Eine Formalisierung des zweiten Satzes von Sylow aus der Gruppentheorie in Naproche im Vergleich zu einer Implementierung in Lean

Moritz Hartlieb, Jonas Lippert

11. März 2020

Inhaltsverzeichnis

1 Einleitung

In der vorliegenden Arbeit stellen wir die Formalisierung des zweiten Sylow-Satzes in der Sprache Lean vor und vergleichen sie mit einer eigenen Formalisierung in Naproche.

Zu Beginn seien die wichtigsten Definitionen und Resultate der Gruppentheorie skizziert auf Basis des Skripts „Eine Einführung in die Algebra (Skript, WS 19/20, Bonn)“ von Prof. Dr. Jan Schröer [1].

1.1 Theoretische Grundlagen

Eine **Gruppe** ist eine Menge G zusammen mit einer Abbildung

$$\circ : G \times G \to G$$

$$(g, h) \mapsto g \circ h = gh$$

sodass gilt:

(i) $\circ$ ist assoziativ.

(ii) Es gibt ein neutrales Element $e \in G$, sodass $e \circ g = g = g \circ e$ für alle $g \in G$.

(iii) Für alle $g \in G$ existiert ein inverses Element $g^{-1} \in G$, sodass $g \circ g^{-1} = g^{-1} \circ g = e$.

Eine **Untergruppe** H von G, H ≤ G, ist eine Teilmenge von G mit

(i) $e \in H$.

(ii) H ist abgeschlossen bzgl. $\circ$ und Inversenbildung.

Seien H ≤ G gegeben. Für $g \in G$ ist

$$gH := \{gh \mid h \in H\}$$

die **Nebenklasse** von H zu g. Es bezeichne G/H die Klasse der Nebenklassen von H. Weiter sei

$$[G : H] := |G/H|$$

der **Index** von H zu G.

Zwei Untergruppen H_1 und H_2 von G sind **konjugiert**, falls es ein $g \in G$ gibt mit

$$H_1 = gH_2g^{-1} := \{g\, h\, g^{-1} \mid h \in H\}.$$

Lemma 1.1. *Seien $H \leq G$ gegeben. Für $g_1, g_2 \in G$ gilt:*

(i) $g_1H \cap g_2H \neq \varnothing \iff g_1H = g_2H \iff g_1^{-1}g_2 \in H$

(ii) Für alle $g \in G$ is die Abbildung

$$H \to gH$$
$$h \mapsto gh$$

bijektiv. Insbesondere gilt im endlichen Fall $|H| = |gH|$.

□

Lemma 1.2 (Lagrange). *Seien $H \leq G$ endlich. Dann folgt aus vorherigem Lemma:*

$$|G| = [G : H] \cdot |H|.$$

□

Für eine Gruppe G und eine nichtleere Menge X ist die Abbildung

$$\phi : G \times X \to X$$
$$(g, x) \mapsto g.x$$

eine **Gruppenaktion**, falls gilt:

(i) $1.x = x$ für alle $x \in X$.

(ii) $(gh).x = g.(h.x)$ für alle $g, h \in G$ und $x \in X$.

Weiter definieren wir für $x \in X$:

(i) den **Orbit** $G.x := \{g.x \mid g \in G\}$ von x,

(ii) den **Stabilisator** $G_x := \{g \in G \mid g.x = x\}$ von x,

(iii) die Menge der **Fixpunkte** $X^G := \{x \in X \mid g.x = x \text{ für alle } g \in G\}$ von X.

Lemma 1.3. *Sei G eine Gruppe und X nichtleer. Die Funktion*

$$G/G_x \to G.x$$
$$(g\, G_x) \mapsto g.x$$

ist bijektiv.

□

Wie wir in Lemma 1.1 gesehen haben, ist eine Gruppe G disjunkte Vereinigung der Nebenklassen bzgl. einer Untergruppe $H \leq G$. Zusammen mit Lemma 1.3 erhalten wir, dass X disjunkte Vereinigung der Orbits bzgl. einer Gruppenaktion ist und die Kardinalität der Orbits entsprechend Lagrange die Gruppenordnung teilen müssen.

Lemma 1.4 (Bahnenformel). *Sei G eine endliche Gruppe, sei $X \neq \varnothing$ endlich und seien $x_1, ..., x_n$ gegeben, sodass X disjunkte Vereinigung der $G.x_i$ ist. Dann gilt*

$$|X| = \sum_{i=1}^{n} [G : G_{x_i}] = |X^G| + \sum_{\substack{1 \leq i \leq n \\ x_i \notin X^G}} [G : G_{x_i}].$$

□

Falls G eine p-Gruppe ist, das heißt $|G| = p^r$ für eine Primzahl p und ein $r \in \mathbb{N}$, dann folgt:

Lemma 1.5.

$$|X| \equiv |X^G| \mod p.$$

□

Für eine Gruppe G mit $|G| = p^r m$, wobei p prim und $p \nmid m$, ist die Menge der **p-Sylowgruppen** definiert durch

$$\mathrm{Syl}_p(G) := \{P \leq G \mid \}.$$

Wir können nun den zweiten Satz von Sylow formulieren:

Theorem 1.6. *Sei p eine Primzahl und G eine endliche Gruppe mit $|G| = p^r * m$, sodass $p \nmid m$. Sei $U \leq G$ eine p-Untergruppe, und sei $P \leq G$ eine p-Sylowgruppe. Dann gilt:*

(i) Es gibt ein $g \in G$ mit

$$gUg^{-1} \subseteq P.$$

(ii) Je zwei p-Sylowgruppen sind konjugiert.

Beweis. (i) Setze $X := {}^{G}/_{P}$ und betrachte die Gruppenaktion

$$\phi : U \times X \to X$$

$$(u, gP) \mapsto ugP.$$

Nach Lemma 1.5 gilt

$$|X| = [G : P] = m \equiv |X^G| \mod p.$$

Da $p \nmid m$, gilt $X^G \neq \varnothing$. Es existiert also ein $g \in G$, sodass $ugP = gP$ für alle $u \in U$. Folglich ist $g^{-1}Ug \subseteq P$ und damit U konjugiert zu P bzgl. g^{-1}.

(ii) Dies gilt insbesondere im Falle $U \in \mathrm{Syl}_p(G)$. □

1.2 Vorüberlegungen zur Formalisierung

Es werden also zunächst Grundbegriffe der Gruppentheorie, endliche Mengen sowie natürliche Zahlen, Primzahlen und Modulo-Rechnung benötigt. Hierzu bieten sich unterschiedliche Herangehensweisen an. Es stellt sich heraus, dass Naproche für kleine Theorien gut geeignet ist, deren Grundlagen axiomatisch eingeführt werden, die ihrerseits in einer eigenen Theorie entwickelt werden (können). Die Implementierung in Lean baut hingegen auf bereits formalisierte Grundlagen auf und ist somit Teil einer einzigen großen Theorie der Mathematik.

Interessant ist die Formalisierung von Nebenklassen. Im Sinne einer kleinen Theorie bietet sich in Naproche eine direkte Konstruktion an:

$$Coset\,(g,H,G) \coloneqq \left\{ g *^G h \mid h << H \right\}.$$

Anschließend ist zu zeigen, dass G disjunkte Vereinigung von Nebenklassen bzgl. einer beliebigen Untergruppe H ist. In Lean wird dagegen bzgl. einer Untergruppe S von G folgende Äquivalenzrelation auf G eingeführt:

$$x \sim_S y :\Leftrightarrow x^{-1} * y \in S.$$

Lean erlaubt uns, den Quotient $G/\sim_S$ zu betrachten. Hier wird Lemma 1.1 implizit verwendet.

Im Folgenden wird zunächst auf die jeweiligen Formalisierungen im Detail eingegangen. Anschließend sollen Vor- und Nachteile der jeweiligen Sprachen verglichen und diskutiert werden.

2 Formalisierung in Lean

Die Formalisierung des zweiten Satzes von Sylow in Lean stammt von Chris Hughes[2]. Um mit der aktuellen Version (Stand März 2020) der Leanbibliothek mathlib [3] kompatibel zu sein, musste der Quelltext an einigen Stellen angepasst werden. Die modifizierte Version lässt sich unter `https://github.com/moritz-hl/sylow2` [4] finden.

2.1 Quotienten in Lean

Endliche Mengen und Nebenklassen sind in Lean als Quotient formalisiert. In der core-Library von Lean sind folgende Konstanten definiert:

```
constant quot : Π {α : Sort u}, (α → α → Prop) → Sort u
constant quot.mk :
Π {α : Sort u} (r : α → α → Prop), α → quot r
axiom quot.ind :
∀ {α : Sort u} {r : α → α → Prop} {β : quot r → Prop},
(∀ a, β (quot.mk r a)) → ∀ (q : quot r), β q
axiom quot.sound :
∀ {α : Type u} {r : α → α → Prop} {a b : α},
r a b → quot.mk r a = quot.mk r b
constant quot.lift :
Π {α : Sort u} {r : α → α → Prop} {β : Sort u} (f : α → β)),
(∀ a b, r a b → f a = f b) → quot r → β
```

Die Klasse von `a` in `quot r` wird durch `quot.mk r a` erzeugt. Das Induktionsaxiom stellt sicher, dass alle Elemente von `quot r` von der Form `quot.mk r a` sind. Die Lifting-Eigenschaft erlaubt es, geeignete Funktionen auf `quot r` zu liften.
Ein `setoid α` ist ein Typ α zusammen mit einer Äquivalenzrelation:

```
class setoid (α : Sort u) :=
(r : α → α → Prop) (iseqv : equivalence r)
```

Der `quotient s` auf einem `s : setoid α` ist dann der Quotient bzgl. einer Äquivalenzrelation:

```
def quotient {α : Sort u} (s : setoid α) :=
@quot α setoid.r
```

Die obigen Eigenschaften von `quot` werden anschließend auf `quotient` übertragen.

2.2 Endliche Mengen in Lean

Endliche Mengen werden auf Basis von Listen definiert. Zunächst erhält man Multimengen als Quotienten, indem Permutationen mit Hilfe der Äquivalenzrelation `perm` miteinander identi-

fiziert werden:

```
inductive perm : list α → list α → Prop
| nil   : perm [] []
| skip  : Π (x : α) {l₁ l₂ : list α},
          perm l₁ l₂ → perm (x::l₁) (x::l₂)
| swap  : Π (x y : α) (l : list α), perm (y::x::l) (x::y::l)
| trans : Π {l₁ l₂ l₃ : list α}, perm l₁ l₂ → perm l₂ l₃ → perm l₁ l₃
```

Unter Verwendung des Beweises `perm.eqv`, dass `perm` eine Äquivalenzrelation ist, wird eine Instanz von `setoid list α` eingeführt, welche dann in der Definition der Multimenge Verwendung findet:

```
instance is_setoid (α) : setoid (list α) :=
setoid.mk (@perm α) (perm.eqv α)

def {u} multiset (α : Type u) : Type u :=
quotient (list.is_setoid α)
```

Eine Multimenge wird zur endlichen Menge, wenn sie keine Duplikate beinhaltet:

```
structure finset (α : Type*) :=
(val : multiset α)
(nodup : nodup val)
```

Die Definition von nodup für Multimengen basiert auf nodup für Listen.

```
def nodup : list α → Prop := pairwise (≠)
```

Hierbei prüft `pairwise`, ob die Relation ≠ paarweise gilt.

```
variables (R : α → α → Prop)
inductive pairwise : list α → Prop
| nil {} : pairwise []
| cons : ∀ {a : α} {l : list α}, (∀ a' ∈ l, R a a') → pairwise l →
  pairwise (a::l)
```

Jetzt wird nodup für Listen auf Multimengen geliftet. Dazu werden `quot.lift_on` folgende Parameter übergeben: der Quotient `s`, die zu liftende Funktion `nodup` und ein Beweis, dass Permutation keine Duplikate erzeugt: Aus den zwei Listen `s t` und dem Beweis `p`, dass diese in Relation bzgl. `perm` zueinander stehen, erzeugt `perm_nodup` die Äquivalenz `nodup s ⇔ nodup t`. Dann wird das Lean-interne Axiom `propext` verwendet, nach dem äquivalente Propositionen gleich sind. (Die Rechtsklammerung bei der Funktionseinsetzung wird hier durch $ gewährleistet.)

```
def nodup (s : multiset α) : Prop :=
quot.lift_on s nodup (λ s t p, propext $ perm_nodup p)

def quot.lift_on {α : Sort u} {β : Sort v} {r : α → α → Prop}
(q : quot r) (f : α → β) (c : ∀ a b, r a b → f a = f b) : β

theorem perm_nodup {l₁ l₂ : list α} : l₁ ~ l₂ →
(nodup l₁ ↔ nodup l₂)
```

Der Beweis von `perm_nodup` kann in perm.lean nachgelesen werden. In der Regel wird nicht `finset` direkt verwendet, sondern die Typenklasse `fintype`, sodass der Typ α selber endlich ist.

```
class fintype (α : Type*) :=
(elems : finset α)
(complete : ∀ x : α, x ∈ elems)
```

2.3 MOD in Lean

Die Definition von $x \pmod y$ auf den natürlichen Zahlen basiert auf der Wohlfundiertheit der <-Relation auf $\mathbb{N}$:

```
inductive acc {α : Sort u} (r : α → α → Prop) : α → Prop
| intro (x : α) (h : ∀ y, r y x → acc y) : acc x

parameters {α : Sort u} {r : α → α → Prop}

local infix `<`:50    := r

inductive well_founded {α : Sort u} (r : α → α → Prop) : Prop
| intro (h : ∀ a, acc r a) : well_founded

class has_well_founded (α : Sort u) : Type u :=
(r : α → α → Prop) (wf : well_founded r)
```

Ein Element `x : α` kann nur die Eigenschaft `acc` haben, falls ein „kleinstes“ Element bzgl. der Relation `R` existiert. Entsprechend wird ein Rekursions- und Induktionsprinzip für wohlfundierte Relationen eingeführt, auf die hier nicht näher eingegangen werden soll. Wichtig ist der Spezialfall, dass sich über wohlfundierte Relationen Funktionen definieren lassen (siehe `def fix` weiter unten):

```
parameter hwf : well_founded r
variable {C : α → Sort v}
```

```
variable F : Π x, (Π y, y ≺ x → C y) → C x

def fix_F (x : α) (a : acc r x) : C x :=
acc.rec_on a (λ x₁ ac₁ ih, F x₁ ih)
```

Die Konstruktion des Fixpunktes nimmt gemäß `acc.rec_on` einen Beweis `a` für `acc r x` und liefert `C x`, falls folgender Sachverhalt gegeben ist:

```
(Π (x₁ : α), (∀ (y : α), r y x₁ → acc r y) →
(Π (y : α), r y x₁ → C y) → C x₁)
```

Durch `acc.rec_on` erhalten wir über den Konstruktor `intro`:

```
x₁ : α

ac₁ : ∀ (y : α), r y x₁ → acc r y

ih : Π (y : α), r y x₁ → C y
```

`F` liefert das Gewünschte. Wir können die Fixpunkteigenschaft nun auf wohlfundierte Relationen übertragen:

```
variables {α : Sort u} {C : α → Sort v} {r : α → α → Prop}

def fix (hwf : well_founded r) (F : Π x, (Π y, r y x → C y) → C x) (
  x : α) : C x :=
fix_F F x (apply hwf x)
```

Ist eine Funktion `F` gegeben, welche die Werte `C x` für bekannte Werte `C y` der „kleineren“ Elemente `y ≺ x` liefert, so ist nach `fix` die Funktion `C` für alle `x : α` definiert. Für `F` setzen wir

```
private def mod.F (x : nat) (f : Π x₁, x₁ < x → nat → nat) (y : nat)
  : nat :=
if h : 0 < y ∧ y ≤ x then f (x - y) (div_rec_lemma h) y else x
```

und damit `C` auf $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$. Hier wurde folgendes Lemma verwendet:

```
div_rec_lemma {x y : nat} : 0 <  y ∧ y ≤ x → x - y <  x.
```

Weiter können wir `mod` und damit eine Instanz der Typenklasse `class has_mod` definieren, die bereits in `core.lean` vordefiniert ist. Sie besitzt einzig den Konstruktor `has_mod.mod` : $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

```
protected def mod := fix lt_wf mod.F
```

```
instance : has_mod nat :=
⟨nat.mod⟩
```

Die Infixnotation für `has_mod.mod n m` ist `n % m`. Schließlich wird `a ≡ b [MOD n]` definiert:

```
def modeq (n a b : ℕ) := a % n = b % n

notation a ` ≡ `:50 b ` [MOD `:50 n `]`:0 := modeq n a b
```

2.4 Gruppen in Lean

Gruppen werden sukzessive durch Erweiterungen von `type calsses` definiert:

```
class has_mul (α : Type u) := (mul : α → α → α)

infix * := has_mul.mul

class semigroup (α : Type u) extends has_mul α :=
(mul_assoc : ∀ a b c : α, a * b * c = a * (b * c))

class monoid (α : Type u) extends semigroup α, has_one α :=
(one_mul : ∀ a : α, 1 * a = a) (mul_one : ∀ a : α, a * 1 = a)

class group (α : Type u) extends monoid α, has_inv α :=
(mul_left_inv : ∀ a : α, a⁻¹ * a = 1)
```

Entsprechendes gilt für Untergruppen

```
variables {α : Type*} [monoid α] {s : set α}

class is_submonoid (s : set α) : Prop :=
(one_mem : (1:α) ∈ s)
(mul_mem {a b} : a ∈ s → b ∈ s → a * b ∈ s)

class is_subgroup (s : set α) extends is_submonoid s : Prop :=
(inv_mem {a} : a ∈ s → a⁻¹ ∈ s)
```

und Gruppenaktionen:

```
class has_scalar (α : Type u) (γ : Type v) :=
(smul : α → γ → γ)

infixr ` · `:73 := has_scalar.smul
```

```
class mul_action (α : Type u) (β : Type v) [monoid α] extends
  has_scalar α β :=
(one_smul : ∀ b : β, (1 : α) · b = b)
(mul_smul : ∀ (x y : α) (b : β), (x * y) · b = x · y · b)
```

Es folgen die üblichen Definitionen bzgl. Gruppenaktionen.

```
variables (α) [monoid α] [mul_action α β]

def orbit (b : β) := set.range (λ x : α, x · b)

variables (α) (β)

def stabilizer (b : β) : set α :=
{x : α | x · b = b}

def fixed_points : set β := {b : β | ∀ x, x ∈ stabilizer α b}
```

Nebenklassen werden als Quotient bzgl. der Relation `left_rel` definiert:

```
def left_rel [group α] (s : set α) [is_subgroup s] : setoid α :=
⟨λ x y, x⁻¹ * y ∈ s,
assume x, by simp [is_submonoid.one_mem],
assume x y hxy,
have (x⁻¹ * y)⁻¹ ∈ s, from is_subgroup.inv_mem hxy,
by simpa using this,
assume x y z hxy hyz,
have x⁻¹ * y * (y⁻¹ * z) ∈ s, from is_submonoid.mul_mem hxy hyz,
by simpa [mul_assoc] using this⟩

def left_cosets [group α] (s : set α) [is_subgroup s] : Type* :=
  quotient (left_rel s)
```

Für eine Untergruppe `H` von `G` ist dann `left_rel H` ein `setoid G` mit entsprechender Relation und dem zugehörigen Beweis, dass es sich um eine Äquivalenzrelation handelt. Mithilfe dieser Definition wird eine Instanz von `fintype` bzgl. `left_costes H` erstellt:

```
noncomputable instance [fintype G] (H : set G) [is_subgroup H] :
  fintype (left_cosets H) :=
quotient.fintype (left_rel H)
```

Der Grund dafür, dass die Instanz als `noncomputable` markiert werden muss, ist die Verwendung von `decidable.eq` in `quotient.fintype`. Das geht auf den Umstand zurück,

dass das Bild einer Funktion auf einem endlichen Typ wieder ein endlicher Typ β ist. Hierbei wird das Bild zunächst als Multimenge betrachtet. Anschließend werden eventuelle Duplikate durch den Operator `to_finset` entfernt, was die Entscheidbarkeit der Gleichheitsrelation auf β voraussetzt. Um den 2. Sylowsatz in Lean zu formulieren, fehlen noch die Definitionen des `conjugate_set` und der p-Sylowgruppen:

```
def conjugate_set (x : G) (H : set G) : set G :=
(λ n, x⁻¹ * n * x) ⁻¹' H

class is_sylow [fintype G] (H : set G) {p : ℕ} (hp : prime p) extends
   is_subgroup H : Prop :=
(card_eq : card H = p ^ dlogn p (card G))

lemma sylow_2 [fintype G] {p : ℕ} (hp : nat.prime p)
(H K : set G) [is_sylow H hp] [is_sylow K hp] :
∃ g : G, H = conjugate_set g K
```

`dlogn p (card G)` ist die Vielfachheit von `p` in `card G`.

`#print axioms sylow.sylow_2` zeigt, dass `propext`, `quot.sound` und `calssical.choice` verwendet wird.

`quot.sound` wird benötigt, um zu beweisen, dass das kanonische Operieren einer Untergruppe `H` auf den Nebenklassen einer Untergruppe `K` eine Gruppenaktion ist:

```
def mul_left_cosets (L₁ L₂ : set G) [is_subgroup L₂] [is_subgroup L₁]
(x : L₂) (y : left_cosets L₁) : left_cosets L₁ :=
quotient.lift_on y (λ y, ⟦(x : G) * y⟧)
(λ a b (hab : _ ∈ L₁), quotient.sound
(show _ ∈ L₁, by rwa [mul_inv_rev, ← mul_assoc, mul_assoc (a⁻¹),
  inv_mul_self, mul_one]))
```

Dafür soll die Funktion `(λ y,⟦(x : G) * y⟧)` auf Nebenklassen geliftet werden. Für zwei Elemente `a` und `b`, die bzgl. `left_rel` äquivalent sind, ist also zu zeigen, dass

```
⟦(x : G) * a⟧ = ⟦(x : G) * b⟧.
```

Mit `quot_sound` genügt dann ein Beweis dafür, dass die erzeugenden Elemente äquivalent sind. Die nötigen Umformungen erledigt `simp`.

Im Beweis von Sylow 2 werden zunächst die vorangegangenen Resultate verwendet, um herzuleiten, dass die Anzahl der Fixpunkte bzgl. obiger Aktion ungleich Null ist. An dieser Stelle liefert `classical.choice` einen solchen Fixpunkt, ohne den die Konstruktion eines geeigneten `conjugate_set` nicht möglich wäre.

2.5 Sylow 2 in Lean

Wir wollen nun auf einige Details im Beweis des Satzes eingehen. Die verwendeten Lemmata, die für das sukzessive Umformen via `rw` oder `simp` genutzt werden, sind dabei weder mathematisch noch hinsichtlich des Formalisierungsprozesses besonders interessant. Es werden zahlreiche technische Lemma über natürliche Zahlen, Kardinalitäten, Funktionen, etc. zusammengetragen, deren Beweise für den Menschen trivial und in Lean im Nachhinein teilweise nur sehr schwer verständlich sind. Es ist prinzipiell klar, dass es auf Basis der konstruierten Begriffe und Strukturen möglich ist, entsprechende Beweise für Lean verständlich zu formulieren und es genügt in der Regel zu wissen, dass sie existieren.

Die ersten drei Hilfslemma liefern uns den gewünschten Fixpunkt.

```
lemma sylow_2 [fintype G] {p : ℕ} (hp : nat.prime p)
(H K : set G) [is_sylow H hp] [is_sylow K hp] :
∃ g : G, H = conjugate_set g K :=

have hs : card (left_cosets K) = card G / (p ^ dlogn p (card G)) :=
(nat.mul_right_inj (pos_pow_of_pos (dlogn p (card G)) hp.pos)).1
$ by rw [ ← card_sylow K hp, ← card_eq_card_cosets_mul_card_subgroup,
  card_sylow K hp,
nat.div_mul_cancel (dlogn_dvd _ hp.1)],

have hmodeq : card G / (p ^ dlogn p (card G)) ≡ card (fixed_points H (
  left_cosets K)) [MOD p] :=
eq.subst hs (mul_action.card_modeq_card_fixed_points hp (card_sylow H
  hp)),

have hfixed : 0 < card (fixed_points H (left_cosets K)) :=
nat.pos_of_ne_zero
(λ h, (not_dvd_div_dlogn (fintype.card_pos_iff.2 ⟨(1 : G)⟩) hp.1)
(by rwa [h, nat.modeq.modeq_zero_iff] at hmodeq)),
```

Zunächst wird in `hs` die Kardinalität der Klasse der Nebenklassen auf Basis der Definition von Sylow-Gruppen umgeformt. Hierbei wird das Lemma von Lagrange unter dem Namen `card_eq_card_cosets_mul_card_subgroup` benutzt.

In `hmodeq` kommt die Lean-Version von Lemma (??) zur Anwendung:

```
lemma card_modeq_card_fixed_points [fintype α] [fintype G]
[fintype (fixed_points G α)]
{p n : ℕ} (hp : nat.prime p) (h : card G = p ^ n) :
card α ≡ card (fixed_points G α) [MOD p]
```

Durch Substitution wird das allgemeine Resultat auf den Spezialfall übertragen.

Mit Hilfe von `hmodeq` lässt sich schließlich `hfixed` zeigen, dass nämlich die Kardinalität der Fixpunkte größer Null ist. Nach `nat.pos_of_ne_zero` genügt es zu zeigen, dass die

Kardinalität der Fixpunkte ungleich Null ist. Es wird also ein Beweis von `false` gefordert unter Annahme `h`, die Kardinalität sei gleich Null. Der Widerspruch wird erzeugt, in dem erst

```
lemma not_dvd_div_dlogn {p a : ℕ} (ha : a > 0) (hp : p > 1) :
¬p ∣ a / (p ^ dlogn p a)
```

auf `a = card G` angewendet wird. Dass `p` aber ein Teiler der rechten Seite sein muss, folgt aus der Annahme `h` zusammen mit

```
modeq_zero_iff : b ≡ 0 [MOD n] ↔ n ∣ b,
```

wobei `b = card G / (p ^ dlogn p a)`.

Um aus dem abstrakten Kardinalitäts-Argument einen konkreten Fixpunkt zu gewinnen, wird das Axiom `classical.choice` in Form des Lemmas `fintype.card_pos_iff` verwendet:

```
let ⟨⟨x, hx⟩⟩ := fintype.card_pos_iff.1 hfixed in

begin

revert hx
```

Die Taktik `revert` generalisiert den Beweis `hx`, dass `x` ein Fixpunkt ist, sodass nun folgendes zu beweisen ist:

```
x ∈ fixed_points H (left_cosets K) →
(∃ (g : G), H = conjugate_set g K)
```

Der Grund dafür ist der, dass wir für die weitere Argumentation einen Vertreter `g : G` für die Äquivalenzklasse `x` benötigen. Dies wird möglich durch das Axiom `quot.ind` in Form von `quotient.induction_on`, nachdem es genügt, das Ziel für einen beliebigen Vertreter zu zeigen:

```
refine quotient.induction_on x
(λ g hg, ⟨g, set.eq_of_card_eq_of_subset _ _⟩)
```

Für ein Element `g : G` und unter der Annahme `hg`, dass ⟦`g`⟧ ein Fixpunkt ist, bleibt nun die ursprünglich Aussage `∃(g : G), H = conjugate_set g K` zu zeigen. Dem anonymen Konstruktor wurde zur Konstruktion der Existenzaussage das Element `g` und folgendes Lemma übergeben:

```
lemma eq_of_card_eq_of_subset {s t : set α} [fintype s] [fintype t]
(hcard : card s = card t) (hsub : s ⊆ t) : s = t
```

Durch die Verwendung der Taktik `refine` und der zwei Unterstriche anstelle der Beweise `hcard` und `hsub`, werden diese Bedingungen als neue Ziele innerhalb des Taktik-Blocks erstellt, wobei das ursprüngliche Ziel als bewiesen gilt. Die beiden Beweise seien hier der Vollständigkeit halber aufgeführt, auch wenn die technischen Details nicht weiter interessant sind:

```
{
rw [conjugate_set_eq_image, set.card_image_of_injective _
  conj_inj_left,
card_sylow K hp, card_sylow H hp] },
{
assume y hy,
have : (y⁻¹ * g)⁻¹ * g ∈ K :=
quotient.exact ((mem_fixed_points' (left_cosets K)).1 hg ⟦y⁻¹ * g⟧
⟨⟨y⁻¹, inv_mem hy⟩, rfl⟩),
simp [conjugate_set_eq_preimage],
simp only [*, mul_assoc, mul_inv_rev] at *,
simp [*, inv_inv] at *}
end
```

3 Formalisierung in Naproche

3.1 Struktur

Die Formalisierung in Naproche teilt sich in 8 Dateien auf:

Name	Inhalt	Abschnitt
01basicgrouptheory.ftl	Gruppentheorie	3.2
02numbers.ftl	Zahlen	3.6
03cards.ftl	Kardinalitäten	3.9
04lagrange.ftl	Satz von Lagrange	3.10
05staborb.ftl	Gruppenaktionen	3.11
06fixedpointsmodp.ftl	Fixpunkte	3.13
07grpaction.ftl	SylowGruppenaktion	3.14
08sylow2.ftl	Der Satz von Sylow	3.15

In jeder Datei werden zu Beginn jeweils die Aussagen und Definitionen der vorangegangen Dateien eingebunden. Hierfür wurden Hilfsdateien `xxExport.ftl` erstellt, in denen zu jeder Datei die daran enthalten Definitionen und Theoreme in Form von Axiomen ohne Beweise festgehalten wurden. Grundsätzlich wäre es möglich diese Dateien über den Befehl `[read dateiname.ftl]` einzulesen, da dieser Befehl jedoch in der jetzigen Version von Naproche nicht auf allen Betriebssystemen gleichermaßen funktioniert, haben wir die Dateien manuell kopiert und dies durch Kommentare im Quelltext vermerkt.

Die Formalisierung ließ sich auf einem MacBook Pro (16 GB RAM, Intel i7 2,7 GhZ) von 2018 innerhalb der Isabelle-Naproche Umgebung[5] in rund 5 Minuten verifizieren. Fügt man jedoch alle Dateien aneinander, ohne die Theoreme jeweils durch Axiome zu ersetzen, dann ist Naproche nicht mehr in der Lage, den Beweis vollständig zu verifizieren.

Der Dateien finden sich unter `https://github.com/moritz-hl/sylowftl` [6].

3.2 Grundlegende Definitionen

Zunächst führen wir einige grundlegenden Definitionen ein. Interessant ist hier insbesondere der Begriff der `disjunct collection` den wir einführen um später über disjunkte Vereinigung ”sprechen” zu können.

```
Let M, N denote sets.
Let x << M stand for x is an element of M.

Definition.
Prod(M,N) = { (x,y) | x << M and y << N }.

Definition.
A subset of M is a set N such that every element of N is an element of M.

Definition.
Let M be a set.
M is empty iff there is no element x of M such that x = x.
```

```
Definition.
Let M be a set such that for all elements N of M N is a set.
\-/ M = {x | There is an element N of M such that x is an element of N}.

Definition.
Let N1, N2 be sets.
N1 \-/ N2 = {x | x is an element of N1 or x is  an element of N2}.

Definition.
Let N1 be a set.
Let N2 be a subset of N1.
N1 \\ N2 = {x | x is an element of N1 and (x is not an element of N2)}.

Definition.
Let N1, N2 be a sets.
N1 and N2 are disjunct iff there is no element x of N1 such that x is an element of
    N2.

Definition.
A disjunct collection is a set M such that
(for all elements N of M N is a set) and for all elements N1, N2 of M (N1 = N2 or (
    N1 and N2 are disjunct)).

Definition.
Let f be a function. Let M,N be sets. f is from M to N iff Dom(f) = M and for every
    element x of M f[x] is an element of N.

Definition.
Let f be a function. Range(f) = {f[x] | x << Dom(f)}.

Definition.
Let f be a function. f is injective iff for all elements x,y of Dom(f) we have (x!=y
    => f[x] != f[y]).

Definition.
Let f be a function. f is surjective onto M iff
(f is from Dom(f) to M
and for every y << M there is an element x of Dom(f) such that f[x]=y).
```

3.3 Einführung des Gruppenbegriffs

Der Gruppenbegriff wird abstrakt als `Notion` eingeführt. Für diese `Notion` werden die Existenz einer Trägermenge `El(G)`, die Existenz der Eins, Inversen und Verknüpfung sowie deren Eigenschaften axiomatisch gefordert. Die Verknüpfung `a *^{G} b` muss hier explizit als Verknüpfung auf der Gruppe `G` gekennzeichnet werden.

```
[synonym group/-s]
[synonym subgroup/-s]

Signature.
A group is a notion.

Let G denote a group.

Signature.
El(G) is a  set.

Signature.
One(G) is an object.

Axiom.
One(G) << El(G).

Signature.
Let a, b be elements of El(G).
a *^{G} b is an element of El(G).

Signature.
Let a be an element of El(G).
Inv(a, G) is an element of El(G).

Axiom Assoc.
Let x, y, z be elements of El(G). x *^{G} ( y *^{G} z) = (x *^{G} y) *^{G} z.

Axiom InvOne.
Let x be an element of El(G). x *^{G} Inv(x, G) = One(G) = Inv(x, G) *^{G} x.

Axiom MulOne.
Let x be an element of El(G). x *^{G} One(G) = x =  One(G) *^{G} x.

Lemma InvUniq.
Let x, y be elements of El(G).
If x *^{G} y = One(G) then y = Inv(x, G).

Lemma OneUniq.
Let x, y be elements of El(G).
If x *^{G} y = x then y = One(G).
```

3.4 Untergruppen

Untergruppen werden als Teilmengen von `El(G)` eingeführt. Im Beweis des zweiten Satzes von Sylow wird genutzt, dass Untergruppen selbst wieder eine Gruppenstruktur annehmen. Da wir den Begriff der Gruppe abstrakt eingeführt haben, ist es hier nicht möglich, zu beweisen,

dass `Gr(U, G)` eine Gruppe ist, weswegen dies hier als Axiom gefordert wird.

```
Definition.
A subgroup of G is set H such that
(H is a subset of El(G))
and (One(G) << H)
and (for every x << H Inv(x, G) << H)
and (for all elements x, y of H x *^{G} y << H).

Definition.
Let U be a subgroup of G.
Gr(U, G) is a group H such that
(El(H) = U)
and (One(H) = One(G))
and (for every x << U Inv(x, H) = Inv(x, G))
and (for all elements x, y of U x *^{Gr(U, G)} y = x *^{G} y).

Lemma.
Let G be a group.
Let H be a subset of El(G).
Assume ((There is a x << H such that x = x) and (for all elements  y, z of H  z *^{G}
     Inv(y, G) << H)).
H is a subgroup of G.
Proof.
  One(G) << H.
    Proof.
      Take x << H.
      Then One(G) = x *^{G} Inv(x, G).
      Thus One(G) << H.
    end.

  For every x << H Inv(x, G) << H.
    Proof.
      Let x be an element of H.
      Then Inv(x, G) = One(G) *^{G} Inv(x, G).
      Thus Inv(x, G) << H.
    end.

  For all elements x, y of H x *^{G} y << H.
  Proof.
    Let x, y be elements of H.
    Then Inv(x, G) << H.
    x *^{G} y = x *^{G}  Inv(Inv(y, G), G).
    Hence x *^{G} y << H.
    end.
Qed.
```

3.5 Nebenklassen

Wir führen Nebenklassen im Gegensatz zum Ansatz der Formalisierung in Lean nicht über Qoutienten ein, sondern über die mengenbasierte Definition. Das in der Einführung genannte Lemma begründet, dass diese Ansätze das gleiche metatheoretische Objekt formalisieren.

```
Definition.
Let g be an element of El(G).
Let H be a subgroup of G.
Coset(g, H, G) = {g *^{G} h | h << H}.

Lemma.
Let H be a subgroup of G.
Let g1, g2 be elements of El(G).
Assume Coset(g1, H, G) and Coset(g2, H, G) are not disjunct.
Inv(g2, G) *^{G} g1 << H.
Proof.
  Take y << El(G) such that (y << Coset(g1, H, G) and y << Coset(g2, H, G)).
  Take b << H such that y = g1 *^{G} b.
  Take c << H such that y = g2 *^{G} c.

  We have g1 = y *^{G} Inv(b, G).
          g2 = y *^{G} Inv(c, G).
          Inv(g2, G) = c *^{G} Inv(y, G).
          Inv(y, G) *^{G} g1 = Inv(b, G).

  Therefore Inv(g2, G) *^{G}  g1 = c *^{G} (Inv(b, G)).
qed.


Lemma.
Let H be a subgroup of G.
Let g1, g2 be elements of El(G).
If Inv(g2, G) *^{G} g1 << H
Then Coset(g1, H, G) = Coset(g2, H, G).
Proof.
Assume Inv(g2, G) *^{G} g1 << H.

Every element of Coset(g1, H, G) is an element of Coset(g2, H, G).
Proof.
  Let y be an element of Coset(g1, H, G).
  Take a << H such that y = g1 *^{G} a.
  (Inv(g2, G) *^{G}  g1) *^{G}  a << H.
  Then y = g1 *^{G}  a = g2 *^{G} ((Inv(g2, G) *^{G}  g1) *^{G}  a).
end.

Every element of Coset(g2, H, G) is an element of  Coset(g1, H, G).
Proof.
```

```
  Let y be an element of Coset(g2, H, G).
  Take a << H such that y = g2 *^{G} a.
  (Inv(g2, G) *^{G}  g1) *^{G}  a << H.
  Then y = g2 *^{G} a =g1 *^{G} ((Inv(g1, G)*^{G} g2)*^{G} a).
end.

Therefore Coset(g1, H, G) = Coset(g2, H, G).
Qed.

Lemma CosEq.
Let H be a subgroup of G.
Let g1, g2 be elements of El(G).
If Coset(g1, H, G) and  Coset(g2, H, G) are not disjunct
then Coset(g1, H, G) = Coset(g2, H, G).

Lemma.
Let H be a subgroup of G.
Let g1, g2 be elements of El(G).
Inv(g2, G)*^{G} g1 << H iff Coset(g1, H, G) = Coset(g2, H, G).

Definition.
Let H be a subgroup of G.
Cosets(H, G) = {Coset(g, H, G) | g << El(G)}.

[synonym coset/-s]
Let a coset of H in G denote an element of Cosets(H, G).

Lemma.
Let U be a subgroup of G.
El(G) = \-/ Cosets(U, G).
Proof.
Let us show that every element of El(G) is an element of \-/ Cosets(U, G).
  Let g be an element of El(G).
  g is an element of Coset(g, U, G).
end.

Let us show that every element of \-/ Cosets(U, G) is an element of El(G).
  Let h be an element of \-/ Cosets(U, G).
  Take an element k of El(G) such that h is an element of Coset(k, U, G).
  Coset(k, U, G) is a subset of El(G).
  Hence h is an element of El(G).
end.

Therefore El(G) = \-/ Cosets(U, G).
Qed.

Lemma.
Let G be a group.
```

```
Let U be a subgroup of G.
Cosets(U, G) is a disjunct collection.
Proof.
Let us show that for every elements N1, N2 of Cosets(U, G) N1 = N2 or (N1 and N2 are
    disjunct).
  Let N1, N2 be cosets of U in G.
  Take elements g1, g2 of El(G) such that N1 = Coset(g1, U, G) and N2 = Coset(g2, U,
    G).
  If N1 and N2 are not disjunct then N1 = N2 (by CosEq).
  Therefore the thesis.
end.
Qed.

Lemma.
Let G be a group.
Let U be a subgroup of G.
U is a coset of U in G.
Proof.
  We have U = Coset(One(G), U, G).
  Therefore the thesis.
Qed.

Definition.
Let g be an element of El(G).
Let U be a subgroup of G.
Conjugate(g, U, G) = {(g *^{G} (u *^{G} Inv(g, G)))| u is an element of U}.

Definition.
Let U, V be subgroups of G.
U and V are conjugates in G iff there is an element g of El(G) such that U =
    Conjugate(g, V, G).
```

3.6 Zahlen

Da wir uns mit endlichen Gruppen, also insbesondere mit Kardinalitäten beschäftigen, führen wir nun die ganzen Zahlen ein. Herkömmliche Texte über die Sylowsätze [1] behandeln Fragen der elementaren Zahlentheorie nicht in voller Tiefe, sondern setzen eine gewisse mathematischen Vorkenntnis voraus. In diesem Sinne führen auch wir die Zahlen abstrakt ein und fordern dann bekannte Eigenschaften axiomatisch.

```
 [synonym integer/-s]
 [synonym number/-s]

 Signature Integers. An integer is a notion.

 Signature Naturals. A natural number is an integer.

 Let a,b,c,d,e,n,m stand for integers.
```

```
Signature NatZero.  0 is a natural number.
Signature NatOne.   1 is a natural number.

Signature IntNeg.   -a is an integer.
Signature IntPlus.  a + b is an integer.
Signature IntMult.  a * b is an integer.

   Signature NatPot. Let b be a natural number.  a ^ b is an integer.

Axiom NatPlus. If a and b are natural numbers then a + b is a natural number.
Axiom NatMult. If a and b are natural numbers then a * b is a natural number.

Signature NatLT. a < b is an atom.

Let a is smaller than b stand for a < b.

Axiom TriCh.
a = b \/ a < b \/ b < a.

Axiom.
a < b iff a != b.

Let a - b stand for a + (-b).

Axiom NatSub.
If a < b then b - a is natural number.

Axiom AddAsso.      a + (b + c) = (a + b) + c.
Axiom AddComm.      a + b = b + a.
Axiom AddZero.      a + 0 = a = 0 + a.
Axiom AddNeg.       a - a = 0 = -a + a.

Axiom MulAsso.      a * (b * c) = (a * b) * c.
Axiom MulComm.      a * b = b * a.
Axiom MulOne.       a * 1 = a = 1 * a.

Axiom Distrib.      a * (b + c) = (a*b) + (a*c) and
     (a + b) * c = (a*c) + (b*c).

Axiom ZeroDiv.      a != 0 /\ b != 0 => a * b != 0.

Axiom PotInj.   Let p be an integer. Let n,m be natural numbers. (p ^ n = p ^  m) =>
    n = m.
Axiom PotAdd. Let p be an integer. Let n, m be natural numbers. p ^ (n + m) = (p ^ n
   ) * (p  ^ m).
Axiom PotNotZero. Let p be an integer. Let k be a natural number. p ^ k  != 0.
```

```
Lemma MulZero.      a * 0 = 0 = 0 * a.
Proof.
  a*(1+(-1)) = (a*1)+(a*(-1))=0.
Qed.

Lemma MulMinOne.    (-1) * a = -a = a * -1.
Proof.
  a+(-1 * a)= (1*a)+(-1 * a) = 0.
Qed.

Lemma IntCanc.
c != 0 /\ a * c = b * c => a = b.
Proof.
  Assume c != 0 /\ a * c = b * c.

  (1) (a+ (-b))*c = (a * c) + ((-b) * c) = 0.

  Therefore a - b = 0 (by ZeroDiv, 1).
Qed.

Let a is nonzero stand for a != 0.
Let p,q stand for nonzero integers.

[synonym divisor/-s] [synonym divide/-s]

Definition Divisor. A divisor of b is a nonzero integer a
     such that for some n (a * n = b).

Let a divides b stand for a is a divisor of b.
   Let a | b stand for a is a divisor of b.

   Axiom.
Let k be a natural number.
k != 0 => a | a^k.

Lemma DivPlus.
q | a /\ q | b =>  q | (a + b).

Definition EquMod.  a = b (mod q) iff q | a-b.

Definition NeqMod.  a != b (mod q) iff not (a = b (mod q)).

Lemma EquModRef.    a = a (mod q).

[ontored on]
Lemma EquModSym.    a = b (mod q) => b = a (mod q).
Proof.
```

```
 Assume that a = b (mod q).
 (1) Take n such that q * n = a - b.
 q * -n .= (-1) * (q * n) (by MulMinOne, MulAsso,MulComm,MulBubble)
       .= (-1) * (a - b) (by 1).
 Therefore q | b-a.
qed.

Lemma EquModTrn.    a = b (mod q) /\ b = c (mod q) => a = c (mod q).
Proof.
 Assume that a = b (mod q) /\ b = c (mod q).
 Take n such that q * n = a - b.
 Take m such that q * m = b - c.
 We have q * (n + m) = a - c.
qed.

Lemma EquModMul. a = b (mod p * q) => a = b (mod p) /\ a = b (mod q).
Proof.
 Assume that a = b (mod p * q).
 Take m such that (p * q) * m = a - b.
 We have p * (q * m) = a - b = q * (p * m).
qed.
[/ontored]
```

3.7 Primzahlen

Auch den Begriff der Primzahl führen wir abstrakt ein. Bemerkenswert ist hier, dass nicht auf die herkömmliche Definition der Primzahlen zurückgegriffen wird, sondern im gesamten Beweis des zweiten Satzes von Sylow nur die durch das folgende Axiom geforderte Eigenschaft benötigt wird.

```
Signature Prime.    a is prime is an atom.

Let a prime stand for a prime nonzero integer.

Axiom.
Let n be a natural number.
Let p be a prime.
Let k be a natural number.
If k | p^n then k = 1 or p | k.
```

3.8 Ein Lemma über Potenzen

Das folgende Lemma wird eine zentrale Rolle im Beweis des zweiten Teiles des Satzes einnehmen. Wir liefern hier einen Beweis, der sich durch ein Symmetrieargument vereinfachen ließe, welches sich jedoch in Naproche nicht abbilden lässt.

```
Lemma DLogN.
Let p be a prime.
Let a, b be natural numbers.
If n = (p^a)*c /\ n = (p^b)*d and p does not divide c and p does not divide d then a
    = b.
Proof.
Assume n = (p^a)*c and n = (p^b)*d and p does not divide c and p does not divide d.

b is not smaller than a.
Proof by Contradiction.
  Assume b < a.
  We have p^a = (p^(a-b))*(p^b).

  (1) (p^a)*c = (p^b)*d.
  (2) ((p^(a-b))*(p^b))*c = (p^b)*d.
  (3) ((p^b)*(p^(a-b)))*c = (p^b)*d (by 1, MulComm).
  (4) (p^b)*((p^(a-b))*c) = (p^b)*d (by 3, MulAsso).
  (5) ((p^(a-b))*c)*(p^b) = d*(p^b) (by 4, MulComm).

  (6)((p^(a-b))*c) = d (by 5, IntCanc, PotNotZero).

  a-b != 0.

  p is a divisor of p^(a-b).
  p is a divisor of ((p^(a-b))*c).

  p does divide d.
  p does not divide  d.

  Contradiction.
end.

a is not smaller than b.
Proof by Contradiction.
  Assume a < b.
  We have p^b = (p^(b-a))*(p^a).

  (1) (p^b)*d = (p^a)*c.
  (2) ((p^(b-a))*(p^a))*d = (p^a)*c.
  (3) ((p^a)*(p^(b-a)))*d = (p^a)*c (by 1, MulComm).
  (4) (p^a)*((p^(b-a))*d) = (p^a)*c (by 3, MulAsso).
  (5) ((p^(b-a))*d)*(p^a) = c*(p^a) (by 4, MulComm).
```

```
  (6)((p^(b-a))*d) = c (by 5, IntCanc, PotNotZero).

  b-a != 0.

  p is a divisor of p^(b-a).
  p is a divisor of ((p^(b-a))*d).

  p does divide c.
  p does not divide  c.

  Contradiction.
end.

Therefore the thesis.
qed.
```

3.9 Endliche Mengen

Endliche Mengen führen wir als abstrakten Begriff ein und fordern dann axiomatisch die bekannten Eigeschaften. Interessant sind hier die Axiome `cardUnion0-2` bezüglich der Vereinigung von Gruppen. Da wir die ganze Zahlen abstrakt und insbesondere ohne ein Induktionsaxiom eingeführt haben, müssen wir diese drei Axiome hier seperat fordern, obwohl sie auf Metaebene aus `cardUnion0` gefolgert werden können.

```
Signature.
A finite set is a set.

Axiom.
Let M be a finite set.
Let N be a subset of M.
N is a finite set.

Axiom.
Let f be a function such that Dom(f) is a finite set.
Range(f) is a finite set.

Axiom.
Let M, N be finite set.
Prod(M, N) is a finite set.

Signature.
Let M be a finite set.
card(M) is a natural number.

Axiom.
Let M be a finite set.
```

```
Let N be a subset of M.
If card(M) = card(N) then M = N.

Axiom.
Let M be a set such that for all elements N of M N is a finite set.
\-/ M is a finite set.

Axiom.
Let N1, N2 be finite sets.
N1 \-/ N2 is a finite set.

Axiom cardUnion0.
Let N1, N2 be finite sets.
If N1 and N2 are disjunct then card(N1 \-/ N2) = card(N1) + card(N2).

Axiom cardUnion1.
Let M be a set.
Let N be an element of M.
If M is a finite set such that every element of M is a finite set
and M is a disjunct collection
and for all elements N1, N2 of M card(N1) = card(N2)
then card(\-/M) = card(N)*card(M).

Axiom cardUnion2.
Let M be a set.
Let k be an integer.
If M is a finite set such that every element of M is a finite set
and M is disjunct collection
and for all elements N of M k | card(N)
then k  |card(\-/ M).

Axiom.
Let N1, N2 be finite sets.
card(N1) = card(N2) iff there is a function f such that (f is from N1 to N2 and f is
    injective and f is surjective onto N2).

Axiom.
Let M be a finite set.
If card(M) != 0 then M is not empty.

Axiom.
Let M be a finite set.
card(M) = 1 iff there is y <<  M such that for all x << M x = y.

Axiom.
Let M be a finite set.
Assume 1 < card(M).
Let x be an element of M.
```

```
There is y << M such that x != y.
```

3.10 Der Satz von Lagrange

Der Satz von Lagrange ist ein grundlegender Satz der endlichen Gruppentheorie.

```
Definition.
A finite group is a group G such that El(G) is a finite set.

Lemma.
Let G be a finite group.
Let U be a subgroup of G.
Cosets(U, G) is a finite set.
Proof.
  Define f[g] = Coset(g, U, G) for g in El(G).
  Cosets(U, G) is a subset of  Range(f).
  Therefore the thesis.
Qed.


Definition.
Let G be a finite group.
Let U be a subgroup of G.
Index(G, U) = card(Cosets(U, G)).

Lemma.
Let G be a finite group.
Let U, V be subgroups of G such that V and U are conjugates in G.
card(U) = card(V).
Proof.
Take an element g of El(G) such that V = Conjugate(g, U, G).
Define f[u] = g *^{G} (u *^{G} Inv(g, G)) for u in U.
Let us show that f is from U to V.
  Dom(f) = U.
  Let us show that for all elements u of U f[u] is an element of V.
    Let u be an element of U.
    f[u] is an element of V.
  end.
end.

Let us show that f is injective.
  Let us show that for all elements u1, u2 of U if f[u1] = f[u2] then u1 = u2.
     Let u1, u2 be elements of U such that f[u1] = f[u2].
     We have u1 = (Inv(g, G) *^{G}  (g *^{G} (u1 *^{G} Inv(g, G)))) *^{G} g.
     We have u2 = (Inv(g, G) *^{G}  (g *^{G} (u2 *^{G} Inv(g, G)))) *^{G} g.
     Therefore u1 = (Inv(g, G) *^{G}  f[u1]) *^{G} g = (Inv(g, G) *^{G}  f[u2]) *^{G}
     g = u2.
```

```
  end.
end.

Let us show that f is surjective onto V.
  Let us show that for every element v of V there is an element u of U such that f[u]
     = v.
    Let v be an element of V.
    We can take an element u of U such that v = (g *^{G} u) *^{G} Inv(g, G).
    Hence v = f[u].
  end.
end.
qed.

Theorem Lagrange.
Let G be a finite group.
Let U be a subgroup of G.
card(El(G)) = card(U)*card(Cosets(U, G)).
Proof.
Let us show that for all elements g of El(G) card(Coset(g, U, G)) = card(U).
  Let g be an element of El(G).
  Define f[u] = g *^{G} u for u in U.
  f is from U to Coset(g, U, G).
  f is injective.
  Proof.
    Let us show that for all  elements u1, u2 of U If f[u1] = f[u2] then u1 = u2.
      Let u1, u2 be elements of U such that f[u1] = f[u2].
      We have u1 = Inv(g, G) *^{G} (g *^{G} u1) = Inv(g, G) *^{G} (g *^{G} u2) = u2.
      Thus u1 = u2.
    end.

    Therefore the thesis.
  end.

  f is surjective onto Coset(g, U, G).
  Proof.
    Let us show that for every element y of Coset(g, U, G) there is an element u of U
     such that f[u] = y.
      Let y be an element of Coset(g, U, G).
      Take an element u of U such that y = g*^{G} u.
      Then f[u] = y.
    end.

    Therefore the thesis.
  end.
end.

(1) Cosets(U, G) is a disjunct collection and for all elements N1, N2 of Cosets(U, G)
     card(N1) = card(N2).
```

```
(2) Cosets(U, G) is a finite set such that for all element N1 of Cosets(U, G) N1 is a
     finite set.
(3) U is an element of Cosets(U, G).

Therefore card(\-/ Cosets(U, G)) = card(U)*card(Cosets(U, G)) (by cardUnion, 1, 2, 3)
    .
Qed.
```

3.11 Gruppenaktionen

Gruppenaktionen betrachten wir als Funktionen von `Prod(El(G), M)` nach `M` mit zusätzlichen Eigenschaften.

```
Definition.
Let M be a set.
Let G be a group.
A group action from G on M is a function f
such that f is from Prod(El(G), M) to M
and (for every element x of M f[(One(G), x)] = x)
and for every element x of M for all elements a, b of El(G)
f[((a *^{G}  b), x)] = f[(a, f[(b, x)])].

Definition.
Let M be a set.
Let G be a group.
Let f be a function from Prod(El(G), M) to M.
Let x be an element of M.
Orbit(x, f, M, G) = { f[(a, x)] | a << El(G)}.


Definition.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
Let x << M.
Stab(x,f, M, G) = {y | y << El(G) and f[(y, x)] = x}.

Lemma.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
Let x << M.
Stab(x,f,M, G) is a subgroup of G.
Proof.

One(G) is an element of Stab(x, f, M, G).
```

```
Let us show that for all elements y, z of Stab(x, f, M, G)  z *^{G} Inv(y, G) << Stab
    (x, f, M, G).
  Let y, z be elements of Stab(x, f, M, G).
  We have f[(z *^{G} Inv(y, G), x)] = x.
  Therefore z *^{G} Inv(y, G) is an element of Stab(x, f, M, G).
end.

Therefore the thesis.
Qed.

Lemma.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
Let x << M.
Let g, h be elements of El(G).
If Coset(g, Stab(x, f, M, G), G) = Coset(h, Stab(x, f, M, G), G) then f[(g, x)] = f[(
    h, x)].
```

3.12 Stabilisator und Orbit

Außerhalb von Beweisen lassen sich in Naproche keine Funktionen definieren. In der herkömmlichen Mathematik werden Funktionen oft definiert und erst nachträglich auf ”Wohldefiniertheit” bzw. Repräsentantenunabhängigkeit überprüft. Man könnte diesen Vorgang über einen Relationsbegriff und weitere Axiome formalisieren, in diesem Text belassen wir es jedoch dabei – auf Metaebene begründet durch das vorangegange Lemma – die Existenz der folgenden Funktion als Axiom zu fordern.

```
Axiom.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
Let x << M.
There is a function h such that
h is from Cosets(Stab(x, f, M, G), G) to Orbit(x, f, M, G)
and (for all elements i of El(G) h[Coset(i,Stab(x, f, M, G),G)] = f[(i, x)]).

Lemma.
Let G be a finite group.
Let f be a group action from G on M.
Let x << M.
Orbit(x,f, M, G) is a finite set.
Proof.
  Define h[g] = f[(g, x)] for g in El(G).
  Dom(h) is a finite set.
  Orbit(x, f, M, G) is a subset of Range(h).
  Proof.
```

```
    Let us show that every element of Orbit(x, f, M, G) is an element of Range(h).
       (1) Let y be an element of Orbit(x, f, M, G).

       We can take an element g1 of El(G) such that y = f[(g1, x)] (by 1).

       Thus y is an element of Range(h).
    end.
  end.
  Therefore Orbit(x, f, M, G) is a finite set.
Qed.

Lemma.
Let M be a set.
Let G be a finite group.
Let f be a group action from G on M.
Let x << M.
Index(G, Stab(x, f, M, G)) = card(Orbit(x, f, M, G)).
Proof.

Take a function h such that
h is from Cosets(Stab(x, f, M, G), G) to Orbit(x, f, M, G)
and (for all elements i of El(G) h[Coset(i,Stab(x, f, M, G),G)] = f[(i, x)]).

h is surjective onto Orbit(x, f, M, G).
Proof.
  Let us show that for every element y of Orbit(x, f, M, G) there is an element z of
    Dom(f) such that f[z] = y.
    Let y be an element of Orbit(x, f, M, G).
    Take an element i of El(G) such that f[(i, x)] = y.
    Then we have h[Coset(i,Stab(x, f, M, G),G)] = y.
  end.
end.

h is injective.
Proof.
  Let us show that for all elements h1, h2 of Cosets(Stab(x, f, M, G), G) if h[h1] =
    h[h2] then h1 = h2.
    Let h1, h2 be elements of Cosets(Stab(x, f, M, G), G) such that h[h1] =h[h2].

    Take elements g1, g2 of El(G) such that h1 = Coset(g1,Stab(x, f, M, G),G) and h2
    = Coset(g2, Stab(x, f, M, G),G).

    Then f[(g1, x)] = f[(g2, x)].
    f[((Inv(g2, G) *^{G} g1), x)] = f[((Inv(g2, G)*^{G} g2), x)] = x.

    Thus Inv(g2, G) *^{G} g1 is an element of Stab(x, f, M, G).
    Therefore h1 = h2.
  end.
```

```
end.
qed.
```

3.13 Fipunkte

Wir führen den Begriff des Fixpunktes und der Menge der Fixpunkte einer Gruppenaktion ein und beweisen ein Lemma.

```
Definition.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
A fixed point of f on M on G is an element y of M such that
for every element a of El(G) f[(a, y)] = y.

Definition.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
fixedPoints(f, M, G) = {y | y is a fixed point of f on M on G}.

Lemma.
Let M be a finite set.
Let G be a group.
Let f be a group action from G on M.
fixedPoints(f, M, G) is a finite set.
Proof.
  Let us show that  every fixed point of f on M on G is an element of M.
 Let x be a fixed point of f on M on G.
 Then x is an element of M.
  end.

   fixedPoints(f, M, G) is a subset of M.

   Therefore the thesis.
Qed.

Lemma.
Let M be a set.
Let G be a finite group.
Let f be a group action from G on M.
Let x be an element of M.
x is a fixed point of f on M on G iff card(Orbit(x, f, M, G)) = 1.


Lemma OrbitsIntersect.
Let M be a set.
Let G be a group.
```

```
Let f be a group action from G on M.
Let x1, x2 be elements of M such that Orbit(x1, f, M, G) and Orbit(x2, f, M ,G) are
   not disjunct.
x1 is an element of Orbit(x2,  f, M, G).
Proof.
Take y << M such that y << Orbit(x1, f, M, G) and y << Orbit(x2, f, M, G).
Take g1 << El(G) such that f[(g1, x1)] = y.
Take g2 << El(G) such that f[(g2, x2)] = y.

x1 = f[(Inv(g1, G) *^{G} g1, x1)] = f[(Inv(g1, G), y)] = f[(Inv(g1, G) *^{G} g2, x2)
   ].

Therefore the thesis.
Qed.

Lemma.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
Let x1, x2 be elements of M such that Orbit(x1,  f, M, G) and Orbit(x2,  f, M, G)
   are not disjunct.
Orbit(x1,  f, M, G) = Orbit(x2, f, M, G).
Proof.
Let us show that every element of Orbit(x1,  f, M, G) is an element of Orbit(x2,  f,
    M, G).
  Let x3 be an element of Orbit(x1,  f, M, G).
  x1 is an element of Orbit(x2,  f, M, G) (by OrbitsIntersect).
  Thus x3 is an element of Orbit(x2,  f, M, G).
end.
Let us show that every element of Orbit(x2, f, M, G) is an element of Orbit(x1,  f,
   M, G).
  Let x3 be an element of Orbit(x2, f, M, G).
  x2 is an element of Orbit(x1,  f, M, G) (by OrbitsIntersect).
  Thus x3 is an element of Orbit(x1,  f, M, G).
end.
Qed.

Definition.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
OrbitsNotFix(f, M, G) = {Orbit(x,  f, M, G) | x is an element of M and x is not a
   fixed point of f on M on G}.

Lemma.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
```

```
OrbitsNotFix(f, M, G) is a disjunct collection.

Lemma.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
\-/ OrbitsNotFix( f, M, G) = M \\ fixedPoints(f, M, G).
Proof.
Let us show that every element of \-/ OrbitsNotFix( f, M, G) is an element of  M \\
   fixedPoints(f, M, G).
  Let x be an element of  \-/ OrbitsNotFix( f, M, G).
  Take an element y of M such that x is an element of Orbit(y,  f, M, G) and y is
   not an element of fixedPoints(f, M, G).
  x is an element of M.
  x is not a fixed point of f on M on G.
  fixedPoints(f, M , G) is a subset of M.
  Therefore  x is an element of M \\ fixedPoints(f, M, G).
end.

Let us show that every element of M \\ fixedPoints(f, M, G) is an element of  \-/
   OrbitsNotFix( f, M, G).
  Let x be an element of M \\ fixedPoints(f, M , G).
  x is an element of M.
  x is not a fixed point of f on M on G.
  Orbit(x, f, M, G) is an element of OrbitsNotFix( f, M, G).
  x is an element of Orbit(x,  f, M, G).
  Therefore  x is an element of  \-/ OrbitsNotFix( f, M, G).
end.
qed.

Lemma.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
\-/  OrbitsNotFix( f, M, G) and fixedPoints(f, M, G) are disjunct.

Lemma.
Let M be a set.
Let G be a group.
Let f be a group action from G on M.
(\-/ OrbitsNotFix( f, M, G)) \-/ fixedPoints( f, M, G) = M.

Lemma.
Let M be a finite set.
Let G be a finite group.
Let f be a group action from G on M.
OrbitsNotFix( f, M, G) is a finite set.
Proof.
```

```
Define h[x] = Orbit(x,  f, M, G) for x in M.

OrbitsNotFix( f, M, G) is a subset of Range(h).
Qed.

Lemma.
Let M be a finite set.
Let G be a finite group.
Let f be a group action from G on M.
card(M) = card(( \-/ OrbitsNotFix( f, M, G)) \-/ fixedPoints( f, M, G))
= card(fixedPoints(f, M, G)) + card(\-/ OrbitsNotFix( f, M, G)).

Signature.
Let p be a prime.
A group of order p is a finite group H such that
(there is a natural number n such that card(El(H)) = p ^ n).

Lemma.
Let M be a finite set.
Let p be a prime.
Let G be a group of order p.
Let f be a group action from G on M.
card(fixedPoints(f, M, G)) = card(M) (mod p).
Proof.

\-/ OrbitsNotFix(f, M, G) is a subset of M.

Let us show that p | card( \-/  OrbitsNotFix( f, M, G)).

  Let us show that for all elements N1 of OrbitsNotFix( f, M, G) p | card(N1).

 Let N be an element of OrbitsNotFix( f, M, G).

 Take an element x of M such that N = Orbit(x, f, M, G).

 Let us show that card(N) != 1.
   Assume the contrary.

   x is not a fixed point of f on M on G .
   x is an element of N.
   Thus x is a fixed point of f on M on G.

   Contradiction.
 end.

 We have card(N) = Index(G, Stab(x, f, M, G)).
 Hence card(El(G)) = card(Stab(x,  f, M, G))*card(N) and card(N) | card(El(G)).
 Therefore p | card(N).
```

```
  end.

  (1) OrbitsNotFix( f, M, G) is a finite set such that every element of OrbitsNotFix
   ( f, M, G) is a finite set.
  (2) OrbitsNotFix( f, M, G) is a disjunct collection and for all elements N of
   OrbitsNotFix( f, M, G) p | card(N).

  Therefore  p | card(\-/ OrbitsNotFix( f, M, G)) (by cardUnion2, 1, 2).
end.

We have card(M) = card(fixedPoints(f, M, G)) + card(\-/ OrbitsNotFix( f, M, G)).

Therefore  card(M) = card(fixedPoints(f, M, G)) (mod p).
qed.
```

3.14 Eine Gruppenaktion

Um den zweiten Satz von Sylow zu beweisen, muss eine explizite Gruppenaktion definiert werden. Im Gegensatz zur abstrakten Formulierung des Gruppen- und Zahlbegriffes in dieser Formalisierung lässt sich aufgrund der Formulierung der Definition von Gruppenaktionen nun in Naproche zeigen, dass es sich bei dieser Funktion um eine Gruppenaktion handelt.

```
Lemma.
Let P be a subgroup of G.
Let U be a subgroup of G.
Let u be an element of U.
Let x, y be elements of El(G) such that Coset(x, P, G) = Coset(y, P, G).
Every element of Coset(u *^{G} x, P, G) is an element of  Coset(u *^{G} y, P, G).
Proof.
    Let i be an element of Coset(u *^{G} x, P, G).
    Take an element p of P such that i =  (u *^{G} x) *^{G} p.
    Then we have Inv(u, G) *^{G} i = Inv(u, G) *^{G} ((u *^{G} x) *^{G} p)
     =((Inv(u, G) *^{G} u ) *^{G} x) *^{G} p
    = x *^{G} p.
    Inv(u, G) *^{G} i is an element of Coset(x, P, G).
    Therefore Inv(u, G) *^{G} i is an element of Coset(y, P, G)
    and i is an element of Coset(u *^{G} y, P, G).
qed.

Lemma.
Let P be a subgroup of G.
Let U be a subgroup of G.
Let u be an element of U.
Let x, y be elements of El(G) such that Coset(x, P, G) = Coset(y, P, G).
Coset(u *^{G} x, P, G) = Coset(u *^{G} y, P, G).
Proof.
  Every element of Coset(u *^{G} x, P, G) is an element of  Coset(u *^{G} y, P, G).
  Every element of Coset(u *^{G} y, P, G) is an element of  Coset(u *^{G} x, P, G).
```

```
  Therefore the thesis.
Qed.
```

Aufgrund des vorangegangen Lemmas führt die folgende Definition nicht zu Problemen hinsichtlich der Konsistenz der Formalisierung (Siehe 3.12).

```
Definition.
Let P be a subgroup of G.
Let U be a subgroup of G.
Op(U, P, G) is a function f
such that f is from Prod(El(Gr(U, G)), Cosets(P, G)) to Cosets(P, G) and
for all elements u of U for all elements x of El(G)
f[(u, Coset(x, P, G))] = Coset(u *^{G}  x,P, G).

Lemma.
Let P be a subgroup of G.
Let U be a subgroup of G.
Op(U, P, G) is a group action from Gr(U, G) on Cosets(P, G).
Proof.
Take a function f such that f = Op(U, P, G).
Take a group H such that  H = Gr(U, G).
Take a set M such that M = Cosets(P, G).

For every element x of M we have f[(One(H), x)] = x.

Let us show that for every element x of M for all elements a, b of El(H)
 f[(a *^{H} b, x)] =  f[(a,  f[(b, x)])].
  Let x be an element of M.
  Let a, b be elements of El(H).

  Take an element g of El(G) such that x = Coset(g, P, G).

  We have f[(b, x)] = Coset(b *^{G} g,P, G).

  f[(a, f[(b, x)])] = Coset(a *^{G} (b *^{G} g),P, G).

  f[(a *^{H}  b, x)] = Coset((a *^{G} b)*^{G} g,P, G).

  Thus f[(a, f[(b, x)])] = f[(a *^{H} b, x)].
end.

Therefore the thesis.
Qed.
```

3.15 Vorbereitung für den zweiten Satz von Sylow

Wir führen die Begriffe des zweiten Satzes von Sylow ein und zeigen, dass zwei Sylowgruppen die gleiche Kardinalität haben.

```
Signature.
Let p be a prime.
A subgroup of G of order p is a subgroup U of G such that
Gr(U, G) is a group of order p.

Definition.
Let p be a prime.
Let G be a finite group.
Syl(p, G) = {P | P is a subgroup of G of order p and  not (p | Index(G, P))}.

Lemma SylSize.
Let p be a prime.
Let G be a finite group.
Let P, U be elements of Syl(p, G).
card(U) = card(P).
Proof.
Take natural numbers n, m such that p^n = card(U) and p^m = card(P).

(1) card(El(G)) = (p^n)*Index(G, U)
  and card(El(G)) = (p^m)*Index(G, P)
  and p does not divide Index(G, U)
  and p does not divide Index(G, P).

Thus we have n = m (by DLogN, 1).

Therefore the thesis.
Qed.
```

3.16 Der zweite Satz von Sylow

Schlussendlich können wir den Beweis des zweiten Satzes von Sylow in Naproche formulieren.

```
Theorem Sylow2a.
Let p be a prime.
Let G be a finite group.
Let P be an element of Syl(p, G).
Let U be a subgroup of G of order p.
There is an element g of El(G) such that Conjugate(g, U, G) is a subset of P.
Proof.
  Take a group action f from Gr(U, G) on Cosets(P, G) such that f = Op(U, P, G).
```

```
  Let us show that card(fixedPoints(f, Cosets(P, G), Gr(U, G))) !=  0.
  We have card(fixedPoints(f, Cosets(P, G), Gr(U, G))) = Index(G, P) (mod p).
    p does not divide Index(G, P).
  Therefore Index(G, P) != 0 (mod p).
  end.

  We can take an element x of fixedPoints(f, Cosets(P, G), Gr(U, G))
  and an element g of El(G) such that x = Coset(g, P, G).

  Let us show that every element of Conjugate(Inv(g, G), U, G) is an element of P.
 Let h  be an element of Conjugate(Inv(g, G), U, G).

 Take an element u of U such that h = Inv(g, G) *^{G} (u *^{G}  g).

 We have Coset(g, P, G) = f[(u,x)] = Coset((u *^{G} g) ,P, G).

 Therefore Inv(g, G) *^{G} (u *^{G}  g) is an element of P.

 Thus h is an element of P.
  end.
Qed.

Theorem Sylow2b.
Let p be a prime.
Let G be a finite group.
Let P, U be elements of Syl(p, G).
P and U are conjugates in G.
Proof.
  Take an element g of El(G) such that Conjugate(g, U, G) is a subset of P.

  card(Conjugate(g, U, G)) = card(U) = card(P).

  Hence Conjugate(g, U, G) = P.
qed.
```

4 Vergleich

4.1 Direkte Gegenüberstellung

4.1.1 Herkömmliche Mathematik

Theorem 4.1. *Sei p eine Primzahl und G eine endliche Gruppe mit $|G| = p^r * m$, sodass $p \nmid m$. Sei $U \leq G$ eine p-Untergruppe, und sei $P \leq G$ eine p-Sylowgruppe. Dann gilt:*

(i) Es gibt ein $g \in G$ mit

$$gUg^{-1} \subseteq P.$$

(ii) Je zwei p-Sylowgruppen sind konjugiert.

Beweis. (i) Setze $X := {}^{G}/_{P}$ und betrachte die Gruppenaktion

$$\phi : U \times X \to X$$

$$(u, gP) \mapsto ugP.$$

Nach Lemma 1.5 gilt

$$|X| = [G : P] = m \equiv |X^G| \mod p.$$

Da $p \nmid m$, gilt $X^G \neq \varnothing$. Es existiert also ein $g \in G$, sodass $ugP = gP$ für alle $u \in U$. Folglich ist $g^{-1}Ug \subseteq P$ und damit U konjugiert zu P bzgl. g^{-1}.

(ii) Dies gilt insbesondere im Falle $U \in \mathrm{Syl}_p(G)$. □

4.1.2 Lean

```
lemma sylow_2 [fintype G] {p : ℕ} (hp : nat.prime p)
(H K : set G) [is_sylow H hp] [is_sylow K hp] :
∃ g : G, H = conjugate_set g K :=

have hs : card (left_cosets K) = card G / (p ^ dlogn p (card G)) :=
(nat.mul_right_inj (pos_pow_of_pos (dlogn p (card G)) hp.pos)).1
$ by rw [ ← card_sylow K hp, ← card_eq_card_cosets_mul_card_subgroup,
  card_sylow K hp,
nat.div_mul_cancel (dlogn_dvd _ hp.1)],

have hmodeq : card G / (p ^ dlogn p (card G)) ≡ card (fixed_points H (
  left_cosets K)) [MOD p] :=
eq.subst hs (mul_action.card_modeq_card_fixed_points hp (card_sylow H
  hp)),

have hfixed : 0 < card (fixed_points H (left_cosets K)) :=
```

```
 nat.pos_of_ne_zero
 (λ h, (not_dvd_div_dlogn (fintype.card_pos_iff.2 ⟨(1 : G)⟩) hp.1)
 (by rwa [h, nat.modeq.modeq_zero_iff] at hmodeq)),
```

4.1.3 Naproche

```
Theorem Sylow2a.
 Let p be a prime.
 Let G be a finite group.
 Let P be an element of Syl(p, G).
 Let U be a subgroup of G of order p.
 There is an element g of El(G) such that Conjugate(g, U, G) is a subset of P.
   Proof.
   Take a group action f from Gr(U, G) on Cosets(P, G) such that f = Op(U, P, G).

   Let us show that card(fixedPoints(f, Cosets(P, G), Gr(U, G))) !=  0.
   We have card(fixedPoints(f, Cosets(P, G), Gr(U, G))) = Index(G, P) (mod p).
     p does not divide Index(G, P).
   Therefore Index(G, P) != 0 (mod p).
   end.

   We can take an element x of fixedPoints(f, Cosets(P, G), Gr(U, G))
   and an element g of El(G) such that x = Coset(g, P, G).

   Let us show that every element of Conjugate(Inv(g, G), U, G) is an element of P.
  Let h  be an element of Conjugate(Inv(g, G), U, G).

  Take an element u of U such that h = Inv(g, G) *^{G} (u *^{G}  g).

  We have Coset(g, P, G) = f[(u,x)] = Coset((u *^{G} g) ,P, G).

  Therefore Inv(g, G) *^{G} (u *^{G}  g) is an element of P.

  Thus h is an element of P.
   end.
Qed.

Theorem Sylow2b.
Let p be a prime.
Let G be a finite group.
Let P, U be elements of Syl(p, G).
P and U are conjugates in G.
Proof.
   Take an element g of El(G) such that Conjugate(g, U, G) is a subset of P.

   card(Conjugate(g, U, G)) = card(U) = card(P).

   Hence Conjugate(g, U, G) = P.
qed.
```

4.2 Beweisführung

Der zentrale Unterschied zwischen den beiden Formalisierungen ist, dass Lean eine funktionale Programmiersprache ist und auf der Typentheorie basiert. Dies führt zu einer sehr technischen Aufbau des Lean-Beweises durch die Komposition von Lambda-Termen. Als Beispiel hierfür

```
one_smul := λ a, quotient.induction_on' a (λ a, quotient_group.eq.2
    (by simp [is_submonoid.one_mem]))
```

Im Gegensatz dazu arbeitet das Naprochesystem in einem deklarativen Stil, das heißt mit aneinandergereihten Aussagen, die durch den internen `Reasoner` sowie den externen `ATP` schrittweise verifiziert werden. Dies ermöglicht es dem Naproche-Text auf der einen Seite, auf sprachlicher Ebene recht nah am Stil der herkömmlichen Mathematik zu bleiben. Auf der anderen Seite ermöglicht die Beschaffenheit von Lean als Programmiersprache hier jedoch gewisse Methoden wie die mehrfache Wiederholung von Argumenten durch sogenannte "Taktiken" umzusetzen, was sich im deklarativen Stil von Naproche momentan nicht abbilden lässt (Betrachte den Beweis in Abschnitt 3.8).

```
example (p q : Prop) (hp : p) (hq : q) : p ∧ q :=
by split; assumption
```

(Beispiel eines Taktik-Beweises in Lean unter Verwendung der Taktikverknüpfung ";", die dazu führt, dass die Taktik `assumption` auf alle verbleibenden `subgoals` angewandt wird [7].)

4.3 Algebraische Strukturen

Ein weiterer Unterschied ist die Behandlung von algebraischen Strukturen hinsichtlich der Begriffsbildung.

So existiert in Lean das System der Typenklassen, während in Naproche keine bis auf einen naiven Mengen und Funktionsbegriff keine Strukturelemente bereitgestellt werden.

Während in der Formalisierung in Lean beispielsweise die Gruppen strukturell als Erweiterung des Typenklasse Monoid entstehen, werden diese in der Naproche-Formalisierung als abstrakte Notion eingeführt und so über allgemeine Gruppen argumentiert. Exemplarisch stellt sich hier die Behandlung der Untergruppen heraus: `instance subtype.group s : set α [is_subgroup] s : group s := ...` Hier wird in Lean gezeigt, dass Untergruppen eine `instance` der Typenklasse `group` ist. Dies sorgt dafür, dass sich Untergruppen in Lean nun ohne weitere Notation auch als Gruppen auffasen lassen. Da Naproche keine solche Strukturen bereitstellt, benötigten wir hierfür die axiomatische geforderte Gruppe `Gr(U, G)` (siehe Abschnitt 3.2).

4.4 Nebenklassen

Der Begriff der Nebenklassen ist in den vorliegenden Formalisierungen insofern interessant, als dass er in Lean und Naproche auf zwei verschiedene Arbeiten umgesetzt wird. Während

diese in Lean als Qoutienten eigenführt werden

```
def left_rel [group α] (s : set α) [is_subgroup s] : setoid α :=
⟨λ x y, x⁻¹ * y ∈ s,
assume x, by simp [is_submonoid.one_mem],
assume x y hxy,
have (x⁻¹ * y)⁻¹ ∈ s, from is_subgroup.inv_mem hxy,
by simpa using this,
assume x y z hxy hyz,
have x⁻¹ * y * (y⁻¹ * z) ∈ s, from is_submonoid.mul_mem hxy hyz,
by simpa [mul_assoc] using this⟩

def left_cosets [group α] (s : set α) [is_subgroup s] : Type* :=
  quotient (left_rel s)
```

haben wir uns in Naproche aufgrund des dort fehlenden Qoutientenbegriffes für eine mengenbasierte Definition entschieden:

```
Definition.
Let g be an element of El(G).
Let H be a subgroup of G.
Coset(g, H, G) = {g *^{G} h | h << H}.
```

Das Lemma 1.1 beweist, dass beide Ansätze das gleiche metatheoretische Objekt beschreiben.

4.5 Funktionsdefinitionen

Während Lean als funktionale Programmiersprache verschieden Möglichkeiten zur Definition von Funktionen und Strukturen bereitstellt, stellt Naproche nur einen eigenschräkten naiven Funktions- und Begriff bereit. Während innerhalb von Beweisen durch Ausdrücke der Form `Let f[x] = y for x in M` Funktionen in Naproche definiert werden können, gibt es auf der globalen Ebene nur die Möglichkeit, Funktionen durch Axiome beziehungsweise 'axiomatische' Definitionen zu fordern. Dies führt zu den in Abschnitt 3.12 angesprochenen Problemen hinsichtlich der "Wohldefiniertheit".

4.6 Struktur

Im herkömmlicheen Beweis [1] werden die grundlegende Eigenschaften von endlichen Mengen und den ganzen Zahlen als bekannt vorrausgesetzt. Dies spiegelt sich in der Formalisierung in Naproche insofern wieder, als diese Eigenschaften axiomatisch gefordert und nicht explizit bewiesen werden. In der Lean-Formalisierung hingegen wird mithilfe der umfassenden mathlib ausgehend von grundlegenden typentheoretischen Definitionen eine abgeschlossene Theorie aufgebaut und verifiziert.

Es wäre wünschenswert die Naproche-Formalisierung als Ganzes verifizieren zu lassen. Hier scheitert jedoch das System aus Komplexitätsgründen, weswegen wir in jeder Datei jeweils die Aussagen und Definitionen der vorangegangen Dateien als Axiome ohne Beweise einbinden. Dieser Ansatz findet sich insofern in der herkömmlichen Mathematik, als dort auch nur die Aussagen eines bereits bewiesenen Theorems zititert werden, wohingegen die exakte Ausgestaltung des Beweises meist eine untergeordnete Rolle spielt: "Nach Lemma 1.5 gilt [...]"[1].

Im Gegensatz hierzu profitiert Lean als Programmiersprache von der Möglichkeit Bibliotheken einzubinden. Es ist inbesondere möglich, Dateien oder ganze Bibliotheken durch den Kommandozeilenbefehl `leanpkg build` in `.olean` Binärdateien zu übersetzen, die von Lean dann weitaus schneller verifiziert werden können. Dies ermöglicht es, mit umfangreichen Bibliotheken wie der mathlib [3] interaktiv zu arbeiten.

4.7 Lesbarkeit

In der vorangegangen Gegenüberstellung ist erkennbar, dass der deklarative Stil von Naproche / ForTheL sowie der Verwendung von kontrollierter natürlicher Sprache dazu führt, dass der Naproche-Text für herkömmliche Mathematiker lesbar ist. So ist bis auf die Layout- und Gestaltungsebene eine große oberflächliche Ähnlichkeit zwischen den Beweisen zu erkennen. Betrachte:

Nach Lemma 1.5 gilt

$$|X| = [G : P] \equiv |X^G| \mod p.$$

```
We have card(fixedPoints(f, Cosets(P, G), Gr(U, G))) = Index(G, P) (mod p).
```

Im Gegensatz hierzu lässt sich die Formalisierung in Lean nur mit einer gewissen Vorkenntnis in funktionaler Programmierung und dem Beweisen mithilfe von Taktiken verstehen. Beispielweise:

```
(λ h, (not_dvd_div_dlogn (fintype.card_pos_iff.2 ⟨(1 : G)⟩) hp.1)
```

4.8 Elaborating

Ein Vorteil von Lean gegenüber Naproche / ForTheL hinsichtlich der Lesbarkeit ist das sogennante ”Elaborating”. Aufgrund der strikten Typisierung ist Lean in der Lage, beispielsweise bei Ausdrücken der Form $a * b$ mit $a, b : G$ selbstständig aus dem Kontext zu ermitteln, dass es sich bei $*$ um die Verknüpfung von G handelt. Da dies in Naproche nicht möglich ist, entstehen sperrige Ausdrücke wie beispielsweise `fixedPoints(Op(U, P, G), Cosets(P, G), Gr(U, G))`, die die Lesbarkeit des Naproche-Textes vermindern. In der herkömmlichen Mathematik findet sich hier oft ein Kompromiss. So werden an einigen Stellen die Abhängigkeiten gekennzeichnet, wie zum Beispiel im Ausdruck $\mathrm{Syl}_p(G)$, während an anderen Stellen, wie in *Je zwei p-Sylowgruppen sind konjugiert*, aus dem Kontext ersichtlich wird, welche Gruppe gemeint ist.

4.9 User Experience

Wir wollen nun einige Aspekte des Arbeitens mit den beiden Systemen beschreiben.

Ein Gesichtspunkt ist hier die Stabilität der Systeme. Solange man die gleiche Lean Version verwendet, kann man einen einmal geschrieben Lean-Text in verschiedenen Umgebungen verifizieren. Für verschiedene Versionen von Lean gilt dies jedoch nicht. So ist zum Beispiel die ursprüngliche Lean Datei [2] mit der Lean Version 3.4.2 nicht mehr kompatibel und musste deswegen angepasst werden. Im Gegensatz hierzu ist das Verifizieren in Naproche sehr instabil. So führt das Verändern weniger Zeilen im Quelltext häufig dazu, dass Aussagen an anderen Stellen, die vorher verifiziert wurden, nicht mehr verifiziert werden. Insbesondere ist hier das Zusammenarbeiten mehrerer Personen mit verschiedenen Betriebssystemen kritisch, da nach unserer Erfahrung unter Linux/Mac verifizierte Dateien unter Windows oftmals nicht ohne weitere Veränderungen verifiziert werden konnten.

Hier scheint insbesondere der `Reasoner` in Naproche eine Rolle zu spielen, der in unserer Arbeit große Schwierigkeiten hatte, Funktionsterme auf ontologische Korrektheit zu überprüfen, weswegen wir deren Einsatz auf ein Minimum reduzieren mussten.

Schlussendlich würden beide Systeme von einer ausgeprägteren Dokumentation profitieren, da momentan viel Zeit darauf verloren geht, sich über einen Trial-and-Error-Prozess mit den Systemen vertraut zu machen. Lean profititiert hier von den ausgeprägten Fehlermeldungen, die oftmals klar erkennbar machen, welche Zeilen in welcher Form geändert werden müssen, um eine valide Datei zu erhalten. Schreibt man beispielsweise anstatt `contradiction` fälschlicherweise `contradictio`, so meldet Lean `unknown identifier 'contradictio'` während Isabelle-Naproche `unexpected .` erst am Ende der Zeile reklamiert. Schlägt die Verifizierung fehl, so gibt Lean beispielsweise folgende Fehlermeldung aus:

```
contradiction tactic failed
state:
x y : ℕ,
h : x = y
⊢ y = x
```

während Isabelle-Naproche nur `[Reasoner] goal failed` zurückgibt.

5 Diskussion

5.1 Begriffsbildung

Um mathematische Beweise zu formalisieren, ist es grundlegend, die im Beweis vorkommenden Begriffe im jeweiligen System zu implementieren. Im vorliegenden Projekt ist insbesondere die Implementierung der Nebenklassen (Siehe Abschnitt 4.4) zu betrachten. Hier werden zwei verschiedene Ansätze genutzt, die auf der Metaebene äquivalent sind.

Interessant wäre es, Möglichkeiten zu schaffen, Formalisierungen in evtl. verschiedener Systeme auf einer Metaebene miteinander zu kombinieren, um die jeweiligen Vorteile auszunutzen. Hätte man diesen Schritt vollzogen, so könnte man die vorliegende Formalisierung des zweiten Satzes von Sylow in Naproche durch die Lean-Formalisierung der ganzen Zahlen und Kardinalitäten von endlichen Mengen vervollständigen, um eine in sich geschlossenen Formalisierung zu erhalten, deren Hauptresultat in der für Mathematiker leichter lesbaren kontrollierten natürlichen Sprache ForTheL vorliegt, während elementare technische Aussagen durch das hier momentan effizientere System Lean mithilfe der mathlib verifiziert werden würden.

Hervorzuheben ist das Projekt "Formal Abstracts" [8], das zentrale Sätze der Mathematik in Lean formuliert, um eine einheitliche Grundlage für weitere Formalisierungen zu schaffen.

5.2 Vorschläge zur Weiterentwicklung an Naproche

Als Resultat des Formalisierungsprozesses sind einige Ideen für die Weiterentwicklung des Naprochesystems entstanden.

- So wäre es wünschenswert eine native Option zu haben, mit algebraischen Strukturen umzugehen, da diese Ausgangspunkt vieler mathematischer Theorien sind und momentan von Naproche nicht unterstützt werden. (Siehe Abschnitt 3.2 bezüglich der Formalisierung des Gruppenbegriffes).

- Des Weiteren ist der aktuell in Naproche implementierte Funktionsbegriff nicht ausreichend, um Konstrukte wie in Abschnitt 3.12 abzudecken. In Lean werden solche Konstruktionen durch den Qoutientbegriff `qout` abgedeckt (Siehe Abschnitt 2.1). Insofern wäre es sinnvoll, einen solchen auch in Naproche zu implementieren.

- Ein großer Teil des Formalisierens in Naproche bechränkte sich darauf, die Ausdrücke so zu modifizieren, dass sie vom `Reasoner` auf ontologische Korrektheit überprüft werden konnten. In Anbetracht der vorangehenden Punkte, wäre es interessant, den Reasoner hinsichtlich evtl. nativen Konstrukte zu optimieren. Momentan scheint dieser gerade bei Funktionstermen große Probleme zu haben, sodass an vielen Stellen weitere Argumente hinzugefügt werden mussten.

- Eine weitere Idee, die während unseres Projektes entstanden ist, ist das Einführen eines weiteren Axiom-Begriffes. So finden sich in unser Formalisierung zum einen "grundlegende" Axiome wie die Forderung des Existenz von Inversen in einer Gruppe (Siehe Abschnitt 3.2) und "vorraussetzenden" Axiome wie `cardUnion1` (Siehe Abschnitt

3.9.), welche im Gegensatz zu den ”grundlegenden” Axiomen sozusagen die mathematische Vorkenntnisse beschreiben, die von einem Leser eines herkömmlichen Textes über die Sylow-Sätze gefordert werden. Hier wäre es eventuell sinnvoll, eine Möglichkeit der expliziten Unterscheidung dieser Axiome vorzunehmen. (Zum Beispiel durch das Einführen des Bezeichners `Presupposition` zusätzlich zu `Axiom`.)

- Für das Einbinden von anderen Dateien existiert in Isabelle-Naproche bereits der `read` Befehl. Dieser kann allerdings in der momentanen Ausführung nur auf Dateien im `examples` Ordner zugreifen. Hier wäre es wünschenswert, auch auf andere Verzeichnisse zugreifen zu können. Im Sinne der `.olean` Binärdatein in Lean wäre es zudem sinnvoll, einen ähnlichen Übersetzungsprozess in Naproche zu implementieren, um bereits verifzierte Texte effizienter in Texte einzubinden und so den von uns gewählten Ansatz des ”Exportierens” von Theoremen als Axiome zu internaliseren.

- Ausgeprägtere Fehlermeldungen in Isabelle-Naproche würden den Prozess des Formalisierens vermutlich vereinfachen, da gerade die Arbeit des `Reasoners` in Kombination mit dem externen `ATP` momentan eine Blackbox ist, was dazu führt, dass das Beheben Fehlern meist nur durch langwieriges Ausprobieren realisierbar ist.

- Es wäre des Weiteren sinnvoll, Isabelle-Naproche dahingehend anzupassen, dass es sich unter den verschiedenen Betriebsystemen identisch verhält. So brechen unter Mac Os X die Beweise momentan nach einer bestimmten Zeit ab, während dies unter Windows nicht der Fall ist. Dies hindert die gemeinsame Entwicklung von Naproche-Texten.

- Schlussendlich würde eine ausgeprägtere Dokumentation des im Sinne des Einführungstextes ”Theorem Proving in Lean” [7], insbesondere den Einstieg in das Arbeiten mit Naproche steigern. So ist momentan beispielsweise nicht klar, welche sprachlichen Konstrukte überhaupt unterstützt werden.

5.3 Fazit

Zusammenfassend ist erkennbar geworden, dass die betrachteten Formalisierungssysteme recht unterschiedliche Ansätze verfolgen die verschiede Vor- und Nachteile haben. Während Lean als typentheoretische funktionale Programmiersprache von der Möglichkeit des effizienten Einbindens von umfassenden Bibliotheken wie der mathlib sowie der interaktiven Unterstützung durch präzise Fehlermeldungen profitiert, liegt der Vorteil von Naproche im deklarativen Stil und der natürlichen Sprache, die es ermöglicht, Formalisierungen zu erstellen, die Texten der herkömmlichen Mathematik ähneln. Erstrebenswert wäre es, Möglichkeiten der Kombination der beiden Systeme zu schaffen, um auch größere mathematische Theorien in einem für herkömmliche Mathematiker verständlichen Stil zu formalisieren.

Literatur

[1] J. Schröer. Eine Einführung in die Algebra (Skript, WS 19/20, Bonn). 2020.

[2] C. Hughes. Sylow. https://github.com/ChrisHughes24/Sylow.

[3] mathlib. https://github.com/leanprover-community/mathlib, .

[4] 2. Satz von Sylow in Lean. https://github.com/moritz-hl/sylow2, .

[5] M. Wenzel. ISABELLE/NAPROCHE FOR AUTOMATIC PROOF-CHECKING OF ORDINARY MATHEMATICAL TEXTS. https://sketis.net/2019/isabelle-naproche-for-automatic-proof-checking-of-ordinary-mathematical-texts.

[6] 2. Satz von Sylow in Naproche. https://github.com/moritz-hl/sylowftl, .

[7] J. Avigad. Theorem Proving in Lean. https://leanprover.github.io/theorem_proving_in_lean/tactics.html.

[8] T. Hales. Formal Abstracts. https://formalabstracts.github.io, 2017.