A Formal Proof of Sylow's Theorem

An Experiment in Abstract Algebra with Isabelle HOL

Florian Kammüller and Lawrence C. Paulson
Computer Laboratory, University of Cambridge, UK

Abstract. The theorem of Sylow is proved in Isabelle HOL. We follow the proof by Wielandt that is more general than the original and uses a non-trivial combinatorial identity. The mathematical proof is explained in some detail leading on to the mechanization of group theory and the necessary combinatorics in Isabelle. We present the mechanization of the proof in detail giving reference to theorems contained in an appendix. Some weak points of the experiment with respect to a natural treatment of abstract algebraic reasoning give rise to a discussion of the use of module systems to represent abstract algebra in theorem provers. Drawing from that, we present tentative ideas for further research into a section concept for Isabelle.

Table of Contents

1. Proofs of Abstract Algebra with Theorem Provers

The first theorem of Sylow is most easily described as the backwards direction of Lagrange's theorem. Lagrange says that the order of a subgroup divides the group's order. Unfortunately, the direct backward direction:

> *if a number divides the group's order then there is a subgroup with corresponding order*

is not generally true. But the theorem of Sylow gives us at least:

> *if p is a prime and p^α divides the order of the group then there is a subgroup of order p^α.*

The proof of the theorem of Lagrange has been performed with the Boyer Moore Prover [Yu90]. E. Gunter formalized group theory in HOL [Gun89]. In the higher order logic theorem prover IMPS [FGT93] some portion of abstract algebra including Lagrange is proved. Mizar's library of formalized mathematics contains probably more abstract algebra theorems than any other system. But to our knowledge none of the known systems has proved Sylow's theorem. We always considered it as a theorem which is hard to prove already in theory and this is definitely a fact which makes it an interesting challenge for theorem provers.

Our personal motivation to prove this theorem is to explore applications of different knowledge domains in abstract theorems. We use the example of abstract algebra to find out about reasoning mechanisms in algebra or in general in abstract reasoning.

The paper first gives a formulation of Sylow's theorem together with its proof and then explains the formalization in Isabelle HOL. Here, some special properties of the representation are highlighted. Section 4 describes the formal proof in its major steps. Basic theorems from set or number theory used in the proof are enumerated; they are not described in detail though their proofs are also part of our work.

Finally, we describe in more detail where we think that some more support of abstract reasoning could enhance treatment of abstract algebra and compare these ideas to existing approaches. The appendix contains the theory and proof files.

2. The First Sylow Theorem

Sylow's theorem gives a criteria for the existence of subgroups of prime power order in finite groups.

THEOREM 1. *If G is a group, p a prime and p^α divides the order of G then G contains a subgroup of order p^α.*

In the following we write $a \,|\, b$ for *a divides b* and $o(G)$ for the order of G.

2.1. Proof of Sylow's Theorem

The proof displayed here and used as the basis for the formal proofs is due to Wielandt [Wie59]. It generalizes the original form found by the Norwegian Mathematician Sylow in 1872. We give the proof following [Her64] but go much more into detail to prepare the description of the formalization.

Proof.

The proof is presented in three major parts. In the second part the existence of a subgroup of G having p^α elements is shown by constructing some subgroup H and the final part proves by combinatorial arguments that H actually has p^α elements. In the construction of the subgroup H we define the set $\mathcal{M}$ of all subsets of G having p^α elements. We have to consider first a combinatorial argument about $\mathcal{M}$ which is used in the final part of the proof.

2.1.1. *Combinatorial Argument*

If $p^\alpha \,|\, o(G)$ we can assume an $m > 0$ such that $o(G) = p^\alpha m$ because G is a group and hence nonempty. The cardinality of the set $\mathcal{M}$ is then $\left(\begin{array}{c} p^\alpha m \\ p^\alpha \end{array}\right)$ because this is the number of ways one can pick a set of p^α elements out of G. We define the number r as the *maximum* natural number such that $p^r \,|\, m$, that is $p^{r+1} \not| m$.

In the following we show that $p^r \,|\, m$ *iff* $p^r \,|\, \left(\begin{array}{c} p^\alpha m \\ p^\alpha \end{array}\right)$. The following argument yields the former equivalence for an arbitrary natural number which can then be particularized to r:

$$\begin{aligned}\left(\begin{array}{c} p^\alpha m \\ p^\alpha \end{array}\right) &= \frac{p^\alpha m(p^\alpha m - 1)\ldots(p^\alpha m - (p^\alpha - 1))}{p^\alpha(p^\alpha - 1)\ldots 1} \\ &= m\frac{(p^\alpha m - 1)\ldots(p^\alpha m - (p^\alpha - 1))}{(p^\alpha - 1)\ldots 1}\end{aligned}$$

The power of p dividing $p^\alpha m - k$ in the numerator is the same as the power of p dividing $p^\alpha - k$ in the denominator for all $k = 1, \ldots, p^\alpha - 1$. This observation holds in the one direction because if $p^s \,|\, p^\alpha - k$ and $k < p^\alpha$ then $s \leq \alpha$ and thus for the quotient x, i.e. $p^s x = p^\alpha - k$, we can construct $p^\alpha m - k$ as $p^s(x + (p^{\alpha - s}(m-1)))$ whereby $p^s \,|\, p^\alpha m - k$.

Conversely, a slightly more difficult argument yields that $s \leq \alpha$: if similarly $k < p^\alpha$ and $p^s \,|\, p^\alpha m - k$ assuming for contradiction that

$p^s > p^\alpha$ then $p^\alpha \,|\, p^s$ and by transitivity of divisibility $p^\alpha \,|\, p^\alpha m - k$. Since $p^\alpha \,|\, p^\alpha m$ it must follow that $p^\alpha \,|\, k$ in contradiction to $k < p^\alpha$ (and $0 < k$). Thus, we can construct $p^\alpha - k$ from $p^s x = p^\alpha m - k$ (where x is again the quotient) as $p^s(x - (p^{\alpha - s}(m-1)))$. The property $s \leq \alpha$ is necessary because we are in $\mathbb{N}$ and thus $s + (\alpha - s) = \alpha$ *only if* $s \leq \alpha$.

Thereby, the powers of p in denominator and numerator all cancel out. Hence

$$p \not| \frac{(p^\alpha m - 1)\dots(p^\alpha m - (p^\alpha - 1))}{(p^\alpha - 1)\dots 1}$$

and thus, the power of p dividing $\binom{p^\alpha m}{p^\alpha}$ is the same as the power of p dividing m. The right hand side of the above formula is an integer because it equals $\binom{p^\alpha m - 1}{p^\alpha - 1}$.

2.1.2. *Construction of the subgroup H*

Consider the set $\mathcal{M}$ of all subsets with p^α elements in G. On this set, define a relation $\sim$ as $M_1 \sim M_2$ if there exists a $g \in G$ such that $M_1 = M_2 g$. It is straightforward to prove that this relation is an equivalence relation on $\mathcal{M}$. Now, for the maximum number r such that $p^r \,|\, m$ we claim that there is an equivalence class M in $\mathcal{M}/\!\sim$ such that $p^{r+1} \not| card(M)$. If not, then p^{r+1} would divide the cardinality of *all* classes in $\mathcal{M}/\!\sim$ and thus $p^{r+1} \,|\, card(\mathcal{M}) = \binom{p^\alpha m}{p^\alpha}$ because equivalence classes partition $\mathcal{M}$. But, this would yield by the combinatorial argument of Section 2.1.1 that $p^{r+1} \,|\, m$ in contradiction to the assumption that r is maximal.

Now, let n be the cardinality of this class M. Since M has n elements, let us name them $M_1, \dots, M_n$. We pick M_1 out of the equivalence class M — which is possible since $n \neq 0$ — and construct the subgroup H from this p^α-set M_1 as

$$H \equiv \{g \in G \mid M_1 g = M_1\}$$

This set H is a subgroup:

- $e \in H$ because $M_1 e = M_1$ for all subsets of G and thus also for M_1.
- for $a, b \in H$ is $M_1 a = M_1$ and $M_1 b = M_1$ by the definition of H. Thereby, $M_1(ab) = (M_1 a)b = M_1 b = M_1$ yields $ab \in H$.

These two criterias are sufficient to show that H is a subgroup.

2.1.3. *Cardinality of H is p^α*

First we establish that $n \cdot o(H) = o(G)$, in other words $card(M) \cdot o(H) = p^\alpha m$. To this end we construct a bijection between M and the set of right cosets G/H of H. By construction of H we get the equivalence:

$$(Ha = Hb) \equiv (ab^{-1} \in H) \equiv (M_1ab^{-1} = M_1) \equiv (M_1a = M_1b)$$

for all $a, b \in G$. That is, whenever a and b are in the same right coset of H (or their cosets are equal, respectively) they form the same $M_1a = M_1b$, name it N; and $N \in M$ because $Nb^{-1} = M_1$, hence $N \sim M_1$. So $Ha \mapsto M_1a$, for all $a \in G$, defines a mapping from G/H to M. Since $N \in M$, N is some M_j, $j \in \{1, \ldots, n\}$, and conversely, each M_j is of the form M_1a for some $a \in G$ by definition. So the mapping $Ha \mapsto M_1a$ for all $a \in G$ is in fact a bijection.

By this bijection we know that $card(M) \cdot card(H) = card(G/H) \cdot card(H)$ which equals $o(G)$ according to Lagrange's theorem (cf. Theorem 2).

Now we prove the two directions separately:

1. $p^\alpha \leq o(H)$:

   We constructed M such that $p^{r+1} \not| n = card(M)$. Hence, for the maximum k such that $p^k \,|\, n$ must hold $k \leq r$, i.e. $p^k \,|\, p^r$. By construction of r holds $p^{\alpha+r} \,|\, p^\alpha m = n \cdot o(H)$ and consequently $p^{\alpha+k} \,|\, n \cdot o(H)$. But since k was already the maximum power of p dividing n we get from this $p^\alpha \,|\, o(H)$ whereby $p^\alpha \leq o(H)$.

2. $o(H) \leq p^\alpha$:

   For some arbitrary $m_1 \in M_1$, we have $m_1h \in M_1$ for all $h \in H$ because of the definition of H as $\{g \in G \mid M_1g = M_1\}$. Since this group operation is an injection, i.e. $h_1 \neq h_2 \Rightarrow m_1h_1 \neq m_1h_2$ (cancellation law for the binary operation), it follows that M_1 must have at least $o(H)$ different elements whereby $o(H) \leq card(M_1)$. Since the set M_1 is in $\mathcal{M}$ it has p^α elements and thereby finally $o(H) \leq p^\alpha$.

Summarizing, the constructed subgroup H has exactly p^α elements. And the proof is finished.

The original form of the theorem of Sylow is a special case of the previous one:

COROLLARY 1. *If G is a group, p a prime, $p^m|o(G)$ and $p^{m+1} \not| o(G)$ then G contains a subgroup of order p^m.*

As we have seen in the previous proof the property of the m here being the maximal power of p dividing the order of G had been internalized which made the theorem more general but the proof harder (i.e. in the combinatorial argument).

3. Formalization of Groups in Isabelle HOL

The proof of Sylow's theorem demands a formalization of groups on a fine scale. We need to consider the group's carrier as a set which has different kinds of subsets, e.g. subgroups, cosets, and arbitrary subsets of certain cardinalities. These subsets play a rôle in the reasoning about the group's factorization in terms of the equivalence relation which is used in the construction of the Sylow subgroup. Hence, we have to be able to view the group's constituents from some completely different perspectives.

After giving a short introduction to Isabelle we explain the formalization of groups we used to handle these different views on groups in our experiment.

3.1. Isabelle

The theorem prover Isabelle [Pau94] is a generic interactive theorem prover. It is generic in the sense that it is a system that can be easily instantiated to form theorem provers for arbitrary logics [Pau90]. Thus, it is well suited for the development and test of new logics. These can be made known to the prover by defining theories that contain sort and type declarations, constants, and related definitions and rules. A powerful parser allows to produce intelligible syntactic abbreviations for user-defined constants.

Apart from being a tool for logical developments, some instantiations of Isabelle have independent value as theorem provers. These are: Zermelo-Fraenkel set theory (ZF), higher order logic (HOL) and constructive type theory (CTT). The best developed and most widely used ones are ZF and HOL. Substantial case studies have been performed in both of them [Pau95].

3.2. Groups in Isabelle

For the formalization of Sylow's theorem we used the theory HOL of Isabelle. We preferred it to pure set theory represented by ZF because we wanted to employ polymorphism for the abstraction over the base set of a group. HOL offers a formulation of typed sets. Sets are here basically a syntactical abbreviation for predicates. By switching in between the set representation and the corresponding predicate we can combine convenience of mathematical notation with the power of higher order logic reasoning with types.

To encapsulate the definition of a group by its operations and corresponding axioms we employ the following definition of groups as a typed set of quadruples:

```
Group_def "Group ==
{(G,f,inv,e). f ∈ G -> G -> G & inv ∈ G -> G & e ∈ G &
           (∀ x ∈ G. ∀ y ∈ G. ∀ z ∈ G. (f (inv x) x = e) &
                 (f e x = x) & (f (f x y) z = f (x) (f y z))))}"
```

The function constructor `A -> B` constructs the set of functions from a set `A` to `B`. The constant `Group` has type `'a set` which is remarkable and only possible because the type of `(G,f,inv,e)` is a product. Products are in HOL internalized types and hence `'a` can be instantiated to the type of `(G,f,inv,e)`.

3.2.1. *Basic Properties*

The definition of `Group` admits stating that a term `G` is a group quite concisely as `G ∈Group`. Unfolding the definition of `Group`, represented by the above set, yields all the defining properties for the constituents of `G`. Naturally, we need to define projection functions for these constituents. Generally, they are available by the projection functions for the product type of HOL, e.g. the function *first* would return the set underlying the group. But, to make notations more self-explanatory we overload the projections for the quadruple `(G,f,inv,e)` with names symbolizing the meaning of the constituents (`carrier`, `bin_op`, `invers`, `unity`).

We have to develop the axioms out of the definition — an additional cost for the neat representation — but this is very schematic and could be optimized by Isabelle's tactics to a high extent.

The definition of groups only assumes the minimal axioms, e.g. for the inverse only the *left* inverse rule $a^{-1}a = e$. We derive from the group definition a number of corresponding meta-level rules. For example `invers_ax2`:

```
[| G ∈ Group; a ∈ carrier G |]
==> bin_op G (invers G a) a = unity G
```

is the meta-level correspondence to the left inverse rule. For the closure properties, e.g. $inv \in G \to G$, we derive a more applicable rule form from the definition, for example `invers_closed`:

```
[| G ∈ Group; a ∈ carrier G |] ==> invers G a ∈ carrier G
```

The symmetric properties, like the right inverse rule are then derived from them in the classical way: first we prove the left cancellation law for the binary operation $xy = xz \Rightarrow y = z$ (`left_cancellation`) and from that the symmetric unity rule $ae = a$; now, we can prove that $aa = a \Rightarrow a = e$ (`idempotent_e`) and by that the symmetric inverse rule (`invers_ax1`). Finally, we can prove with the latter two the right cancellation law.

3.2.2. *Subgroups*

Building onto the basic properties of groups we consider the notion of a subgroup using the syntax `H <<= G` for *H is subgroup of G*. In the definition of the subgroup property we can use an elegant approach which reads informally: *a subset H of G is a subgroup if it is a group with G's operations.*

```
subgroup_def "H <<= G ==
H <= carrier(G) & (H,bin_op(G),invers(G),unity(G)) ∈ Group"
```

It is not completely trivial that this definition is possible, because it depends on the way that groups are formalized (cf. Section 5.1).

Basic derived results are `SG_unity` — the unit of G is an element of every subgroup — from which we get that a subgroup is nonempty, or `0 < card(H)` (`SG_card1`). Related theorems about subgroups are that they are closed under product, i.e. $a, b \in H \Rightarrow ab \in H$ and under inverse, i.e. $a \in H \Rightarrow a^{-1} \in H$ (`SG_bin_op_closed` and `SG_invers_closed`).

An introduction rule for the subgroup property is `subgroupI`:

```
[| G ∈ Group; H <= carrier G; H ~= {}; ∀ a ∈ H. invers G a ∈ H;
   ∀ a ∈ H. ∀ b ∈ H. bin_op G a b ∈ H|] ==> H <<= G"
```

That is, for a nonempty subset H of a group G it is sufficient to check that it is closed under inverse and binary operation to conclude that H is a subgroup. Ideally, we want to have an introduction rule where it is sufficient to show that a nonempty subset of G is closed under `bin_op G` to gain the subgroup property. Actually, this is the characterization of subgroup used in the Sylow proof (cf. Section 2.1.2). But, this result uses an argument about finite sets and repetitions of a^n for $n \to \infty$ if G is finite which is quite complicated to prove formally. On the other hand, it is straightforward to prove the additional closure under inverse construction for the Sylow subgroup. Hence, we deviate at this single point from the mathematical proof of Sylow's theorem by using the longer characterization `subgroupI` (cf. Section 4).

The proof of Sylow's theorem uses Lagrange's theorem as well as an equivalence relation which is ranging over subsets with p^α elements. For both we need the notion of *cosets*.

3.2.3. *Factorization of Groups*

If H is a subgroup of a group G then the *right coset* of a with respect to H in G — Ha — is the set $\{ha \mid h \in H\}$. We consider only right cosets here and sometimes refer to them as just *cosets*. The division of a group into cosets is a partition. The coset construction is needed when we consider so-called factorizations of a group. Then we look at H, the factor, as the unit and each coset as a member of the factorization with respect to the induced operation on cosets. An interesting point is to find out how the induced operation behaves on the factorization. For example, one can reason about the criteria which make the factorization G/H together with the induced operation on the cosets again a group.

Though the construction of a group factorization is defined merely for subgroups it can as well be applied to arbitrary subsets of groups. Hence, in our definition we leave out the condition that the factor is a subgroup and define `r_cosets` as

```
r_coset_def "r_coset G H a == {b . ∃ h ∈ H. bin_op G h a = b}"
```

The definition is equivalent to $\{ha \,|\, h \in H\}$[1].

To be able to talk about the factorization of a group into cosets we further define the set of right cosets G/H as:

```
set_r_cos_def
"set_r_cos G H == {C . ∃ a ∈ carrier G. C = r_coset G H a}"
```

The notation for `r_coset` is not very satisfying. It is necessary to quote the group `G` in `r_coset G H a` for which we consider the coset construction. The mathematical notation is just Ha where the group G to which we refer should be clear from the context. Here, we need more notational support to get at least some notation like `H #> a`. This issue is addressed when we perform the actual Sylow proof (cf. Section 4). For the purpose of general group results we deal with the verbose notation.

To prepare for the reasoning with cosets we derive some theorems about cosets. They are partly concerned with the arithmetic for the induced operation: `coset_mul_assoc`, `coset_mul_unity`, `coset_join1`, `coset_join2`, `coset_mul_invers1`, and `coset_mul_invers2`. Further results are: the union of the set of all cosets equals the group itself (`set_r_cos_part_G`), cosets are subsets of G (`r_cosetGHa_subset_G`), cosets have equal cardinality (`card_cosets_equal`), unequal cosets are disjoint (`r_coset_disjunct`), and the set of cosets is a subset of the powerset of G (`set_r_cos_subset_PowG`).

The last few of these general results join together to prove Lagrange's theorem as we shall see in the following section.

3.2.4. *Lagrange's Theorem*

In contrast to the formalization as seen in [Yu90] the form of Lagrange that we need here is not just the one stating that the order of the

[1] Actually, Isabelle HOL offers a set definition as follows $\{ha \,|\, h.h{\in}H\}$ which is an abbreviation for the above definition but which we abandon here to make the formalization easier to understand.

subgroup divides the order of the group but instead gives the precise representation of the group's order as the product of order of the subgroup and the *index* of this subgroup in G, i.e.

THEOREM 2. *If G is a finite group and H is a subgroup of G, then* $o(G) = |H| * |G/H|$

The term G/H stands for G *modulo* H and is the factorization of G in right cosets of H. Its cardinality $|G/H|$ is defined as *index of H in G.* We sketch the proof here already in terms of our formalization.

Proof. The proof of this theorem in our Isabelle formalization of group is quite straightforward. The basic idea is to reduce it to theorems about cosets using a fact that we can derive in general for finite sets (`card_partition`):

```
[| finite C; finite (Union C); ∀ c ∈ C. card c = k & finite c;
∀ c1 ∈ C. ∀ c2 ∈ C. c1 ~= c2 --> c1 ∩ c2 = {} |]
  ==> k * card(C) = card (Union C)
```

Application of this to the original conjecture leaves us with the following subgoals:

```
1. finite (set_r_cos G H)
2. finite (Union set_r_cos G H)
3. ∀ c ∈ set_r_cos G H. card c = k & finite c;
4. ∀ c1 ∈ set_r_cos G H.
   ∀ c2 ∈ set_r_cos G H . c1 ~= c2 --> c1 ∩ c2 = {}
```

The group G is finite by assumption. The subgoals `3` and `4` are the rules mentioned in the previous Section: the cardinality of cosets is equal, and since they are subsets of G they are all finite. Their intersection is pairwise empty. Another derived result states that the powerset of a finite set is finite (`finite_Pow`). Together with `set_r_cos_part_G` and `set_r_cos_subset_PowG` we get also `1` and `2`. The finer scale of formalization here leaves us with a quite general and concise proof.

4. Sylow's Theorem in Isabelle HOL

As mentioned in Section 3.2.3, the syntax for cosets is not very intelligible. Also, we would like to have a nicer notation for the group's binary operation. So far we must write `bin_op G a b` for *ab*. Since we need to quote the group G, the only way around this difficulty at the present state of Isabelle seems to abuse the theory file technique and declare a constant `G` and a type `i` of elements of `G` to build a context for the Sylow proof. To get nicer syntax and also to avoid repeating global premises in each subtheorem of the proof of Sylow's theorem — i.e. to save listing:

```
G ∈ Group, finite G, p ∈ prime, o(G) = (p ^ a)* m
```

— we define a theory Sylow.thy which contains a type i, a constant G, and the above premises as rules. We plan to extend Isabelle's reasoning facilities to allow to build temporary contexts for such purposes (cf. Section 5.2). Generally, the proof can be easily transformed into the explicit version abandoning the syntactical improvements, hence our approach here does not influence the soundness of the formalization of Sylow's theorem.

With this context at hand we can abbreviate the verbose notation for bin_op and r_coset in terms of the constant G by the definitions:

```
  r_coset_abbr  "H #> x == r_coset G H x"
  bin_op_abbr   "x # y  == bin_op G x y"
```

Furthermore, we can define unity G as e and invers G a as inv a. In the Sylow theory we additionally define an identifier for the set $\mathcal{M}$ of p^α-subsets of G and for the equivalence relation $\sim$ over this set:

```
  "calM == {s. s <= carrier(G) & card(s) = (p ^ a)}"
  "RelM  == {(N1,N2).(N1,N2) ∈ calM × calM &
                    (∃ g ∈ carrier(G). N1 = (N2 #> g))}"
```

4.1. Prerequisites

Besides the theorems about groups and cosets, we need to derive properties about finite sets and cardinalities for this case study, as the already mentioned card_partition or finite_Pow. Furthermore, some additional arithmetical results are needed and theorems and functions of combinatorics, e.g. binomial coefficients, numbers of subsets, and divisibility rules, have to be defined or derived. In the present section we show some of these; when displaying the rules we often leave out preconditions to enhance readability. When not introduced in the text, names of theorems are displayed in square brackets at the right margin. The function nat_rec defines primitive recursive functions.

4.1.1. *Arithmetic*

The arithmetical rules which have to be derived in addition to the already existing ones of the Isabelle theory Arith are mostly obvious but some are nevertheless nontrivial. For example, the rule

```
a - (b - c) = a - b + c                                 [diff_assoc]
```

needs the additional assumptions `c <= b` and `b <= a` because `-` is the difference for natural numbers. Similarly, the left cancellation law for natural number multiplication

```
k * a = k * b ==> a = b                        [mult_left_cancel]
```

is not generally valid (`k` must not equal zero).

Apart from such additional theorems about already existing functions, we define for the the present case study an integer power operation by

```
power_def   "m ^ n == nat_rec 1 (λ u v. m * v) n"
```

The use of the primitive recursion given by the functional `nat_rec` allows us to derive typical properties of this power function.

Finally, we have to define divisibility and derive basic facts about it. The most advanced rule about divisibility we derive is `div_combine` — the main argument for the proof part of Section 2.1.3.1 is to show that $p^\alpha \,|\, o(H)$:

```
[|...;~(p ^ (r+1) | n); p ^ (a+r) | n * k |] ==> p ^ a | k
```

Prime numbers are defined as a set, letting us quantify over all primes. Observe the syntactical overloading of the operator `|`, once as divisibility and once as logical or:

```
prime_def "prime ==
{p. 1 < p & (∀ a b. p | a * b --> (p | a) | (p | b))}"
```

We do not need many extra theorems for primes.

4.1.2. *Finite Sets and Cardinalities*

We have to prove that the cardinality of a finite set A is less or equal the cardinality of a finite set B if there exists an injection from A into B (`card_inj`). From this, we can immediately show that if there is a bijection from A to B then their cardinalities are equal (`card_bij`).

We derive the already mentioned `finite_Pow` — the powerset of a finite set is finite — and the counting theorem `card_partition` (cf. Section 3.2.4). Furthermore, we need in the Sylow proof that if a number k divides the cardinalities of all classes of a set S factorized by an equivalence relation then k divides the cardinality of S (`equiv_partition`).

4.1.3. *Binomial Coefficients and k-subsets*

The definition of the choose operator is inspired by the HOL tutorial [Hol] by taking the Pascal triangle for the definition of $\binom{n}{k}$ instead of a ratio of factorials. We use again the nat_rec functional to define $\binom{n}{k}$ as

```
choose_def "binomial n == nat_rec (λ k. if k = 0 then 1 else 0)
          (λ u r k. if k = 0 then 1 else (r (k -1) + r (k))) n"
```

By this definition we gain the primitive recursive function binomial n for each n which behaves for each k as we want it, i.e. (with the infix syntax definition n choose k for binomial n k) we get

```
Suc n choose Suc k =                                 [chooseD_add]
(n choose Suc k) + (n choose k)
```

From that we get all other necessary prerequisites quite easily: n_choose_0, zero_le_choose, less_choose, n_choose_n, choose_Suc, and n_choose_1 for the basic ones and a multiplicative decomposition:

```
k <= n ==>                                          [chooseD_mult]
Suc n * (n choose k) = (Suc n choose Suc k) * Suc k
```

From that we can derive the theorems

```
k <= n ==> (Suc n * (n choose k)) mod Suc k = 0    [choose_mod1]
k <= n ==>                                          [choose_defT]
(Suc n choose Suc k) = (Suc n *(n choose k))div Suc k
```

which are decisive in the first combinatorial part of Sylow's proof.

We can now prove:

```
card {s. s <= M & card s = k} = (n choose k)         [n_subsets]
```

if card M = n and k <= n. In the induction scheme for finite sets, applied to M, we can use a x not in M. Using the decomposition:

```
{s. s<=insert x M & card s = Suc k} = {s. s<=M & card s = Suc k}
∪ {s. ∃ s1 ∈ {s. s<=M & card s = k}. s = insert x s1 }
```

for x not in M, we can use the induction hypothesis to show that

```
card {s. s <= M & card s = Suc k} = ((card M) choose (Suc k))
```

on the one hand. On the other hand, we construct a bijection between

```
{s. ∃ s1 ∈ {s. s <= M & card s = k}. s = insert x s1}
```

and

```
{s. s <= insert x M & card s = k}
```

provided that x is not in M. We use the induction hypothesis again to show that

```
card {s. ∃ s1 ∈ {s. s <= M & card s = k}. s = insert x s1} =
((card M) choose k)
```

The cardinalities we derived for the two components of the decomposition match the formula `chooseD_add`. Hence, after showing that the cardinality of the union of two disjoint sets is just the sum of the cardinalities of these sets, we can apply `chooseD_add` to finish the proof of the theorem `n_subsets`.

4.1.4. *Preparation for Combinatorial Argument*

The combinatorial argument of Sylow's proof is formalized by first defining a maximum number predicate `max-n` as

```
max_nat_def  "max-n k. P(k) == @k. P(k)&(∀ m. k < m --> ~P(m))"
```

Thereby, we can state the combinatorial argument as

```
(max-n r. (p ^ r | m)) =                             [const_p_fac]
(max-n r. (p ^ r | (((p ^ a) * m) choose p ^ a)))
```

which is a natural way of encoding this proposition. Unfortunately, the `max-n` construct uses the Hilbert-operator `@` which names an element that fulfills a given predicate `P` but forces us to show the existence of such an element first. To make the proof easier we observe that the maximum power of p dividing a number n is the integer logarithm of n to the base p. By defining this integer logarithm function as

```
log_def "log p n == wfrec (trancl pred_nat)
              (λ f j. if ((0 < j) & (j mod p = 0))
                      then Suc(f (j div p)) else 0) n"
```

we gain the desired function which improves the performance of the main proposition `const_p_fac`. We first show some properties about this logarithm function to allow later calculations.

We show that this logarithm represents actually the maximum power of p dividing a number s by deriving

```
p ^ log p s | s                                        [max_p_div]
& (∀ m. log p s < m --> ~(p ^ m | s))
```

This enables us to replace the `max-n` term by a `log` term and we can derive the unique existence of the logarithm (`unique_max_power_div_s`, `log_p_unique`, `max_p_div_eq_log`).

The theorem we finally use in the combinatorial argument is a combination of the latter ones, namely:

```
[|...; ∀ r. ((p ^ r | a) = (p ^ r | b))|]          [div_eq_log_p]
==>  log p a = log p b
```

The results we need to calculate with the new logarithm operation are

```
n = (p ^ log p n)*(n div (p ^ log p n)) [log_power_div_equality]
```

and

```
log p (a * b) = (log p a) + (log p b)             [log_mult_add]
```

4.2. Proof

According to the structure of the mathematical proof we present the formal proof of Sylow's theorem in three parts.

4.2.1. *Combinatorial Argument*

We have to show the conjecture const_p_fac (cf. Section 4.1.4). Using unique_max_power_div_s we can immediately reduce the conjecture to the logarithm equality:

```
log p m = log p ((p ^ a)* m choose p ^ a)
```

By chooseD_mult (or choose_defT, more precisely) this can be transformed into

```
log p m = log p ((p^a)* m *((p^a)*m-1 choose (p^a)-1) div (p^a))
```

Cancellation (div_mult1) yields:

```
log p m = log p (m * ((p ^ a)* m - 1 choose (p ^ a) - 1))
```

which can be decomposed by log_mult_add into

```
log p m = log p m + log p ((p ^ a)* m - 1 choose (p ^ a) - 1)
```

By arithmetical rules (add_0_right, add_left_cancel) we reduce to

```
0 = log p ((p ^ a)* m - 1 choose (p ^ a) - 1)
```

which can be derived from:

```
~( p | ((p ^ a)* m - 1 choose (p ^ a) - 1))
```

So far the calculation above is straightforward. That p does not divide the remaining ratio or choose formula can also be shown following the outline of the mathematical proof. To this end, we derive the two directions. The forward direction is p_fac_forw:

```
[|...; k < (p^a); (p^r) | (p^a)* m - k |] ==> (p^r) | (p^a)- k
```

for which we have to derive r <= a as in the mathematical proof under the same premises as above. The backward direction p_fac_backw:

```
[|...; k < (p^a); (p^r) | (p^a)- k |] ==> (p^r) | (p^a)* m - k
```

needs again `r <= a`. Now, we derive a theorem characterizing `(p | n choose k)` if `log p k = log p n` under more general preconditions (`p_not_div_choose`). Instantiating the latter to $p^\alpha m - 1$ and $p^\alpha - 1$ by plugging in the previous two lemmas we attain

```
~(p | ((p ^ a) * m - 1 choose (p ^ a) - 1))  [const_p_fac_right]
```

which solves the combinatorial argument.

4.2.2. *Construction of the subgroup H*

Another additional cost for the nicer representation is that we have to instantiate the rules derived for groups to the constants of the theory Sylow. The instantiations of the rules are marked by a leading `I` in their names, e.g. `Ibin_op_closed` is the instantiation of `bin_op_closed` to the proof context and reads

```
[|x ∈ carrier G; y ∈ carrier G|] ==> x # y ∈ carrier G
```

Before we start to get into the concrete parts of the proof we have to derive some facts about `calM` and `RelM`. First, `RelM` is an equivalence relation over `calM` (`RelM_equiv`). The proof is a straightforward check of the definition of equivalence relation.

The assumptions $M \in \mathcal{M}/\!\sim$, $p^{r+1} \not| card(M)$, and $M_1 \in M$ are always made in the following derivations. They have to be proved finally, then they become obsolete at the top level of the proof.

Under the assumption of `M` and `M1` we prove now

```
{g. g ∈ carrier G & M1 #> g = M1} <<= G
```

Here we see that our approach to abuse the theory mechanism to get a nicer syntax pays off. The syntactical form of the constructed subgroup is very similar to the mathematical notation.

As already mentioned in Section 3.2.2 we use the alternative characterization `subgroupI` to tackle this task. It leaves us with some subgoals of which the first one is

```
{g. g ∈ carrier G & M1 #> g = M1} ~= {}
```

It can be solved by showing that `e` is in this set. The two closure conditions, i.e. the potential subgroup is closed under the binary operation and under inverse construction, are derivable by insertion and definition expansion.

We avoid defining an abbreviation for the constructed subgroup in the proof because this construction is only visible inside the proof and vanishes at the top level.

4.2.3. *Cardinality of H is p^{α}*

The most difficult part of this subproof is to construct the bijection between M and `set_r_cos G H` (or G/H, respectively). Though mathematically it is sufficient to derive $(Ha = Hb) \equiv (M_1 a = M_1 b)$ and to define the actual bijection as the mapping $Ha \mapsto M_1 a$ for all $a \in G$, this is not as easy formally. The problem is, we have, for one direction, some M_j which has an equivalent form $M_1 a$ for some a. Unfortunately, we do not know this a but need to use it to construct the inverse image of $M_j = M_1 a$ as Ha. For the backward map we have the same problem.

We solve this problem by employing the Hilbert operator `@`, though this makes the proof quite messy. We tackle the bijection to two injections $f \in M \rightarrow G/H$ and g vice versa. They are (where H abbreviates {`g. g` ∈`carrier G & M1 #> g = M1`})

```
λ M.  H #> (@g. g ∈ carrier G & M1 #> g = M)
λ C. M1 #> (@g. g ∈ carrier G &  H #> g = C)
```

That is, we just define these maps by the properties we expect from them — this is like saying that the map is $Ha \mapsto M_1 a$ but is less clear; we do not know this a instead just describe it by its properties.

Though the terms in this proof grow quite large, making the derivation hard to read, the proof is again straightforward. Additional results needed in the course of the derivation are

```
card(M1) = card(M1 #> g)                        [M1_RelM_rcosetGM1g]
(M1, M1 #> g) ∈ RelM                         [M1_card_eq_rcosetGM1g]
```

With the bijection at hand, we solve the index lemma:

```
card(M) * card{g. g ∈ carrier G & M1 #> g = M1}       [index_lem]
= card G
```

by reducing it to Lagrange's theorem. This reduction is performed by the derivation `card_M_eq_IndexH` which entails the constructed bijection `bij_M_GmodH`.

The two inequalities yielding $o(H) = p^{\alpha}$ can now be derived by plugging in all the prepared theorems according to the textbook proof.

1. `p ^ a <= card {g. g ∈ carrier G & M1 #> g = M1}`

   This task can be reduced by the divisibility lemma `div_order` to

   ```
   p ^ a | card {g. g ∈ carrier G & M1 #> g = M1}
   ```

   The main divisibility rule for this subproof (`div_combine`) leaves us with

```
1. ~(p ^ (max-n r. p ^ r | m)+ 1 | card(M))
2. p ^ (a + max-n r. p ^ r | m) |
        card(M) * card {g. g ∈ carrier G & M1 #> g = M1}
```

The first subgoal is entailed in the assumption about `M` for this proof. To the second one we can apply the previously derived index lemma to transform into

```
p ^ (a + max-n r. p ^ r | m) | order(G)
```

After replacing `order(G)` by `(p ^ a)* m` this can be reduced by basic arithmetic and divisibility rules to

```
p ^ (max-n r. p ^ r | m) | m
```

which is entailed in `max_p_div` (cf Section. 4.1.4).

2. `card {g. g ∈ carrier G & M1 #> g = M1} <= p ^ a`

   We substitute `card(M1)` for `p ^ a`. By the preconstructed injection of M_1 into H (`M1_inj_H`), discussed in Section 2.1.3.2 the task is reduced to subgoals `finite M1` and `finite H` which can be solved by the already derived lemmas (see above).

The main proof finally puts together the parts `H_is_SG`, `lemma_leq1`, and `lemma_leq2` (the above inequalities 1 and 2).

Still, we have the two assumptions

```
M ∈ calM / RelM & ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M))
M1 ∈ M
```

We discharge the former one finally in the top level proof `Sylow1` by applying `lemmaA1`. This lemma proves the existence of such an M by contraposition. Assuming for contradiction that such a set does *not* exist, we would have

```
∀ M ∈ calM / RelM. p ^ ((max-nr. p ^ r | m) + 1) | card M
```

But then, since equivalence classes are a partition (`equiv_partition`)

```
p ^ ((max-nr. p ^ r | m) + 1) | card calM
```

Since `card calM` $= \binom{p^\alpha m}{p^\alpha}$ this contradicts `const_p_fac_right`.

The assumption `M1 ∈ M` is canceled by the theorem `existM1inM`. The existence of a set M_1 in the class M, whose cardinality is not divided by p^{r+1}, is derived from $M \in \mathcal{M}/\!\sim$. Since the empty set is *not* a member of $\mathcal{M}/\!\sim$, but M is a member, it follows that M is not the empty set; hence we can assume an $M_1 \in M$.

The formal proof of Sylow's theorem necessitates some smaller lemmas not visible in the textbook proof. They are mostly concerned with the existence of elements in the sets $\mathcal{M}$, M or M_1, inclusions between those sets and G and the cardinalities of those sets. Their names indicate this already: `zero_less_oG`, `zero_less_m`, `card_calM`, `exists_x_in_M1`, `M1_subset_G`, `finite_calM`.

5. Structural Concepts for Abstract Algebraic Reasoning

As we have seen in the formalization of groups, there is a need to support structures to enhance the reasoning in abstract algebra. We modeled structures by defining a typed set for the structure of groups and a single theory for the definitions and assumptions of Sylow's theorem. The former case required a lot of extra work for the neat representation (cf. Section 3.2.1) and the method for the latter case is an abuse of the theory facility because we define a *constant* `G` to be able to define readable syntax. The present section wants to mention the concept of modules for theorem provers which seems to improve the representation of structures in algebraic reasoning but has severe limitations. Hence, we propose another concept for our further research in Section 5.2. Finally, Section 5.3 gives some conclusions on this case study.

5.1. Module Systems

An algebraic structure, like a group, describes an abstract object consisting of a set and operations on this set. The abstraction is meant to admit generalization over different sets, for example rational numbers or $\mathbb{Z}_5$. We want to prove facts about the general structures but they should be applicable to concrete examples; otherwise the results would be useless.

Module systems in theorem provers, e.g. IMPS, PVS [OSR93], and Larch [GH93], are designed for encapsulating types and predicates or functions over those types. These modules are also called *theories* and allow their body to assume properties for its parameters. In descriptions of such tools we find suggestions how to represent abstract algebraic structures — groups, rings, and so forth — in terms of these module concepts as:[2]

[2] We use a *pseudo* module notation vaguely similar to the existing ones.

```
theory group (G: TYPE, bin_op: G -> G -> G, inv: G -> G, e: G)
begin
  unity_ax_1: bin_op a e = a
  ...
end
```

The advantage of this treatment of abstract algebra is in the reuse possibilities and concise structuring of the group theory. This corresponds nicely to some processes of algebraic reasoning. The properties of groups are only visible in the context of the theory. Thus, reasoning about elements of groups and its operations can be performed in the scope of the theory of groups and derived theorems are visible inside there. They can be invoked by an instantiation of the theory.

But, this representation of groups has some drawbacks. For example, if we look at subgroups we see that the modeling in terms of modules is not appropriate. For the notion of subgroups we might want to characterize[3] a subgroup as a subset H of a group G such that H is a group together with G's operations. Though there might be possibilities to approximate this in the described representation of groups by modules, e.g. IMPORTING group(H,bin_op,inv,e) in PVS or a theory interpretation of group to *itself* with some obscure subset H of (the type!) G, these possibilities are quite artificial. Moreover, IMPORTING and interpretations do not represent logical formulas. Instead, these invocations of the group module represent a kind of check or meta-level knowledge that the actual parameter H is a subgroup; we get all implications about the constituents of the subgroup but not a formula stating the subgroup property. In other words, modules are not *first class citizens* of the logic and thereby items defined in terms of modules cannot be accessed by logical reasoning.

A reasonable conclusion from this observation is to abandon the use of modules for abstract algebra, at least if we do not want to restrict the reasoning to merely *internal* properties of groups. Most advanced algebraic theorems, as for example Sylow's theorem, demand more. Naturally, we could also perform the formalization and proofs of the current paper in IMPS, PVS or Larch, respectively, but not in terms of the characterization of groups by modules as seen above.

Modules are a sensible concept for organising logical theories, but not for a complete and natural representation of abstract algebra. Still, it would be nice to preserve the obviously neat properties of an encapsulated object representing mathematical entities like groups. To this end, we use the the case study of Sylow's theorem to formulate

[3] as in our formalization of subgroups for Sylow's theorem

a concept of *sections* which are more appropriate for the needs of algebraic reasoning.

5.2. Sections

The concept of a section described here is tentative. We state the requirements for sections; the actual construction was still under consideration when the present paper was submitted. The concept resembles the ones in Coq [Dow90] and AUTOMATH [DeB91, DeB80]. In the meantime we implemented a corresponding concept of *locales* [KW98] in Isabelle.

Sections delimit a scope in which assumptions can be made and theorems depending on these assumptions are proved. Sections resemble modules but their parameterization is restricted to terms of the logic. They can be regarded as abbreviations for formulas of the meta-logic; a section using the assumptions $\Gamma_1, \ldots, \Gamma_n$ proving the theorems $t_1, \ldots, t_n$ may be regarded as the Isabelle meta-level formula
`[|` $\Gamma_1; \ldots; \Gamma_n$ `|] ==>` t_1 `&` $\ldots$ `&` t_n.

This allows us to regard an object defined by a section from the inside, when it is defined or invoked, or from the outside which is the meta-level view.

Inside a section, a concrete type, like `i` in the Sylow case study, might be necessary to disambiguate the notation. But, if the section is considered from the outside, this type must vanish; possibly by employing polymorphism.

The present case study gives an example for the use of sections. The theory we defined for Sylow's theorem can be represented by a section. The constant G and the assumptions of the theorem are the contents. The outside view is the theorem in the usual form: all assumptions as premises of the meta-level. The definition of groups itself is a further example where sections can be applied.

To realize the concept of sections the possibility of syntax definition in Isabelle has to be extended to allow us the same readable formalization as in the present case study. Invocation of a section is complicated because we have to consider multiple invocations and nested sections.

5.3. Conclusion

Sylow's theorem is a fundamental result of finite group theory. It usually stands at the end of a lecture course on group theory because it unifies most of the basic results about finite groups. A lot of textbooks on abstract algebra skip the theorem, or at least its proof, because it is difficult. The proof by H. Wielandt that is the raw model for our formalization is quite concise (less than one page). Herstein [Her64] being a bit more detailed uses almost two pages. In our introduction we need three pages.

The formalization of Sylow's theorem is quite a big experiment. Altogether we proved 278 theorems, of which only 52 are in the theory `Sylow`. The theory `Group` contains 121 theorems of which only 34 are concerned with group properties; the other 87 are dealing with the self defined operators: natural number logarithm, power, choose operator, the function set constructor `->`, and bijections (which use `->`). Though these are mostly arithmetical results, one could not expect them to be in the theorem library because the operators are rather exceptional and did not exist in Isabelle before. Another 104 theorems are extensions to the existing Isabelle theories of the HOL logic, sets, arithmetic, finite sets, equivalence relations, products and natural numbers.

The reasoning processes are quite subtle, mainly in the combinatorial argument. A lot of the reasoning deals with divisibility and finite set properties. As other formalizations show [PG96], reasoning with finite sets is tricky — though most arguments seem intuitively obvious. It necessitates a quite well structured analysis to avoid losing track of the proof. Often during the development we had severe problems because of small side conditions. Some simple looking subtheorems turned out to be quite hard to prove. One could say that this is typical in proof development but we think that it is obvious from the present paper that the proof is just very subtle though elegant.

Elegant constructions and proofs can be especially hard to mechanize. That is, they connect different domains of reasoning by unusual associations. This is intuitively appealing, but tedious for mechanical proofs. We are lucky if there is another unelegant but straightforward proof if we want to mechanize it.

What do we learn from the experiment? First of all we think that hard proofs are good benchmarks for systems because they are well suited to reveal incompleteness of formal approaches. Tools or frameworks constructed for formal proofs might turn out to be not suited for intuitive associations connecting domains of mathematical reasoning. Especially, methods which prescribe the way of approaching a problem are naturally inflexible and can only be changed with unnatural complications to solve nonstandard tasks. An example for this are the module systems.

Another point which becomes obvious by performing big or difficult case studies is how far theory libraries are sufficiently equipped with theorems for such tasks. Surely, it is an impossible task to provide all possible lemmas which might arise in some obscure proof, but at least a certain level of knowledge about predefined theories must be provided. The Isabelle theory library is most sufficient, though some probably not too frequently demanded theorems about equivalence relations and finite sets caused some additional difficulties.

Last but not least, we have to admit our admiration for mathematicians. While reconstructing a hard mathematical proof from the pure logic, one comes to points where it seems that there must be mistakes in the proof — but then it turns out that it is sound and just a bit trickier than expected. Having machine support to reassure about difficult theorems makes it almost impossible to believe that this can be performed soundly by just relying on a human brain which is often biased by the desire to solve the problem.

Appendix

A. Theory Files

This appendix contains the code of the two Isabelle theory files for groups and the one for the proof of Sylow's theorem providing a context for the proof. The prefix `!` is the universal quantifier, `:` represents $\in$ for HOL sets, and the question mark `'?'` is the existential quantifier of HOL. The symbol `%` stands for λ; `Un` stands for set union $\cup$.

A.1. Group Theory

```
Group = Univ + Finite + Equiv +
consts
  prime   :: "nat set"
  divides :: "[nat, nat] => bool"     ("_ | _" [50,51]50)
  power   :: "[nat,nat] => nat"       ("_ ^ _" [60,61]60)
  log     :: "[nat,nat] => nat"
  binomial:: "[nat,nat] => nat"       ("_ choose _" [50,51]50)
defs
  prime_def   "prime ==
         {p. 1 < p & (! a b. p | a * b --> (p | a)|(p | b))}"
  divides_def "a | b == ? (k :: nat). b = k * a"
  power_def   "m ^ n == nat_rec 1 (%u v. m * v) n"
  choose_def  "binomial n ==
               nat_rec (%k. if k = 0 then 1 else 0)
        (% u r k. if k = 0 then 1 else (r (k -1) + r (k))) n"
  log_def "log p n == wfrec (trancl pred_nat)
           (% f j. if ((0 < j) & (j mod p = 0)) then
                          Suc(f (j div p)) else 0) n"
consts
  funcset ::
  "['a set,'b set]=>('a => 'b) set"  ("_ -> _" [91,90]90)
  subgroup ::
  "['a set, ('a set *(['a, 'a] =>'a)*('a =>'a)* 'a)] => bool"
                                     ("_ <<= _" [51,50]50)
defs
  funcset_def "A -> B == {f. ! x:A. f(x) : B}"
consts
  Group :: "'a  set"
constdefs
  carrier :: "('a set*(['a,'a] =>'a)*('a =>'a)*'a)=>'a set"
             "carrier(G) == fst(G)"
  bin_op  ::
  "('a set *(['a,'a] =>'a)*('a =>'a)* 'a) =>(['a, 'a] =>'a)"
             "bin_op(G)  == fst(snd(G))"
  invers :: "('a set *(['a,'a] =>'a)*('a =>'a)*'a) =>('a =>'a)"
             "invers(G) == fst(snd(snd(G)))"
  unity     :: "('a set *(['a, 'a] =>'a)*('a => 'a)* 'a) =>'a"
 "unity(G)     == snd(snd(snd(G)))"
  order     :: "('a set *(['a, 'a] =>'a)*('a =>'a)* 'a) => nat"
   "order(G) == card(fst(G))"
consts
  r_coset ::
  "[('a set*(['a,'a]=>'a)*('a =>'a)*'a),'a set,'a]=>'a set"
  set_r_cos ::
  "[('a set*(['a, 'a] =>'a)*('a =>'a)*'a),'a set]=>'a set set"
  max_nat    :: "(nat => bool) => nat"     (binder "max-n" 10)
defs
  r_coset_def "r_coset G H a == {b.? h : H. bin_op G h a = b}"
  set_r_cos_def "set_r_cos G H ==
               {C . ? a: carrier G. C = r_coset G H a}"
rules
  subgroup_def "H <<= G ==  H <= carrier(G) &
                       (H,bin_op(G),invers(G),unity(G)) : Group"
  Group_def "Group ==
          {(G,f,inv,e). f : G -> G -> G & inv : G -> G & e : G &
                     (! x: G. ! y: G. !z: G. (f (inv x) x = e) &
                  (f e x = x) & (f (f x y) z = f (x) (f y z)))}"
  max_nat_def "max-n k. P(k) ==
               @k :: nat. P(k) & (! m. k < m --> ~P(m))"
end
```

A.2. Theory File for Sylow's Theorem

```
Sylow = Group +
types i
arities i::term
consts
```

```
  G :: "i set * ([i, i] => i) * (i => i) * i"
  p, a, m :: "nat"
  r_cos      :: "[i set, i] => i set"   ("_ #> _" [60,61]60)
  "#"       :: "[i, i] => i"           (infixl 60)
defs
  r_coset_abbr  "H #> x == r_coset G H x"
  bin_op_abbr   "x # y  == bin_op G x y"
constdefs
  e   :: "i"  "e == unity G"
  inv :: "i => i" "inv == invers G"
  calM      :: "i set set"
           "calM == {s. s <= carrier(G) & card(s) = (p ^ a)}"
  RelM      ::  "(i set * i set)set"
           "RelM  ==
 {(N1,N2).(N1,N2):calM Times calM &(? g:carrier(G).N1=(N2#>g))}"
rules
p1    "p : prime"
p2    "G : Group"
p3    "order(G) = (p ^ a) * m"
p4    "finite (carrier G)"
end
```

B. Some Theorems

This appendix contains some selected theorems of group theory, some that deal with the combinatorial argument, and all from the theory for Sylow's theorem. It does not contain any proofs.

B.1. Group Theory

```
coset_mul_assoc "[| G : Group; M <= carrier G; g : carrier G;
                h : carrier G |] ==> r_coset G (r_coset G M g) h
                = r_coset G M (bin_op G g h)";

coset_mul_unity "[| G: Group; x : carrier G; H <<= G; x : H |]
                ==> r_coset G H x = H";

coset_mul_invers1
                "[| G: Group; x : carrier G; y : carrier G;
                M <= carrier G;
                r_coset G M (bin_op G x (invers G y)) = M |]
                ==> r_coset G M x = r_coset G M y";

coset_mul_invers2
```

```
                "[| G: Group; x : carrier G; y : carrier G;
                M <= carrier G; r_coset G M x = r_coset G M y|]
                ==> r_coset G M (bin_op G x (invers G y)) = M ";

coset_join1     "[| G: Group; x : carrier G; H <<= G;
                r_coset G H x = H |] ==> x : H";

coset_join2     "[| G: Group; x : carrier G; H <<= G;
                x : H |] ==> r_coset G H x = H";

set_r_cos_part_G
                "[| G: Group; H <<= G|]
                ==> Union (set_r_cos G H) = carrier G";

rcosetGHa_subset_G
                "[| G: Group; H <= carrier G; a : carrier G |]
                ==> r_coset G H a <= carrier G";

card_cosets_equal
                "[|G : Group; H <= carrier G;
                finite(carrier G)|] ==>
                ! c: set_r_cos G H. card c = card H & finite c";

r_coset_disjunct
                "[| G: Group; H <<= G |] ==>
                ! c1: set_r_cos G H. ! c2: set_r_cos G H.
                c1 ~= c2 --> c1 Int c2 = {}";

set_r_cos_subset_PowG
                "[| G: Group; H <<= G |]
                ==> set_r_cos G H <= Pow( carrier G)";

Lagrange        "[| G: Group; finite(carrier G); H <<= G |]
                ==> card(set_r_cos G H) * card(H) = order(G)";
```

B.2. Combinatorics

```
n_choose_0      "(n choose 0) = 1";

zero_le_choose  "k <= n ==> 0 < (n choose k)";

less_choose     "n < k ==> (n choose k) = 0";

n_choose_n      "(n choose n) = 1";

choose_Suc      "(Suc n choose n) = Suc n";
```

```
n_choose_1       "n choose 1 = n";

max_p_div        "[| 1 < p; 0 < s |] ==> p ^ log p s | s &
                 (! m. log p s < m --> ~(p ^ m | s))";

unique_max_power_div_s
                 "[| 1 < p; 0 < s |] ==>
                 (max-n r. p ^ r | s) = log p s";

log_p_unique     "[| 1 < p; 0 < s |] ==> ?! x.  p ^ x | s &
                 (! m. x < m --> ~(p ^ m | s))";

max_p_div_eq_log
                 "[| 1 < p; 0 < s;
                 p ^ x dvd s & (! m. x < m  --> ~(p ^ m dvd s))|]
                 ==>  log p s = x";

div_eq_log_p     "[|1 < p; 0 < a ; 0 < b;
                 ! (r :: nat). ((p ^ r | a) = (p ^ r | b))|]
                 ==>  log p a = log p b";

log_power_div_equality
                "[|1 < p; 0 < n |] ==>
                n = (p ^ log p n)*(n div (p ^ log p n))";

equiv_partition
                "[| finite S; equiv S rel; ! x : S / rel.
                k dvd card(x)& finite x |] ==> k dvd card(S)";

constr_bij      "[| finite M; x ~: M |] ==>
                card {s. ? s1 :
                {s. s <= M & card(s) = k}. s = insert x s1}
                = card {s. s <= M & card(s) = k}";

n_subsets       "[| finite M; card(M) = n; k <= n |] ==>
                card {s. s <= M & card(s) = k} = (n choose k)";

Rettung         "!! p r m k. [| 0 < m; 0 < k;
                k < (p^a); (p^r) | (p^a)* m - k |]
                ==> r <= a";

p_fac_forw      "!! p r m k. [| 0 < m; 0 < k; p : prime;
                k < (p^a); (p^r) | (p^a)* m - k |]
                ==> (p^r) | (p^a)- k";
```

```
r_le_a_forw    "!! p r k. [| 0 < k; k < (p^a);
               0 < p; (p^r) | (p^a) - k |] ==> r <= a";

p_fac_backw    "!! p r m k. [| 0 < m; 0 < k; p : prime;
               k < (p^a); (p^r) | (p^a)- k |]
               ==> (p^r) | (p^a)* m - k";

logp_eq_logp   "[| p : prime; 0 < m |] ==>
               ! k n. (k < (p ^ a) & n < (p ^ a) * m &
               0 < k & 0 < n & n - k = (p ^ a) * m -  (p ^ a) &
               (p ^ a) <= (p ^ a) * m & k <= n)
               --> log p k = log p n";

p_not_div_choose
               "[| p : prime; ! k n. (k < p1 & n < p2 & 0 < k &
               0 < n & n - k = p2 - p1 & p1 <= p2 & k <= n)
               --> log p k = log p n ;p1 <= p2|] ==>
               k <= n & k < p1 & n < p2 & n - k = p2 - p1
               --> ~(p | (n choose k))";

const_p_fac_right
               "[| p : prime; 0 < m |] ==>
               ~(p | ((p ^ a) * m - 1 choose (p ^ a) - 1))";

const_p_fac    "[| p : prime; 0 < m |] ==>
               (max-n r :: nat. (p ^ r | (m :: nat))) =
               (max-n r.
               (p ^ r | (((p ^ a) * m) choose p ^ a)))";
```

B.3. Theory Sylow

```
RelM_equiv     "equiv calM RelM";

M_subset_calM  "M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M))
               ==> M <= calM";

card_M1        "[|M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M|] ==> card(M1) = p ^ a";

exists_x_in_M1 "[|M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M|] ==>  ? x. x : M1";

M1_subset_G    "[| M : calM / RelM &
```

```
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M|] ==>   M1 <= carrier G";

M1_inj_H       "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M|] ==>
               ? f: {g. g : carrier G & M1 #> g = M1} -> M1.
               inj_onto f {g. g : carrier G & M1 #> g = M1}";

RangeNotEmpty  "[| {} = RelM ^^ {x}; x : calM |] ==> False";

EmptyNotInEquivSet
               "{} ~: calM / RelM";

existsM1inM    "M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M))
               ==> ? M1. M1 : M";

zero_less_o_G  "0 < order(G)";

zero_less_m    "0 < m";

card_calM      "card(calM) = ((p ^ a) * m choose p ^ a)";

max_p_div_calM "~(p ^ ((max-n r. p ^ r | m)+ 1) | card(calM))";

finite_calM    "finite calM";

lemma_A1       "? M. M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M))";

bin_op_closed_lemma
               "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M; x : {g. g : carrier G & M1 #> g = M1};
               xa : {g. g : carrier G & M1 #> g = M1}|]
               ==> x # xa : {g. g : carrier G & M1 #> g = M1}";

H_is_SG        "[|M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M |] ==>
               {g. g : carrier G & M1 #> g = M1} <<= G";

M_elem_map     "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1: M; M2: M|] ==> ? g: carrier G. M1 #> g = M2";
```

```
H_elem_map     "[|M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M)); M1:M;
               H : set_r_cos G {g. g : carrier G & M1 #> g = M1}
               |] ==> ? g: carrier G.
               {g. g : carrier G & M1 #> g = M1} #> g = H";

rcosetGM1g_subset_G
               "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1: M; g : carrier G; x : M1 #> g |]
               ==> x : carrier G";

finite_M1      "[|M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1: M|] ==> finite M1";

finite_rcosetGM1g
               "[|M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1: M; g : carrier G|] ==> finite (M1 #> g)";

M1_cardeq_rcosetGM1g
               "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1: M; g : carrier G|]
               ==> card(M1) = card(M1 #> g)";

M1_RelM_rcosetGM1g
               "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1: M; g : carrier G|] ==> (M1, M1 #> g) : RelM";

bij_M_GmodH    "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M|] ==>
               (? f: M ->
               set_r_cos G {g. g : carrier G & M1 #> g = M1}.
               inj_onto f M ) &
               (? g: (set_r_cos G {g. g: carrier G & M1#>g=M1})
               -> M. inj_onto g
               (set_r_cos G {g. g: carrier G & M1 #> g = M1}))";

calM_subset_PowG
               "calM <= Pow(carrier G)";
```

```
finite_M       "M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M))
               ==> finite M";

cardMeqIndexH  "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M |] ==>
               card(M) =  card(set_r_cos G
               {g. g : carrier G & M1 #> g = M1})";

index_lem      "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M|] ==>
               (card(M)*card({g. g : carrier G & M1 #> g = M1}))
               = order(G)";

lemma_leq1     "[| M : calM / RelM &
               ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
               M1 : M |] ==>
               p ^ a <=
               card({g. g : carrier G & M1 #> g = M1})";

lemma_leq2    "[| M : calM / RelM &
              ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
              M1 : M |] ==>
              card({g. g : carrier G & M1 #> g = M1})
              <= p ^ a";

main_proof    "[| M : calM / RelM &
              ~(p ^ ((max-n r. p ^ r | m)+ 1) | card(M));
              M1 : M|] ==>
              {g. g : carrier G & M1 #> g = M1} <<= G &
              card({g. g : carrier G & M1 #> g = M1}) = p ^ a";

Sylow1        "? H. H <<= G & card(H) = p ^ a";
```

References

DeB91.. N. G. de Bruijn. Telescoping Mappings in Typed Lambda Calculus. *Information and Computation*, 91:189–204, 91.

DeB80.. N.G. de Bruijn. A Survey of the Project AUTOMATH. In J.P. Seldin and J.R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic*, Academic Press Limited, pages 579–606. 1980.

Dow90.. G. Dowek. Naming and Scoping in a Mathematical Vernacular. Technical Report 1283, INRIA, Rocquencourt, 1990.

FGT93.. W. M. Farmer, J. D. Guttman, and F. J. Thayer. IMPS: an Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11:213–248, 1993.

GH93.. John V. Guttag and James J. Horning, editors. *Larch: Languages and Tools for Formal Specification.* Texts and Monographs in Computer Science. Springer-Verlag, 1993. With Stephen J. Garland, Kevin D. Jones, Andrés Modet, and Jeannette M. Wing.

Gun89.. E. L. Gunter. Doing Algebra in Simple Type Theory. Technical Report MS-CIS-89-38, Dep. of Computer and Information Science, University of Pennsylvania, 1989.

Her64.. I. N. Herstein. *Topics in Algebra.* Xerox, 1964.

Hol. . The HOL System, Tutorial. Available on the Web as `http://lal.cs.byu.edu/lal/holdoc/tutorial.html`.

KW98.. F. Kammüller and M. Wenzel. Locales – a Sectioning Concept for Isabelle. Technical Report 449, University of Cambridge, Computer Laboratory, 1998.

OSR93.. S. Owre, N. Shankar, and J. M. Rushby. The PVS Specification Language (Beta Release). Technical report, SRI International, 1993.

Pau90.. L. C. Paulson. Isabelle: The Next 700 Theorem Provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–386. Academic Press, 1990.

Pau94.. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS.* Springer, 1994.

Pau95.. L. C. Paulson. First Isabelle User's Workshop. Technical Report 379, Computer Laboratory, University of Cambridge, September 1995.

PG96.. L. C. Paulson and K. Grabczewski. Mechanizing Set Theory. *Journal of Automated Reasoning*, 17:291–323, 1996.

Wie59.. H. Wielandt. Ein Beweis für die Existenz der Sylowgruppen. *Archiv der Mathematik*, 10:401–402, 1959.

Yu90. . Y. Yu. Computer Proofs in Group Theory. *Journal of Automated Reasoning*, 6:251–286, 1990.