Foothills and cathedrals: organising the libraries behind big proofs

Georges Gonthier
Inria

Some computer proof landmarks

1976 – The four color theorem *new*

1997 – The Robbins conjecture (Otter) *new*

Kepler Conjecture *new*

2002 – Compendium of continuous lattices (Mizar)

2005 – Four color theorem (Coq)

2012 – The Odd Order Theorem (Coq)

Homotopy Type Theory (Coq, Agda) *new*

2014 – Flyspeck (Hol Light)

2016 – Pythagorean Triple Conjecture (SAT) *new*

2020 – Perfectoid spaces (Lean)

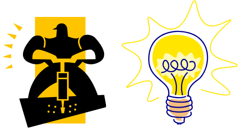
A personal journey

1994: The Caml Garbage Collector

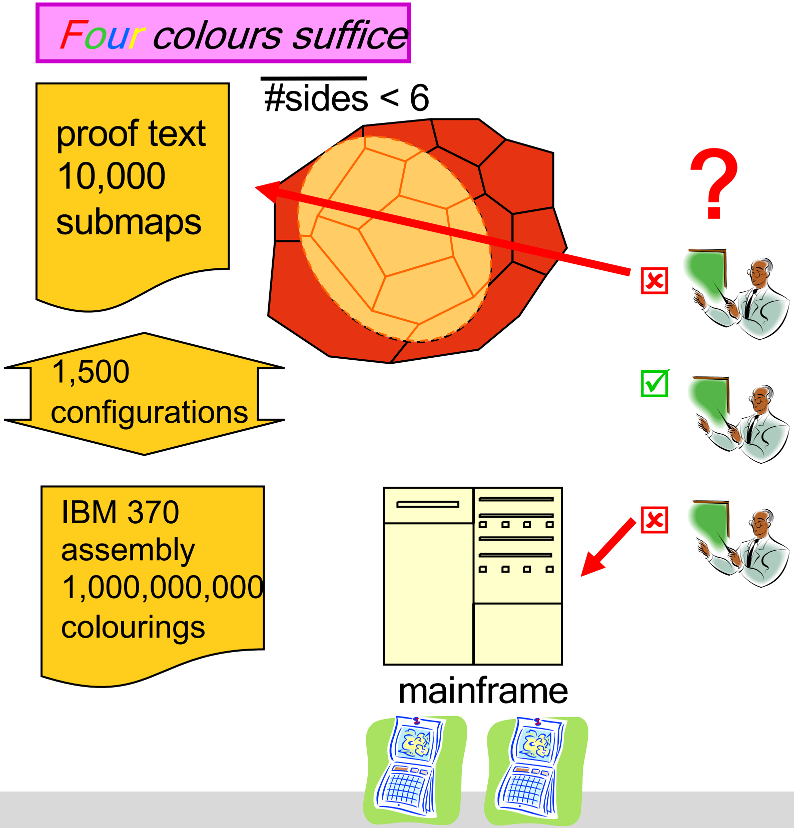
2004: The Four-Color Theorem

2012: The Odd Order Theorem

The granddad of computer proofs

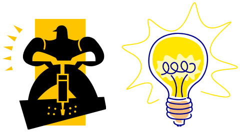


Appel & Haken
1976



The granddad of computer proofs

Four colours suffice



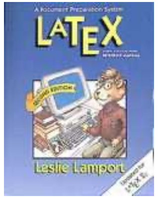
Robertson, Saunders, Seymour & Thomas
1995

proof text
35 pages

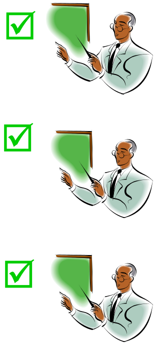
633
configurations

?

C programs
10,000
submaps
1,000,000,000
colourings

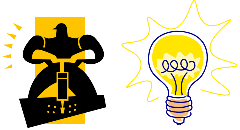


PC



The granddad of computer proofs

Four colours suffice !



Gonthier & Werner
2004

Coq formal
proof text

Coq programs
10,000
submaps
1,000,000,000
colourings

Coq program
proof

coq

The whole proof

Find a set of configurations such that:

(A) *unavoidability*: At least one appears in any planar map.

10,000 cases

(B) *reducibility*: Each one can be coloured to match any planar ring colouring.

1,000,000,000 cases

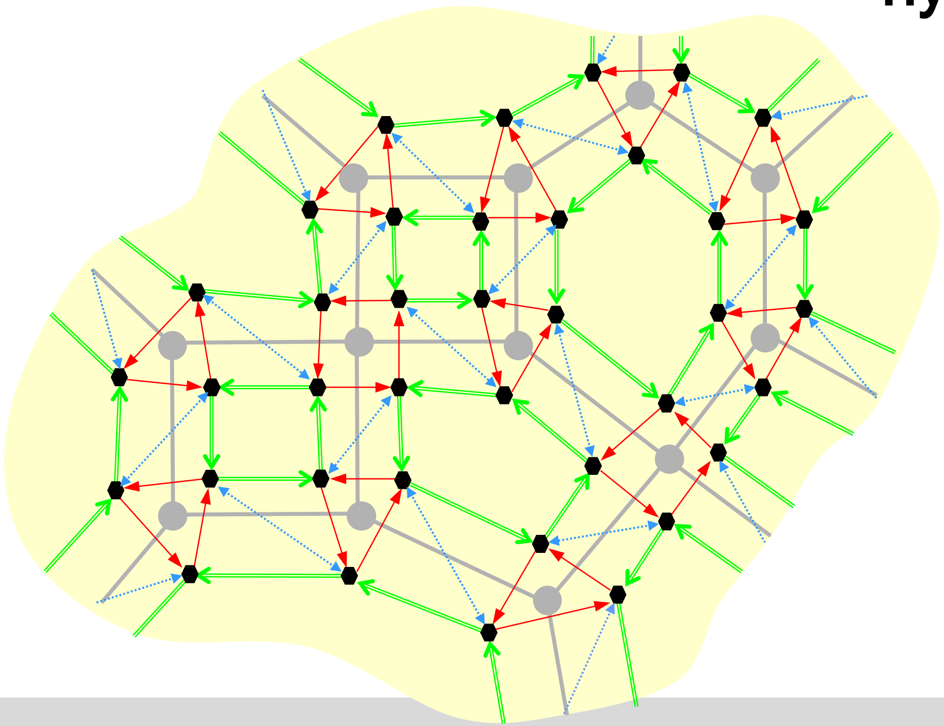
Verify that the combinatorics fit the topology (graph theory + analysis).

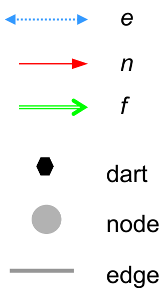
Crafting map descriptions

Euler: #edge + #node + #face = #dart + 2 * #comp

graph

hypermap





Proof by folklore

(**3.3**) *Let K be a configuration appearing in a triangulation T, and let S be the free completion of K. Then there is a projection ϕ of S into T such that $\phi(x) = x$ for all $x \in V(G(K)) \cup E(G(K)) \cup F(G(K))$.*

This is a "folklore" theorem, and we omit its [lengthy] proof…

```
Definition phi x :=
  if ac x then h x else
  if ac (edge x) then edge (h (edge x)) else
  if ac (node x) then face (edge (h (node x))) else
  edge (node (node (h (node (edge x))))).
```

Size matters

117 lines

```
(* Despite the larger number of cases, this proof is 15% shorter than the one *)
(* reling on the general transforms, because in each case we only need to     *)
(* a single subpath to the Walkup map, and we have extra facts on x, y and z  *)
(* that simplify the proof. In particular we only handle the #|G| = 3 case    *)
(* between steps 5) and 6), where most of G is known.                          *)
Theorem planar_Jordan G : planar G -> Jordan G.
Proof.
move: {-1}_.+1 (ltnSn #|G|) => m; elim: m G => // m IHm G leGm planarG [//|].
have{leGm} ltG'm z: #|@WalkupE G z| < m by rewrite -card_S_Walkup.
have IHe z := IHm (WalkupE z) (ltG'm z) (planar_WalkupE z planarG).
have IHn z := IHm (WalkupN z) (ltG'm z) (planar_WalkupN z planarG).
have{m IHm ltG'm} IHf z := IHm (WalkupF z) (ltG'm z) (planar_WalkupF z planarG).
have injG z : injective (val : WalkupE z -> G) := val_inj.
pose ofG (z x : G) : x != z -> WalkupE z := Sub x.
have uniqG z := map_inj_uniq (injG z); have mem2G z := mem2_map (injG z).
have clink_eq := sameP clinkP pred2P.
pose map_cpath f x p := {q | (f q.1, map f q.2) = (x : G, p) & cpath q.1 q.2}.
pose lift_cpath z H f x p := z \notin x :: p -> cpath x p -> map_cpath H f x p.
have liftE z x p: lift_cpath z (WalkupE z) val x p.
  rewrite /lift_cpath -has_pred1 /= => /norP[z'x]; pose u := ofG z x z'x.
  elim: p => [|y p IHp] /= in x z'x u *; first by exists (u, nil).
  case/norP=> z'y _ /andP[xCy /IHp[]// [v q] /= [Dv Dq] vCq].
  exists (u, v :: q); rewrite /= ?clink_eq -?val_eqE /= Dv ?Dq //.
  by have [<- | ->] := clinkP xCy; rewrite (negPf z'x, negPf z'y) eqxx ?orbT.
have liftN z x p: face z \notin p -> lift_cpath z (WalkupN z) val x p.
  rewrite /lift_cpath -!has_pred1 /= => fz'p /norP[z'x]; pose u := ofG z x z'x.
  move: fz'p; elim: p => [|y p IHp] /= in x z'x u *; first by exists (u, nil).
  case/norP=> fz'y _ /norP[z'y _] /andP[xCy /IHp[]// [v q] /= [Dv Dq] vCq].
  exists (u, v :: q); rewrite /= ?clink_eq -?val_eqE /= Dv ?Dq //.
  rewrite (canF_eq (canF_sym faceK)) (negPf fz'y).
  have [<- | ->] := clinkP xCy; last by rewrite orbC ifN ?eqxx.
  by rewrite (negPf z'x) if_same eqxx.
have liftF z x p: face (edge z) \notin p -> lift_cpath z (WalkupF z) val x p.
  rewrite /lift_cpath -!has_pred1 /= => fez'p /norP[z'x]; pose u := ofG z x z'x.
  move: fez'p; elim: p => [|y p IHp] /= in x z'x u *; first by exists (u, nil).
  case/norP=> fez'y _ /norP[z'y _] /andP[xCy /IHp[]// [v q] /= [Dv Dq] vCq].
  exists (u, v :: q); rewrite /= ?clink_eq -?val_eqE /= Dv ?Dq //.
  rewrite !(canF_eq nodeK) ifN_eq //.
```

Size matters

117 lines

```
move=> x p; have [t Lp]: {t | last x p = node t} := exist _ _ (esym (edgeK _)).
apply/and3P; rewrite Lp (finv_f nodeI) => -[/=/andP[p'x Up] xCp ptnx].
have /predT_subset-sGp: G \subset x :: p.
  apply/subsetPn=> -[z _ p'z]; have /liftE[//| [u q] /= [Du Dq] uCq] := p'z.
  have z't: t != z by rewrite (memPn p'z) // mem_behead ?(mem2l ptnx).
  have Lq: last u q = node (ofG z t z't).
    by apply/injG; rewrite -last_map /= Du Dq /= -Lp ifN ?(memPn p'z) ?mem_last.
  case/and3P: (IHe z (u :: q)); rewrite Lq (finv_f nodeI) -mem2G -uniqG /= Dq.
  by rewrite Du p'x ifN // (memPn p'z) // mem_behead ?(mem2r ptnx).
have oG: #|G| = (size p).+1 by rewrite -(eq_card sGp) (card_uniqP _) //= p'x.
case: p Up xCp => //= y p /andP[p'y Up] /andP[xCy yCp] in ptnx p'x Lp sGp oG *.
have x'y: y != x by rewrite (memPn p'x) ?mem_head.
have x't: t != x by rewrite (memPn p'x) ?(mem2l ptnx).
have x'nt: node t != x by rewrite -Lp (memPn p'x) ?mem_last.
case: p => [|z p] /= in ptnx p'x p'y Up yCp Lp sGp oG *.
  by rewrite mem2_seq1 Lp (inj_eq nodeI) andbC eq_sym (negPf x't) in ptnx.
have y'nt: node t != y by rewrite -Lp (memPn p'y) ?mem_last.
have{xCy} Dfx: face x = y.
  case/clinkP: xCy => // Dny; have /liftE[//| [u q] /= [Du Dq] uCq] := p'x.
  have Lq: finv node (last u q) = ofG x t x't.
    by apply/(canLR (finv_f nodeI))/injG; rewrite -last_map /= Du Dq /= ifN.
  case/and3P: (IHe x (u :: q)); rewrite Lq -mem2G -uniqG /= Du Dq -Dny eqxx p'y.
  by rewrite mem2_cons ifN // -(inj_eq nodeI) -Dny eq_sym in ptnx.
have{yCp Up} [[yCz zCp] [p'z Up]] := (andP yCp, andP Up).
move: ptnx p'x; rewrite !inE !negb_or mem2_cons => ptnx /and3P[y'x z'x p'x].
have z'y: y != z by rewrite (memPnC p'y) ?mem_head.
have{yCz} Dfy: face y = z.
  case/clinkP: yCz => // Dnz; pose u : WalkupN y := ofG y x y'x.
  have /liftF[|//| [v q] /= [Dv Dq] vCq] := p'y; first by rewrite Dnz nodeK.
  have Lq: finv node (last v q) = insubd v t.
    apply/(canLR (finv_f nodeI))/injG; rewrite -last_map Dq /= val_insubd /= Dv.
    by rewrite Lp; case: (t =P y) => [-> | _]; rewrite /= -?Dnz ?eqxx ?ifN.
  case/and3P: (IHf y (u :: v :: q)); rewrite Lq -mem2G -uniqG /= val_insubd Dq.
  rewrite vCq clink_eq -!val_eqE /= Dv Dnz nodeK !(inj_eq nodeI) -Dnz eqxx.
  rewrite -Dfx (inj_eq faceI) Dfx (negPf x'y) (negPf z'x) (negPf z'y) !eqxx.
  move: ptnx; rewrite orbT !negb_or p'x p'z inE Dnz (inj_eq nodeI) (negPf z'x).
  by rewrite eq_sym; case: ifP; rewrite // mem2_cons eqxx.
have Dt: t = y.
  have /liftE[//| [v q] /= [Dv Dq] vCq] := p'y.
  apply: contraNeq (IHe y (ofG y x y'x :: v :: q)) => y't; apply/and3P.
  have ->: finv node (last v q) = ofG y t y't.
    by apply/(canLR (finv_f nodeI))/injG; rewrite -last_map /= Dv Dq /= ifN.
```

```
  rewrite -mem2G -uniqG /= clink_eq orbC -val_eqE /= Dv Dq Dfx Dfy !eqxx /=.
  split=> //; first by rewrite negb_or z'x p'x p'z.
  by rewrite ifN_eqC // in ptnx; rewrite ifN // (memPn p'y) ?(mem2r ptnx).
move: ptnx Lp x'nt y'nt; rewrite {t x't}Dt eqxx /= => p_nx Lp x'ny y'ny.
have{p_nx} Dnx: node x = y.
  have /liftN[|//| [v q] /= [Dv Dq] vCq] := p'y; first by rewrite Dfy.
  apply: contraNeq (IHn y (ofG y x y'x :: v :: q)) => y'nx; apply/and3P.
  have ->: finv node (last v q) = v.
    apply/(canLR (finv_f nodeI))/injG; rewrite /= -last_map Dv Dq ifN //.
    by rewrite -Dfy faceK Dfy eqxx.
  rewrite -mem2G -uniqG /= clink_eq -!val_eqE /= Dv Dq negb_or z'x p'x p'z.
  rewrite (negPf y'ny) vCq -Dfy faceK Dfy Dfx !eqxx orbT; split=> //.
  rewrite -(inj_eq faceI) nodeK Dfy !ifN // mem2_cons eqxx /=.
  by rewrite inE (negPf y'nx) /= in p_nx.
rewrite inE negb_or z'y /= in p'y.
case: p zCp => /= [_ | t p /andP[zCt tCp]] in p'x p'y p'z Up Lp sGp oG *.
  case/idPn: planarG; rewrite -lt0n half_gt0 ltn_subRL -addnA /Euler_lhs {}oG.
  have oG1 f: injective f -> f x = y /\ f y = z -> f z = x /\ fcard f G = 1.
    move=> injf [Dy Dz]; rewrite -Dy -Dz in sGp; split.
      by have/predU1P[|/norP[]] := sGp (f z); rewrite ?orbF ?inj_eq ?(eq_sym z).
    rewrite // -(n_comp_connect (fconnect_sym injf) x).
    apply/esym/eq_n_comp_r => t; have:= sGp t; rewrite !inE -Dy.
    by case/or3P=> /eqP->; rewrite -?same_fconnect1_r ?connect0.
  have [[//|Dfz ->] [//|Dnz ->]]:= (oG1 _ faceI, oG1 _ nodeI).
  have [|_ ->] := oG1 _ edgeI; first by rewrite -Dnz -{2}Dnx -Dfz -Dfy !faceK.
  by rewrite !addnS [n_comp _ G](cardD1 (root glink x)) inE (roots_root glinkC).
have z'ny: node y != z by rewrite -Lp (memPn p'z) ?mem_last.
have{zCt} Dnt: node t = z.
  case/clinkP: zCt => // Dfz; have /liftE[//| [w q] /= [Dw Dq] wCq] := p'z.
  have /and3P[] := IHe z [:: ofG z x z'x, ofG z y z'y, w & q].
  rewrite mem2_cons -uniqG -(canF_eq (finv_f nodeI)) /= !clink_eq -!val_eqE /=.
  rewrite -last_map Dw Dq wCq Lp Dfx Dfy Dfz (negPf z'y) !eqxx !orbT.
  split=> //; first by rewrite negb_or y'x p'x p'y.
  by rewrite (negPf z'ny) eqxx /= !inE -val_eqE /= Dnx (negPf z'y) eqxx.
have /liftF[|//| [w q] /= [Dw Dq] wCq] := p'z.
  by rewrite -Dnt nodeK; case/andP: Up.
have /and3P[] := IHf z [:: ofG z x z'x, ofG z y z'y, w & q].
rewrite mem2_cons -uniqG -(canF_eq (finv_f nodeI)) /= !clink_eq -!val_eqE /=.
rewrite (negPf z'ny) -last_map Dw Dq Lp eqxx inE negb_or y'x p'x p'y Up wCq.
split=> //; last by rewrite inE -val_eqE /= Dnx (negPf z'y) eqxx.
rewrite Dfx (negPf z'ny) -Dfy (inj_eq faceI) Dfy -Dnt !(inj_eq nodeI) nodeK Dnt.
by rewrite (eq_sym z) (negPf z'y) !eqxx ifN ?eqxx ?orbT //; case/norP: p'z.
Qed.
```

Size matters

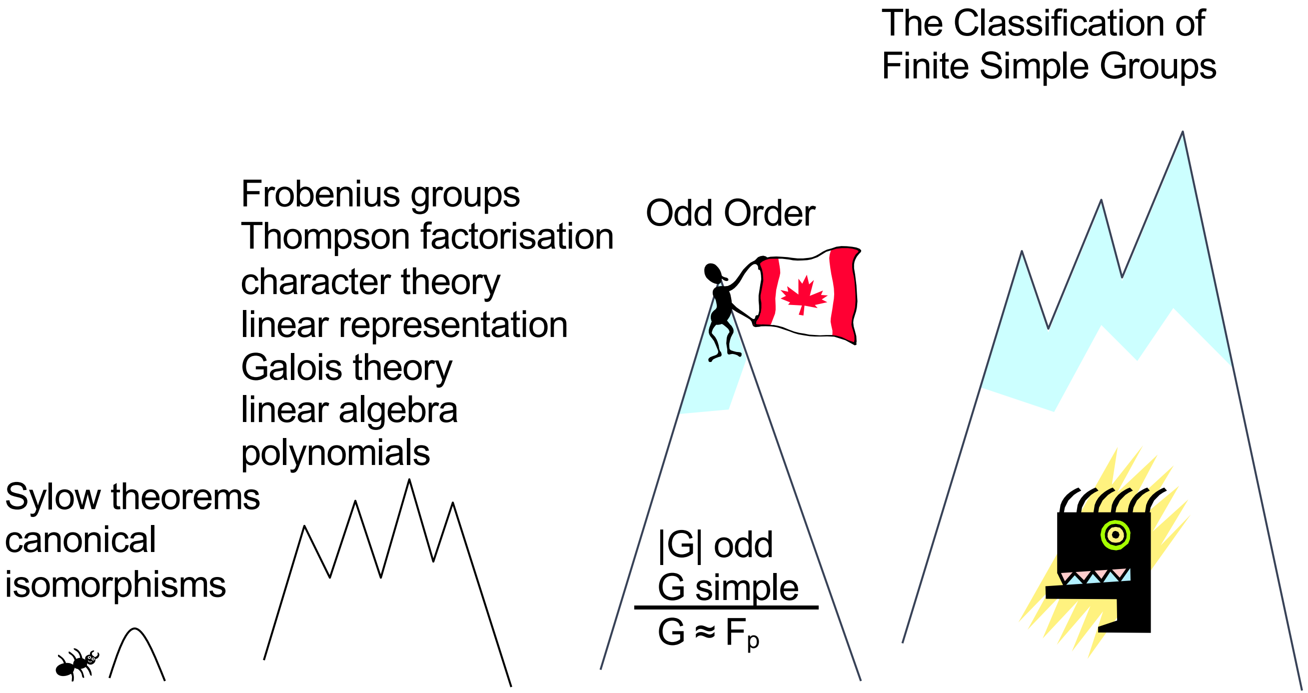
```
(* THE COMBINATORIAL JORDAN CURVE THEOREM *)

let lemmaLIPYTUI = prove(
  `!(H:(A)hypermap). planar_hypermap H ==> ~(?(p:num->A) k:num. is_Moebius_contour H p k)`,
   GEN_TAC
   THEN ABBREV_TAC `n = CARD (dart (H:(A)hypermap))`
   THEN POP_ASSUM MP_TAC
   THEN REWRITE_TAC[IMP_IMP]
   THEN SPEC_TAC (`H:(A)hypermap`, `H:(A)hypermap`)
   THEN SPEC_TAC (`n:num`, `n:num`)
   THEN INDUCT_TAC
   THENL[REPEAT STRIP_TAC
      THEN POP_ASSUM (MP_TAC o CONJUNCT2 o CONJUNCT2 o MATCH_MP lemma_darts_on_Moebius_contour)
      THEN DISCH_THEN (MP_TAC o MATCH_MP (ARITH_RULE `SUC (k:num) <= l:num ==> ~(l = 0)`))
      THEN ASM_REWRITE_TAC[]; ALL_TAC]
   THEN POP_ASSUM (LABEL_TAC "FI")
   THEN GEN_TAC
     THEN DISCH_THEN (CONJUNCTS_THEN2 (LABEL_TAC "F1") (LABEL_TAC "F2"))
```

600+ lines proof

1000+ lines technical lemmas

```
                                                                          CONJ th2 th1)))))
       THEN POP_ASSUM (fun ...                                  ...REWRITE_TAC[MATCH_MP lemma...
       THEN REWRITE_TAC[CONTRAPOS_THM]
       THEN STRIP_TAC
       THEN EXISTS_TAC `p:num->A` THEN EXISTS_TAC `k:num`
       THEN POP_ASSUM (fun th -> REWRITE_TAC[th]); ALL_TAC]
   THEN POP_ASSUM MP_TAC
   THEN REWRITE_TAC[]
   THEN USE_THEN "F3" MP_TAC
   THEN REWRITE_TAC[is_Moebius_contour]
   THEN DISCH_THEN (CONJUNCTS_THEN2 (LABEL_TAC "F7") (LABEL_TAC "F8"))
   THEN USE_THEN "F7" MP_TAC
   THEN REWRITE_TAC[lemma_def_inj_contour]
   THEN DISCH_THEN (CONJUNCTS_THEN2 (LABEL_TAC "F9") (LABEL_TAC "F10"))
   THEN DISCH_THEN (LABEL_TAC "F11")
   THEN REMOVE_THEN "F8" (X_CHOOSE_THEN `m:num` (X_CHOOSE_THEN `t:num` (CONJUNCTS_THEN2 (LABEL_TAC "F12") (CONJUNCTS_THEN2 (LABEL_TAC "F14") (CONJUNCTS_THEN2 (LABEL_TAC "F15") (CONJUNC
   THEN USE_THEN "F9" MP_TAC
   THEN REWRITE_TAC[lemma_def_contour]
   THEN DISCH_THEN (LABEL_TAC "F17")
   THEN ASM_CASES_TAC `m:num < t:num`
   THENL[POP_ASSUM (fun th -> (LABEL_TAC "G1" (MATCH_MP (ARITH_RULE `m:num < t:num ==> SUC m <= t`) th)))
       THEN USE_THEN "F17" (MP_TAC o SPEC `m:num`)
       THEN USE_THEN "F14" (fun th1 -> (USE_THEN "F15" (fun th2 -> (LABEL_TAC "G2" (MP (ARITH_RULE `m:num <= t:num /\ t < k:num ==> m < k`) (CONJ th1 th2))))))
       THEN USE_THEN "G2" (fun th -> REWRITE_TAC[th])
       THEN REWRITE_TAC[one_step_contour]
       THEN STRIP_TAC
       THENL[POP_ASSUM (fun th -> (LABEL_TAC "G3" th THEN MP_TAC th))
          THEN REWRITE_TAC[ADD1]
```

The Finite Group Challenge



The Odd Order Theorem

Theorem (Feit & Thompson, 1963):

All finite groups of odd order are solvable.

Proof. – *255 pages, 50 years*

Proofread. – 240 pages, 20 years

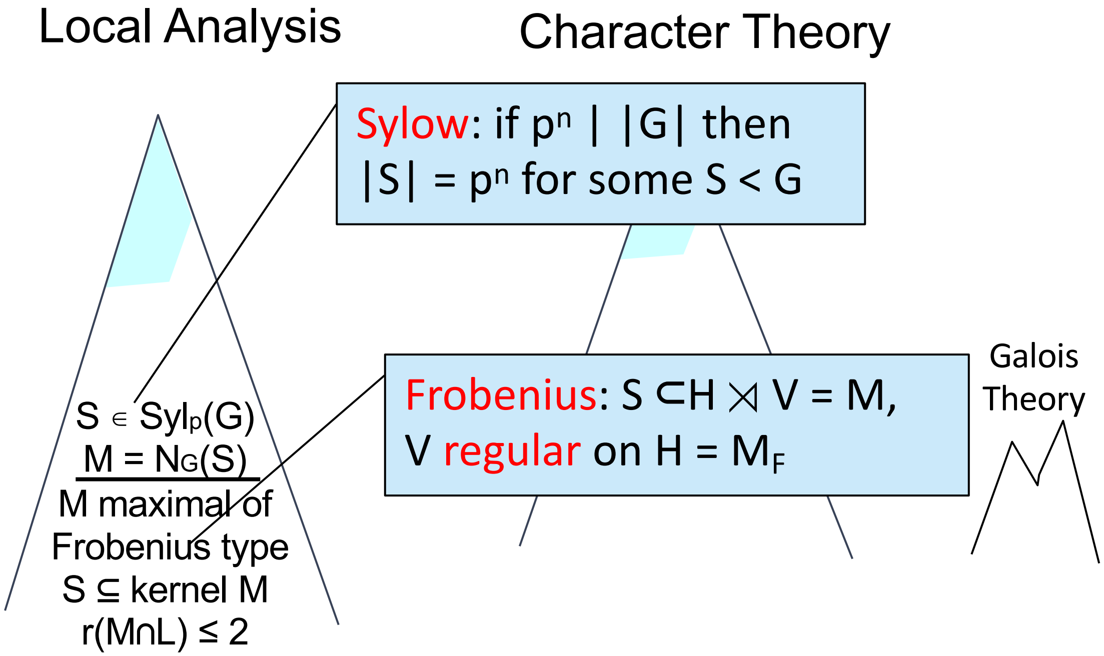
```
Theorem Feit_Thompson (gT : finGroupType) (G : {group gT}) :
  odd #|G| -> solvable G.
```

Definitions. – *54 LOC*

Proof. – *45,000 LOC, 2 years (+ 4 for the library)*

The Feit-Thompson proof

All finite groups of odd order are solvable.
Proof : Let G be a minimal counter-example ...



The Feit-Thompson proof

All finite groups of odd order are solvable.

Proof : Let G be a minimal counter-example ...

Local Analysis

Character Theory

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

M_i maximal Frobenius,
$M_i \neq M_j$, K_i kernel of M_i

characters form a
normed space

Galois
Theory

$\chi \in \mathrm{irr}(G)$

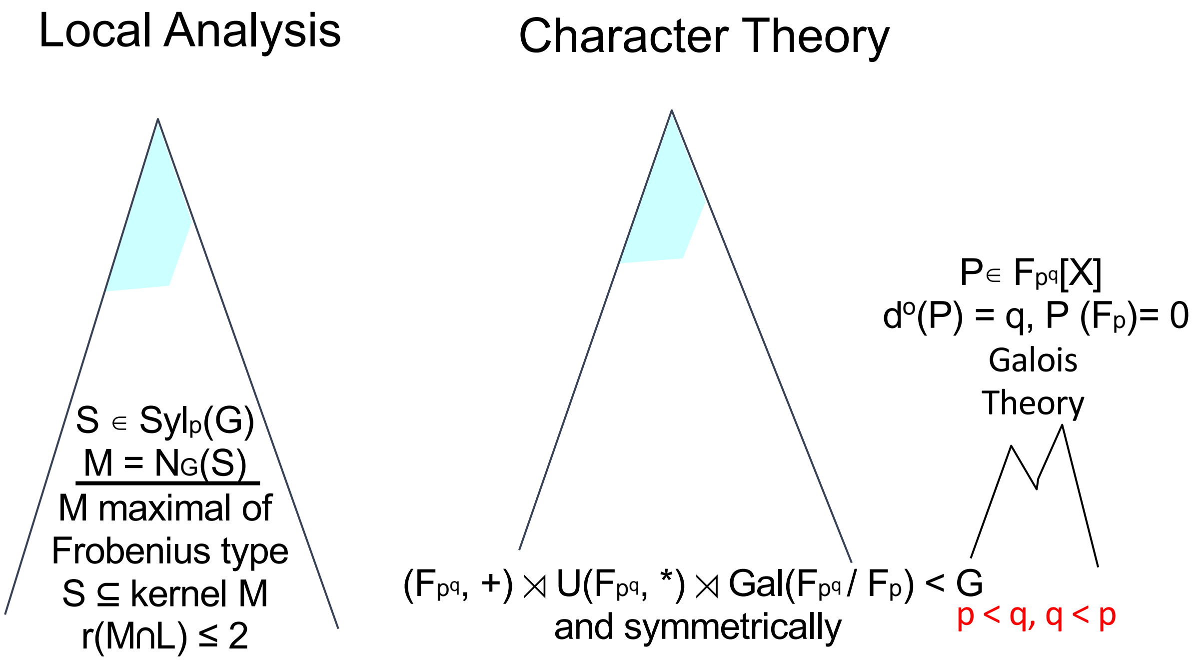
$G_0 = G - \bigcup K_i^G$

$\sum(|\chi(g_0)|^2 - 1) \leq |G| \sum(|K_i|/|M_i| - \|\chi_{M_i}\|^2)$

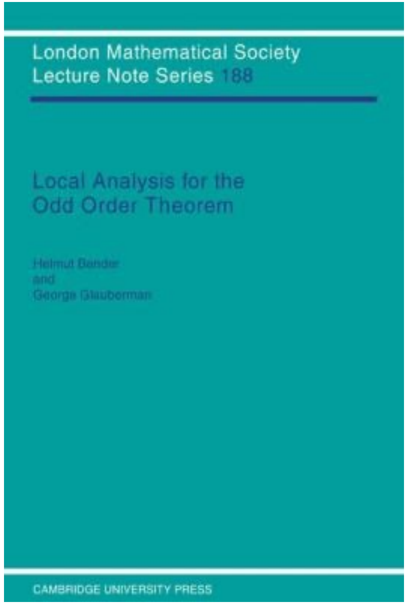
impossible

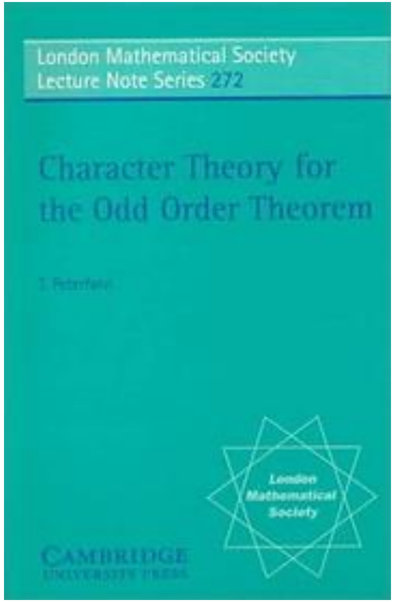
The Feit-Thompson proof

All finite groups of odd order are solvable.
Proof : Let G be a minimal counter-example ...

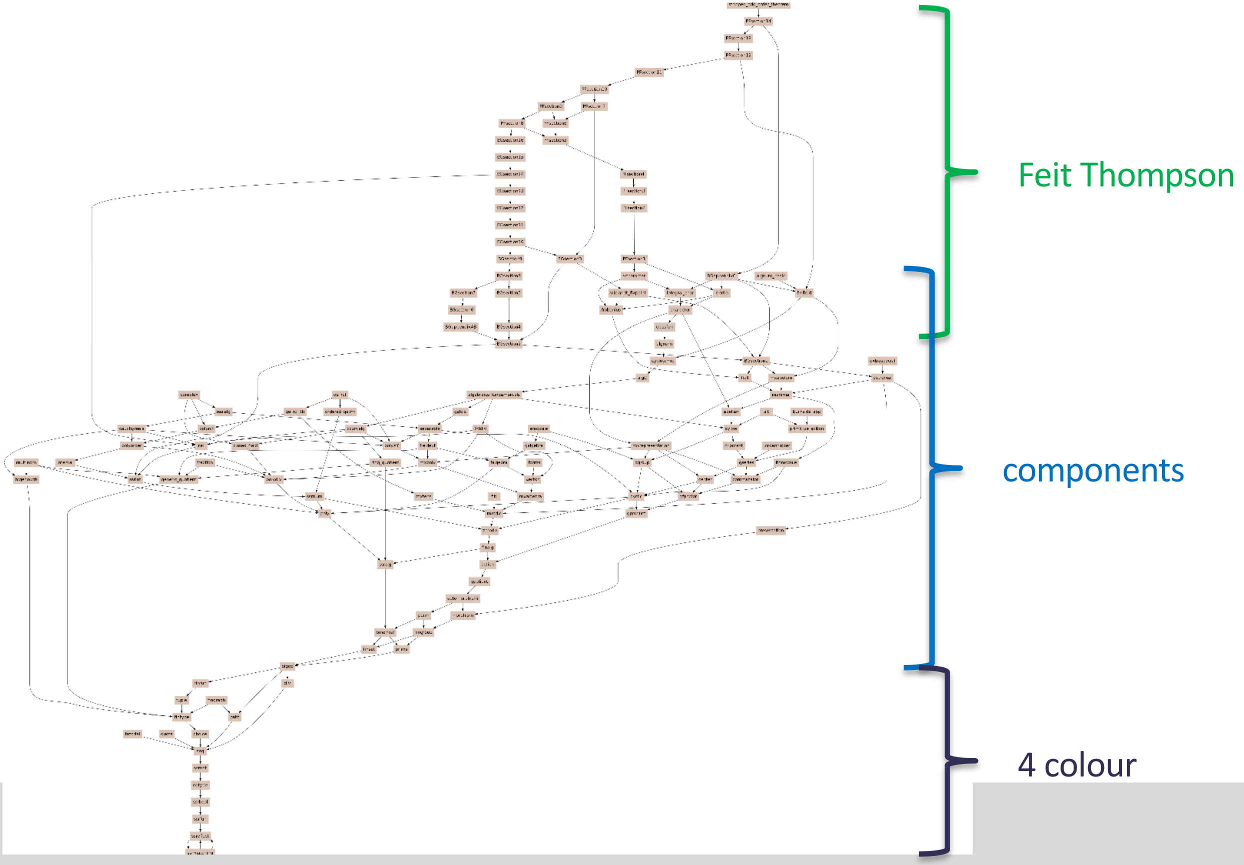


A mathematical library shelf

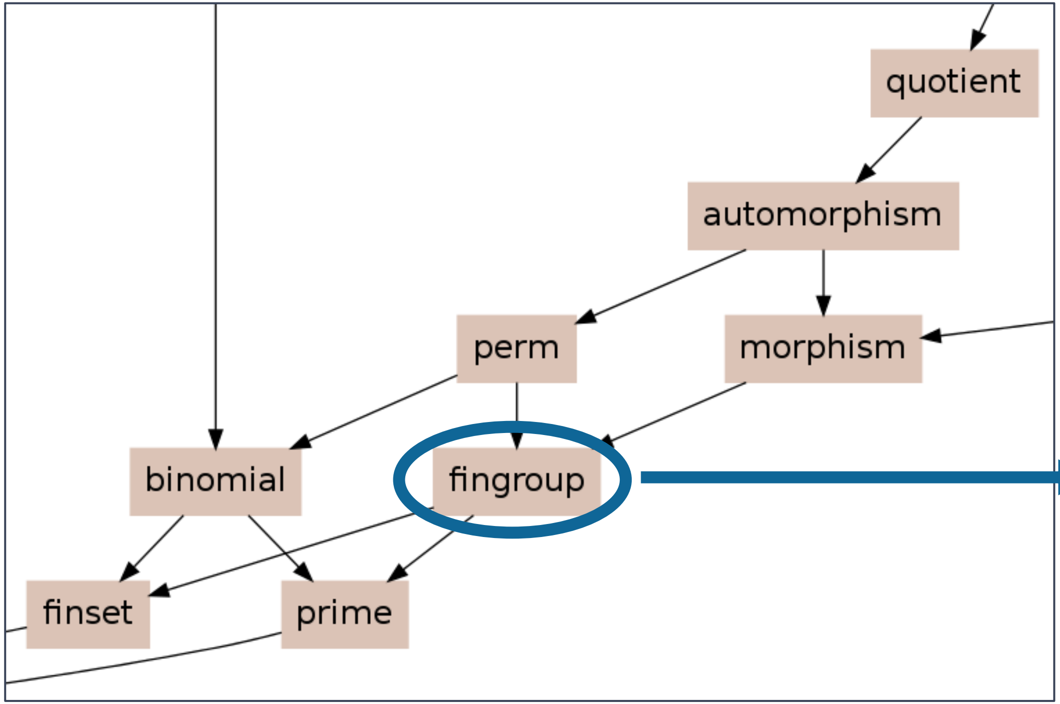


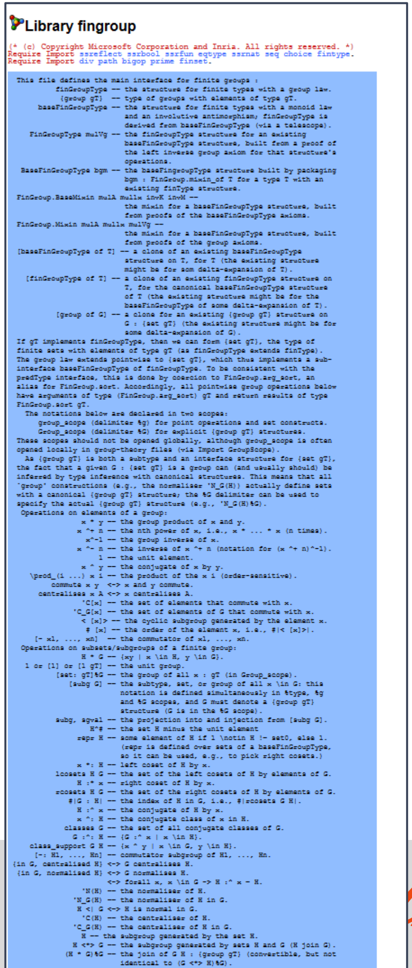


A mathematical library shelf



A mathematical library shelf





Library fingroup

```
(* (c) Copyright Microsoft Corporation and Inria. All rights reserved. *)
Require Import ssreflect ssrbool ssrfun eqtype ssrnat seq choice fintype.
Require Import div path bigop prime finset.

This file defines the main interface for finite groups :
        finGroupType -- the structure for finite types with a group law.
         {group gT}  -- type of groups with elements of type gT.
    baseFinGroupType -- the structure for finite types with a monoid law
                        and an involutive antimorphism; finGroupType is
                        derived from baseFinGroupType (via a telescope).
  FinGroupType mulVg -- the finGroupType structure for an existing
                        baseFinGroupType structure, built from a proof of
                        the left inverse group axiom for that structure's
                        operations.
 BaseFinGroupType bgm -- the baseFingroupType structure built by packaging
                        bgm : FinGroup.mixin_of T for a type T with an
                        existing finType structure.
FinGroup.BaseMixin mulA mul1x invK invM --
                        the mixin for a baseFinGroupType structure, built
                        from proofs of the baseFinGroupType axioms.
FinGroup.Mixin mulA mul1x mulVg --
                        the mixin for a baseFinGroupType structure, built
                        from proofs of the group axioms.
[baseFinGroupType of T] -- a clone of an existing baseFinGroupType
                        structure on T, for T (the existing structure
                        might be for som delta-expansion of T).
  [finGroupType of T] -- a clone of an existing finGroupType structure on
                        T, for the canonical baseFinGroupType structure
                        of T (the existing structure might be for the
                        baseFinGroupType of some delta-expansion of T).
       [group of G] -- a clone for an existing {group gT} structure on
                        G : {set gT} (the existing structure might be for
                        some delta-expansion of G).
If gT implements finGroupType, then we can form {set gT}, the type of
finite sets with elements of type gT (as finGroupType extends finType).
The group law extends pointwise to {set gT}, which thus implements a sub-
interface baseFinGroupType of finGroupType. To be consistent with the
predType interface, this is done by coercion to FinGroup.arg_sort, an
alias for FinGroup.sort. Accordingly, all pointwise group operations below
have arguments of type (FinGroup.arg_sort) gT and return results of type
FinGroup.sort gT.
  The notations below are declared in two scopes:
     group_scope (delimiter %g) for point operations and set constructs.
     Group_scope (delimiter %G) for explicit {group gT} structures.
These scopes should not be opened globally, although group_scope is often
opened locally in group-theory files (via Import GroupScope).
  As {group gT} is both a subtype and an interface structure for {set gT},
the fact that a given G : {set gT} is a group can (and usually should) be
inferred by type inference with canonical structures. This means that all
`group' constructions (e.g., the normaliser 'N_G(H)) actually define sets
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
            repr H -- some element of H if 1 \notin H != set0, else 1.
                       (repr is defined over sets of a baseFinGroupType,
                       so it can be used, e.g., to pick right cosets.)
            x *: H -- left coset of H by x.
        lcosets H G -- the set of the left cosets of H by elements of G.
            H :* x -- right coset of H by x.
        rcosets H G -- the set of the right cosets of H by elements of G.
           #|G : H| -- the index of H in G, i.e., #|rcosets G H|.
            H :^ x -- the conjugate of H by x.
            x ^: H -- the conjugate class of x in H.
          classes G -- the set of all conjugate classes of G.
           G :^: H -- {G :^ x | x \in H}.
    class_support G H -- {x ^ y | x \in G, y \in H}.
     [~: H1, ..., Hn] -- commutator subgroup of H1, ..., Hn.
{in G, centralised H} <-> G centralises H.
 {in G, normalised H} <-> G normalises H.
                      <-> forall x, x \in G -> H :^ x = H.
              'N(H) -- the normaliser of H.
            'N_G(H) -- the normaliser of H in G.
            H <| G <-> H is normal in G.
              'C(H) -- the centraliser of H.
            'C_G(H) -- the centraliser of H in G.
               H -- the subgroup generated by the set H.
           H <*> G -- the subgroup generated by sets H and G (H join G).
          (H * G)%G -- the join of G H : {group gT} (convertible, but not
                       identical to (G <*> H)%G).
```

inventors for the digital world

A mathematical library shelf

```
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

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

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

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

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

```
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.
```

Textbook vs. digital formal text

Theorem 14.7. Suppose $M \in \mathcal{M}_{\mathcal{P}}$ 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 $\widehat{Z} = Z - (K \cup K^*)$. Then, for some other $M^* \in \mathcal{M}_{\mathcal{P}}$ not conjugate to M,

(a) $\mathcal{M}(C_G(X)) = \{M^*\}$ 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^*, $\Rightarrow \sigma(M) \cap \pi(M^*) = \kappa(M^*)$

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

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

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

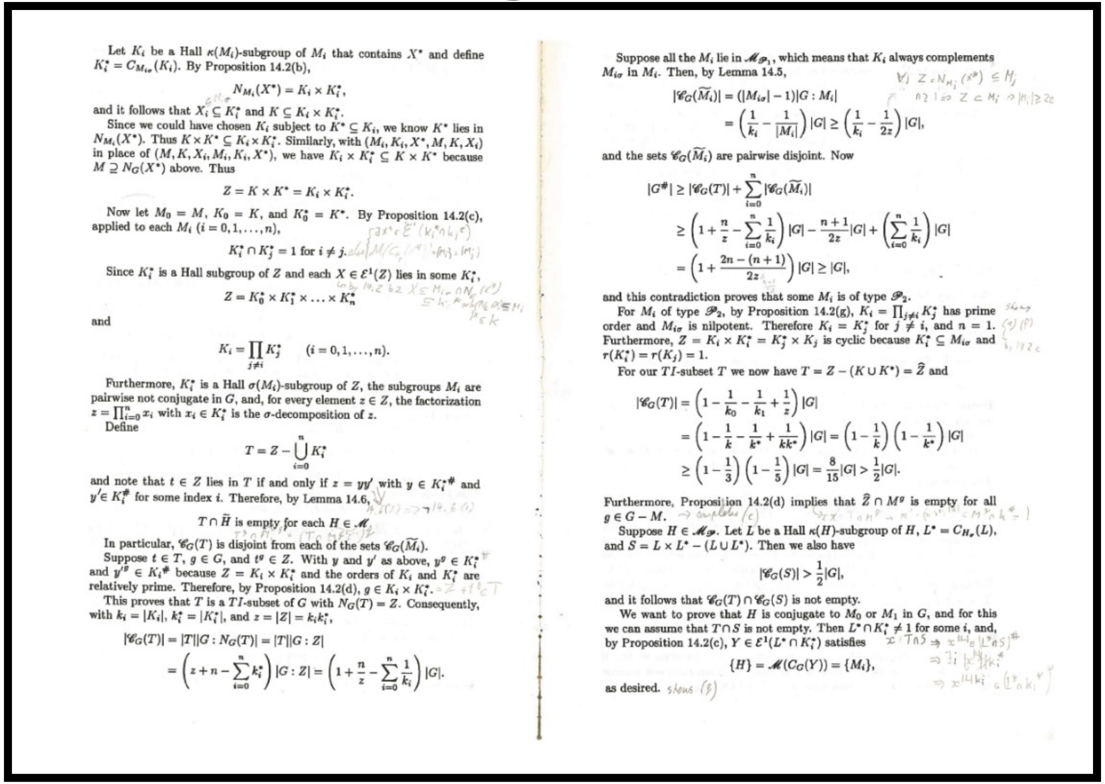
(g) every $H \in \mathcal{M}_{\mathcal{P}}$ is conjugate to M or M^* in G, and

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

Textbook vs. digital formal text

```
Theorem Ptype_embedding 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]].
```

Textbook vs. digital formal text



Let K_i be a Hall $\kappa(M_i)$-subgroup of M_i that contains X^* and define $K_i^* = C_{M_{i\sigma}}(K_i)$. By Proposition 14.2(b),

$$N_{M_i}(X^*) = K_i \times K_i^*,$$

and it follows that $X_i \subseteq K_i^*$ and $K \subseteq K_i \times K_i^*$.

Since we could have chosen K_i subject to $K^* \subseteq K_i$, we know K^* lies in $N_{M_i}(X^*)$. Thus $K \times K^* \subseteq K_i \times K_i^*$. Similarly, with $(M_i, K_i, X^*, M, K, X_i)$ in place of $(M, K, X_i, M_i, K_i, X^*)$, we have $K_i \times K_i^* \subseteq K \times K^*$ because $M \supseteq N_G(X^*)$ above. Thus

$$Z = K \times K^* = K_i \times K_i^*.$$

Now let $M_0 = M$, $K_0 = K$, and $K_0^* = K^*$. By Proposition 14.2(c), applied to each M_i $(i = 0, 1, \ldots, n)$,

$$K_i^* \cap K_j^* = 1 \text{ for } i \neq j.$$

Since K_i^* is a Hall subgroup of Z and each $X \in \mathcal{E}^1(Z)$ lies in some K_i^*,

$$Z = K_0^* \times K_1^* \times \ldots \times K_n^*$$

and

$$K_i = \prod_{j \neq i} K_j^* \qquad (i = 0, 1, \ldots, n).$$

Furthermore, K_i^* is a Hall $\sigma(M_i)$-subgroup of Z, the subgroups M_i are pairwise not conjugate in G, and, for every element $z \in Z$, the factorization $z = \prod_{i=0}^n x_i$ with $x_i \in K_i^*$ is the σ-decomposition of z.

Define

$$T = Z - \bigcup_{i=0}^n K_i^*$$

and note that $t \in Z$ lies in T if and only if $z = yy'$ with $y \in K_i^{*\#}$ and $y' \in K_i^{\#}$ for some index i. Therefore, by Lemma 14.6,

$$T \cap \tilde{H} \text{ is empty for each } H \in \mathscr{M}.$$

In particular, $\mathscr{C}_G(T)$ is disjoint from each of the sets $\mathscr{C}_G(\widetilde{M}_i)$.

Suppose $t \in T$, $g \in G$, and $t^g \in Z$. With y and y' as above, $y^g \in K_i^*$ and $y'^g \in K_i^{\#}$ because $Z = K_i \times K_i^*$ and the orders of K_i and K_i^* are relatively prime. Therefore, by Proposition 14.2(d), $g \in K_i \times K_i^*$.

This proves that T is a TI-subset of G with $N_G(T) = Z$. Consequently, with $k_i = |K_i|$, $k_i^* = |K_i^*|$, and $z = |Z| = k_i k_i^*$,

$$|\mathscr{C}_G(T)| = |T||G : N_G(T)| = |T||G : Z|$$
$$= \left(z + n - \sum_{i=0}^n k_i^*\right)|G : Z| = \left(1 + \frac{n}{z} - \sum_{i=0}^n \frac{1}{k_i}\right)|G|.$$

Suppose all the M_i lie in $\mathscr{M}_{\mathscr{P}_1}$, which means that K_i always complements $M_{i\sigma}$ in M_i. Then, by Lemma 14.5,

$$|\mathscr{C}_G(\widetilde{M}_i)| = (|M_{i\sigma}| - 1)|G : M_i|$$
$$= \left(\frac{1}{k_i} - \frac{1}{|M_i|}\right)|G| \geq \left(\frac{1}{k_i} - \frac{1}{2z}\right)|G|,$$

and the sets $\mathscr{C}_G(\widetilde{M}_i)$ are pairwise disjoint. Now

$$|G^{\#}| \geq |\mathscr{C}_G(T)| + \sum_{i=0}^n |\mathscr{C}_G(\widetilde{M}_i)|$$
$$\geq \left(1 + \frac{n}{z} - \sum_{i=0}^n \frac{1}{k_i}\right)|G| - \frac{n+1}{2z}|G| + \left(\sum_{i=0}^n \frac{1}{k_i}\right)|G|$$
$$= \left(1 + \frac{2n - (n+1)}{2z}\right)|G| \geq |G|,$$

and this contradiction proves that some M_i is of type $\mathscr{P}_2$.

For M_i of type $\mathscr{P}_2$, by Proposition 14.2(g), $K_i = \prod_{j \neq i} K_j^*$ has prime order and $M_{i\sigma}$ is nilpotent. Therefore $K_i = K_j^*$ for $j \neq i$, and $n = 1$. Furthermore, $Z = K_i \times K_i^* = K_j^* \times K_j$ is cyclic because $K_i^* \subseteq M_{i\sigma}$ and $r(K_i^*) = r(K_j) = 1$.

For our TI-subset T we now have $T = Z - (K \cup K^*) = \widehat{Z}$ and

$$|\mathscr{C}_G(T)| = \left(1 - \frac{1}{k_0} - \frac{1}{k_1} + \frac{1}{z}\right)|G|$$
$$= \left(1 - \frac{1}{k} - \frac{1}{k^*} + \frac{1}{kk^*}\right)|G| = \left(1 - \frac{1}{k}\right)\left(1 - \frac{1}{k^*}\right)|G|$$
$$\geq \left(1 - \frac{1}{3}\right)\left(1 - \frac{1}{5}\right)|G| = \frac{8}{15}|G| > \frac{1}{2}|G|.$$

Furthermore, Proposition 14.2(d) implies that $\widehat{Z} \cap M^g$ is empty for all $g \in G - M$.

Suppose $H \in \mathscr{M}_{\mathscr{P}}$. Let L be a Hall $\kappa(H)$-subgroup of H, $L^* = C_{H_\sigma}(L)$, and $S = L \times L^* - (L \cup L^*)$. Then we also have

$$|\mathscr{C}_G(S)| > \frac{1}{2}|G|,$$

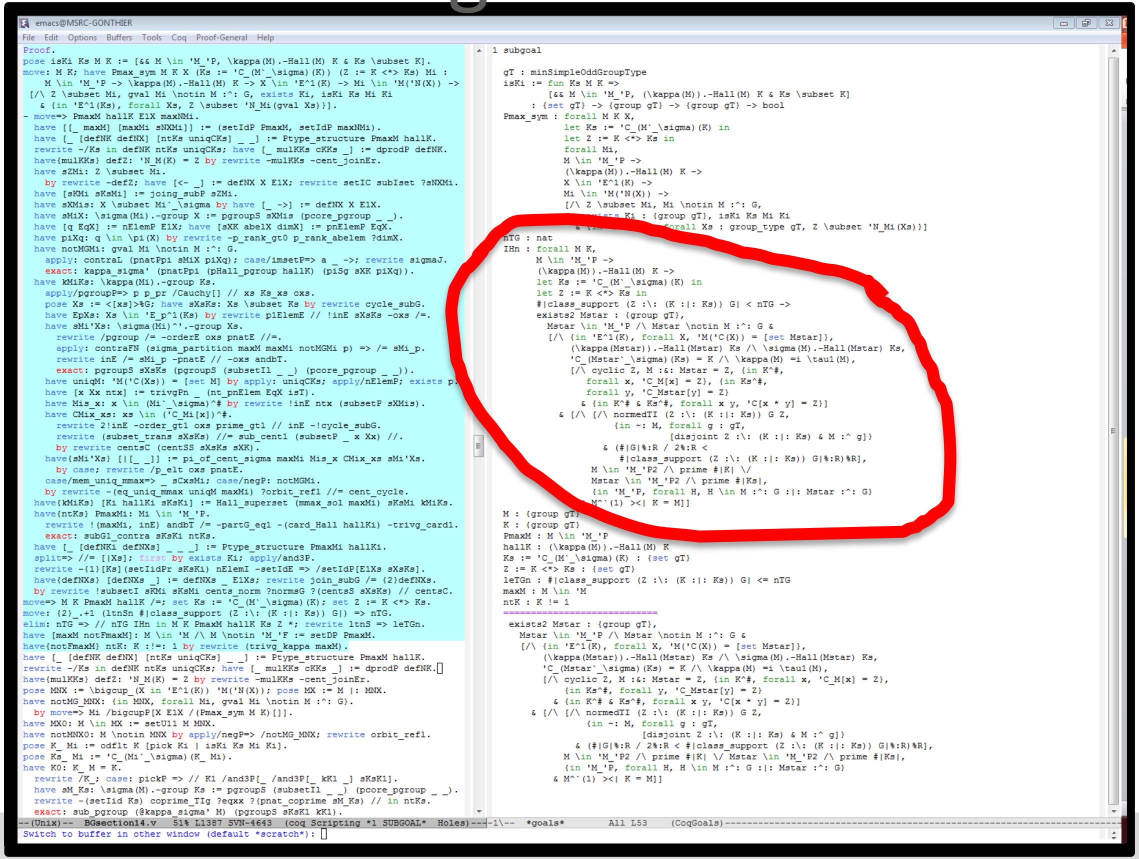
and it follows that $\mathscr{C}_G(T) \cap \mathscr{C}_G(S)$ is not empty.

We want to prove that H is conjugate to M_0 or M_1 in G, and for this we can assume that $T \cap S$ is not empty. Then $L^* \cap K_i^* \neq 1$ for some i, and, by Proposition 14.2(c), $Y \in \mathcal{E}^1(L^* \cap K_i^*)$ satisfies

$$\{H\} = \mathscr{M}(C_G(Y)) = \{M_i\},$$

as desired.

Textbook vs. digital formal text



inventors for the digital world

Interactive Math

$tr\ AB$	$= \sum_i (AB)_{i,i}$	*expand trace*
	$= \sum_i \sum_j A_{i,j} B_{j,i}$	*expand multiply*
	$= \sum_j \sum_i A_{i,j} B_{j,i}$	*exchange sums*
	$= \sum_j \sum_i B_{j,i} A_{i,j}$	*commute scalars*
	$= \sum_j (AB)_{i,i}$	*unexpand multiply*
	$= tr\ BA$	$\square$

$$tr\ A = \sum_i A_{i,i}$$

$$(AB)_{i,j} = \sum_k A_{i,k} B_{k,j}$$

Interactive Math

$tr\, AB = \sum_i (AB)_{i,i}$

$= \sum_i \sum_j A_{i,j} B_{j,i}$

$= \ldots$

$= \sum_j \sum_i A_{i,j} B_{j,i}$

$= \ldots$

$= \sum_j \sum_i B_{j,i} A_{i,j}$

$= \sum_j (AB)_{i,i}$

$= tr\, BA$

$tr\, A = \sum_i A_{i,i}$

$(AB)_{i,j} = \sum_k A_{i,k} B_{k,j}$

```
Lemma mxtrace_mulC m n (A : 'M[R]_(m, n)) B :
   \tr (A *m B) = \tr (B *m A).
Proof.
gen have trE m n A B / \tr (A *m B) = \sum_i \sum_j A i j * B j i.
  by apply: eq_bigr => i _; rewrite mxE.
rewrite {}!trE exchange_big.
by do 2!apply: eq_bigr => ? _; apply: mulrC.
Qed.
```

```
  ...
  trE : forall (m n : nat) (A: 'M_(m, n)) (B : 'M_(n, m)),
          \tr (A *m B) = \sum_i \sum_j A i j * B j i
  ============================
\tr (A *m B) = \tr (B *m A)
  ============================
   \tr (A *m B) = \tr (B *m A)
```

Interactive Math

$$tr\, AB = \textstyle\sum_i (AB)_{i,i}$$

$$= \textstyle\sum_i \sum_j A_{i,j}\, B_{j,i}$$

$$= \textstyle\sum_j \sum_i A_{i,j}\, B_{j,i}$$

$$= \textstyle\sum_j \sum_i B_{j,i}\, A_{i,j}$$

$$= \textstyle\sum_j (AB)_{i,i}$$

$$= tr\, BA$$

$$tr\, A = \textstyle\sum_i A_{i,i}$$

$$(AB)_{i,j} = \textstyle\sum_k A_{i,k}\, B_{k,j}$$

```
Lemma mxtrace_mulC m n (A : 'M[R]_(m, n)) B :
   \tr (A *m B) = \tr (B *m A).
Proof.
gen have trE m n A B / \tr (A *m B) = \sum_i \sum_j A i j * B j i.
  by apply: eq_bigr => i _; rewrite mxE.
rewrite {}!trE exchange_big.
by do 2!apply: eq_bigr => ? _; apply: mulrC.
Qed.
```

```
  R : comRingType
  m : nat
  n : nat
  A : 'M_(m, n)
  B : 'M_(n, m)
  ============================
   \tr (A *m B) = \tr (B *m A)
```

Interactive Math

$$tr\,AB = \sum_i (AB)_{i,i}$$
$$= \sum_i \sum_j A_{i,j} B_{j,i}$$
$$= \sum_j \sum_i A_{i,j} B_{j,i}$$
$$= \sum_j \sum_i B_{j,i} A_{i,j}$$
$$= \sum_j (AB)_{j,j}$$
$$= tr\,BA$$

$$tr\,A = \sum_i A_{i,i}$$
$$(AB)_{i,j} = \sum_k A_{i,k} B_{k,j}$$

```
Lemma mxtrace_mulC m n (A : 'M[R]_(m, n)) B :
   \tr (A *m B) = \tr (B *m A).
Proof.
gen have trE m n A B / \tr (A *m B) = \sum_i \sum_j A i j * B j i.
  by apply: eq_bigr => i _; rewrite mxE.
rewrite {}!trE exchange_big.
by do 2!apply: eq_bigr => ? _; apply: mulrC.
Qed.
```

```
2 subgoals
   ...
   ============================
   \tr (A *m B) = \sum_i \sum_j A i j * B j i

subgoal 2 is:
 \tr (A *m B) = \tr (B *m A)
```

Interactive Math

$$tr\,AB = \sum_i (AB)_{i,i}$$
$$= \sum_i \sum_j A_{i,j}\, B_{j,i}$$
$$= \sum_j \sum_i A_{i,j}\, B_{j,i}$$
$$= \sum_j \sum_i B_{j,i}\, A_{i,j}$$
$$= \sum_j (AB)_{j,j}$$
$$= tr\,BA$$

$$tr\,A = \sum_i A_{i,i}$$
$$(AB)_{i,j} = \sum_k A_{i,k}\, B_{k,j}$$

```
Lemma mxtrace_mulC m n (A : 'M[R]_(m, n)) B :
   \tr (A *m B) = \tr (B *m A).
Proof.
gen have trE m n A B / \tr (A *m B) = \sum_i \sum_j A i j * B j i.
  by apply: eq_bigr => i _; rewrite mxE.
rewrite {}!trE exchange_big.
by do 2!apply: eq_bigr => ? _; apply: mulrC.
Qed.
```

```
  ...
  trE : forall (m n : nat) (A: 'M_(m, n)) (B : 'M_(n, m)),
           \tr (A *m B) = \sum_i \sum_j A i j * B j i
  ==============================
\tr (A *m B) = \tr (B *m A)
```

Interactive Math

$$tr\,AB = \textstyle\sum_i (AB)_{i,i}$$

$$= \textstyle\sum_i \sum_j A_{i,j}\, B_{j,i}$$

$$= \textstyle\sum_j \sum_i A_{i,j}\, B_{j,i}$$

$$= \textstyle\sum_j \sum_i B_{j,i}\, A_{i,j}$$

$$= \textstyle\sum_j (AB)_{i,i}$$

$$= tr\,BA$$

$$tr\,A = \textstyle\sum_i A_{i,i}$$

$$(AB)_{i,j} = \textstyle\sum_k A_{i,k}\, B_{k,j}$$

```
Lemma mxtrace_mulC m n (A : 'M[R]_(m, n)) B :
   \tr (A *m B) = \tr (B *m A).
Proof.
gen have trE m n A B / \tr (A *m B) = \sum_i \sum_j A i j * B j i.
  by apply: eq_bigr => i _; rewrite mxE.
rewrite {}!trE exchange_big.
by do 2!apply: eq_bigr => ? _; apply: mulrC.
Qed.
```

```
  R : comRingType
  m : nat
  n : nat
  A : 'M_(m, n)
  B : 'M_(n, m)
  ==============================
  \sum_j \sum_i A i j * B j i = \sum_i \sum_j B i j * A j i
```

Interactive Math

$$tr\,AB = \textstyle\sum_i (AB)_{i,i}$$

$$= \textstyle\sum_i \sum_j A_{i,j}\, B_{j,i}$$

$$= \textstyle\sum_j \sum_i A_{i,j}\, B_{j,i}$$

$$= \textstyle\sum_j \sum_i B_{j,i}\, A_{i,j}$$

$$= \textstyle\sum_j (AB)_{i,i}$$

$$= tr\,BA$$

$$tr\,A = \textstyle\sum_i A_{i,i}$$

$$(AB)_{i,j} = \textstyle\sum_k A_{i,k}\, B_{k,j}$$

```
Lemma mxtrace_mulC m n (A : 'M[R]_(m, n)) B :
   \tr (A *m B) = \tr (B *m A).
Proof.
gen have trE m n A B / \tr (A *m B) = \sum_i \sum_j A i j * B j i.
  by apply: eq_bigr => i _; rewrite mxE.
rewrite {}!trE exchange_big.
by do 2!apply: eq_bigr => ? _; apply: mulrC.
Qed.
```

```
  Proof completed.
```

```
mxtrace_mulC is defined
```

Algebraic Notation

$$\sum_{i<n} a_i\, x^i$$

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

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

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

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

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

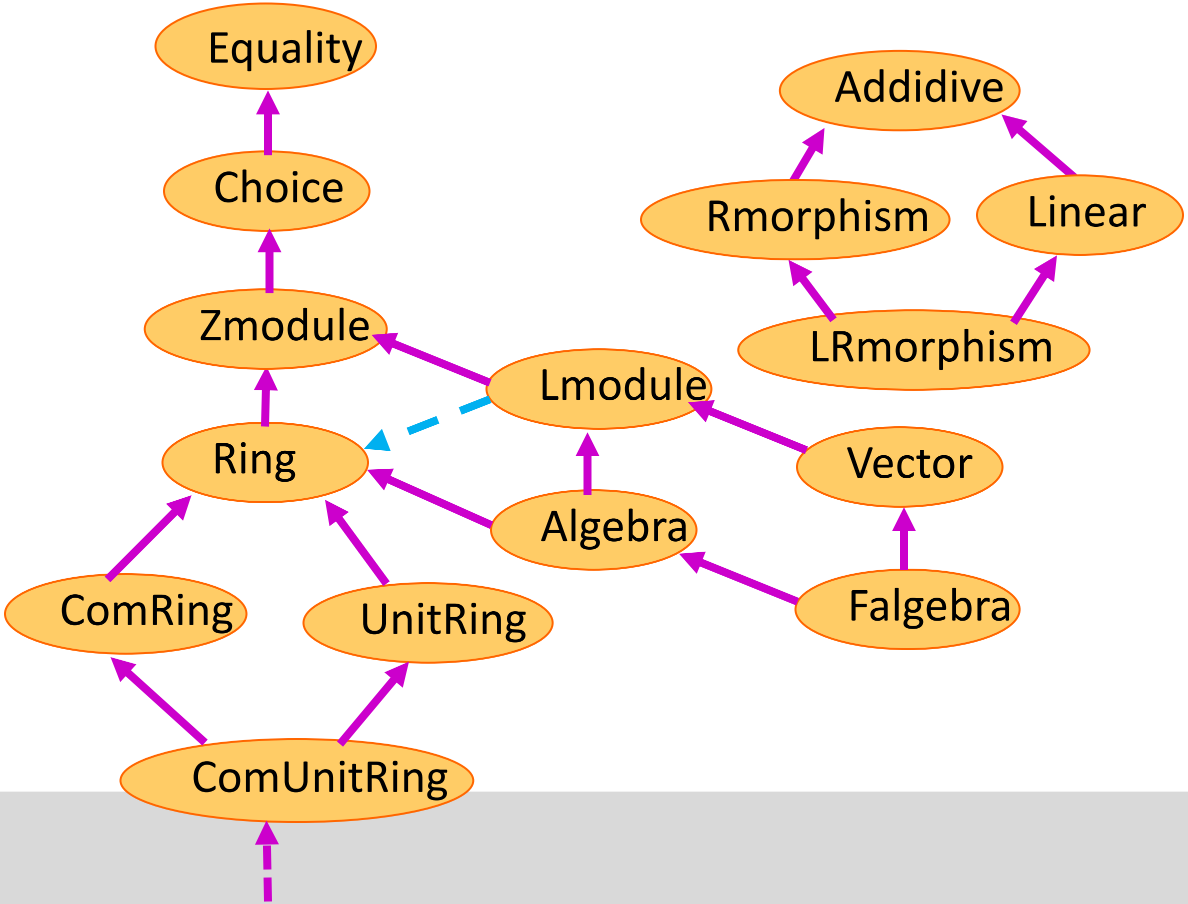
```
\bigcap_{H < G \atop H {\rm\ maximal}} H
```

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

Implementing notation

```
Definition mxtrace (R : ringType) n (A : 'M[R]_n) :=
   @bigop R 'I_n 0 +%R (index_enum _)
       (fun i : 'I_n => fun_of_matrix A i i)
```

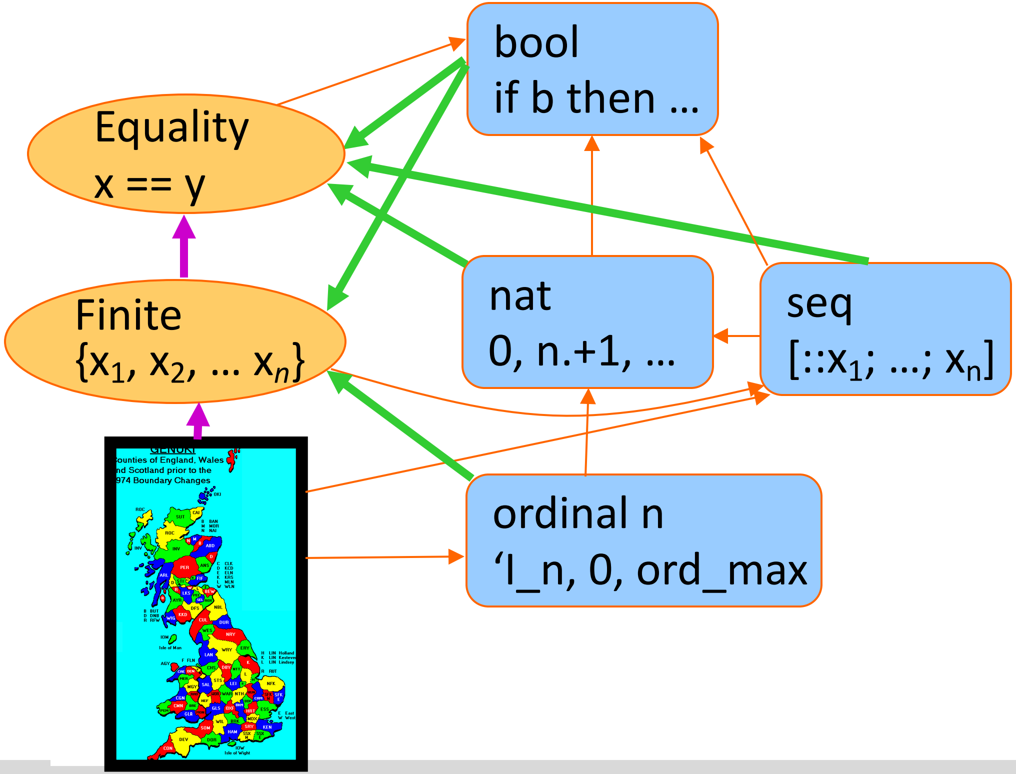
Algebra interfaces



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)
```

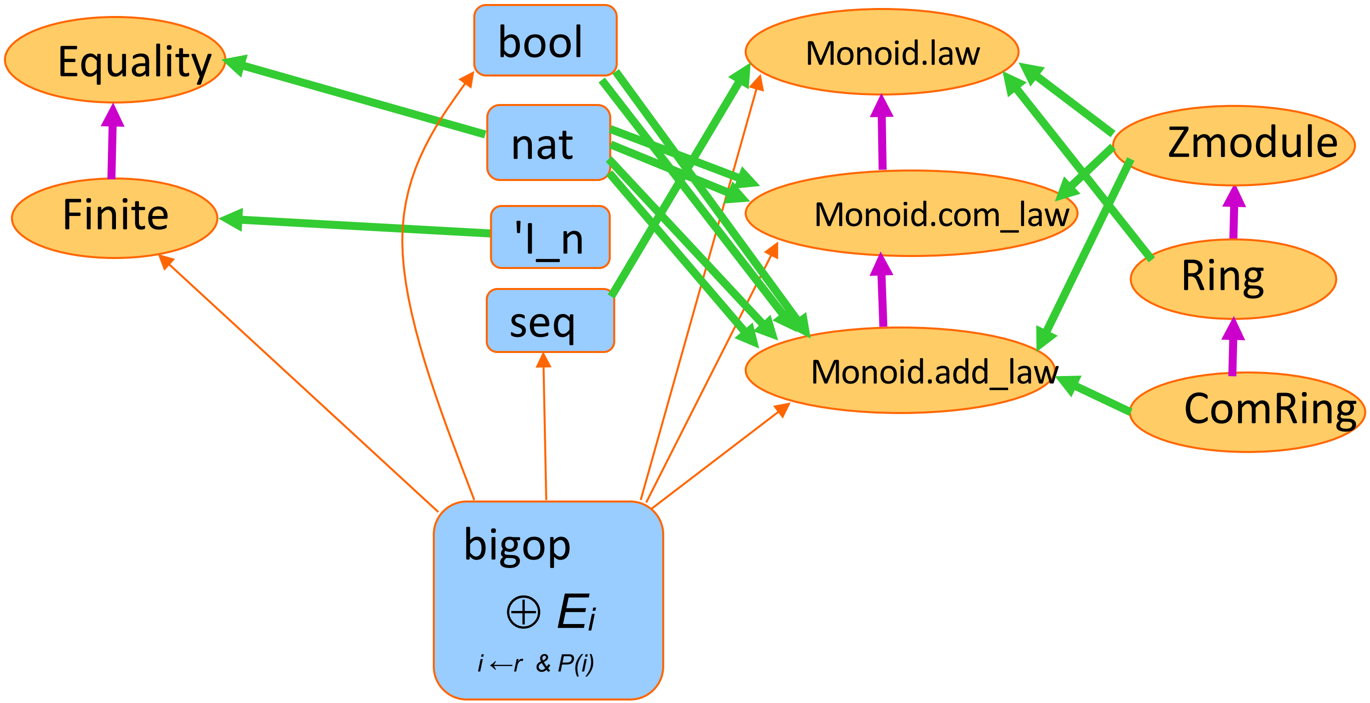
Basic interfaces and objects



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)
```

Interfacing big operators



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

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 $\mathbf{H}$	$\{1\} \cup H^2 = H$
normaliser $\mathbf{N\ (H)}$	$\{x \in G \mid Hx = xH \text{ (or } H^x = H)\}$
normal subgroup $\mathbf{H \trianglelefteq G}$	$H \leq G \leq N\ (H)$
factor group $\mathbf{G / H}$	$\{Hx \mid x \in N_G(H)\}$
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;
  _ : cos
```

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

```
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.
```

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.

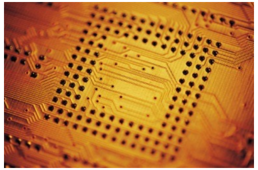
```
Definition is_class_fun (B : {set gT}) (f : {ffun gT ->
   algC}) := [forall x, forall y in B, f (x ^ y) == f x]
    && (support f \subset B).
Record classfun :=
  Classfun {cfun_val; _ : is_class_fun G cfun_val}.
```

Dot product, orthogonal predicates don't use G.

Interface encapsulates "character are a semiring".

Proof Algorithms

Reducibility -> Model checking



decision diagrams

circuit design

Unavoidability -> Combinatorial search

Davies-Putnam, αβ pruning

Device drivers

Link -> Abstract interpretation

Flight software

Reflecting reducibility

Setup

Variable cf : config.

Definition cfreducible : Prop := …

Definition check_reducible : bool := …

Lemma check_reducible_valid : check_reducible -> cfreducible.

Usage

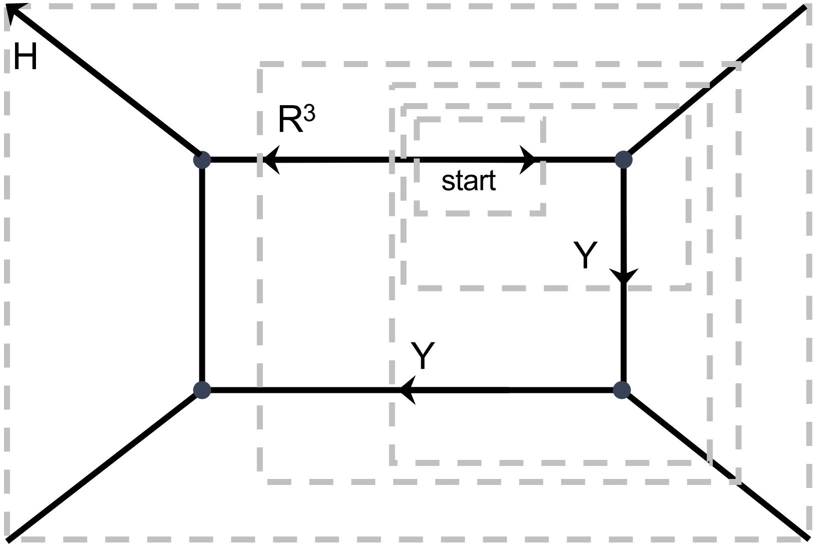
Lemma cfred232 : cfreducible (Config 11 33 37 H 2 H 13 Y 5 H 10 H 1 H 1 Y 3 H 11 Y 4 H 9 H 1 Y 3 H 9 Y 6 Y 1 Y 1 Y 3 Y 1 Y Y 1 Y).

Proof. apply check_reducible_valid; by compute. Qed.

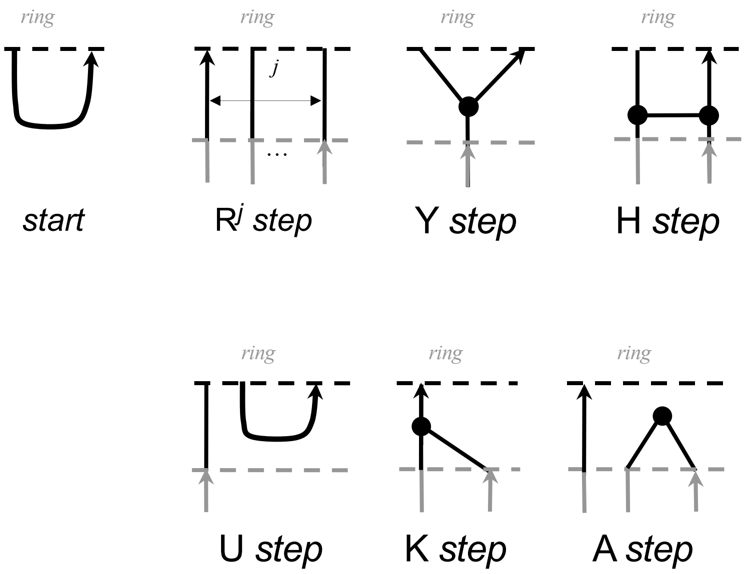
20,000,000 cases

Building a square

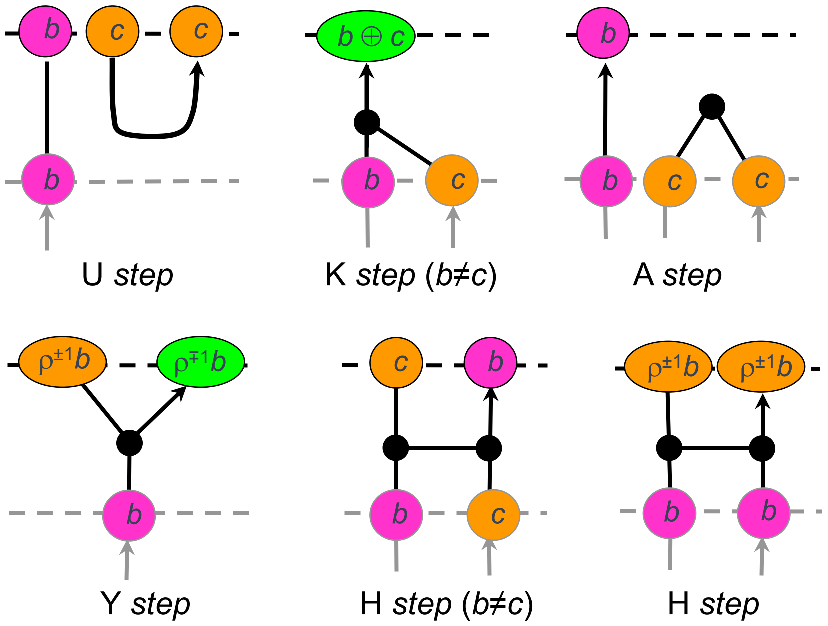
H R^3 Y Y



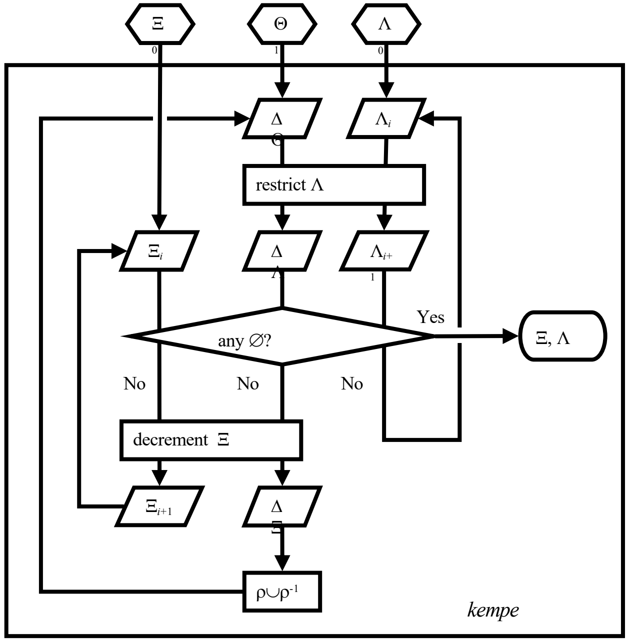
Building configurations



Colouring interpretation



Model checking colourings



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) $.
```

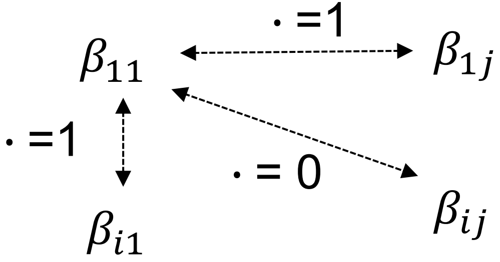
Coprime cycle coherence

In the Feit-Thompson proof of the Odd Order theorem, one need to extend an isometry σ from a submodule of the virtual characters of $W \subset G$ to those of G, to all characters of W.

Characters are traces of $\mathbb{C}$ matrix representations. Here $W \sim Z_{pq}$ so characters in W are pq-roots of unity.

A dot product matrix puzzle

The submodule is generated by 1 and the $\alpha_{ij} = \omega_{ij} - \omega_{i0} - \omega_{0j}$;
set $\beta_{ij} = \alpha_{ij}^{\sigma}$.



$\|\cdot\|^2$ =3

Is the obvious solution

$$\beta_{ij} = \chi_{ij} + \chi_{i0} + \chi_{0j}$$

unique?

An integer norm problem

Infer integral vectors from dot product

Naïve SMT fails

Naïve SAT bit-blast fails

Tailored SAT encoding longish

x_1 y_1 z_{11}	x_1 y_2 z_{12}
x_2 y_1 z_{11}	x_2 y_2 z_{22}
x_3 y_1 z_{31}	x_3 y_2 z_{32}
x_4 y_1 z_{11}	x_4 y_2 z_{42}

Sorting a combinatorial mess

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 $i = 1, 2, 3$,

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

We consider two cases:

Case I. There are indices i and i' such that $1 \le i < i' \le 3$ and

$$A_{i1} \cap A_{i'1} \cap A_{41} \ne \emptyset.$$

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

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

Proof. Suppose that $\chi_1 \in A_{12}$. By $\mathrm{O}(12, 21)$, $-\chi_4$ or $-\chi_5 \in A_{12}$. Suppose that $-\chi_4 \in A_{12}$. Then it follows from $\mathrm{O}(12, 31)$ and $\mathrm{L}(12, 11)$ that $\chi_6 \in A_{12}$, which contradicts $\mathrm{O}(12, 41)$. 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 $\mathrm{O}(32, 11)$, $-\chi_1$ or $-\chi_3 \in A_{32}$. Suppose that $-\chi_1 \in A_{32}$. Then, by $\mathrm{O}(32, 21)$ and $\mathrm{O}(32, 41)$, 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 $\beta_{12} = \chi_2 - \chi_4 + \chi_5$.*

Proof. By (3.5.4.1), (3.5.4.2) and $\mathrm{L}(12, 32)$, χ_1 does not belong to A_{12}. By $\mathrm{L}(12, 11)$, χ_2 or $\chi_3 \in A_{12}$. But, by $\mathrm{L}(12, 32)$ and (3.5.4.2), $\chi_3 \notin A_{12}$. Thus $\chi_2 \in A_{12}$. By $\mathrm{O}(12, 31)$, $-\chi_4$ 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 $\chi_5 \in A_{12}$ by $\mathrm{O}(12, 21)$. □

(3.5.4.4) $\beta_{22} = \chi_5 + \chi_8 + \chi_9$.

Proof. By (3.5.4.1), (3.5.4.3) and $\mathrm{L}(22, 12)$, χ_1 does not belong to A_{22}. By $\mathrm{L}(22, 21)$, χ_4 or $\chi_5 \in A_{22}$. Then $\mathrm{L}(22, 12)$ shows that $\chi_5 \in A_{22}$. By $\mathrm{L}(22, 12)$, $\chi_2 \notin A_{22}$ and so $\mathrm{L}(22, 32)$ implies that $-\chi_3$ or $\chi_8 \in A_{22}$. But, by $\mathrm{O}(22, 11)$, $-\chi_3 \notin A_{22}$ 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 $\mathrm{L}(42, 12)$, χ_1 does not belong to A_{42}. Suppose that χ_2 or $-\chi_3 \in A_{42}$. By $\mathrm{O}(42, 11)$, χ_2 and $-\chi_3 \in A_{42}$, which, with (3.5.4.2)

Sorting a combinatorial mess

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 $i \ne i'$, $j \ne j'$.*

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 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 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 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 $i = 1, 2, 3$,

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

We consider two cases:

Case I. There are indic

Case II. For $1 \le i < i' <$

Suppose that Case I hol

(3.5.4.1)

If $\chi_1 \in A_{12}$, then $\beta_{12} =$

If $\chi_1 \in A_{22}$, then $\beta_{22} =$

If $\chi_1 \in A_{42}$, then $\beta_{42} =$

Proof. Suppose that $\chi_1 \in$ $-\chi_4 \in A_{12}$. Then it follows contradicts O(12, 41). Thu interchanging the roles of

The other two assertion β_{41}.

(3.5.4.2) We may assume

Proof. By the symmetry assume that $A_{32} \cap A_{31} =$ that $-\chi_1 \in A_{32}$. Then, by in $A_{21} \cap A_{41}$, which is a co of A_{32} cannot be one of th

(3.5.4.3) We may assume

Proof. By (3.5.4.1), (3.5. L(12, 11), χ_2 or $\chi_3 \in A_{12}$ $\chi_2 \in A_{12}$. By O(12, 31), functions β_{21} and β_{41}, we O(12, 21).

(3.5.4.4) $\beta_{22} = \chi_5 + \chi_8 +$

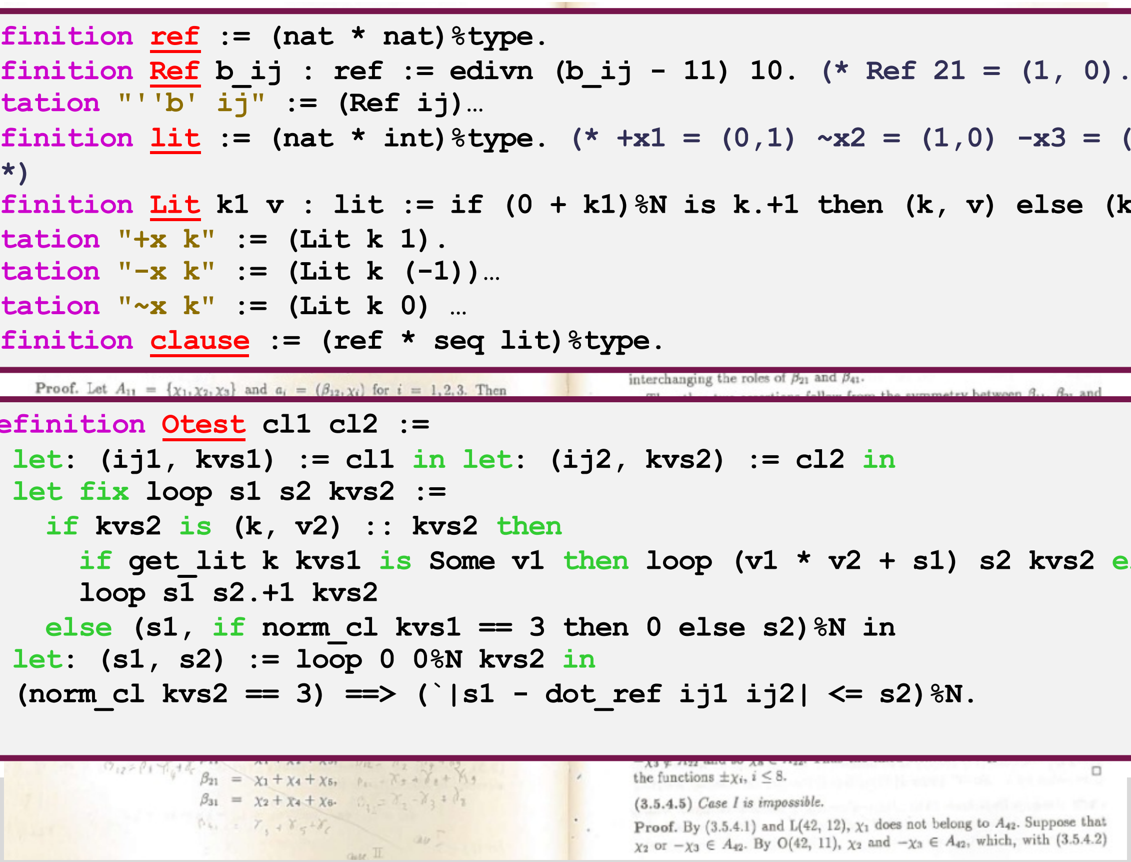
Proof. By (3.5.4.1), (3.5. L(22, 21), χ_4 or $\chi_5 \in A_{22}$. $\chi_2 \notin A_{22}$ and so L(22, 32) $-\chi_3 \notin A_{22}$ and so $\chi_8 \in A$ the functions $\pm\chi_i$, $i \le 8$.

(3.5.4.5) Case I is impos

Proof. By (3.5.4.1) and I χ_2 or $-\chi_3 \in A_{42}$. By O(

```
Let unsat_Ii : unsat |= & x1 in b11 & x1 in b21 & ~x1 in b31.
Proof.
uwlog Db11: (& b11 = x1 + x2 + x3) by do 2!fill b11.
uwlog Db21: (& b21 = x1 + x4 + x5).
  by uhave ~x2, ~x3 in b21 as L(21, 11); do 2!fill b21; uexact Db21.
uwlog 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).
uwlog 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
```

Sorting a combinatorial mess


Proof. Let $A_{11} = \{\chi_1, \chi_2, \chi_3\}$ and $a_i = (\beta_{i2}, \chi_i)$ for $i = 1, 2, 3$. Then

$\beta_{21} = \chi_1 + \chi_4 + \chi_5$,

$\beta_{31} = \chi_2 + \chi_4 + \chi_6$.

interchanging the roles of β_{21} and β_{41}.

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)

```
Definition ref := (nat * nat)%type.
Definition Ref b_ij : ref := edivn (b_ij - 11) 10. (* Ref 21 = (1, 0). *)
Notation "''b' ij" := (Ref ij)…
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).
Notation "-x k" := (Lit k (-1))…
Notation "~x k" := (Lit k 0) …
Definition clause := (ref * seq lit)%type.
```

```
Definition Otest cl1 cl2 :=
  let: (ij1, kvs1) := cl1 in let: (ij2, kvs2) := cl2 in
  let fix loop s1 s2 kvs2 :=
    if kvs2 is (k, v2) :: kvs2 then
      if get_lit k kvs1 is Some v1 then loop (v1 * v2 + s1) s2 kvs2 else
      loop s1 s2.+1 kvs2
    else (s1, if norm_cl kvs1 == 3 then 0 else s2)%N in
  let: (s1, s2) := loop 0 0%N kvs2 in
  (norm_cl kvs2 == 3) ==> (`|s1 - dot_ref ij1 ij2| <= s2)%N.
```

Conclusions

- **Computers systems can be taught maths in practice**
- **Doing computer proofs can teach us new mathematical ideas**
- **… and new presentations of mathematical concepts**

- **Questions ?**