Addendum About Groups

Reynald Affeldt

March 21, 2017

Contents

The following is a commented reading of the formalization of finite groups provided by the MathComp library. The goal is to explain the definitions and the notations, and to replay some of the proofs in a less compact way for didactic purposes. I used the following references for the mathematics [2, 3] and received comments from Cyril Cohen.

1 Basics About Groups

1.1 Basic Definitions

The formalization of finite groups is essentially built on top of `finset.v`, the formalization of finite sets, hence the following sequence of `Import`s.

```
Require Import mathcomp.ssreflect.ssreflect.
From mathcomp
Require Import ssrbool ssrfun eqtype ssrnat seq choice fintype.
From mathcomp
Require Import div path bigop prime finset fingroup.

Set Implicit Arguments.
Unset Strict Implicit.
Unset Printing Implicit Defensive.
```

```
[Loading ML file ssrmatching_plugin.cmxs ... done]

Small Scale Reflection version 1.6 loaded.
Copyright 2005-2016 Microsoft Corporation and INRIA.
Distributed under the terms of the CeCILL-B license.
```

```
[Loading ML file ssreflect_plugin.cmxs ... done]
```

In MathComp, finite groups are "subgroups" of a "container group". The first part of the container group structure can be found in the `Record mixin_of` (file: `fingroup.v`; `Module FinGroup`): a binary operation, a special element (the neutral), a unary operation (inverse), associativity, left identity, involution, antimorphism (i.e., $(xy)^{-1} = y^{-1}x^{-1}$). Such a structure is not a group because $x^{-1}x = 1$ may not hold.

```
Print FinGroup.mixin_of.
```

```
Record mixin_of (T : Type) : Type := BaseMixin
  { mul : T -> T -> T;
    one : T;
    inv : T -> T;
    _ : associative mul;
    _ : left_id one mul;
    _ : involutive inv;
    _ : {morph inv : x y / mul x y >-> mul y x} }

For BaseMixin: Arguments T, mul, one, inv are implicit
For mixin_of: Argument scope is [type_scope]
For BaseMixin: Argument scopes are [type_scope function_scope _
                 function_scope _ _ _ _]
```

The carrier of a container group is put together with

1. the assumption that it satisfies the structure above and
2. the assumption that the carrier is a finite type.

This forms a `baseFinGroupType`.

```
Print FinGroup.base_type.
Print baseFinGroupType.
```

```
Record base_type : Type := PackBase
  { sort : Type;  _ : FinGroup.mixin_of sort;  _ : Finite.class_of sort }

For PackBase: Argument sort is implicit
For PackBase: Argument scopes are [type_scope _ _]
```

Then, the `baseFinGroupType` is put together with the following law: $\forall x, x^{-1}x = 1$. This gives a group that will act as a "container group" hereafter.

```
Print FinGroup.type.
Print finGroupType.
```

```
Record type : Type := Pack
  { base : baseFinGroupType;
    _ : left_inverse (FinGroup.one base) (FinGroup.inv base)
          (FinGroup.mul base) }

For Pack: Argument base is implicit
```

Finally, a finite group is defined as a set of elements from a finite container group together with the assumption that it is a `group_set`, i.e., that it contains the neutral element and that it is stable by the binary operation. It is worth noticing that the definitive definition of a finite group appears very late in the `fingroup.v` file (almost in the middle of a 3,000 lines file).

```
Print group_set.
Print group_type.
```

```
group_set =
fun (gT : finGroupType) (A : {set gT}) =>
(1%g \in A) && ((A * A)%g \subset A)
     : forall gT : finGroupType, {set gT} -> bool

Argument gT is implicit and maximally inserted
Argument scopes are [_ set_scope]
```

1.2 Basic Usage

How to declare a finite group? First declare a finite container group `gT : finGroupType` and then a finite group using the dedicated notation `G : \{group gT\}` (scope: `type_scope`, notation for `group_of (Phant gT)`). `group _` has type `predArgType` which means that it comes with the generic notation `\in`.

```
Section group_example.

Variable gT : finGroupType.
Variable G : {group gT}.
```

Groups enjoy the following notations.

```
Local Open Scope group_scope.

Check (1 : gT).
Fail Check (1 : G).
Check (1 \in G).
Check (1 * 1 : gT).
Check (1 ^-1 : gT).

Lemma neutral_in_group : 1 \in G.
Proof.
Check group1.
rewrite group1.
done.
Qed.

Lemma neutral_neutral_in_group : 1 * 1 \in G.
Proof.
Search _ left_id mulg in fingroup.
rewrite mul1g.
rewrite group1.
done.
Qed.
```

Point multiplication and inverse are lifted to set of points.

```
Check set_mulg.
Check set_invg.
```

```
set_mulg
     : forall gT : baseFinGroupType,
       {set gT} -> {set gT} -> {set FinGroup.finType gT}
```

For two nonempty subsets A, B of G, let $AB := \{ab|a \in A, b \in B\}$. AB is the *(complex) product* of A and B. When $A = \{a\}$, we write aB instead of AB. A product is not necessarily a group (the multiplication needs to commute for that) but it is at least a `group_set_of_baseGroupType`.

```
Fail Check (G * G : {group gT}).
Check (G * G : group_set_of_baseGroupType gT).
Set Printing All.
Check (1 = [set 1] :> {set gT}).
Unset Printing All.

Lemma neutral_in_group_group : 1 \in G * G.
Proof.
Check mulSGid.
rewrite mulSGid.
rewrite group1.
done.
rewrite subxx.
done.
Qed.
```

```
The command has indeed failed with message:
In environment
gT : finGroupType
G : {group gT}
The term "G * G" has type "FinGroup.sort (group_set_of_baseGroupType gT)"
while it is expected to have type "{group gT}".
```

Let A and B be subgroups of G. Then AB is a subgroup of G iff $AB = BA$.

```
Lemma group_set_group_group : group_set (G * G).
Proof.
Search _ group_set reflect in fingroup.
apply/comm_group_setP.
Search (commute _ _).
apply commute_refl.
Qed.
```

```
1 subgoal

  gT : finGroupType
  G : {group gT}
  ============================
  group_set (G * G)
```

Let U be a subgroup of G and $x \in G$. The product Ux is a *right coset* of U in G. The right coset of H by x is noted `H :* x` in MathComp (notation scope: `group_scope`; file: `fingroup.v`). There is another definition of right cosets (`Definition rcoset`) that is proved equivalent (`Lemma rcosetE`).

```
Variable x : gT.
Variable H : {group gT}.
Check (H :* x : {set gT}).
Locate ":*".
```

```
x is declared
```

The map $H \to Ha; h \mapsto ha$ is injective. Thus a coset Ha has cardinal $|H|$.

```
Lemma mycard_rcoset : #|H :* x| = #| H |.
Proof.
Check card_rcoset.
Check card_imset.
rewrite -[in X in _ = X](@card_imset _ _ (mulg^~ x)); last first.
  Search _ left_injective mulg.
  exact: mulIg.
```

```
rewrite -rcosetE.
rewrite /rcoset.
done.
Qed.
```

```
1 subgoal

  gT : finGroupType
  G : {group gT}
  x : gT
  H : {group gT}
  ============================
  #|H :* x| = #|H|
```

The set of the right cosets of H by elements of G is denoted by `rcosets H G` (file: `fingroup.v`).

```
Check (rcosets H G : {set {set gT}}).
Check (rcosets H G).
```

```
rcosets H G : {set {set gT}}
     : {set {set gT}}
```

If the set of right cosets of U in G is finite then the number of right cosets of U in G is the *index* of U in G, denoted by $|G : U|$ (`Definition indexg`; file `fingroup.v`).

```
Print indexg.

End group_example.
```

```
indexg =
fun (gT : finGroupType) (B A : {set gT}) => #|rcosets A B|
     : forall gT : finGroupType, {set gT} -> {set gT} -> nat

Argument gT is implicit
Argument scopes are [_ group_scope group_scope]
```

2 Lagrange's Theorem

Lagrange's theorem is already proved in `fingroup.v`. In the following, we replay this proof in a less compact way.

```
Check LagrangeI.

Section myLagrange.

Variable gT : finGroupType.

Local Open Scope group_scope.

Variable (H G : {group gT}).
Hypothesis HG : H \subset G.
```

```
LagrangeI
     : forall (gT : finGroupType) (G H : {group gT}),
       #|G :&: H| * #|G : H|%g = #|G|
```

The relation $xy^{-1} \in H$ is an equivalence relation. The equivalence class of x (the set of y such that $xy^{-1} \in H$) is actually the right coset Hx. The set of cosets forms a partition of G. We first prove this fact.

```
Print equivalence_partition.

Lemma rcosets_equivalence_partition :
  rcosets H G = equivalence_partition [rel x y | x * y^-1 \in H] G.
Proof.
apply/setP => /= X.
case/boolP : (X \in equivalence_partition _ _).
  case/imsetP => x Hx.
  move=> ->.
  apply/rcosetsP.
  exists x => //.
  apply/setP => y.
  rewrite inE /=.
  case/boolP : (_ \in _ :* _).
    case/rcosetP => z Hz.
    move=> ->.
    rewrite invMg.
    rewrite mulgA.
    rewrite mulgV.
    rewrite mul1g.
    rewrite groupVr //.
    rewrite andbT.
    rewrite groupM //.
    move/subsetP : HG.
    by apply.
  apply: contraNF.
  case/andP => Hy xy.
  apply/rcosetP.
  exists (y * x^-1).
    rewrite groupVl //.
    rewrite invMg.
    by rewrite invgK.
  rewrite -mulgA.
  rewrite mulVg.
  by rewrite mulg1.
apply: contraNF.
case/rcosetsP => x Hx ->.
apply/imsetP.
exists x => //.
apply/setP => /= y.
rewrite inE.
case/boolP : (_ && _).
  case/andP => Hy xy.
  apply/rcosetP.
  exists (y * x^-1).
    rewrite groupVl //.
    rewrite invMg.
    by rewrite invgK.
  rewrite -mulgA.
  rewrite mulVg.
  by rewrite mulg1.
apply: contraNF.
case/rcosetP => z Hz ->.
apply/andP; split.
  rewrite groupM //.
  move/subsetP : HG.
  by apply.
rewrite invMg.
rewrite mulgA.
rewrite mulgV.
rewrite mul1g.
by rewrite groupVr.
Qed.

Lemma myrcosets_partition : partition (rcosets H G) G.
Proof.
rewrite rcosets_equivalence_partition.
apply/equivalence_partitionP.
move=> x y z Hx Hy Hz /=.
split.
```

```
  rewrite mulgV.
  by rewrite group1.
move=> xy.
case/boolP : (y * _ \in _) => yz.
  move: (groupM xy yz).
  rewrite mulgA.
  rewrite -(mulgA x).
  rewrite mulVg.
  by rewrite mulg1.
apply: contraNF yz => yz.
move/groupVr in xy.
move: (groupM xy yz).
rewrite invMg.
rewrite invgK.
rewrite mulgA.
rewrite -(mulgA y).
rewrite mulVg.
by rewrite mulg1.
Qed.
```

```
equivalence_partition =
fun (T : finType) (R : rel T) (D : {set T}) =>
let Px := fun x : T => [set y in D | R x y] in [set Px x | x in D]
     : forall T : finType, rel T -> {set T} -> {set set_of_finType T}

Argument T is implicit
Argument scopes are [_ _ set_scope]
```

Lagrange's theorem follows from the fact that the right cosets form a partition of G and that each coset has the same cardinal as H.

```
Lemma myLagrange : #| G | = (#|H| * #|G : H|)%nat.
Proof.
have -> : #|G| = \sum_(U in rcosets H G) #|U|.
  move: myrcosets_partition.
  move/card_partition.
  done.
transitivity (\sum_(U in rcosets H G) #|H|).
  apply eq_bigr => /= U HU.
  case/rcosetsP : HU => u hG ->.
  by rewrite mycard_rcoset.
rewrite big_const.
rewrite iter_addn.
rewrite addn0.
rewrite -/(#|G : H|).
```

```
done.
Qed.

End myLagrange.
```

```
2 subgoals

  gT : finGroupType
  H, G : {group gT}
  HG : H \subset G
  X : {set gT}
  x : gT
  Hx : x \in G
  y : gT
  z : gT
  Hz : z \in H
  ============================
  z \in G

subgoal 2 is:
 x * (z * x)^-1 \in H
```

3 Normal Subgroups

For $x, a \in G$ set $x^a := a^{-1}xa$. This element x^a is the *conjugate* of x by a. (notation: `x ^ y`; notation scope: `group_scope`; file: `fingroup.v`).

```
Print conjg.
Locate "^".
```

```
conjg =
fun (T : finGroupType) (x y : T) => y^-1 * (x * y)
     : forall T : finGroupType, T -> T -> T

Argument T is implicit and maximally inserted
Argument scopes are [_ group_scope group_scope]
```

Sample property: $x^1 = x$

```
Check conjg1.
```

```
conjg1
     : forall (T : finGroupType) (x : T), (x ^ 1)%g = x
```

For $g \in G$ we set $B^g := g^{-1}Bg$ and say that B^g is the conjugate of B by g. In MathComp, the conjugate of H by x is denoted by `H :^ x`.

```
Print conjugate.
```

```
conjugate =
fun (gT : finGroupType) (A : {set gT}) (x : gT) => [set (x0 ^ x)%g | x0 in A]
     : forall gT : finGroupType, {set gT} -> gT -> {set FinGroup.finType gT}

Argument gT is implicit and maximally inserted
Argument scopes are [_ group_scope group_scope]
```

The *normalizer* of A? $\{x|A^x \subseteq A\}$ (Notation: `'N(A)`; definition; file: `fingroup.v`).

```
Print normaliser.
Locate "'N".

Section normalisersect.

Variable gT : finGroupType.
Variables A B : {group gT}.
Local Open Scope group_scope.
Hypothesis nor : B \subset 'N(A).

Lemma normaliser_equiv b : b \in B -> A \subset A :^ b.
Proof.
move=> bB.
suff : A :^ b^-1 \subset A.
  by rewrite -sub_conjgV.
move/subsetP : nor.
move/(_ b^-1).
move/groupVr in bB.
move/(_ bB).
rewrite /normaliser.
rewrite inE.
done.
Qed.

End normalisersect.
```

```
normaliser =
fun (gT : finGroupType) (A : {set gT}) => [set x | (A :^ x)%g \subset A]
     : forall gT : finGroupType, {set gT} -> {set gT}

Argument gT is implicit
Argument scopes are [_ group_scope]
```

There are many ways to state the fact that a subgroup is normal. For example, a subgroup N of G that satisfies $Nx = xN$ for all $x \in G$ is a *normal* subgroup of G (or is normal in G). We write $N \trianglelefteq G$ if N is normal in G. H is normal in G is noted `H <| G` in MathComp, it is a boolean binary predicate (definition: `normal`; notation scope: `group_scope`; file: `fingroup.v`).

```
Print normal.
```

```
normal =
fun (gT : finGroupType) (A B : {set gT}) =>
(A \subset B) && (B \subset ('N(A))%g)
     : forall gT : finGroupType, {set gT} -> {set gT} -> bool

Argument gT is implicit and maximally inserted
Argument scopes are [_ group_scope group_scope]
```

The following example was originally taken from [1] (`exercises-10.v`).

```
Section normalsect.

Variable gT : finGroupType.
Variables (H G : {group gT}).
Local Open Scope group_scope.
Hypothesis HG : H <| G.

Lemma normal_commutes : H * G = G * H.
Proof.
case/normalP : HG => HG' nor.
apply/setP => x.
case/boolP : (_ \in G * _).
  case/mulsgP => x1 x2 Hx1 Hx2 x1x2.
  move: (nor _ Hx1).
  move/setP/(_ x2).
  rewrite Hx2.
  case/imsetP => h1 Hh1 x2x1.
  rewrite x1x2 x2x1.
  rewrite -conjgC.
  apply/mulsgP.
  by exists h1 x1.
rewrite (mulGSid HG').
rewrite (mulSGid HG').
by move/negbTE.
Qed.

End normalsect.
```

References

[1] Yves Berthot, Assia Mahboubi, Laurence Rideau, Pierre-Yves Strub, Enrico Tassi, and Laurent Théry. International Spring School on Formalization of Mathematics (MAP 2012), March 12–16, 2012, Sophia Antipolis, France, 2012. Available at: `http://www-sop.inria.fr/manifestations/MapSpringSchool`. Last access: 2014/08/05.

[2] Hans Kurzweil and Bernd Stellmacher. *The Theory of Finite Groups—An Introduction*. Springer, 2004.

[3] Frédérique Oggier. Algebraic methods. Available at: `http://www1.spms.ntu.edu.sg/~frederique/AA11.pdf`, Nov. 2011.