Proof engineering, from the Four Color to the Odd Order Theorem

Georges Gonthier
Microsoft Research Cambridge
An old puzzle’s story

Four colours suffice

proof text
1 submap

1 configuration

calculations
3 colourings

Heawood 1890

De Morgan

publication 1878

Guthrie 1852

Kempe 1879
Saved by the computer?

Four colours suffice

Coq formal proof text
10,000 submaps
1,000,000,000 colourings

Four colours suffice

#sides < 6

1 configuration

proof text

Coq program proof
10,000 submaps
1,500 configurations

Robertson, Sanders, Seymour, & Thomas 1995

Appel & Haken 1976

Gonthier & Werner 2004

Diagram showing the process of formal verification with Coq programs.
Early lessons

• It is possible to build rigorously self-certifying program/proofs.
  – *proof by computation is feasible.*

• A computer proof assistant can be used to explore the logical structure of a proof.
  – *new math can be gleaned from a formalization.*

• Software Engineering *matters* in formal proofs.
  – old rules and *new techniques.*
Coloring by induction

reducible

configuration context
The whole proof

• Find a set of configurations such that:
  (A) **unavoidability**: At least one appears in any planar map.
  (B) **reducibility**: Each one can be coloured to match any planar ring colouring.
• Verify that the combinatorics fit the topology (graph theory + analysis).
The Poincaré principle

- How do you prove: \(2 + 2 = 4\) ?
- Given \(2 \overset{\text{def}}{=} 1 + (1+0)\)
  \[4 \overset{\text{def}}{=} 1 + (1 + (1 + (1 + 0)))\]
  \[n + m \overset{\text{def}}{=} \text{if } n \text{ is } 1 + n' \text{ then } 1 + (n' + m) \text{ else } m\]
  (a recursive program)
- \(a: \quad 0 + 2 = 2\) (neutral left)
- \(b: \quad (1 + 0) + 2 = 1 + (0 + 2)\) (associativity)
- \(c: \quad 2 + 2 = 1 + ((1 + 0) + 2)\) (def, associativity)
- \(d: \quad 2 + 2 = 1 + (1 + (0 + 2))\) (replace b in c)
- \(e: \quad \) (replace calculation, def)
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 33 37 H 2 H 13 Y 5 H 10 H 1 H 1 Y 3 H 11 Y 4 H 9 H 1 13 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
Describing a map

Euler: $\#\text{edge} + \#\text{node} + \#\text{face} = \#\text{dart} + 2 \times \#\text{comp}$
Group Theory

- The theory of invertible operators...
  - and of puzzles
- Due to Évariste Galois
  - \( x^5 + 3x^3 + 7 = 0 \)
- Explains quantum mechanics
- Crystallography, cryptography…
The Swiss army knife of Group Theory

- Theorem (Jordan-Hölder): *Any finite group factors uniquely into a series of simple groups*

- Theorem (Classification): *Finite simple groups belong to either one of 4 general classes, or one of 26 sporadic exceptions*
The Finite Group Challenge

The Classification of Finite Simple Groups

Frobenius groups
Thompson factorisation
character theory
linear representation
Galois theory
linear algebra
polynomials

Odd Order

$|G| \text{ odd}$

$G \text{ simple}$

$G \cong F_p$

Sylow theorems
canonical isomorphisms
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)
A mathematical library shelf

Section Lagrange.

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

Proof.
rewrite -[#|G|]sum1_card (partition_big_imset (rcoset H)) /=.
rewrite mulfC -sum_nat_const; apply: eq_bigr => __ /rcosetsP[x Gx ->].
rewrite -(card_rcoset _ x) -sum1_card; apply: eq_bigr => y.
rewrite rcosetE eqEcard mulGS !card_rcoset leqnn andbT.
by rewrite group_modr subset // inE.
Qed.

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

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

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

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

- Notation
- Exercises
- Interfaces
- Components
- Definitions
- Theorems
- Lemmas

Diagram:
- Notation and Exercises connected to Interfaces and Components.
- Definitions connected to Theorems and Lemmas.
- Interfaces and Components are central to the diagram, linking all other components.
Theorem \text{Ptype_embedding}:
\[
\forall M \in \mathbb{M}_P \setminus \kappa(M) \setminus \text{Hall}(M) : \exists M_\ast \in \mathbb{M}_P \setminus \text{Hall}(M) \setminus \kappa(M) \setminus \text{Hall}(M) \\
\text{such that } (M_\ast, \mathcal{C}_{\text{Hall}}(M_\ast)) \subseteq (M, \mathcal{C}_M)
\]
Demonstration

**Lemma mxtrace_mulC**  \( m \, n \)  \( (A : 'R[M](m, n)) \, B : \)  
\[ \text{tr} (A \times B) = \text{tr} (B \times A). \]

**Proof.**  
\begin{align*}
\text{tr} (A \times B) &= \sum_j \sum_i A_{i,j} \times B_{j,i} \\
&= \sum_j (BA)_{j,j} = \text{tr} BA
\end{align*}

**mxtrace_mulC is defined**  
\[ \text{tr} (A \times B) = \sum_j \sum_i A_{i,j} \times B_{j,i} \]
Formal mathematics

- Lemmas
- Functions

- implement
- Reflection

- Logic
- CiC/Coq

- infer types
- compute types

- Proof script
- SSReflect

- define types
- package computation

- invoke computation
- control computation
Algebraic notation

\[ \sum a_i x^i \quad \sum \left( \Phi(n/d) m^d \right) \quad \bigcap_{H < G \atop \text{H maximal}} \bigcap_{H < G} H \]

\[ \sum (\sigma) \prod A_{i, \sigma} \quad \bigcap H \quad \bigoplus V_i \quad V_i \approx W \]

Definition `determinant n (A : 'M_n) : R := \sum_(s : 'S_n) (-1)^s * \prod_i A_i (s_i).`
Definition \texttt{mxtrace} (R : \texttt{ringType}) n (A : \texttt{\textquotesingle M[R]_n}) :=
@\texttt{bigop} R \texttt{\textquotesingle I_n 0 +\%R \ (index_enum \_)}
\texttt{(fun i : \texttt{\textquotesingle I_n => fun_of_matrix A i i)}}
Algebra interfaces

- Equality
- Choice
- Zmodule
- Ring
- ComRing
- UnitRing
- ComUnitRing
- Lmodule
- Algebra
- ComUnitRing
- Falgebra
- Additive
- Linear
- Rmorphism
- Lrmorphism
- Vector
Definition \texttt{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

Equality
- \( x == y \)

Finite
- \( \{x_1, x_2, \ldots, x_n\} \)

bool
- \( \text{if } b \text{ then ...} \)

nat
- \( 0, n+1, \ldots \)

seq
- \( [::x_1; \ldots; x_n] \)

ordinal
- \( l_n, 0, \text{ord_max} \)
Ad hoc inference

Definition **mxtrace** \( (R : \text{ringType}) \ n \ (A : \ 'M[R]_n) \ := \) 
\[ @\text{bigop} \ R \ 'I_n \ 0 \ (@\text{Gring}\text{.add} \ (\text{Ring}\text{.ZmodType} \ R)) \]
\( (\text{index_enum} \ (\text{ordinal}\_\text{finType} \ n)) \)
\( (\text{fun} \ i : \ 'I_n \ => \ \text{fun}_\text{of}_\text{matrix} \ A \ i \ i) \)
Generic Lemmas

Pull, split, reindex, exchange ...

Lemma **bigD1** (I : finType) (j : I) P F :

\[ P \ j \ \rightarrow \ \big[\text{\*M/1}\big]_\text{(i | P i)} F \ i \]

\[ = \ F \ j \ \ast \ \big[\text{\*M/1}\big]_\text{(i | P i && (i != j))} F \ i \]

Lemma **big_split** I (r : list I) P F1 F2 :

\[ \big[\text{\*M/1}\big]_\text{(i <- r | P i)} (F1 \ i \ \ast \ F2 \ i) = \]

\[ \big[\text{\*M/1}\big]_\text{(i <- r | P i)} F1 \ i \ \ast \ \big[\text{\*M/1}\big]_\text{(i <- r | P i)} F2 \ i. \]

Lemma **reindex** (I J : finType) (h : J -> I) P F :

\{on P, bijective h\} ->

\[ \big[\text{\*M/1}\big]_\text{(i | P i)} F \ i = \ \big[\text{\*M/1}\big]_\text{(j | P (h j))} F \ (h j) \]

Lemma **bigA_distr_bigA** (I J : finType) F :

\[ \big[\text{\*M/1}\big]_\text{(i : I)} \ \big[\text{+%M/0}\big]_\text{(j : J)} F \ i \ j \]

\[ = \ \big[\text{+%M/0}\big]_\text{(f : {ffun I -> J})} \ \big[\text{\*M/1}\big]_\text{(i)} F \ i \ (f \ i). \]
Operator structures

Polymorphism for values!

Structure law : Type := Law {
    operator :> T -> T -> T;
    _ : associative operator;
    _ : left_id idx operator;
    _ : right_id idx operator
}.

Canonical addn_monoid := Monoid.Law addnA add0n addn0.
Canonical addn_abeloid := Monoid.ComLaw addnC.
Canonical muln_monoid := Monoid.Law mulnA muln1 muln1.

Structure com_law : Type := AbelianLaw {
    com_operator :> law;
    _ : commutative com_operator
}.

Canonical ring_add_monoid := Monoid.Law addrA add0r addr0.
Canonical ring_add_abeloid := Monoid.ComLaw addrC.

...
Interfacing big operators

Equality

Finite

bool

nat

'\l_n

seq

Monoid.law

Monoid.com_law

Monoid.add_law

Zmodule

Ring

ComRing

bigop

\bigoplus_{i \leftarrow r \& P(i)} E_i
More mathematical components…

- Finite group theory: morphisms, actions, characteristic & functor subgroups, p-groups, Frobenius & extremal groups…
- Character theory, representation and module theory, vector geometry.
- Finite field and Galois theory, algebraic number theory.
- Linear algebra, matrix rank.
Linear algebra interface?

Matrices

- compute
- shape

Vector spaces

- row spaces
- kernels
- coordinates
- bases

- aggregate
- dimension

group representation

\[ \Xi : G \rightarrow M_n(\mathbb{C}) \]

group character

\[ \chi = \text{tr} \, \Xi : G \rightarrow \mathbb{C} \]
Notation abuse

In math:
\[ S = A + \sum_i B_i \text{ is direct} \]
iff rank \( S = \text{rank } A + \sum_i \text{rank } B_i \)

In Coq:

\begin{verbatim}
Lemma mxdirectP n (E : mxsum_expr n) :
    reflect (\rank E = mxsum_rank E) (mxdirect E).
\end{verbatim}

This is generic in the shape of \( E \)

Let \( \text{sumV} := (\sum_{i < h} V_i) \%\text{MS} \).

(* This is B & G, Proposition 2.4(a) *)

Lemma mxdirect_sum_eigenspace_cycle :
    (sumV :=: 1%M) %\text{MS} /\ mxdirect sumV.
Recurrences

B. The Puig Subgroup

Proof.
Again we use induction for (a). For \( n = 0 \) we know (a) is true by hypothesis. Now suppose that \( n > 0 \) and \( L(G) \). Then

\[ L(G) \to L_{2n+1}(H). \]

Hence

\[ L_{2n+1}(H) \subseteq L_{2n}(L(G)) = L_{2n}(G). \]

Furthermore,

\[ L_{2n+1}(H) = L_{2n}(L(G)) = L(G) \subseteq H. \]

Thus

\[ L(G) \subseteq L_{2n+1}(H). \]

Again, (b) follows from Lemma B.1(c). 

By Step 1 and Step 2 we can now conclude that \( L(G) \).

Let \( \mu \) be a Sylow \( p \)-subgroup of \( G \) and suppose that \( S \) is a Sylow \( p \)-subgroup of \( G \) and \( T = \mu S(G) \).

Now \( L_{2n+1}(T) \) is a normal subgroup of \( G \) and \( \mu S(G) \).

By Lemma B.2 and Theorem A.5, \( \mu S(G) \).

Hence, by (B.2),

\[ L_{2n+1}(S) \subseteq L(T_{2n+1}(T)). \]

(\( B.1 \)) holds for some \( n \). Since \( L_{2n+1}(S) = L_{2n+1}(T) \).

Now \( L_{2n+1}(T) \) is a normal subgroup of \( G \) and, by Lemma B.2,

\[ L_{2n+1}(T) = \mathcal{C}_T(L_{2n+1}(T)). \]

Thus, by (B.2) and Theorem A.5, \( \mu S(G) \).

\[ L_{2n+1}(S) \subseteq T. \]

which is trivial.

Assume \( (B.1) \) holds for some \( n \). Since \( L_{2n+1}(S) = L_{2n+1}(T) \).

Therefore, by Lemma B.2 and Theorem A.5, \( \mu S(G) \).

\[ L_{2n+1}(S) \subseteq T. \]

(\( B.2 \)) holds for some \( n \). Since \( L_{2n+1}(S) = L_{2n+1}(T) \).

Now \( L_{2n+1}(T) \) is a normal subgroup of \( G \) and, by Lemma B.2,

\[ L_{2n+1}(T) = \mathcal{C}_T(L_{2n+1}(T)). \]

Thus, by (B.2) and Theorem A.5, \( \mu S(G) \).

\[ L_{2n+1}(S) \subseteq T. \]

(\( B.3 \)) holds for some \( n \). Since \( L_{2n+1}(S) = L_{2n+1}(T) \).

Consequently, by Lemma B.1(a),

\[ L_{2n+1}(S) = \mathcal{C}_T(L_{2n+1}(T)). \]

By Lemma B.1(b),

\[ L_{2n+1}(S) \subseteq T. \]

For \( n = 0 \) the statement reduces to

\[ 1 \subseteq 1 \subseteq T \subseteq S, \]

which is trivial.

Assume \( (B.1) \) holds for some \( n \). Since \( L_{2n+1}(S) = L_{2n+1}(T) \).

Therefore, by Lemma B.2 and Theorem A.5, \( \mu S(G) \).

\[ L_{2n+1}(S) \subseteq T. \]

(\( B.4 \)) holds for some \( n \). Since \( L_{2n+1}(S) = L_{2n+1}(T) \).

By Lemma B.1(b),

\[ L_{2n+1}(S) \subseteq T. \]
Telescopic algebra

have [[U_a U1 U2 P0s1 Dusv1]] /sus_modP-Duv1 := (usv1P, usv1P).
have [[U _ U2 Uv2 P0s2 _]] [U_b U3 Uv3 P0s3 _]] := (usv2P, usv3P).
suffices : (congrl sigma): s ^+ 2 = s ^* v1 ^* s ^* a ^* -1 ^* t ^+ 3.
  rewrite inl sigmaX : sigma_s sigmaM ?memj_p -?psiE ?nUt1n => => ->.
    by rewrite addrK -!inlpsi !mem_imset ?nUt2n.
rewrite groupV in Ua; have [Hsl Hs3]: s1 \in H \; s3 \in H by rewrite !sD0H.
rewrite nt_s1: s1 := 1 by apply: nt_sUs usv1P.
have nt_s3: s3 := 1 by apply: nt_sUs usv3P.
have sUsxp Dsp: s2def (w1 ^+ p) (w2 ^+ p) (w3 ^+ p).
rewrite !^=^~ conjxg _ _ p, expUmp ?grouplV ?l[t]exppl ?nUt2n ?nUt3n //.
apply: ds2 usv1p usv2p usv3p => //.
    by rewrite !psiX // !Frobenius_autoE -!morphismD Dab rmorphf nat.
have[Da2] Ds2: s2def w1 w2 w3 by apply: Ds2 usv1P usv2D usv3P.
wlog [Uw1 Uw2 Uw3]: w1 w2 w3 Dsp Ds2 / [/w1 \in U, w2 \in U & w3 \in U].
    by move/\_ w1 w2 w3 => ; rewrite ?(nUt2n, nUt3n 18N, nUt3n 18N, in_group).
have[Da2p] Dwp: (w2 ^- p ^* w1 ^- p ^- p ^- s3 ^* w2) ^ t ^+ 2 = w3 ^* p ^- 1 ^* s1 ^- l.
  rewrite ![w1 ^+ _] (mulKg w1) ![w3 ^+ _] (mulKg w3) !expgs !expgsr !predK //.
  rewrite -(canlr (mulKg _ _) Dsp) -(canlr (mulKg _ _) Da2) 6!invvG !invvG.
    by rewrite mulAg mulKg [2]lock /conjg !mulAg mulVg mulGl mulGk.
  have w_id w: w \in U => w ^+ p ^- 1 == 1 => w = 1.
    by move=> Uw /eqF/(canRL in (expkg _ _)) Uw => ; rewrite !expgln ?cU.
have[Uw3] Dwp: w3 = 1.
  apply: w_id => //; have:= @not_splitU s1 ^* s1 ^* s1 ^* (w3 ^* p ^- 1).
  rewrite ![grouplV mulVg eqxx andE] (2)invvK (negFf nt_s1) groupX // => => //.
  have /tih D1 <=: t ^+ 2 \in D1^*.
    by rewrite oddgt2 ?order_gt1 // orderE defp0 (odd2g sdefp0).
by rewrite -mulAg -conjgE inE -(2)Dwp mamJ conjg !in_group ?Hs1 // sUh.
have[Dwp] Dwp: w2 ^+ p ^- 1 = w3 ^- ^- p ^- 1 ^* s3.
  apply ![mulKg w2] /eqF; rewrite !expgsr !predK // eq_mulVg1 mulAg.
by rewrite (canRL (conjgK _ _) Dwp) Dwp expgln !conjg1.
have[Uw1] Dwl: w1 = 1.
  apply: w_id => //; have:= @not_splitU s3 ^* s3 ^* (w1 ^* p ^- 1).
  rewrite mulVg (negFf nt_s3) andB -mulAg -conjgE -Dwp !in_group //.
    by rewrite eqxx andB eq_invv1 /= => => .
have[w1 w2 w3 Dw1 Dw3 w_id Uw2 Dw2p Ds2] Ds2: t ^* s2 ^* t ^* s3 ^* t ^+ 2 ^* s1.
Proof by reflection

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

**Proof of (3.5).**

(3.5.1) Let $\beta_0 = \sum_{i,j} \beta_{ij} x_{ij} - 1 \leq i \leq w_1, 1 \leq j \leq w_2$. Then $\beta_{ij} = 0$ and $\|\beta_j\|^2 = 0$ for all $i, j$, where $\beta_{ij} = 0$ and $\beta_{ij} = 0$ for all $i \neq i, j \neq j$.

**Proof.** That $\|\beta_{ij}\|^2 = 0$ follows from Frobenius reciprocity, and so $\beta_{ij} = 0$. The other relations follow from the fact that $\sum_{i,j} \beta_{ij} w_{ij} = 0$.

Let $1 \leq i \leq w_1, 1 \leq j \leq w_2$. By (3.3.1) and the fact that $\beta_{ij} = 0$, we see that $\beta_{ij} = \sum_{i,j} \alpha_{ij} x_{ij}$, where $\alpha_{ij}$ is a set of three pairwise orthogonal elements of $(\mathbb{R}(G) - \{1\})$.

(3.5.2) We have $\|\beta_{ij}\|^2 = 0$ and $\sum_{i,j} \alpha_{ij} = 0$.

**Proof.** Let $\beta_{ij} = \sum_{i,j} \alpha_{ij} x_{ij}$ and $\alpha_{ij} = (\beta_{ij} x_{ij})$ for $i, j \in [1, 2]$. Then $\beta_{ij} = \sum_{i,j} \alpha_{ij} x_{ij} = 0$ for $i \neq i, j \neq j$. The numbers $\alpha_{ij}$ are thus either 0, 0, or 1, 1, 1. If $\alpha_{ij}$ is 0, 0, or 1, 1, 1, 1, then we may assume that $\beta_{ij} = x_{ij} + x_{jl} - x_{ij}$, where $\beta_{il} = \beta_{jl} = 0$. By induction, $\beta_{ij} = 0$.

**Lemma (3.5.2) clearly holds.**

By Hypothesis (3.5), $\sum_{i,j} \alpha_{ij} x_{ij} = 0$. By the symmetry between $\alpha_{ii}$ and $\alpha_{ij}$, we assume

(3.5.3) $\alpha_{ii} \geq 0$.

In the proof which follows, the functions $x_i$ and $x_{ij}$ are pairwise orthogonal elements of $(\mathbb{R}(G) - \{1\})$.

(3.5.4) $\sum_{i,j} \alpha_{ij} x_{ij} = 0$.

**Proof.** Suppose that (3.5.4) is false. By (3.5.2), we can then write, for some choice of indices $i, j, k$,

\[ \beta_{ij} = x_{ij} + x_{ik} - x_{ij}, \]
\[ \beta_{jk} = x_{jk} + x_{ik} - x_{jk}, \]
\[ \beta_{kl} = x_{kl} + x_{ik} - x_{kl}, \]
\[ \beta_{ik} = x_{ik} + x_{ij} - x_{ik}. \]

By induction, $\beta_{ij} = 0$. Thus, $\sum_{i,j} \alpha_{ij} = 0$, which contradicts (3.5.4).

(by unmet theorems)

**Proof.** Suppose that (3.5.4) is false. By (3.5.2), we can then write, for some choice of indices $i, j, k$,

\[ \beta_{ij} = x_{ij} + x_{ik} - x_{ij}, \]
\[ \beta_{jk} = x_{jk} + x_{ik} - x_{jk}, \]
\[ \beta_{kl} = x_{kl} + x_{ik} - x_{kl}, \]
\[ \beta_{ik} = x_{ik} + x_{ij} - x_{ik}. \]

By induction, $\beta_{ij} = 0$. Thus, $\sum_{i,j} \alpha_{ij} = 0$, which contradicts (3.5.4).

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)

(by unmet theorems)
Wandering typo

- B & G 15.7
  - (e)(2) $p = |X|$ is a prime in $\sigma(M) - \beta(M)$, $O_p(H)$ is not abelian, $O_p'(H)$ is cyclic, ...

- Theorem 15.7. Suppose $F(M)$ is not a TI-subgroup of $G$. Let $H = M_F$ and choose $g \in G - M$ such that $X = F(M) \cap F(M)^g$ is not trivial. Take $E, E_1, E_2, E_3$ as in Sections 12-13. Then
  - (a) $M \in \mathcal{M}_\mathcal{G} \cup \mathcal{M}_{\mathcal{G}_1}$ and $H = M_\sigma$,
  - (b) $X \subseteq H$ and $X$ is cyclic,
  - (c) $M' \cong F(M) = M_\sigma \times O_{\sigma(M)}(F(M))$,
  - (d) $E_3 = 1, E_2 \triangleleft E$, and $E/E_2 \cong E_1$, which is cyclic, and
  - (e) one of the following conditions holds:
    - (1) $M \in \mathcal{M}_\mathcal{G}$ and $H$ is abelian of rank two,
    - (2) $p = |X|$ is a prime in $\sigma(M) - \beta(M)$, $O_p(H)$ is not abelian, $O_p'(H)$ is cyclic, and the exponent of $M/H$ divides $q - 1$ for every $q \in \pi(H)$,
    - (3) $p = |X|$ is a prime in $\sigma(M) - \beta(M)$, $O_p'(H)$ is cyclic, $O_p(H)$ has order $p^3$ and is not abelian, $M \in \mathcal{M}_{\mathcal{G}_1}$, and $|M/H|$ divides $p + 1$. 

- Peterfalvi (8.3) - (b) ... there is a prime divisor $p$ of $|H|$ such that $O_p'(M)$ is cyclic.
Things to look forward to

- Certification
  - of computer computations
  - of complex proofs
- Collaboration
  - safe contributions from diverse backgrounds
- Inspiration
  - explore logic, dependencies, and factoring