A bit of group theory

Enrico Tassi
15 March

MAP INTERNATIONAL
SPRING SCHOOL
ON FORMALIZATION OF
MATHEMATICS 2012
SOPHIA ANTIPOLIS, FRANCE / 12-16 MARCH

Inria informatics mathematics
ihp Institut Henri Poincaré
MICROSOFT RESEARCH INRIA JOINT CENTRE

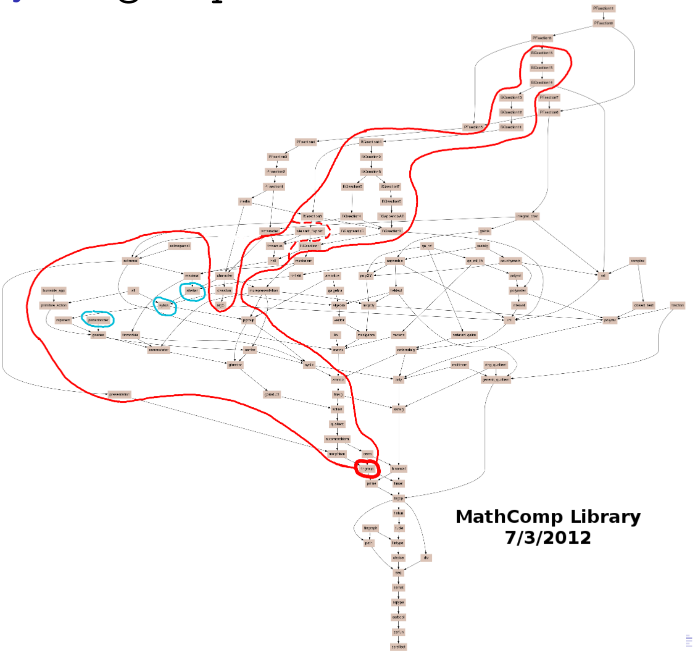
Outline

Groups in type theory

- Guided tour to the fingroup library
- Talking about groups in Coq

Lagrange theorem

Why `fingroup.v`



MathComp Library
7/3/2012

Groups in a nutshell

from “The theory of finite groups” by H.Kurzweil and B.Stellmacher

A nonempty set `G` is a group, if to every `x` and `y` both `\in G` an element `x * y \in G` is assigned, the product of `x` and `y`, satisfying the following axioms:

Associativity: `x * (y * z)= (x * y)* z` for all `x,y,z \in G`

Existence of identity: There exists an element `1 \in G` such that `1 * x = x * 1 = x` for all `x \in G`

Existence of inverses: For every `x \in G` there exists an element `x^-1 \in G` such that `x * x^-1 = 1 = x^-1 * x`

Looking around

Groups have point-wise operations:

- `g * h`, `g^-1`
- `g^h := h^-1 * (g * h)`
  Note: `commute g h -> g^h = g`
- `[~ g, h] := g^-1 * g^h`
  Note: `g * h = h * g * [~ g, h]`

Groups have mixed operations:

- `H :* g := H * [set g]`
- `G :^ h := [set g ^ h | g <- G]`

Looking around

Groups have set-wise operations:

- `#|G|`
- `G * H := [set g * h | g <- G, h <- H]`
- `G :&: H := [set x | (x \in A) && (x \in B)]`
- `#|G : H| := #|rcosets H G| =`
  `#| [set rcoset H g | g <- G] |`
- `'N(G):= [set x | G :^ x \subset G]`

Groups have predicates:

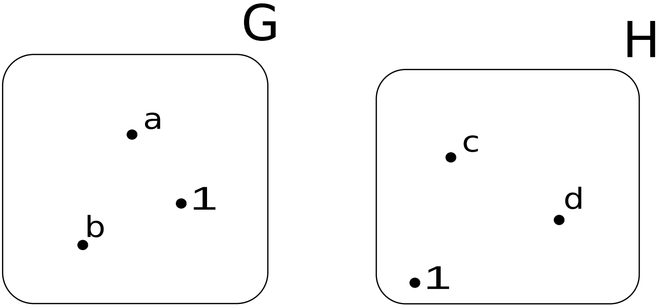
- `H <| G := (H \subset G) && (G \subset 'N(H))`

Groups have funny properties:

- `group_modr A B G :`
  `B \subset G -> (G :&: A) * B = G :&: A * B`

Most of the concepts and proofs are at the set level

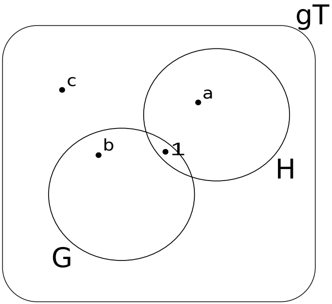
Group = Type



One may model groups as types:

- very general case (equality means isomorphic), but…
- common stuff is hard to write: `a * c`, `G :&: H`, `'Z(G)`
- most of the time, `G` and `H` have a common super group

Group $\neq$ Type



A global (finite) container equipped with group laws

- a group `A` is (sub)set that validates
  `group_set A := (1 \in A) && (A * A \subset A)`
- constructions are mostly set operations (like `:&:`, `'Z( )`)
- set wise constructions lifted to the group level thanks to `Canonical Structure`.

Example

```
Lemma group1 (gT : finGroupType) (G : {group gT}):
  1 \in G.

Lemma example (G H : {group gT}) : 1 \in G :&: H.
set W := G :&: H.
```

As you can see, `G :&: H` is a set:

```
gT : finGroupType
G : {group gT}
H : {group gT}
W := G :&: H : {set gT}
============================
 1 \in W
```

But `rewrite group1` lifts it on the fly using a `Canonical Structure` that contains the lemma:

```
group_setI (G H : {group gT}) : group_set (G :&: H)
```

Finite (intensional) sets

We must find a good "encoding" for sets.

- Sets as characteristic functions
- In Coq functions are not extensional
  $(\forall x.f\ x = g\ x) \not\to f = g$

In a finite setting we can represent functions as their graphs, and finite sets as bitmasks

1	a	b	c	⋯	c^{-1}	b^{-1}	a^{-1}
1	0	0	1	⋯	0	1	0

Equal bitmasks, equal sets: $\forall x.b_1[x] = b_2[x]) \to b_1 = b_2$

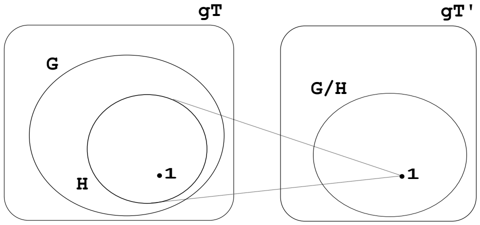
```
setP A B : A =1 B -> A = B
```

Quotients

bad idea

Quotienting `G` by `H` makes sense only if `H <| G`:

`(H :* g1) * (H :* g2) = H :* (g1 * g2)`



Quotients

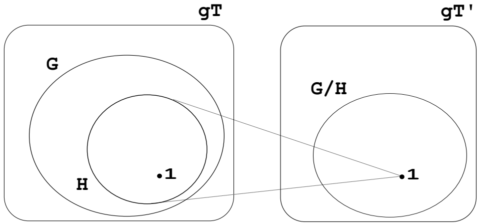
bad idea

Quotienting `G` by `H` makes sense only if `H <| G`:

```
(H :* g1) * (H :* g2)      =      H :* (g1 * g2)

           ||                            ||

  g1 *: (H * H) :* g2 = g1 *: H :* g2
```



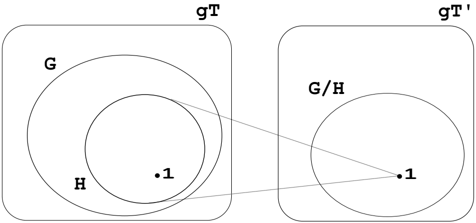
Quotients

bad idea

Quotienting `G` by `H` makes sense only if `H <| G`:

```
(H :* g1) * (H :* g2)    =    H :* (g1 * g2)
          ||                        ||
  g1 *: (H * H) :* g2 = g1 *: H :* g2
```

(bad) idea: `G /p H` where `p` proves `H <| G`



Quotients

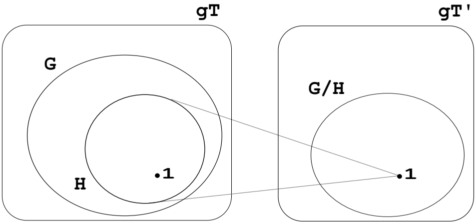
bad idea

Quotienting `G` by `H` makes sense only if `H <| G`:

```
(H :* g1) * (H :* g2)      =      H :* (g1 * g2)
          ||                            ||
   g1 *: (H * H) :* g2 = g1 *: H :* g2
```

(bad) idea: `G /p H` where `p` proves `H <| G`

```
third_isog : (G / H / (K / H)) \isog (G / K)
```



Quotients

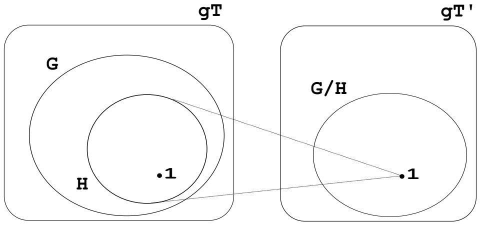
bad idea

Quotienting `G` by `H` makes sense only if `H <| G`:

```
(H :* g1) * (H :* g2)      =      H :* (g1 * g2)
          ||                            ||
     g1 *: (H * H) :* g2 = g1 *: H :* g2
```

(bad) idea: `G /p H` where `p` proves `H <| G`

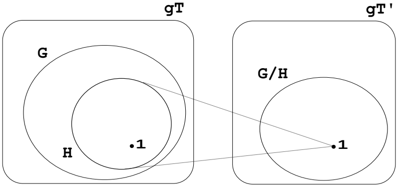
```
third_isog : (G /p1 H /p2 (K /p3 H)) \isog (G /p4 K)
```



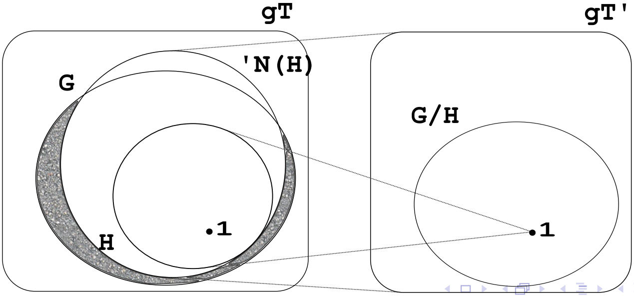
Quotients

making _/_ a total operation

(bad) idea: `G /p H` where `p` proves `H <| G`



(good) idea: `G / H := (G :&: 'N(H)) / H`



Looking around

Quotients

Lemmas about quotients may or may not require the normality assumption:

- ```
  quotientMl A B :
     A \subset 'N(H) ->
     A * B / H = (A / H) * (B / H)
  ```
- ```
  quotientI A B :
  (A :&: B) / H \subset A / H :&: B / H
  ```

Looking around

Other partial constructions

Lemmas about quotients may or may not require the normality assumption:

```
Definition A ><| B :=
  if A :&: B \subset 1%G then ... else set0
```

Looking around

Other partial constructions

Lemmas about quotients may or may not require the normality assumption:

```
Definition A ><| B :=
  if A :&: B \subset 1%G then ... else set0

Lemma sdprodP (A B : {set gT}) (G : {group gT}) :
  A ><| B = G ->
  [/\ are_groups A B, A * B = G,
      B \subset 'N(A) & A :&: B = 1]
```

Outline

The statement

```
H \subset G -> #|H| * #|G : H| = #|G|
```

The proof

an overview

```
Lemma TutorialLaGrange G H : H \subset G -> (#|H| * #|G : H|)%N = #|G| .
move/setIidPr. move=> {1}<-.
rewrite mulnC. rewrite -sum_nat_const.
rewrite -[#|G|]sum1_card.
rewrite (partition_big_imset (rcoset H)) /= -/(rcosets H G).
apply: eq_bigr => /= Hg.
case/rcosetsP. move=> g Gg -> {Hg}.
rewrite -(card_rcoset _ g).
rewrite -sum1_card.
apply: eq_bigl => k.
rewrite group_modr ?sub1set //.
rewrite inE. congr (_ && _).
rewrite rcosetE. apply/idP/eqP.
  by apply: rcoset_transl.
move=> <-.
rewrite -[X in X \in _](mul1g k).
rewrite mem_mulg //.
by rewrite in_set1.
Qed.
```

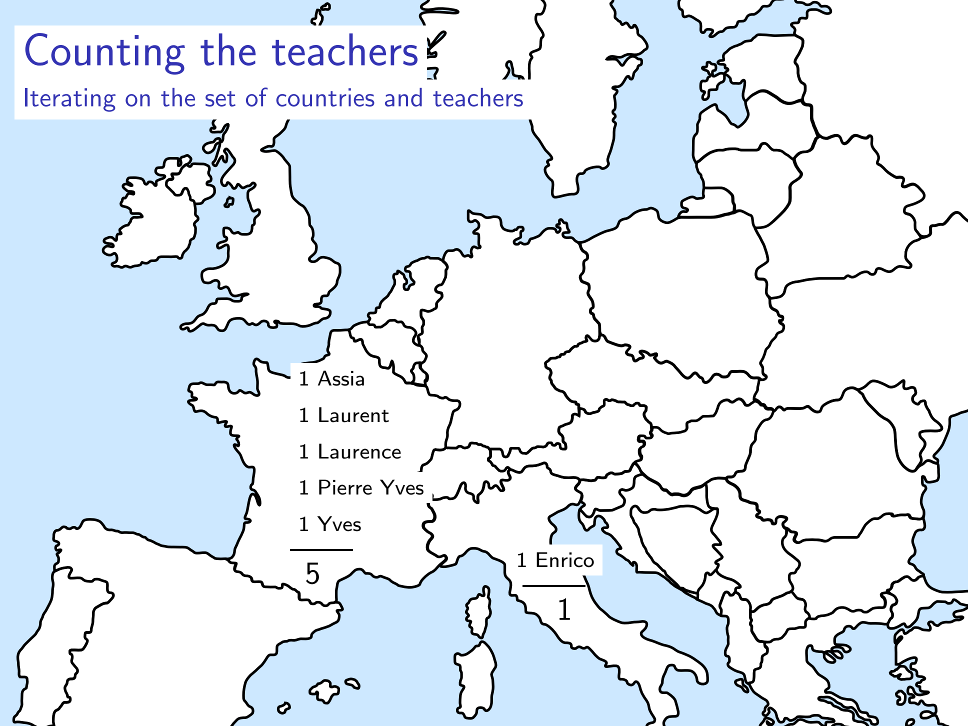
Counting the teachers

Iterating on the set of teachers

Assia	1
Enrico	1
Laurent	1
Laurence	1
Pierre-Yves	1
Yves	1
	6

Counting the teachers

Iterating on the set of countries and teachers



partition_big_imset

```
Lemma partition_big_imset h A F :
  \big[aop/idx]_(i \in A) F i =
     \big[aop/idx]_(j \in h @: A)
      \big[aop/idx]_(i \in A | h i == j) F i.
```

The proof

demo

```
Lemma TutorialLaGrange G H : H \subset G -> (#|H| * #|G : H|)%N = #|G| .
move/setIidPr. move=> {1}<-.
rewrite mulnC. rewrite -sum_nat_const.
rewrite -[#|G|]sum1_card.
rewrite (partition_big_imset (rcoset H)) /= -/(rcosets H G).
apply: eq_bigr => /= Hg.
case/rcosetsP. move=> g Gg -> {Hg}.
rewrite -(card_rcoset _ g).
rewrite -sum1_card.
apply: eq_bigl => k.
rewrite group_modr ?sub1set //.
rewrite inE. congr (_ && _).
rewrite rcosetE. apply/idP/eqP.
  by apply: rcoset_transl.
move=> <-.
rewrite -[X in X \in _](mul1g k).
rewrite mem_mulg //.
by rewrite in_set1.
Qed.
```