)] unfolding Pi_def by (intro
finprod_cong; auto)
    moreover have "restrict f I ∈ Pi_E I S" using f by simp
    ultimately show "f i = 1" using allf f i by metis
  qed
  thus "∀f∈Pi_E I S. finprod G f I = 1 ⟶ (∀i∈I. f i = 1) ⟹ compl_fam
S I"
    using triv_finprod_iff_compl_fam_Pi[OF assms] by presburger
qed
```

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)"
  using assms
proof (induction A rule: finite_induct)
  case (insert x A)
  have "(⨂z∈Sigma (insert x A) B. case z of (x, y) ⇒ g x y) =
          (⨂z∈Pair x ‘ B x. case z of (x, y) ⇒ g x y) ⊗ (⨂z∈Sigma
A B. case z of (x, y) ⇒ g x y)"
    unfolding Sigma_insert using insert.prems insert.hyps
    by (subst finprod_Un_disjoint) auto
  also have "(⨂z∈Sigma A B. case z of (x, y) ⇒ g x y) = (⨂x∈A. ⨂y∈B
x. g x y)"
    using insert.prems insert.hyps by (subst insert.IH [symmetric]) auto
  also have "(⨂z∈Pair x ‘ B x. case z of (x, y) ⇒ g x y) = (⨂y∈B
x. g x y)"
```

```
    using insert.prems insert.hyps by (subst finprod_reindex) (auto intro:
inj_onI)
  finally show ?case
    using insert.hyps insert.prems by simp
qed auto
```

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(cases "g ∈ gs")
  case True
  hence "gs = (gs ∪ {g})" by blast
  thus ?thesis using assms(4) by auto
next
  case gngs: False
  show ?thesis
  proof (intro is_idirprod_subgroup_suffices)
    have gsgc: "gs ∪ {g} ⊆ carrier G" using assms(2, 3) by blast
    thus "generate G (gs ∪ {g}) = IDirProds G (λg. generate G {g}) (gs
∪ {g})"
      unfolding IDirProds_def using generate_idem_Un by presburger
    show "∀i∈gs ∪ {g}. subgroup (generate G {i}) G" using generate_is_subgroup
gsgc by auto
    have sg: "subgroup (generate G {g}) G" by (intro generate_is_subgroup,
use assms(3) in blast)
    from assms(4) is_idirprod_def have ih: "∀x. x ∈ gs ⟶ generate
G {x} ◁ G"
                                              "compl_fam (λg. generate G
{g}) gs"
      by fastforce+
    hence ca: "complementary (generate G {a}) (generate G (gs - {a}))"
if "a ∈ gs" for a
      unfolding compl_fam_def IDirProds_def
      using gsgc generate_idem_Un[of "gs - {a}"] that by fastforce
    have aux: "gs ∪ {g} - {i} ⊆ carrier G" for i using gsgc by blast
    show "compl_fam (λg. generate G {g}) (gs ∪ {g})"
    proof(unfold compl_fam_def IDirProds_def, subst generate_idem_Un[OF
aux],
          rule, rule ccontr)
      fix h
      assume h: "h ∈ gs ∪ {g}"
      assume c: "¬ complementary (generate G {h}) (generate G (gs ∪
{g} - {h}))"
      show "False"
```

```
      proof (cases "h = g")
        case True
        with c have "¬ complementary (generate G {g}) (generate G (gs
- {g}))" by auto
        moreover have "complementary (generate G {g}) (generate G (gs
- {g}))"
          by (rule subgroup_subset_complementary[OF generate_is_subgroup
generate_is_subgroup[of gs]
                  generate_is_subgroup mono_generate], use assms(2, 3,
5) in auto)
        ultimately show False by blast
      next
        case hng: False
        hence h: "h ∈ gs" "h ≠ g" using h by blast+
        hence "gs ∪ {g} - {h} = gs - {h} ∪ {g}" by blast
        with c have c: "¬ complementary (generate G {h}) (generate G
(gs - {h} ∪ {g}))" by argo
        then obtain k where k: "k ∈ generate G {h}" "k ∈ generate G
(gs - {h} ∪ {g})" "k ≠ 1"
          unfolding complementary_def using generate.one by blast
        with ca have kngh: "k ∉ generate G (gs - {h})" using h unfold-
ing complementary_def by blast
        from k(2) generate_eq_finprod_PiE_image[of "gs - {h} ∪ {g}"]
assms(1) gsgc
        obtain f where f:
          "k = finprod G f (gs - {h} ∪ {g})" "f ∈ (Π_E a∈gs - {h} ∪ {g}.
generate G {a})"
          by blast
        have fg: "f a ∈ generate G {a}" if "a ∈ (gs - {h} ∪ {g})" for
a using that f(2) by blast
        have fc: "f a ∈ carrier G" if "a ∈ (gs - {h} ∪ {g})" for a
        proof -
          have "generate G {a} ⊆ carrier G" if "a ∈ (gs - {h} ∪ {g})"
for a
            using that generate_incl[of "{a}"] gsgc by blast
          thus "f a ∈ carrier G" using that fg by auto
        qed
        have kp: "k = f g ⊗ finprod G f (gs - {h})"
        proof -
          have "(gs - {h} ∪ {g}) = insert g (gs - {h})" by fast
          moreover have "finprod G f (insert g (gs - {h})) = f g ⊗ finprod
G f (gs - {h})"
            by (intro finprod_insert, use fc assms(1) gngs in auto)
          ultimately show ?thesis using f(1) by argo
        qed
        have fgsh: "finprod G f (gs - {h}) ∈ generate G (gs - {h})"
        proof(intro finprod_closed_subgroup[OF generate_is_subgroup])
          show "gs - {h} ⊆ carrier G" using gsgc by blast
          have "f a ∈ generate G (gs - {h})" if "a ∈ (gs - {h})" for
```

```
a
          using mono_generate[of "{a}" "gs - {h}"] fg that by blast
        thus "f ∈ gs - {h} → generate G (gs - {h})" by blast
      qed
      have "f g ⊗ finprod G f (gs - {h}) ∉ generate G gs"
      proof
        assume fpgs: "f g ⊗ finprod G f (gs - {h}) ∈ generate G gs"
        from fgsh have fgsgs: "finprod G f (gs - {h}) ∈ generate G
gs"
          using mono_generate[of "gs - {h}" gs] by blast
        have fPi: "f ∈ (Π a∈(gs - {h}). generate G {a})" using f by
blast
        have gI: "generate G (gs - {h})
              = (λx. finprod G x (gs - {h})) ‘ (Π a∈gs - {h}. generate
G {a})"
          using generate_eq_finprod_Pi_image[of "gs - {h}"] assms(1,
2) by blast
        have fgno: "f g ≠ 1"
        proof (rule ccontr)
          assume o: "¬ f g ≠ 1"
          hence kf: "k = finprod G f (gs - {h})" using kp finprod_closed
fc by auto
          hence "k ∈ generate G (gs - {h})" using fPi gI by blast
          thus False using k ca h unfolding complementary_def by blast
        qed
        from fpgs have "f g ∈ generate G gs"
          using subgroup.mult_in_cancel_right[OF generate_is_subgroup[OF
assms(2)] fc[of g] fgsgs]
          by blast
        with fgno assms(5) fg[of g] show "False" unfolding complementary_def
by blast
      qed
      moreover have "k ∈ generate G gs" using k(1) mono_generate[of
"{h}" gs] h(1) by blast
      ultimately show False using kp by blast
    qed
  qed
 qed
qed

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)"
  using assms group_hom.generate_img group_hom_axioms by blast
```

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 -
  have c: "inv y ⊗_G x ∈ carrier G" using assms by simp
  then have e: "x = y ⊗_G (inv y ⊗_G x)" using assms G.m_assoc
    using G.inv_solve_left by blast
  then have "h x = h (y ⊗_G (inv y ⊗_G x))" by simp
  then have "h x = h y ⊗_H h (inv y ⊗_G x)" using c assms by simp
  then have "1_H = h (inv y ⊗_G x)" using assms by simp
  then have "(inv y ⊗_G x) ∈ kernel G H h" unfolding kernel_def using
c by simp
  thus ?thesis using e that by blast
qed
```

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(intro equalityI subsetI)
  fix x
  assume assm: "x ∈ A <#> kernel G H h"
  then have xc: "x ∈ carrier G" unfolding kernel_def set_mult_def us-
ing assms by blast
  from assm obtain a b where ab: "a ∈ A" "b ∈ kernel G H h" "x = a
⊗_G b"
    unfolding set_mult_def by blast
  then have abc: "a ∈ carrier G" "b ∈ carrier G" unfolding kernel_def
using assms by auto
  from ab have "h x = h (a ⊗_G b)" by blast
  also have "... = h a ⊗_H h b" using abc by simp
  also have "... = h a ⊗_H 1_H" using ab(2) unfolding kernel_def by simp
  also have "... = h a" using abc by simp
  also have "... ∈ h ` A" using ab by blast
  finally have "h x ∈ h ` A" .
  thus "x ∈ {x ∈ carrier G. h x ∈ h ` A}" using xc by blast
next
  fix x
```

```
  assume "x ∈ {x ∈ carrier G. h x ∈ h ` A}"
  then have x: "x ∈ carrier G" "h x ∈ h ` A" by simp+
  then obtain y where y: "y ∈ A" "h x = h y" "y ∈ carrier G" using assms
by auto
  with kernel_assoc_elem obtain z where "x = y ⊗G z" "z ∈ kernel G
H h" using x by metis
  thus "x ∈ A <#> kernel G H h" unfolding set_mult_def using y by blast
qed

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)"
  using vimage_eq_set_mult_kern_right[of A] G.set_mult_subset_generate[of
"A" "kernel G H h"] assms
  unfolding kernel_def by blast
```

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
  show "?J ⊆ carrier G" by blast
  show "1 ∈ ?J" using subgroup.one_closed[of I H] assms by simp
  fix x
  assume x: "x ∈ ?J"
  then have hx: "h x ∈ I" by blast
  show "inv x ∈ ?J"
  proof -
    from hx have "invH (h x) ∈ I" using subgroup.m_inv_closed assms
by fast
    moreover have "inv x ∈ carrier G" using x by simp
    moreover have "invH (h x) = h (inv x)" using x by auto
    ultimately show "inv x ∈ ?J" by simp
  qed
  fix y
  assume y: "y ∈ ?J"
  then have hy: "h y ∈ I" by blast
  show "x ⊗ y ∈ {x ∈ carrier G. h x ∈ I}"
  proof -
    have "h (x ⊗ y) = h x ⊗H h y" using x y by simp
    also have "... ∈ I" using hx hy assms subgroup.m_closed by fast
    finally have "h (x ⊗ y) ∈ I" .
    moreover have "x ⊗ y ∈ carrier G" using x y by simp
    ultimately show ?thesis by blast
  qed
qed

lemma (in group_hom) iso_kernel:
  assumes "h ∈ iso G H"
```

```
  shows "kernel G H h = {1G}"
  unfolding kernel_def using assms
  using hom_one iso_iff by blast


lemma (in group_hom) induced_group_hom_same_group:
  assumes "subgroup I G"
  shows "group_hom (G ⦇ carrier := I ⦈) H h"
proof -
  have "h ∈ hom (G ⦇ carrier := I ⦈) H"
    using homh subgroup.mem_carrier[OF assms] unfolding hom_def by auto
  thus ?thesis
    unfolding group_hom_def group_hom_axioms_def
    using subgroup.subgroup_is_group[OF assms G.is_group] by simp
qed
```

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 -
  have "h a [^]H (G.ord a) = h (a [^]G G.ord a)"
    using assms local.hom_nat_pow by presburger
  also have "... = h (1G)" using G.pow_ord_eq_1 assms by simp
  also have "... = 1H" by simp
  finally have "h a [^]H G.ord a = 1H" .
  then show ?thesis using pow_eq_id assms by simp
qed
```

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"
  using hom_ord_dvd_ord assms by fastforce
```

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 (rule antisym)
  show "H.ord (h a) ≤ G.ord a" using hom_ord_dvd_ord assms by force
  show "G.ord a ≤ H.ord (h a)"
  proof -
    have "1H = h (a [^]G H.ord(h a))" using H.pow_ord_eq_1 assms
      by (simp add: local.hom_nat_pow)
    then have "a [^]G H.ord (h a) = 1G" using assms inj_on_one_iff by
simp
```

```
    then have "G.ord a dvd H.ord (h a)" using G.pow_eq_id assms(1) by
blast
    thus ?thesis using assms finite_ord_stays_finite by fastforce
  qed
qed

lemma (in group_hom) one_in_kernel:
  "1 ∈ kernel G H h"
  using subgroup.one_closed subgroup_kernel by blast

lemma hom_in_carr:
  assumes "f ∈ hom G H"
  shows "⋀x. x ∈ carrier G ⟹ f x ∈ carrier H"
  using assms unfolding hom_def bij_betw_def by blast

lemma iso_in_carr:
  assumes "f ∈ iso G H"
  shows "⋀x. x ∈ carrier G ⟹ f x ∈ carrier H"
  using assms unfolding iso_def bij_betw_def by blast

lemma triv_iso:
  assumes "group G" "group H" "carrier G = {1_G}" "carrier H = {1_H}"
  shows "G ≅ H"
proof(unfold is_iso_def iso_def)
  interpret G: group G by fact
  interpret H: group H by fact
  let ?f = "λ_. 1_H"
  have "?f ∈ hom G H" by (intro homI, auto)
  moreover have "bij_betw ?f (carrier G) (carrier H)" unfolding bij_betw_def
    using assms(3, 4) by auto
  ultimately show "{h ∈ hom G H. bij_betw h (carrier G) (carrier H)}
≠ {}" by blast
qed
```

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 -
  interpret G: group G by simp
  interpret H: group H by simp
  interpret ih: subgroup "h ` (carrier G)" H using img_is_subgroup by
blast
  interpret ih: group "H⦇carrier := h ` (carrier G)⦈" using subgroup.subgroup_is_group
by blast
  interpret h: group_hom G "H⦇carrier := h ` (carrier G)⦈"
    by (unfold_locales, unfold hom_def, auto)
  interpret k: subgroup "kernel G (H⦇carrier := h ` carrier G⦈) h" G us-
```

```
ing h.subgroup_kernel by blast
  from h.FactGroup_iso
  have "G Mod kernel G (H(|carrier := h ` carrier G|)) h ≅ H(|carrier :=
h ` carrier G|)" by auto
  hence "card (h ` (carrier G)) = order (G Mod kernel G (H(|carrier :=
h ` carrier G|)) h)"
    using iso_same_card unfolding order_def by fastforce
  moreover have "order (G Mod kernel G (H(|carrier := h ` carrier G|))
h)
                 * card (kernel G (H(|carrier := h ` carrier G|)) h) = order
G"
    using G.lagrange[OF k.subgroup_axioms] unfolding order_def FactGroup_def
by force
  moreover have "kernel G (H(|carrier := h ` carrier G|)) h = kernel G
H h"
    unfolding kernel_def by auto
  ultimately show ?thesis by argo
qed

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"
  using ord_ge_1[of x] assms by auto

lemma (in finite_group) order_gt_0 [simp,intro]: "order G > 0"
  by (subst order_gt_0_iff_finite) auto

lemma (in finite_group) finite_ord_conv_Least:
  assumes "x ∈ carrier G"
  shows "ord x = (LEAST n::nat. 0 < n ∧ x [^] n = 1)"
  using pow_order_eq_1 order_gt_0_iff_finite ord_conv_Least assms by auto
```

```
lemma (in finite_group) non_trivial_group_ord_gr_1:
  assumes "carrier G ≠ {1}"
  shows "∃e ∈ carrier G. ord e > 1"
proof -
  from one_closed obtain e where e: "e ≠ 1" "e ∈ carrier G" using assms
carrier_not_empty by blast
  thus ?thesis using ord_eq_1[of e] le_neq_implies_less ord_ge_1 by fastforce
qed


lemma (in finite_group) max_order_elem:
  obtains a where "a ∈ carrier G" "∀x ∈ carrier G. ord x ≤ ord a"
proof -
  have "∃x. x ∈ carrier G ∧ (∀y. y ∈ carrier G ⟶ ord y ≤ ord x)"
  proof (rule ex_has_greatest_nat[of _ 1 _ "order G + 1"], safe)
    show "1 ∈ carrier G"
      by auto
  next
    fix x assume "x ∈ carrier G"
    hence "ord x ≤ order G"
      by (intro ord_le_group_order fin)
    also have "... < order G + 1"
      by simp
    finally show "ord x < order G + 1" .
  qed
  thus ?thesis using that by blast
qed

lemma (in finite_group) iso_imp_finite:
  assumes "G ≅ H" "group H"
  shows "finite_group H"
proof -
  interpret H: group H by fact
  show ?thesis
  proof(unfold_locales)
    show "finite (carrier H)" using iso_same_card[OF assms(1)]
      by (metis card_gt_0_iff order_def order_gt_0)
  qed
qed

lemma (in finite_group) finite_FactGroup:
  assumes "H ◁ G"
  shows "finite_group (G Mod H)"
proof -
  interpret H: normal H G by fact
  interpret Mod: group "G Mod H" using H.factorgroup_is_group .
  show ?thesis
    by (unfold_locales, unfold FactGroup_def RCOSETS_def, simp)
qed
```

```
lemma (in finite_group) bigger_subgroup_is_group:
  assumes "subgroup H G" "card H ≥ order G"
  shows "H = carrier G"
  using subgroup.subset fin assms by (metis card_seteq order_def)
```

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)"
  using generate_incl[of A] rev_finite_subset[of "carrier G" "generate
G A"] assms by simp
```

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 -
  define A where A: "A = carrier G"
  hence gA: "generate G A = carrier G"
    using generate_incl[of "carrier G"] generate_sincl[of "carrier G"]
by simp
  hence "finite A" using fin A by argo
  moreover have "A0 ⊆ A" using assms(1) A by argo
  moreover have "A ⊆ carrier G" using A by simp
  moreover have "generate G A0 ⊆ generate G A" using gA generate_incl[OF
assms(1)] by argo
  ultimately have "P (G(|carrier := generate G A|))" using assms(2, 3)
  proof (induction "A" taking: card rule: measure_induct_rule)
    case (less A)
    then show ?case
    proof(cases "generate G A0 = generate G A")
      case True
      thus ?thesis using less by force
    next
      case gA0: False
      with less(3) have s: "A0 ⊂ A" by blast
      then obtain a where a: "a ∈ A - A0" by blast
      have P1: "P (G(|carrier := generate G (A - {a})|))"
      proof(rule less(1))
```

```
        show "card (A - {a}) < card A" using a less(2) by (meson DiffD1
card_Diff1_less)
        show "A0 ⊆ A - {a}" using a s by blast
        thus "generate G A0 ⊆ generate G (A - {a})" using mono_generate
by presburger
      qed (use less a s in auto)
      show ?thesis
      proof (cases "generate G A = generate G (A - {a})")
        case True
        then show ?thesis using P1 by simp
      next
        case False
        have "a ∈ carrier G - generate G (A - {a})"
        proof -
          have "a ∉ generate G (A - {a})"
          proof
            assume a2: "a ∈ generate G (A - {a})"
            have "generate G (A - {a}) = generate G A"
            proof (rule equalityI)
              show "generate G (A - {a}) ⊆ generate G A" using mono_generate
by auto
              show "generate G A ⊆ generate G (A - {a})"
              proof(subst (2) generate_idem[symmetric])
                show "generate G A ⊆ generate G (generate G (A - {a}))"
                  by (intro mono_generate, use generate_sincl[of "A -
{a}"] a2 in blast)
              qed (use less in auto)
            qed
            with False show False by argo
          qed
          with a less show ?thesis by fast
        qed
        from less(7)[OF _ this _ P1] less(4) s a have "P (G⦇carrier :=
generate G (A - {a} ∪ {a})⦈)"
          by blast
        moreover have "A - {a} ∪ {a} = A" using a by blast
        ultimately show ?thesis by auto
      qed
    qed
  qed
  with gA show ?thesis by simp
qed
```

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 -
  interpret H: group H by fact
  interpret H: comm_group H by (intro iso_imp_comm_group[OF assms(1)],
unfold_locales)
  interpret H: finite_group H by (intro iso_imp_finite[OF assms(1)], unfold_locales)
  show ?thesis by unfold_locales
qed

lemma (in finite_comm_group) finite_comm_FactGroup:
  assumes "subgroup H G"
  shows "finite_comm_group (G Mod H)"
  unfolding finite_comm_group_def
proof(safe)
  show "finite_group (G Mod H)" using finite_FactGroup[OF subgroup_imp_normal[OF
assms]] .
  show "comm_group (G Mod H)" by (simp add: abelian_FactGroup assms)
qed


lemma (in finite_comm_group) subgroup_imp_finite_comm_group:
  assumes "subgroup H G"
  shows   "finite_comm_group (G⦇carrier := H⦈)"
proof -
  interpret G': group "G⦇carrier := H⦈" by (intro subgroup_imp_group)
fact+
  interpret H: subgroup H G by fact
  show ?thesis by standard (use finite_subset[OF H.subset] in ‹auto simp:
m_comm›)
qed
```

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 -
  from generator have x_g:"x ∈ generate G {gen}" using assms by fast
  with generate_pow[of gen] show ?thesis using gen_closed by blast
qed
```

Every cyclic group is commutative/abelian.

```
sublocale cyclic_group ⊆ comm_group
proof(unfold_locales)
  fix x y
  assume "x ∈ carrier G" "y ∈ carrier G"
  then obtain a b where ab:"x = gen [^] (a::int)" "y = gen [^] (b::int)"
    using elem_is_gen_pow by presburger
  then have "x ⊗ y = gen [^] (a + b)" by (simp add: int_pow_mult)
  also have "... = y ⊗ x" using ab int_pow_mult
    by (metis add.commute gen_closed)
  finally show "x ⊗ y = y ⊗ x" .
qed
```

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"
  using assms by (unfold_locales; auto)

lemma (in group) cyclic_groupI1:
  assumes "a ∈ carrier G" "carrier G ⊆ generate G {a}"
  shows "cyclic_group G a"
  using assms by (unfold_locales, use generate_incl[of "{a}"] in auto)

lemma (in group) cyclic_groupI2:
  assumes "a ∈ carrier G"
  shows "cyclic_group (G(|carrier := generate G {a}|)) a"
proof (intro group.cyclic_groupI0)
  show "group (G(|carrier := generate G {a}|))"
    by (intro subgroup.subgroup_is_group group.generate_is_subgroup, use
assms in simp_all)
  show "a ∈ carrier (G(|carrier := generate G {a}|))" using generate.incl[of
a "{a}"] by auto
  show "carrier (G(|carrier := generate G {a}|)) = generate (G(|carrier
:= generate G {a}|)) {a}"
    using assms
    by (simp add: generate_consistent generate.incl group.generate_is_subgroup)
qed
```

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 (cases "finite (carrier G)")
  case True
  with generator show "ord gen = order G"
    using generate_pow_card[of gen] order_def[of G] gen_closed by simp
next
  case False
  thus ?thesis
```

```
    using generate_pow_card generator order_def[of G] card_eq_0_iff[of
"carrier G"] by force
qed
```

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 (unfold_locales)
  show "a ∈ carrier G" using assms(1) by simp
  show "carrier G = generate G {a}"
    using assms bigger_subgroup_is_group[OF generate_is_subgroup]
    by (metis empty_subsetI fin generate_pow_card insert_subset ord_le_group_order)
qed
```

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 (unfold_locales)
  obtain p where order_p: "order G = p" and p_prime: "Factorial_Ring.prime
p" using assms by blast
  then have "card (carrier G) ≥ 2" by (simp add: order_def prime_ge_2_nat)
  then obtain a where a_in: "a ∈ carrier G" and a_not_one: "a ≠ 1"
using one_unique
    by (metis (no_types, lifting) card_2_iff' obtain_subset_with_card_n
subset_iff)
  interpret fin: finite_group G
    using assms order_gt_0_iff_finite unfolding order_def by unfold_locales
auto
  have "ord a dvd p" using a_in order_p ord_dvd_group_order by blast
  hence "ord a = p" using prime_nat_iff[of p] p_prime ord_eq_1 a_in a_not_one
by blast
  then interpret cyclic_group G a
    using fin.element_ord_generates_cyclic order_p a_in by simp
  show ?thesis using that cyclic_group_axioms .
qed
```

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 -
  from x obtain n :: int where n: "x = gen [^] n"
    using elem_is_gen_pow[of x] by auto
  show ?thesis
  proof (cases "n ≥ 0")
    case True
    have "P (gen [^] nat n)"
      by (rule IH1)
    with True n show ?thesis by simp
  next
    case False
    have "P (inv (gen [^] nat (-n)))"
      by (intro IH1 IH2) auto
    also have "gen [^] nat (-n) = gen [^] (-n)"
      using False by simp
    also have "inv ... = x"
      using n by (simp add: int_pow_neg)
    finally show ?thesis .
  qed
qed
```

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
  by unfold_locales

lemma (in finite_cyclic_group) ord_gen_gt_zero:
  "ord gen > 0"
  using ord_ge_1[OF fin gen_closed] by simp
```

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 -
  from ord_gen_gt_zero generate_nat_pow[OF _ gen_closed] obtain n::nat
where n: "x = gen [^] n"
    using generator x by blast
```

```
  thus ?thesis by (induction n arbitrary: x, use assms in auto)
qed

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(rule generator_induct0[OF x])
  show "⋀x. ⟦x ∈ carrier G; P x⟧ ⟹ P (x ⊗ gen)" using IH2 by blast
  have "P x" if "n > 0" "x = gen [^] n" for n::nat and x using that
    by (induction n arbitrary: x; use assms in fastforce)
  from this[OF ord_pos[OF gen_closed] pow_ord_eq_1[OF gen_closed, symmetric]]
show "P 1" .
qed
```

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 -
  have "a = a [^] (1::int)" using assms by auto
  moreover have "a [^] (1::int) = a [^] (SOME x::int. a [^] (1::int)
= a [^] x)"
    by (intro someI_ex[of "λx::int. a [^] (1::int) = a [^] x"]; blast)
  ultimately show ?thesis unfolding get_exp_def by simp
qed

lemma (in group) get_exp_self:
  assumes "a ∈ carrier G"
  shows "get_exp a a mod ord a = (1::int) mod ord a"
  by (intro pow_eq_int_mod[OF assms], use get_exp_self_fulfills[OF assms]
assms in auto)
```

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 -
  from elem_is_gen_pow[OF assms] obtain k::int where k: "a = gen [^]
k" by blast
  moreover have "gen [^] k = gen [^] (SOME x::int. gen [^] k = gen [^]
x)"
    by(intro someI_ex[of "λx::int. gen [^] k = gen [^] x"]; blast)
  ultimately show ?thesis unfolding get_exp_def by blast
qed

lemma (in cyclic_group) get_exp_non_zero:
  assumes"b ∈ carrier G" "b ≠ 1"
  shows "get_exp gen b ≠ 0"
  using assms get_exp_fulfills[OF assms(1)] by auto
```

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 (intro pow_eq_int_mod[OF gen_closed])
  from get_exp_fulfills[of "a ⊗ b"] have "gen [^] get_exp gen (a ⊗ b)
= a ⊗ b" using assms by simp
  moreover have "gen [^] (get_exp gen a + get_exp gen b) = a ⊗ b"
  proof -
    have "gen [^] (get_exp gen a + get_exp gen b) = gen [^] (get_exp gen
a) ⊗ gen [^] (get_exp gen b)"
      using int_pow_mult by blast
    with get_exp_fulfills assms show ?thesis by simp
  qed
  ultimately show "gen [^] get_exp gen (a ⊗ b) = gen [^] (get_exp gen
a + get_exp gen b)" by simp
qed
```

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
  interpret H: group H by fact
  let ?A = "G(|carrier := generate G {a}|)"
  let ?B = "H(|carrier := generate H {b}|)"
  interpret A: cyclic_group ?A a by (intro group.cyclic_groupI2; use assms(1)
in simp)
```

```
  interpret B: cyclic_group ?B b by (intro group.cyclic_groupI2; use assms(2)
in simp)
  have sA: "subgroup (generate G {a}) G" by (intro generate_is_subgroup,
use assms(1) in simp)
  have sB: "subgroup (generate H {b}) H" by (intro H.generate_is_subgroup,
use assms(2) in simp)
  fix x
  assume x: "x ∈ {f. ∀k∈(UNIV::int set). f (a [^] k) = b [^]_H k}"
  have hom: "x ∈ hom ?A ?B"
  proof (intro homI)
    fix c
    assume c: "c ∈ carrier ?A"
    from A.elem_is_gen_pow[OF this] obtain k::int where k: "c = a [^]
k"
      using int_pow_consistent[OF sA generate.incl[of a]] by auto
    with x have "x c = b [^]_H k" by blast
    thus "x c ∈ carrier ?B"
      using B.int_pow_closed H.int_pow_consistent[OF sB] generate.incl[of
b "{b}" H] by simp
    fix d
    assume d: "d ∈ carrier ?A"
    from A.elem_is_gen_pow[OF this] obtain l::int where l: "d = a [^]
l"
      using int_pow_consistent[OF sA generate.incl[of a]] by auto
    with k have "c ⊗ d = a [^] (k + l)" by (simp add: int_pow_mult assms(1))
    with x have "x (c ⊗_?A d) = b [^]_H (k + l)" by simp
    also have "... = b [^]_H k ⊗_H b [^]_H l" by (simp add: H.int_pow_mult
assms(2))
    finally show "x (c ⊗_?A d) = x c ⊗_?B x d" using x k l by simp
  qed
  then interpret xgh: group_hom ?A ?B x unfolding group_hom_def group_hom_axioms_def
by blast
  have "kernel ?A ?B x = {1}"
  proof(intro equalityI)
    show "{1} ⊆ kernel ?A ?B x" using xgh.one_in_kernel by auto
    have "c = 1" if "c ∈ kernel ?A ?B x" for c
    proof -
      from that have c: "c ∈ carrier ?A" unfolding kernel_def by blast
      from A.elem_is_gen_pow[OF this] obtain k::int where k: "c = a [^]
k"
        using int_pow_consistent[OF sA generate.incl[of a]] by auto
      moreover have "x c = 1_H" using that x unfolding kernel_def by
auto
      ultimately have "1_H = b [^]_H k" using x by simp
      with assms(3) have "a [^] k = 1"
        using int_pow_eq_id[OF assms(1), of k] H.int_pow_eq_id[OF assms(2),
of k] by simp
      thus "c = 1" using k by blast
    qed
```

```
    thus "kernel ?A ?B x ⊆ {1}" by blast
  qed
  moreover have "carrier ?B ⊆ x ` carrier ?A"
  proof
    fix c
    assume c: "c ∈ carrier ?B"
    from B.elem_is_gen_pow[OF this] obtain k::int where k: "c = b [^]_H
k"
      using H.int_pow_consistent[OF sB generate.incl[of b]] by auto
    then have "x (a [^] k) = c" using x by blast
    moreover have "a [^] k ∈ carrier ?A"
      using int_pow_consistent[OF sA generate.incl[of a]] A.int_pow_closed
generate.incl[of a]
      by fastforce
    ultimately show "c ∈ x ` carrier ?A" by blast
  qed
  ultimately show "x ∈ iso ?A ?B" using hom xgh.iso_iff unfolding kernel_def
by auto
qed
```

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(intro is_isoI)
  interpret H: cyclic_group H h by fact
  define f where "f = (λa. h [^]_H get_exp gen a)"
  from assms(2) have o: "ord gen = H.ord h" using ord_gen_is_group_order
H.ord_gen_is_group_order
    by simp
  have "∀k ∈ (UNIV::int set). f (gen [^] k) = h [^]_H k"
  proof
    fix k
    assume k: "k ∈ (UNIV::int set)"
    have "gen [^] k = gen [^] (SOME x::int. gen [^] k = gen [^] x)"
      by(intro someI_ex[of "λx::int. gen [^] k = gen [^] x"]; blast)
    moreover have "(SOME x::int. gen [^] k = gen [^] x) = (SOME x::int.
h [^]_H k = h [^]_H x)"
    proof -
      have "gen [^] k = gen [^] x ⟷ h [^]_H k = h [^]_H x" for x::int
        by (simp add: o group.int_pow_eq)
      thus ?thesis by simp
    qed
    moreover have "h [^]_H k = h [^]_H (SOME x::int. h [^]_H k = h [^]_H
x)"
      by(intro someI_ex[of "λx::int. h [^]_H k = h [^]_H x"]; blast)
    ultimately show "f (gen [^] k) = h [^]_H k" unfolding f_def get_exp_def
by metis
```

```
  qed
  thus "f ∈ iso G H"
    using iso_cyclic_groups_generate[OF gen_closed H.gen_closed o H.is_group]
    by (auto simp flip: generator H.generator)
qed
```

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(unfold cyclic_group_def cyclic_group_axioms_def, safe)
  show "group (Z n)" using group_integer_mod_group .
  then interpret group "Z n" .
  show oc: "1 ∈ carrier (Z n)"
    unfolding integer_mod_group_def integer_group_def using assms by
force
  show "x ∈ generate (Z n) {1}" if "x ∈ carrier (Z n)" for x
    using generate_pow[OF oc] that int_pow_integer_mod_group solve_equation
subgroup_self
    by fastforce
  show "x ∈ carrier (Z n)" if "x ∈ generate (Z n) {1}" for x using generate_incl[of
"{1}"] that oc
    by fast
qed

lemma Z1_cyclic_group: "cyclic_group (Z 1) 0"
proof(unfold cyclic_group_def cyclic_group_axioms_def, safe)
  show "group (Z 1)" using group_integer_mod_group .
  then interpret group "Z 1" .
  show "0 ∈ carrier (Z 1)" unfolding integer_mod_group_def by simp
  thus "x ∈ carrier (Z 1)" if "x ∈ generate (Z 1) {0}" for x using generate_incl[of
"{0}"] that
    by fast
  show "x ∈ generate (Z 1) {0}" if "x ∈ carrier (Z 1)" for x
  proof -
    from that have "x = 0" unfolding integer_mod_group_def by auto
    with generate.one[of "Z 1" "{0}"] show "x ∈ generate (Z 1) {0}" un-
folding integer_mod_group_def
      by simp
  qed
qed

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

```
  using Z1_cyclic_group Zn_neq1_cyclic_group by metis
```

Moreover, its order is just n.

```
lemma Zn_order: "order (Z n) = n"
  by (unfold integer_mod_group_def integer_group_def order_def, auto)
```

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"
  using Zn_order Zn_cyclic_group iso_cyclic_groups_same_order assms by
metis

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 ⊗_G i y i) I),
                    one = restrict (λi. 1_G i) I ⦈"
```

Basic lemmas about `DirProds`.

```
lemma DirProds_empty:
  "carrier (DirProds f {}) = {1_DirProds f {}}"
  unfolding DirProds_def by auto

lemma DirProds_order:
  assumes "finite I"
  shows "order (DirProds G I) = prod (order ∘ G) I"
  unfolding order_def DirProds_def using assms by (simp add: card_PiE)

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)"
  unfolding DirProds_def using assms by auto
```

```
lemma comp_in_carr:
  assumes "x ∈ carrier (DirProds G I)" "i ∈ I"
  shows "x i ∈ carrier (G i)"
  using assms unfolding DirProds_def by auto

lemma comp_mult:
  assumes "i ∈ I"
  shows "(x ⊗DirProds G I y) i = (x i ⊗G i y i)"
  using assms unfolding DirProds_def by simp

lemma comp_exp_nat:
  fixes k::nat
  assumes "i ∈ I"
  shows "(x [^]DirProds G I k) i = x i [^]G i k"
proof (induction k)
  case 0
  then show ?case using assms unfolding DirProds_def by simp
next
  case i: (Suc k)
  have "(x [^]DirProds G I k ⊗DirProds G I x) i = (x [^]DirProds G I k)
i ⊗G i x i"
    by(rule comp_mult[OF assms])
  also from i have "... = x i [^]G i k ⊗G i x i" by simp
  also have "... = x i [^]G i Suc k" by simp
  finally show ?case by simp
qed

lemma DirProds_m_closed:
  assumes "x ∈ carrier (DirProds G I)" "y ∈ carrier (DirProds G I)"
"⋀i. i ∈ I ⟹ group (G i)"
  shows "x ⊗DirProds G I y ∈ carrier (DirProds G I)"
  using assms monoid.m_closed[OF group.is_monoid[OF assms(3)]] unfold-
ing DirProds_def by fastforce

lemma partial_restr:
  assumes "a ∈ carrier (DirProds G I)" "J ⊆ I"
  shows "restrict a J ∈ carrier (DirProds G J)"
  using assms unfolding DirProds_def by auto

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"
  using assms unfolding DirProds_def by fastforce

lemma mult_restr:
  assumes "a ∈ carrier (DirProds G I)" "b ∈ carrier (DirProds G I)"
"J ⊆ I"
  shows "a ⊗DirProds G J b = restrict (a ⊗DirProds G I b) J"
```

```
  using assms unfolding DirProds_def by force

lemma DirProds_one:
  assumes "x ∈ carrier (DirProds G I)"
  shows "(∀i ∈ I. x i = 1_G i) ⟷ x = 1_DirProds G I"
  using assms unfolding DirProds_def by fastforce

lemma DirProds_one':
  "i∈I ⟹ 1_DirProds G I i = 1_G i"
  unfolding DirProds_def by simp

lemma DirProds_one'':
  "1_DirProds G I = restrict (λi. 1_G i) I"
  by (unfold DirProds_def, simp)

lemma DirProds_mult:
  "(⊗_DirProds G I) = (λx y. restrict (λi. x i ⊗_G i y i) I)"
  unfolding DirProds_def by simp

lemma DirProds_one_iso: "(λx. x G) ∈ iso (DirProds f {G}) (f G)"
proof (intro isoI homI)
  show "bij_betw (λx. x G) (carrier (DirProds f {G})) (carrier (f G))"
  proof (unfold bij_betw_def, rule)
    show "inj_on (λx. x G) (carrier (DirProds f {G}))"
      by (intro inj_onI, unfold DirProds_def PiE_def Pi_def extensional_def,
fastforce)
    show "(λx. x G) ` carrier (DirProds f {G}) = carrier (f G)"
    proof(intro equalityI subsetI)
      show "x ∈ carrier (f G)" if "x ∈ (λx. x G) ` carrier (DirProds
f {G})" for x
        using that unfolding DirProds_def by auto
      show "x ∈ (λx. x G) ` carrier (DirProds f {G})" if xc: "x ∈ carrier
(f G)" for x
      proof
        show "(λk∈{G}. x) ∈ carrier (DirProds f {G})" unfolding DirProds_def
using xc by auto
        moreover show "x = (λk∈{G}. x) G" by simp
      qed
    qed
  qed
qed (unfold DirProds_def PiE_def Pi_def extensional_def, auto)

lemma DirProds_one_cong: "(DirProds f {G}) ≅ (f G)"
  using DirProds_one_iso is_isoI by fast

lemma DirProds_one_iso_sym: "(λx. (λ_∈{G}. x)) ∈ iso (f G) (DirProds
f {G})"
proof (intro isoI homI)
  show "bij_betw (λx. λ_∈{G}. x) (carrier (f G)) (carrier (DirProds f
```

```
{G}))"
  proof (unfold bij_betw_def, rule)
    show "inj_on (λx. (λ_∈{G}. x)) (carrier (f G))"
      by (intro inj_onI, metis imageI image_constant image_restrict_eq
member_remove remove_def)
    show "(λx. (λ_∈{G}. x)) ‘ carrier (f G) = carrier (DirProds f {G})"
      unfolding DirProds_def by fastforce
  qed
qed (unfold DirProds_def, auto)

lemma DirProds_one_cong_sym: "(f G) ≅ (DirProds f {G})"
  using DirProds_one_iso_sym is_isoI by fast
```

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(rule groupI)
  show one_closed: "1_{DirProds G I} ∈ carrier (DirProds G I)" unfolding
DirProds_def
    by (simp add: assms group.is_monoid)
  fix x
  assume x: "x ∈ carrier (DirProds G I)"
  have one_: "⋀i. i ∈ I ⟹ 1_{G i} = 1_{DirProds G I} i" unfolding DirProds_def
by simp
  have "⋀i. i ∈ I ⟹ 1_{DirProds G I} i ⊗_{G i} x i = x i"
  proof -
    fix i
    assume i: "i ∈ I"
    interpret group "G i" using assms[OF i] .
    have "x i ∈ carrier (G i)" using x i comp_in_carr by fast
    thus "1_{DirProds G I} i ⊗_{G i} x i = x i" by(subst one_[OF i, symmetric];
simp)
  qed
  with one_ x show "1_{DirProds G I} ⊗_{DirProds G I} x = x" unfolding DirProds_def
by force
  have "restrict (λi. inv_{G i} (x i)) I ∈ carrier (DirProds G I)" using
x group.inv_closed[OF assms]
    unfolding DirProds_def by fastforce
  moreover have "restrict (λi. inv_{G i} (x i)) I ⊗_{DirProds G I} x = 1_{DirProds G I}"
    using x group.l_inv[OF assms] unfolding DirProds_def by fastforce
  ultimately show "∃y∈carrier (DirProds G I). y ⊗_{DirProds G I} x = 1_{DirProds G I}"
by blast
  fix y
  assume y: "y ∈ carrier (DirProds G I)"
  from DirProds_m_closed[OF x y assms] show m_closed: "x ⊗_{DirProds G I}
y ∈ carrier (DirProds G I)"
    by blast
  fix z
```

```
  assume z: "z ∈ carrier (DirProds G I)"
  have "⋀i. i ∈ I ⟹ (x ⊗DirProds G I y ⊗DirProds G I z) i
                   = (x ⊗DirProds G I (y ⊗DirProds G I z)) i"
  proof -
    fix i
    assume i: "i ∈ I"
    have "(x ⊗DirProds G I y ⊗DirProds G I z) i = (x ⊗DirProds G I y) i
⊗G i z i"
      using assms by (simp add: comp_mult i m_closed z)
    also have "... = x i ⊗G i y i ⊗G i z i" by (simp add: assms comp_mult
i x y)
    also have "... = x i ⊗G i (y i ⊗G i z i)" using i assms x y z
      by (meson Group.group_def comp_in_carr monoid.m_assoc)
    also have "... = (x ⊗DirProds G I (y ⊗DirProds G I z)) i" by (simp
add: DirProds_def i)
    finally show "(x ⊗DirProds G I y ⊗DirProds G I z) i
              = (x ⊗DirProds G I (y ⊗DirProds G I z)) i" .
  qed
  thus "x ⊗DirProds G I y ⊗DirProds G I z = x ⊗DirProds G I (y ⊗DirProds G I
z)"
    unfolding DirProds_def by auto
qed

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 -
  interpret DP: group "DirProds G I" by fact
  from comp_in_carr[OF DP.one_closed] DirProds_one' have ao: "∀j∈I.
1G j ∈ carrier (G j)" by metis
  let ?r = "restrict ((λj. 1G j)(i := x)) I"
  have "?r ∈ carrier (DirProds G I)"
    unfolding DirProds_def PiE_def Pi_def using assms(2, 3) ao by auto
  moreover have "?r i = x" using assms(2) by simp
  ultimately show "(⋀k. ⟦k ∈ carrier (DirProds G I); k i = x⟧ ⟹ thesis)
⟹ thesis" by blast
qed

lemma DirProds_group_imp_groups:
  assumes "group (DirProds G I)" and i: "i ∈ I"
  shows "group (G i)"
proof (intro groupI)
  let ?DP = "DirProds G I"
  interpret DP: group ?DP by fact
  show "1G i ∈ carrier (G i)" using DirProds_one' comp_in_carr[OF DP.one_closed
i] i by metis
  show "x ⊗G i y ∈ carrier (G i)" if "x ∈ carrier (G i)" "y ∈ carrier
(G i)" for x y
  proof -
```

```
    from DirProds_obtain_elem_carr[OF assms that(1)] obtain k where k:
"k ∈ carrier ?DP" "k i = x" .
    from DirProds_obtain_elem_carr[OF assms that(2)] obtain l where l:
"l ∈ carrier ?DP" "l i = y" .
    have "k ⊗?DP l ∈ carrier ?DP" using k l by fast
    from comp_in_carr[OF this i] comp_mult[OF i] show ?thesis using k
l by metis
  qed
  show "x ⊗G i y ⊗G i z = x ⊗G i (y ⊗G i z)"
    if x: "x ∈ carrier (G i)" and y: "y ∈ carrier (G i)" and z: "z ∈
carrier (G i)" for x y z
  proof -
    from DirProds_obtain_elem_carr[OF assms x] obtain k where k: "k ∈
carrier ?DP" "k i = x" .
    from DirProds_obtain_elem_carr[OF assms y] obtain l where l: "l ∈
carrier ?DP" "l i = y" .
    from DirProds_obtain_elem_carr[OF assms z] obtain m where m: "m ∈
carrier ?DP" "m i = z" .
    have "x ⊗G i y ⊗G i z = (k i) ⊗G i (l i) ⊗G i (m i)" using k l m
by argo
    also have "... = (k ⊗?DP l ⊗?DP m) i" using comp_mult[OF i] k l m
by metis
    also have "... = (k ⊗?DP (l ⊗?DP m)) i"
    proof -
      have "k ⊗?DP l ⊗?DP m = k ⊗?DP (l ⊗?DP m)" using DP.m_assoc[OF
k(1) l(1) m(1)] .
      thus ?thesis by simp
    qed
    also have "... = (k i) ⊗G i ((l i) ⊗G i (m i))" using comp_mult[OF
i] k l m by metis
    finally show ?thesis using k l m by blast
  qed
  show "1G i ⊗G i x = x" if "x ∈ carrier (G i)" for x
  proof -
    from DirProds_obtain_elem_carr[OF assms that(1)] obtain k where k:
"k ∈ carrier ?DP" "k i = x" .
    hence "1?DP ⊗?DP k = k" by simp
    with comp_mult k DirProds_one[OF DP.one_closed] that i show ?thesis
by metis
  qed
  show "∃y∈carrier (G i). y ⊗G i x = 1G i" if "x ∈ carrier (G i)" for
x
  proof -
    from DirProds_obtain_elem_carr[OF assms that(1)] obtain k where k:
"k ∈ carrier ?DP" "k i = x" .
    hence ic: "inv?DP k ∈ carrier ?DP" by simp
    have "inv?DP k ⊗?DP k = 1?DP" using k by simp
    hence "(inv?DP k) i ⊗G i k i= 1G i" using comp_mult[OF i] DirProds_one'[OF
i] by metis
```

```
    with k(2) comp_in_carr[OF ic i] show ?thesis by blast
  qed
qed

lemma DirProds_group_iff: "group (DirProds G I) ⟷ (∀i∈I. group (G
i))"
  using DirProds_is_group DirProds_group_imp_groups by metis

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 -
  interpret DP: group "DirProds G I" by fact
  interpret Gi: group "G i" using DirProds_group_imp_groups[OF DP.is_group
i] .
  have ixc: "inv_(DirProds G I) x ∈ carrier (DirProds G I)" using x by
blast
  hence "inv_(DirProds G I) x ⊗_DirProds G I x = 1_DirProds G I" using x by
simp
  hence "(inv_(DirProds G I) x ⊗_DirProds G I x) i = 1_G i" by (simp add:
DirProds_one' i)
  moreover from comp_mult[OF i]
  have "(inv_(DirProds G I) x ⊗_DirProds G I x) i = ((inv_(DirProds G I) x)
i) ⊗_G i (x i)"
    by blast
  ultimately show ?thesis using x ixc by (simp add: comp_in_carr[OF _
i] group.inv_equality)
qed
```

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
  interpret group ?DP using assms DirProds_is_group unfolding comm_group_def
by metis
  show "carrier ?DP ⊆ Units ?DP" "1_?DP ∈ carrier ?DP" by simp_all
  fix x
  assume x[simp]: "x ∈ carrier ?DP"
  show "1_?DP ⊗_?DP x = x" "x ⊗_?DP 1_?DP = x" by simp_all
  fix y
  assume y[simp]: "y ∈ carrier ?DP"
  show "x ⊗_?DP y ∈ carrier ?DP" by simp
  show "x ⊗_DirProds G I y = y ⊗_DirProds G I x"
  proof (rule eq_parts_imp_eq[of _ G I])
    show "x ⊗_?DP y ∈ carrier ?DP" by simp
    show "y ⊗_?DP x ∈ carrier ?DP" by simp
    show "⋀i. i∈I ⟹ (x ⊗_DirProds G I y) i = (y ⊗_DirProds G I x) i"
```

```
    proof -
      fix i
      assume i: "i ∈ I"
      interpret gi: comm_group "(G i)" using assms(1)[OF i] .
      have "(x ⊗?DP y) i = x i ⊗G i y i"
        by (intro comp_mult[OF i])
      also have "... = y i ⊗G i x i" using comp_in_carr[OF _ i] x y gi.m_comm
by metis
      also have "... = (y ⊗?DP x) i" by (intro comp_mult[symmetric, OF
i])
      finally show "(x ⊗DirProds G I y) i = (y ⊗DirProds G I x) i" .
    qed
  qed
  fix z
  assume z[simp]: "z ∈ carrier ?DP"
  show "x ⊗?DP y ⊗?DP z = x ⊗?DP (y ⊗?DP z)" using m_assoc by simp
qed

lemma DirProds_comm_group_imp_comm_groups:
  assumes "comm_group (DirProds G I)" and i: "i ∈ I"
  shows "comm_group (G i)"
proof -
  interpret DP: comm_group "DirProds G I" by fact
  interpret Gi: group "G i" using DirProds_group_imp_groups[OF DP.is_group
i] .
  show "comm_group (G i)"
  proof
    show "x ⊗G i y = y ⊗G i x" if x: "x ∈ carrier (G i)" and y: "y ∈
carrier (G i)" for x y
    proof -
      obtain a where a[simp]: "a ∈ carrier (DirProds G I)" "a i = x"
        using DirProds_obtain_elem_carr[OF DP.is_group i x] .
      obtain b where b[simp]: "b ∈ carrier (DirProds G I)" "b i = y"
        using DirProds_obtain_elem_carr[OF DP.is_group i y] .
      have "a ⊗DirProds G I b = b ⊗DirProds G I a" using DP.m_comm by
simp
      hence "(a ⊗DirProds G I b) i = (b ⊗DirProds G I a) i" by argo
      with comp_mult[OF i] have "a i ⊗G i b i = b i ⊗G i a i" by metis
      with a b show "x ⊗G i y = y ⊗G i x" by blast
    qed
  qed
qed

lemma DirProds_comm_group_iff: "comm_group (DirProds G I) ⟷ (∀i∈I.
comm_group (G i))"
  using DirProds_is_comm_group DirProds_comm_group_imp_comm_groups by
metis
```

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 -
  have "group (G i)" if "i∈I" for i using assms(1)[OF that] unfolding
finite_group_def by blast
  from DirProds_is_group[OF this] interpret DP: group "DirProds G I" by
fast
  show ?thesis
  proof(unfold_locales)
    have "order (DirProds G I) ≠ 0"
    proof(unfold DirProds_order[OF assms(2)])
      have "(order ∘ G) i ≠ 0" if "i∈I" for i
        using assms(1)[OF that] by (simp add: finite_group.order_gt_0)
      thus "prod (order ∘ G) I ≠ 0" by (simp add: assms(2))
    qed
    thus "finite (carrier (DirProds G I))" unfolding order_def by (meson
card.infinite)
  qed
qed

lemma DirProds_finite_imp_finite_groups:
  assumes "finite_group (DirProds G I)" "finite I"
  shows "⋀i. i∈I ⟹ finite_group (G i)"
proof -
  fix i assume i: "i ∈ I"
  interpret DP: finite_group "DirProds G I" by fact
  interpret group "G i" by (rule DirProds_group_imp_groups[OF DP.is_group
i])
  show "finite_group (G i)"
  proof(unfold_locales)
    have oDP: "order (DirProds G I) ≠ 0" by blast
    with DirProds_order[OF assms(2), of G] have "order (G i) ≠ 0" us-
ing i assms(2) by force
    thus "finite (carrier (G i))" unfolding order_def by (meson card_eq_0_iff)
  qed
qed

lemma DirProds_finite_group_iff:
  assumes "finite I"
  shows "finite_group (DirProds G I) ⟷ (∀i∈I. finite_group (G i))"
  using DirProds_is_finite_group DirProds_finite_imp_finite_groups assms
by metis

lemma DirProds_finite_comm_group_iff:
  assumes "finite I"
  shows "finite_comm_group (DirProds G I) ⟷ (∀i∈I. finite_comm_group
(G i))"
  using DirProds_finite_group_iff[OF assms] DirProds_comm_group_iff un-
```

```
folding finite_comm_group_def by fast
```

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 -
  from assms(2) have assm: "J = IDirProds G S I"
                           "compl_fam S I"
    unfolding is_idirprod_def by auto
  from assms(1, 2) have assm': "⋀i. i ∈ I ⟹ subgroup (S i) (G(|carrier
:= J|))"
    using normal_imp_subgroup subgroup_incl by (metis IDirProds_incl assms(2)
is_idirprod_def)
  interpret J: comm_group ?J using subgroup_is_comm_group[OF assms(1)]
.
  interpret DP: comm_group ?DP
    by (intro DirProds_is_comm_group; use J.subgroup_is_comm_group[OF
assm'] in simp)
  have inJ: "S i ⊆ J" if "i ∈ I" for i using subgroup.subset[OF assm'[OF
that]] by simp
  have hom: "?fp ∈ hom ?DP ?J"
  proof (rule homI)
    fix x
    assume x[simp]: "x ∈ carrier ?DP"
    show "finprod G x I ∈ carrier ?J"
    proof (subst finprod_subgroup[OF _ assms(1)])
      show "x ∈ I → J" using inJ comp_in_carr[OF x] by auto
      thus "finprod ?J x I ∈ carrier ?J" by (intro J.finprod_closed;
simp)
    qed
    fix y
    assume y[simp]: "y ∈ carrier ?DP"
    show "finprod G (x ⊗?DP y) I = finprod G x I ⊗?J finprod G y I"
    proof(subst (1 2 3) finprod_subgroup[of _ _ J])
      show xyJ: "x ∈ I → J" "y ∈ I → J" using x y inJ comp_in_carr[OF
x] comp_in_carr[OF y]
        by auto
      show xyJ1: "x ⊗?DP y ∈ I → J" using inJ x y comp_in_carr[of "x
⊗?DP y"] by fastforce
      show "subgroup J G" using assms(1) .
      show "finprod ?J (x ⊗?DP y) I = finprod ?J x I ⊗?J finprod ?J
y I"
      proof (rule J.finprod_cong_split)
        show "x ∈ I → carrier ?J" "y ∈ I → carrier ?J" using xyJ by
simp_all
```

```
        show "x ⊗?DP y ∈ I → carrier ?J" using xyJ1 by simp
        fix i
        assume i: "i ∈ I"
        then have "x i ⊗G(|carrier := (S i) |) y i = (x ⊗?DP y) i"
          by (intro comp_mult[symmetric])
        thus "x i ⊗?J y i = (x ⊗?DP y) i" by simp
      qed
    qed
  qed
  then interpret fp: group_hom ?DP ?J ?fp unfolding group_hom_def group_hom_axioms_def
by blast
  have s: "subgroup (S i) G" if "i ∈ I" for i using incl_subgroup[OF assms(1)
assm'[OF that]] .
  have "kernel ?DP ?J ?fp = {1?DP}"
  proof -
    have "a = 1?DP" if "a ∈ kernel ?DP ?J ?fp" for a
    proof -
      from that have a: "finprod G a I = 1" "a ∈ carrier ?DP" unfold-
ing kernel_def by simp_all
      from compl_fam_imp_triv_finprod[OF assm(2) assms(3) s a(1)] comp_in_carr[OF
a(2)]
      have "∀i∈I. a i = 1" by simp
      then show ?thesis using DirProds_one[OF a(2)] by fastforce
    qed
    thus ?thesis using fp.one_in_kernel by blast
  qed
  moreover have "J ⊆ ?fp ‘ carrier ?DP"
    using assm(1) IDirProds_eq_finprod_PiE[OF assms(3) incl_subgroup[OF
assms(1) assm']]
    unfolding DirProds_def PiE_def Pi_def by simp
  ultimately show ?thesis using hom fp.iso_iff unfolding kernel_def by
auto
qed

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"
  using subgroup_iso_DirProds_IDirProds[OF subgroup_self assms(1, 2)]
by auto

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"
  by (intro is_isoI, use iso_DirProds_IDirProds[OF assms] in blast)
```

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 -
  interpret DG: group "DirProds Gs I" using DirProds_is_group assms(3)
by blast
  interpret DH: group "DirProds Hs J" using DirProds_is_group assms(4)
by blast
  from assms(1) obtain g where g: "g = inv_into I f" "bij_betw g J I"
by (meson bij_betw_inv_into)
  hence fgi: "⋀i. i ∈ I ⟹ g (f i) = i" "⋀j. j ∈ J ⟹ f (g j) =
j"
    using assms(1) bij_betw_inv_into_left[OF assms(1)] bij_betw_inv_into_right[OF
assms(1)] by auto
  from assms(2) have "⋀i. i ∈ I ⟹ (∃h. h ∈ iso (Gs i) (Hs (f i)))"
unfolding is_iso_def by blast
  then obtain h where h: "⋀i. i ∈ I ⟹ h i ∈ iso (Gs i) (Hs (f i))"
by metis
  let ?h = "(λx. (λj. if j∈J then (h (g j)) (x (g j)) else undefined))"
  have hc: "?h x ∈ carrier (DirProds Hs J)" if "x ∈ carrier (DirProds
Gs I)" for x
  proof -
    have xc: "x ∈ carrier (DirProds Gs I)" by fact
    have "h (g j) (x (g j)) ∈ carrier (Hs j)" if "j ∈ J" for j
    proof(intro iso_in_carr[OF _ comp_in_carr[OF xc], of "h (g j)" "g
j" "Hs j"])
      show "g j ∈ I" using g(2)[unfolded bij_betw_def] that by blast
      from h[OF this] show "h (g j) ∈ Group.iso (Gs (g j)) (Hs j)" us-
ing fgi(2)[OF that] by simp
    qed
    thus ?thesis using xc unfolding DirProds_def PiE_def extensional_def
by auto
  qed
  moreover have "?h (x ⊗DirProds Gs I y)= ?h x ⊗DirProds Hs J ?h y"
    if "x ∈ carrier (DirProds Gs I)" "y ∈ carrier (DirProds Gs I)" for
x y
  proof(intro eq_parts_imp_eq[OF hc[OF DG.m_closed[OF that]] DH.m_closed[OF
hc[OF that(1)] hc[OF that(2)]]])
    fix j
    assume j: "j ∈ J"
    hence gj: "g j ∈ I" using g unfolding bij_betw_def by blast
    from assms(3)[OF gj] assms(4)[OF j] have g: "group (Gs (g j))" "Group.group
(Hs j)" .
    from iso_imp_homomorphism[OF h[OF gj]] fgi(2)[OF j] g
    interpret hjh: group_hom "Gs (g j)" "Hs j" "h (g j)"
      unfolding group_hom_def group_hom_axioms_def by simp
    show "(?h (x ⊗DirProds Gs I y)) j = (?h x ⊗DirProds Hs J ?h y) j"
    proof(subst comp_mult)
      show "(if j ∈ J then h (g j) (x (g j) ⊗Gs (g j) y (g j)) else undefined)
```

```
          = (?h x ⊗DirProds Hs J ?h y) j"
      proof(subst comp_mult)
        have "h (g j) (x (g j) ⊗Gs (g j) y (g j)) = h (g j) (x (g j))
⊗Hs j h (g j) (y (g j))"
          using comp_in_carr[OF that(1) gj] comp_in_carr[OF that(2) gj]
by simp
        thus "(if j ∈ J then h (g j) (x (g j) ⊗Gs (g j) y (g j)) else
undefined) =
              (if j ∈ J then h (g j) (x (g j)) else undefined)
      ⊗Hs j (if j ∈ J then h (g j) (y (g j)) else undefined)"
          using j by simp
      qed (use j g that hc in auto)
    qed (use gj g that in auto)
  qed
  ultimately interpret hgh: group_hom "DirProds Gs I" "DirProds Hs J" ?h
    unfolding group_hom_def group_hom_axioms_def by (auto intro: homI)
  have "carrier (DirProds Hs J) ⊆ ?h ` carrier (DirProds Gs I)"
  proof
    show "x ∈ ?h ` carrier (DirProds Gs I)" if xc: "x ∈ carrier (DirProds
Hs J)" for x
    proof -
      from h obtain k where k: "⋀i. i∈I ⟹ k i = inv_into (carrier
(Gs i)) (h i)" by fast
      hence kiso: "⋀i. i∈I ⟹ k i ∈ iso (Hs (f i)) (Gs i)"
        using h by (simp add: assms(3) group.iso_set_sym)
      hence hk: "y = (h (g j) ∘ (k (g j))) y" if "j ∈ J" "y ∈ carrier
(Hs j)" for j y
      proof -
        have gj: "g j ∈ I" using that g[unfolded bij_betw_def] by blast
        thus ?thesis
          using h[OF gj, unfolded iso_def] k[OF gj] that fgi(2)[OF that(1)]
bij_betw_inv_into_right
          unfolding comp_def by fastforce
      qed
      let ?k = "(λi. if i∈I then k i else (λ_. undefined))"
      let ?y = "(λi. (?k i) (x (f i)))"
      have "x j = (λj. if j ∈ J then h (g j) (?y (g j)) else undefined)
j" for j
      proof (cases "j ∈ J")
        case True
        thus ?thesis using hk[OF True comp_in_carr[OF that True]]
                           fgi(2)[OF True] g[unfolded bij_betw_def] by
auto
      next
        case False
        thus ?thesis using that[unfolded DirProds_def] by auto
      qed
      moreover have "?y ∈ carrier (DirProds Gs I)"
      proof -
```

```
        have "?y i ∈ carrier (Gs i)" if i: "i ∈ I" for i
          using k[OF i] h[OF i] comp_in_carr[OF xc] assms(1) bij_betwE
iso_in_carr kiso that
          by fastforce
        moreover have "?y i = undefined" if i: "i ∉ I" for i using i
by simp
        ultimately show ?thesis unfolding DirProds_def PiE_def Pi_def
extensional_def by simp
      qed
      ultimately show ?thesis by fast
    qed
  qed
  moreover have "x = 1_DirProds Gs I"
    if "x ∈ carrier (DirProds Gs I)" "?h x = 1_DirProds Hs J" for x
  proof -
    have "∀i∈I. x i = 1_Gs i"
    proof
      fix i
      assume i: "i ∈ I"
      interpret gi: group "Gs i" using assms(3)[OF i] .
      interpret hfi: group "Hs (f i)" using assms(4) i assms(1)[unfolded
bij_betw_def] by blast
      from h[OF i] interpret hi: group_hom "(Gs i)" "Hs (f i)" "h i"
        unfolding group_hom_def group_hom_axioms_def iso_def by blast
      from that have hx: "?h x ∈ carrier (DirProds Hs J)" by simp
      from DirProds_one[OF this] that(2)
      have "(if j ∈ J then h (g j) (x (g j)) else undefined) = 1_Hs j"
if "j ∈ J" for j
        using that by blast
      hence "h (g (f i)) (x (g (f i))) = 1_Hs (f i)" using i assms(1)[unfolded
bij_betw_def] by auto
      hence "h i (x i) = 1_Hs (f i)" using fgi(1)[OF i] by simp
      with hi.iso_iff h[OF i] comp_in_carr[OF that(1) i] show "x i =
1_Gs i" by fast
    qed
    with DirProds_one that show ?thesis using assms(3) by blast
  qed
  ultimately show ?thesis unfolding is_iso_def using hgh.iso_iff by blast
qed

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 -
  interpret DP: group "DirProds Gs I" using DirProds_is_group assms by
metis
  interpret fDP: group "DirProds (f ∘ Gs) I" using DirProds_is_group assms
by metis
```

```
  from assms have "∀i∈I. (∃g. g ∈ iso (Gs i) ((f ∘ Gs) i))" unfolding is_iso_def by blast
  then obtain J where J: "∀i∈I. J i ∈ iso (Gs i) ((f ∘ Gs) i)" by metis
  let ?J = "(λi. if i∈I then J i else (λ_. undefined))"
  from J obtain K where K: "∀i∈I. K i = inv_into (carrier (Gs i)) (J i)" by fast
  hence K_iso: "∀i∈I. K i ∈ iso ((f ∘ Gs) i) (Gs i)" using group.iso_set_sym assms J by metis
  let ?K = "(λi. if i∈I then K i else (λ_. undefined))"
  have JKi: "(?J i) ((?K i) (x i)) = x i" if "x ∈ carrier (DirProds (f ∘ Gs) I)" for i x
  proof -
    have "(J i) ((K i) (x i)) = x i" if "x ∈ carrier (DirProds (f ∘ Gs) I)" "i ∈ I" for i x
    proof -
      from J that have "(J i) ` (carrier (Gs i)) = carrier ((f ∘ Gs) i)"
        unfolding iso_def bij_betw_def by blast
      hence "∃y. y ∈ carrier (Gs i) ∧ (J i) y = x i" using that by (metis comp_in_carr imageE)
      with someI_ex[OF this] that show "(J i) ((K i) (x i)) = x i"
        using K J K_iso unfolding inv_into_def by auto
    qed
    moreover have "(?J i) ((K i) (x i)) = x i" if "x ∈ carrier (DirProds (f ∘ Gs) I)" "i ∉ I" for i x
      using that unfolding DirProds_def PiE_def extensional_def by force
    ultimately show ?thesis using that by simp
  qed
  let ?r = "(λe. restrict (λi. ?J i (e i)) I)"
  have hom: "?r ∈ hom (DirProds Gs I) (DirProds (f ∘ Gs) I)"
  proof (intro homI)
    show "?r x ∈ carrier (DirProds (f ∘ Gs) I)" if "x ∈ carrier (DirProds Gs I)" for x
      using that J comp_in_carr[OF that] unfolding DirProds_def iso_def bij_betw_def by fastforce
    show "?r (x ⊗DirProds Gs I y) = ?r x ⊗DirProds (f ∘ Gs) I ?r y"
      if "x ∈ carrier (DirProds Gs I)" "y ∈ carrier (DirProds Gs I)" for x y
      using that J comp_in_carr[OF that(1)] comp_in_carr[OF that(2)]
      unfolding DirProds_def iso_def hom_def by force
  qed
  then interpret r: group_hom "(DirProds Gs I)" "(DirProds (f ∘ Gs) I)" ?r
    unfolding group_hom_def group_hom_axioms_def by blast
  have "carrier (DirProds (f ∘ Gs) I) ⊆ ?r ` carrier (DirProds Gs I)"
  proof
    show "x ∈ ?r ` carrier (DirProds Gs I)" if "x ∈ carrier (DirProds (f ∘ Gs) I)" for x
    proof
```

```
      show "x = (λi∈I. ?J i ((?K i) (x i)))"
        using JKi[OF that] that unfolding DirProds_def PiE_def by (simp
add: extensional_restrict)
      show "(λi. ?K i (x i)) ∈ carrier (DirProds Gs I)" using K_iso iso_in_carr
that
        unfolding DirProds_def PiE_def Pi_def extensional_def by fastforce
    qed
  qed
  moreover have "x = 1DirProds Gs I"
    if "x ∈ carrier (DirProds Gs I)" "?r x = 1DirProds (f ∘ Gs) I" for x
  proof -
    have "∀i∈I. x i = 1Gs i"
    proof
      fix i
      assume i: "i ∈ I"
      with J assms interpret Ji: group_hom "(Gs i)" "(f ∘ Gs) i" "J i"
        unfolding group_hom_def group_hom_axioms_def iso_def by blast
      from that have rx: "?r x ∈ carrier (DirProds (f ∘ Gs) I)" by simp
      from i DirProds_one[OF this] that
      have "(λi∈I. (if i ∈ I then J i else (λ_. undefined)) (x i)) i
= 1(f ∘ Gs) i" by blast
      hence "(J i) (x i) = 1(f ∘ Gs) i" using i by simp
      with Ji.iso_iff mp[OF spec[OF J[unfolded Ball_def]] i] comp_in_carr[OF
that(1) i]
      show "x i = 1Gs i" by fast
    qed
    with DirProds_one[OF that(1)] show ?thesis by blast
  qed
  ultimately show ?thesis unfolding is_iso_def using r.iso_iff by blast
qed

lemma DirProds_iso2:
  assumes "inj_on f A" "group (DirProds g (f ` A))"
  shows "DirProds (g ∘ f) A ≅ DirProds g (f ` A)"
proof (intro DirProds_iso[of f])
  show "bij_betw f A (f ` A)" using assms(1) unfolding bij_betw_def by
blast
  show "⋀i. i ∈ A ⟹ (g ∘ f) i ≅ g (f i)" unfolding comp_def using
iso_refl by simp
  from assms(2) show "⋀i. i ∈ (f ` A) ⟹ group (g i)" using DirProds_group_imp_groups
by fast
  with assms(1) show "⋀i. i ∈ A ⟹ group ((g ∘ f) i)" by auto
qed
```

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 (intro is_isoI isoI)
```

```
  let ?f = "λx. restrict (case_prod x) (Sigma I J)"
  show hom: "?f ∈ hom ?L ?R"
  proof(intro homI)
    show "?f a ∈ carrier ?R" if "a ∈ carrier ?L" for a
      using that unfolding DirProds_def PiE_def Pi_def extensional_def
by auto
    show "?f (a ⊗?L b) = ?f a ⊗?R ?f b" if "a ∈ carrier ?L" and "b
∈ carrier ?L" for a b
      using that unfolding DirProds_def PiE_def Pi_def extensional_def
by auto
  qed
  show "bij_betw ?f (carrier ?L) (carrier ?R)"
  proof (intro bij_betwI)
    let ?g = "λx. (λi. if i∈I then (λj. if j∈(J i) then x(i, j) else
undefined) else undefined)"
    show "?f ∈ carrier ?L → carrier ?R" unfolding DirProds_def by fastforce
    show "?g ∈ carrier ?R → carrier ?L" unfolding DirProds_def by fastforce
    show "?f (?g x) = x" if "x ∈ carrier ?R" for x
      using that unfolding DirProds_def PiE_def Pi_def extensional_def
by auto
    show "?g (?f x) = x" if "x ∈ carrier ?L" for x
      using that unfolding DirProds_def PiE_def Pi_def extensional_def
by force
  qed
qed

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"
  unfolding relations_def using assms by blast
```

```
lemma (in comm_group) triv_rel:
  "restrict (λ_. 0::int) A ∈ relations A"
proof
  show "(⨂a∈A. a [^] (λ_∈A. 0::int) a) = 1" by (intro finprod_one_eqI,
simp)
qed simp

lemma (in comm_group) not_triv_relI:
  assumes "a ∈ A" "f a ≠ (0::int)"
  shows "f ≠ (λ_∈A. 0::int)"
  using assms by auto

lemma (in comm_group) rel_in_carr:
  assumes "A ⊆ carrier G" "r ∈ relations A"
  shows "(λa. a [^] r a) ∈ A → carrier G"
  by (meson Pi_I assms(1) int_pow_closed subsetD)
```

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 (rule ccontr; safe)
  fix z::int
  assume z: "z ≠ 0" "a [^] z = 1"
  have "restrict ((λx. 0)(a := z)) A ∈ relations A"
    by (intro in_relationsI finprod_one_eqI, use z in auto)
  thus False using z assms by auto
qed

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"
  using assms relations_zero_imp_pow_not_one[OF assms(1, 2)]
  by (meson finite_cyclic_subgroup_int infinite_cyclic_subgroup_order)

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(rule, rule)
  fix f
  assume f: "f ∈ (Π_E a∈A. generate G {a})" "finprod G f A = 1"
  with generate_pow assms(1) have "∀a∈A. ∃k::int. f a = a [^] k" by
blast
```

```
  then obtain r::"'a ⇒ int" where r: "∀a∈A. f a = a [^] r a" by metis
  have "restrict r A ∈ relations A"
  proof(intro in_relationsI)
    have "(⨂a∈A. a [^] restrict r A a) = finprod G f A"
      by (intro finprod_cong, use assms r in auto)
    thus "(⨂a∈A. a [^] restrict r A a) = 1" using f by simp
  qed simp
  with assms(2) have z: "restrict r A = (λ_∈A. 0)" by blast
  have "(restrict r A) a = r a" if "a ∈ A" for a using that by auto
  with r z show "∀a∈A. f a = 1" by auto
qed

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(rule, rule)
  fix f
  assume f: "f ∈ Pi_E ((λa. generate G {a}) ` A) id" "finprod G f ((λa.
generate G {a}) ` A) = 1"
  define B where "B = inv_into A (λa. generate G {a}) ` ((λa. generate
G {a}) ` A)"
  have Bs: "B ⊆ A"
  proof
    fix x
    assume x: "x ∈ B"
    then obtain C where C: "C ∈ ((λa. generate G {a}) ` A)" "x = inv_into
A (λa. generate G {a}) C"
      unfolding B_def by blast
    then obtain c where c: "C = generate G {c}" "c ∈ A" by blast
    with C someI_ex[of "λy. y ∈ A ∧ generate G {y} = C"] show "x ∈
A"
      unfolding inv_into_def by blast
  qed
  have sI: "(λx. generate G {x}) ` B = (λx. generate G {x}) ` A"
  proof
    show "(λx. generate G {x}) ` B ⊆ (λx. generate G {x}) ` A" using
Bs by blast
    show "(λx. generate G {x}) ` A ⊆ (λx. generate G {x}) ` B"
    proof
      fix C
      assume C: "C ∈ (λx. generate G {x}) ` A"
      then obtain x where x: "x = inv_into A (λa. generate G {a}) C"
unfolding B_def by blast
      then obtain c where c: "C = generate G {c}" "c ∈ A" using C by
blast
      with C x someI_ex[of "λy. y ∈ A ∧ generate G {y} = C"] have "generate
```

```
G {x} = C"
        unfolding inv_into_def by blast
      with x C show "C ∈ (λx. generate G {x}) ‘ B" unfolding B_def by
blast
    qed
  qed
  have fBc: "f (generate G {b}) ∈ carrier G" if "b ∈ B" for b
  proof -
    have "f (generate G {b}) ∈ generate G {b}" using f(1)
      by (subst (asm) sI[symmetric], use that in fastforce)
    moreover have "generate G {b} ⊆ carrier G" using assms(1) that Bs
generate_incl by blast
    ultimately show ?thesis by blast
  qed
  let ?r = "restrict (λa. if a∈B then f (generate G {a}) else 1) A"
  have "?r ∈ Pi_E A (λa. generate G {a})"
  proof
    show "?r x = undefined" if "x ∉ A" for x using that by simp
    show "?r x ∈ generate G {x}" if "x ∈ A" for x using that generate.one
B_def f(1) by auto
  qed
  moreover have "finprod G ?r A = 1"
  proof (cases "finite A")
    case True
    have "A = B ∪ (A - B)" using Bs by auto
    then have "finprod G ?r A = finprod G ?r (B∪(A-B))" by auto
    moreover have "... = finprod G ?r B ⊗ finprod G ?r (A - B)"
    proof(intro finprod_Un_disjoint)
      from True Bs finite_subset show "finite B" "finite (A - B)" "B
∩ (A - B) = {}" by auto
      show "(λa∈A. if a ∈ B then f (generate G {a}) else 1) ∈ A - B
→ carrier G" using Bs by simp
      from fBc show "(λa∈A. if a ∈ B then f (generate G {a}) else 1)
∈ B → carrier G"
        using Bs by auto
    qed
    moreover have "finprod G ?r B = 1"
    proof -
      have "finprod G ?r B = finprod G (f ∘ (λa. generate G {a})) B"
      proof(intro finprod_cong')
        show "?r b = (f ∘ (λa. generate G {a})) b" if "b ∈ B" for b us-
ing that Bs by auto
        show "f ∘ (λa. generate G {a}) ∈ B → carrier G" using fBc by
simp
      qed simp
      also have "... = finprod G f ((λa. generate G {a}) ‘ B)"
      proof(intro finprod_comp[symmetric])
        show "(f ∘ (λa. generate G {a})) ‘ B ⊆ carrier G" using fBc
by auto
```

```
        show "inj_on (λa. generate G {a}) B"
          by (intro inj_onI, unfold B_def, metis (no_types, lifting) f_inv_into_f
inv_into_into)
      qed
      also have "... = finprod G f ((λa. generate G {a}) ‘ A)" using sI
by argo
      finally show ?thesis using f(2) by argo
    qed
    moreover have "finprod G ?r (A - B) = 1" by(intro finprod_one_eqI,
simp)
    ultimately show ?thesis by fastforce
  next
    case False
    then show ?thesis unfolding finprod_def by simp
  qed
  ultimately have a: "∀a∈A. ?r a = 1" using assms(2) by blast
  then have BA: "∀a∈B∩A. ?r a = 1" by blast
  from Bs sI have "∀a∈A. (generate G {a}) ∈ ((λx. generate G {x}) ‘
B)" by simp
  then have "∀a∈A. ∃b∈B. f (generate G {a}) = f (generate G {b})" by
force
  thus "∀H∈(λa. generate G {a}) ‘ A. f H = 1" using a BA Bs by fastforce
qed

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)"
  using assms finprod_relations_triv_harder_better_stronger stronger_PiE_finprod_imp
by presburger

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 -
  have "⋀r. r ∈ relations A ⟹ r = (λ_∈A. 0::int)"
  proof
    fix r x
    assume r: "r ∈ relations A"
    show "r x = (λ_∈A. 0::int) x"
    proof (cases "x ∈ A")
      case True
      let ?r = "restrict (λa. a [^] r a) A"
      have rp: "?r ∈ Pi_E A (λa. generate G {a})"
      proof -
        have "?r ∈ extensional A" by blast
```

```
        moreover have "?r ∈ Pi A (λa. generate G {a})"
        proof
          fix a
          assume a: "a ∈ A"
          then have sga: "subgroup (generate G {a}) G" using generate_is_subgroup
assms(1) by auto
          show "a [^] r a ∈ generate G {a}"
            using generate.incl[of a "{a}" G] subgroup_int_pow_closed[OF
sga] by simp
        qed
        ultimately show ?thesis unfolding PiE_def by blast
      qed
      have "finprod G ?r A = (⨂a∈A. a [^] r a)" by(intro finprod_cong,
use assms(1) in auto)
      with r have "finprod G ?r A = 1" unfolding relations_def by simp
      with assms(3) rp have "∀a∈A. ?r a = 1" by fast
      then have "∀a∈A. a [^] r a = 1" by simp
      with assms(1, 2) True have "r x = 0"
        using finite_cyclic_subgroup_int infinite_cyclic_subgroup_order
by blast
      thus ?thesis using True by simp
    next
      case False
      thus ?thesis using r unfolding relations_def extensional_def by
simp
    qed
  qed
  thus ?thesis using triv_rel by blast
qed

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"
  using triv_finprod_iff_compl_fam_PiE[of _ "λg. generate G {g}", OF assms(1)
generate_is_subgroup]
        ord_zero_strong_imp_rel_triv[OF assms(2, 3)]
        finprod_relations_triv_harder_better_stronger[OF assms(2)] assms
by blast

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"
  using generate_incl generate_sincl by (unfold_locales, auto)
```

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")
  using assms
proof (induction "card A" arbitrary: A rule: nat_less_induct)
  case i: 1
  show ?case
  proof (cases "relations A = {restrict (λ_. 0::int) A}")
    case True
    have fi: "finite A" by fact
    then obtain gs where gs: "set gs = A" "distinct gs" by (meson finite_distinct_list)
    have o: "ord g = 0" if "g ∈ set gs" for g
      by (intro relations_zero_imp_ord_zero[OF that], use i(3) that True
gs in auto)
    have m: "map ord gs = replicate (length gs) 0" using o
      by (induction gs; auto)
    show ?thesis
    proof(rule, safe)
      show "⋀x. x ∈ set gs ⟹ x ∈ generate G A" using gs generate.incl[of
_ A G] by blast
      show "distinct gs" by fact
      show "is_idirprod (generate G A) (λg. generate G {g}) (set gs)"
      proof(unfold is_idirprod_def, intro conjI, rule)
        show "generate G {g} ⊲ G" if "g ∈ set gs" for g
          by (intro subgroup_imp_normal, use that generate_is_subgroup
i(3) gs in auto)
        show "generate G A = IDirProds G (λg. generate G {g}) (set gs)"
unfolding IDirProds_def
```

```
          by (subst gs(1), use generate_idem_Un i(3) in blast)
        show "compl_fam (λg. generate G {g}) (set gs)" using compl_fam_iff_relations_triv[OF
i(2, 3)] o gs(1) True
          by blast
      qed
      show "successively (dvd) (map ord gs)" using m
      proof (induction gs)
        case c: (Cons a gs)
        thus ?case by(cases gs; simp)
      qed simp
      show "card (set gs) ≤ card A" using gs by blast
    qed
  next
    case ntrel: False
    then have Ane: "A ≠ {}"
      using i(2) triv_rel[of A] unfolding relations_def extensional_def
by fastforce
    from ntrel obtain a where a: "a ∈ A" "∃r ∈relations A. r a ≠ 0"
using i(2) triv_rel[of A]
      unfolding relations_def extensional_def by fastforce
    hence ac: "a ∈ carrier G" using i(3) by blast
    have iH: "⋀B.⟦card B < card A; finite B; B ⊆ carrier G⟧ ⟹ ?t B"
      using i(1) by blast
    have iH2: "⋀B. ⟦?t B; generate G A = generate G B; card B < card
A⟧ ⟹ ?t A"
      by fastforce
    show ?thesis
    proof(cases "inv a ∈ (A - {a})")
      case True
      have "generate G A = generate G (A - {a})"
      proof(intro generate_subset_eqI[OF i(3)])
        show "A - (A - {a}) ⊆ generate G (A - {a})"
        proof -
          have "A - (A - {a}) = {a}" using a True by auto
          also have "... ⊆ generate G {inv a}" using generate.inv[of "inv
a" "{inv a}" G] ac by simp
          also have "... ⊆ generate G (A - {a})" by (intro mono_generate,
use True in simp)
          finally show ?thesis .
        qed
      qed simp
      moreover have "?t (A - {a})"
        by (intro iH[of "A - {a}"], use i(2, 3) a(1) in auto, meson Ane
card_gt_0_iff diff_Suc_less)
      ultimately show ?thesis using card.remove[OF i(2) a(1)] by fastforce
    next
      case inv: False
      define n where n: "n = card A"
      define all_gens where
```

```
        "all_gens = {gs∈Pow (generate G A). finite gs ∧ card gs ≤ n
∧ generate G gs = generate G A}"

      define exps where "exps = (⋃gs'∈all_gens. ⋃rel∈relations gs'.
nat ‘ {e∈rel‘gs'. e > 0})"
      define min_exp where "min_exp = Inf exps"

      have "exps ≠ {}"
      proof -
        let ?B = "A - {a} ∪ {inv a}"
        have "A ∈ all_gens" unfolding all_gens_def using generate.incl
n i(2) by fast
        moreover have "?B ∈ all_gens"
        proof -
          have "card (A - {a}) = n - 1" using a n by (meson card_Diff_singleton_if
i(2))
          hence "card ?B = n" using inv i(2, 3) n a(1)
            by (metis Un_empty_right Un_insert_right card.remove card_insert_disjoint
finite_Diff)
          moreover have "generate G A = generate G ?B"
          proof(intro generate_one_switched_eqI[OF i(3) a(1), of _ "inv
a"])
            show "inv a ∈ generate G A" using generate.inv[OF a(1), of
G] .
            show "a ∈ generate G ?B"
            proof -
              have "a ∈ generate G {inv a}" using generate.inv[of "inv
a" "{inv a}" G] ac by simp
              also have "... ⊆ generate G ?B" by (intro mono_generate,
blast)
              finally show ?thesis .
            qed
          qed simp
          moreover hence "?B ⊆ generate G A" using generate_sincl by
simp
          ultimately show ?thesis unfolding all_gens_def using i(2) by
blast
        qed
        moreover have "(∃r ∈ relations A. r a > 0) ∨ (∃r ∈ relations
?B. r (inv a) > 0)"
        proof(cases "∃r ∈ relations A. r a > 0")
          case True
          then show ?thesis by blast
        next
          case False
          with a obtain r where r: "r ∈ relations A" "r a < 0" by force
          have rc: "(λx. x [^] r x) ∈ A → carrier G" using i(3) int_pow_closed
by fast
          let ?r = "restrict (r(inv a := - r a)) ?B"
```

```
          have "?r ∈ relations ?B"
          proof
            have "finprod G (λx. x [^] ?r x) ?B = finprod G (λx. x [^]
r x) A"
            proof -
              have "finprod G (λx. x [^] ?r x) ?B
                  = finprod G (λx. x [^] ?r x) (insert (inv a) (A - {a}))"
by simp
              also have "... = (inv a) [^] ?r (inv a) ⊗ finprod G (λx.
x [^] ?r x) (A - {a})"
              proof(intro finprod_insert[OF _ inv])
                show "finite (A - {a})" using i(2) by fast
                show "inv a [^] ?r (inv a) ∈ carrier G"
                  using int_pow_closed[OF inv_closed[OF ac]] by fast
                show "(λx. x [^] ?r x) ∈ A - {a} → carrier G" using
int_pow_closed i(3) by fast
              qed
              also have "... = a [^] r a ⊗ finprod G (λx. x [^] r x) (A
- {a})"
              proof -
                have "(inv a) [^] ?r (inv a) = a [^] r a"
                  by (simp add: int_pow_inv int_pow_neg ac)
                moreover have "finprod G (λx. x [^] r x) (A - {a})
                        = finprod G (λx. x [^] ?r x) (A - {a})"
                proof(intro finprod_cong)
                  show "((λx. x [^] r x) ∈ A - {a} → carrier G) = True"
using rc by blast
                  have "i [^] r i = i [^] ?r i" if "i ∈ A - {a}" for i
using that inv by auto
                  thus "⋀i. i ∈ A - {a} =simp=> i [^] r i = i [^] restrict
(r(inv a := - r a)) (A - {a} ∪ {inv a}) i"
                    by algebra
                qed simp
                ultimately show ?thesis by argo
              qed
              also have "... = finprod G (λx. x [^] r x) A"
                by (intro finprod_minus[symmetric, OF a(1) rc i(2)])
              finally show ?thesis .
            qed
            also have "... = 1" using r unfolding relations_def by fast
            finally show "finprod G (λx. x [^] ?r x) ?B = 1" .
          qed simp
          then show ?thesis using r by fastforce
        qed
        ultimately show ?thesis unfolding exps_def using a by blast
      qed
      hence me: "min_exp ∈ exps"
        unfolding min_exp_def using Inf_nat_def1 by force
      from cInf_lower min_exp_def have le: "⋀x. x ∈ exps ⟹ min_exp
```

```
≤ x" by blast
      from me obtain gs rel g
        where gr: "gs ∈ all_gens" "rel ∈ relations gs" "g ∈ gs" "rel
g = min_exp" "min_exp > 0"
        unfolding exps_def by auto
      from gr(1) have cgs: "card gs ≤ card A" unfolding all_gens_def
n by blast
      with gr(3) have cgsg: "card (gs - {g}) < card A"
        by (metis Ane card.infinite card_Diff1_less card_gt_0_iff finite.emptyI
                  finite.insertI finite_Diff2 i.prems(1) le_neq_implies_less
less_trans)
      from gr(1) have fgs: "finite gs" and gsg: "generate G gs = generate
G A"
        unfolding all_gens_def n using i(2) card.infinite Ane by force+
      from gsg have gsc: "gs ⊆ carrier G" unfolding all_gens_def
        using generate_incl[OF i(3)] generate_sincl[of gs] by simp
      hence gc: "g ∈ carrier G" using gr(3) by blast
      have ihgsg: "?t (gs - {g})"
        by (intro iH, use cgs fgs gsc gr(3) cgsg in auto)
      then obtain hs where
        hs: "set hs ⊆ generate G (gs - {g})" "distinct hs"
            "is_idirprod (generate G (gs - {g})) (λg. generate G {g})
(set hs)"
            "successively (dvd) (map ord hs)" "card (set hs) ≤ card (gs
- {g})" by blast
      hence hsc: "set hs ⊆ carrier G"
        using generate_sincl[of "set hs"] generate_incl[of "gs - {g}"]
gsc by blast
      from hs(3) have ghs: "generate G (gs - {g}) = generate G (set hs)"
        unfolding is_idirprod_def IDirProds_def using generate_idem_Un[OF
hsc] by argo
      have dvot: "?t A ∨ (∀e∈rel`gs. rel g dvd e)"
      proof(intro disjCI)
        assume na: "¬ (∀e∈rel ` gs. rel g dvd e)"
        have "⋀x. ⟦x ∈ gs; ¬rel g dvd rel x⟧ ⟹ ?t A"
        proof -
          fix x
          assume x: "x ∈ gs" "¬ rel g dvd rel x"
          hence xng: "x ≠ g" by auto
          from x have xc: "x ∈ carrier G" using gsc by blast
          have rg: "rel g > 0" using gr by simp
          define r::int where r: "r = rel x mod rel g"
          define q::int where q: "q = rel x div rel g"
          from r rg x have "r > 0"
            using mod_int_pos_iff[of "rel x" "rel g"] mod_eq_0_iff_dvd
by force
          moreover have "r < rel g" using r rg by simp
          moreover have "rel x = q * rel g + r" using r q by presburger
          ultimately have rq: "rel x = q * (rel g) + r" "0 < r" "r < rel
```

```
g" by auto
          define t where t: "t = g ⊗ x [^] q"
          hence tc: "t ∈ carrier G" using gsc gr(3) x by fast
          define s where s: "s = gs - {g} ∪ {t}"
          hence fs: "finite s" using fgs by blast
          have sc: "s ⊆ carrier G" using s tc gsc by blast
          have g: "generate G gs = generate G s"
          proof(unfold s, intro generate_one_switched_eqI[OF gsc gr(3),
of _ t])
            show "t ∈ generate G gs"
            proof(unfold t, intro generate.eng)
              show "g ∈ generate G gs" using gr(3) generate.incl by
fast
              show "x [^] q ∈ generate G gs"
                using x generate_pow[OF xc] generate_sincl[of "{x}"] mono_generate[of
"{x}" gs]
                by fast
            qed
            show "g ∈ generate G (gs - {g} ∪ {t})"
            proof -
              have gti: "g = t ⊗ inv (x [^] q)"
                using inv_solve_right[OF gc tc int_pow_closed[OF xc, of
q]] t by blast
              moreover have "t ∈ generate G (gs - {g} ∪ {t})" by (intro
generate.incl[of t], simp)
              moreover have "inv (x [^] q) ∈ generate G (gs - {g})"
              proof -
                have "x [^] q ∈ generate G {x}" using generate_pow[OF
xc] by blast
                from generate_m_inv_closed[OF _ this] xc
                have "inv (x [^] q) ∈ generate G {x}" by blast
                moreover have "generate G {x} ⊆ generate G (gs - {g})"
                  by (intro mono_generate, use x a in force)
                finally show ?thesis .
              qed
              ultimately show ?thesis
                using generate.eng mono_generate[of "gs - {g}" "gs - {g}
∪ {t}"] by fast
            qed
          qed simp
          show "⟦x ∈ gs; ¬ rel g dvd rel x⟧ ⟹ ?t A"
          proof (cases "t ∈ gs - {g}")
            case xt: True
            from xt have gts: "s = gs - {g}" using x s by auto
            moreover have "card (gs - {g}) < card gs" using fgs gr(3)
by (meson card_Diff1_less)
            ultimately have "card (set hs) < card A" using hs(5) cgs by
simp
            moreover have "set hs ⊆ generate G (set hs)" using generate_sincl
```

```
by simp
          moreover have "distinct hs" by fact
          moreover have "is_idirprod (generate G (set hs)) (λg. generate
G {g}) (set hs)"
            using hs ghs unfolding is_idirprod_def by blast
          moreover have "generate G A = generate G (set hs)" using
g gts ghs gsg by argo
          moreover have "successively (dvd) (map ord hs)" by fact
          ultimately show "?t A" using iH2 by blast
        next
          case tngsg: False
          hence xnt: "x ≠ t" using x xng by blast
          have "rel g dvd rel x"
          proof (rule ccontr)
            have "nat r ∈ exps" unfolding exps_def
            proof
              show "s ∈ all_gens" unfolding all_gens_def
                using gsg g fgs generate_sincl[of s] switch_elem_card_le[OF
gr(3), of t] cgs n s
                by auto
              have xs: "x ∈ s" using s xng x(1) by blast
              have ts: "t ∈ s" using s by fast
              show "nat r ∈ (⋃rel∈relations s. nat ‘ {e ∈ rel ‘ s.
0 < e})"
              proof
                let ?r = "restrict (rel(x := r, t := rel g)) s"
                show "?r ∈ relations s"
                proof
                  have "finprod G (λx. x [^] ?r x) s = finprod G (λx.
x [^] rel x) gs"
                  proof -
                    have "finprod G (λx. x [^] ?r x) s = x [^] r ⊗
(t [^] rel g ⊗ finprod G (λx. x [^] rel x) (gs - {g} - {x}))"
                    proof -
                      have "finprod G (λx. x [^] ?r x) s = x [^] ?r
x ⊗ finprod G (λx. x [^] ?r x) (s - {x})"
                        by (intro finprod_minus[OF xs _ fs], use sc
in auto)
                      moreover have "finprod G (λx. x [^] ?r x) (s
- {x}) = t [^] ?r t ⊗ finprod G (λx. x [^] ?r x) (s - {x} - {t})"
                        by (intro finprod_minus, use ts xnt fs sc in
auto)
                      moreover have "finprod G (λx. x [^] ?r x) (s
- {x} - {t}) = finprod G (λx. x [^] rel x) (s - {x} - {t})"
                        unfolding s by (intro finprod_cong', use gsc
in auto)
                      moreover have "s - {x} - {t} = gs - {g} - {x}"
unfolding s using tngsg by blast
                      moreover hence "finprod G (λx. x [^] rel x) (s
```

```
- {x} - {t}) = finprod G (λx. x [^] rel x) (gs - {g} - {x})" by simp
                        moreover have "x [^] ?r x = x [^] r" using xs
xnt by auto
                        moreover have "t [^] ?r t = t [^] rel g" us-
ing ts by simp
                        ultimately show ?thesis by argo
                      qed
                      also have "... = x [^] r ⊗ t [^] rel g ⊗ finprod
G (λx. x [^] rel x) (gs - {g} - {x})"
                        by (intro m_assoc[symmetric], use xc tc in simp_all,
intro finprod_closed, use gsc in fast)
                      also have "... = g [^] rel g ⊗ x [^] rel x ⊗ finprod
G (λx. x [^] rel x) (gs - {g} - {x})"
                      proof -
                        have "x [^] r ⊗ t [^] rel g = g [^] rel g ⊗
x [^] rel x"
                        proof -
                          have "x [^] r ⊗ t [^] rel g = x [^] r ⊗ (g
⊗ x [^] q) [^] rel g" using t by blast
                          also have "... = x [^] r ⊗ x [^] (q * rel g)
⊗ g [^] rel g"
                          proof -
                            have "(g ⊗ x [^] q) [^] rel g = g [^] rel
g ⊗ (x [^] q) [^] rel g"
                              using gc xc int_pow_distrib by auto
                            moreover have "(x [^] q) [^] rel g = x [^]
(q * rel g)" using xc int_pow_pow by auto
                            moreover have "g [^] rel g ⊗ x [^] (q *
rel g) = x [^] (q * rel g) ⊗ g [^] rel g"
                              using m_comm[OF int_pow_closed[OF xc] int_pow_closed[OF
gc]] by simp
                            ultimately have "(g ⊗ x [^] q) [^] rel g
= x [^] (q * rel g) ⊗ g [^] rel g" by argo
                            thus ?thesis by (simp add: gc m_assoc xc)
                          qed
                          also have "... = x [^] rel x ⊗ g [^] rel g"
                          proof -
                            have "x [^] r ⊗ x [^] (q * rel g) = x [^]
(q * rel g + r)"
                              by (simp add: add.commute int_pow_mult xc)
                            also have "... = x [^] rel x" using rq by
argo
                            finally show ?thesis by argo
                          qed
                          finally show ?thesis by (simp add: gc m_comm
xc)
                        qed
                        thus ?thesis by simp
                      qed
```

```
                    also have "... = g [^] rel g ⊗ (x [^] rel x ⊗ finprod
G (λx. x [^] rel x) (gs - {g} - {x}))"
                      by (intro m_assoc, use xc gc in simp_all, intro
finprod_closed, use gsc in fast)
                    also have "... = g [^] rel g ⊗ finprod G (λx. x
[^] rel x) (gs - {g})"
                    proof -
                      have "finprod G (λx. x [^] rel x) (gs - {g}) =
x [^] rel x ⊗ finprod G (λx. x [^] rel x) (gs - {g} - {x})"
                        by (intro finprod_minus, use xng x(1) fgs gsc
in auto)
                      thus ?thesis by argo
                    qed
                    also have "... = finprod G (λx. x [^] rel x) gs"
by (intro finprod_minus[symmetric, OF gr(3) _ fgs], use gsc in auto)
                    finally show ?thesis .
                  qed
                  thus "finprod G (λx. x [^] ?r x) s = 1" using gr(2)
unfolding relations_def by simp
                qed auto
                show "nat r ∈ nat ` {e ∈ ?r ` s. 0 < e}" using xs
xnt rq(2) by fastforce
              qed
            qed
            from le[OF this] rq(3) gr(4, 5) show False by linarith
          qed
          thus "⟦x ∈ gs; ¬ rel g dvd rel x⟧ ⟹ ?t A" by blast
        qed
      qed
      thus "?t A" using na by blast
    qed
    show "?t A"
    proof (cases "∀e∈rel`gs. rel g dvd e")
      case dv: True
      define tau where "tau = finprod G (λx. x [^] ((rel x) div rel
g)) gs"
      have tc: "tau ∈ carrier G"
        by (subst tau_def, intro finprod_closed[of "(λx. x [^] ((rel
x) div rel g))" gs], use gsc in fast)
      have gts: "generate G gs = generate G (gs - {g} ∪ {tau})"
      proof(intro generate_one_switched_eqI[OF gsc gr(3), of _ tau])
        show "tau ∈ generate G gs" by (subst generate_eq_finprod_Pi_int_image[OF
fgs gsc], unfold tau_def, fast)
        show "g ∈ generate G (gs - {g} ∪ {tau})"
        proof -
          have "tau = g ⊗ finprod G (λx. x [^] ((rel x) div rel g))
(gs - {g})"
          proof -
            have "finprod G (λx. x [^] ((rel x) div rel g)) gs = g [^]
```

```
(rel g div rel g) ⊗ finprod G (λx. x [^] ((rel x) div rel g)) (gs - {g})"
            by (intro finprod_minus[OF gr(3) _ fgs], use gsc in fast)
          moreover have "g [^] (rel g div rel g) = g" using gr gsc
by auto
          ultimately show ?thesis unfolding tau_def by argo
        qed
        hence gti: "g = tau ⊗ inv finprod G (λx. x [^] ((rel x) div
rel g)) (gs - {g})"
          using inv_solve_right[OF gc tc finprod_closed[of "(λx. x
[^] ((rel x) div rel g))" "gs - {g}"]] gsc
          by fast
        have "tau ∈ generate G (gs - {g} ∪ {tau})" by (intro generate.incl[of
tau], simp)
        moreover have "inv finprod G (λx. x [^] ((rel x) div rel
g)) (gs - {g}) ∈ generate G (gs - {g})"
        proof -
          have "finprod G (λx. x [^] ((rel x) div rel g)) (gs - {g})
∈ generate G (gs - {g})"
            using generate_eq_finprod_Pi_int_image[of "gs - {g}"]
fgs gsc by fast
          from  generate_m_inv_closed[OF _ this] gsc show ?thesis
by blast
        qed
        ultimately show ?thesis by (subst gti, intro generate.eng,
use mono_generate[of "gs - {g}"] in auto)
      qed
    qed simp
    with gr(1) have gt: "generate G (gs - {g} ∪ {tau}) = generate
G A" unfolding all_gens_def by blast
    have trgo: "tau [^] rel g = 1"
    proof -
      have "tau [^] rel g = finprod G (λx. x [^] ((rel x) div rel
g)) gs [^] rel g" unfolding tau_def by blast
      also have "... = finprod G ((λx. x [^] rel g) ∘ (λx. x [^] ((rel
x) div rel g))) gs"
        by (intro finprod_exp, use gsc in auto)
      also have "... = finprod G (λa. a [^] rel a) gs"
      proof(intro finprod_cong')
        show "((λx. x [^] rel g) ∘ (λx. x [^] ((rel x) div rel g)))
x = x [^] rel x" if "x ∈ gs" for x
        proof -
          have "((λx. x [^] rel g) ∘ (λx. x [^] ((rel x) div rel g)))
x = x [^] (((rel x) div rel g) * rel g)"
            using that gsc int_pow_pow by auto
          also have "... = x [^] rel x" using dv that by auto
          finally show ?thesis .
        qed
      qed (use gsc in auto)
      also have "... = 1" using gr(2) unfolding relations_def by blast
```

```
          finally show ?thesis .
        qed
        hence otdrg: "ord tau dvd rel g" using tc int_pow_eq_id by force
        have ot: "ord tau = rel g"
        proof -
          from gr(4, 5) have "rel g > 0" by simp
          with otdrg have "ord tau ≤ rel g" by (meson zdvd_imp_le)
          moreover have "¬ord tau < rel g"
          proof
            assume a: "int (ord tau) < rel g"
            define T where T: "T = gs - {g} ∪ {tau}"
            hence tT: "tau ∈ T" by blast
            let ?r = "restrict ((λ_.(0::int))(tau := int(ord tau))) T"
            from T have "T ∈ all_gens"
              using gt generate_sincl[of "gs - {g} ∪ {tau}"] switch_elem_card_le[OF
gr(3), of tau] fgs cgs n
              unfolding all_gens_def by auto
            moreover have "?r ∈ relations T"
            proof(intro in_relationsI finprod_one_eqI)
              have "tau [^] int (ord tau) = 1" using tc pow_ord_eq_1[OF
tc] int_pow_int by metis
              thus "x [^] ?r x = 1" if "x ∈ T" for x using tT that by(cases
"¬x = tau", auto)
            qed auto
            moreover have "?r tau = ord tau" using tT by auto
            moreover have "ord tau > 0" using dvd_nat_bounds gr(4) gr(5)
int_dvd_int_iff otdrg by presburger
            ultimately have "ord tau ∈ exps" unfolding exps_def using
tT by (auto, force)
            with le a gr(4) show False by force
          qed
          ultimately show ?thesis by auto
        qed
        hence otnz: "ord tau ≠ 0" using gr me exps_def by linarith
        define l where l: "l = tau#hs"
        hence ls: "set l = set hs ∪ {tau}" by auto
        with hsc tc have slc: "set l ⊆ carrier G" by auto
        have gAhst: "generate G A = generate G (set hs ∪ {tau})"
        proof -
          have "generate G A = generate G (gs - {g} ∪ {tau})" using gt
by simp
          also have "... = generate G (set hs ∪ {tau})"
            by (rule generate_subset_change_eqI, use hsc gsc tc ghs in
auto)
          finally show ?thesis .
        qed
        have glgA: "generate G (set l) = generate G A" using gAhst ls
by simp
        have lgA: "set l ⊆ generate G A"
```

```
          using ls gt gts hs(1)
                mono_generate[of "gs - {g}" gs] generate.incl[of tau "gs
- {g} ∪ {tau}"]
          by fast
        show ?thesis
        proof (cases "ord tau = 1")
          case True
          hence "tau = 1" using ord_eq_1 tc by blast
          hence "generate G A = generate G (gs - {g})"
            using gAhst generate_one_irrel hs(3) ghs by auto
          from iH2[OF ihgsg this cgsg] show ?thesis .
        next
          case otau: False
          consider (nd) "¬distinct l" | (ltn) "length l < n ∧ distinct
l" | (dn) "length l = n ∧ distinct l"
          proof -
            have "length l ≤ n"
            proof -
              have "length l = length hs + 1" using l by simp
              moreover have "length hs ≤ card (gs - {g})" using hs(2,
5) by (metis distinct_card)
              moreover have "card (gs - {g}) + 1 ≤ n"
                using n cgsg gr(3) fgs Ane i(2) by (simp add: card_gt_0_iff)
              ultimately show ?thesis by linarith
            qed
            thus "⟦¬ distinct l ⟹ thesis; length l < n ∧ distinct l
⟹ thesis; length l = n ∧ distinct l ⟹ thesis⟧ ⟹ thesis"
              by linarith
          qed
          thus ?thesis
          proof(cases)
            case nd
            with hs(2) l have ths: "set hs = set hs ∪ {tau}" by auto
            hence "set l = set hs" using l by auto
            hence "generate G (gs - {g}) = generate G A" using gAhst
ths ghs by argo
            moreover have "card (set hs) ≤ card A"
              by (metis Diff_iff card_mono cgs dual_order.trans fgs hs(5)
subsetI)
            ultimately show ?thesis using hs by auto
          next
            case ltn
            then have cl: "card (set l) < card A" using n by (metis distinct_card)
            from iH[OF this] hsc tc ls have "?t (set l)" by blast
            thus ?thesis by (subst (1 2) gAhst, use cl ls in fastforce)
          next
            case dn
            hence ln: "length l = n" and dl: "distinct l" by auto
            have c: "complementary (generate G {tau}) (generate G (gs
```

```
- {g}))"
          proof -
            have "x = 1" if "x ∈ generate G {tau} ∩ generate G (set
hs)" for x
            proof -
              from that generate_incl[OF hsc] have xc: "x ∈ carrier
G" by blast
              from that have xgt: "x ∈ generate G {tau}" and xgs:
"x ∈ generate G (set hs)"
                by auto
              from generate_nat_pow[OF otnz tc] xgt have "∃a. a ≥
0 ∧ a < ord tau ∧ x = tau [^] a"
                unfolding atLeastAtMost_def atLeast_def atMost_def
                by (auto, metis Suc_pred less_Suc_eq_le neq0_conv otnz)
              then obtain a where a: "0 ≤ a" "a < ord tau" "x = tau
[^] a" by blast
              then have ix: "inv x ∈ generate G (set hs)"
                using xgs generate_m_inv_closed ghs hsc by blast
              with generate_eq_finprod_Pi_int_image[OF _ hsc] obtain
f where
                f: "f ∈ Pi (set hs) (λ_. (UNIV::int set))" "inv x =
finprod G (λg. g [^] f g) (set hs)"
                by blast
              let ?f = "restrict (f(tau := a)) (set l)"
              have fr: "?f ∈ relations (set l)"
              proof(intro in_relationsI)
                from ls dl l have sh: "set hs = set l - {tau}" by auto
                have "finprod G (λa. a [^] ?f a) (set l) = tau [^] ?f
tau ⊗ finprod G (λa. a [^] ?f a) (set hs)"
                  by (subst sh, intro finprod_minus, use l slc in auto)
                moreover have "tau [^] ?f tau = x" using a l int_pow_int
by fastforce
                moreover have "finprod G (λa. a [^] ?f a) (set hs)
= finprod G (λg. g [^] f g) (set hs)"
                  by (intro finprod_cong', use slc dl l in auto)
                ultimately have "finprod G (λa. a [^] ?f a) (set l)
= x ⊗ inv x" using f by argo
                thus "finprod G (λa. a [^] ?f a) (set l) = 1" using
xc by auto
              qed blast
              have "¬a > 0"
              proof
                assume ag: "0 < a"
                have "set l ∈ all_gens" unfolding all_gens_def us-
ing glgA lgA dn distinct_card
                  by fastforce
                moreover have "int a = ?f tau" using l by auto
                moreover have "tau ∈ set l" using l by simp
                ultimately have "a ∈ exps" using fr ag unfolding exps_def
```

```
by (auto, force)
                  from le[OF this] a(2) ot gr(4) show False by simp
                qed
                hence "a = 0" using a by blast
                thus "x = 1" using tc a by force
              qed
              thus ?thesis unfolding complementary_def using generate.one
ghs by blast
            qed
            moreover have idl: "is_idirprod (generate G A) (λg. generate
G {g}) (set l)"
            proof -
              have "is_idirprod (generate G (set hs ∪ {tau})) (λg. generate
G {g}) (set hs ∪ {tau})"
                by (intro idirprod_generate_ind, use tc hsc hs(3) ghs
c in auto)
              thus ?thesis using ls gAhst by auto
            qed
            moreover have "¬?t A ⟹ successively (dvd) (map ord l)"
            proof (cases hs)
              case Nil
              thus ?thesis using l by simp
            next
              case (Cons a list)
              hence ac: "a ∈ carrier G" using hsc by auto
              assume nA: "¬?t A"
              have "ord tau dvd ord a"
              proof (rule ccontr)
                assume nd: "¬ ord tau dvd ord a"
                then have ‹0 < ord a mod ord tau›
                  using mod_eq_0_iff_dvd by auto
                have "int (ord tau) > 0" using otnz by simp
                obtain r q :: int where rq: "ord a = q * (ord tau) +
r" "0 < r" "r < ord tau"
                  by (rule that [of ‹ord a div ord tau› ‹ord a mod ord
tau›])
                    (use otnz ‹0 < ord a mod ord tau› in ‹simp_all add:
div_mult_mod_eq flip: of_nat_mult of_nat_add›)
                define b where b: "b = tau ⊗ a [^] q"
                hence bc: "b ∈ carrier G" using hsc tc Cons by auto
                have g: "generate G (set (b#hs)) = generate G (set l)"
                proof -
                  have se: "set (b#hs) = set l - {tau} ∪ {b}" using l
Cons dl by auto
                  show ?thesis
                  proof(subst se, intro generate_one_switched_eqI[symmetric,
of _ tau _ b])
                    show "b ∈ generate G (set l)"
                    proof -
```

```
                    have "tau ∈ generate G (set l)" using l generate.incl[of
tau "set l"] by auto
                    moreover have "a [^] q ∈ generate G (set l)"
                      using mono_generate[of "{a}" "set l"] generate_pow[OF
ac] Cons l by auto
                    ultimately show ?thesis using b generate.eng by
fast
                  qed
                  show "tau ∈ generate G (set l - {tau} ∪ {b})"
                  proof -
                    have "tau = b ⊗ inv(a [^] q)" by (simp add: ac
b m_assoc tc)
                    moreover have "b ∈ generate G (set l - {tau} ∪
{b})"
                      using generate.incl[of b "set l - {tau} ∪ {b}"]
by blast
                    moreover have "inv(a [^] q) ∈ generate G (set l
- {tau} ∪ {b})"
                    proof -
                      have "generate G {a} ⊆ generate G (set l - {tau}
∪ {b})"
                        using mono_generate[of "{a}" "set l - {tau}
∪ {b}"] dl Cons l by auto
                      moreover have "inv(a [^] q) ∈ generate G {a}"
                        by (subst generate_pow[OF ac], subst int_pow_neg[OF
ac, of q, symmetric], blast)
                      ultimately show ?thesis by fast
                    qed
                    ultimately show ?thesis using generate.eng by fast
                  qed
                qed (use bc tc hsc dl Cons l in auto)
              qed
              show False
              proof (cases "card (set (b#hs)) ≠ n")
                case True
                hence cln: "card (set (b#hs)) < n"
                  using l Cons ln by (metis card_length list.size(4)
nat_less_le)
                hence seq: "set (b#hs) = set hs"
                proof -
                  from dn l Cons True have "b ∈ set hs"
                    by (metis distinct.simps(2) distinct_card list.size(4))
                  thus ?thesis by auto
                qed
                with cln have clA: "card (set hs) < card A" using n
by auto
                moreover have "set hs ⊆ generate G (set hs)" using
generate_sincl by simp
                moreover have "distinct hs" by fact
```

```
                  moreover have "is_idirprod (generate G (set hs)) (λg.
generate G {g}) (set hs)"
                    by (intro is_idirprod_generate, use hs[unfolded is_idirprod_def]
hsc in auto)
                  moreover have "generate G A = generate G (set hs)"
using glgA g seq by argo
                  moreover have "successively (dvd) (map ord hs)" by
fact
                  ultimately show False using iH2 nA by blast
                next
                  case False
                  hence anb: "a ≠ b"
                    by (metis card_distinct distinct_length_2_or_more
l list.size(4) ln local.Cons)
                  have "nat r ∈ exps" unfolding exps_def
                  proof(rule)
                    show "set (b#hs) ∈ all_gens" unfolding all_gens_def
                      using gAhst g ls generate_sincl[of "set (b#hs)"]
False by simp
                    let ?r = "restrict ((λ_. 0::int)(b := ord tau, a :=
r)) (set (b#hs))"
                    have "?r ∈ relations (set (b#hs))"
                    proof(intro in_relationsI)
                      show "finprod G (λx. x [^] ?r x) (set (b#hs)) =
1"
                      proof -
                        have "finprod G (λx. x [^] ?r x) (set (b#hs))
= b [^] ?r b ⊗ finprod G (λx. x[^] ?r x) (set (b#hs) - {b})"
                          by (intro finprod_minus, use hsc Cons bc in
auto)
                        moreover have "finprod G (λx. x[^] ?r x) (set
(b#hs) - {b}) = a [^] ?r a ⊗ finprod G (λx. x[^] ?r x) (set (b#hs) -
{b} - {a})"
                          by (intro finprod_minus, use hsc Cons False
n anb in auto)
                        moreover have "finprod G (λx. x[^] ?r x) (set
(b#hs) - {b} - {a}) = 1"
                          by (intro finprod_one_eqI, simp)
                        ultimately have "finprod G (λx. x [^] ?r x) (set
(b#hs)) = b [^] ?r b ⊗ (a [^] ?r a ⊗ 1)"
                          by argo
                        also have "... = b [^] ?r b ⊗ a [^] ?r a" us-
ing Cons hsc by simp
                        also have "... = b [^] int(ord tau) ⊗ a [^] r"
using anb Cons by simp
                        also have "... = 1"
                        proof -
                          have "b [^] int (ord tau) = tau [^] int (ord
tau) ⊗ (a [^] q) [^] int (ord tau)"
```

```
                              using b bc hsc int_pow_distrib local.Cons
tc by force
                            also have "... = (a [^] q) [^] int (ord tau)"
                              using trgo hsc local.Cons ot by force
                            finally have "b [^] int (ord tau) ⊗ a [^] r
= (a [^] q) [^] int (ord tau) ⊗ a [^] r"
                              by argo
                            also have "... = a [^] (q * int (ord tau) +
r)" using Cons hsc
                              by (metis comm_group_axioms comm_group_def
group.int_pow_pow
                                         int_pow_mult list.set_intros(1)
subsetD)
                            also have "... = a [^] int (ord a)" using rq
by argo
                            finally show ?thesis using Cons hsc int_pow_eq_id
by simp
                          qed
                          finally show ?thesis .
                        qed
                      qed simp
                      moreover have "r ∈ {e ∈ ?r ‘ set (b # hs). 0 < e}"
                      proof (rule, rule, rule)
                        show "0 < r" by fact
                        show "a ∈ set (b#hs)" using Cons by simp
                        thus "r = ?r a" by auto
                      qed
                      ultimately show "nat r ∈ (⋃rel∈relations (set (b
# hs)). nat ‘ {e ∈ rel ‘ set (b # hs). 0 < e})"
                        by fast
                    qed
                    moreover have "nat r < min_exp" using ot rq(2, 3) gr(4)
by linarith
                    ultimately show False using le by fastforce
                  qed
                qed
                thus ?thesis using hs(4) Cons 1 by simp
              qed
              ultimately show ?thesis using lgA n dn by (metis card_length)
            qed
          qed
        qed (use dvot in blast)
      qed
    qed
qed

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"
  using assms ex_idirgen by meson
```

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 -
  from fundamental_subgr[OF fin_gen gens_closed] obtain gs where
  gs: "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" using
generators by auto
  hence cf: "compl_fam (λg. generate G {g}) (set gs)" by simp
  let ?r = "remove1 1 gs"
  have r: "set ?r = set gs - {1}" using gs by auto
  have "set ?r ⊆ carrier G" using gs by auto
  moreover have "distinct ?r" using gs by auto
  moreover have "is_idirprod (carrier G) (λg. generate G {g}) (set ?r)"
  proof (intro is_idirprod_generate)
    show "set ?r ⊆ carrier G" using gs by auto
    show "compl_fam (λg. generate G {g}) (set (remove1 1 gs))"
      by (rule compl_fam_generate_subset[OF cf gs(1)], use set_remove1_subset
in fastforce)
    show "carrier G = generate G (set ?r)"
    proof -
      have "generate G (set ?r) = generate G (set gs)" using generate_one_irrel'
r by simp
      with gs(3) show ?thesis by simp
    qed
  qed
  moreover have "successively (dvd) (map ord ?r)"
  proof (cases gs)
    case (Cons a list)
    have r: "(map ord (remove1 1 gs)) = remove1 1 (map ord gs)" using
gs(1)
    proof(induction gs)
      case (Cons a gs)
      hence "a ∈ carrier G" by simp
```

```
      with Cons ord_eq_1[OF this] show ?case by auto
    qed simp
    show ?thesis by (unfold r,
                    rule transp_successively_remove1[OF _ gs(4), unfolded
transp_def],
                    auto)
  qed simp
  moreover have "card (set ?r) ≤ card gen" using gs(5) r
    by (metis List.finite_set card_Diff1_le dual_order.trans)
  moreover have "1 ∉ set ?r" using gs(2) by auto
  ultimately show ?thesis using that by blast
qed

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 -
  from invariant_factor_decomposition_idirprod obtain gs where
    gs: "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" by blast
  with cong_DirProds_IDirProds[OF gs(3)] gs
  have "DirProds (λg. G⦇carrier := generate G {g}⦈) (set gs) ≅ G" by
blast
  with gs that show ?thesis by auto
qed

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 -
  from invariant_factor_decomposition_dirprod obtain gs where
  gs: "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" by blast
  let ?gen = "(λg. generate G {g})"
  let ?Hs = "map (λg. ?gen g) gs"
  have "subgroup H G" if "H ∈ set ?Hs" for H using that gs by (auto intro:
generate_is_subgroup)
  moreover have "distinct ?Hs"
    using compl_fam_imp_generate_inj[OF gs(1)] gs distinct_map by blast
```

```
  moreover have "DirProds (λH. G⦇carrier := H⦈) (set ?Hs) ≅ G"
  proof -
    have gg: "group (G⦇carrier := ?gen g⦈)" if "g ∈ set gs" for g
      by (use gs that in ‹auto intro: subgroup.subgroup_is_group generate_is_subgroup›)
    then interpret og: group "DirProds (λg. G⦇carrier := ?gen g⦈) (set
gs)"
      using DirProds_group_iff by blast
    have "DirProds (λg. G⦇carrier := ?gen g⦈) (set gs) ≅ DirProds (λH.
G⦇carrier := H⦈) (set ?Hs)"
    proof (intro DirProds_iso[of ?gen])
      show "bij_betw ?gen (set gs) (set ?Hs)"
        using ‹distinct ?Hs› gs(2) compl_fam_imp_generate_inj[OF gs(1,
6)]
        by (simp add: bij_betw_def)
      show "G⦇carrier := ?gen g⦈ ≅ G⦇carrier := ?gen g⦈" if "g ∈ set
gs" for g by simp
      show "group (G⦇carrier := ?gen g⦈)" if "g ∈ set gs" for g using
that by fact
      show "Group.group (G⦇carrier := H⦈)" if "H ∈ set ?Hs" for H
        by (use gs that in ‹auto intro: subgroup.subgroup_is_group generate_is_subgroup›)
    qed
    from group.iso_sym[OF og.is_group this] show ?thesis using gs iso_trans
by blast
  qed
  moreover have "successively (dvd) (map card ?Hs)"
  proof -
    have "card (generate G {g}) = ord g" if "g ∈ set gs" for g
      using generate_pow_card that gs(1) by auto
    hence "map card ?Hs = map ord gs" by simp
    thus ?thesis using gs(4) by argo
  qed
  moreover have "card (set ?Hs) ≤ card gen" using gs
    by (metis ‹distinct ?Hs› distinct_card length_map)
  moreover have "compl_fam id (set ?Hs)"
    using compl_fam_cong[OF _ compl_fam_imp_generate_inj[OF gs(1, 6)],
of id] using gs by auto
  moreover have "{1} ∉ set ?Hs" using generate_singleton_one gs by auto
  ultimately show ?thesis using that by blast
qed
```

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 -
  from invariant_factor_decomposition_dirprod obtain gs where
    gs: "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" by blast
  let ?DP = "DirProds (λg. G⦇carrier := generate G {g}⦈) (set gs)"
  have "∃ns. DirProds (λn. Z (ns!n)) {..<length ns} ≅ G
          ∧ successively (dvd) ns ∧ length ns ≤ card gen"
  proof (cases gs, rule)
    case Nil
    from gs(3) Nil have co: "carrier ?DP = {1?DP}" unfolding DirProds_def
by auto
    let ?ns = "[]"
    have "DirProds (λn. Z ([] ! n)) {} ≅ ?DP"
    proof(intro triv_iso DirProds_is_group)
      show "carrier (DirProds (λn. Z ([] ! n)) {}) = {1DirProds (λn. Z ([] ! n)) {}}"
        using DirProds_empty by blast
    qed (use co group_integer_mod_group Nil in auto)
    from that[of ?ns] gs co iso_trans[OF this gs(3)]
    show "DirProds (λn. Z (?ns ! n)) {..<length ?ns} ≅ G
        ∧ successively (dvd) ?ns ∧ length ?ns ≤ card gen"
      unfolding lessThan_def by simp
  next
    case c: (Cons a list)
    let ?l = "map ord gs"
    from c have l: "length ?l > 0" by auto
    have "DirProds (λn. Z (?l ! n)) {..<length ?l} ≅ G"
    proof -
      have "DirProds (λn. Z (?l ! n)) {..<length ?l} ≅ ?DP"
      proof(intro DirProds_iso[where ?f = "λn. gs!n"])
        show "bij_betw ((!) gs) {..<length ?l} (set gs)" using gs
          by (simp add: bij_betw_nth)
        show "Z (map ord gs ! i) ≅ G⦇carrier := generate G {gs ! i}⦈"
if "i ∈ {..<length ?l}" for i
        proof(rule group.iso_sym[OF subgroup.subgroup_is_group[OF generate_is_subgroup]
                  cyclic_group.Zn_iso[OF cyclic_groupI2]])
          show "order (G⦇carrier := generate G {gs ! i}⦈) = map ord gs
! i"
            unfolding order_def using that generate_pow_card[of "gs !
i"] gs(1) by force
        qed (use gs(1) that in auto)
        show "Group.group (Z (map ord gs ! i))" if "i ∈ {..<length (map
ord gs)}" for i
          using group_integer_mod_group by blast
        show "Group.group (G⦇carrier := generate G {g}⦈)" if "g ∈ set
gs" for g
          using that gs(1) subgroup.subgroup_is_group[OF generate_is_subgroup]
by auto
      qed
      from iso_trans[OF this gs(3)] show ?thesis .
    qed
    moreover have "length ?l ≤ card gen" using gs by (metis distinct_card
```

```
length_map)
    ultimately show ?thesis using gs c by fastforce
  qed
  thus ?thesis using that by blast
qed
```

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 (cases "n = 1")
  case True
  show ?thesis by (intro triv_iso[OF group_integer_mod_group DirProds_is_group],
                   use DirProds_empty carrier_integer_mod_group True in
auto)
next
  case nno: False
  interpret DP: group ?DP by (intro DirProds_is_group, use group_integer_mod_group
in blast)
  have "order ?DP = prod (order ∘ (λp. Z (p ^ multiplicity p n))) (prime_factors
n)"
    by (intro DirProds_order, blast)
  also have "... = prod (λp. p ^ multiplicity p n) (prime_factors n)"
using Zn_order by simp
  also have n: "... = n" using prod_prime_factors[OF assms] by simp
  finally have oDP: "order ?DP = n" .
  then interpret DP: finite_group ?DP
    by (unfold_locales, unfold order_def, metis assms card.infinite)
  let ?f = "λp∈(prime_factors n). 1"
  have fc: "?f ∈ carrier ?DP"
  proof -
    have p: "0 < multiplicity p n" if "p ∈ prime_factors n" for p
      using prime_factors_multiplicity that by auto
    have pk: "1 < p ^ k" if "Factorial_Ring.prime p" "0 < k" for p k::nat
      using that one_less_power prime_gt_1_nat by blast
    show ?thesis unfolding DirProds_def PiE_def
      by(use carrier_integer_mod_group assms nno pk p in auto,
         metis in_prime_factors_iff nat_int of_nat_power one_less_nat_eq)
  qed
  have of: "DP.ord ?f = n"
  proof -
    have "n dvd j" if j: "?f [^]?DP j = 1?DP" for j
    proof (intro pairwise_coprime_dvd'[OF _ _ n[symmetric]])
      show "finite (prime_factors n)" by simp
      show "∀a∈#prime_factorization n. a ^ multiplicity a n dvd j"
      proof
```

```
        show "p ^ multiplicity p n dvd j" if "p ∈ prime_factors n" for
p
        proof -
          from j have "(?f [^]?DP j) p = 0"
            using that unfolding DirProds_def one_integer_mod_group by
auto
          hence "?f p [^]Z (p ^ multiplicity p n) j = 0" using comp_exp_nat[OF
that] by metis
          hence "group.ord (Z (p ^ multiplicity p n)) (?f p) dvd j" us-
ing comp_in_carr[OF fc that]
            by (metis group.pow_eq_id group_integer_mod_group one_integer_mod_group)
          moreover have "group.ord (Z (p ^ multiplicity p n)) (?f p)
= p ^ multiplicity p n"
            by (metis (no_types, lifting) Zn_neq1_cyclic_group Zn_order
comp_in_carr
                                                cyclic_group.ord_gen_is_group_order
fc integer_mod_group_1
                                                restrict_apply' that)
          ultimately show ?thesis by simp
        qed
      qed
      show "coprime (i ^ multiplicity i n) (j ^ multiplicity j n)"
        if "i ∈# prime_factorization n" "j ∈# prime_factorization n" "i
≠ j" for i j
        using that diff_prime_power_imp_coprime by blast
    qed
    thus ?thesis using fc DP.ord_dvd_group_order gcd_nat.asym oDP by
force
  qed
  interpret DP: cyclic_group ?DP ?f by (intro DP.element_ord_generates_cyclic,
use of oDP fc in auto)
  show ?thesis using DP.iso_sym[OF DP.Zn_iso[OF oDP]] .
qed

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 -
  have cp: "(λp. Z (p ^ multiplicity p n)) = (λp. Z p) ∘ (λp. p ^ multiplicity
p n)" by auto
  have "DirProds (λp. Z (p ^ multiplicity p n)) (prime_factors n) ≅ ?DP"
  proof(subst cp, intro DirProds_iso2)
    show "inj_on (λp. p ^ multiplicity p n) (prime_factors n)"
      by (intro inj_onI; simp add: prime_factors_multiplicity prime_power_inj'')
    show "group (DirProds Z ((λp. p ^ multiplicity p n) ` prime_factors
n))"
      by (intro DirProds_is_group, use group_integer_mod_group in auto)
  qed
```

```
  with Zn_iso_DirProds_prime_powers[OF assms] show ?thesis using Group.iso_trans
by blast
qed

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 -
  from invariant_factor_decomposition_Zn obtain ms where
    ms: "DirProds (λm. Z (ms!m)) {..<length ms} ≅ G" "successively (dvd)
ms" "length ms ≤ card gen"
    by blast
  let ?I = "{..<length ms}"
  let ?J = "λi. if ms!i = 0 then {0} else (λp. p ^ multiplicity p (ms!i))
‘ (prime_factors (ms!i))"
  let ?G = "λi. Z"
  let ?f = "λi. DirProds (?G i) (?J i)"
  have "DirProds (λm. Z (ms!m)) {..<length ms} ≅ DirProds ?f {..<length
ms}"
  proof (intro DirProds_iso[of id])
    show "bij_betw id {..<length ms} {..<length ms}" by blast
    show "Z (ms ! i) ≅ ?f (id i)" if "i ∈ {..<length ms}" for i
      by (cases "ms!i = 0",
          simp add: DirProds_one_cong_sym,
          auto intro: Zn_iso_DirProds_prime_powers’)
    show "⋀i. i ∈ {..<length ms} ⟹ group (Z (ms ! i))" by auto
    show "group (?f j)" if "j ∈ {..<length ms}" for j by (auto intro:
DirProds_is_group)
  qed
  also have "... ≅ DirProds (λ(i,j). ?G i j) (Sigma ?I ?J)"
    by(rule DirProds_Sigma)
  finally have G1: "G ≅ DirProds (λ(i,j). ?G i j) (Sigma ?I ?J)" using
ms(1)
    by (metis (no_types, lifting) DirProds_is_group Group.iso_trans group.iso_sym
group_integer_mod_group)
  have "∃ps. set ps = Sigma ?I ?J ∧ distinct ps" by(intro finite_distinct_list,
auto)
  then obtain ps where ps: "set ps = Sigma ?I ?J" "distinct ps" by blast
  define ns where ns: "ns = map snd ps"
  have "DirProds (λn. Z (ns!n)) {..<length ns} ≅ DirProds (λ(i,j). ?G
i j) (Sigma ?I ?J)"
  proof -
    obtain b::"nat ⇒ (nat × nat)"
      where b: "∀i<length ns. ns!i = snd (b i)" "bij_betw b {..<length
ns} (Sigma ?I ?J)"
      using ns ps bij_betw_nth by fastforce
    moreover have "Z (ns ! i) ≅ (case b i of (i, x) ⇒ Z x)" if "i ∈
```

```
{..<length ns}" for i
    proof -
      have "ns ! i = snd (b i)" using b that by blast
      thus ?thesis by (simp add: case_prod_beta)
    qed
    ultimately show ?thesis by (auto intro: DirProds_iso)
  qed
  with G1 have "DirProds (λn. Z (ns!n)) {..<length ns} ≅ G" using Group.iso_trans
iso_sym by blast
  moreover have "n = 0 ∨ (∃p k. Factorial_Ring.prime p ∧ k > 0 ∧ n
= p ^ k)" if "n∈set ns" for n
  proof -
    have "k = 0 ∨ (∃p∈prime_factors (ms!i). k = p ^ multiplicity p (ms!i))"
if "k ∈ ?J i" for k i
      by (cases "ms!i = 0", use that in auto)
    with that ns ps show ?thesis
      by (auto, metis (no_types, lifting) mem_Collect_eq neq0_conv prime_factors_multiplici
  qed
  ultimately show
  "(⋀ns. ⟦DirProds (λn. Z (ns ! n)) {..<length ns} ≅ G;
          ∀n∈set ns. n = 0 ∨ (∃p k. Factorial_Ring.prime p ∧ k > 0 ∧
n = p ^ k)⟧ ⟹ thesis)
    ⟹ thesis" by blast
qed
```

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 -
  from primary_decomposition_Zn obtain ns where
    ns: "DirProds (λn. Z (ns ! n)) {..<length ns} ≅ G"
        "∀n∈set ns. n = 0 ∨ (∃p k. normalization_semidom_class.prime
p ∧ 0 < k ∧ n = p ^ k)"
    by blast
  have "False" if "n ∈ {..<length ns}" "ns!n = 0" for n
  proof -
    from that have "order (DirProds (λn. Z (ns ! n)) {..<length ns})
= 0"
      using DirProds_order[of "{..<length ns}" "λn. Z (ns!n)"] Zn_order
by auto
    with fin iso_same_card[OF ns(1)] show False unfolding order_def by
auto
  qed
  hence "∀n∈set ns. n≠0"
    by (metis in_set_conv_nth lessThan_iff)
  with ns show ?thesis using that by blast
qed
```

```
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.