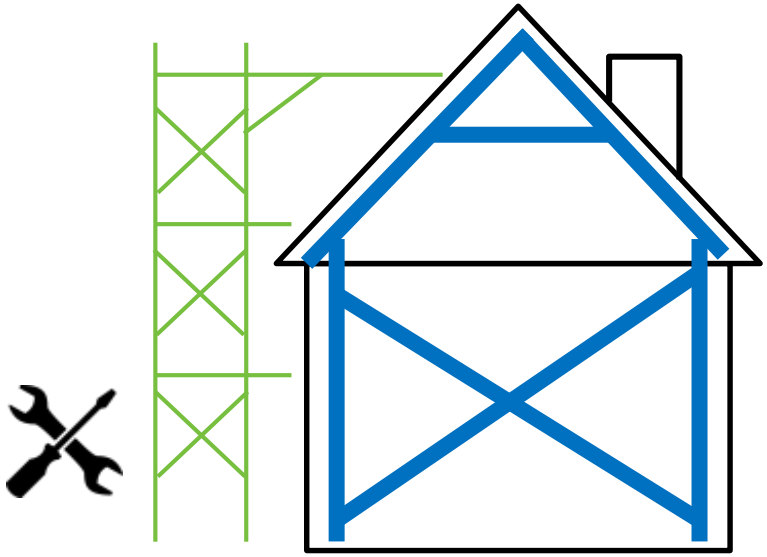
Scaffolds and frames: the MathComp formal algebra library

Georges Gonthier

Inria, Université Paris Saclay

and A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Le Roux, A. Mahboubi, R. O'Connor, S. Ould Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi , L. Théry

The house that Georges built

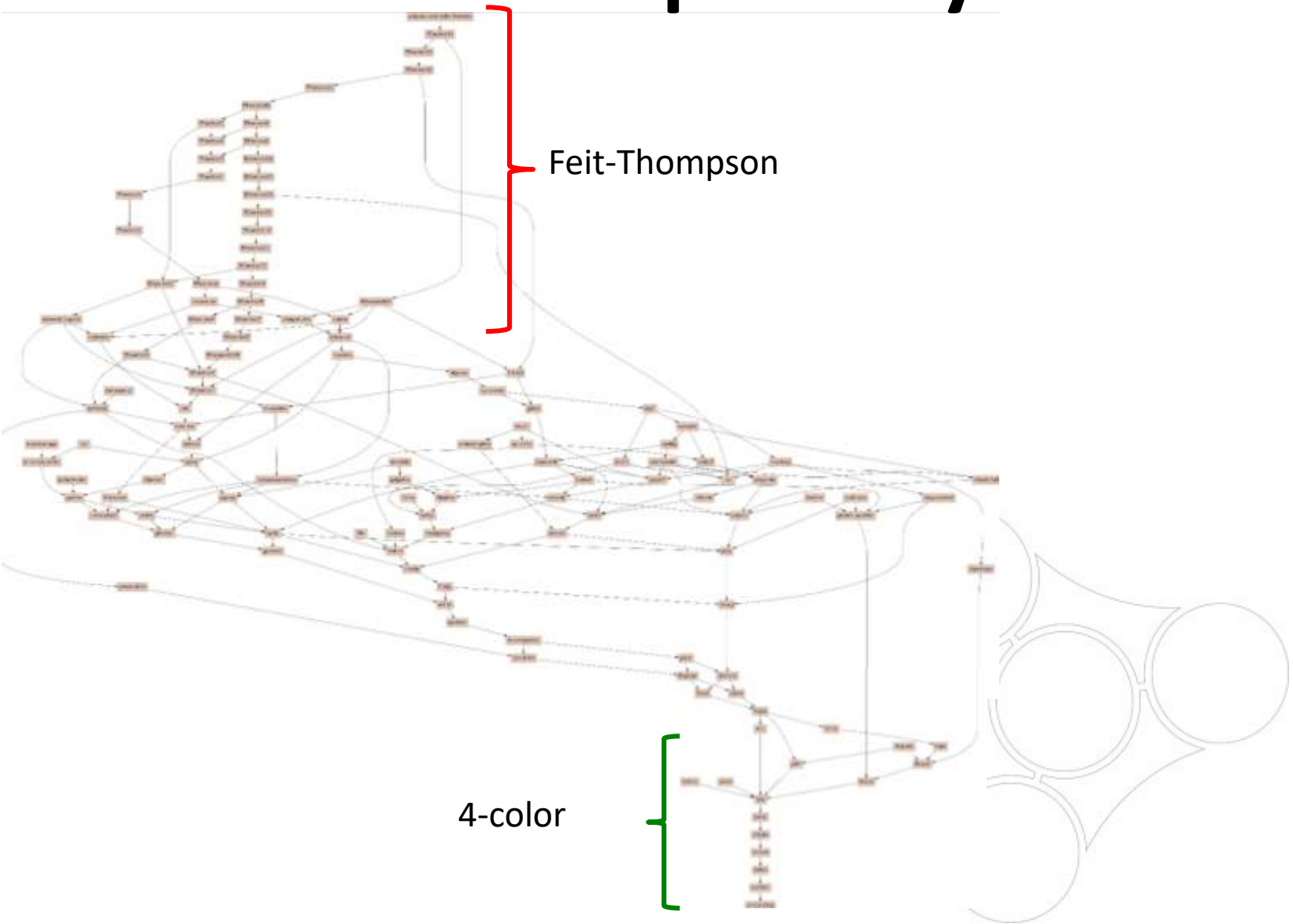


- **Frame**: timber that holds up the house
- **Scaffold**: *temporary* frame
- **Tools**: let builders put stuff together

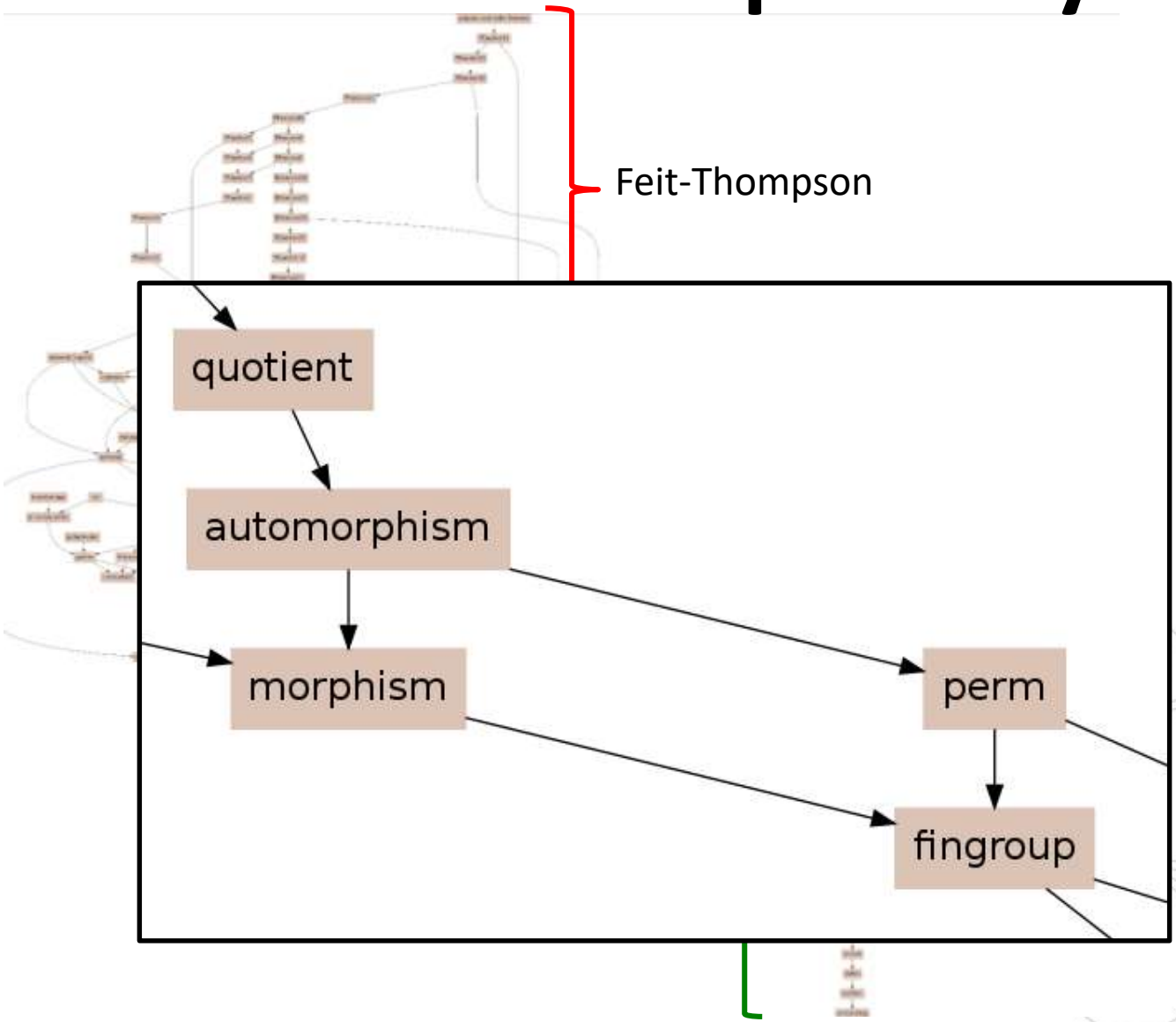
MathComp TOC

- Combinatorics: basic arithmetic, finite lists, sets, graphs and functions; divisibility and bigops.
- Undergraduate algebra: structures, polynomials, matrices, spaces, Galois theory, group and character theory.
- Graduate group theory: modular representations, solvable, Frobenius and special groups, exceptional characters, Feit-Thompson.
- Constructive maths: quantifier elimination, algebraic numbers.

The MathComp library



The MathComp library



The MathComp library

```
`group' constructions (e.g., the normaliser 'N_G(H)) actually define set
with a canonical {group gT} structure; the %G delimiter can be used to
specify the actual {group gT} structure (e.g., 'N_G(H)%G).
 Operations on elements of a group:
              x * y == the group product of x and y.
             x ^+ n == the nth power of x, i.e., x * ... * x (n times).
               x^-1 == the group inverse of x.
             x ^- n == the inverse of x ^+ n (notation for (x ^+ n)^-1)
                  1 == the unit element.
              x ^ y == the conjugate of x by y.
   \prod_(i ...) x i == the product of the x i (order-sensitive).
        commute x y  <-> x and y commute.
     centralises x A <-> x centralises A.
              'C[x] == the set of elements that commute with x.
            'C_G[x] == the set of elements of G that commute with x.
             < [x]> == the cyclic subgroup generated by the element x.
              # [x] == the order of the element x, i.e., #|< [x]>|.
    [~ x1, ..., xn]  == the commutator of x1, ..., xn.
 Operations on subsets/subgroups of a finite group:
              H * G == {xy | x \in H, y \in G}.
  1 or [1] or [1 gT] == the unit group.
         [set: gT]%G == the group of all x : gT (in Group_scope).
            [subg G] == the subtype, set, or group of all x \in G: this
                        notation is defined simultaneously in %type, %g
                        and %G scopes, and G must denote a {group gT}
                        structure (G is in the %G scope).
        subg, sgval == the projection into and injection from [subg G].
               H^# == the set H minus the unit element
```

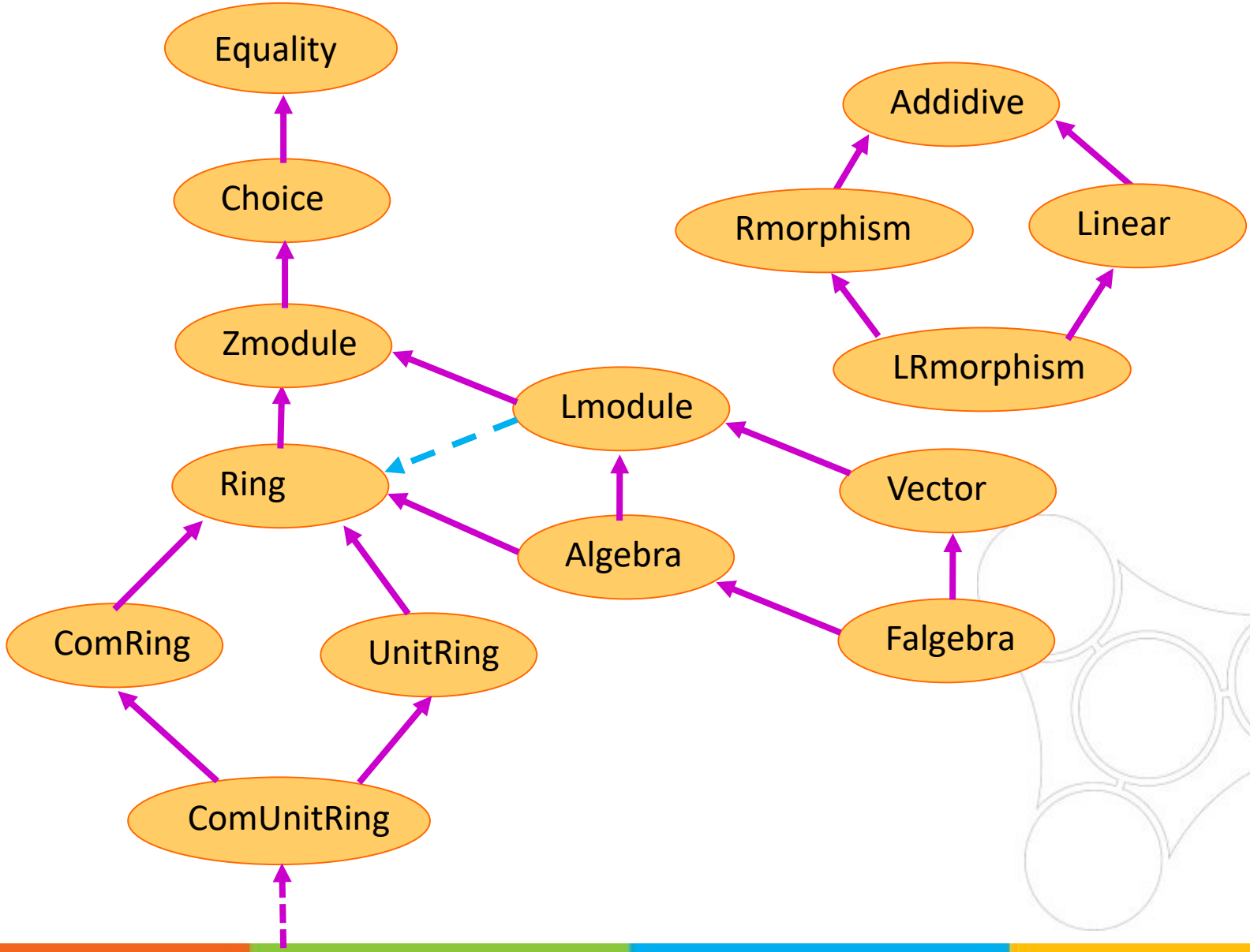
The MathComp library

```
Notation "\prod_ ( i < n ) F" :=
  (\big[joinG/1%G]_(i < n) F%G) : Group_scope.
Notation "\prod_ ( i 'in' A | P ) F" :=
  (\big[joinG/1%G]_(i in A | P%B) F%G) : Group_scope.
Notation "\prod_ ( i 'in' A ) F" :=
  (\big[joinG/1%G]_(i in A) F%G) : Group_scope.

Section Lagrange.

Variable gT : finGroupType.
Implicit Types G H K : {group gT}.

Lemma LagrangeI G H : (#|G :&: H| × #|G : H|)%N = #|G|.

Lemma divgI G H : #|G| %/ #|G :&: H| = #|G : H|.

Lemma divg_index G H : #|G| %/ #|G : H| = #|G :&: H|.

Lemma dvdn_indexg G H : #|G : H| %| #|G|.

Theorem Lagrange G H : H \subset G → (#|H| × #|G : H|)%N = #|G|.

Lemma cardSg G H : H \subset G → #|H| %| #|G|.

Lemma lognSg p G H : G \subset H → logn p #|G| ≤ logn p #|H|.

Lemma piSg G H : G \subset H → {subset \pi(gval G) ≤ \pi(gval H)}.

Lemma divgS G H : H \subset G → #|G| %/ #|H| = #|G : H|.

Lemma divg_indexS G H : H \subset G → #|G| %/ #|G : H| = #|H|.

Lemma coprimeSg G H p : H \subset G → coprime #|G| p → coprime #|H| p.

Lemma coprimegS G H p : H \subset G → coprime p #|G| → coprime p #|H|.

Lemma indexJg G H x : #|G :^ x : H :^ x| = #|G : H|.

Lemma indexgg G : #|G : G| = 1%N.
```

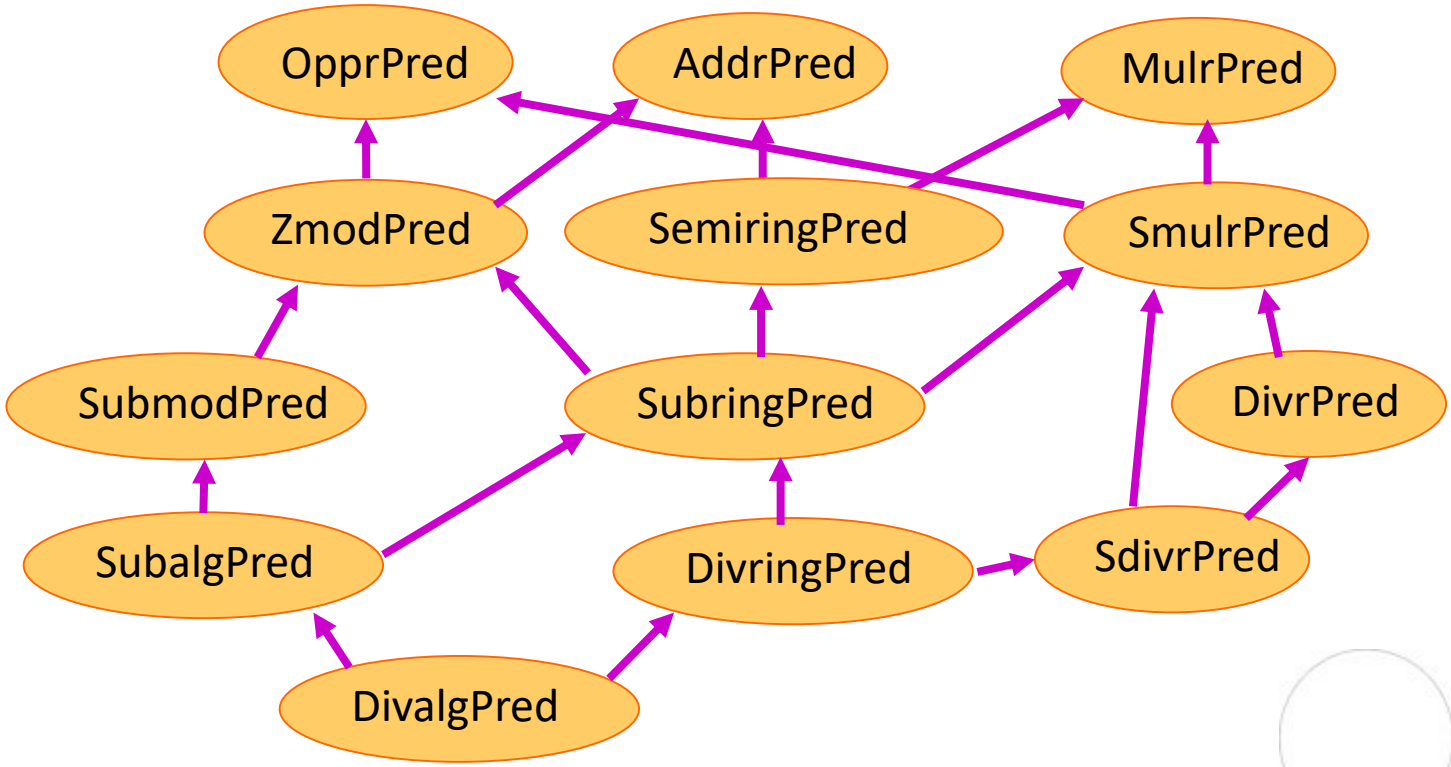
The MathComp library

```
Section Lagrange.

Variable gT : finGroupType.
Implicit Types G H K : {group gT}.

Lemma LagrangeI G H : (#|G :&: H| * #|G : H|)%N = #|G|.
Proof.
rewrite -[#|G|]sum1_card (partition_big_imset (rcoset H)) /=.
rewrite mulnC -sum_nat_const; apply: eq_bigr => _ /rcosetsP[x Gx ->].
rewrite -(card_rcoset _ x) -sum1_card; apply: eq_bigl => y.
rewrite rcosetE eqEcard mulGS !card_rcoset leqnn andbT.
by rewrite group_modr sub1set // inE.
Qed.

Lemma divgI G H : #|G| %/ #|G :&: H| = #|G : H|.
Proof. by rewrite -(LagrangeI G H) mulKn ?cardG_gt0. Qed.

Lemma divg_index G H : #|G| %/ #|G : H| = #|G :&: H|.
Proof. by rewrite -(LagrangeI G H) mulnK. Qed.

Lemma dvdn_indexg G H : #|G : H| %| #|G|.
Proof. by rewrite -(LagrangeI G H) dvdn_mull. Qed.

Theorem Lagrange G H : H \subset G -> (#|H| * #|G : H|)%N = #|G|.
Proof. by move/setIidPr=> sHG; rewrite -{1}sHG LagrangeI. Qed.
```

Algebra interface hierarchy

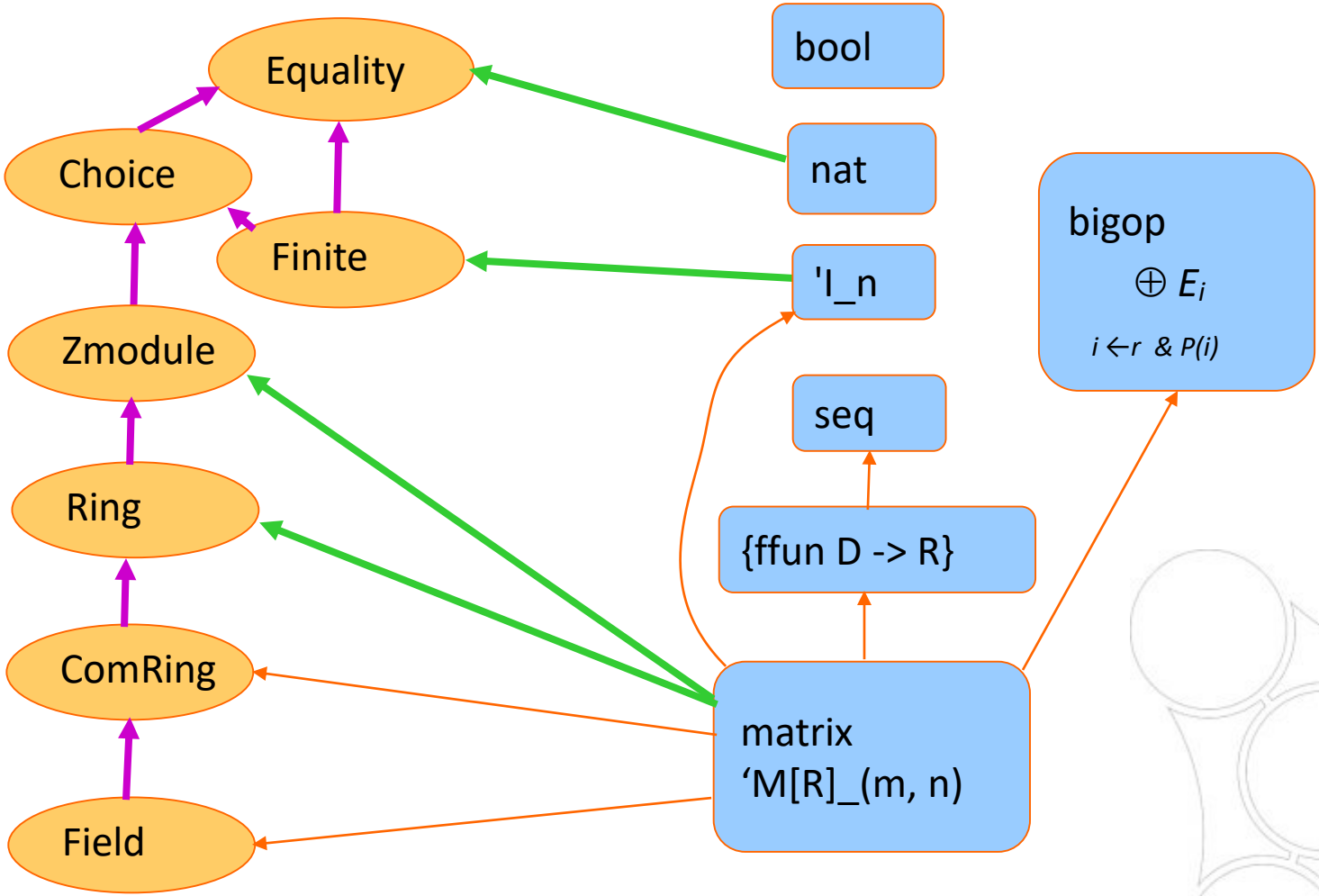


Algebraic subsets

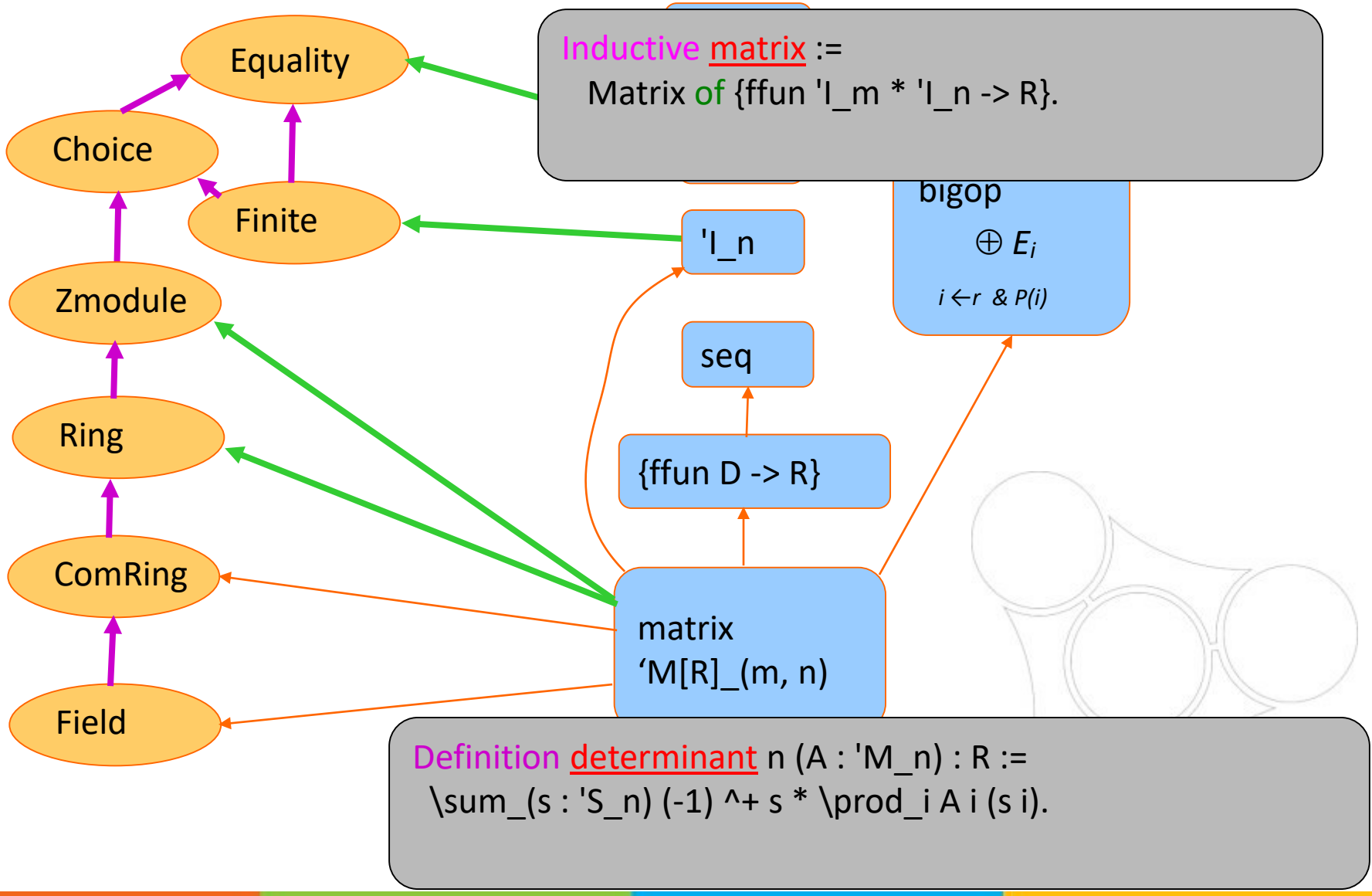


z \in Cnat alpha \in algInt
p \is monic phi \is a character

Interfacing matrices



Interfacing matrices



There's a subtext to math texts

- Definitions and theorems don't state their intended use.
- Notation is concise for efficiency, thus ambiguous/overloaded.
- Proof argument patterns aren't always textbook logic.
- Formalizations must capture some of these conventions to be feasible/relevant.

Wrong paths

- Dependent type family

```
Inductive path T (e : rel T) : T -> T -> Type :=
| Pnil x : path e x x
| Pcons x y z & e x y & path e y z : path e x z
```

- Dependent type

```
Inductive path T (e : rel T) x :=
| Pnil | Pcons y & e x y & path e y z.
Fixpoint last T (e : rel T) x (p : @path T e x) :=
 if p is Pcons _ q then last q else x.
```

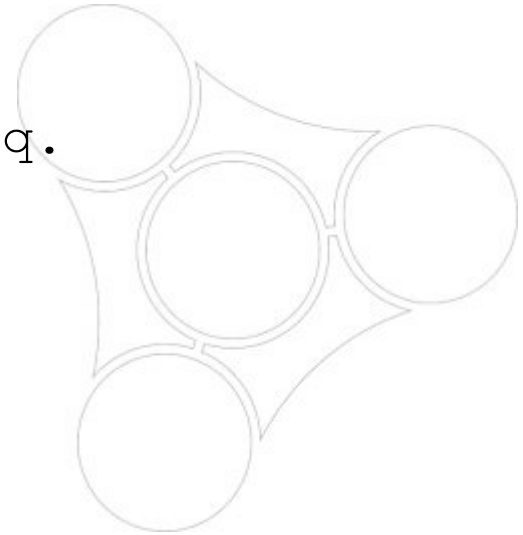
Soft type

- Make the trail explicit

```
Fixpoint path T (e : rel T) x (p : list T) :=
 if p isn't y :: q then true else
 e x y && path e y q.
```

- Use list operations

```
Lemma cat path T (e : rel T) x p q :
  path e x (p ++ q)
   = path e x p && path e (last x p) q.
```



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 $x \in G$ & $x \in H \rightarrow$ 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.

```
Inductive group_of := Group gval & group_set gval.
```

Subgroup theory

group H	$\{1\} \cup H^2 = H$
normaliser $N(H)$	$\{x \mid Hx = xH \text{ (or } H^x = H)\}$
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 $xy, 1, x^{-1}$

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 }.
```

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

```
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.
Infix "/" := quotient.
```

Group characters

Definition:

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

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

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.

More theory tweaks

- p-group G means G = 1 if p is not a prime
- character theory uses algebraic numbers
- complex numbers are ordered horizontally
- characters are allowed to be 0
- representation theory uses plain (matrix) linear algebra
- 1 / 0 = 0

Textbook to digital formal text

Theorem 14.7. Suppose $M \in \mathscr{M}_{\mathscr{P}}$ and K is a Hall $\kappa(M)$-subgroup of M. Let $K^* = C_{M_\sigma}(K)$, $k = |K|$, $k^* = |K^*|$, $Z = K \times K^*$, and $\hat{Z} = Z - (K \cup K^*)$. Then, for some other $M^* \in \mathscr{M}_{\mathscr{P}}$ not conjugate to M,

(a) $\mathscr{M}(C_G(X)) = \{M^*\}$ for every $X \in \mathcal{E}^1(K)$,

(b) K^* is a Hall $\kappa(M^*)$-subgroup of M^* and a Hall $\sigma(M)$-subgroup of M^*,

(c) $K = C_{M^*_\sigma}(K^*)$ and $\kappa(M) = \tau_1(M)$,

(d) Z is cyclic and for every $x \in K^{\#}$ and $y \in K^{*\#}$, $M \cap M^* = Z = C_M(x) = C_{M^*}(y) = C_G(xy)$,

(e) $\hat{Z}$ is a TI-subset of G with $N_G(\hat{Z}) = Z$, $\hat{Z} \cap M^g$ empty for all $g \in G - M$, and

$$|\mathscr{C}_G(\hat{Z})| = \left(1 - \frac{1}{k} - \frac{1}{k^*} + \frac{1}{kk^*}\right)|G| > \frac{1}{2}|G|,$$

(f) M or M^* lies in $\mathscr{M}_{\mathscr{P}_2}$ and, accordingly, K or K^* has prime order,

(g) every $H \in \mathscr{M}_{\mathscr{P}}$ is conjugate to M or M^* in G, and

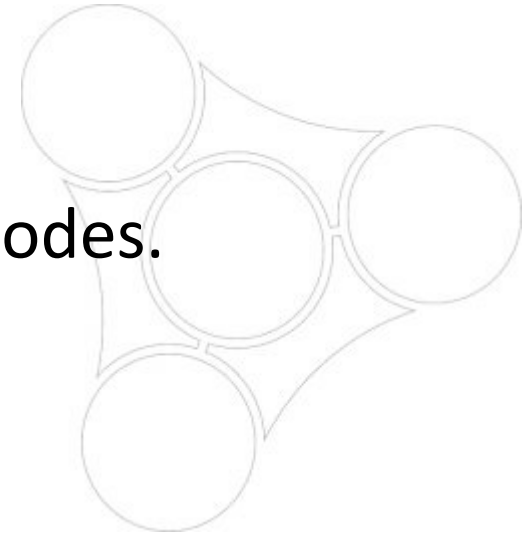
(h) M' is a complement of K in M.

Textbook to digital formal text

```
Theorem Ptype_embedding : forall M K,
    M \in 'M_'P -> \kappa(M).-Hall(M) K ->
 exists2 Mstar, Mstar \in 'M_'P /\ gval Mstar \notin M :^: G
 & let Kstar := 'C_(M`_\sigma)(K) in
   let Z := K <*> Kstar in let Zhat := Z :\: (K :|: Kstar) in
 [/\ (*a*) {in 'E^1(K), forall X, 'M('C(X)) = [set Mstar]},
     (*b*) \kappa(Mstar).-Hall(Mstar) Kstar /\ \sigma(M).-Hall(Mstar) Kstar,
     (*c*) 'C_(Mstar`_\sigma)(Kstar) = K /\ \kappa(M) =i \tau1(M),
     (*d*) [/\ cyclic Z, M :&: Mstar = Z,
           {in K^#, forall x, 'C_M[x] = Z}, {in Kstar^#, forall y, 'C_Mstar[y] = Z}
         & {in K^# & Kstar^#, forall x y, 'C[x * y] = Z}]
& [/\ (*e*) [/\ trivIset (Zhat :^: G), 'N(Zhat) = Z,
           {in ~: M, forall g, [disjoint Zhat & M :^ g]}
         & (#|G|%:R / 2%:R < #|class_support Zhat G|%:R :> qnum)%R ],
     (*f*) M \in 'M_'P2 /\ prime #|K| \/ Mstar \in 'M_'P2 /\ prime #|Kstar|,
     (*g*) {in 'M_'P, forall H, gval H \in M :^: G :|: Mstar :^: G}
   & (*h*) M^`(1) >< K = M]].
```

Overloading

- Ad hoc: substitute symbols/constants while reading.
  - Precise meaning
  - No uniformity
- Parametric: interpret symbol using context/parameter.
  - Uniform
  - No meaning
- Mathematical conventions mix these modes.
- Isomorphism theorem $G/K/H/K \approx G / H$



Parametric overloading

```
Structure finGroupType := FinGroupType {
  gSort : Type;
  mulg : gSort → gSort →gSort; ...}

Infix "*" := (mulg _).
Variables (T : finGroupType)   (x y : gSort T).
```

- Generic product:

```
x * y

   mulg  T  x y.

gSort  T    ≡    gSort T
```

Ad hoc overloading

```
Definition coset_gType H := FinGroupType (coset_of H) ....
    gSort (coset_gType H) ≡ coset_of H
Canonical coset_gType.
```

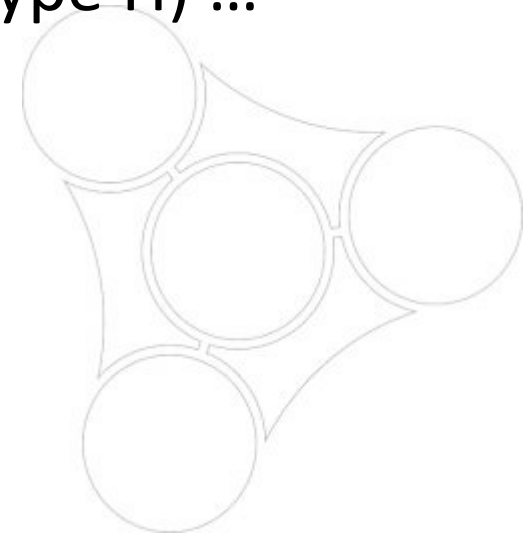
- Specific product

  coset H x * coset H y ≡ mulg (coset_gType H) …

  gSort $?_1$ ≡ coset_of H

  ↓

  fun gT => let: gType T _ … := gT in T



Algebraic notation

$$\sum_{i<n} a_i X^i$$

$$\sum_{d \mid n} \Phi(n/d)\, m^d$$

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

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

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

$$\bigoplus_{V_i \approx W} V_i$$

```
\bigcap_{H < G \atop H {\rm\ maximal}} H
```

```
Definition determinant n (A : 'M_n) : R :=
  \sum_(s : 'S_n) (-1) ^+ s * \prod_i A i (s i).
```

Notation at work

$$
\begin{aligned}
|AB| &= \sum_{\sigma \in S_n} (-1)^{\sigma} \prod_i \left(\sum_j A_{i,j} B_{j,i\sigma} \right) \\
&= \sum_{\rho} \prod_i A_{i,i\rho} \sum_{\sigma \in S_n} (-1)^{\sigma} \prod_i B_{i\rho,i\sigma} \\
&= \sum_{\rho \in S_n} \prod_i A_{i,i\rho} \sum_{\sigma \in S_n} (-1)^{\sigma} \prod_j B_{j,j\rho^{-1}\sigma} && i = j\rho^{-1} \\
&\quad + \sum_{\rho \notin S_n} \prod_i A_{i,i\rho} \sum_{\sigma \in S_n} (-1)^{\sigma} \prod_i B_{i\rho,i\sigma} \\
&= \left(\sum_{\rho \in S_n} (-1)^{\rho} \prod_i A_{i,i\rho} \right) \left(\sum_{\tau \in S_n} (-1)^{\tau} \prod_j B_{j,j\tau} \right) && \sigma = \rho\tau \\
&\quad + \sum_{\rho \notin S_n} \prod_i A_{i,i\rho} \left| (B_{i\rho,j}) \right| \\
&= |A| \, |B|
\end{aligned}
$$

Notation at work

```
\begin{eqnarray*}
 \det AB & = & \sum_{\sigma \in S_n} (-1)^\sigma \prod_i
                    \left(\sum_j A_{ij} B_{j\sigma(i)}\right)\\
        & = &  \sum_{\phi :\; [1, n] \rightarrow [1, n]}
                  \sum_{\sigma \in S_n}
                 (-1)^\sigma \prod_i A_{i\phi(i)} B_{\phi(i)\sigma(i)}\\
        & = &  \sum_{\phi \notin S_n} \sum_{\sigma \in S_n}
                   (-1)^\sigma \prod_i A_{i\phi(i)} B_{\phi(i)\sigma(i)}
                    + \sum_{\phi \in S_n} \sum_{\sigma \in S_n}
                     (-1)^\sigma \prod_i A_{i\phi(i)} B_{\phi(i)\sigma(i)}\\
        & = &  \sum_{\phi \notin S_n} \left(\prod_i A_{i\phi(i)}\right)
                    \sum_{\sigma \in S_n}
                      (-1)^\sigma \prod_i B_{\phi(i)\sigma(i)}\\
        &   &  {} + \sum_{\phi \in S_n}
                          (-1)^\phi \left(\prod_i A_{i\phi(i)}\right)
                        \sum_{\sigma \in S_n}
                        (-1)^{\phi^{-1}\sigma}
                                \prod_k B_{k\sigma(\phi^{-1}(k))}\\
        & = &  \sum_{\phi \notin S_n} \left(\prod_i A_{i\phi(i)}\right)
                   \det \left(B_{\phi(i)j}\right)_{ij}
             + (\det A)\sum_{\tau \in S_n}(-1)^\tau \prod_k B_{k\tau(k)}\\
              & = & 0 + (\det A)(\det B)\\
\end{eqnarray*}
```

The first step swaps the iterated product of the Leibnitz formula with the sum in the general term of the matrix product, generating a sum over all functions from indices to indices. This is split into a sum over non-injective functions and a sum over permutations. The former is rearranged into a weighted sum of determinants of matrices with repeated rows, while the latter is reindexed, using the group properties of permutations, to become the desired product of determinants.

Notation at work

```
\begin{eqnarray*}
 \det AB f = f \sum_{\sigma \in S_n} (-1)^\sigma \prod_i
```

```
Lemma determinantM n (A B : M_(n)) : \det (A *m B) = \det A * \det B.
Proof.
rewrite big_distrl /=.
pose F := {ffun 'I_n -> 'I_n}; pose AB s i j := A i j * B j (s i).
transitivity (\sum_(f : F) \sum_(s : 'S_n) (-1) ^+ s * \prod_i AB s i (f i)).
  rewrite exchange_big; apply: eq_bigr => /= s _; rewrite -big_distrr /=.
  congr (_ * _); rewrite -(bigA_distr_bigA (AB s)) /=.
  by apply: eq_bigr => x _; rewrite mxE.
rewrite (bigID (fun f : F => injectiveb f)) /= addrC big1 ?add0r => [|f Uf].
  rewrite (reindex (@pval _)) /=; last first.
    pose in_Sn f : 'S_n := insubd 1%g f.
    by exists in_Sn => /= f Uf; first apply: val_inj; exact: insubdK.
  apply: eq_big => /= [s | s _]; rewrite ?(valP s) // big_distrr /=.
  rewrite (reindex_inj (mulgI s)); apply: eq_bigr => t _ /=.
  rewrite big_split /= mulrA mulrCA mulrA mulrCA mulrA.
  rewrite -signr_addb odd_permM !pvalE; congr (_ * _); symmetry.
  by rewrite (reindex_inj (@perm_inj _ s)); apply: eq_bigr => i; rewrite permM.
transitivity (\det (\matrix_(i, j) B (f i) j) * \prod_i A i (f i)).
  rewrite mulrC big_distrr /=; apply: eq_bigr => s _.
  rewrite mulrCA big_split //=; congr (_ * (_ * _)).
  by apply: eq_bigr => x _; rewrite mxE.
case/injectivePn: Uf => i1 [i2 Di12 Ef12].
by rewrite (alternate_determinant Di12) ?simp //= => j; rewrite !mxE Ef12.
Qed.
```

into a weighted sum of determinants of matrices with repeated rows,
while the latter is reindexed, using the group properties
of permutations, to become the desired product of determinants.

Under the bigop hood

```
Definition reducebig R I op nil r P (F : I -> R) : R :=
  foldr (fun i x => if P i then op (F i) x else x) nil r.
```

- Present the options

```
Notation "\big[ op / nil ]_ ( i <- r | P ) F" :=
  (reducebig op nil r (fun i => P) (fun i => F)).
```

- Hide or fill the options

```
Notation "\big[ op / nil ]_( i <- r ) F" :=
  (\big[op/nil]_(i <- r | true) F).
Notation "\sum_ ( i <- r ) F" :=
  (\big[addn/0%nat]_(i <- r) F) : nat_scope.
```

- Generic filling

```
Notation "\big[ op / nil ]_ i F" :=
  (\big[op/nil]_(i <- fintype.enum _) F).
```

Generic Lemmas

- Pull, split, reindex, exchange …

```
Lemma bigD1 (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 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 (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 (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).
```

Linear operator interface

- 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.
```

General linear operators

- Encapsulate f (λv) = λ^{σ}(f v)

```
Module Linear….
Variables (R : ringType) (U : lmodType R) (V : zmodType).
Variable (s : R -> V -> V).

Definition mixin_of (f : U -> V)  :=
  forall a, {morph f : u / a *: u >-> s a u}.

Record class_of f : Prop := Class {base : additive f; mixin : mixin_of f}.
Structure map := Pack {apply :> Type; class : class_of apply}.
…

 (* horner_morph mulCx_nu P := (map nu P).[x] *)
Fact …:  scalable_for (nu \; *%R) (horner_morph mulCx_nu).
```

General linearity

- Rewrite f (λv) = λ^{σ}(f v) in *both* directions

```
Variables (R : ringType) (U : lmodType R) (V : zmodType).
Variables (s : R -> V -> V) (S : ringType) (h : S -> V -> V).
Variable h_law : Scale.law h.

Lemma linearZ c a (h_c := Scale.op h_law c)
                  (f : Linear.map_for U s a h_c) u :
  f (a *: u) = h_c (Linear.wrap f u).
```

Direct sums

In math:

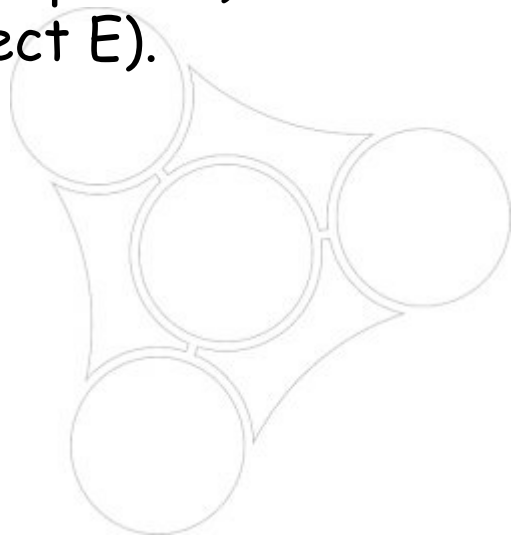
$S = A + \sum_i B_i$ is direct

iff rank S = rank A + $\sum_i$ rank B_i

In Coq:

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

This is generic in the *shape* of S



Circular inequalities

```
Lemma leqif_trans : forall m1 m2 m3 c1 c2,
  m1 <= m2 ?= iff c1 -> m2 <= m3 ?= iff c2 -> m1 <= m3 ?= iff c1 && c2.

Lemma leqif_add : forall m1 n1 c1 m2 n2 c2,
    m1 <= n1 ?= iff c1 -> m2 <= n2 ?= iff c2 ->
  m1 + m2 <= n1 + n2 ?= iff c1 && c2.

Lemma mxrank_sum_leqif : forall m n (S : mxsum_expr m n),
  \rank (unwrap S) <= unwrap (mxsum_rank S) ?= iff mxdirect (unwrap S).
```

```
have Bfree_if: forall ZxH, ZxH \in clPqH^# ->
                  \rank <<B (b ZxH)>> <= #|ZxH| ?= iff row_free (B (b ZxH)) by...
have B1_if: \rank <<B (b 1%g)>> <= 1 ?= iff (<<B (b 1%g)>> == mxvec 1%:M)%MS by ...
have rankEP: \rank (1%:M : 'A[F]_q) = (\sum_(ZxH \in clPqH) #|ZxH|)%N by ...
have cl1: 1%g \in clPqH by ...
have{B1_if Bfree_if}:= leqif_add B1_if (leqif_sum Bfree_if).
case/(leqif_trans (mxrank_sum_leqif _)) => _ /=.
rewrite -{1}(big_setD1 _ cl1) sumB {}rankEP (big_setD1 1%g) // cards1 eqxx.
move/esym; case/and3P=> dxB; move/eqmxP=> defB1; move/forall_inP=> /= Bfree.
```

Circular inequalities

```
Bfree_if : forall ZxH : {set coset_of Z}, ZxH \in clPqH^# ->
          \rank <<B (b ZxH)>> <= #|ZxH| ?= iff row_free (B (b ZxH))
 B1_if : \rank <<B (b 1%g)>> <= 1 ?= iff (<<B (b 1%g)>> == mxvec 1%:M)%MS
 rankEP : \rank 1%:M = (\sum_(ZxH \in clPqH) #|ZxH|)%N
 cl1 : 1%g \in clPqH
 ============================
 ...
```

```
have Bfree_if: forall ZxH, ZxH \in clPqH^# ->
                    \rank <<B (b ZxH)>> <= #|ZxH| ?= iff row_free (B (b ZxH)) by...
have B1_if: \rank <<B (b 1%g)>> <= 1 ?= iff (<<B (b 1%g)>> == mxvec 1%:M)%MS by ...
have rankEP: \rank (1%:M : 'A[F]_q) = (\sum_(ZxH \in clPqH) #|ZxH|)%N by ...
have cl1: 1%g \in clPqH by ...
have{B1_if Bfree_if}:= leqif_add B1_if (leqif_sum Bfree_if).
case/(leqif_trans (mxrank_sum_leqif _)) => _ /=.
rewrite -{1}(big_setD1 _ cl1) sumB {}rankEP (big_setD1 1%g) // cards1 eqxx.
move/esym; case/and3P=> dxB; move/eqmxP=> defB1; move/forall_inP=> /= Bfree.
```

Circular inequalities

```
 rankEP : \rank 1%:M = (\sum_(ZxH \in clPqH) #|ZxH|)%N
  cl1 : 1%g \in clPqH
  ============================
\rank <<B (b 1%g)>> + \sum_(i \in clPqH^#) \rank <<B (b i)>>
    <= 1 + \sum_(i \in clPqH^#) #|i|
    ?= iff (<<B (b 1%g)>> == mxvec 1%:M)%MS &&
          (forallb i, (i \in clPqH^#) ==> row_free (B (b i))) ->

...
```

```
have Bfree_if: forall ZxH, ZxH \in clPqH^# ->
                    \rank <<B (b ZxH)>> <= #|ZxH| ?= iff row_free (B (b ZxH)) by...
have B1_if: \rank <<B (b 1%g)>> <= 1 ?= iff (<<B (b 1%g)>> == mxvec 1%:M)%MS by ...
have rankEP: \rank (1%:M : 'A[F]_q) = (\sum_(ZxH \in clPqH) #|ZxH|)%N by ...
have cl1: 1%g \in clPqH by ...
have{B1_if Bfree_if}:= leqif_add B1_if (leqif_sum Bfree_if).
case/(leqif_trans (mxrank_sum_leqif _)) => _ /=.
rewrite -{1}(big_setD1 _ cl1) sumB {}rankEP (big_setD1 1%g) // cards1 eqxx.
move/esym; case/and3P=> dxB; move/eqmxP=> defB1; move/forall_inP=> /= Bfree.
```

Circular inequalities

```
rankEP : \rank 1%:M = (\sum_(ZxH \in clPqH) #|ZxH|)%N
 cl1 : 1%g \in clPqH
 ============================
true =
  [&& mxdirect (<<B (b 1%g)>> + \sum_(i \in clPqH^#) <<B (b i)>>),
      (<<B (b 1%g)>> == mxvec 1%:M)%MS
    & forallb i, (i \in clPqH^#) ==> row_free (B (b i))] ->
...
```

```
have Bfree_if: forall ZxH, ZxH \in clPqH^# ->
                    \rank <<B (b ZxH)>> <= #|ZxH| ?= iff row_free (B (b ZxH)) by...
have B1_if: \rank <<B (b 1%g)>> <= 1 ?= iff (<<B (b 1%g)>> == mxvec 1%:M)%MS by ...
have rankEP: \rank (1%:M : 'A[F]_q) = (\sum_(ZxH \in clPqH) #|ZxH|)%N by ...
have cl1: 1%g \in clPqH by ...
have{B1_if Bfree_if}:= leqif_add B1_if (leqif_sum Bfree_if).
case/(leqif_trans (mxrank_sum_leqif _)) => _ /=.
rewrite -{1}(big_setD1 _ cl1) sumB {}rankEP (big_setD1 1%g) // cards1 eqxx.
move/esym; case/and3P=> dxB; move/eqmxP=> defB1; move/forall_inP=> /= Bfree.
```

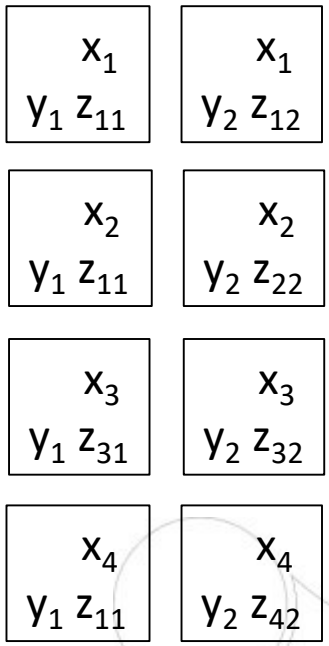
Circular inequalities

```
rankEP : \rank 1%:M = (\sum_(ZxH \in clPqH) #|ZxH|)%N
 cl1 : 1%g \in clPqH
 dxB : mxdirect (<<B (b 1%g)>> + \sum_(i \in clPqH^#) <<B (b i)>>)
 defB1 : (<<B (b 1%g)>> :=: mxvec 1%:M)%MS
 Bfree : forall x : {set coset_of Z}, x \in clPqH^# -> row_free (B (b x))
 ============================

...
```

```
have Bfree_if: forall ZxH, ZxH \in clPqH^# ->
                    \rank <<B (b ZxH)>> <= #|ZxH| ?= iff row_free (B (b ZxH)) by...
have B1_if: \rank <<B (b 1%g)>> <= 1 ?= iff (<<B (b 1%g)>> == mxvec 1%:M)%MS by ...
have rankEP: \rank (1%:M : 'A[F]_q) = (\sum_(ZxH \in clPqH) #|ZxH|)%N by ...
have cl1: 1%g \in clPqH by ...
have{B1_if Bfree_if}:= leqif_add B1_if (leqif_sum Bfree_if).
case/(leqif_trans (mxrank_sum_leqif _)) => _ /=.
rewrite -{1}(big_setD1 _ cl1) sumB {}rankEP (big_setD1 1%g) // cards1 eqxx.
move/esym; case/and3P=> dxB; move/eqmxP=> defB1; move/forall_inP=> /= Bfree.
```

An integer norm problem

- Infer integral vectors from dot product
- Naïve SMT fails
- Naïve SAT bit-blast fails
- Tailored SAT encoding longish



Modeling vectors

```
(* Clause left-hand side, a reference to a value of beta; in the reference    *)
(* model m, (i, j) stands for beta_ (inord i.+1) (inord j.+1).                 *)
Definition ref := (nat * nat)%type.
Implicit Type ij : ref.
Definition Ref b_ij : ref := edivn (b_ij - 11) 10. (* Ref 21 = (1, 0). *)
Notation "''b' ij" := (Ref ij) (at level 0, ij at level 0, format "''b' ij").
Notation b11 := 'b11. Notation b12 := 'b12…

Definition bbox := (nat * nat)%type. (* bounding box for refs. *)
Definition sub_bbox bb1 bb2 := (bb1.1 <= bb2.1)%N && (bb1.2 <= bb2.2)%N.
Definition wf_ref bb := [pred ij : ref | (ij.1 < bb.1)%N && (ij.2 < bb.2)%N].
Definition dot_ref ij1 ij2 := ((ij1.1 == ij2.1).+1 * (ij1.2 == ij2.2).+1 - 1)%N.

Lemma bbox_refl bb : sub_bbox bb bb. Proof. exact/andP. Qed.

(* Clause right-hand side litteral, denoting the projection of the left-hand  *)
(* side on an irreducible character of G: in a valid model m, (k, v) stands   *)
(* for the component m`_k *~ v = (model_xi m)`_k, and for the projection       *)
(* constraint '[m i j, m`_k] == v%:~R.                                        *)
Definition lit := (nat * int)%type. (* +x1 = (0,1) ~x2 = (1,0) -x3 = (2, -1)  *)
Definition Lit k1 v : lit := if (0 + k1)%N is k.+1 then (k, v) else (k1, v).
Notation "+x k" := (Lit k 1) (at level 0, k at level 0, format "+x k").
Notation "-x k" := (Lit k (-1)) (at level 0, k at level 0, format "-x k").
Notation "~x k" := (Lit k 0) (at level 0, k at level 0, format "~x k").
```

Modeling vectors

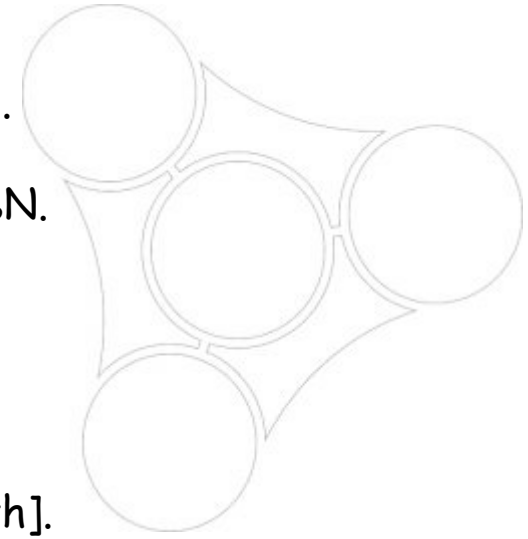
```
(* Clause left-hand side, a reference to a value of beta; in the reference    *)
(* model m, (i, j) stands for beta_ (inord i.+1) (inord j.+1).                 *)
Definition ref := (nat * nat)%type.
Implicit Type ij : ref.
Definition Ref b_ij : ref := edivn (b_ij - 11) 10. (* Ref 21 = (1, 0). *)
Notation "''b' ij" := (Ref ij) (at level 0, ij at level 0, format "''b' ij").
Notation b11 := 'b11. Notation b12 := 'b12…

Definition bbox := (nat * nat)%type. (* bounding box for refs. *)
Definition sub_bbox bb1 bb2 := (bb1.1 <= bb2.1)%N && (bb1.2 <= bb2.2)%N.
Definition wf_ref bb := [pred ij : ref | (ij.1 < bb.1)%N && (ij.2 < bb.2)%N].
Definition dot_ref ij1 ij2 := ((ij1.1 == ij2.1).+1 * (ij1.2 == ij2.2).+1 - 1)%N.

Lemma bbox_refl bb : sub_bbox bb bb. Proof. exact/andP. Qed.

(* Clause right-hand side litteral, denoting the projection of the left-hand  *)
(* side on an irreducible character of G: in a valid model m, (k, v) stands   *)
(* for the component m`_k *~ v = (model_xi m)`_k, and for the projection       *)
(* constraint '[m i j, m`_k] == v%:~R.                                        *)
Definition lit := (nat * int)%type. (* +x1 = (0,1) ~x2 = (1,0) -x3 = (2, -1)  *)
Definition Lit k1 v : lit := if (0 + k1)%N is k.+1 then (k, v) else (k1, v).
Notation "+x k" := (Lit k 1) (at level 0, k at level 0, format "+x k").
Notation "-x k" := (Lit k (-1)) (at level 0, k at level 0, format "-x k").
Notation "~x k" := (Lit k 0) (at level 0, k at level 0, format "~x k").
```

Modeling vectors

```
(* Clause left-hand side, a reference to a value of beta; in the reference    *)
(* model m, (i, j) stands for beta_ (inord i.+1) (inord j.+1).                 *)
Definition ref := (nat * nat)%type.
Implicit Type ij : ref.
Definition Ref b_ij : ref := edivn (b_ij - 11) 10. (* Ref 21 = (1, 0). *)
Notation "''b' ij" := (Ref ij) (at level 0, ij at level 0, format "''b' ij").
Notation b11 := 'b11. Notation b12 := 'b12…

Definition bbox := (nat * nat)%type. (* bounding box for refs. *)
Definition sub_bbox bb1 bb2 := (bb1.1 <= bb2.1)%N && (bb1.2 <= bb2.2)%N.
Definition wf_ref bb := [pred ij : ref | (ij.1 < bb.1)%N && (ij.2 < bb.2)%N].
Definition dot_ref ij1 ij2 := ((ij1.1 == ij2.1).+1 * (ij1.2 == ij2.2).+1 - 1)%N.

Lemma bbox_refl bb : sub_bbox bb bb. Proof. exact/andP. Qed.

(* Clause right-hand side litteral, denoting the projection of the left-hand  *)
(* side on an irreducible character of G: in a valid model m, (k, v) stands   *)
(* for the component m`_k *~ v = (model_xi m)`_k, and for the projection       *)
(* constraint '[m i j, m`_k] == v%:~R.                                        *)
Definition lit := (nat * int)%type. (* +x1 = (0,1) ~x2 = (1,0) -x3 = (2, -1)  *)
Definition Lit k1 v : lit := if (0 + k1)%N is k.+1 then (k, v) else (k1, v).
Notation "+x k" := (Lit k 1) (at level 0, k at level 0, format "+x k").
Notation "-x k" := (Lit k (-1)) (at level 0, k at level 0, format "-x k").
Notation "~x k" := (Lit k 0) (at level 0, k at level 0, format "~x k").
```

Model-checking vectors

```
Definition clause := (ref * seq lit)%type.
Definition Clause ij kvs : clause := (ij, kvs).
Notation "& kv1 , .. , kvn 'in' ij" := (Clause ij (AndLit .. (AndLit nil kv1) .. kvn)) ….

Definition theory := seq clause.
Definition AddClause th cl : theory := cl :: th.
Notation "|= cl1 .. cln" := (AddClause .. (AddClause nil cl1) .. cln)…

Fixpoint set_cl cl2 th : wrapped theory :=
  if th is cl :: th1 then
    let: Wrap th2 := set_cl cl2 th1 in
    if cl.1 == cl2.1 then Wrap (AddClause th2 cl2) else Wrap (AddClause th2 cl)
  else Wrap nil.

Definition ext_cl th cl k v :=
  let: (ij, kvs) := cl in set_cl (Clause ij (AndLit kvs (Lit k.+1 v))) th.

Definition wf_ext_cl cl k rk := (k \notin unzip1 cl.2) && (k < rk)%N.

Lemma ext_clP cl1 th k v (cl1k := (cl1.1, (k, v) :: cl1.2)) :
    cl1 \in th ->
  exists2 th1, ext_cl th cl1 k v = Wrap th1
       & cl1k \in th1
       /\ th1 =i [pred cl | if cl.1 == cl1.1 then cl == cl1k else cl \in th].
```



Character reflection

Assume that (3.5) has been shown. Set $\omega_{ij}^{\sigma} = \chi_{ij}$ and extend σ to $CF(W)$ by linearity. Then (a) and (b) of Theorem (3.2) are established, and assertions (c) and (d) of Theorem (3.2) follow from (1.3).

Proof of (3.5).

(3.5.1) *Let* $\beta_{ij} = \mathrm{Ind}_W^G \alpha_{ij} - 1_G$ $(1 \le i < w_1, 1 \le j < w_2)$. *Then* $(\beta_{ij}, 1_G) = 0$ *and* $\|\beta_{ij}\|^2 = 3$ *for all* i, j *while* $(\beta_{ij}, \beta_{ij'}) = (\beta_{ij}, \beta_{i'j}) = 1$ *and* $(\beta_{ij}, \beta_{i'j'}) = 0$ *for* $i \neq i'$, $j \neq j'$.

Proof. That $(\mathrm{Ind}_W^G \alpha_{ij}, 1_G) = (\alpha_{ij}, 1_W) = 1$ follows from Frobenius reciprocity, and so $(\beta_{ij}, 1_G) = 0$. The other relations follow from the fact that Ind_W^G is an isometry on $CF(W, V)$. □

Let $1 \le i < w_1$, $1 \le j < w_2$. By (3.5.1) and the fact that $\beta_{ij} \in \mathbb{Z}[\mathrm{Irr}(G)]$, we see that $\beta_{ij} = \sum_{\chi \in A_{ij}} \chi$, where A_{ij} is a set of three pairwise orthogonal elements of $\pm(\mathrm{Irr}(G) - \{1_G\})$.

(3.5.2) *We have* $|A_{11} \cap A_{12}| = 1$ *and* $A_{11} \cap (-A_{12}) = \emptyset$.

Proof. Let $A_{11} = \{\chi_1, \chi_2, \chi_3\}$ and $a_i = (\beta_{12}, \chi_i)$ for $i = 1, 2, 3$. Then $(\beta_{12}, \beta_{11}) = a_1 + a_2 + a_3 = 1$ and $a_i \in \{0, 1, -1\}$. The numbers a_i are thus either $1, 0, 0$, or $1, 1, -1$. In the second case, we may assume that $\beta_{12} = \chi_1 + \chi_2 - \chi_3$ whence $2\chi_3 = \beta_{11} - \beta_{12} = \mathrm{Ind}_W^G(\alpha_{11} - \alpha_{12})$ vanishes on $1 \in G$, which is a contradiction. □

Lemma (3.5.2) clearly holds with A_{ij} and $A_{i'j'}$ in place of A_{11} and A_{12} if $i = i'$ and $j \neq j'$ or if $i \neq i'$ and $j = j'$. We refer to this lemma for A_{ij} and $A_{i'j'}$ as $L(ij, i'j')$. We also refer to the statement $(\beta_{ij}, \beta_{i'j'}) = 0$ for $i \neq i'$ and $j \neq j'$ as $O(ij, i'j')$.

By Hypothesis (3.1), $\sup(w_1, w_2) \ge 5$. By the symmetry between w_1 and w_2, we will assume

(3.5.3) $w_1 \ge 5$.

In the proof which follows, the functions χ_i and χ_{ij} are pairwise orthogonal elements of $\pm(\mathrm{Irr}(G) - \{1_G\})$.

(3.5.4) $|\bigcap_{1 \le i < w_1} A_{i1}| = 1$.

Proof. Suppose that (3.5.4) is false. By (3.5.2), we can then write, for some choice of indices $i = 1, 2, 3$,

$$\begin{aligned} \beta_{11} &= \chi_1 + \chi_2 + \chi_3, \\ \beta_{21} &= \chi_1 + \chi_4 + \chi_5, \\ \beta_{31} &= \chi_2 + \chi_4 + \chi_6. \end{aligned}$$

We consider two cases:

Case I. There are indices i and i' such that $1 \le i < i' \le 3$ and $A_{i1} \cap A_{i'1} \cap A_{41} \neq \emptyset$.

Case II. For $1 \le i < i' < i'' \le 4$, $A_{i1} \cap A_{i'1} \cap A_{i''1} = \emptyset$.

Suppose that Case I holds. Up to choice of notation, we have, by (3.5.2),

$$\beta_{41} = \chi_1 + \chi_6 + \chi_7.$$

(3.5.4.1)

If $\chi_1 \in A_{12}$, *then* $\beta_{12} = \chi_1 - \chi_5 - \chi_7$.

If $\chi_3 \in A_{22}$, *then* $\beta_{22} = \chi_1 - \chi_3 - \chi_7$.

If $\chi_3 \in A_{42}$, *then* $\beta_{42} = \chi_1 - \chi_3 - \chi_5$.

Proof. Suppose that $\chi_1 \in A_{12}$. By $O(12, 21)$, $-\chi_4$ or $-\chi_5 \in A_{12}$. Suppose that $-\chi_4 \in A_{12}$. Then it follows from $O(12, 31)$ and $L(12, 11)$ that $\chi_6 \in A_{12}$, which contradicts $O(12, 41)$. Thus $-\chi_5 \in A_{12}$. Similarly, we see that $-\chi_7 \in A_{12}$ by interchanging the roles of β_{21} and β_{41}.

The other two assertions follow from the symmetry between β_{11}, β_{21} and β_{41}. □

(3.5.4.2) *We may assume that* $\beta_{32} = \chi_2 - \chi_3 + \chi_8$.

Proof. By the symmetry between the functions β_{11}, β_{21} and β_{41}, we may assume that $A_{32} \cap A_{31} = \{\chi_2\}$. By $O(32, 11)$, $-\chi_1$ or $-\chi_3 \in A_{32}$. Suppose that $-\chi_1 \in A_{32}$. Then, by $O(32, 21)$ and $O(32, 41)$, the third element of A_{32} is in $A_{21} \cap A_{41}$, which is a contradiction. Thus $-\chi_3 \in A_{32}$ and the third element of A_{32} cannot be one of the functions $\pm\chi_i$, $i \le 7$. □

(3.5.4.3) *We may assume that* $\beta_{12} = \chi_2 - \chi_4 + \chi_8$.

Proof. By (3.5.4.1), (3.5.4.2) and $L(12, 32)$, χ_1 does not belong to A_{12}. By $L(12, 11)$, χ_2 or $\chi_3 \in A_{12}$. But, by $L(12, 32)$ and (3.5.4.2), $\chi_2 \notin A_{12}$. Thus $\chi_2 \in A_{12}$. By $O(12, 31)$, $-\chi_4$ or $-\chi_6 \in A_{12}$. By the symmetry between the functions β_{21} and β_{41}, we may assume that $-\chi_4 \in A_{12}$. Then $\chi_8 \in A_{12}$ by $O(12, 21)$. □

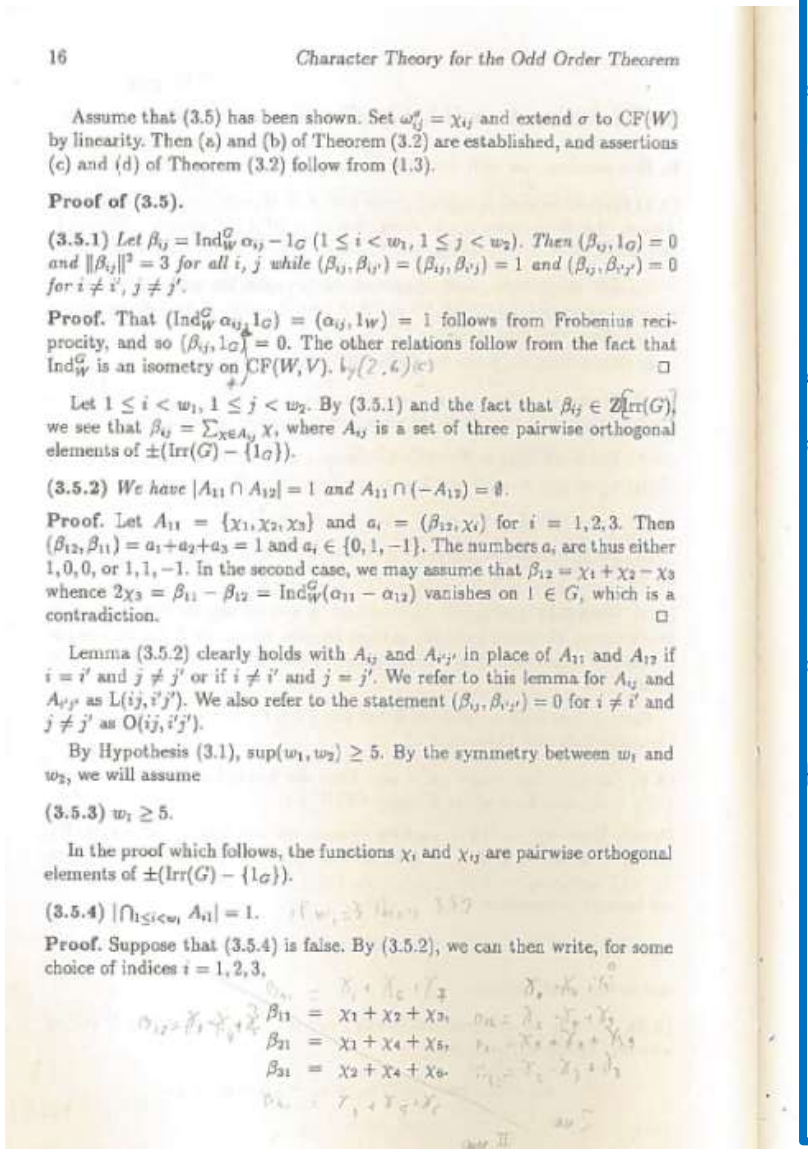
(3.5.4.4) $\beta_{22} = \chi_8 + \chi_8 + \chi_9$.

Proof. By (3.5.4.1), (3.5.4.3) and $L(22, 12)$, χ_1 does not belong to A_{22}. By $L(22, 21)$, χ_4 or $\chi_8 \in A_{22}$. Then $L(22, 12)$ shows that $\chi_8 \in A_{22}$. By $L(22, 12)$, $\chi_2 \notin A_{22}$ and so $L(22, 32)$ implies that $-\chi_3$ or $\chi_8 \in A_{22}$. But, by $O(22, 11)$, $-\chi_3 \notin A_{22}$ and so $\chi_8 \in A_{22}$. Thus the third element of A_{22} cannot be one of the functions $\pm\chi_i$, $i \le 8$. □

(3.5.4.5) *Case I is impossible.*

Proof. By (3.5.4.1) and $L(42, 12)$, χ_1 does not belong to A_{42}. Suppose that χ_2 or $-\chi_3 \in A_{42}$. By $O(42, 11)$, χ_2 and $-\chi_3 \in A_{42}$, which, with (3.5.4.2)

Character reflection



16 *Character Theory for the Odd Order Theorem*

Assume that (3.5) has been shown. Set $\omega_{ij}^{\sigma} = \chi_{ij}$ and extend σ to $\mathrm{CF}(W)$ by linearity. Then (a) and (b) of Theorem (3.2) are established, and assertions (c) and (d) of Theorem (3.2) follow from (1.3).

Proof of (3.5).

(3.5.1) *Let* $\beta_{ij} = \mathrm{Ind}_W^G \alpha_{ij} - 1_G$ $(1 \le i < w_1, 1 \le j < w_2)$. *Then* $(\beta_{ij}, 1_G) = 0$ *and* $\|\beta_{ij}\|^2 = 3$ *for all* i, j *while* $(\beta_{ij}, \beta_{ij'}) = (\beta_{ij}, \beta_{i'j}) = 1$ *and* $(\beta_{ij}, \beta_{i'j'}) = 0$ *for* $i \ne i'$, $j \ne j'$.

Proof. That $(\mathrm{Ind}_W^G \alpha_{ij}, 1_G) = (\alpha_{ij}, 1_W) = 1$ follows from Frobenius reciprocity, and so $(\beta_{ij}, 1_G) = 0$. The other relations follow from the fact that Ind_W^G is an isometry on $\mathrm{CF}(W, V)$. □

Let $1 \le i < w_1$, $1 \le j < w_2$. By (3.5.1) and the fact that $\beta_{ij} \in \mathbb{Z}[\mathrm{Irr}(G)]$, we see that $\beta_{ij} = \sum_{\chi \in A_{ij}} \chi$, where A_{ij} is a set of three pairwise orthogonal elements of $\pm(\mathrm{Irr}(G) - \{1_G\})$.

(3.5.2) *We have* $|A_{11} \cap A_{12}| = 1$ *and* $A_{11} \cap (-A_{12}) = \emptyset$.

Proof. Let $A_{11} = \{\chi_1, \chi_2, \chi_3\}$ and $a_i = (\beta_{12}, \chi_i)$ for $i = 1, 2, 3$. Then $(\beta_{12}, \beta_{11}) = a_1 + a_2 + a_3 = 1$ and $a_i \in \{0, 1, -1\}$. The numbers a_i are thus either $1, 0, 0$, or $1, 1, -1$. In the second case, we may assume that $\beta_{12} = \chi_1 + \chi_2 - \chi_3$ whence $2\chi_3 = \beta_{11} - \beta_{12} = \mathrm{Ind}_W^G(\alpha_{11} - \alpha_{12})$ vanishes on $1 \in G$, which is a contradiction. □

Lemma (3.5.2) clearly holds with A_{ij} and $A_{i'j'}$ in place of A_{11} and A_{12} if $i = i'$ and $j \ne j'$ or if $i \ne i'$ and $j = j'$. We refer to this lemma for A_{ij} and $A_{i'j'}$ as $\mathrm{L}(ij, i'j')$. We also refer to the statement $(\beta_{ij}, \beta_{i'j'}) = 0$ for $i \ne i'$ and $j \ne j'$ as $\mathrm{O}(ij, i'j')$.

By Hypothesis (3.1), $\sup(w_1, w_2) \ge 5$. By the symmetry between w_1 and w_2, we will assume

(3.5.3) $w_1 \ge 5$.

In the proof which follows, the functions χ_i and χ_{ij} are pairwise orthogonal elements of $\pm(\mathrm{Irr}(G) - \{1_G\})$.

(3.5.4) $|\bigcap_{1 \le i < w_1} A_{i1}| = 1$.

Proof. Suppose that (3.5.4) is false. By (3.5.2), we can then write, for some choice of indices $i = 1, 2, 3$,

$$\begin{aligned} \beta_{11} &= \chi_1 + \chi_2 + \chi_3, \\ \beta_{21} &= \chi_1 + \chi_4 + \chi_5, \\ \beta_{31} &= \chi_2 + \chi_4 + \chi_6. \end{aligned}$$

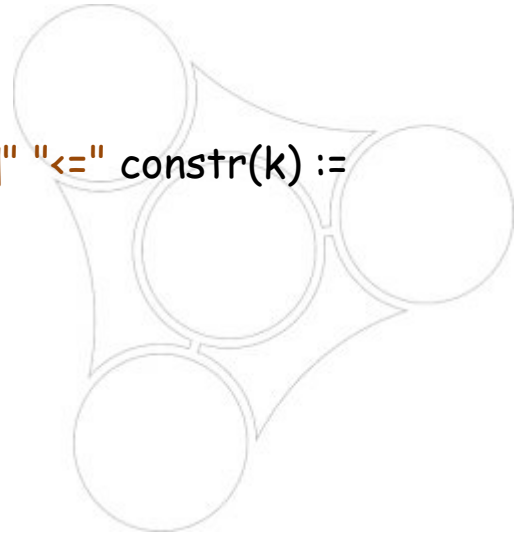
```
Let unsat Ii : unsat |= & x1 in b11 & x1 in b21 & ~x1 in b31.
Proof.
wlog Db11: (& b11 = x1 + x2 + x3) by do 2!fill b11.
wlog Db21: (& b21 = x1 + x4 + x5).
  by uhave ~x2, ~x3 in b21 as L(21, 11); do 2!fill b21; uexact Db21.
wlog Db31: (& b31 = x2 + x4 + x6).
  uwlog b31x2: x2 | ~x2 in b31 as L(31, 11).
    by uhave x3 in b31 as O(31, 11); symmetric to b31x2.
  uwlog b31x4: x4 | ~x4 in b31 as L(31, 21).
    by uhave x5 in b31 as O(31, 21); symmetric to b31x4.
  uhave ~x3 in b31 as O(31, 11); uhave ~x5 in b31 as L(31, 21).
  by fill b31; uexact Db31.
consider b41; uwlog b41x1: x1 | ~x1 in b41 as L(41, 11).
  uwlog Db41: (& b41 = x3 + x5 + x6) => [|{b41x1}].
    uhave ~x2 | x2 in b41 as L(41, 11); last symmetric to b41x1.
    uhave ~x4 | x4 in b41 as L(41, 21); last symmetric to b41x1.
    uhave x3 in b41 as O(41, 11); uhave x5 in b41 as O(41, 21).
    by uhave x6 in b41 as O(41, 31); uexact Db41.
  consider b12; uwlog b12x1: x1 | ~x1 in b12 as L(12, 11).
    uhave ~x2 | x2 in b12 as L(12, 11); last symmetric to b12x1.
    by uhave x3 in b12 as O(12, 11); symmetric to b12x1.
  uwlog b12x4: -x4 | ~x4 in b12 as O(12, 21).
    by uhave -x5 in b12 as O(12, 21); symmetric to b12x4.
  uhave ~x2, ~x3 in b12 as L(12, 11); uhave ~x5 in b12 as O(12, 21).
  by uhave x6 in b12 as O(12, 31); counter to O(12, 41).
wlog Db41: (& b41 = x1 + x6 + x7).
  uhave ~x2, ~x3 in b41 as L(41, 11); uhave ~x4, ~x5 in b41 as L(41, 21).
  by uhave x6 in b41 as O(41, 31); fill b41; uexact Db41.
consider b32; uwlog Db32: (& b32 = x6 - x7 + x8).
  uwlog b32x6: x6 | ~x6 in b32 as L(32, 31).
    uhave ~x2 | x2 in b32 as L(32, 31); last symmetric to b32x6.
    by uhave x4 in b32 as O(32, 31); symmetric to b32x6.
  uhave ~x2, ~x4 in b32 as L(32, 31).
  uhave -x7 | ~x7 in b32 as O(32, 41).
    uhave ~x1 in b32 as O(32, 41); uhave ~x3 in b32 as O(32, 11).
    by uhave ~x5 in b32 as O(32, 21); fill b32; uexact Db32.
  uhave -x1 in b32 as O(32, 41).
  by uhave x3 in b32 as O(32, 11); counter to O(32, 21).
consider b42; uwlog Db42: (& b42 = x6 - x4 + x5).
  uhave ~x6 | x6 in b42 as L(42, 41).
    uhave ~x7 | x7 in b42 as L(42, 41); last counter to O(42, 32).
    uhave x1 in b42 as O(42, 41); uhave x8 in b42 as O(42, 32).
    uhave ~x2 | ~x2 in b42 as O(42, 11); last counter to O(42, 21).
-(Unix)--  PFsection3.v   59% L1115 SVN-4447  (coq Scripting *3 SUBGOALS*
 b41x1 : unsat
           |= & b11 = x1 + x2 + x3
              & b21 = x1 + x4 + x5
              & b31 = x2 + x4 + x6
              & x1 in b41

 Db41 : unsat
          |= & b11 = x1 + x2 + x3
             & b21 = x1 + x4 + x5
             & b31 = x2 + x4 + x6
             & b41 = x3 + x5 + x6
 ============================
  unsat
    |= & b11 = x1 + x2 + x3
       & b21 = x1 + x4 + x5
       & b31 = x2 + x4 + x6
       & ~x1, ~x2 in b41
```

Reflection: DIY logic

```
Pcase L2_1: s[3] <= 6.
      Pcase: s[3] <= 5.
        Reducible.
      Pcase: s[4] <= 5.
        Reducible.
      Pcase: s[5] > 6.
        Hubcap $[1,2]<=0 $[3,4]<=(-6) $[5]<=(-4) $.
      Pcase: s[5] > 5.
        Hubcap $[1,2]<=0 $[3,4]<=(-7) $[5]<=(-3) $.
      Reducible.
    Pcase: s[5] <= 6.
      Similar to *L2_1[3].
    Pcase: s[4] > 5.
      Hubcap $[1,2]<=0 $[3,4]<=(-6) $[5]<=(-4) $.
    Hubcap $[1,2]<=0 $[3,4]<=(-5) $[5]<=(-5) $.
```

Domain-Specific Scripting

```
Tactic Notation "Presentation" ident(red) := apply: exclude_arity => /= red.

Ltac Reducible := by apply succeed_by_reducibility; last exact <: isT.

Export HubcapSyntax.

Ltac Hubcap hc := by apply (succeed_by_hubcap hc); last exact <: isT.

Tactic Notation "Similar" "to" "*" ident(lab) "[" constr(n) "]" :=
  apply (succeed_by_similarity n true lab); compute; do 2!split.
Tactic Notation "Similar" "to" ident(lab) "[" constr(n) "]" :=
  apply (succeed_by_similarity n false lab); compute; do 2!split.

Export SublocSyntax.

Tactic Notation "Pcase" ident(label) " : " constr(i) "[" constr(j) "]" "<=" constr(k) :=
  apply (succeed_by_split i j.-1 k true) => //= [|label].
```



Questions?

- Formalization libraries should provide more than a conventional mathematical framework.
- They need to cover the abundant implicit contents in mathematics, such as notation and proof conventions.
- This can be achieved through a more specific framework and specialized tools, but also with throwaway scaffolding.