First-class Object Hierarchies

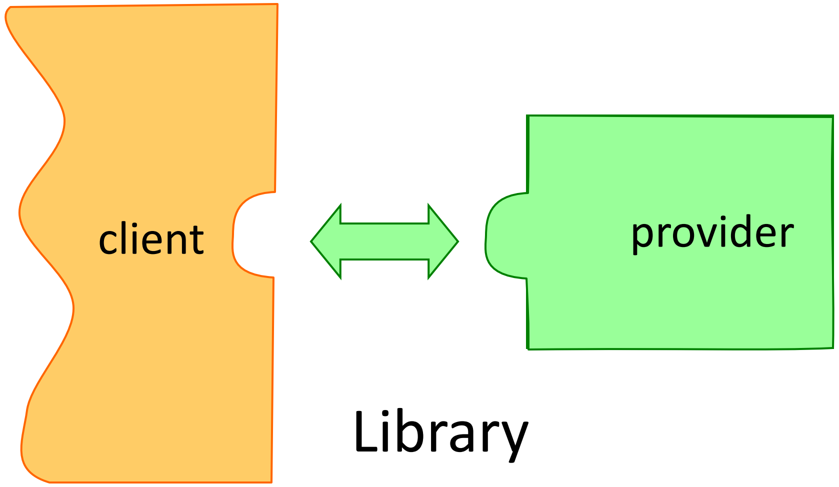
Georges Gonthier
Inria Saclay

inventors for the digital world

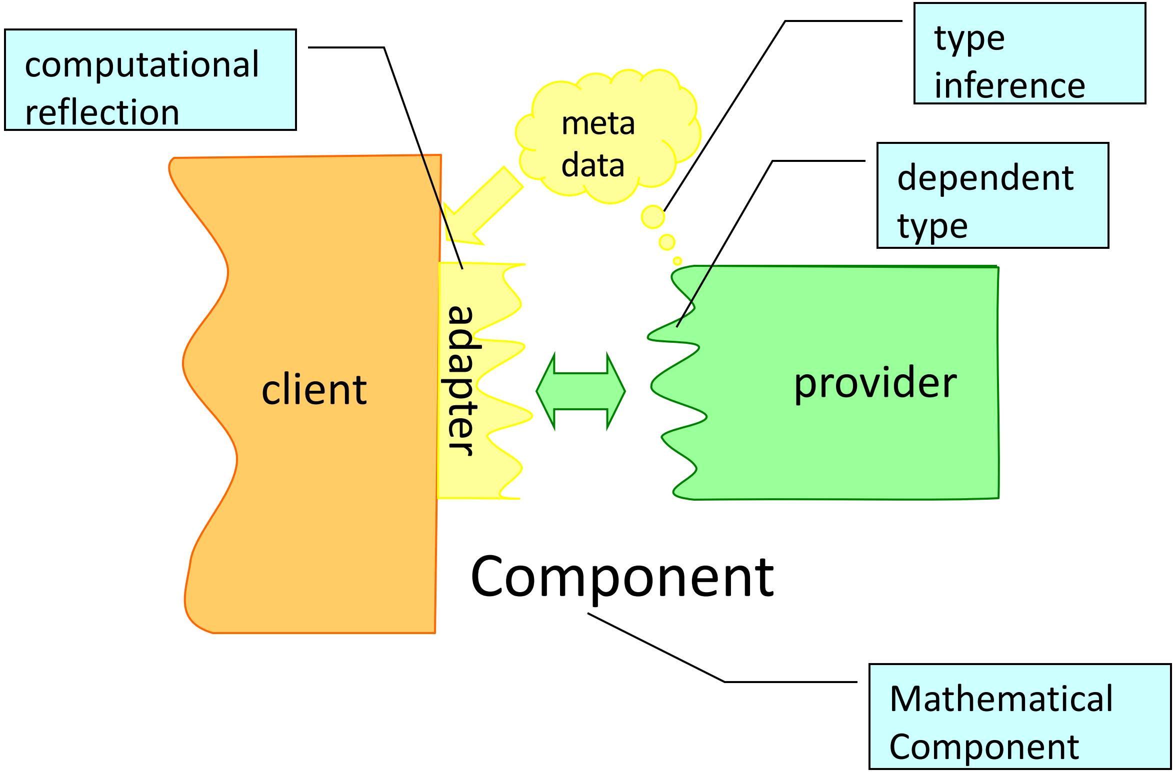
Support for mathematical abstraction

- Modern mathematics master complexity by combining abstract/generic concepts and objects.
- Mathematicians intuitively find the appropriate specialization for such generics.
- Computer mathematics can (partly) imitate this by leveraging object hierarchies.

The mathematical component thesis



The mathematical component thesis



The Math Notation Challenge

$G / \ker_G \varphi \approx \varphi(G)$

$G/K \,/\, H/K \approx G / H$

$HK / K \approx H / H \cap K$

$\sum_{\sigma \in S_n} (-1)^\sigma \prod_i A_{i,i\sigma}$

$\bigwedge_{i=1}^{n}{}_{GCD} Q_i(X)$

$\sum_i V_i$ is direct

$\bigcap_{\substack{H < G \\ H \text{ maximal}}} H$

$D_{2n} \approx \mathrm{Grp}\,(x, y : x^{2^{n-1}}, y^2, x^y = x^{-1})$

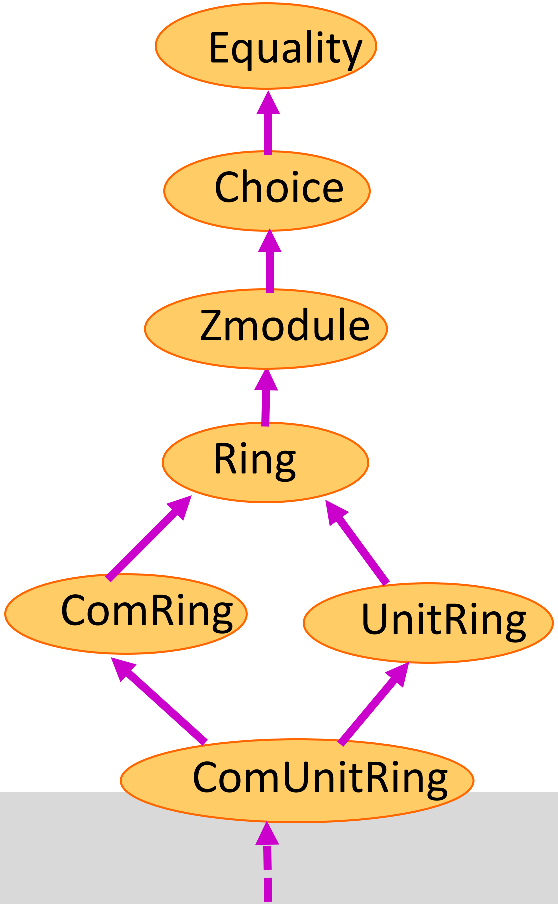
Tool review

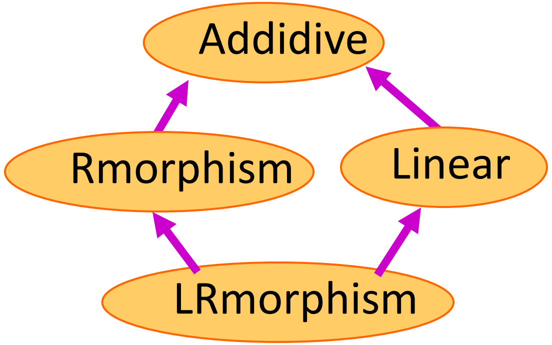
- **Data (inductive) types / propositions**
- **Computational reflection**
  - compute values, types, and propositions
- **Dependent types**
  - first-class Structures
- **Type / value inference**
  - controlled by Coercion / Canonical Structure
- **User notation**

Implementing notation

```
Definition mxtrace (R : ringType) n (A : `M[R]_n) :=
   \sum_i A i i

   @bigop _ _ 0 +%R (index_enum _) (fun i => A i i)

   @bigop R `I_n 0 +%R (index_enum _)
        (fun i : `I_n => fun_of_matrix A i i)
```

Algebra interfaces

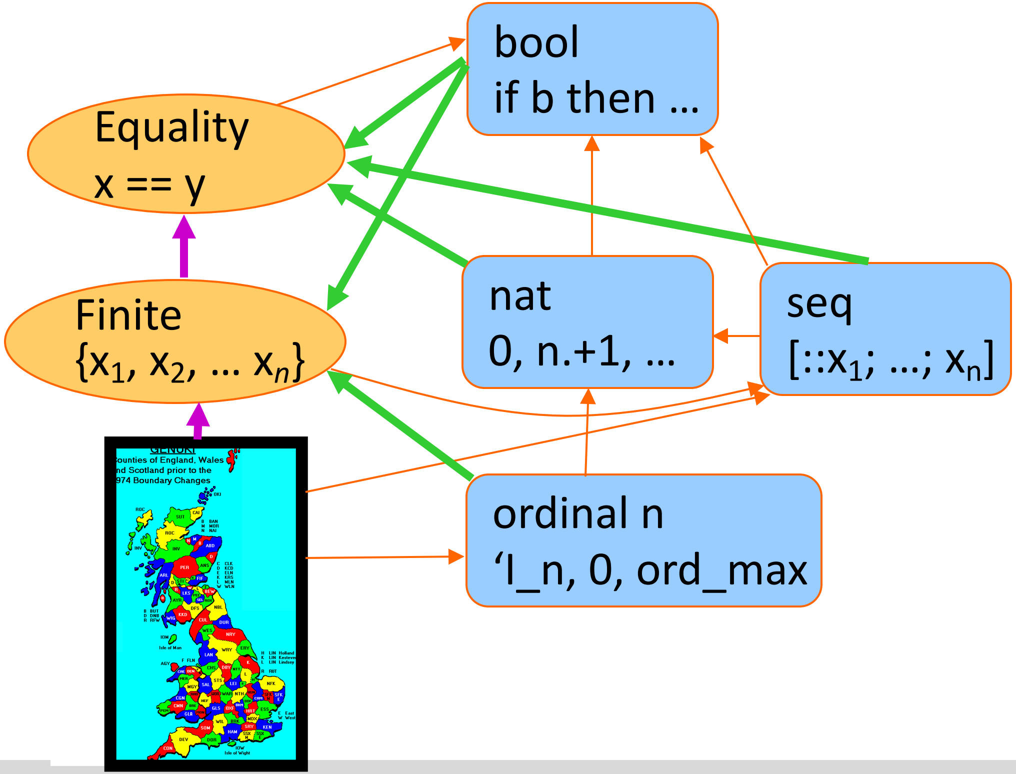




Inferring notation

```
Definition mxtrace (R : ringType) n (A : `M[R]_n) :=
   @bigop R `I_n 0 (@Gring.add (Ring.ZmodType R))
       (index_enum _)
       (fun i : `I_n => fun_of_matrix A i i)
```

Basic interfaces and objects



Ad hoc inference

```
Definition mxtrace (R : ringType) n (A : `M[R]_n) :=
   @bigop R `I_n 0 (@Gring.add (Ring.ZmodType R))
       (index_enum (ordinal_finType n))
       (fun i : `I_n => fun_of_matrix A i i)
```

Little math

The maths of pencil and paper

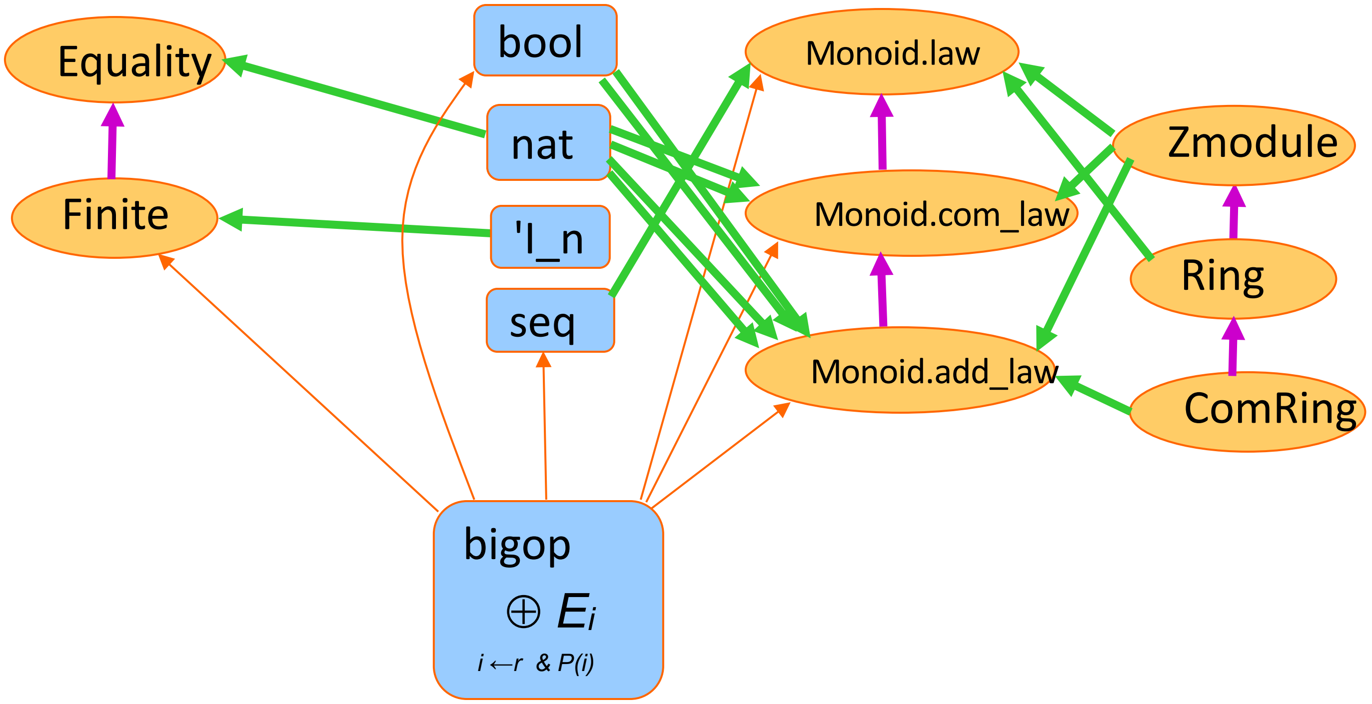
Combinatorics, linguistics, arithmetic, …

Instrumental to taming formalization size

… and making sense of informal statements

Makes for better math, better notation

Interfacing big operators



Linear operator interface : a function class

Encapsulate f (λv) = λ(f v)

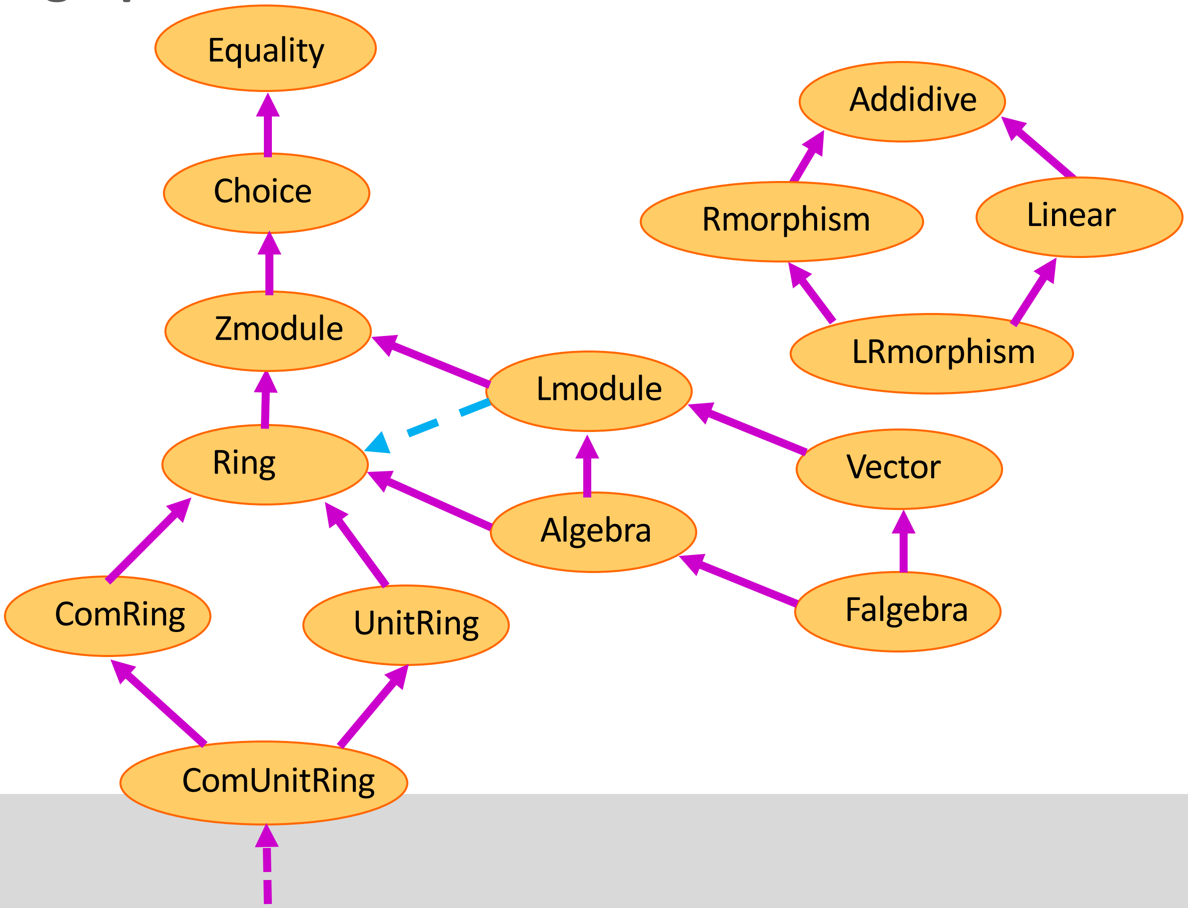
```
Module Linear.
Section ClassDef.
Variables (R : ringType) (U V : lmodType R).
Definition mixin_of (f : U -> V)  :=
   forall a, {morph f : u / a *: u}.
Record class_of f : Prop :=
   Class {base : additive f; mixin : mixin_of f}.
Structure map :=
  Pack {apply :> U -> V; class : class_of apply}.
Structure additive cT := Additive (base (class cT)).
End Linear.
```

Generic Lemmas

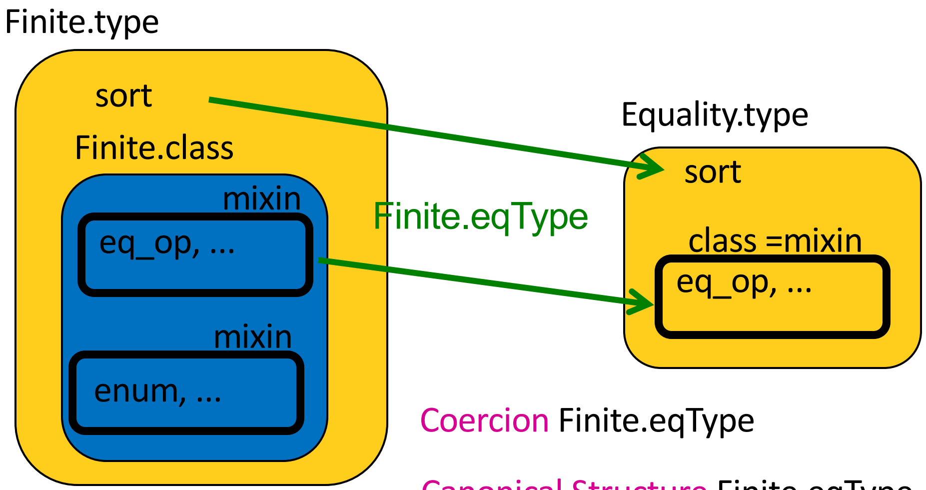
Pull, split, reindex, exchange …

```
Lemma bigD1 : forall (I : finType) (j : I) P F,
  P j -> \big[*%M/1]_(i | P i) F i
    = F j * \big[*%M/1]_(i | P i && (i != j)) F i.

Lemma big_split : forall I (r : list I) P F1 F2,
  \big[*%M/1]_(i <- r | P i) (F1 i * F2 i) =
    \big[*%M/1]_(i <- r | P i) F1 i * \big[*%M/1]_(i <- r | P i) F2 i.

Lemma reindex : forall (I J : finType) (h : J -> I) P F,
  {on P, bijective h} ->
  \big[*%M/1]_(i | P i) F i = \big[*%M/1]_(j | P (h j)) F (h j).

Lemma bigA_distr_bigA : forall (I J : finType) F,
  \big[*%M/1]_(i : I) \big[+%M/0]_(j : J) F i j
    = \big[+%M/0]_(f : {ffun I -> J}) \big[*%M/1]_(i) F i (f i).
```

Inheritance graph



Class structures

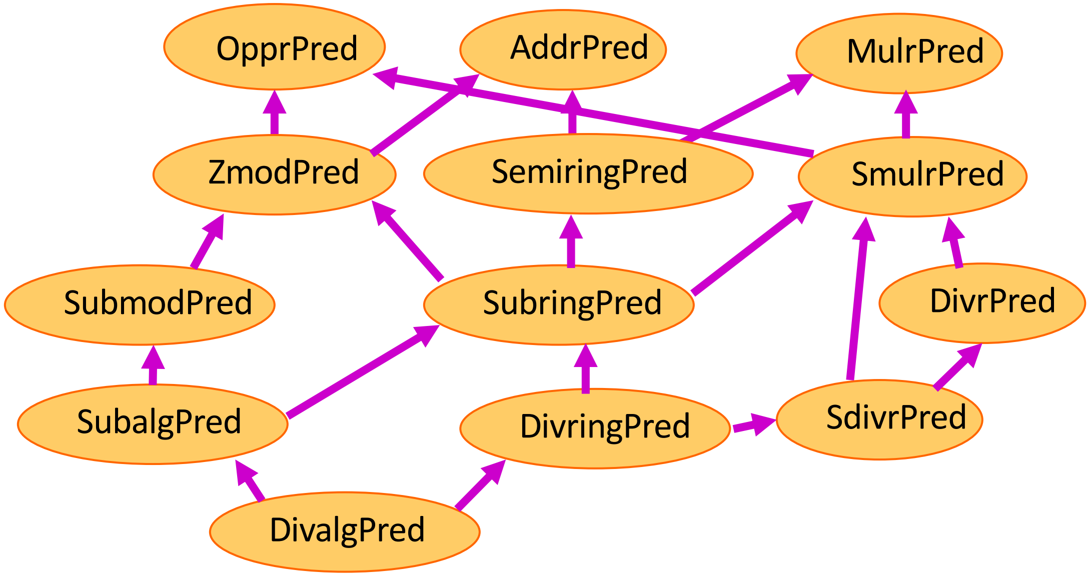


Coercion Finite.eqType

Canonical Structure Finite.eqType

aT ≡ Finite.sort aT ≡ Equality.sort (Finite.eqType aT)

Algebraic subsets



z \in Cnat alpha \in algInt
p \is monic phi \is a character

Value classes

```
Structure tuple n T := Tuple {tval :> seq T; _ : size tval == n}.

Notation "n .-tuple" := (tuple n) : type_scope.

Definition tuple_of n T t of phantom (@tval n T t) := t.

Notation "[ 'tuple' 'of' s ]" := (tuple_of (Phantom s)).

Let half_rot3 t := [tuple of map half (rot 3 t)].
```

Some group theory notions

subgroup $H \leq G$ — $\{1\} \cup H^2 = H \subset G$

normaliser $N_G(H)$ — $\{x \in G \mid Hx = xH \text{ (or } H^x = H)\}$

normal subgroup $H \trianglelefteq G$ — $H \leq G \leq N_G(H)$

factor group G / H — $\{Hx \mid x \in N_G(H)\}$

morphism $\varphi : G \to H$ — $\varphi(xy) = (\varphi x)(\varphi y)$ if $x, y \in G$

action $\alpha : S \to G \to S$ — $a(xy)_\alpha = ax_\alpha\, y_\alpha$ if $x, y \in G$

\+ **group set** A AB, 1, A^{-1} pointwise

\+ **group type** xy, 1, x^{-1}

Groups are sets

Need x∈G & x∈H → groups are not types

Group theory is really subgroup theory.

In Coq :

```
Variable gT : finGroupType.
Definition group_set (G : {set gT}) :=
    (1 ∈ G) && (G*G ⊆ G).
```

Need G : {set gT} and gG : group_set G

but gG can be inferred from G.

Subgroup theory

group H	$\{1\} \cup H^2 = H$
normaliser N (H)	$\{x \in G \mid Hx = xH \text{ (or } H^x = H)\}$
normal subgroup $H \trianglelefteq G$	$H \leq G \leq N(H)$
factor group G / H	$\{Hx \mid x \in N_G(H)\}$
morphism $\varphi : G \to H$	$\varphi(xy) = (\varphi x)(\varphi y)$ if $x, y \in G$
action $\alpha : S \to G \to S$	$a(xy)_\alpha = ax_\alpha\, y_\alpha$ if $x, y \in G$

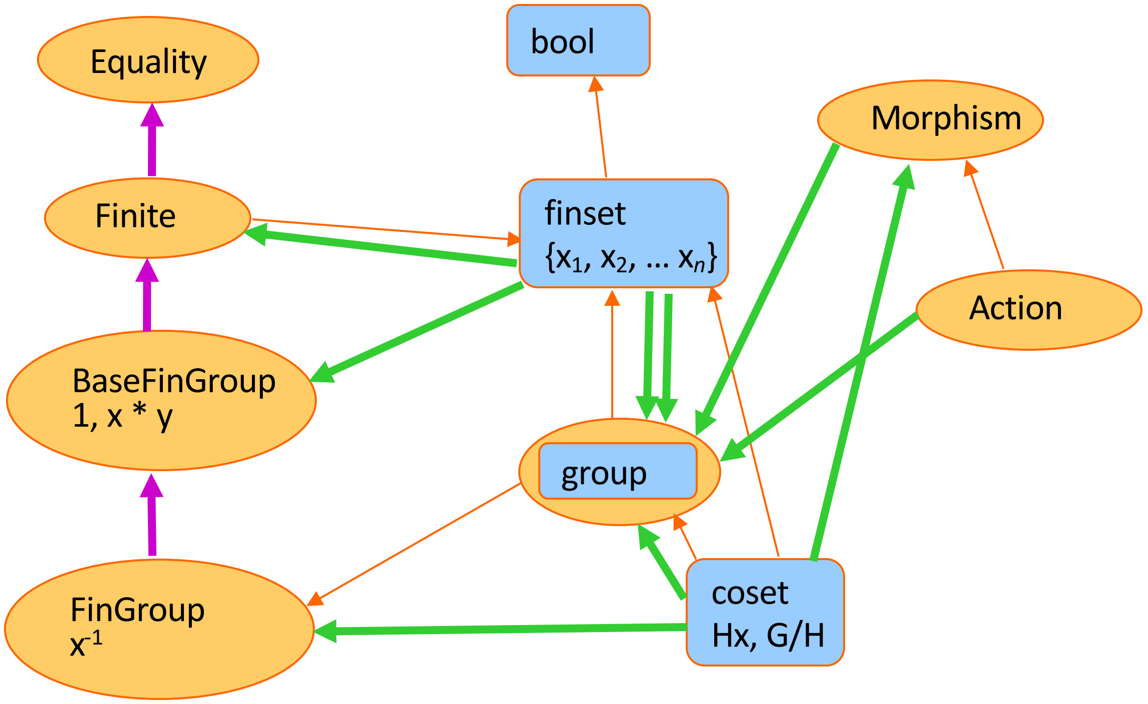
+ **group set** A   AB, 1, A^{-1} pointwise

+ **group type**   xy, 1, x^{-1}

Groups as objects

```
Definition Gset gT := {set gT}.
Structure group gT := Group {
  gval :> Gset gT;
  _ : group_set gval
}.

Identity Coercion Gset_elt : Gset >-> FinGroup.sort.
Identity Coercion Gset_set : Gset >-> set_type.
```

Interfacing groups



Cosets and quotients

```
Notation H := <<A>>.
Definition coset_range := [pred B in rcosets H 'N(A)].
Record coset_of := Coset {
  set_of_coset :> Gset gT;
   _ : coset_range set_of_coset }.
```

$G/H \stackrel{\text{def}}{=} N_G(H)\langle H\rangle \mathbin{/} \langle H\rangle$!!

```
Definition coset x : coset_of := insubd (1 : coset_of) (H :* x).
Lemma coset_morphM :
   {in 'N(A) &, {morph coset : x y / x * y}}.
Canonical coset_morphism := Morphism coset_morphM.
Definition quotient Q : {set coset_of} := coset @* Q.
```

Sylow's theorem

Let p be a prime and G is a group.

Definition: P is a Sylow p-subgroup of G if $P \subset G$, $|P|$ is a power of p and p does not divide $|G : P|$.

Write $Syl_p(G)$ for the set of Sylow p-subgroups of G.

Theorem (Sylow):

a) The Sylow p-subgroups of G are its maximal p-subgroups.
b) G acts transitively by conjugation on $Syl_p(G)$.
c) $|Syl_p(G)| = |N_G(P)|$
d) $|Syl_p(G)| \equiv 1 \pmod{p}$

Sylow's theorem

```
Definition pSylow p A B :=
  [&& B \subset A, p.-nat #|B| & p^'.-nat #|A : B|].
Definition Syl p A := [set P : {group gT} | pHall p A P].
Theorem Sylow's_theorem :
  [/\ forall P, [max P | p.-subgroup(G) P] = p.-Sylow(G) P,
      [transitive (G | 'JG) on 'Syl_p(G)],
      forall P, p.-Sylow(G) P -> #|'Syl_p(G)| = #|G : 'N_G(P)|
   &  prime p -> #|'Syl_p(G)| %% p = 1%N ].
```

Sylow's theorem

```
Theorem Sylow's theorem :
  [/\ forall P, [max P | p.-subgroup(G) P] = p.-Sylow(G) P,
      [transitive (G | 'JG) o
      forall P, p.-Sylow(G) P
   &  prime p -> #|'Syl_p(G)|
Proof.
pose maxp A P := [max P | p.-
pose oG := orbit 'JG%act G.
have actS: [acts (G | 'JG) on
  apply/subsetP=> x Gx; rewri
  exact: max_pgroupJ.
have S_pG: forall P, P \in S
  by move=> P; rewrite inE; c
have SmaxN: forall P Q, Q \in
  move=> P Q; rewrite inE; ca
  apply/maxgroupP; rewrite /p
  by split=> // R; rewrite su
have nrmG: forall P, P \subse
  by move=> P sPG; rewrite /n
have sylS: forall P, P \in S
  move=> P S_P; have [sPG  pP
  by rewrite normal_max_pgrou
have{SmaxN} defCS: forall P,
  move=> P S_P; apply/setP=>
  apply/andP/set1P=> [[S_Q nQ
  apply: val_inj; symmetry; case: (S_pG Q) => //= sQG _.
  by apply: uniq_normal_Hall (SmaxN Q _ _ _) => //=; rewr
```

```
have{defCS} oG_mod: {in S &, forall P Q, #|oG P| %% p = (Q \in oG P) %% p}.
  move=> P Q S_P S_Q; have [sQG pQ] := S_pG _ S_Q.
  have soP_S: oG P \subset
  have: [acts (Q | 'JG) on
    apply/actsP=> x; move/(
    exact: mem_imset.
  move/pgroup_fix_mod=> ->
  rewrite (cardsD1 Q) setDE
  by rewrite inE set11 andb
have [P S_P]: exists P, P \
  have: p.-subgroup(G) 1 by
  by case/(@maxgroup_exists
have trS: [transitive (G |
  apply/imsetP; exists P =>
  rewrite eqEsubset andbC a
  have:= S_P; rewrite inE;
  case/pgroup_1Vpr=> [|[p_p
    move/group_inj=> -> max
    by rewrite (group_inj (
  have:= oG_mod _ _ S_P S_P
  by case: {+}(Q \in _) =>
```

```
have oS1: prime p -> #|S| %% p = 1%N.
  move=> pr_p; rewrite -(atransP trS P S_P) (oG_mod P P) //.
  by rewrite orbit_refl modn_small ?prime_gt1.
have oSiN: forall Q, Q \in S -> #|S| = #|G : 'N_G(Q)|.
  by move=> Q S_Q; rewrite -(atransP trS Q S_Q) card_orbit
     conjG_astab1.
have sylP: p.-Sylow(G) P.
  rewrite pHallE; case: (S_pG P) => // -> /= pP.
  case p_pr: (prime p); last first.
    rewrite p_part lognE p_pr /=.
    by case/pgroup_1Vpr: pP p_pr => [-> _ | [-> //]]; rewrite cards1.
  rewrite -(LaGrangeI G 'N(P)) /= mulnC partn_mul ?cardG_gt0 //
     part_p'nat.
    by rewrite mul1n (card_Hall (sylS P S_P)).
  by rewrite p'natE // -indexgI -oSiN // /dvdn oS1.
have eqS: forall Q, maxp G Q = p.-Sylow(G) Q.
  move=> Q; apply/idP/idP=> [S_Q|]; last exact: Hall_max.
  have{S_Q} S_Q: Q \in S by rewrite inE.
  rewrite pHallE -(card_Hall sylP); case: (S_pG Q) => // -> _ /=.
  by case: (atransP2 trS S_P S_Q) => x _ ->; rewrite cardJg.
have ->: 'Syl_p(G) = S by apply/setP=> Q; rewrite 2!inE.
by split=> // Q sylQ; rewrite -oSiN ?inE ?eqS.
Qed.
```

Sylow's theorem

```
pose maxp A P := [max P | p.-subgroup(A) P]; pose S := [set P | maxp G P].
pose oG := orbit 'JG%act G.
have actS: [acts (G | 'JG) on S].
  apply/subsetP=> x Gx; rewrite inE; apply/subsetP=> P; rewrite 3!inE.
  exact: max_pgroupJ.
have S_pG: forall P, P \in S -> P \subset G /\ p.-group P.
  by move=> P; rewrite inE; case/maxgroupP; case/andP.
have SmaxN: forall P Q, Q \in S -> Q \subset 'N(P) -> maxp 'N_G(P) Q.
  move=> P Q; rewrite inE; case/maxgroupP; case/andP=> sQG pQ maxQ nPQ.
  apply/maxgroupP; rewrite /psubgroup subsetI sQG nPQ.
  by split=> // R; rewrite subsetI -andbA andbCA; case/andP=> _; exact: maxQ.
have nrmG: forall P, P \subset G -> P <| 'N_G(P).
  by move=> P sPG; rewrite /normal subsetIr subsetI sPG normG.
have sylS: forall P, P \in S -> p.-Sylow('N_G(P)) P.
  move=> P S_P; have [sPG  pP] := S_pG P S_P.
  by rewrite normal_max_pgroup_Hall ?nrmG //; apply: SmaxN; rewrite ?normG.
have{SmaxN} defCS: forall P, P \in S -> 'C_S(P | 'JG) = [set P].
  move=> P S_P; apply/setP=> Q; rewrite {1}in_setI {1}conjG_fix.
  apply/andP/set1P=> [[S_Q nQP]|->{Q}]; last by rewrite normG.
  apply: val_inj; symmetry; case: (S_pG Q) => //= sQG _.
  by apply: uniq_normal_Hall (SmaxN Q _ _ _) => //=; rewrite ?sylS ?nrmG.
```

inventors for the digital world

Group characters

Definition:

χ character: $\chi(g) = \text{tr}\, X(g)$ for some $X : G \rightarrow M_n(\mathbb{C})$

1. All characters are class functions, $\varphi(g^x) = \varphi(g)$.
2. Characters belong to a euclidean space (norm by average).

3. Irrereducible characters χ_i afforded by the $|G^G|$ irreducible representations X_i form an orthonormal basis.
4. Characters have positive integer coordinates over the irreducibles. Virtual characters (character differences) have integer coordinates.
5. A virtual character ϕ of M with TI-support A extends to an induced virtual character ϕ^G of G with support A^G.
6. $\phi \mapsto \phi^G$ is an isometry.
7. $\phi \mapsto \phi^G$ extends to some coherent sets of characters.

inventors for the digital world

Formalizing characters

Soft typing?

```
Variable gT : finGroupType.
Definition Cfun := {ffun gT -> algC}.
Definition class_fun (G : {set gT}) (phi : Cfun) :=
  {in G &, forall x y, phi (x ^ y) = phi x}.
Definition character G phi :=
  class_fun G phi /\ (forall i, coord (irr G) phi \in
  Cnat).
Definition cfdot (G : {set gT}) (phi psi : Cfun) :=
  #|G|%:R^-1 * \sum_(x in G) phi x * (psi x)^*.
Notation "'[ phi , psi ]_ G" := (cfdot G phi psi).
```

A better interface

Problem: typing assumptions are ubiquitous.

Non/mixed-class-functions never occur.

Make `class_fun G` into a type `` `CF(G) ``, also encapsulating support restriction.

```
Definition is_class_fun (B : {set gT}) (f : {ffun gT ->
   algC}) := [forall x, forall y in B, f (x ^ y) == f x]
    && (support f \subset B).
Record classfun :=
  Classfun {cfun_val; _ : is_class_fun G cfun_val}.
```

Dot product, orthogonal predicates don't use G.

Interface encapsulates character are a semiring.

Shallow reflection

```
Let sumV := (\sum_(i < h) 'V_i)%MS.
(* This is B & G, Proposition 2.4(a) *)
Lemma mxdirect_sum_eigenspace_cycle :
   (sumV :=: 1%:M)%MS /\ mxdirect sumV.
```

In math:

$S = A + \sum_i B_i$ is direct

iff $\text{rank } S = \text{rank } A + \sum_i \text{rank } B_i$

In Coq:

```
Lemma mxdirectP n (E : mxsum_expr n) :
   reflect (\rank E = mxsum_rank E) (mxdirect E).
```

This is generic in the *shape* of E

More type classes

- **Group functors (F. Garillot)**
  - Map “characteristic” subgroups Z(G), $O_p(G)$, $G^{(1)}$,… with (mono/epi/iso)morphisms
  - Including (some) composites
  - Precursor to categories
- **Lemma overloading (Ziliani, Nanevski, Dreyer)**
  - Automatic heap shape matching for spatial Hoare Logic
  - Evolved to Mtac2 (and Lean3)

What else?

- Support for (re)structuring proofs
- Database $\leftrightarrow$ Typeclass $\leftrightarrow$ Unification $\leftrightarrow$ Tactic interoperability
- Open/incomplete proofs (big-O, event spaces)
- Automating the class hierarchy construction (next!)

Questions?