The Jordan-Hölder Theorem

Jakob von Raumer

May 26, 2024

Abstract

This submission contains theories that lead to a formalization of the proof of the Jordan-Hölder theorem about composition series of finite groups. The theories formalize the notions of isomorphism classes of groups, simple groups, normal series, composition series, maximal normal subgroups. Furthermore, they provide proofs of the second isomorphism theorem for groups, the characterization theorem for maximal normal subgroups as well as many useful lemmas about normal subgroups and factor groups. The formalization is based on the work work in my first AFP submission [vR14] while the proof of the Jordan-Hölder theorem itself is inspired by course notes of Stuart Rankin [Ran05].

Contents

1 Facts about maximal normal subgroups **1**

2 Normal series and Composition series **4**
2.1 Preliminaries 4
2.2 Normal Series 6
2.3 Composition Series 11

3 Isomorphism Classes of Groups **24**

4 The Jordan-Hölder Theorem **26**

theory *MaximalNormalSubgroups*
imports *HOL−Algebra.Algebra*
begin

1 Facts about maximal normal subgroups

A maximal normal subgroup of G is a normal subgroup which is not contained in other any proper normal subgroup of G.

locale *max-normal-subgroup* = *normal* +

assumes *proper*: $H \neq$ *carrier* G
assumes *max-normal*: $\bigwedge J.\ J \lhd G \Longrightarrow J \neq H \Longrightarrow J \neq$ *carrier* $G \Longrightarrow \neg\ (H \subseteq J)$

Another characterization of maximal normal subgroups: The factor group is simple.

theorem (**in** *normal*) *max-normal-simple-quotient*:
assumes *finite*: *finite* (*carrier G*)
shows *max-normal-subgroup H G* = *simple-group* (*G Mod H*)
proof
assume *max-normal-subgroup H G*
then interpret *maxH*: *max-normal-subgroup H G*.
show *simple-group* (*G Mod H*) **unfolding** *simple-group-def simple-group-axioms-def*
proof (*intro conjI factorgroup-is-group allI impI disjCI*)
have *gt0*: $0 <$ *card* (*rcosets H*)
by (*metis gr-zeroI lagrange-finite assms mult-is-0 order-gt-0-iff-finite subgroup-axioms*)
from *maxH.proper finite* **have** *carrier* (*G Mod H*) $\neq \{\mathbf{1}_{G\ Mod\ H}\}$ **using** *fact-group-trivial-iff* **by** *auto*
hence $1 \neq$ *order* (*G Mod H*) **using** *factorgroup-is-group group.order-one-triv-iff* **by** *metis*
with *gt0* **show** $1 <$ *order* (*G Mod H*) **unfolding** *order-def FactGroup-def* **by** *auto*
next
fix A'
assume A'*normal*: $A' \lhd G$ *Mod H* **and** A'*nottriv*: $A' \neq \{\mathbf{1}_{G\ Mod\ H}\}$
define A **where** $A = \bigcup A'$
have *A2*: $A \lhd G$ **using** A'*normal* **unfolding** *A-def* **by** (*rule factgroup-subgroup-union-normal*)
have $H \in A'$ **using** A'*normal normal-imp-subgroup subgroup.one-closed* **unfolding** *FactGroup-def* **by** *force*
hence $H \subseteq A$ **unfolding** *A-def* **by** *auto*
hence *A1*: $H \lhd (G(\!|$*carrier* $:= A|\!))$
by (*simp add*: *A2 normal-axioms normal-invE*(*1*) *normal-restrict-supergroup*)
have *A3*: $A' =$ *rcosets*$_{G(\!|carrier\ :=\ A|\!)}$ H
unfolding *A-def* **using** *factgroup-subgroup-union-factor* A'*normal normal-imp-subgroup* **by** *auto*
from *A1* **interpret** *normalHA*: *normal H* $(G(\!|$*carrier* $:= A|\!))$ **by** *metis*
have $H \subseteq A$ **using** *normalHA.is-subgroup subgroup.subset* **by** *force*
with *A2* **have** $A = H \vee A =$ *carrier G* **using** *maxH.max-normal* **by** *auto*
thus $A' =$ *carrier* (*G Mod H*)
proof
assume $A = H$
hence *carrier* $(G(\!|$*carrier* $:= A|\!)$ *Mod H*$) = \{\mathbf{1}_{(G(\!|carrier\ :=\ A|\!)\ Mod\ H)}\}$
using *cosets-finite subgroup-in-rcosets subset assms normalHA.fact-group-trivial-iff* **by** *force*
then have $A' = \{\mathbf{1}_{G\ Mod\ H}\}$
using *A3* **unfolding** *FactGroup-def* **by** *simp*
with A'*nottriv* **show** *?thesis*..
next

assume $A = carrier\ G$
thus $A' = carrier\ (G\ Mod\ H)$ **using** *A3* **unfolding** *FactGroup-def* **by** *simp*
qed
qed
next
assume *simple*: *simple-group* $(G\ Mod\ H)$
show *max-normal-subgroup* $H\ G$
proof
from *simple* **have** $carrier\ (G\ Mod\ H) \neq \{\mathbf{1}_{G\ Mod\ H}\}$ **unfolding** *simple-group-def simple-group-axioms-def order-def* **by** *auto*
with *finite fact-group-trivial-iff* **show** $H \neq carrier\ G$ **by** *auto*
next
fix A
assume A: $A \lhd G\ A \neq H\ A \neq carrier\ G$
show $\neg\ H \subseteq A$
proof
assume *HA*: $H \subseteq A$
hence $H \lhd (G(|carrier := A|))$ **by** (*metis* $A(1)$ *inv-op-closed2 is-subgroup normal-inv-iff normal-restrict-supergroup*)
then interpret *normalHA*: *normal* $H\ (G(|carrier := A|))$ **by** *simp*
from *finite* **have** *finiteA*: *finite* A
by (*meson* $A(1)$ *normal-inv-iff finite-subset subgroup.subset*)
have $rcosets_{(G(|carrier := A|))}\ H \lhd G\ Mod\ H$
by (*simp add*: $A(1)$ *HA normal-axioms normality-factorization*)
with *simple* **have** $rcosets_{(G(|carrier := A|))}\ H = \{\mathbf{1}_{G\ Mod\ H}\} \vee rcosets_{(G(|carrier := A|))}\ H = carrier\ (G\ Mod\ H)$
unfolding *simple-group-def simple-group-axioms-def* **by** *auto*
thus *False*
proof
assume $rcosets_{G(|carrier := A|)}\ H = \{\mathbf{1}_{G\ Mod\ H}\}$
with *finiteA* **have** $H = A$
using *normalHA.fact-group-trivial-iff* **unfolding** *FactGroup-def* **by** *auto*
with $A(2)$ **show** *?thesis* **by** *simp*
next
assume *AHGH*: $rcosets_{G(|carrier := A|)}\ H = carrier\ (G\ Mod\ H)$
have $A = carrier\ G$ **unfolding** *FactGroup-def RCOSETS-def*
proof
show $A \subseteq carrier\ G$ **using** $A(1)$ *normal-imp-subgroup subgroup.subset* **by** *metis*
next
show $carrier\ G \subseteq A$
proof
fix x
assume x: $x \in carrier\ G$
hence $H\ \#>\ x \in rcosets\ H$ **unfolding** *RCOSETS-def* **by** *auto*
with *AHGH* **have** $H\ \#>\ x \in rcosets_{G(|carrier := A|)}\ H$ **unfolding** *FactGroup-def* **by** *simp*
then obtain x' **where** x': $x' \in A\ H\ \#{>}x = H\ \#{>}_{G(|carrier := A|)}\ x'$

unfolding *RCOSETS-def* **by** *auto*
hence *H* #> *x* = *H* #> *x'* **unfolding** *r-coset-def* **by** *auto*
hence *x* ∈ *H* #> *x'* **by** (*metis is-subgroup rcos-self x*)
hence *x* ∈ *A* #> *x'* **using** *HA* **unfolding** *r-coset-def* **by** *auto*
thus *x* ∈ *A* **using** *x'*(*1*) **unfolding** *r-coset-def* **using** *subgroup.m-closed* *A*(*1*) *normal-imp-subgroup* **by** *force*
qed
qed
with *A*(*3*) **show** *?thesis* **by** *simp*
qed
qed
qed
qed

end

theory *CompositionSeries*
imports
MaximalNormalSubgroups Secondary-Sylow.SndSylow
begin

hide-const (**open**) *Divisibility.prime*

2 Normal series and Composition series

2.1 Preliminaries

A subgroup which is unique in cardinality is normal:

lemma (**in** *group*) *unique-sizes-subgrp-normal*:
assumes *fin*: *finite* (*carrier G*)
assumes ∃!*Q*. *Q* ∈ *subgroups-of-size q*
shows (*THE Q*. *Q* ∈ *subgroups-of-size q*) ⊲ *G*
proof −
from *assms* **obtain** *Q* **where** *Q* ∈ *subgroups-of-size q* **by** *auto*
define *Q* **where** *Q* = (*THE Q*. *Q* ∈ *subgroups-of-size q*)
with *assms* **have** *Qsize*: *Q* ∈ *subgroups-of-size q* **using** *theI* **by** *metis*
hence *QG*: *subgroup Q G* **and** *cardQ*: *card Q* = *q* **unfolding** *subgroups-of-size-def* **by** *auto*
from *QG* **have** *Q* ⊲ *G* **apply**(*rule normalI*)
proof
fix *g*
assume *g*: *g* ∈ *carrier G*
hence *invg*: *inv g* ∈ *carrier G* **by** (*metis inv-closed*)
with *fin Qsize* **have** *conjugation-action q* (*inv g*) *Q* ∈ *subgroups-of-size q* **by** (*metis conjugation-is-size-invariant*)
with *g Qsize* **have** (*inv g*) <# (*Q* #> *inv* (*inv g*)) ∈ *subgroups-of-size q* **unfolding** *conjugation-action-def* **by** *auto*

with *invg g* **have** *inv g <# (Q #> g) = Q* **by** (*metis Qsize assms*(*2*) *inv-inv*)
with *QG QG g* **show** *Q #> g = g <# Q* **by** (*rule conj-wo-inv*)
qed
with *Q-def* **show** *?thesis* **by** *simp*
qed

A group whose order is the product of two distinct primes p and q where $p < q$ has a unique subgroup of size q:

lemma (**in** *group*) *pq-order-unique-subgrp*:
assumes *finite*: *finite* (*carrier G*)
assumes *orderG*: *order G = q ∗ p*
assumes *primep*: *prime p* **and** *primeq*: *prime q* **and** *pq*: *p < q*
shows ∃!*Q*. *Q* ∈ (*subgroups-of-size q*)
proof −
from *primep primeq pq* **have** *nqdvdp*: ¬ (*q dvd p*) **by** (*metis less-not-refl3 prime-nat-iff*)
define *calM* **where** *calM* = {*s*. *s* ⊆ *carrier G* ∧ *card s* = *q* ^ *1*}
define *RelM* **where** *RelM* = {(*N1*, *N2*). *N1* ∈ *calM* ∧ *N2* ∈ *calM* ∧ (∃ *g*∈*carrier G*. *N1* = *N2* #> *g*)}
interpret *syl*: *snd-sylow G q 1 p calM RelM*
unfolding *snd-sylow-def sylow-def snd-sylow-axioms-def sylow-axioms-def*
using *is-group primeq orderG finite nqdvdp calM-def RelM-def* **by** *auto*
obtain *Q* **where** *Q*: *Q* ∈ *subgroups-of-size q* **by** (*metis* (*lifting*, *mono-tags*) *mem-Collect-eq power-one-right subgroups-of-size-def syl.sylow-thm*)
thus *?thesis*
proof (*rule ex1I*)
fix *P*
assume *P*: *P* ∈ *subgroups-of-size q*
have *card* (*subgroups-of-size q*) *mod q* = *1* **by** (*metis power-one-right syl.p-sylow-mod-p*)

moreover have *card* (*subgroups-of-size q*) *dvd p* **by** (*metis power-one-right syl.num-sylow-dvd-remainder*)
then have *card* (*subgroups-of-size q*) = *p* ∨ *card* (*subgroups-of-size q*) = *1*
using *primep* **by** (*auto simp add*: *prime-nat-iff*)
ultimately have *card* (*subgroups-of-size q*) = *1* **using** *pq*
by *auto*
with *Q P* **show** *P = Q* **by** (*auto simp*:*card-Suc-eq*)
qed
qed

... And this unique subgroup is normal.

corollary (**in** *group*) *pq-order-subgrp-normal*:
assumes *finite*: *finite* (*carrier G*)
assumes *orderG*: *order G = q ∗ p*
assumes *primep*: *prime p* **and** *primeq*: *prime q* **and** *pq*: *p < q*
shows (*THE Q*. *Q* ∈ *subgroups-of-size q*) ⊲ *G*
using *assms* **by** (*metis pq-order-unique-subgrp unique-sizes-subgrp-normal*)

The trivial subgroup is normal in every group.

lemma (**in** *group*) *trivial-subgroup-is-normal*:
shows $\{\mathbf{1}\} \lhd G$
unfolding *normal-def normal-axioms-def r-coset-def l-coset-def* **by** (*auto intro*: *normalI subgroupI simp*: *is-group*)

2.2 Normal Series

We define a normal series as a locale which fixes one group G and a list $\mathfrak{G}$ of subsets of G's carrier. This list must begin with the trivial subgroup, end with the carrier of the group itself and each of the list items must be a normal subgroup of its successor.

locale *normal-series* = *group* +
fixes $\mathfrak{G}$
assumes *notempty*: $\mathfrak{G} \neq []$
assumes *hd*: *hd* $\mathfrak{G} = \{\mathbf{1}\}$
assumes *last*: *last* $\mathfrak{G}$ = *carrier G*
assumes *normal*: $\bigwedge i.\ i + 1 < length\ \mathfrak{G} \Longrightarrow (\mathfrak{G}\ !\ i) \lhd G(\!|carrier := \mathfrak{G}\ !\ (i + 1)|\!)$

lemma (**in** *normal-series*) *is-normal-series*: *normal-series G* $\mathfrak{G}$ **by** (*rule normal-series-axioms*)

For every group there is a "trivial" normal series consisting only of the group itself and its trivial subgroup.

lemma (**in** *group*) *trivial-normal-series*:
shows *normal-series G* $[\{\mathbf{1}\}$, *carrier G*$]$
unfolding *normal-series-def normal-series-axioms-def*
using *is-group trivial-subgroup-is-normal* **by** *auto*

We can also show that the normal series presented above is the only such with a length of two:

lemma (**in** *normal-series*) *length-two-unique*:
assumes *length* $\mathfrak{G} = 2$
shows $\mathfrak{G} = [\{\mathbf{1}\}$, *carrier G*$]$
proof(*rule nth-equalityI*)
from *assms* **show** *length* $\mathfrak{G}$ = *length* $[\{\mathbf{1}\}$, *carrier G*$]$ **by** *auto*
next
show $\mathfrak{G}\ !\ i = [\{\mathbf{1}\}$, *carrier G*$]\ !\ i$ **if** *i*: $i <$ *length* $\mathfrak{G}$ **for** *i*
proof −
have $i = 0 \lor i = 1$ **using** *that assms* **by** *auto*
thus $\mathfrak{G}\ !\ i = [\{\mathbf{1}\}$, *carrier G*$]\ !\ i$
proof(*rule disjE*)
assume *i*: $i = 0$
hence $\mathfrak{G}\ !\ i$ = *hd* $\mathfrak{G}$ **by** (*metis hd-conv-nth notempty*)
thus $\mathfrak{G}\ !\ i = [\{\mathbf{1}\}$, *carrier G*$]\ !\ i$ **using** *hd i* **by** *simp*
next
assume *i*: $i = 1$
with *assms* **have** $\mathfrak{G}\ !\ i$ = *last* $\mathfrak{G}$ **by** (*metis diff-add-inverse last-conv-nth nat-1-add-1 notempty*)

thus 𝔊 ! *i* = [{**1**}, *carrier G*] ! *i* **using** *last i* **by** *simp*
qed
qed
qed

We can construct new normal series by expanding existing ones: If we append the carrier of a group G to a normal series for a normal subgroup $H \lhd G$ we receive a normal series for G.

lemma (**in** *group*) *normal-series-extend*:
assumes *normal*: *normal-series* (*G*(|*carrier* := *H*|)) 𝔥
assumes *HG*: *H* ◁ *G*
shows *normal-series G* (𝔥 @ [*carrier G*])
proof −
from *normal* **interpret** *normalH*: *normal-series* (*G*(|*carrier* := *H*|)) 𝔥**.**
from *normalH.hd* **have** *hd* 𝔥 = {**1**} **by** *simp*
with *normalH.notempty* **have** *hdTriv*: *hd* (𝔥 @ [*carrier G*]) = {**1**} **by** (*metis hd-append2*)
show *?thesis* **unfolding** *normal-series-def normal-series-axioms-def* **using** *is-group*
proof *auto*
fix *x*
assume *x* ∈ *hd* (𝔥 @ [*carrier G*])
with *hdTriv* **show** *x* = **1** **by** *simp*
next
from *hdTriv* **show** **1** ∈ *hd* (𝔥 @ [*carrier G*]) **by** *simp*
next
fix *i*
assume *i*: *i* < *length* 𝔥
show (𝔥 @ [*carrier G*]) ! *i* ◁ *G*(|*carrier* := (𝔥 @ [*carrier G*]) ! *Suc i*|)
proof (*cases i + 1 < length* 𝔥)
case *True*
with *normalH.normal* **have** 𝔥 ! *i* ◁ *G*(|*carrier* := 𝔥 ! (*i* + *1*)|) **by** *auto*
with *i* **have** (𝔥 @ [*carrier G*]) ! *i* ◁ *G*(|*carrier* := 𝔥 ! (*i* + *1*)|) **using** *nth-append* **by** *metis*
with *True* **show** (𝔥 @ [*carrier G*]) ! *i* ◁ *G*(|*carrier* := (𝔥 @ [*carrier G*]) ! (*Suc i*)|) **using** *nth-append Suc-eq-plus1* **by** *metis*
next
case *False*
with *i* **have** *i2*: *i* + *1* = *length* 𝔥 **by** *simp*
from *i* **have** (𝔥 @ [*carrier G*]) ! *i* = 𝔥 ! *i* **by** (*metis nth-append*)
also from *i2 normalH.notempty* **have** ... = *last* 𝔥 **by** (*metis add-diff-cancel-right′ last-conv-nth*)
also from *normalH.last* **have** ... = *H* **by** *simp*
finally have (𝔥 @ [*carrier G*]) ! *i* = *H***.**
moreover from *i2* **have** (𝔥 @ [*carrier G*]) ! (*i* + *1*) = *carrier G* **by** (*metis nth-append-length*)
ultimately show *?thesis* **using** *HG* **by** *auto*
qed
qed
qed

All entries of a normal series for G are subgroups of G.

lemma (**in** *normal-series*) *normal-series-subgroups*:
  shows *i* < *length* 𝔊 ⟹ *subgroup* (𝔊 ! *i*) *G*
proof −
  have *i* + *1* < *length* 𝔊 ⟹ *subgroup* (𝔊 ! *i*) *G*
  proof (*induction length* 𝔊 − (*i* + *2*) *arbitrary*: *i*)
    case *0*
    hence *i*: *i* + *2* = *length* 𝔊 **by** *simp*
    hence *ii*: *i* + *1* = *length* 𝔊 − *1* **by** *force*
    from *i normal* **have** 𝔊 ! *i* ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ **by** *auto*
     with *ii last notempty* **show** *subgroup* (𝔊 ! *i*) *G* **using** *last-conv-nth normal-imp-subgroup* **by** *fastforce*
  next
    case (*Suc k*)
     from *Suc*(*3*) *normal* **have** *i*: *subgroup* (𝔊 ! *i*) (*G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈) **using** *normal-imp-subgroup* **by** *auto*
    from *Suc*(*2*) **have** *k*: *k* = *length* 𝔊 − ((*i* + *1*) + *2*) **by** *arith*
    with *Suc* **have** *subgroup* (𝔊 ! (*i* + *1*)) *G* **by** *simp*
    with *i* **show** *subgroup* (𝔊 ! *i*) *G*
      using *incl-subgroup* **by** *blast*
  qed
  moreover have *i* + *1* = *length* 𝔊 ⟹ *subgroup* (𝔊 ! *i*) *G*
   using *last notempty last-conv-nth* **by** (*metis add-diff-cancel-right′ subgroup-self*)
  ultimately show *i* < *length* 𝔊 ⟹ *subgroup* (𝔊 ! *i*) *G* **by** *force*
qed

The second to last entry of a normal series is a normal subgroup of G.

lemma (**in** *normal-series*) *normal-series-snd-to-last*:
  shows 𝔊 ! (*length* 𝔊 − *2*) ⊲ *G*
proof (*cases 2* ≤ *length* 𝔊)
  case *False*
 with *notempty* **have** *length*: *length* 𝔊 = *1* **by** (*metis Suc-eq-plus1 leI length-0-conv less-2-cases plus-nat.add-0*)
  with *hd* **have** 𝔊 ! (*length* 𝔊 − *2*) = {**1**} **using** *hd-conv-nth notempty* **by** *auto*
  with *length* **show** *?thesis* **by** (*metis trivial-subgroup-is-normal*)
next
  case *True*
  hence (*length* 𝔊 − *2*) + *1* < *length* 𝔊 **by** *arith*
  with *normal last* **have** 𝔊 ! (*length* 𝔊 − *2*) ⊲ *G*⦇*carrier* := 𝔊 ! ((*length* 𝔊 − *2*) + *1*)⦈ **by** *auto*
  have *1* + (*1* + (*length* 𝔊 − (*1* + *1*))) = *length* 𝔊
    using *True le-add-diff-inverse* **by** *presburger*
  then have 𝔊 ! (*length* 𝔊 − *2*) ⊲ *G*⦇*carrier* := 𝔊 ! (*length* 𝔊 − *1*)⦈
    by (*metis* ‹𝔊 ! (*length* 𝔊 − *2*) ⊲ *G* ⦇*carrier* := 𝔊 ! (*length* 𝔊 − *2* + *1*)⦈› *add.commute add-diff-cancel-left′ one-add-one*)
  with *notempty last* **show** *?thesis* **using** *last-conv-nth* **by** *force*
qed

Just like the expansion of normal series, every prefix of a normal series is again a normal series.

lemma (**in** *normal-series*) *normal-series-prefix-closed*:
assumes *i* ≤ *length* 𝔊 **and** *0* < *i*
shows *normal-series* (*G*⦇*carrier* := 𝔊 ! (*i* − *1*)⦈) (*take i* 𝔊)
unfolding *normal-series-def normal-series-axioms-def*
using *assms*
apply (*auto simp*: *hd del*:*equalityI*)
apply (*simp add*: *is-group normal-series-subgroups subgroup.subgroup-is-group*)
apply (*simp add*: *last-conv-nth min.absorb2 notempty*)
using *assms*(*1*) *normal* **apply** *simp*
done

If a group's order is the product of two distinct primes p and q, where $p < q$, we can construct a normal series using the only subgroup of size q.

lemma (**in** *group*) *pq-order-normal-series*:
assumes *finite*: *finite* (*carrier G*)
assumes *orderG*: *order G* = *q* ∗ *p*
assumes *primep*: *prime p* **and** *primeq*: *prime q* **and** *pq*: *p* < *q*
shows *normal-series G* [{**1**}, (*THE H*. *H* ∈ *subgroups-of-size q*), *carrier G*]
proof −
define *H* **where** *H* = (*THE H*. *H* ∈ *subgroups-of-size q*)
with *assms* **have** *HG*: *H* ⊲ *G* **by** (*metis pq-order-subgrp-normal*)
then interpret *groupH*: *group G*⦇*carrier* := *H*⦈ **unfolding** *normal-def* **by** (*metis subgroup-imp-group*)
have *normal-series* (*G*⦇*carrier* := *H*⦈) [{**1**}, *H*] **using** *groupH.trivial-normal-series* **by** *auto*
with *HG* **show** *?thesis* **unfolding** *H-def* **by** (*metis append-Cons append-Nil normal-series-extend*)
qed

The following defines the list of all quotient groups of the normal series:

definition (**in** *normal-series*) *quotients*
where *quotients* = *map* (λ*i*. *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*) [*0*..<((*length* 𝔊) − *1*)]

The list of quotient groups has one less entry than the series itself:

lemma (**in** *normal-series*) *quotients-length*:
shows *length quotients* + *1* = *length* 𝔊
proof −
have *length quotients* + *1* = *length* [*0*..<((*length* 𝔊) − *1*)] + *1* **unfolding** *quotients-def* **by** *simp*
also have ... = (*length* 𝔊 − *1*) + *1* **by** (*metis diff-zero length-upt*)
also with *notempty* **have** ... = *length* 𝔊
by (*simp add*: *ac-simps*)
finally show *?thesis* **.**
qed

lemma (**in** *normal-series*) *last-quotient*:

assumes *length* $\mathfrak{G}$ > *1*
shows *last quotients* = *G Mod* $\mathfrak{G}$! (*length* $\mathfrak{G}$ − *1* − *1*)
proof −
from *assms* **have** *lsimp*: *length* $\mathfrak{G}$ − *1* − *1* + *1* = *length* $\mathfrak{G}$ − *1* **by** *auto*
from *assms* **have** *quotients* ≠ [] **unfolding** *quotients-def* **by** *auto*
hence *last quotients* = *quotients* ! (*length quotients* − *1*) **by** (*metis last-conv-nth*)
also have ... = *quotients* ! (*length* $\mathfrak{G}$ − *1* − *1*) **by** (*metis add-diff-cancel-left′ quotients-length add.commute*)
also have ... = *G*(|*carrier* := $\mathfrak{G}$! ((*length* $\mathfrak{G}$ − *1* − *1*) + *1*)|) *Mod* $\mathfrak{G}$! (*length* $\mathfrak{G}$ − *1* − *1*)
unfolding *quotients-def* **using** *assms* **by** *auto*
also have ... = *G*(|*carrier* := $\mathfrak{G}$! (*length* $\mathfrak{G}$ − *1*)|) *Mod* $\mathfrak{G}$! (*length* $\mathfrak{G}$ − *1* − *1*) **using** *lsimp* **by** *simp*
also have ... = *G Mod* $\mathfrak{G}$! (*length* $\mathfrak{G}$ − *1* − *1*) **using** *last last-conv-nth notempty* **by** *force*
finally show *?thesis* **.**
qed

The next lemma transports the constituting properties of a normal series along an isomorphism of groups.

lemma (**in** *normal-series*) *normal-series-iso*:
assumes *H*: *group H*
assumes *iso*: Ψ ∈ *iso G H*
shows *normal-series H* (*map* (*image* Ψ) $\mathfrak{G}$)
apply (*simp add*: *normal-series-def normal-series-axioms-def*)
using *H notempty* **apply** *simp*
proof (*rule conjI*)
from *H is-group iso* **have** *group-hom*: *group-hom G H* Ψ **unfolding** *group-hom-def group-hom-axioms-def iso-def* **by** *auto*
have *hd* (*map* (*image* Ψ) $\mathfrak{G}$) = Ψ ‘ {**1**} **by** (*metis hd-map hd notempty*)
also have ... = {Ψ **1**} **by** (*metis image-empty image-insert*)
also have ... = $\{\mathbf{1}_H\}$ **using** *group-hom group-hom.hom-one* **by** *auto*
finally show *hd* (*map* ((‘) Ψ) $\mathfrak{G}$) = $\{\mathbf{1}_H\}$**.**
next
show *last* (*map* ((‘) Ψ) $\mathfrak{G}$) = *carrier H* ∧ (∀ *i*. *Suc i* < *length* $\mathfrak{G}$ ⟶ Ψ ‘ $\mathfrak{G}$! *i* ◁ *H*(|*carrier* := Ψ ‘ $\mathfrak{G}$! *Suc i*|))
proof (*auto del*: *equalityI*)
have *last* (*map* ((‘) Ψ) $\mathfrak{G}$) = Ψ ‘ (*carrier G*) **using** *last last-map notempty* **by** *metis*
also have ... = *carrier H* **using** *iso* **unfolding** *iso-def bij-betw-def* **by** *simp*
finally show *last* (*map* ((‘) Ψ) $\mathfrak{G}$) = *carrier H***.**
next
fix *i*
assume *i*: *Suc i* < *length* $\mathfrak{G}$
hence *norm*: $\mathfrak{G}$! *i* ◁ *G*(|*carrier* := $\mathfrak{G}$! *Suc i*|) **using** *normal* **by** *simp*
moreover have *restrict* Ψ ($\mathfrak{G}$! *Suc i*) ∈ *iso* (*G*(|*carrier* := $\mathfrak{G}$! *Suc i*|)) (*H*(|*carrier* := Ψ ‘ $\mathfrak{G}$! *Suc i*|))
by (*metis H i is-group iso iso-restrict normal-series-subgroups*)
moreover have *group* (*G*(|*carrier* := $\mathfrak{G}$! *Suc i*|)) **by** (*metis i normal-series-subgroups*

subgroup-imp-group)
moreover hence *subgroup* ($\mathfrak{G}$! *Suc i*) *G* **by** (*metis i normal-series-subgroups*)
hence *subgroup* (Ψ ‘ $\mathfrak{G}$! *Suc i*) *H*
by (*simp add*: *H iso subgroup.iso-subgroup*)
hence *group* (*H*(|*carrier* := Ψ ‘ $\mathfrak{G}$! *Suc i*|)) **by** (*metis H subgroup.subgroup-is-group*)
ultimately have *restrict* Ψ ($\mathfrak{G}$! *Suc i*) ‘ $\mathfrak{G}$! *i* $\lhd$ *H*(|*carrier* := Ψ ‘ $\mathfrak{G}$! *Suc i*|)
using *is-group H iso-normal-subgroup* **by** (*auto cong del*: *image-cong-simp*)
moreover from *norm* **have** $\mathfrak{G}$! *i* $\subseteq$ $\mathfrak{G}$! *Suc i* **unfolding** *normal-def subgroup-def* **by** *auto*
hence $\{y.\ \exists x\in\mathfrak{G}\ !\ i.\ y = (\textit{if } x \in \mathfrak{G}\ !\ \textit{Suc } i \textit{ then } \Psi\ x \textit{ else undefined})\} = \{y.\ \exists x\in\mathfrak{G}\ !\ i.\ y = \Psi\ x\}$ **by** *auto*
ultimately show Ψ ‘ $\mathfrak{G}$! *i* $\lhd$ *H*(|*carrier* := Ψ ‘ $\mathfrak{G}$! *Suc i*|) **unfolding** *restrict-def image-def* **by** *auto*
qed
qed

2.3 Composition Series

A composition series is a normal series where all consecutive factor groups are simple:

locale *composition-series* = *normal-series* +
assumes *simplefact*: $\bigwedge i.\ i + 1 < \textit{length}\ \mathfrak{G} \Longrightarrow$ *simple-group* (*G*(|*carrier* := $\mathfrak{G}$! (*i* + *1*)|) *Mod* $\mathfrak{G}$! *i*)

lemma (**in** *composition-series*) *is-composition-series*:
shows *composition-series G* $\mathfrak{G}$
by (*rule composition-series-axioms*)

A composition series for a group G has length one if and only if G is the trivial group.

lemma (**in** *composition-series*) *composition-series-length-one*:
shows (*length* $\mathfrak{G}$ = *1*) = ($\mathfrak{G}$ = [{**1**}])
proof
assume *length* $\mathfrak{G}$ = *1*
with *hd* **have** *length* $\mathfrak{G}$ = *length* [{**1**}] $\wedge$ ($\forall$ *i* < *length* $\mathfrak{G}$. $\mathfrak{G}$! *i* = [{**1**}] ! *i*) **using** *hd-conv-nth notempty* **by** *force*
thus $\mathfrak{G}$ = [{**1**}] **using** *list-eq-iff-nth-eq* **by** *blast*
next
assume $\mathfrak{G}$ = [{**1**}]
thus *length* $\mathfrak{G}$ = *1* **by** *simp*
qed

lemma (**in** *composition-series*) *composition-series-triv-group*:
shows (*carrier G* = {**1**}) = ($\mathfrak{G}$ = [{**1**}])
proof
assume *G*: *carrier G* = {**1**}
have *length* $\mathfrak{G}$ = *1*
proof (*rule ccontr*)

assume *length* $\mathfrak{G} \neq 1$
with *notempty* **have** *length*: *length* $\mathfrak{G} \geq 2$ **by** (*metis Suc-eq-plus1 length-0-conv less-2-cases not-less plus-nat.add-0*)
with *simplefact hd hd-conv-nth notempty* **have** *simple-group* ($G(|carrier := \mathfrak{G}$! *1*|) *Mod* {**1**}) **by** *force*
moreover have *SG*: *subgroup* ($\mathfrak{G}$! *1*) *G* **using** *length normal-series-subgroups* **by** *auto*
hence *group* ($G(|carrier := \mathfrak{G}$! *1*|)) **by** (*metis subgroup-imp-group*)
ultimately have *simple-group* ($G(|carrier := \mathfrak{G}$! *1*|)) **using** *group.trivial-factor-iso simple-group.iso-simple* **by** *fastforce*
moreover from *SG G* **have** *carrier* ($G(|carrier := \mathfrak{G}$! *1*|)) = {**1**} **unfolding** *subgroup-def* **by** *auto*
ultimately show *False* **using** *simple-group.simple-not-triv* **by** *force*
qed
thus $\mathfrak{G}$ = [{**1**}] **by** (*metis composition-series-length-one*)
next
assume $\mathfrak{G}$ = [{**1**}]
with *last* **show** *carrier G* = {**1**} **by** *auto*
qed

The inner elements of a composition series may not consist of the trivial subgroup or the group itself.

lemma (**in** *composition-series*) *inner-elements-not-triv*:
assumes $i + 1 <$ *length* $\mathfrak{G}$
assumes $i > 0$
shows $\mathfrak{G}$! $i \neq$ {**1**}
proof
from *assms* **have** $(i - 1) + 1 <$ *length* $\mathfrak{G}$ **by** *simp*
hence *simple*: *simple-group* ($G(|carrier := \mathfrak{G}$! $((i - 1) + 1)$|) *Mod* $\mathfrak{G}$! $(i - 1)$) **using** *simplefact* **by** *auto*
assume *i*: $\mathfrak{G}$! i = {**1**}
moreover from *assms* **have** $(i - 1) + 1 = i$ **by** *auto*
ultimately have $G(|carrier := \mathfrak{G}$! $((i - 1) + 1)$|) *Mod* $\mathfrak{G}$! $(i - 1)$ = $G(|carrier$:= {**1**}|) *Mod* $\mathfrak{G}$! $(i - 1)$ **using** *i* **by** *auto*
hence *order* ($G(|carrier := \mathfrak{G}$! $((i - 1) + 1)$|) *Mod* $\mathfrak{G}$! $(i - 1)$) = *1* **unfolding** *FactGroup-def order-def RCOSETS-def* **by** *force*
thus *False* **using** *i simple* **unfolding** *simple-group-def simple-group-axioms-def* **by** *auto*
qed

A composition series of a simple group always is its trivial one.

lemma (**in** *composition-series*) *composition-series-simple-group*:
shows (*simple-group G*) = ($\mathfrak{G}$ = [{**1**}, *carrier G*])
proof
assume $\mathfrak{G}$ = [{**1**}, *carrier G*]
with *simplefact* **have** *simple-group* (*G Mod* {**1**}) **by** *auto*
moreover have *the-elem* $\in$ *iso* (*G Mod* {**1**}) *G* **by** (*rule trivial-factor-iso*)
ultimately show *simple-group G* **by** (*metis is-group simple-group.iso-simple*)
next

assume *simple*: *simple-group G*
have *length* 𝔊 > *1*
proof (*rule ccontr*)
assume ¬ *1* < *length* 𝔊
hence *length* 𝔊 = *1* **by** (*simp add*: *Suc-leI antisym notempty*)
hence *carrier G* = {**1**} **using** *hd last* **by** (*metis composition-series-length-one composition-series-triv-group*)
hence *order G* = *1* **unfolding** *order-def* **by** *auto*
with *simple* **show** *False* **unfolding** *simple-group-def simple-group-axioms-def* **by** *auto*
qed
moreover have *length* 𝔊 ≤ *2*
proof (*rule ccontr*)
define *k* **where** *k* = *length* 𝔊 − *2*
assume ¬ (*length* 𝔊 ≤ *2*)
hence *gt2*: *length* 𝔊 > *2* **by** *simp*
hence *ksmall*: *k* + *1* < *length* 𝔊 **unfolding** *k-def* **by** *auto*
from *gt2* **have** *carrier*: 𝔊 ! (*k* + *1*) = *carrier G* **using** *notempty last last-conv-nth k-def*
by (*metis Nat.add-diff-assoc Nat.diff-cancel* ‹¬ *length* 𝔊 ≤ *2*› *add.commute nat-le-linear one-add-one*)
from *normal ksmall* **have** 𝔊 ! *k* ⊲ *G*(| *carrier* := 𝔊 ! (*k* + *1*)|) **by** *simp*
from *simplefact ksmall* **have** *simplek*: *simple-group* (*G*(|*carrier* := 𝔊 ! (*k* + *1*)|) *Mod* 𝔊 ! *k*) **by** *simp*
from *simplefact ksmall* **have** *simplek'*: *simple-group* (*G*(|*carrier* := 𝔊 ! ((*k* − *1*) + *1*)|) *Mod* 𝔊 ! (*k* − *1*)) **by** *auto*
have 𝔊 ! *k* ⊲ *G* **using** *carrier k-def gt2 normal ksmall* **by** *force*
with *simple* **have** (𝔊 ! *k*) = *carrier G* ∨ (𝔊 ! *k*) = {**1**} **unfolding** *simple-group-def simple-group-axioms-def* **by** *simp*
thus *False*
proof (*rule disjE*)
assume 𝔊 ! *k* = *carrier G*
hence *G*(|*carrier* := 𝔊 ! (*k* + *1*)|) *Mod* 𝔊 ! *k* = *G Mod* (*carrier G*) **using** *carrier* **by** *auto*
with *simplek self-factor-not-simple* **show** *False* **by** *auto*
next
assume 𝔊 ! *k* = {**1**}
with *ksmall k-def gt2* **show** *False* **using** *inner-elements-not-triv* **by** *auto*
qed
qed
ultimately have *length* 𝔊 = *2* **by** *simp*
thus 𝔊 = [{**1**}, *carrier G*] **by** (*rule length-two-unique*)
qed

Two consecutive elements in a composition series are distinct.

lemma (**in** *composition-series*) *entries-distinct*:
assumes *finite*: *finite* (*carrier G*)
assumes *i*: *i* + *1* < *length* 𝔊
shows 𝔊 ! *i* ≠ 𝔊 ! (*i* + *1*)

proof
  from *finite* **have** *finite* (($\mathfrak{G}$! (*i* + *1*))
    using *i normal-series-subgroups subgroup.subset rev-finite-subset* **by** *metis*
  hence *fin*: *finite* (*carrier* (*G*⦇*carrier* := $\mathfrak{G}$! (*i* + *1*)⦈)) **by** *auto*
  from *i* **have** *norm*: $\mathfrak{G}$! *i* ◁ (*G*⦇*carrier* := $\mathfrak{G}$! (*i* + *1*)⦈) **by** (*rule normal*)
  assume $\mathfrak{G}$! *i* = $\mathfrak{G}$! (*i* + *1*)
  hence $\mathfrak{G}$! *i* = *carrier* (*G*⦇*carrier* := $\mathfrak{G}$! (*i* + *1*)⦈) **by** *auto*
  hence *carrier* ((*G*⦇*carrier* := ($\mathfrak{G}$! (*i* + *1*))⦈) *Mod* ($\mathfrak{G}$! *i*)) = {**1**(*G*⦇*carrier* := $\mathfrak{G}$! (*i* + *1*)⦈) *Mod* $\mathfrak{G}$! *i*}
    using *norm fin normal.fact-group-trivial-iff* **by** *metis*
  hence ¬ *simple-group* ((*G*⦇*carrier* := ($\mathfrak{G}$! (*i* + *1*))⦈) *Mod* ($\mathfrak{G}$! *i*)) **by** (*metis simple-group.simple-not-triv*)
  thus *False* **by** (*metis i simplefact*)
qed

The normal series for groups of order $p * q$ is even a composition series:

lemma (**in** *group*) *pq-order-composition-series*:
  assumes *finite*: *finite* (*carrier G*)
  assumes *orderG*: *order G* = *q* ∗ *p*
  assumes *primep*: *prime p* **and** *primeq*: *prime q* **and** *pq*: *p* < *q*
  shows *composition-series G* [{**1**}, (*THE H*. *H* ∈ *subgroups-of-size q*), *carrier G*]
unfolding *composition-series-def composition-series-axioms-def*
apply(*auto*)
using *assms* **apply**(*rule pq-order-normal-series*)
proof −
  define *H* **where** *H* = (*THE H*. *H* ∈ *subgroups-of-size q*)
  from *assms* **have** *exi*: ∃!*Q*. *Q* ∈ (*subgroups-of-size q*) **by** (*auto simp*: *pq-order-unique-subgrp*)
  hence *Hsize*: *H* ∈ *subgroups-of-size q* **unfolding** *H-def* **using** *theI′* **by** *metis*
  hence *HsubG*: *subgroup H G* **unfolding** *subgroups-of-size-def* **by** *auto*
  then interpret *Hgroup*: *group G*⦇*carrier* := *H*⦈ **by** (*metis subgroup-imp-group*)
  fix *i*
  assume *i* < *Suc* (*Suc 0*)
  hence *i* = *0* ∨ *i* = *1* **by** *auto*
  thus *simple-group* (*G*⦇*carrier* := [*H*, *carrier G*] ! *i*⦈ *Mod* [{**1**}, *H*, *carrier G*] ! *i*)
  proof
    assume *i*: *i* = *0*
      from *Hsize* **have** *orderH*: *order* (*G*⦇*carrier* := *H*⦈) = *q* **unfolding** *subgroups-of-size-def order-def* **by** *simp*
    hence *order-eq-q*: *order* (*G*⦇*carrier* := *H*⦈ *Mod* {**1**}) = *q*
      using *Hgroup.trivial-factor-iso iso-same-order* **by** *auto*
    have *normal* {**1**} (*G*⦇*carrier* := *H*⦈)
      by (*simp add*: *HsubG group.normal-restrict-supergroup subgroup.one-closed trivial-subgroup-is-normal*)
    hence *group* (*G*⦇*carrier* := *H*⦈ *Mod* {**1**}) **by** (*metis normal.factorgroup-is-group*)
    with *orderH primeq* **have** *simple-group* (*G*⦇*carrier* := *H*⦈ *Mod* {**1**})
      by (*metis order-eq-q group.prime-order-simple*)
    with *i* **show** *?thesis* **by** *simp*
  next
    assume *i*: *i* = *1*
    from *assms exi* **have** *H* ◁ *G* **unfolding** *H-def* **by** (*metis pq-order-subgrp-normal*)

hence *groupGH*: *group* (*G Mod H*) **by** (*metis normal.factorgroup-is-group*)
from *primeq* **have** $q \neq 0$ **by** (*metis not-prime-0*)
from *HsubG finite orderG* **have** *card* (*rcosets H*) ∗ *card H* = *q* ∗ *p* **unfolding** *subgroups-of-size-def* **using** *lagrange* **by** *simp*
with *Hsize* **have** *card* (*rcosets H*) ∗ *q* = *q* ∗ *p* **unfolding** *subgroups-of-size-def* **by** *simp*
with ‹$q \neq 0$› **have** *card* (*rcosets H*) = *p* **by** *auto*
hence *order* (*G Mod H*) = *p* **unfolding** *order-def FactGroup-def* **by** *auto*
with *groupGH primep* **have** *simple-group* (*G Mod H*) **by** (*metis group.prime-order-simple*)
with *i* **show** *?thesis* **by** *auto*
qed
qed

Prefixes of composition series are also composition series.

lemma (**in** *composition-series*) *composition-series-prefix-closed*:
assumes *i* ≤ *length* 𝔊 **and** *0* < *i*
shows *composition-series* (*G*⦇*carrier* := 𝔊 ! (*i* − *1*)⦈) (*take i* 𝔊)
unfolding *composition-series-def composition-series-axioms-def*
proof *auto*
from *assms* **show** *normal-series* (*G*⦇*carrier* := 𝔊 ! (*i* − *Suc 0*)⦈) (*take i* 𝔊) **by** (*metis One-nat-def normal-series-prefix-closed*)
next
fix *j*
assume *j*: *Suc j* < *length* 𝔊 *Suc j* < *i*
with *simplefact* **show** *simple-group* (*G*⦇*carrier* := 𝔊 ! *Suc j*⦈ *Mod* 𝔊 ! *j*) **by** (*metis Suc-eq-plus1*)
qed

The second element in a composition series is simple group.

lemma (**in** *composition-series*) *composition-series-snd-simple*:
assumes *2* ≤ *length* 𝔊
shows *simple-group* (*G*⦇*carrier* := 𝔊 ! *1*⦈)
proof −
from *assms* **interpret** *compTake*: *composition-series G*⦇*carrier* := 𝔊 ! *1*⦈ *take 2* 𝔊 **by** (*metis add-diff-cancel-right′ composition-series-prefix-closed one-add-one zero-less-numeral*)
from *assms* **have** *length* (*take 2* 𝔊) = *2* **by** (*metis add-diff-cancel-right′ append-take-drop-id diff-diff-cancel length-append length-drop*)
hence (*take 2* 𝔊) = [{$\mathbf{1}_{(G⦇carrier := 𝔊 ! 1⦈)}$}, *carrier* (*G*⦇*carrier* := 𝔊 ! *1*⦈)] **by** (*rule compTake.length-two-unique*)
thus *?thesis* **by** (*metis compTake.composition-series-simple-group*)
qed

As a stronger way to state the previous lemma: An entry of a composition series is simple if and only if it is the second one.

lemma (**in** *composition-series*) *composition-snd-simple-iff*:
assumes *i* < *length* 𝔊
shows (*simple-group* (*G*⦇*carrier* := 𝔊 ! *i*⦈)) = (*i* = *1*)

proof
assume *simpi*: *simple-group* (*G*(|*carrier* := 𝔊 ! *i*|))
hence 𝔊 ! *i* ≠ {**1**} **using** *simple-group.simple-not-triv* **by** *force*
hence *i* ≠ *0* **using** *hd hd-conv-nth notempty* **by** *auto*
then interpret *compTake*: *composition-series* *G*(|*carrier* := 𝔊 ! *i*|) *take* (*Suc i*) 𝔊
using *assms composition-series-prefix-closed* **by** (*metis diff-Suc-1 less-eq-Suc-le zero-less-Suc*)
from *simpi* **have** (*take* (*Suc i*) 𝔊) = [{$\mathbf{1}_{G(|carrier\ :=\ 𝔊\ !\ i|)}$}, *carrier* (*G*(|*carrier* := 𝔊 ! *i*|))]
by (*metis compTake.composition-series-simple-group*)
hence *length* (*take* (*Suc i*) 𝔊) = *2* **by** *auto*
hence *min* (*length* 𝔊) (*Suc i*) = *2* **by** (*metis length-take*)
with *assms* **have** *Suc i* = *2* **by** *force*
thus *i* = *1* **by** *simp*
next
assume *i*: *i* = *1*
with *assms* **have** *2* ≤ *length* 𝔊 **by** *simp*
with *i* **show** *simple-group* (*G*(|*carrier* := 𝔊 ! *i*|)) **by** (*metis composition-series-snd-simple*)
qed

The second to last entry of a normal series is not only a normal subgroup but actually even a *maximal* normal subgroup.

lemma (**in** *composition-series*) *snd-to-last-max-normal*:
assumes *finite*: *finite* (*carrier G*)
assumes *length*: *length* 𝔊 > *1*
shows *max-normal-subgroup* (𝔊 ! (*length* 𝔊 − *2*)) *G*
unfolding *max-normal-subgroup-def max-normal-subgroup-axioms-def*
proof (*auto del*: *equalityI*)
show 𝔊 ! (*length* 𝔊 − *2*) ⊲ *G* **by** (*rule normal-series-snd-to-last*)
next
define *G′* **where** *G′* = 𝔊 ! (*length* 𝔊 − *2*)
from *length* **have** *length21*: *length* 𝔊 − *2* + *1* = *length* 𝔊 − *1* **by** *arith*
from *length* **have** *length* 𝔊 − *2* + *1* < *length* 𝔊 **by** *arith*
with *simplefact* **have** *simple-group* (*G*(|*carrier* := 𝔊 ! ((*length* 𝔊 − *2*) + *1*)|) *Mod G′*) **unfolding** *G′-def* **by** *auto*
with *length21* **have** *simple-last*: *simple-group* (*G Mod G′*) **using** *last notempty last-conv-nth* **by** *fastforce*
{
assume *snd-to-last-eq*: *G′* = *carrier G*
hence *carrier* (*G Mod G′*) = {$\mathbf{1}_{G\ Mod\ G'}$}
using *normal-series-snd-to-last finite normal.fact-group-trivial-iff* **unfolding** *G′-def* **by** *metis*
with *snd-to-last-eq* **have** ¬ *simple-group* (*G Mod G′*) **by** (*metis self-factor-not-simple*)
with *simple-last* **show** *False* **unfolding** *G′-def* **by** *auto*
}
{
have *G′G*: *G′* ⊲ *G* **unfolding** *G′-def* **by** (*rule normal-series-snd-to-last*)
fix *J*

assume *J*: $J \lhd G$ $J \neq G'$ $J \neq carrier\ G$ $G' \subseteq J$
hence *JG'GG'*: $rcosets_{(G(\!|carrier := J|\!))}\ G' \lhd G\ Mod\ G'$ **using** *normality-factorization normal-series-snd-to-last* **unfolding** *G'-def* **by** *auto*
from *G'G J(1,4)* **have** *G'J*: $G' \lhd (G(\!|carrier := J|\!))$ **by** (*metis normal-imp-subgroup normal-restrict-supergroup*)
from *finite J(1)* **have** *finJ*: *finite J* **by** (*auto simp*: *normal-imp-subgroup subgroup-finite*)
from *JG'GG' simple-last* **have** $rcosets_{G(\!|carrier := J|\!)}\ G' = \{\mathbf{1}_{G\ Mod\ G'}\} \vee rcosets_{G(\!|carrier := J|\!)}\ G' = carrier\ (G\ Mod\ G')$
unfolding *simple-group-def simple-group-axioms-def* **by** *auto*
thus *False*
proof
assume $rcosets_{G(\!|carrier := J|\!)}\ G' = \{\mathbf{1}_{G\ Mod\ G'}\}$
hence $rcosets_{G(\!|carrier := J|\!)}\ G' = \{\mathbf{1}_{(G(\!|carrier := J|\!))\ Mod\ G'}\}$ **unfolding** *FactGroup-def* **by** *simp*
hence $G' = J$ **using** *G'J finJ normal.fact-group-trivial-iff* **unfolding** *FactGroup-def* **by** *fastforce*
with *J(2)* **show** *False* **by** *simp*
next
assume *facts-eq*: $rcosets_{G(\!|carrier := J|\!)}\ G' = carrier\ (G\ Mod\ G')$
have $J = carrier\ G$
proof
show $J \subseteq carrier\ G$ **using** *J(1) normal-imp-subgroup subgroup.subset* **by** *force*
next
show $carrier\ G \subseteq J$
proof
fix *x*
assume *x*: $x \in carrier\ G$
hence $G'\ \#>\ x \in carrier\ (G\ Mod\ G')$ **unfolding** *FactGroup-def RCOSETS-def* **by** *auto*
hence $G'\ \#>\ x \in rcosets_{G(\!|carrier := J|\!)}\ G'$ **using** *facts-eq* **by** *auto*
then obtain *j* **where** *j*: $j \in J$ $G'\ \#>\ x = G'\ \#>\ j$ **unfolding** *RCOSETS-def r-coset-def* **by** *force*
hence $x \in G'\ \#>\ j$ **using** *G'G normal-imp-subgroup x repr-independenceD* **by** *fastforce*
then obtain *g'* **where** *g'*: $g' \in G'$ $x = g' \otimes j$ **unfolding** *r-coset-def* **by** *auto*
hence $g' \in J$ **using** *G'J normal-imp-subgroup subgroup.subset* **by** *force*
with *g'(2) j(1)* **show** $x \in J$ **using** *J(1) normal-imp-subgroup subgroup.m-closed* **by** *fastforce*
qed
qed
with *J(3)* **show** *False* **by** *simp*
qed
}
qed

For the next lemma we need a few facts about removing adjacent duplicates.

lemma *remdups-adj-obtain-adjacency*:
  assumes *i* + *1* < *length* (*remdups-adj xs*) *length xs* > *0*
  obtains *j* **where** *j* + *1* < *length xs*
    (*remdups-adj xs*) ! *i* = *xs* ! *j* (*remdups-adj xs*) ! (*i* + *1*) = *xs* ! (*j* + *1*)
using *assms* **proof** (*induction xs arbitrary*: *i thesis*)
  case *Nil*
  hence *False* **by** (*metis length-greater-0-conv*)
  thus *thesis*..
next
  case (*Cons x xs*)
  then have *xs* ≠ []
    by *auto*
  then obtain *y xs′* **where** *xs*: *xs* = *y* # *xs′*
    by (*cases xs*) *blast*
  from ‹*xs* ≠ []› **have** *lenxs*: *length xs* > *0* **by** *simp*
  from *xs* **have** *rem*: *remdups-adj* (*x* # *xs*) = (*if x* = *y then remdups-adj* (*y* # *xs′*) *else x* # *remdups-adj* (*y* # *xs′*)) **using** *remdups-adj.simps*(*3*) **by** *auto*
  show *thesis*
  proof (*cases x* = *y*)
    case *True*
    with *rem xs* **have** *rem2*: *remdups-adj* (*x* # *xs*) = *remdups-adj xs* **by** *auto*
    with *Cons*(*3*) **have** *i* + *1* < *length* (*remdups-adj xs*) **by** *simp*
    with *Cons.IH lenxs* **obtain** *k* **where** *j*: *k* + *1* < *length xs remdups-adj xs* ! *i* = *xs* ! *k*
        remdups-adj xs ! (*i* + *1*) = *xs* ! (*k* + *1*) **by** *auto*
    thus *thesis* **using** *Cons*(*2*) *rem2* **by** *auto*
  next
    case *False*
    with *rem xs* **have** *rem2*: *remdups-adj* (*x* # *xs*) = *x* # *remdups-adj xs* **by** *auto*
    show *thesis*
    proof (*cases i*)
      case *0*
      have *0* + *1* < *length* (*x* # *xs*) **using** *lenxs* **by** *auto*
      moreover have *remdups-adj* (*x* # *xs*) ! *i* = (*x* # *xs*) ! *0*
      proof −
        have *remdups-adj* (*x* # *xs*) ! *i* = (*x* # *remdups-adj* (*y* # *xs′*)) ! *0* **using** *xs rem2 0* **by** *simp*
        also have ... = *x* **by** *simp*
        also have ... = (*x* # *xs*) ! *0* **by** *simp*
        finally show *?thesis*.
      qed
      moreover have *remdups-adj* (*x* # *xs*) ! (*i* + *1*) = (*x* # *xs*) ! (*0* + *1*)
      proof −
        have *remdups-adj* (*x* # *xs*) ! (*i* + *1*) = (*x* # *remdups-adj* (*y* # *xs′*)) ! *1* **using** *xs rem2 0* **by** *simp*
        also have ... = *remdups-adj* (*y* # *xs′*) ! *0* **by** *simp*
        also have ... = (*y* # (*remdups* (*y* # *xs′*))) ! *0* **by** (*metis nth-Cons′ remdups-adj-Cons-alt*)

```
      also have ... = y by simp
      also have ... = (x # xs) ! (0 + 1) unfolding xs by simp
      finally show ?thesis.
    qed
    ultimately show thesis by (rule Cons.prems(1))
  next
    case (Suc k)
    with Cons(3) have k + 1 < length (remdups-adj (x # xs)) − 1 by auto
     also have ... ≤ length (remdups-adj xs) + 1 − 1 by (metis One-nat-def
le-refl list.size(4) rem2)
    also have ... = length (remdups-adj xs) by simp
    finally have k + 1 < length (remdups-adj xs).
    with Cons.IH lenxs obtain j where j: j + 1 < length xs remdups-adj xs ! k
= xs ! j
      remdups-adj xs ! (k + 1) = xs ! (j + 1) by auto
    from j(1) have Suc j + 1 < length (x # xs) by simp
    moreover have remdups-adj (x # xs) ! i = (x # xs) ! (Suc j)
    proof −
      have remdups-adj (x # xs) ! i = (x # remdups-adj xs) ! i using rem2 by
simp
      also have ... = (remdups-adj xs) ! k using Suc by simp
      also have ... = xs ! j using j(2) .
      also have ... = (x # xs) ! (Suc j) by simp
      finally show ?thesis .
    qed
    moreover have remdups-adj (x # xs) ! (i + 1) = (x # xs) ! (Suc j + 1)
    proof −
       have remdups-adj (x # xs) ! (i + 1) = (x # remdups-adj xs) ! (i + 1)
using rem2 by simp
      also have ... = (remdups-adj xs) ! (k + 1) using Suc by simp
      also have ... = xs ! (j + 1) using j(3).
      also have ... = (x # xs) ! (Suc j + 1) by simp
      finally show ?thesis.
    qed
    ultimately show thesis by (rule Cons.prems(1))
  qed
 qed
qed

lemma hd-remdups-adj[simp]: hd (remdups-adj xs) = hd xs
 by (induction xs rule: remdups-adj.induct) simp-all

lemma remdups-adj-adjacent:
 Suc i < length (remdups-adj xs) ⟹ remdups-adj xs ! i ≠ remdups-adj xs ! Suc i
proof (induction xs arbitrary: i rule: remdups-adj.induct)
 case (3 x y xs i)
 thus ?case by (cases i, cases x = y) (simp, auto simp: hd-conv-nth[symmetric])
qed simp-all
```

Intersecting each entry of a composition series with a normal subgroup of G

and removing all adjacent duplicates yields another composition series.

lemma (**in** *composition-series*) *intersect-normal*:
assumes *finite*: *finite* (*carrier G*)
assumes *KG*: $K \lhd G$
shows *composition-series* ($G(\!|$*carrier* := $K|\!)$) (*remdups-adj* (*map* ($\lambda H.\ K \cap H$) $\mathfrak{G}$))
unfolding *composition-series-def composition-series-axioms-def normal-series-def normal-series-axioms-def*
apply (*auto simp only*: *conjI del*: *equalityI*)
proof −
show *group* ($G(\!|$*carrier* := $K|\!)$) **using** *KG normal-imp-subgroup subgroup-imp-group* **by** *auto*
next
— Show, that removing adjacent duplicates doesn't result in an empty list.
assume *remdups-adj* (*map* ((∩) *K*) 𝔊) = []
hence *map* ((∩) *K*) 𝔊 = [] **by** (*metis remdups-adj-Nil-iff*)
hence 𝔊 = [] **by** (*metis Nil-is-map-conv*)
with *notempty* **show** *False*..
next
— Show, that the head of the reduced list is still the trivial group
have 𝔊 = {**1**} # *tl* 𝔊 **using** *notempty hd* **by** (*metis list.sel*(*1*,*3*) *neq-Nil-conv*)
hence *map* ((∩) *K*) 𝔊 = *map* ((∩) *K*) ({**1**} # *tl* 𝔊) **by** *simp*
hence *remdups-adj* (*map* ((∩) *K*) 𝔊) = *remdups-adj* ((*K* ∩ {**1**}) # (*map* ((∩) *K*) (*tl* 𝔊))) **by** *simp*
also have ... = (*K* ∩ {**1**}) # *tl* (*remdups-adj* ((*K* ∩ {**1**}) # (*map* ((∩) *K*) (*tl* 𝔊)))) **by** *simp*
finally have *hd* (*remdups-adj* (*map* ((∩) *K*) 𝔊)) = *K* ∩ {**1**} **using** *list.sel*(*1*) **by** *metis*
thus *hd* (*remdups-adj* (*map* ((∩) *K*) 𝔊)) = $\{\mathbf{1}_{G(\!|carrier\ :=\ K|\!)}\}$
using *KG normal-imp-subgroup subgroup.one-closed* **by** *force*
next
— Show that the last entry is really *K* ∩ *G*. Since we don't have a lemma ready to talk about the last entry of a reduced list, we reverse the list twice.
have *rev* 𝔊 = (*carrier G*) # *tl* (*rev* 𝔊) **by** (*metis list.sel*(*1*,*3*) *last last-rev neq-Nil-conv notempty rev-is-Nil-conv rev-rev-ident*)
hence *rev* (*map* ((∩) *K*) 𝔊) = *map* ((∩) *K*) ((*carrier G*) # *tl* (*rev* 𝔊)) **by** (*metis rev-map*)
hence *rev*: *rev* (*map* ((∩) *K*) 𝔊) = (*K* ∩ (*carrier G*)) # (*map* ((∩) *K*) (*tl* (*rev* 𝔊))) **by** *simp*
have *last* (*remdups-adj* (*map* ((∩) *K*) 𝔊)) = *hd* (*rev* (*remdups-adj* (*map* ((∩) *K*) 𝔊)))
by (*metis hd-rev map-is-Nil-conv notempty remdups-adj-Nil-iff*)
also have ... = *hd* (*remdups-adj* (*rev* (*map* ((∩) *K*) 𝔊))) **by** (*metis remdups-adj-rev*)
also have ... = *hd* (*remdups-adj* ((*K* ∩ (*carrier G*)) # (*map* ((∩) *K*) (*tl* (*rev* 𝔊))))) **by** (*metis rev*)
also have ... = *hd* ((*K* ∩ (*carrier G*)) # (*remdups-adj* ((*K* ∩ (*carrier G*)) # (*map* ((∩) *K*) (*tl* (*rev* 𝔊)))))) **by** (*metis list.sel*(*1*) *remdups-adj-Cons-alt*)
also have ... = *K* **using** *KG normal-imp-subgroup subgroup.subset* **by** *force*
finally show *last* (*remdups-adj* (*map* ((∩) *K*) 𝔊)) = *carrier* ($G(\!|$*carrier* := $K|\!)$)

by *auto*
next
— The induction step, using the second isomorphism theorem for groups.
fix *j*
assume *j*: *j* + *1* < *length* (*remdups-adj* (*map* ((∩) *K*) 𝔊))
have *KGnotempty*: (*map* ((∩) *K*) 𝔊) ≠ [] **using** *notempty* **by** (*metis Nil-is-map-conv*)
with *j* **obtain** *i* **where** *i*: *i* + *1* < *length* (*map* ((∩) *K*) 𝔊)
(*remdups-adj* (*map* ((∩) *K*) 𝔊)) ! *j* = (*map* ((∩) *K*) 𝔊) ! *i*
(*remdups-adj* (*map* ((∩) *K*) 𝔊)) ! (*j* + *1*) = (*map* ((∩) *K*) 𝔊) ! (*i* + *1*)
using *remdups-adj-obtain-adjacency* **by** *force*
from *i*(*1*) **have** *i'*: *i* + *1* < *length* 𝔊 **by** (*metis length-map*)
hence *GiSi*: 𝔊 ! *i* ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ **by** (*metis normal*)
hence *GiSi'*: 𝔊 ! *i* ⊆ 𝔊 ! (*i* + *1*) **using** *normal-imp-subgroup subgroup.subset* **by** *force*
from *i'* **have** *finGSi*: *finite* (𝔊 ! (*i* + *1*)) **using** *normal-series-subgroups finite* **by** (*metis subgroup-finite*)
from *GiSi KG i' normal-series-subgroups* **have** *GSiKnormGSi*: 𝔊 ! (*i* + *1*) ∩ *K* ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈
using *second-isomorphism-grp.normal-subgrp-intersection-normal*
unfolding *second-isomorphism-grp-def second-isomorphism-grp-axioms-def* **by** *auto*
with *GiSi* **have** 𝔊 ! *i* ∩ (𝔊 ! (*i* + *1*) ∩ *K*) ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈
by (*metis group.normal-subgroup-intersect group.subgroup-imp-group i' is-group is-normal-series normal-series.normal-series-subgroups*)
hence *K* ∩ (𝔊 ! *i* ∩ 𝔊 ! (*i* + *1*)) ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ **by** (*metis inf-commute inf-left-commute*)
hence *KGinormGSi*: *K* ∩ 𝔊 ! *i* ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ **using** *GiSi'* **by** (*metis le-iff-inf*)
moreover have *K* ∩ 𝔊 ! *i* ⊆ *K* ∩ 𝔊 ! (*i* + *1*) **using** *GiSi'* **by** *auto*
moreover have *groupGSi*: *group* (*G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈) **using** *i normal-series-subgroups subgroup-imp-group* **by** *auto*
moreover have *subKGSiGSi*: *subgroup* (*K* ∩ 𝔊 ! (*i* + *1*)) (*G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈) **by** (*metis GSiKnormGSi inf-sup-aci*(*1*) *normal-imp-subgroup*)
ultimately have *fstgoal*: *K* ∩ 𝔊 ! *i* ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*), *carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈
using *group.normal-restrict-supergroup* **by** *force*
thus *remdups-adj* (*map* ((∩) *K*) 𝔊) ! *j* ⊲ *G*⦇*carrier* := *K*, *carrier* := *remdups-adj* (*map* ((∩) *K*) 𝔊) ! (*j* + *1*)⦈
using *i* **by** *auto*
from *simplefact* **have** *Gisimple*: *simple-group* (*G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*) **using** *i'* **by** *simp*
hence *Gimax*: *max-normal-subgroup* (𝔊 ! *i*) (*G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈)
using *normal.max-normal-simple-quotient GiSi finGSi* **by** *force*
from *GSiKnormGSi GiSi* **have** 𝔊 ! *i* <#>$_{G⦇carrier := 𝔊 ! (i + 1)⦈}$ 𝔊 ! (*i* + *1*) ∩ *K* ⊲ (*G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈)
using *groupGSi group.normal-subgroup-set-mult-closed set-mult-consistent* **by** *fastforce*
hence 𝔊 ! *i* <#> 𝔊 ! (*i* + *1*) ∩ *K* ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ **unfolding** *set-mult-def* **by** *auto*

hence 𝔊 ! *i* <#> *K* ∩ 𝔊 ! (*i* + *1*) ⊲ *G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ **using** *inf-commute* **by** *metis*
moreover have 𝔊 ! *i* ⊆ 𝔊 ! *i* <#>$_{G⦇carrier := 𝔊 ! (i + 1)⦈}$ *K* ∩ 𝔊 ! (*i* + *1*)
using *second-isomorphism-grp.H-contained-in-set-mult*
unfolding *second-isomorphism-grp-def second-isomorphism-grp-axioms-def*
using *subKGSiGSi GiSi normal-imp-subgroup* **by** *fastforce*
hence 𝔊 ! *i* ⊆ 𝔊 ! *i* <#> *K* ∩ 𝔊 ! (*i* + *1*) **unfolding** *set-mult-def* **by** *auto*
ultimately have *KGdisj*: 𝔊 ! *i* <#> *K* ∩ 𝔊 ! (*i* + *1*) = 𝔊 ! *i* ∨ 𝔊 ! *i* <#> *K* ∩ 𝔊 ! (*i* + *1*) = 𝔊 ! (*i* + *1*)
using *Gimax* **unfolding** *max-normal-subgroup-def max-normal-subgroup-axioms-def*
by *auto*
obtain φ **where** φ ∈ *iso* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (𝔊 ! *i* ∩ (*K* ∩ 𝔊 ! (*i* + *1*))))
(*G*⦇*carrier* := 𝔊 ! *i* <#>$_{G⦇carrier := 𝔊 ! (i + 1)⦈}$ *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*)
using *second-isomorphism-grp.normal-intersection-quotient-isom*
unfolding *second-isomorphism-grp-def second-isomorphism-grp-axioms-def*
using *GiSi subKGSiGSi normal-imp-subgroup* **by** *fastforce*
hence φ ∈ *iso* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! (*i* + *1*) ∩ 𝔊 ! *i*))
(*G*⦇*carrier* := 𝔊 ! *i* <#>$_{G⦇carrier := 𝔊 ! (i + 1)⦈}$ *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*)
by (*metis inf-commute*)
hence φ ∈ *iso* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ (𝔊 ! (*i* + *1*) ∩ 𝔊 ! *i*)))
(*G*⦇*carrier* := 𝔊 ! *i* <#>$_{G⦇carrier := 𝔊 ! (i + 1)⦈}$ *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*)
by (*metis Int-assoc*)
hence φ ∈ *iso* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*))
(*G*⦇*carrier* := 𝔊 ! *i* <#>$_{G⦇carrier := 𝔊 ! (i + 1)⦈}$ *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*)
by (*metis GiSi′ Int-absorb2 Int-commute*)
hence φ: φ ∈ *iso* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*))
(*G*⦇*carrier* := 𝔊 ! *i* <#> *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*)
unfolding *set-mult-def* **by** *auto*
from *fstgoal* **have** *KGsiKGigroup*: *group* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*)) **using** *normal.factorgroup-is-group* **by** *auto*
from *KGdisj* **show** *simple-group* (*G*⦇*carrier* := *K*, *carrier* := *remdups-adj* (*map* ((∩) *K*) 𝔊) ! (*j* + *1*)⦈ *Mod remdups-adj* (*map* ((∩) *K*) 𝔊) ! *j*)
proof *auto*
have *groupGi*: *group* (*G*⦇*carrier* := 𝔊 ! *i*⦈) **using** *i′ normal-series-subgroups subgroup-imp-group* **by** *auto*
assume 𝔊 ! *i* <#> *K* ∩ 𝔊 ! *Suc i* = 𝔊 ! *i*
with φ **have** φ ∈ *iso* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*)) (*G*⦇*carrier* := 𝔊 ! *i*⦈ *Mod* 𝔊 ! *i*) **by** *auto*
moreover obtain ψ **where** ψ ∈ *iso* (*G*⦇*carrier* := 𝔊 ! *i*⦈ *Mod* (*carrier* (*G*⦇*carrier* := 𝔊 ! *i*⦈))) (*G*⦇*carrier* := {**1**$_{G⦇carrier := 𝔊 ! i⦈}$}⦈)
using *group.self-factor-iso groupGi* **by** *force*
ultimately obtain π **where** π ∈ *iso* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K*

∩ 𝔊 ! *i*)) (*G*⦇*carrier* := {**1**}⦈)
using *iso-set-trans* **by** *fastforce*
hence *order* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*)) = *order* (*G*⦇*carrier* := {**1**}⦈)
by (*meson iso-same-order*)
hence *order* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*)) = *1* **unfolding** *order-def* **by** *auto*
hence *carrier* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*)) = {**1**$_{G⦇carrier := K ∩ 𝔊 ! (i + 1)⦈ Mod (K ∩}$
using *group.order-one-triv-iff KGsiKGigroup* **by** *blast*
moreover from *fstgoal* **have** *K* ∩ 𝔊 ! *i* ⊲ *G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ **by** *auto*
moreover from *finGSi* **have** *finite* (*carrier* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈)) **by** *auto*
ultimately have *K* ∩ 𝔊 ! *i* = *carrier* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈) **by** (*metis normal.fact-group-trivial-iff*)
hence (*remdups-adj* (*map* ((∩) *K*) 𝔊)) ! *j* = (*remdups-adj* (*map* ((∩) *K*) 𝔊)) ! (*j* + *1*) **using** *i* **by** *auto*
with *j* **have** *False* **using** *remdups-adj-adjacent KGnotempty Suc-eq-plus1* **by** *metis*
thus *simple-group* (*G*⦇*carrier* := *remdups-adj* (*map* ((∩) *K*) 𝔊) ! *Suc j*⦈ *Mod* *remdups-adj* (*map* ((∩) *K*) 𝔊) ! *j*)**..**
next
assume 𝔊 ! *i* <#> *K* ∩ 𝔊 ! *Suc i* = 𝔊 ! *Suc i*
with φ **have** φ ∈ *iso* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*)) (*G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*)
by *auto*
then obtain φ′ **where** φ′ ∈ *iso* (*G*⦇*carrier* := 𝔊 ! (*i* + *1*)⦈ *Mod* 𝔊 ! *i*) (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*))
using *KGsiKGigroup group.iso-set-sym* **by** *auto*
with *Gisimple KGsiKGigroup* **have** *simple-group* (*G*⦇*carrier* := *K* ∩ 𝔊 ! (*i* + *1*)⦈ *Mod* (*K* ∩ 𝔊 ! *i*)) **by** (*metis simple-group.iso-simple*)
with *i* **show** *simple-group* (*G*⦇*carrier* := *remdups-adj* (*map* ((∩) *K*) 𝔊) ! *Suc j*⦈ *Mod* *remdups-adj* (*map* ((∩) *K*) 𝔊) ! *j*)
by *auto*
qed
qed

lemma (**in** *group*) *composition-series-extend*:
assumes *composition-series* (*G*⦇*carrier* := *H*⦈) 𝔥
assumes *simple-group* (*G Mod H*) *H* ⊲ *G*
shows *composition-series G* (𝔥 @ [*carrier G*])
unfolding *composition-series-def composition-series-axioms-def*
proof *auto*
from *assms*(*1*) **interpret** *comp*𝔥: *composition-series G*⦇*carrier* := *H*⦈ 𝔥 **.**
show *normal-series G* (𝔥 @ [*carrier G*]) **using** *assms*(*3*) *comp*𝔥*.is-normal-series* **by** (*metis normal-series-extend*)
fix *i*
assume *i*: *i* < *length* 𝔥
show *simple-group* (*G*⦇*carrier* := (𝔥 @ [*carrier G*]) ! *Suc i*⦈ *Mod* (𝔥 @ [*carrier*

G]) ! *i*)
  proof (*cases i* = *length* 𝔉 − *1*)
    case *True*
    hence (𝔉 @ [*carrier G*]) ! *Suc i* = *carrier G* **by** (*metis i diff-Suc-1 lessE nth-append-length*)
    moreover have (𝔉 @ [*carrier G*]) ! *i* = 𝔉 ! *i***by** (*metis butlast-snoc i nth-butlast*)
    hence (𝔉 @ [*carrier G*]) ! *i* = *H* **using** *True last-conv-nth comp*𝔉*.notempty comp*𝔉*.last* **by** *auto*
    ultimately show *?thesis* **using** *assms*(*2*) **by** *auto*
  next
    case *False*
    hence *Suc i* < *length* 𝔉 **using** *i* **by** *auto*
    hence (𝔉 @ [*carrier G*]) ! *Suc i* = 𝔉 ! *Suc i* **using** *nth-append* **by** *metis*
    moreover from *i* **have** (𝔉 @ [*carrier G*]) ! *i* = 𝔉 ! *i* **using** *nth-append* **by** *metis*
    ultimately show *?thesis* **using** ‹*Suc i* < *length* 𝔉› *comp*𝔉*.simplefact* **by** *auto*
  qed
qed

lemma (**in** *composition-series*) *entries-mono*:
  assumes *i* ≤ *j j* < *length* 𝔊
  shows 𝔊 ! *i* ⊆ 𝔊 ! *j*
using *assms* **proof** (*induction j* − *i arbitrary*: *i j*)
  case *0*
  hence *i* = *j* **by** *auto*
  thus 𝔊 ! *i* ⊆ 𝔊 ! *j* **by** *auto*
next
  case (*Suc k i j*)
  hence *i′*: *i* + (*Suc k*) = *j i* + *1* < *length* 𝔊 **by** *auto*
  hence *ij*: *i* + *1* ≤ *j* **by** *auto*
  have 𝔊 ! *i* ⊆ 𝔊 ! (*i* + *1*) **using** *i′ normal normal-imp-subgroup subgroup.subset* **by** *force*
  moreover have *j* − (*i* + *1*) = *k j* < *length* 𝔊 **using** *Suc assms* **by** *auto*
  hence 𝔊 ! (*i* + *1*) ⊆ 𝔊 ! *j* **using** *Suc*(*1*) *ij* **by** *auto*
  ultimately show 𝔊 ! *i* ⊆ 𝔊 ! *j* **by** *simp*
qed

end

theory *GroupIsoClasses*
imports
  HOL−Algebra.Coset
begin

3 Isomorphism Classes of Groups

We construct a quotient type for isomorphism classes of groups.

typedef $'a$ *group* = $\{G :: 'a$ *monoid. group G*$\}$
proof
  show $\bigwedge a.$ (|*carrier* = $\{a\}$, *mult* = $(\lambda x\ y.\ x)$, *one* = *a*|) $\in \{G.$ *group G*$\}$
  unfolding *group-def group-axioms-def monoid-def Units-def* **by** *auto*
qed

definition *group-iso-rel* :: $'a$ *group* $\Rightarrow$ $'a$ *group* $\Rightarrow$ *bool*
  where *group-iso-rel G H* = $(\exists \varphi.\ \varphi \in$ *iso* (*Rep-group G*) (*Rep-group H*))

quotient-type $'a$ *group-iso-class* = $'a$ *group* / *group-iso-rel*
  morphisms *Rep-group-iso Abs-group-iso*
proof (*rule equivpI*)
  show *reflp group-iso-rel*
  proof (*rule reflpI*)
    fix *G* :: $'b$ *group*
    show *group-iso-rel G G*
      unfolding *group-iso-rel-def* **using** *iso-set-refl* **by** *blast*
  qed
next
  show *symp group-iso-rel*
  proof (*rule sympI*)
    fix *G H* :: $'b$ *group*
    assume *group-iso-rel G H*
     then obtain φ **where** $\varphi \in$ *iso* (*Rep-group G*) (*Rep-group H*) **unfolding** *group-iso-rel-def* **by** *auto*
   then obtain φ' **where** $\varphi' \in$ *iso* (*Rep-group H*) (*Rep-group G*) **using** *group.iso-sym Rep-group*
      using *group.iso-set-sym* **by** *blast*
    thus *group-iso-rel H G* **unfolding** *group-iso-rel-def* **by** *auto*
  qed
next
  show *transp group-iso-rel*
  proof (*rule transpI*)
    fix *G H I* :: $'b$ *group*
    assume *group-iso-rel G H group-iso-rel H I*
     then obtain φ ψ **where** $\varphi \in$ *iso* (*Rep-group G*) (*Rep-group H*) $\psi \in$ *iso* (*Rep-group H*) (*Rep-group I*)
      unfolding *group-iso-rel-def* **by** *auto*
    then obtain π **where** $\pi \in$ *iso* (*Rep-group G*) (*Rep-group I*)
      using *iso-set-trans* **by** *blast*
    thus *group-iso-rel G I* **unfolding** *group-iso-rel-def* **by** *auto*
  qed
qed

This assigns to a given group the group isomorphism class

definition (**in** *group*) *iso-class* :: $'a$ *group-iso-class*
  where *iso-class* = *Abs-group-iso* (*Abs-group* (*monoid.truncate G*))

Two isomorphic groups do indeed have the same isomorphism class:

lemma *iso-classes-iff*:
  assumes *group G*
  assumes *group H*
  shows (∃ φ. φ ∈ *iso G H*) = (*group.iso-class G* = *group.iso-class H*)
proof −
 from *assms*(*1*,*2*) **have** *groups*:*group* (*monoid.truncate G*) *group* (*monoid.truncate H*)
   unfolding *monoid.truncate-def group-def group-axioms-def Units-def monoid-def* **by** *auto*
  have (∃ φ. φ ∈ *iso G H*) = (∃ φ. φ ∈ *iso* (*monoid.truncate G*) (*monoid.truncate H*))
    unfolding *iso-def hom-def monoid.truncate-def* **by** *auto*
 also have ... = *group-iso-rel* (*Abs-group* (*monoid.truncate G*)) (*Abs-group* (*monoid.truncate H*))
    unfolding *group-iso-rel-def* **using** *groups group.Abs-group-inverse* **by** (*metis mem-Collect-eq*)
 also have ... = (*group.iso-class G* = *group.iso-class H*) **using** *group.iso-class-def assms group-iso-class.abs-eq-iff* **by** *metis*
  finally show *?thesis*.
qed

end

theory *JordanHolder*
imports
  CompositionSeries
  MaximalNormalSubgroups
  HOL−Library.Multiset
  GroupIsoClasses
begin

4 The Jordan-Hölder Theorem

locale *jordan-hoelder* = *group*
  + *comp*𝔥*?*: *composition-series G* 𝔥
  + *comp*𝔊*?*: *composition-series G* 𝔊 **for** 𝔥 **and** 𝔊
  + **assumes** *finite*: *finite* (*carrier G*)

Before we finally start the actual proof of the theorem, one last lemma: Cancelling the last entry of a normal series results in a normal series with quotients being all but the last of the original ones.

lemma (**in** *normal-series*) *quotients-butlast*:
  assumes *length* 𝔊 > *1*
  shows *butlast quotients* = *normal-series.quotients* (*G*⦇*carrier* := 𝔊 ! (*length* 𝔊 − *1* − *1*)⦈) (*take* (*length* 𝔊 − *1*) 𝔊)
proof (*rule nth-equalityI*)
  define *n* **where** *n* = *length* 𝔊 − *1*

hence $n = length\ (take\ n\ \mathfrak{G})\ n > 0\ n < length\ \mathfrak{G}$ **using** *assms notempty* **by** *auto*
interpret *normal𝔊butlast*: *normal-series* $(G(\!|carrier := \mathfrak{G}\ !\ (n - 1)|\!))$ *take n* 𝔊

using *normal-series-prefix-closed* ‹$n > 0$› ‹$n < length\ \mathfrak{G}$› **by** *auto*
have *length* (*butlast quotients*) = *length quotients* − *1* **by** (*metis length-butlast*)
also have ... = *length* 𝔊 − *1* − *1* **by** (*metis add-diff-cancel-right′ quotients-length*)
also have ... = *length* (*take n* 𝔊) − *1* **by** (*metis* ‹*n* = *length* (*take n* 𝔊)› *n-def*)
also have ... = *length normal𝔊butlast.quotients* **by** (*metis normal𝔊butlast.quotients-length diff-add-inverse2*)
finally show *length* (*butlast quotients*) = *length normal𝔊butlast.quotients* **.**
have ∀ *i*<*length* (*butlast quotients*). *butlast quotients* ! *i* = *normal𝔊butlast.quotients* ! *i*
proof *auto*
fix *i*
assume *i*: $i < length\ quotients - Suc\ 0$
hence *i′*: $i < length\ \mathfrak{G} - 1\ i < n\ i + 1 < n$ **unfolding** *n-def* **using** *quotients-length* **by** *auto*
from *i* **have** *butlast quotients* ! *i* = *quotients* ! *i* **by** (*metis One-nat-def length-butlast nth-butlast*)
also have ... = $G(\!|carrier := \mathfrak{G}\ !\ (i + 1)|\!)$ *Mod* 𝔊 ! *i* **unfolding** *quotients-def* **using** *i′*(*1*) **by** *auto*
also have ... = $G(\!|carrier := (take\ n\ \mathfrak{G})\ !\ (i + 1)|\!)$ *Mod* (*take n* 𝔊) ! *i* **using** *i′*(*2*,*3*) *nth-take* **by** *metis*
also have ... = *normal𝔊butlast.quotients* ! *i* **unfolding** *normal𝔊butlast.quotients-def* **using** *i′* **by** *fastforce*
finally show *butlast* (*normal-series.quotients G* 𝔊) ! *i* = *normal-series.quotients* $(G(\!|carrier := \mathfrak{G}\ !\ (n - Suc\ 0)|\!))$ (*take n* 𝔊) ! *i* **by** *auto*
qed
thus ⋀*i*. *i* < *length* (*butlast quotients*)
⟹ *butlast quotients* ! *i*
= *normal-series.quotients* $(G(\!|carrier := \mathfrak{G}\ !\ (length\ \mathfrak{G} - 1 - 1)|\!))$ (*take* (*length* 𝔊 − *1*) 𝔊) ! *i*
unfolding *n-def* **by** *auto*
qed

The main part of the Jordan Hölder theorem is its statement about the uniqueness of a composition series. Here, uniqueness up to reordering and isomorphism is modelled by stating that the multisets of isomorphism classes of all quotients are equal.

theorem *jordan-hoelder-multisets*:
assumes *group G*
assumes *finite* (*carrier G*)
assumes *composition-series G* 𝔊
assumes *composition-series G* 𝔋
shows *mset* (*map group.iso-class* (*normal-series.quotients G* 𝔊))
= *mset* (*map group.iso-class* (*normal-series.quotients G* 𝔋))
using *assms*
proof (*induction length* 𝔊 *arbitrary*: 𝔊 𝔋 *G rule*: *full-nat-induct*)

```
  case (1 𝔊 𝔋 G)
  then interpret comp𝔊: composition-series G 𝔊 by simp
  from 1 interpret comp𝔋: composition-series G 𝔋 by simp
  from 1 interpret grpG: group G by simp
  show ?case
  proof (cases length 𝔊 ≤ 2)
  next
    case True
    hence  length 𝔊 = 0 ∨ length 𝔊 = 1 ∨ length 𝔊 = 2 by arith
    with comp𝔊.notempty have  length 𝔊 = 1 ∨ length 𝔊 = 2 by simp
    thus ?thesis
    proof (auto simp del: mset-map)
      — First trivial case: 𝔊 is the trivial group.
      assume length 𝔊 = Suc 0
      hence length: length 𝔊 = 1 by simp
      hence length [] + 1 = length 𝔊 by auto
     moreover from length have char𝔊: 𝔊 = [{𝟭_G}] by (metis comp𝔊.composition-series-length-one)
      hence carrier G = {𝟭_G} by (metis comp𝔊.composition-series-triv-group)
       with length char𝔊 have 𝔊 = 𝔋 using comp𝔋.composition-series-triv-group
by simp
      thus ?thesis by simp
    next
      — Second trivial case: 𝔊 is simple.
      assume length 𝔊 = 2
      hence 𝔊char: 𝔊 = [{𝟭_G}, carrier G] by (metis comp𝔊.length-two-unique)
     hence simple: simple-group G by (metis comp𝔊.composition-series-simple-group)
       hence 𝔋 = [{𝟭_G}, carrier G] using comp𝔋.composition-series-simple-group
by auto
      with 𝔊char have 𝔊 = 𝔋 by simp
      thus ?thesis by simp
    qed
  next
    case False
    — Non-trivial case: 𝔊 has length at least 3.
    hence length: length 𝔊 ≥ 3 by simp
    — First we show that 𝔋 must have a length of at least 3.
   hence ¬ simple-group G using comp𝔊.composition-series-simple-group by auto
   hence 𝔋 ≠ [{𝟭_G}, carrier G] using comp𝔋.composition-series-simple-group by
auto
    hence length 𝔋 ≠ 2 using comp𝔋.length-two-unique by auto
   moreover from length have carrier G ≠ {𝟭_G} using comp𝔊.composition-series-length-one
comp𝔊.composition-series-triv-group by auto
   hence length 𝔋 ≠ 1 using comp𝔋.composition-series-length-one comp𝔋.composition-series-triv-group
by auto
    moreover from comp𝔋.notempty have length 𝔋 ≠ 0 by simp
    ultimately have length𝔋big: length 𝔋 ≥ 3 using comp𝔋.notempty by arith
    define m where m = length 𝔋 − 1
    define n where n = length 𝔊 − 1
    from length𝔋big have m': m > 0 m < length 𝔋 (m − 1) + 1 < length 𝔋 m
```

− *1* = *length* 𝔥 − *2 m* − *1* + *1* = *length* 𝔥 − *1 m* − *1* < *length* 𝔥
unfolding *m-def* **by** *auto*
from *length* **have** *n′*: *n* > *0 n* < *length* 𝔊 (*n* − *1*) + *1* < *length* 𝔊 *n* − *1* < *length* 𝔊 *Suc n* ≤ *length* 𝔊
n − *1* = *length* 𝔊 − *2 n* − *1* + *1* = *length* 𝔊 − *1* **unfolding** *n-def* **by** *auto*
define 𝔊*Pn* **where** 𝔊*Pn* = *G*(|*carrier* := 𝔊 ! (*n* − *1*)|)
define 𝔥*Pm* **where** 𝔥*Pm* = *G*(|*carrier* := 𝔥 ! (*m* − *1*)|)
then interpret *grp*𝔊*Pn*: *group* 𝔊*Pn* **unfolding** 𝔊*Pn-def* **using** *n′* **by** (*metis comp*𝔊*.normal-series-subgroups comp*𝔊*.subgroup-imp-group*)
interpret *grp*𝔥*Pm*: *group* 𝔥*Pm* **unfolding** 𝔥*Pm-def* **using** *m′ comp*𝔥*.normal-series-subgroups 1*(*2*) *group.subgroup-imp-group* **by** *force*
have *finGbl*: *finite* (*carrier* 𝔊*Pn*) **using** ‹*n* − *1* < *length* 𝔊› *1*(*3*) **unfolding** 𝔊*Pn-def* **using** *comp*𝔊*.normal-series-subgroups comp*𝔊*.subgroup-finite* **by** *auto*
have *finHbl*: *finite* (*carrier* 𝔥*Pm*) **using** ‹*m* − *1* < *length* 𝔥› *1*(*3*) **unfolding** 𝔥*Pm-def* **using** *comp*𝔥*.normal-series-subgroups comp*𝔊*.subgroup-finite* **by** *auto*
have *quots*𝔊*notempty*: *comp*𝔊*.quotients* ≠ [] **using** *comp*𝔊*.quotients-length length* **by** *auto*
have *quots*𝔥*notempty*: *comp*𝔥*.quotients* ≠ [] **using** *comp*𝔥*.quotients-length length*𝔥*big* **by** *auto*

— Instantiate truncated composition series since they are used for both cases
interpret 𝔥*butlast*: *composition-series* 𝔥*Pm take m* 𝔥 **using** *comp*𝔥*.composition-series-prefix-closed m′*(*1*,*2*) 𝔥*Pm-def* **by** *auto*
interpret 𝔊*butlast*: *composition-series* 𝔊*Pn take n* 𝔊 **using** *comp*𝔊*.composition-series-prefix-closed n′*(*1*,*2*) 𝔊*Pn-def* **by** *auto*
have *ltaken*: *n* = *length* (*take n* 𝔊) **using** *length-take n′*(*2*) **by** *auto*
have *ltakem*: *m* = *length* (*take m* 𝔥) **using** *length-take m′*(*2*) **by** *auto*
show *?thesis*
proof (*cases* 𝔥 ! (*m* − *1*) = 𝔊 ! (*n* − *1*))
— If 𝔥 ! (*l* − *1*) = 𝔊 ! *1*, everything is simple...
case *True*
— The last quotients of 𝔊 and 𝔥 are equal.
have *lasteq*: *last comp*𝔊*.quotients* = *last comp*𝔥*.quotients*
proof −
from *length* **have** *lg*: *length* 𝔊 − *1* − *1* + *1* = *length* 𝔊 − *1* **by** (*metis Suc-diff-1 Suc-eq-plus1 n′*(*1*) *n-def*)
from *length*𝔥*big* **have** *lh*: *length* 𝔥 − *1* − *1* + *1* = *length* 𝔥 − *1* **by** (*metis Suc-diff-1 Suc-eq-plus1* ‹*0* < *m*› *m-def*)
have *last comp*𝔊*.quotients* = *G Mod* 𝔊 ! (*n* − *1*) **using** *length comp*𝔊*.last-quotient* **unfolding** *n-def* **by** *auto*
also have ... = *G Mod* 𝔥 ! (*m* − *1*) **using** *True* **by** *simp*
also have ... = *last comp*𝔥*.quotients* **using** *length*𝔥*big comp*𝔥*.last-quotient* **unfolding** *m-def* **by** *auto*
finally show *?thesis* **.**
qed
from *ltaken* **have** *ind*: *mset* (*map group.iso-class* 𝔊*butlast.quotients*) = *mset* (*map group.iso-class* 𝔥*butlast.quotients*)
using *1*(*1*) *True n′*(*5*) *grp*𝔊*Pn.is-group finGbl* 𝔊*butlast.is-composition-series* 𝔥*butlast.is-composition-series* **unfolding** 𝔊*Pn-def* 𝔥*Pm-def* **by** *metis*

have *mset* (*map group.iso-class comp*𝔊*.quotients*)
= *mset* (*map group.iso-class* (*butlast comp*𝔊*.quotients* @ [*last comp*𝔊*.quotients*])) **by** (*simp add*: *quots*𝔊*notempty*)
also have ... = *mset* (*map group.iso-class* (𝔊*butlast.quotients* @ [*last* (*comp*𝔊*.quotients*)]))
using *comp*𝔊*.quotients-butlast length* **unfolding** *n-def* 𝔊*Pn-def* **by** *auto*
also have ... = *mset* ((*map group.iso-class* 𝔊*butlast.quotients*) @ [*group.iso-class* (*last* (*comp*𝔊*.quotients*))]) **by** *auto*
also have ... = *mset* (*map group.iso-class* 𝔊*butlast.quotients*) + {# *group.iso-class* (*last* (*comp*𝔊*.quotients*)) #} **by** *auto*
also have ... = *mset* (*map group.iso-class* 𝔋*butlast.quotients*) + {# *group.iso-class* (*last* (*comp*𝔊*.quotients*)) #} **using** *ind* **by** *simp*
also have ... = *mset* (*map group.iso-class* 𝔋*butlast.quotients*) + {# *group.iso-class* (*last* (*comp*𝔋*.quotients*)) #} **using** *lasteq* **by** *simp*
also have ... = *mset* ((*map group.iso-class* 𝔋*butlast.quotients*) @ [*group.iso-class* (*last* (*comp*𝔋*.quotients*))]) **by** *auto*
also have ... = *mset* (*map group.iso-class* (𝔋*butlast.quotients* @ [*last* (*comp*𝔋*.quotients*)]))
by *auto*
also have ... = *mset* (*map group.iso-class* (*butlast comp*𝔋*.quotients* @ [*last comp*𝔋*.quotients*])) **using** *length*𝔋*big comp*𝔋*.quotients-butlast* **unfolding** *m-def* 𝔋*Pm-def*
by *auto*
also have ... = *mset* (*map group.iso-class comp*𝔋*.quotients*) **using** *append-butlast-last-id quots*𝔋*notempty* **by** *simp*
finally show *?thesis* **.**
next
case *False*
define 𝔋*PmInt*𝔊*Pn* **where** 𝔋*PmInt*𝔊*Pn* = *G*⦇*carrier* := 𝔋 ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)⦈
interpret 𝔊*Pnmax*: *max-normal-subgroup* 𝔊 ! (*n* − *1*) *G* **unfolding** *n-def*
by (*metis add-lessD1 diff-diff-add n*′(*3*) *add.commute one-add-one 1*(*3*) *comp*𝔊*.snd-to-last-max-normal*)
interpret 𝔋*Pmmax*: *max-normal-subgroup* 𝔋 ! (*m* − *1*) *G* **unfolding** *m-def*
by (*metis add-lessD1 diff-diff-add m*′(*3*) *add.commute one-add-one 1*(*3*) *comp*𝔋*.snd-to-last-max-normal*)
have 𝔋*PmnormG*: 𝔋 ! (*m* − *1*) ◁ *G* **using** *comp*𝔋*.normal-series-snd-to-last m*′(*4*) **unfolding** *m-def* **by** *auto*
have 𝔊*PnnormG*: 𝔊 ! (*n* − *1*) ◁ *G* **using** *comp*𝔊*.normal-series-snd-to-last n*′(*6*) **unfolding** *n-def* **by** *auto*
have 𝔋*Pmint*𝔊*PnnormG*: 𝔋 ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*) ◁ *G* **using** 𝔋*PmnormG* 𝔊*PnnormG* **by** (*rule comp*𝔊*.normal-subgroup-intersect*)
have *Intnorm*𝔊*Pn*: 𝔋 ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*) ◁ 𝔊*Pn* **using** 𝔊*PnnormG* 𝔋*PmnormG Int-lower2* **unfolding** 𝔊*Pn-def*
by (*metis comp*𝔊*.normal-restrict-supergroup comp*𝔊*.normal-series-subgroups comp*𝔊*.normal-subgroup-intersect n*′(*4*))
then interpret *grp*𝔊*PnMod*𝔋*Pmint*𝔊*Pn*: *group* 𝔊*Pn Mod* 𝔋 ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*) **by** (*rule normal.factorgroup-is-group*)
have *Intnorm*𝔋*Pm*: 𝔋 ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*) ◁ 𝔋*Pm* **using** 𝔋*PmnormG* 𝔊*PnnormG Int-lower2 Int-commute* **unfolding** 𝔋*Pm-def*
by (*metis comp*𝔊*.normal-restrict-supergroup comp*𝔊*.normal-subgroup-intersect comp*𝔋*.normal-series-subgroups m*′(*6*))

then interpret *grpℌPmModℌPmint𝔊Pn*: *group ℌPm Mod ℌ* ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*) **by** (*rule normal.factorgroup-is-group*)

— Show that the second to last entries are not contained in each other.
have *notℌPmSub𝔊Pn*: ¬ (ℌ ! (*m* − *1*) ⊆ 𝔊 ! (*n* − *1*)) **using** *ℌPm-max.max-normal 𝔊PnnormG False*[*symmetric*] *𝔊Pnmax.proper* **by** *simp*
have *not𝔊PnSubℌPm*: ¬ (𝔊 ! (*n* − *1*) ⊆ ℌ ! (*m* − *1*)) **using** *𝔊Pn-max.max-normal ℌPmnormG False ℌPmmax.proper* **by** *simp*

— Show that *G Mod* ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*) is a simple group.
have *ℌPmSubSetmult*: ℌ ! (*m* − *1*) ⊆ ℌ ! (*m* − *1*) <#>$_G$ 𝔊 ! (*n* − *1*)
using *𝔊Pnmax.subgroup-axioms ℌPmnormG second-isomorphism-grp.H-contained-in-set-mult*

second-isomorphism-grp-axioms-def second-isomorphism-grp-def **by** *blast*
have *𝔊PnSubSetmult*: 𝔊 ! (*n* − *1*) ⊆ ℌ ! (*m* − *1*) <#>$_G$ 𝔊 ! (*n* − *1*)
by (*metis 𝔊Pnmax.subset 𝔊PnnormG ℌPmSubSetmult ℌPmmax.max-normal ℌPmmax.subgroup-axioms ℌPmnormG*
comp𝔊.normal-subgroup-set-mult-closed comp𝔊.set-mult-inclusion)
have 𝔊 ! (*n* − *1*) ≠ (ℌ ! (*m* − *1*)) <#>$_G$ (𝔊 ! (*n* − *1*)) **using** *ℌPmSubSetmult notℌPmSub𝔊Pn* **by** *auto*
hence *set-multG*: (ℌ ! (*m* − *1*)) <#>$_G$ (𝔊 ! (*n* − *1*)) = *carrier G*
by (*metis 𝔊PnSubSetmult 𝔊Pnmax.max-normal 𝔊PnnormG ℌPmnormG comp𝔊.normal-subgroup-set-mult-closed*)
then obtain φ **where** φ ∈ *iso* (*𝔊Pn Mod* (ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*))) (*G*⦇*carrier* := *carrier G*⦈ *Mod* ℌ ! (*m* − *1*))
by (*metis second-isomorphism-grp.normal-intersection-quotient-isom ℌPm-normG*
𝔊Pn-def 𝔊Pnmax.subgroup-axioms second-isomorphism-grp-axioms-def second-isomorphism-grp-def)
hence φ: φ ∈ *iso* (*𝔊Pn Mod* (ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*))) (*G Mod* ℌ ! (*m* − *1*)) **by** *auto*
then obtain *φ2* **where** *φ2*: *φ2* ∈ *iso* (*G Mod* ℌ ! (*m* − *1*)) (*𝔊Pn Mod* (ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)))
using *group.iso-set-sym grp𝔊PnModℌPmint𝔊Pn.is-group* **by** *auto*
moreover have *simple-group* (*G*⦇*carrier* := ℌ ! (*m* − *1* + *1*)⦈ *Mod* ℌ ! (*m* − *1*)) **using** *compℌ.simplefact m′*(*3*) **by** *simp*
hence *simple-group* (*G Mod* ℌ ! (*m* − *1*)) **using** *compℌ.last last-conv-nth compℌ.notempty m′*(*5*) **by** *fastforce*
ultimately have *simple𝔊PnModInt*: *simple-group* (*𝔊Pn Mod* (ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)))
using *simple-group.iso-simple grp𝔊PnModℌPmint𝔊Pn.is-group* **by** *auto*
interpret *grpGModℌPm*: *group* (*G Mod* ℌ ! (*m* − *1*)) **by** (*metis ℌPmnormG normal.factorgroup-is-group*)

— Show analogues of the previous statements for ℌ ! (*m* − *1*) instead of 𝔊 ! (*n* − *1*).
have *ℌPmSubSetmult′*: ℌ ! (*m* − *1*) ⊆ 𝔊 ! (*n* − *1*) <#>$_G$ ℌ ! (*m* − *1*)
by (*metis 𝔊PnnormG ℌPmSubSetmult comp𝔊.commut-normal ℌPmnormG normal-imp-subgroup*)

have 𝔊*PnSubSetmult'*: 𝔊 ! (*n* − *1*) ⊆ 𝔊 ! (*n* − *1*) <#>$_G$ ℌ ! (*m* − *1*)
by (*metis* ℌ*PmnormG normal-imp-subgroup* 𝔊*PnSubSetmult* 𝔊*PnnormG comp*𝔊*.commut-normal*)
have ℌ ! (*m* − *1*) ≠ (𝔊 ! (*n* − *1*)) <#>$_G$ (ℌ ! (*m* − *1*)) **using** 𝔊*PnSubSetmult' not*𝔊*PnSub*ℌ*Pm* **by** *auto*
hence *set-multG*: (𝔊 ! (*n* − *1*)) <#>$_G$ (ℌ ! (*m* − *1*)) = *carrier G*
using ℌ*Pmmax.max-normal* 𝔊*PnnormG comp*𝔊*.normal-subgroup-set-mult-closed* ℌ*PmSubSetmult'* ℌ*PmnormG* **by** *blast*
from *set-multG* **obtain** *ψ* **where**
ψ ∈ *iso* (ℌ*Pm Mod* (𝔊 ! (*n* − *1*) ∩ ℌ ! (*m* − *1*))) (*G*⦇*carrier* := *carrier G*⦈ *Mod* 𝔊 ! (*n* − *1*))
by (*metis* ℌ*Pm-def* ℌ*PmnormG second-isomorphism-grp-axioms-def second-isomorphism-grp-def*
second-isomorphism-grp.normal-intersection-quotient-isom 𝔊*PnnormG normal-imp-subgroup*)

hence *ψ*: *ψ* ∈ *iso* (ℌ*Pm Mod* (ℌ ! (*m* − *1*) ∩ (𝔊 ! (*n* − *1*)))) (*G*⦇*carrier* := *carrier G*⦈ *Mod* 𝔊 ! (*n* − *1*)) **using** *Int-commute* **by** *metis*
then obtain *ψ2* **where**
ψ2: *ψ2* ∈ *iso* (*G Mod* 𝔊 ! (*n* − *1*)) (ℌ*Pm Mod* (ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)))
using *group.iso-set-sym grp*ℌ*PmMod*ℌ*Pmint*𝔊*Pn.is-group* **by** *auto*
moreover have *simple-group* (*G*⦇*carrier* := 𝔊 ! (*n* − *1* + *1*)⦈ *Mod* 𝔊 ! (*n* − *1*)) **using** *comp*𝔊*.simplefact n'*(*3*) **by** *simp*
hence *simple-group* (*G Mod* 𝔊 ! (*n* − *1*)) **using** *comp*𝔊*.last last-conv-nth comp*𝔊*.notempty n'*(*7*) **by** *fastforce*
ultimately have *simple*ℌ*PmModInt*: *simple-group* (ℌ*Pm Mod* (ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)))
using *simple-group.iso-simple grp*ℌ*PmMod*ℌ*Pmint*𝔊*Pn.is-group* **by** *auto*
interpret *grpGMod*𝔊*Pn*: *group* (*G Mod* 𝔊 ! (*n* − *1*)) **by** (*metis* 𝔊*PnnormG normal.factorgroup-is-group*)

— Instantiate several composition series used to build up the equality of quotient multisets.
define 𝔎 **where** 𝔎 = *remdups-adj* (*map* ((∩) (ℌ ! (*m* − *1*))) 𝔊)
define 𝔏 **where** 𝔏 = *remdups-adj* (*map* ((∩) (𝔊 ! (*n* − *1*))) ℌ)
interpret 𝔎: *composition-series* ℌ*Pm* 𝔎 **using** *comp*𝔊*.intersect-normal 1*(*3*) ℌ*PmnormG* **unfolding** 𝔎*-def* ℌ*Pm-def* **by** *auto*
interpret 𝔏: *composition-series* 𝔊*Pn* 𝔏 **using** *comp*ℌ*.intersect-normal 1*(*3*) 𝔊*PnnormG* **unfolding** 𝔏*-def* 𝔊*Pn-def* **by** *auto*

— Apply the induction hypothesis on 𝔊*butlast* and 𝔏
from *n'*(*2*) **have** *Suc* (*length* (*take n* 𝔊)) ≤ *length* 𝔊 **by** *auto*
hence *multisets*𝔊*butlast*𝔏: *mset* (*map group.iso-class* 𝔊*butlast.quotients*) = *mset* (*map group.iso-class* 𝔏*.quotients*)
using *1.hyps grp*𝔊*Pn.is-group finGbl* 𝔊*butlast.is-composition-series* 𝔏*.is-composition-series* **by** *metis*
hence *length*𝔏: *n* = *length* 𝔏 **using** 𝔊*butlast.quotients-length* 𝔏*.quotients-length*

length-map size-mset ltaken **by** *metis*
    hence *length𝔏′*: *length 𝔏 > 1 length 𝔏 − 1 > 0 length 𝔏 − 1 ≤ length 𝔏*
using *n′(6) length* **by** *auto*
  have *Inteq𝔏sndlast*: *ℌ ! (m − 1) ∩ 𝔊 ! (n − 1) = 𝔏 ! (length 𝔏 − 1 − 1)*
  proof −
   have *length 𝔏 − 1 − 1 + 1 < length 𝔏* **using** *length𝔏′* **by** *auto*
    moreover have *KGnotempty*: *(map ((∩) (𝔊 ! (n − 1))) ℌ) ≠ []* **using**
compℌ.notempty **by** *(metis Nil-is-map-conv)*
   ultimately obtain *i* **where** *i*: *i + 1 < length (map ((∩) (𝔊 ! (n − 1))) ℌ)*
    *𝔏 ! (length 𝔏 − 1 − 1) = (map ((∩) (𝔊 ! (n − 1))) ℌ) ! i 𝔏 ! (length 𝔏
− 1 − 1 + 1) = (map ((∩) (𝔊 ! (n − 1))) ℌ) ! (i + 1)*
    using *remdups-adj-obtain-adjacency* **unfolding** *𝔏-def* **by** *force*
   hence *𝔏 ! (length 𝔏 − 1 − 1) = ℌ ! i ∩ 𝔊 ! (n − 1) 𝔏 ! (length 𝔏 − 1 −
1 + 1) = ℌ ! (i + 1) ∩ 𝔊 ! (n − 1)* **by** *auto*
   hence *𝔏 ! (length 𝔏 − 1) = ℌ ! (i + 1) ∩ 𝔊 ! (n − 1)* **using** *length𝔏′(2)*
by *(metis Suc-diff-1 Suc-eq-plus1)*
   hence *𝔊Pnsubℌ Pm*: *𝔊 ! (n − 1) ⊆ ℌ ! (i + 1)* **using** *𝔏.last 𝔏.notempty
last-conv-nth* **unfolding** *𝔊Pn-def* **by** *auto*
   from *i(1)* **have** *i + 1 < m + 1* **unfolding** *m-def* **by** *auto*
    moreover have *¬ (i + 1 ≤ m − 1)* **using** *compℌ.entries-mono m′(6)
not𝔊PnSubℌPm 𝔊PnsubℌPm* **by** *fastforce*
   ultimately have *m − 1 = i* **by** *auto*
   with *i* **show** *?thesis* **by** *auto*
  qed
  hence *𝔏sndlast*: *ℌPmInt𝔊Pn = (𝔊Pn⦇carrier := 𝔏 ! (length 𝔏 − 1 − 1)⦈)*
unfolding *ℌPmInt𝔊Pn-def 𝔊Pn-def* **by** *auto*
  then interpret *𝔏butlast*: *composition-series ℌPmInt𝔊Pn take (length 𝔏 −
1) 𝔏* **using** *length𝔏′ 𝔏.composition-series-prefix-closed* **by** *metis*
  from *‹length 𝔏 > 1›* **have** *quots𝔏notemtpy*: *𝔏.quotients ≠ []* **unfolding**
𝔏.quotients-def **by** *auto*

  — Apply the induction hypothesis on *𝔏butlast* and *𝔎butlast*
  have *length 𝔎 > 1*
  proof *(rule ccontr)*
   assume *¬ length 𝔎 > 1*
    with *𝔎.notempty* **have** *length 𝔎 = 1* **by** *(metis One-nat-def Suc-lessI
length-greater-0-conv)*
    hence *carrier ℌPm* = $\{\mathbf{1}_{\mathfrak{H}Pm}\}$ **using** *𝔎.composition-series-length-one
𝔎.composition-series-triv-group* **by** *auto*
   hence *carrier ℌPm* = $\{\mathbf{1}_G\}$ **unfolding** *ℌPm-def* **by** *auto*
    hence *carrier ℌPm ⊆ 𝔊 ! (n − 1)* **using** *𝔊Pnmax.is-subgroup sub-
group.one-closed* **by** *auto*
   with *notℌPmSub𝔊Pn* **show** *False* **unfolding** *ℌPm-def* **by** *auto*
  qed
  hence *length𝔎′*: *length 𝔎 − 1 > 0 length 𝔎 − 1 ≤ length 𝔎* **by** *auto*
  have *Inteq𝔎sndlast*: *ℌ ! (m − 1) ∩ 𝔊 ! (n − 1) = 𝔎 ! (length 𝔎 − 1 − 1)*
  proof −
   have *length 𝔎 − 1 − 1 + 1 < length 𝔎* **using** *length𝔎′* **by** *auto*
    moreover have *KGnotempty*: *(map ((∩) (ℌ ! (m − 1))) 𝔊) ≠ []* **using**

comp𝔊.notempty **by** (*metis Nil-is-map-conv*)
ultimately obtain *i* **where** *i*: *i* + *1* < *length* (*map* ((∩) (𝔋 ! (*m* − *1*))) 𝔊)
𝔎 ! (*length* 𝔎 − *1* − *1*) = (*map* ((∩) (𝔋 ! (*m* − *1*))) 𝔊) ! *i* 𝔎 ! (*length* 𝔎 − *1* − *1* + *1*) = (*map* ((∩) (𝔋 ! (*m* − *1*))) 𝔊) ! (*i* + *1*)
using *remdups-adj-obtain-adjacency* **unfolding** *𝔎-def* **by** *force*
hence 𝔎 ! (*length* 𝔎 − *1* − *1*) = 𝔊 ! *i* ∩ 𝔋 ! (*m* − *1*) 𝔎 ! (*length* 𝔎 − *1* − *1* + *1*) = 𝔊 ! (*i* + *1*) ∩ 𝔋 ! (*m* − *1*) **by** *auto*
hence 𝔎 ! (*length* 𝔎 − *1*) = 𝔊 ! (*i* + *1*) ∩ 𝔋 ! (*m* − *1*) **using** *length𝔎′*(*1*) **by** (*metis Suc-diff-1 Suc-eq-plus1*)
hence *𝔋Pmsub𝔊Pn*: 𝔋 ! (*m* − *1*) ⊆ 𝔊 ! (*i* + *1*) **using** *𝔎.last 𝔎.notempty last-conv-nth* **unfolding** *𝔋Pm-def* **by** *auto*
from *i*(*1*) **have** *i* + *1* < *n* + *1* **unfolding** *n-def* **by** *auto*
moreover have ¬ (*i* + *1* ≤ *n* − *1*) **using** *comp𝔊.entries-mono n′*(*2*) *not𝔋PmSub𝔊Pn 𝔋Pmsub𝔊Pn* **by** *fastforce*
ultimately have *n* − *1* = *i* **by** *auto*
with *i* **show** *?thesis* **by** *auto*
qed
have *composition-series* (*G*⦇*carrier* := 𝔎 ! (*length* 𝔎 − *1* − *1*)⦈) (*take* (*length* 𝔎 − *1*) 𝔎)
using *length𝔎′ 𝔎.composition-series-prefix-closed* **unfolding** *𝔋PmInt𝔊Pn-def 𝔋Pm-def* **by** *fastforce*
then interpret *𝔎butlast*: *composition-series 𝔋PmInt𝔊Pn* (*take* (*length* 𝔎 − *1*) 𝔎) **using** *Inteq𝔎sndlast* **unfolding** *𝔋PmInt𝔊Pn-def* **by** *auto*
from *finGbl* **have** *finInt*: *finite* (*carrier 𝔋PmInt𝔊Pn*) **unfolding** *𝔋PmInt𝔊Pn-def 𝔊Pn-def* **by** *simp*
moreover have *Suc* (*length* (*take* (*length* 𝔏 − *1*) 𝔏)) ≤ *length* 𝔊 **using** *length𝔏* **unfolding** *n-def* **using** *n′*(*2*) **by** *auto*
ultimately have *multisets𝔎𝔏butlast*: *mset* (*map group.iso-class 𝔏butlast.quotients*) = *mset* (*map group.iso-class 𝔎butlast.quotients*)
using *1.hyps 𝔏butlast.is-group 𝔎butlast.is-composition-series 𝔏butlast.is-composition-series* **by** *auto*
hence *length* (*take* (*length* 𝔎 − *1*) 𝔎) = *length* (*take* (*length* 𝔏 − *1*) 𝔏)
using *𝔎butlast.quotients-length 𝔏butlast.quotients-length length-map size-mset* **by** *metis*
hence *length* (*take* (*length* 𝔎 − *1*) 𝔎) = *n* − *1* **using** *length𝔏 n′*(*1*) **by** *auto*
hence *length𝔎*: *length* 𝔎 = *n* **by** (*metis Suc-diff-1 𝔎.notempty butlast-conv-take length-butlast length-greater-0-conv n′*(*1*))

— Apply the induction hypothesis on 𝔎 and *𝔋butlast*
from *Inteq𝔎sndlast* **have** *𝔎sndlast*: *𝔋PmInt𝔊Pn* = (*𝔋Pm*⦇*carrier* := 𝔎 ! (*length* 𝔎 − *1* − *1*)⦈) **unfolding** *𝔋PmInt𝔊Pn-def 𝔋Pm-def 𝔎-def* **by** *auto*
from *length𝔎* **have** *Suc* (*length* 𝔎) ≤ *length* 𝔊 **using** *n′*(*2*) **by** *auto*
hence *multisets𝔋butlast𝔎*: *mset* (*map group.iso-class 𝔋butlast.quotients*) = *mset* (*map group.iso-class 𝔎.quotients*)
using *1.hyps grp𝔋Pm.is-group finHbl 𝔋butlast.is-composition-series 𝔎.is-composition-series* **by** *metis*
hence *length𝔎*: *m* = *length* 𝔎 **using** *𝔋butlast.quotients-length 𝔎.quotients-length length-map size-mset ltakem* **by** *metis*

hence *length* 𝔎 > *1 length* 𝔎 − *1* > *0 length* 𝔎 − *1* ≤ *length* 𝔎 **using** *m′(4) length*ℌ*big* **by** *auto*
hence *quots*𝔎*notemtpy*: 𝔎.*quotients* ≠ [] **unfolding** 𝔎.*quotients-def* **by** *auto*

interpret 𝔎*butlastadd*𝔊*Pn*: *composition-series* 𝔊*Pn* (*take* (*length* 𝔎 − *1*) 𝔎) @ [𝔊 ! (*n* − *1*)]
using *grp*𝔊*Pn.composition-series-extend* 𝔎*butlast.is-composition-series simple*𝔊*PnModInt Intnorm*𝔊*Pn*
unfolding 𝔊*Pn-def* ℌ*PmInt*𝔊*Pn-def* **by** *auto*
interpret 𝔏*butlastadd*ℌ*Pm*: *composition-series* ℌ*Pm* (*take* (*length* 𝔏 − *1*) 𝔏) @ [ℌ ! (*m* − *1*)]
using *grp*ℌ*Pm.composition-series-extend* 𝔏*butlast.is-composition-series simple*ℌ*PmModInt Intnorm*ℌ*Pm*
unfolding ℌ*Pm-def* ℌ*PmInt*𝔊*Pn-def* **by** *auto*

— Prove equality of those composition series.
have *mset* (*map group.iso-class comp*𝔊.*quotients*)
= *mset* (*map group.iso-class* ((*butlast comp*𝔊.*quotients*) @ [*last comp*𝔊.*quotients*])) **using** *quots*𝔊*notempty* **by** *simp*
also have ... = *mset* (*map group.iso-class* (𝔊*butlast.quotients* @ [*G Mod* 𝔊 ! (*n* − *1*)]))
using *comp*𝔊.*quotients-butlast comp*𝔊.*last-quotient length* **unfolding** *n-def* 𝔊*Pn-def* **by** *auto*
also have ... = *mset* (*map group.iso-class* ((*butlast* 𝔏.*quotients*) @ [*last* 𝔏.*quotients*])) + {# *group.iso-class* (*G Mod* 𝔊 ! (*n* − *1*)) #}
using *multisets*𝔊*butlast*𝔏 *quots*𝔏*notemtpy* **by** *simp*
also have ... = *mset* (*map group.iso-class* (𝔏*butlast.quotients* @ [𝔊*Pn Mod* ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)])) + {# *group.iso-class* (*G Mod* 𝔊 ! (*n* − *1*)) #}
using 𝔏.*quotients-butlast* 𝔏.*last-quotient* ‹*length* 𝔏 > *1*› 𝔏*sndlast Inteq*𝔏*sndlast* **unfolding** *n-def* **by** *auto*
also have ... = *mset* (*map group.iso-class* 𝔎*butlast.quotients*) + {# *group.iso-class* (𝔊*Pn Mod* ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)) #} + {# *group.iso-class* (*G Mod* 𝔊 ! (*n* − *1*)) #}
using *multisets*𝔎𝔏*butlast* **by** *simp*
also have ... = *mset* (*map group.iso-class* 𝔎*butlast.quotients*) + {# *group.iso-class* (*G Mod* ℌ ! (*m* − *1*)) #} + {# *group.iso-class* (ℌ*Pm Mod* ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)) #}
using *φ ψ2 iso-classes-iff grp*𝔊*PnMod*ℌ*Pmint*𝔊*Pn.is-group grpGMod*ℌ*Pm.is-group grpGMod*𝔊*Pn.is-group grp*ℌ*PmMod*ℌ*Pmint*𝔊*Pn.is-group*
by *metis*
also have ... = *mset* (*map group.iso-class* 𝔎*butlast.quotients*) + {# *group.iso-class* (ℌ*Pm Mod* ℌ ! (*m* − *1*) ∩ 𝔊 ! (*n* − *1*)) #} + {# *group.iso-class* (*G Mod* ℌ ! (*m* − *1*)) #}
by *simp*
also have ... = *mset* (*map group.iso-class* ((*butlast* 𝔎.*quotients*) @ [*last* 𝔎.*quotients*])) + {# *group.iso-class* (*G Mod* ℌ ! (*m* − *1*)) #}
using 𝔎.*quotients-butlast* 𝔎.*last-quotient* ‹*length* 𝔎 > *1*› 𝔎*sndlast Inteq*𝔎*sndlast* **unfolding** *m-def* **by** *auto*
also have ... = *mset* (*map group.iso-class* ℌ*butlast.quotients*) + {# *group.iso-class*

(*G Mod* 𝔥 ! (*m* − *1*)) #}
using *multisets*𝔥*butlast*𝔎 *quots*𝔎*notemtpy* by *simp*
also have ... = *mset* (*map group.iso-class* ((*butlast comp*𝔥*.quotients*) @ [*last comp*𝔥*.quotients*]))
using *comp*𝔥*.quotients-butlast comp*𝔥*.last-quotient length*𝔥*big* unfolding *m-def* 𝔥*Pm-def* by *auto*
also have ... = *mset* (*map group.iso-class comp*𝔥*.quotients*) using *quots*𝔥*notempty* by *simp*
finally show *?thesis* .
qed
qed
qed

As a corollary, we see that the composition series of a fixed group all have the same length.

corollary (**in** *jordan-hoelder*) *jordan-hoelder-size*:
shows *length* 𝔊 = *length* 𝔥
proof −
have *length* 𝔊 = *length comp*𝔊*.quotients* + *1* **by** (*metis comp*𝔊*.quotients-length*)
also have ... = *size* (*mset* (*map group.iso-class comp*𝔊*.quotients*)) + *1* **by** (*metis length-map size-mset*)
also have ... = *size* (*mset* (*map group.iso-class comp*𝔥*.quotients*)) + *1*
using *jordan-hoelder-multisets is-group finite is-composition-series comp*𝔥*.is-composition-series* **by** *metis*
also have ... = *length comp*𝔥*.quotients* + *1* **by** (*metis size-mset length-map*)
also have ... = *length* 𝔥 **by** (*metis comp*𝔥*.quotients-length*)
finally show *?thesis*.
qed

end

References

[Ran05] Stuart Rankin. The jordan-hölder theorem, 2005.

[vR14] Jakob von Raumer. Secondary sylow theorems. *Archive of Formal Proofs*, January 2014. http://isa-afp.org/entries/Secondary_Sylow.shtml, Formal proof development.