The Rooster and the Butterflies

Assia Mahboubi

Microsoft Research - Inria Joint Centre

Abstract. This paper describes a machine-checked proof of the Jordan-Hölder theorem for finite groups. This purpose of this description is to discuss the representation of the elementary concepts of finite group theory inside type theory. The design choices underlying these representations were crucial to the successful formalization of a complete proof of the Odd Order Theorem with the Coq system.

1 Introduction

The Odd Order Theorem due to Feit and Thompson [7] is a major result of finite group theory which is a cornerstone of the classification of finite simple groups. Originally published in 1963, this was considered at its time as a demonstration of an uncommon length and intricacy, whose 255 pages filled an entire volume of the Pacific Journal of Mathematics. Later simplified and improved by a collective revision effort [3, 20], it remains a long and difficult proof, combining a broad panel of algebraic theories. In September 2012, the Mathematical Components team, lead by Georges Gonthier, completed [10] a formalization of this result using the Coq system [1, 4].

This achievement is evidence of the maturity of *proof assistants*, a family of software systems pioneered by N. G. de Bruijn's AUTOMATH system [6], that aims at "doing mathematics with a computer". The ambition of proof assistant is to realize an old dream: automate the verification of mathematical proofs. Check a theorem with a proof assistant consists in providing a description of the statement and of its candidate proof in formal logic and then having a generic and relatively small program checking the well-formedness of this proof with respect to the elementary rules of logic. A proof assistant provides the support necessary to obtain both a high confidence in proof checking and the mandatory set of tools required to ease the process of describing statements and proofs.

For the last decade, proof assistants have been successfully employed in a variety of contexts, from hardware and software verification to combinatorics or number theory. However the distinguishing feature of the complete formal proof of the Odd Order Theorem is that the corresponding libraries of formalized mathematics cover a range of algebraic theories that is both wide and deep. The proof of the Odd Order Theorem actually relies on a number advanced results that necessitate non-trivial combinations of arguments arising from several areas of mathematics.

When assisting the user in his verification task, the proof assistant is not expected to invent new results or new justifications. Yet a substantial part of the effort required by such a large scale endeavor consists in reworking the mathematics described in the standard literature so that it can be organized in a satisfactory and modular manner. The software engineering effort leading to (re)usable and composable libraries of formalized mathematics hence also involves re-thinking the mathematical definitions and proof methods. The formalization of the basics has to accommodate the variety of its usage in more advanced parts of the theory.

In this paper, we outline how elementary concepts of finite group theory have been revisited in the low-level libraries of this formal proof of the Odd Order Theorem. We do not claim novelty here: part of the material exposed here has been already described at various levels of detail in other venues that we list in the preamble of each section. Previous publications were mostly written for readers familiar with the Coq system, and most of them deal with programming issues. By contrast, we have tried here to provide a mathematical documentation of a few Coq libraries[1], as distant as possible from Coq syntax, since this intuition might be difficult to grasp from the documentation headers of the corresponding files. These documentation headers can be browsed on-line at:

`http://ssr.msr-inria.inria.fr/~jenkins/current/index.html`

In the next sections, by *formalized*, we mean implemented in the Coq system. The words *formal* and *formally* refer to Coq syntax. The words *informal* and *informally* refer to the corresponding mathematical notations we use throughout the paper to improve the rendering. We however maintain a precise correspondence between formal syntax and informal notations. We also use the collective "we" pronoun to refer to the team of authors of these libraries [10].

The rest of the paper is organized as follows. Section 2 explains the representation adopted for finite types and finite sets. Section 3 is devoted to the definition of finite groups and their morphisms. Section 4 describes the formalization of the quotient operation of group theory. Finally, section 5 illustrates how this material is used in the proof of a standard result of finite group theory, the Jordan-Hölder theorem.

2 Preliminaries

In this section, we recall the formal definitions of the preliminary notions we rely on. This material has already been presented in earlier publications [12, 8, 10, 11], with emphasis on the techniques used for their definition in Coq. These lower layers of formalized mathematics are quite constrained by the features of the logic underlying the Coq system and, in particular, by its constructiveness. A significant effort is put in the formalization of the theory of objects that behave mostly the same in either a classical or a constructive setting. The purpose, and the challenge, of the corresponding libraries is to provide enough infrastructure for the user to safely ignore the choices adopted for the implementation of these lower-level definitions. When these patterns of reasoning are effective, using an

[1] `https://gforge.inria.fr/frs/?group_id=401`

excluded-middle argument or performing a choice operation should be as convenient on top of these axiom-free Coq libraries as in the setting of a classical logic, like the one assumed by most of the mathematical literature.

2.1 Types with Decidable Equality

The type theory implemented by the Coq proof assistant is a constructive framework: the excluded middle principle is not allowed for an arbitrary statement without postulating a global axiom. Reasoning by case analysis on the validity of a predicate is valid constructively when it is possible to implement a (total) boolean function which decides this validity: the type of boolean values reflects the class of statements on which the excluded middle principle holds constructively. For instance, we do not need any axiom in order to reason by case analysis on the equality of two arbitrary natural numbers because equality on natural numbers is decidable. A decidable predicate is a predicate that can be (and in our libraries, that is) formalized as a function with boolean values.

The prelude libraries of the system, automatically imported when a Coq session is started, define an an equality predicate parametrized by an arbitrary type T, which is the smallest binary reflexive relation on T. This equality is often referred to as *Leibniz equality*. However, not all types are a priori equipped with an associated total and boolean comparison function, testing the validity of a Leibniz equality. However, the vast majority of the data we manipulate can be modeled with types equipped with such an operator. This operator legitimates constructive reasoning and programming by case analysis on the equality of two objects of such a type and witnesses the decidability of the associated Leibniz equality predicate. The library hence defines a structure for *types with a decidable equality*, formally called `eqType`. This structure packages a type with a binary boolean function on this type, plus a proof that the boolean test faithfully models Leibniz equality. For instance, finite types, natural numbers, rational numbers, and real or complex algebraic numbers are instances of this structure. Moreover, pairs, sequences or subtypes of instances of this structure are also canonically types with a decidable equality.

Another important feature of types with a decidable equality is the fact that they enjoy the property of *uniqueness of identity proofs* [14]. This plays a crucial role in our formalization but is out of the scope of the present paper. The interested reader can refer to previous publications [12, 11, 8, 10] for more information on the formalization of this structure.

In all that follows, and unless explicitly stated, by *type* we always implicitly mean *type with a decidable equality*. Hence the reader can safely forget about the constructiveness issues mentioned in this section: case analysis on the equality of two objects is allowed as well as on the membership of an object to a sequence, etc.

Libraries. The corresponding file to this subsection is `eqtype.v`.

2.2 Finite Types, Finite Sets

The library also defines an interface for *finite types*, which are types with a finite number of elements. Formally this structure is called `finType`. It packages a sequence enumerating exhaustively the elements of the type[2] and a proof that this sequence is duplicate-free. Finite types are a instance of a more general interface for types equipped with a choice operator: for an arbitrary non-empty decidable predicate, the choice operator outputs a canonical witness. In the case of a finite type, the choice operator just inspects the enumeration and picks the first witness encountered.

This representation of finite types is especially convenient to define functions with a finite domain. Our motivation here is to craft a datastructure for functions so that they provably verify the so-called extensionality principle:

$$\forall x, f x = g x \quad \Leftrightarrow \quad f = g$$

which states that the point-wise equality of two functions f and g on their whole domain type is equivalent to the Leibniz equality of these functions. Again, in Coq's type theory, this principle is not valid in general: two programs that output the same values on the same inputs are not necessarily identified by the Leibniz equality predicate. By contrast, we would like for instance to work with a definition of sets which allows us to equate sets that have point-wise characteristic functions.

Let us consider a finite type F, with e the enumeration of its elements. Let $|e|$ be the length of e. We represent a total function f with arguments in F and values in a type T by a finite sequence Im_f of length $|e|$, of elements in type T. Hence the value of f at e_i, the element at position $i < |e|$ in the enumeration e, is the i-th element of the sequence Im_f. We call such a function a *finite function*. This representation validates the extensionality principle: the right-to-left implication is trivial and the left-to-right implication holds because according to our definition of a finite function the Leibniz equality of two finite functions really *is* the Leibniz equality of their respective finite graphs Im_f and Im_g. This equality is granted by the hypothesis of point-wise equality. Note that we do not need to assume any finiteness property on the codomain type. If `(aT : finType )` is a finite type and `(rT : Type)` an arbitrary type, the type of finite functions with (finite) domain `aT` and codomain `rT` is formally denoted by `{ffun aT >-> rT}`. Most of the theory of finite functions however assumes that the codomain type is an instance of type with decidable equality.

Finite functions with boolean values represent characteristic functions of sets of elements of their domain type. In other words, a finite set over a finite type F is coded by a sequence of boolean values which is a mask on the enumeration of F: *true* values select the elements that belong to the set. Now two finite sets with point-wise equal characteristic functions are (Leibniz) equal by the previous extensionality principle.

[2] These points are objects of a previously known type with decidable equality.

For any finite type F, `{set F}` formally denotes the type of finite sets of elements of type `F`. Remark that type `{set F}` has itself a finite number of elements and is hence an instance of finite type. We can therefore form the type `{set {set F}}`, which is the powerset associated to the finite type `F`. The library on finite sets provides definitions and notations for the standard concepts related to sets. For instance `x \in A` denotes the (decidable) test of membership of the element `x` in the set `A`, informally denoted by $x \in A$. Similarly, `A \subset B` denotes formally the (decidable) test of inclusion of the set `A` in the set `B`, which tests whether the true values of the mask defining `A` are also true values in the mask defining `B`. Informally we denote this test by $A \subset B$. The expression `A :&: B` (resp. `A :|: B`) denotes the intersection (resp. union) of two sets over the same finite carrier. The corresponding informal notation is $A \cap B$ (resp. $A \cup B$). The expression `#|A|` denotes the cardinal of a finite set, which is the number of *true* values in the mask. The notation `f @: A` is used for the image set of `A` by the function `f` from a finite type to an other finite type [3], which we denote informally by $f(A)$. The notation `f @^-1: A` is used for the preimage set of `A` by the function `f` from a finite type to an arbitrary type, which we denote informally by $f^{-1}(A)$. We will also use in section 3.1 the possibility of defining a set by comprehension: the expression `[set x | P x]` formally denotes the set of elements satisfying the (decidable) property `P`, and we denote this set informally by $\{x \quad | \quad P(x)\}$.

In all what follows, by *set* we mean a finite set of elements in a finite type. The reader can safely forget about the implementation described in the present section to apprehend the rest of this paper and rely on his or her classical intuition of sets.

Libraries. The corresponding files to this subsection are `choice.v`, `fintype.v`, `finfun.v` and `finset.v`.

3 Elementary Notions of Finite Group Theory

In this section we describe the datastructures adopted in the libraries about the elementary concepts of finite group theory. The design choices evolved in time and are now different from their earliest published description [12]. Garillot's PhD thesis [8] provides a more recent and accurate account of these choices, targeted at an audience expert in proof assistants.

The datastructures representing formally the operations defining finite groups of interest and the operations combining finite groups are shaped by two important remarks. First, we model groups as certain subsets of an ambient, larger group, which fixes the data all its subgroups share: the type of the elements, the group operation, the identity element. Hence groups are not types but objects, namely some sets of a finite type. This choice is motivated by the observation that finite group theory is not about the properties of the elements of a given group, but mostly about the study of how finite *subgroups* (of a larger finite

[3] Since we define an image *set* we need the codomain type to be also a finite type.

group) can combine. The second remark is that it is possible to revisit the standard definitions of the literature, so that they apply to arbitrary subsets of an ambient group, and not only to the special subsets that are also groups. The motivation for this generalization is to make the related constructions total and the statements of the related results less constrained and hence more usable.

3.1 Finite Groups

We reproduce below excerpts borrowed from the preliminary results of Aschbacher's book [2].

Definition 1 (Group, subgroup). *A* group *is a set G together with an associative binary operation which possesses an identity and such that each element of G possesses an inverse. In the remainder of this section G is a group written multiplicatively. (...) A* subgroup *of G is a nonempty subset H of G such that for each $x, y \in H$, xy and x^{-1} are in H. This insures that the binary operation on G restricts to a binary operation on H which makes H into a group with the same identity as G and the same inverse.*

Definition 2 (Product). *For $X, Y \subseteq G$ define $XY = \{xy;\ x \in X, y \in Y\}$. The set XY is the product of X with Y.*

In definition 2, we can observe that the group G is only here to fix the group operation and identity shared by the two sets X and Y and is not otherwise part of the definition. Moreover, these standard definitions and notational conventions are an instance of the standard practice which consists in using product notations both for points and sets: a similar convention apply for the inverse $X^{-1} = \{x^{-1};\ x \in X\}$ of a set $X \leq G$ and the constant 1 denotes both the identity of the group and the singleton $\{1\}$.

The library for elementary finite group theory defines two main structures. A first structure packages a finite type with a monoid operation and an involutive antimorphism. This structure is formally called `baseFinGroupType`[4] and all its instances share three common notations: the infix notation `*` denotes the monoid operation, the postfix `^-1` notation denotes the involution and `1` denotes the neutral element. A second structure enriches the previous one to obtain all of the group axioms, hence describes what we call *group types* in the sequel. This second structure is formally called `finGroupType` and its instances inherit from the notations for the group operation, for the inverse and for the identity.

Let $\mathcal{G}$ be a group type. Both $\mathcal{G}$ and the type of sets of $\mathcal{G}$ are instances of the `baseFinGroupType` structure. For $\mathcal{G}$, this holds by construction of a group type. For the type of sets of $\mathcal{G}$, this comes from the properties of set product and set inverse. We can therefore utilize the notations `*`, `^-1` and `1` for both point-wise and set-wise operations of $\mathcal{G}$. Informally, we use a multiplicative convention and denote by xy the product of the element x by the element y of a group type.

[4] We do not use an informal name for this concept which we use only once in the rest of this text.

Similarly, we denote by AB the set product of two sets A and B of a group type. In order to avoid useless rigid type constraints in the formalized statements of finite group theory, we generalize as much as possible the standard concepts of finite group theory to sets of group type. In all that follows a *mere set*, sometimes even abbreviated in a *set*, refers to an arbitrary set of a group type $\mathcal{G}$. Elements of a mere set A can be multiplied by the group operation defined by $\mathcal{G}$, although this product does not necessarily belong to A, nor the identity of $\mathcal{G}$.

For instance if A is an *arbitrary mere set* of a group type $\mathcal{G}$ and x an arbitrary element of $\mathcal{G}$, we define the *conjugate* of A by x as the set of conjugates $x^{-1}ax$ of elements $a \in A$ by x:

$$A^x := \{x^{-1}ax \quad | \quad a \in A\}$$

Note that we use here the group operation of $\mathcal{G}$ to describe the elements of A^x. Formally this set is defined as the image set of the set A by the function $y \mapsto x^{-1}yx$. Similarly, we define A^B, the conjugate of the set A by the set B as the image of the set B by the function $x \mapsto A^x$. The *normalizer of a mere set* A is defined as:

$$N(A) := \{x \quad | \quad A^x \subseteq A\}$$

The definition of $N(A)$ is formalized using the comprehension-style construction mentioned in section 2.2. The *centralizer of a mere set* A is the intersection of the normalizers of all the singleton sets of its elements:

$$C(A) := \bigcap_{x \in A} N(\{x\})$$

The formal definition of $C(A)$ uses a library about iterated operators [5], which provides a modular infrastructure for notations, theory and computation of indexed constructions like $\bigcap_{x\in A}$. We also introduce a notation for the localization of the normalizer and centralizer of a set A to a set B: we denote informally by $N_B(A)$ (resp. $C_B(A)$) the intersection $N(A) \cap B$ (resp. $C(A) \cap B$).

Finally, for any group type $\mathcal{G}$, a *group* of $\mathcal{G}$ (or just a *group*) is a mere set G of $\mathcal{G}$ which contains the identity element ($1 \in G$) and is closed under the product ($GG \subseteq G$). Note that what we call a group here is necessarily a finite group. The singleton set $\{1\}$ of the identity element of $\mathcal{G}$ is a group of $\mathcal{G}$ as well as the total set containing all the elements of $\mathcal{G}$. This total set actually plays the role of the ambient group G postulated in many statements, like for instance in definition 2. A subgroup H of a group G is a group whose underlying set is a subset of the one underlying G. Formally, `{group gT}` denotes the type of groups of a group type `gT`. For any set A, the group *generated* by A is obtained as the intersection of all the sets that are groups and contain A, formalized by the means of aforementioned library about iterated operators [5]. As usual, the group generated by A is formally defined as a set, later equipped with a canonical structure of group. We denote informally by $\langle A\rangle$ and formally by `<<A>>` the group generated by a mere set A. We also prove that some of the above constructions on mere sets preserve the property of being a group: the intersection of two groups is a group, the normalizer of a group is a group.... Note that the set-level product of two groups is not necessarily a group.

Lemma 1 (Product group [2]). *Let X, Y be subgroups of a group G. Then XY is a subgroup of G if and only if $XY = YX$.*

In our formal library, we model this fact by defining an alternate product operation on sets of a group type that always produces a group: the join product of two sets A and B is simply the group generated by A and B, which coincides with AB when $AB = BA$.

As a conclusion of this subsection, let us summarize two differences in flavor between the usual paper versions of statements and definitions in finite group theory and their formal versions. First, the standard constructions of new groups from known groups like normalizer, centralizer, etc. are defined as the construction of new sets from known sets. The resulting sets are later equipped with a group structure under the suitable assumption on their components. Second, every formal statement features one more universal quantification or parameter, for the parameter group type. For instance, the original statement of the Odd Order Theorem is the following:

Theorem 1 (Odd Order theorem [7]). *Every finite group of odd order is solvable.*

And its formal statement in Coq is:

```
Theorem Feit_Thompson : forall (gT : finGroupType),
forall (G : {group gT}), odd #|G| -> solvable G.
```

This formal version is not less general than the original one: it can be read as "every subgroup of odd order of a group is solvable", where the group and the subgroup can for instance be the same.

Libraries. The corresponding file to this subsection is `fingroup.v`.

3.2 Group Morphisms, Isomorphisms

We again quote Aschbacher's definition [2] of the homomorphisms associated with the structure of group:

Definition 3 (Group homomorphism). *A* group homomorphism *from a group G into a group H is a function $\alpha : G \to H$ of the set G into the set H which preserves the group operations: that is for all x, y in G, $(xy)\alpha = x\alpha\ y\alpha$. Notice that I usually write my maps on the right, particularly those that are homomorphisms. The homomorphism α is an* isomorphism *is a bijection. (..) H is said to be a* homomorphic image *of G if there is a surjective homomorphism of G onto H.*

In all that follows, we slightly depart from Aschbacher's choices: we use the words *group morphism* instead of *group homomorphism* and we write these maps applicatively ($\alpha(x)$) instead of on the right ($x\alpha$). Definition 3 describes a group morphism as a function whose domain is a specified group. In type theory, a function is defined as an object `(f : A -> B)` whose type `A -> B` specifies the domain type `A` of its arguments and the codomain type `B` of its values. Such a function `f` is necessarily total on its domain type `A`. However we argued in section 3.1 that groups are better represented not as types but as objects, namely as sets of an ambient group type. Hence in our formalization, two groups G and H are modeled as sets of two (group) types $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively and definition 3 only specifies a morphism $\phi : G \to H$ as a function and its morphism properties for certain $\mathcal{G}_1$, the ones in G. We are hence left to choosing one of the several standard ways of dealing with partiality issues in type theory: assigning a clever default value outside the domain, restricting the type of the domain, using a monadic style....

Several approaches have been successively considered for the formal definition of group morphisms, leading to different versions of the related libraries. We eventually reverted the choice described in our earliest publication [12]. Garillot [8] has discussed the motivations for the change to the datastructure we describe hereafter. The current structure of group morphism, formally denoted by `{morphism D >-> rT}`, has three parameters:

- a group type `aT` called the *domain type*;
- a group type `rT` called the *codomain type*;
- a mere set `D` of the group type `aT` called the *domain.*

The domain type is not displayed to the user in the type `{morphism D >-> rT}` because it is implicit: it can be inferred from the type of the parameter `D`. This interface describes functions of type `aT -> rT` which distribute over the product of two elements if they both belong to the set `D`.

Since this definition ensures the distributive property only on the domain of a morphism, it becomes natural to consider alternative definitions of images and preimages for group morphisms. More precisely, consider f a group morphism and denote D the domain of f. Let A be a set of the domain type of f. We define the *morphic image* of the set A by the group morphism f as the image by f of the intersection of A with the domain D:

$$f^*(A) := f(A \cap D)$$

where $f(A \cap D)$ refers to the image set by f (see section 2.2). Formally, this morphic image is denoted by `f @* A`. Similarly, we define the *morphic preimage* of the set R by the group morphism f as the intersection of the domain D with the preimage by f of R:

$$f^{-1*}(R) := f^{-1}(R) \cap D$$

where $f^{-1}(R)$ is the preimage set of R by f. In Coq, this morphic preimage is denoted by `f @*^-1 R`. Note that the image and the morphic image of the domain D coincide. When both the domain D of f and the set A (resp. the set

R) are groups, the morphic image (resp. the morphic preimage) of A (resp. of R) is a group. For instance, the morphic preimage of the singleton set $\{1\}$ of the identity is a group, called the *kernel* of the group morphism. Informally we denote by *Ker* f the kernel of a group morphism and by *Im* f its image. We denote by Ker_A f the intersection of the kernel of a morphism with a set A.

As a consequence of this formalization choice, each new definition of a morphism should come with the explicit mention of the domain it is a morphism on. However, we use the facilities offered by Coq's type inference mechanism to compute automatically a non-trivial domain for morphisms resulting from standard operations. For instance if f and g are two group morphisms, under the obvious compatibility condition on their domain and codomain types, Coq can infer that the composition $g \circ f$ is a morphism with domain (at least) $f^{-1*}(H)$ where H is the domain of the morphism g. This means that if we dispose of `(f : {morphism G >-> hT})` and `(g : {morphism H >-> rT})` two morphisms, Coq infers the type `(g \o f : {morphism f @*^-1 H >-> rT})` automatically. Similarly, the inverse of an injective morphism f with domain D is canonically a morphism with domain $f(D)$. However, it might sometimes be necessary to restrict by hand the domain of a morphism. We hence provide an operator which constructs a new morphism g with domain the set A from a known morphism f and a proof that $A \subseteq D$. Let us mention a last example of useful operation on group morphisms. Let f_1 and f_2 be two group morphisms with possibly different codomain types but with the same domain type. Let D be the domain of f_1 and G be the domain of f_2. We moreover assume that G is a group. Under this assumption we can construct the natural *factor morphism* mapping $f_1^*(G)$ to $f_2^*(G)$ provided that G is included in D and that the kernel of f_1 is included in the kernel of f_2. The kernel of this morphism is the morphic image $f_1^*(\text{Ker } f_2)$ of the kernel of f_2 by f_1.

When there exists a group morphism such that a set B is the morphic image of the set A by this morphism, we say that B is the *homomorphic image* of A. In Coq, this is denoted by `(B \homg A)`. We say that a group morphism f with domain D maps a set A *isomophically* to the set B when both A is included in D and the image of the complement of A (in the domain type) by f is *equal* to the complement of B (in the codomain type). This seemingly contrived definition is a concise way of ensuring both that the morphism is injective and that B is the image of A. In Coq, we denote by `(B \isog A)` the existence of a group morphism which maps A to B isomophically. Note that once again we have posed these definitions at the level of sets of the group type.

Let us conclude this subsection by mentioning that we adopt a completely different datastructure for the set $Aut(G)$ of automorphisms of a group G, which is the set of endomorphisms of G that are also isomorphisms. The set of automorphisms of a group G of a group type $\mathcal{G}$ is defined as the set of permutations of $\mathcal{G}$ that are the identity function outside G and distribute over the group product inside G. It is easy to interpret canonically such a permutation as a group morphism but defining automorphisms as permutations actually allows us to transpose the (group) theory already developed for the permutations of a set to the $Aut(G)$ group.

Libraries. The corresponding files to this subsection are `morphism.v` and `automorphism.v`

4 Cosets, Normal Subgroups and Quotients

In this section again we quote Aschbacher's definitions [2].

Definition 4 (Normal subgroup). *A subgroup H of G is* normal *if $g^{-1}hg \in H$ for each $g \in G$ and $h \in H$. Write $H \trianglelefteq G$ to indicate H is a normal subgroup.*

Definition 5 (Cosets, coset space). *Let H be a subgroup of G. For $x \in G$ write $Hx = \{hx : h \in H\}$ and $xH = \{xh : h \in H\}$. Hx and xH are cosets of H in G. Hx is a* right coset *and xH is a* left coset*. To be consistent I'll work with right cosets Hx in this section. G/H denotes the set of all (right) cosets of H in G. G/H is the* coset space *of H in G.*

Both the definition of the "normal" predicate and the one of right and left cosets are literally formalized in our formal development, except that they are defined as usual for mere sets instead of groups. However we depart from Aschbacher's choice when it comes to the definition of coset spaces. Instead, we somehow take backward the following definition [2]:

Definition 6 (Factor group). *If $H \trianglelefteq G$ the coset space G/H is made into a group by defining multiplication via*

$$(Hx)(Hy) = Hxy \quad x, y \in G$$

Moreover there is a natural surjective homomorphism $\pi : G \to G/H$ defined by $\pi : x \mapsto Hx$. Notice $ker(\pi) = H$. Conversely if $\alpha : G \to L$ is a surjective homomorphism with $ker(\alpha) = H$ then the map $\beta : Hx \mapsto x\alpha$ is an isomorphism of G/H with L such that $\pi\beta = \alpha$. G/H is called the factor group *of G by H. Therefore the factor groups of G over its various normal subgroups are, up to isomorphism, precisely the homomorphic images to G.*

Instead of studying the entire coset space in the sense of definition 5, we formally define the coset space of a mere set A as the type whose elements are the right cosets Ax, with x spanning the normalizer $N(A)$ of A (see section 3.1). Informally, we denote by $\mathcal{C}_A$ the coset space of a set A according to this alternate definition. An element of $\mathcal{C}_A$ is a *bilateral coset*, since for $x \in N(A)$, we have $Ax = xA$. Note that when H is a group, $N(H)$ happens to be the largest group (for inclusion) in which H is normal.

Coset spaces as defined in 5 are actually of little use when H is not normal in G, assumption which make the coset space a group. Hence the theory of coset spaces described in the literature actually boils down to the theory of quotients of a group by one of its normal subgroup. In our setting, we are able to define a better theory of coset spaces which allows us to simplify greatly the manipulation of quotients by erasing these normality assumptions. Without this effort, the requirements put in the type constraints are soon too demanding for the lemmas to be usable as such. We explain this formalization in the rest of this subsection and illustrate its impact on the three isomorphism theorems of group theory.

Consider a group H of a group type $\mathcal{G}$. Its coset space $\mathcal{C}_H$ is an instance of group type: the group product operation and the inverse operation are respectively the set product and the set inverse operations, and the identity element $1 \in \mathcal{C}_H$ is $1H = H$. In Coq, this new instance $\mathcal{C}_H$ of group type is named `(coset_groupType H)`. For $x, y \in N(H)$, the product $xHyH$ of two bilateral cosets is the coset xyH of the product $xy \in N(H)$. But interestingly, if we extend this correspondence by associating each element $x \notin N(H)$ with the identity value $1 \in \mathcal{C}_H$, we obtain a total function on $\mathcal{G}$, which is an instance of group morphism, in the sense discussed in section 3.2. The domain type of this morphism is $\mathcal{G}$, its codomain type is $\mathcal{C}_H$ and its domain is the normalizer $N(H)$. Informally, we denote this morphism by $./H$. Formally, we denote this morphism by `(coset H)`. The quotient of a mere set A by the group H, denoted A/H, is defined as the *morphic image* of the set A by this morphism $./H$.

When H is a normal subgroup of A, our definition coincide with the one of the standard literature. It is however more general for it provides a precise meaning to the quotient of a mere set A by H which requires A neither to be a group nor to be included in $N(H)$. Using the standard definition 6, what we define as A/H would be indeed described as $N_A(H)H/H$. In fact we make the definition of quotient even more general: if A and B are sets of $\mathcal{G}$, then A/B denotes the quotient of the set A by the group $\langle B\rangle$ generated by B: A/B is a set of the group type $\mathcal{C}_{\langle B\rangle}$.

We conclude this subsection by commenting three elementary but crucial results of finite group theory, called isomorphism theorems [18]. The first one is a rephrasing of the result contained in Aschbacher's definition 6.

Theorem 2 (First isomorphism theorem). *Let f be a group morphism from G to K. Then*

$$G/(Ker\ f) \to H \quad \textit{with} \quad (Ker\ f)x \mapsto xf$$

is an injective group morphism. In particular

$$G/(Ker\ f) \textit{ is isomorphic to } (Im\ f).$$

In our setting, starting from a group morphism f with domain a group G, we construct a new morphism g with domain $G/(Ker\ f)$ which is injective and such that for any set A, the morphic image $f^*(A)$ of A by f is equal to $g^*(A/(Ker\ f))$. This is a slight generalization of the above statement, which corresponds to the case where we take $A = G$. This morphism g can be obtained as the factor morphism mentioned in section 3.2, taking f as f_2 and $./(Ker\ f)$ as f_1: the kernel of $./(Ker\ f)$ is $(Ker\ f)$, hence included in (in fact equal to) the one of f. Moreover the domain of a group morphism is included in the normalizer of its kernel, thus the domain of f is included in the one of $./(Ker\ f)$. We can therefore form this factor morphism g, which sends $A/(\mathrm{Ker}\ f)$ to $f^*(A)$ for any set A. The kernel of this morphism is $(Ker\ f)/(Ker\ f)$ which is trivial, hence the morphism is injective.

An easy corollary follows from this first isomorphism theorem. For an extra subgroup H of G, we can construct a morphism g with domain $H/(Ker_H\ f)$, which is injective and such that for any subset A of H, the morphic image $f^*(A)$ of A by f is equal to $g^*(A/(Ker_H\ f))$.

The second isomorphism theorem is a central ingredient in the butterfly argument of the proof of Jordan-Hölder theorem (see section 5).

Theorem 3 (Second isomorphism theorem). *Let G be a group and H and K two subgroups of G such that $H \subset N(K)$. Then HK is a subgroup of G and thus K is a normal subgroup of HK and:*

$$\phi : H \to HK/K \quad \textit{with} \quad u \mapsto uK$$

is an injective group morphism with $Ker\ \phi = H \cap K$ and

$$H/H \cap K \textit{ is isomorphic to } HK/K.$$

In this theorem, we can observe an instance of the partiality issues raised by the standard definition of quotients in the literature. The statement of the theorem has two parts: the first one establishes the conditions under which some objects are well defined. The second one is an isomorphism involving these objects. The situation is quite easier using the generalized definitions set up in the previous subsections.

In our setting, an ambient group type $\mathcal{G}$ plays the role of the above G, and we consider two groups H and K of $\mathcal{G}$, such that $H \subset N(K)$. We prove the second isomorphism theorem by constructing a morphism g with domain $H/H \cap K$, which is injective and such that $g^*(A/H \cap K) = A/K$ for any subset A of H. Noticing that $H \cap K$ is $Ker_H\ (./K)$, the existence of g is a direct application of the above corollary of the first isomorphism theorem, applied to the morphism $./K$. The usual version of the second isomorphism theorem, as stated in theorem 3, follows from this construction: in our setting HK/K, which is the morphic image of HK by $./K$, is *equal* to the morphic image H/K of H by $./K$.

The conclusion of the third isomorphism theorem uses three distinct quotient operations, each of which deserves a side condition of well-formedness.

Theorem 4 (Third isomorphism theorem). *Let H and K be normal subgroups of G such that H is a subgroup of K. Then*

$$\phi : G/H \to G/K \quad \textit{with} \quad Hx \mapsto Kx$$

is an injective morphism with $Ker\ \phi = K/H$, and

$$(G/H)/(K/H) \textit{ is isomorphic to } G/K.$$

We consider three groups G, H, and K of a group type $\mathcal{G}$. We suppose that H and K are normal subgroups of G and that H is a subgroup of K. Again, we slightly generalize the above statement by constructing a morphism g with domain $(G/H)/(K/H)$ such that for any subset A of G, $g^*(A/H/(K/H)) = A/K$. We proceed in three steps. First, we consider the morphism $./K$, whose domain is $N(K)$. We restrict it to G, using the restriction operation mentioned in section 3.2. This is possible since $G \subset N(K)$ by hypothesis. Then, we factor the morphism $./H$ by this restriction. This means we apply the factor operation described in section 3.2 with f_1 equal to the morphism $./H$ and f_2 equal to the restriction of $./K$ to G. We hence obtain a morphism g' which maps any subset A of G/H to A/K and has kernel K/H. Finally, we apply the corollary of the first isomorphism theorem to g', which constructs the announced morphism g.

Libraries. The corresponding file to this section is `quotient.v`.

5 The Jordan Hölder Theorem(s)

In this section, we sketch the well-known proof of the Jordan-Hölder theorem for finite groups [17, 15] as formalized in Coq on top of the infrastructure presented in section 3.

5.1 Simple Groups, Composition Series

A normal series is a sequence of successive quotients of a group.

Definition 7 (Normal series, factors). *A* normal series *for a group G is a sequence* $1 = G_0 \trianglelefteq G_1 \ldots \trianglelefteq G_n = G$*, and the successive quotients* $(G_{k+1}/G_k)_{0 \leq k < n}$ *are called the* factors *of the series.*

Formalizing normal series poses no particular problem: it is a sequence of groups where the sets underlying two consecutive elements are related by the relation $\trianglelefteq$. The corresponding formal definition is actually obtained from a more general pattern, that we call subgroup series. Subgroup series are defined as sequences of groups for which the sets of two consecutive elements related by a binary relation (on sets). This simple definition suits the formalization of several notions like normal series, ascending series, descending series, chief series....

A formal definition of the sequence of factors of a normal series is however slightly more uneasy at first sight. Let G be a group of the group type $\mathcal{G}$, and $(G_k)_{0 \leq k \leq n}$ a normal series for G. All the elements of the series are groups of $\mathcal{G}$. By contrast, each factor G_{k+1}/G_k is a group of the group type $\mathcal{C}_{G_k}$. Since the elements of the sequence of factors have pairwise distinct types a formal definition of this sequence would be very intricate. Instead, we represent a factor (G_{k+1}/G_k) of a normal series by a pair of groups (G'_{k+1}, G_k) where (G'_{k+1}/G_k) is a canonical representative of the isomorphism class of (G_{k+1}/G_k) inside $\mathcal{C}_{G_k}$. The sequence of factors can hence be represented as a homogeneous sequence, whose elements are pairs of group of $\mathcal{G}$. The use of the isomorphism representative is motivated by the proof described in section 5.2. The formalization of this definition uses the choice operator mentioned in section 2.2.

Definition 8 (Simple group). *A group G is* simple *when its only proper normal subgroup is the trivial group* 1.

Definition 9 (Composition series). *A normal series whose factors are all simple groups is called a* composition series.

Definitions 8 and 9 are translated literally in the libraries, using the material presented so far. Simple groups are exactly groups with composition series of length 1 (containing only the group itself and 1). Similarly trivial groups are are exactly groups with empty composition series (containing only the group itself).

Lemma 2 (Existence of a composition series). *Every finite group has a composition series.*

The proof of lemma 2 is an induction on the cardinal of a group G, which is either simple, or trivial, or has a non-trivial proper normal subgroup H, maximal for inclusion. In the last case, the quotient G/H is simple and we conclude by applying the induction hypothesis to H.

Libraries. The corresponding files to this subsection are `gseries.v` and `jordanholder.v`.

5.2 Uniqueness of Composition Series

The Jordan-Hölder theorem states that the (simple) factors of a composition series play a role analogous to the prime factors of a number. They however do not control completely the structure of a group: unlike natural numbers non-isomorphic groups may have composition series with isomorphic factors.

Theorem 5 (Jordan-Hölder Uniqueness). *Two composition series of a same group have the same length and the same factors up to permutation and isomorphism.*

Let G be a group of a group type $\mathcal{G}$. We prove that for any two composition series of G, the corresponding sequences of factors are equal up to permutation, since we have already picked canonical isomorphism representatives for the factors. Again we proceed by induction on the cardinal of the group G. In the inductive case, we can assume that G is neither trivial nor simple, and we consider two non empty composition series of G, $(N_i)_{0\leq i\leq r+1}$ and $(M_j)_{0\leq j\leq s+1}$. Note that $G = N_{r+1} = M_{s+1}$. We call N (resp. M) the group N_r (resp. M_s): $(N_i)_{0\leq i\leq r}$ is a composition series of N and $(M_j)_{0\leq j\leq s}$ is a composition series of M. Both N and M are normal subgroups of G. If N and M are equal, then the theorem is proved from the induction hypothesis. Otherwise we pose $I = M \cap N$, which is normal in both N and M. Now comes the crux of the demonstration: G/N is isomorphic to M/I and G/M is isomorphic to N/I. This step is called the butterfly lemma, or also Zassenhaus lemma [23] and both these isomorphisms are easy consequences of the second isomorphism theorem 3.

To finish the proof, we use lemma 2 to construct a composition series $(I_k)_{0\leq k\leq t}$ for I. The butterfly lemma ensures that the quotient N/I is simple. Therefore $(I_k)_{0\leq k\leq t}$ extends to a composition series for N by taking $I_{t+1} := N$. We dispose of two composition series $(I_k)_{0\leq k\leq t+1}$ and $(N_i)_{0\leq i\leq r}$ for the group N whose cardinal is smaller that the one of G: the induction hypothesis applies and these series have the same length and the same factors. Similarly we apply the induction hypothesis to the two composition series we dispose of for M. Hence up to isomorphism the set of factors associated with $(N_i)_{0\leq i\leq r+1}$ is G/N, N/I and a set F_N of other factors, the set of factors associated with $(M_i)_{0\leq i\leq r+1}$ is G/M, M/I and a set F_M of other factors, such that F_N and F_M are the same up to isomorphism and permutation. The isomorphisms established by the butterfly lemma conclude the proof.

Libraries. The corresponding file to the subsection is `jordanholder.v`.

5.3 More Butterflies

The more general version of the Jordan-Hölder theorem for finite groups deals with a more general kind of composition series: given a set A which acts on a group G, an A-composition series is an increasing sequence $(G_k)_{0\leq k\leq n}$ of subgroups of G, with $G = G_n$ and such that for each k, G_k is a maximal subgroup of G_{k+1} invariant by the action of A. A finite group G has an A-composition series as soon as A acts on G and the uniqueness theorem transposes to the factors of A-composition series of a same group, with a little more work, in particular for establishing the butterfly lemma. We have also formalized this more general version [2], which we do not detail here by lack of space.

The library also features the analogue Jordan-Hölder theorem for the theory of representations of finite groups [16], whose proof is again analogue in shape. However the algebraic structures at stake in that case are much more sophisticated than the ones of finite groups, and their formalization is based on a significant reworking of the standard mathematical presentations of elementary linear algebra [9].

As a final remark we would like to mention that these butterfly lemmas are quite typical, although rather simple, examples where two objects play a symmetrical role, which is broken *without loss of generality* at the beginning of the proof. The version of the Jordan-Hölder theorem we detailed in section 5.2 is so simple that no additional support is really needed in that proof. However the code formalizing the two more advanced versions we mentioned above are using a specific feature of the proof shell [13] used to develop these libraries, called the `wlog` tactic. This command is a key ingredient in order to avoid extremely painful redundancy in the script describing these mathematical arguments based on symmetries. This quite elementary feature of the tactic language has actually been instrumental at several places of the libraries, including advanced group theory for the proof of the Odd Order Theorem [10].

Libraries. The corresponding files to this subsection are `jordanholder.v` and `mxrepresentation.v`. See for instance `PFsection9.v` for an instance of `wlog` tactic, on a tricky chain of circular inequalities with equality conditions.

6 Conclusion

The structure of the paper reflects in miniature the one of the whole set of libraries of the formal proof of the Odd Order Theorem. Libraries on elementary concepts, like types with decidable equality or finite sets, are tightly related to the type system underlying the Coq proof assistant. They provide an infrastructure which allows us to ignore the details of their implementation when it comes to formalizing finite groups as finite sets of a group type. Here again part of the basic libraries about finite groups, morphisms, and quotients are devoted to the infrastructure work which aims at providing the same flavor of mathematical notations and packaging as in the standard literature of finite group theory. As a result, there is not much left to say when it comes to describing the formalized proof of the Jordan-Hölder, and this was precisely the purpose of the upstream effort. The elementary examples from finite group theory presented here also illustrate the fact that textbook presentations of abstract algebra are not necessarily sufficient references in order to design the appropriate abstractions for formal libraries to scale. Future formalizations will show whether the techniques employed in the present libraries are general enough to apply to more mathematical structures. The design of these patterns will for sure be impacted by improvements in the implementation of proof assistants [19] but also possibly by evolutions of the type theory they implement [21].

The difference in purpose of the different layers of libraries affect their de Bruijn factor [22], a criterion measuring the difference in size between the code describing a formal proof and the code of the typeset description of a paper proof. Lower level libraries feature by far the highest de Bruijn factor because they describe a lot of material which addresses the implicit content of paper mathematics. This implicit content is not only about datastructures, but also about how to recompute the implicit content of notational conventions, or abuse thereof, without which a paper text appears as extremely pedantic and soon unreadable. By contrast, for advanced libraries like the ones corresponding to the final chapters of the proof of the Odd Order theorem, it is possible to obtain a one to one correspondence, and even sometimes a shorter formal proof, which illustrates the benefits of the re-factoring of the mathematics.

Libraries. An example of infrastructure file with a very large de Bruijn factor is `bigop.v`. By contrast, file `BGappendixC.v` has a very small de Bruijn factor (3 pages for 170 lines of script according to G. Gonthier, author of the script). In file `PFsection3.v`, the local definition of some appropriate boilerplate [10] significantly shortens a pedestrian computational proof.

Acknowledgments. The author wishes to thank Jacques Carette, William Farmer, Laurence Rideau and Enrico Tassi for their proofreading.

References

[1] The Coq Proof Assistant, http://coq.inria.fr
[2] Aschbacher, M.: Finite Group Theory. Cambridge Studies in Advanced Mathematics. Cambridge University Press (2000)
[3] Bender, H., Glauberman, G.: Local analysis for the Odd Order Theorem. London Mathematical Society, LNS, vol. 188. Cambridge University Press (1994)
[4] Bertot, Y., Castéran, P.: Interactive theorem proving and program development: Coq'Art: The calculus of inductive constructions. Springer, Berlin (2004)
[5] Bertot, Y., Gonthier, G., Ould Biha, S., Pasca, I.: Canonical big operators. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 86–101. Springer, Heidelberg (2008)
[6] de Bruijn, N.G.: The mathematical language AUTOMATH, its usage, and some of its extensions. In: Laudet, M., Lacombe, D., Nolin, L., Schützenberger, M. (eds.) Symposium on Automatic Demonstration. Lecture Notes in Mathematics, vol. 125, pp. 29–61. Springer, Heidelberg (1970)
[7] Feit, W., Thompson, J.G.: Solvability of groups of odd order. Pacific Journal of Mathematics 13(3), 775–1029 (1963)
[8] Garillot, F.: Generic Proof Tools and Finite Group Theory. PhD thesis, École polytechnique (2011)
[9] Gonthier, G.: Point-free, set-free concrete linear algebra. In: van Eekelen, M., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) ITP 2011. LNCS, vol. 6898, pp. 103–118. Springer, Heidelberg (2011)
[10] Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Roux, S.L., Mahboubi, A., O'Connor, R., Biha, S.O., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., Théry, L.: A machine-checked proof of the odd order theorem. To appear in the Proceedings of the ITP 2013 Conference (2013)
[11] Gonthier, G., Mahboubi, A.: An introduction to small scale reflection in Coq. Journal of Formalized Reasoning 3(2), 95–152 (2010)
[12] Gonthier, G., Mahboubi, A., Rideau, L., Tassi, E., Théry, L.: A Modular Formalisation of Finite Group Theory. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007. LNCS, vol. 4732, pp. 86–101. Springer, Heidelberg (2007)
[13] Gonthier, G., Mahboubi, A., Tassi, E.: A Small Scale Reflection Extension for the Coq system. Rapport de recherche RR-6455, INRIA (2012)
[14] Hedberg, M.: A coherence theorem for Martin-Löf's type theory. Journal of Functional Programming 8(4), 413–436 (1998)
[15] Hölder, O.: Zurückführung einer beliebigen algebraischen Gleichung auf eine Kette von Gleichungen. Mathematische Annalen 34(1), 26–56 (1889)
[16] Isaacs, I.: Character Theory of Finite Groups. AMS Chelsea Pub. Series (1976)
[17] Jordan, C.: Traité des substitutions et des équations algébriques. Gauthier-Villars, Paris (1870)
[18] Kurzweil, H., Stellmacher, B.: The Theory of Finite Groups: An Introduction. Universitext Series. Springer (2010)
[19] Mahboubi, A., Tassi, E.: Canonical structures for the working coq user. To appear in the Proceedings of the ITP 2013 Conference (2013)
[20] Peterfalvi, T.: Character Theory for the Odd Order Theorem. London Mathematical Society, LNS, vol. 272. Cambridge University Press (2000)
[21] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study (2013)
[22] Wiedijk, F.: The "de Bruijn factor", http://www.cs.ru.nl/~freek/factor/
[23] Zassenhaus, H.: Zum satz von Jordan-Hölder-Schreier. Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg 10(1), 106–108 (1934)