Formal Proof

Thomas C. Hales

January 2016

Formal Proof

A *formal proof* is a proof that has been verified from first principles. Generally speaking, formal proofs are constructed by computer.

Zermelo-Fraenkel set theory.

A *formal proof* is a proof that has been verified from first principles. Generally speaking, formal proofs are constructed by computer.

Zermelo-Fraenkel set theory. Morse-Kelly set theory. Von Neumann-Bernays-Gödel set theory. Tarski-Grothendieck set theory. Higher-Order Logic. Calculus of Inductive Constructions. Martin-Löf type theory. Homotopy Type theory.

Some Pseudo-history

During much of mathematical history, logic was unable to provide a comprehensive list of rules needed for mathematical reasoning.

Frege and others wrote down a complete list of logical inference rules that suffice for any mathematical deduction.

Russell's paradox in 1901 (and related paradoxes in set theory) showed there was a need for increased attention to foundations.

$$\text{If } \; A = \{x : x \notin x\}, \quad \text{then } (A \in A \Leftrightarrow A \notin A).$$

Response ZFC

Zermelo (1908) gave a list of axioms that most mathematicians now presume to be consistent.

Most mathematicians use ZFC as their foundational system in a vague way.

Thurston Three-Dimensional Geometry and Topology.

Problem 1.3.9. What are the possible values of the $i(X, z)$ when X is a general vector field? Can you find a formula for $i(X, z)$ valid for all vector fields with isolated zeros?

Proposition 1.3.10 (Poincaré index theorem). *If S is a smooth surface and X is a vector field on S with isolated zeros, the Euler number of S is*

$$\chi(S) = \sum_{z \in \text{zeros}} i(X, z).$$

Consequently, the Euler number of a surface does not depend on the cell division used to compute it—it is a topological invariant.

Proof of 1.3.10. Given a finite cell division of S, start by replacing it with a differentiable triangulation, as discussed just before Proposition 1.3.3. Subdivide and jiggle the triangulation as necessary to make all the zeros lie inside faces, no more than one to a face. Enclose each zero with a polygon contained in a face and transverse to the field, as explained in the paragraph preceding Lemma 1.3.5. Triangulate the annulus formed by taking away the polygon from the face (Exercise 1.3.6). Finally, make the rest of the triangulation transverse, again by using the technique in the proof of Problem 1.3.4.

Each polygon's contribution to the Euler number is the index of the vector field at the corresponding zero. Each triangle's contribution outside the polygons is zero. This proves the formula.

The last sentence of the proposition follows because every closed surface admits a vector field with isolated zeros (Exercise 1.3.8).

1.3.10

but what good is a tinkertoy set if the holes are all filled in with modeling clay?

In the present exposition, many of the "holes" or questions are explicitly labeled as exercises, questions, or problems. *Most of these are not walking-the-dog exercises* where the dog follows behind on a leash until the awaited event. You may or may not be able to answer the questions, even if you completely understand the text. Some of the questions form connections with ideas discussed more fully later on. Other questions have to do with details that otherwise would have been "left as an exercise for the reader." Still others relate the material under discussion to topics which are neither discussed nor assumed in the main text.

It is important to read through and think about the exercises, questions and problems. It should be possible to solve some of the more straightforward questions. But please don't be discouraged if you can't solve all, or even most, of the questions, any more than you are discouraged when you can't immediately answer questions which occur to you spontaneously.

The Rising Sea

Caution about foundational issues. We will not concern ourselves with subtle foundational issues (set-theoretic issues, universes, etc.). It is true that some people should be careful about these issues. But is that really how you want to live your life? (R. Vakil, “The Rising Sea” . Preface 0.3.1)

[Let $\mathcal{V}$ be the category whose objects are the k-vector spaces k^n for each $n \geq 0$ (there is one vector space for each n), and whose morphisms are linear transformations.]

Show that $\mathcal{V} \to f.d.Vec_k$ gives an equivalence of categories, by describing an "inverse" functor. (Recall that we are being cavalier about set-theoretic assumptions,. . . , To make this precise, you will need to use Gödel-Bernays set theory or else replace $f.d.Vec_k$ with a very similar small category, but we won't worry about this.) (Vakil, page 30, Exercise 1.2.D)

Bourbaki and Formal Mathematics

Many mathematicians associate Bourbaki with ultimate standards of rigor and formal mathematics. In fact, Bourbaki is extremely far from commonly accepted norms of formalized mathematics.

Bourbaki's *Theory of Sets* makes a complete mess of the foundations of set theory. Bourbaki declares that formalized mathematics is "absolutely unrealizable" and gives a list of reasons why.

- Formal proofs are too long.
- It is a burden to give up abuses of notation and abbreviations.
- Metamathematical arguments are useful.

Today all of these problems have been solved.

Response #2: Types

There was a second response to Russell's paradox. Russell himself introduced type theory as a way out of his paradox: $x \in x$ does not type check and hence is not a well-formed formula.

Church (1932) introduced the simply typed λ-calculus, which has had an enormous influence on the design of programming languages and type theory in computer science.

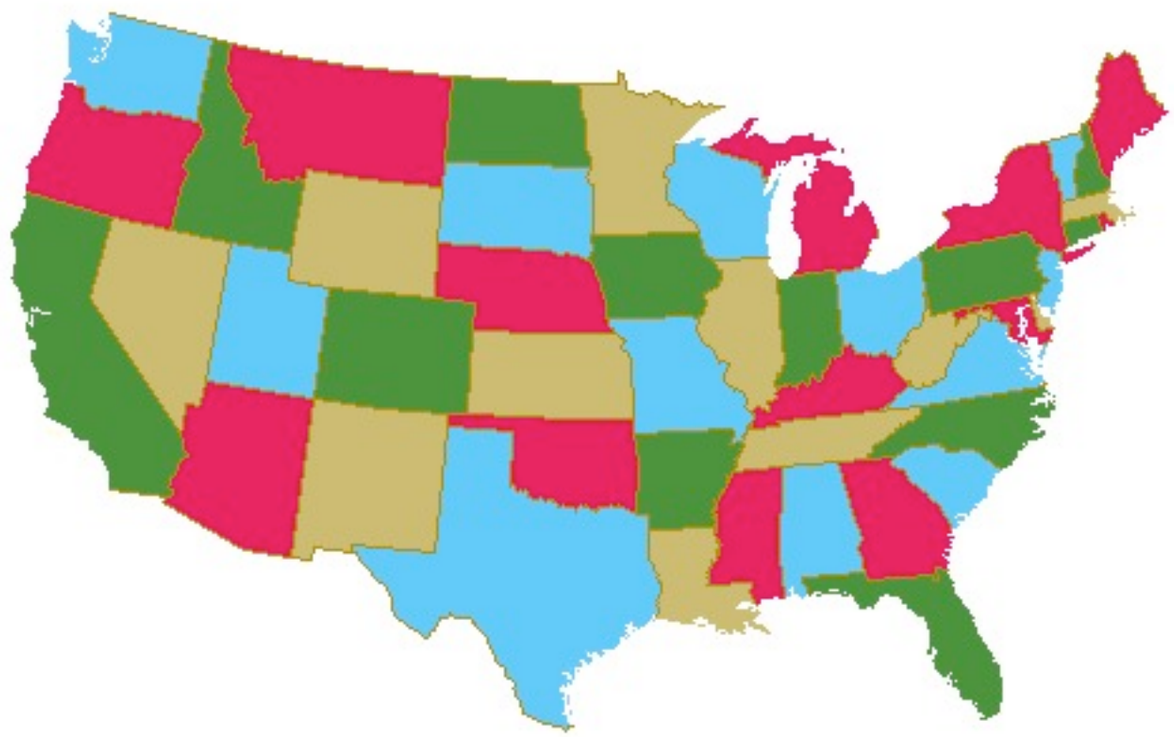
Dana Scott introduced LCF (logic of computable functions), which was implemented by Milnor in a computer. A functional programming language, ML, was designed as a proof-control language for LCF.

A descendant of LCF is the HOL (higher order logic) family of proof assistants. Many proof assistants (Coq, Isabelle/HOL, HOL Light) are implemented in various dialects of ML.

4CT

The Four-Color theorem was formalized in Coq [23].

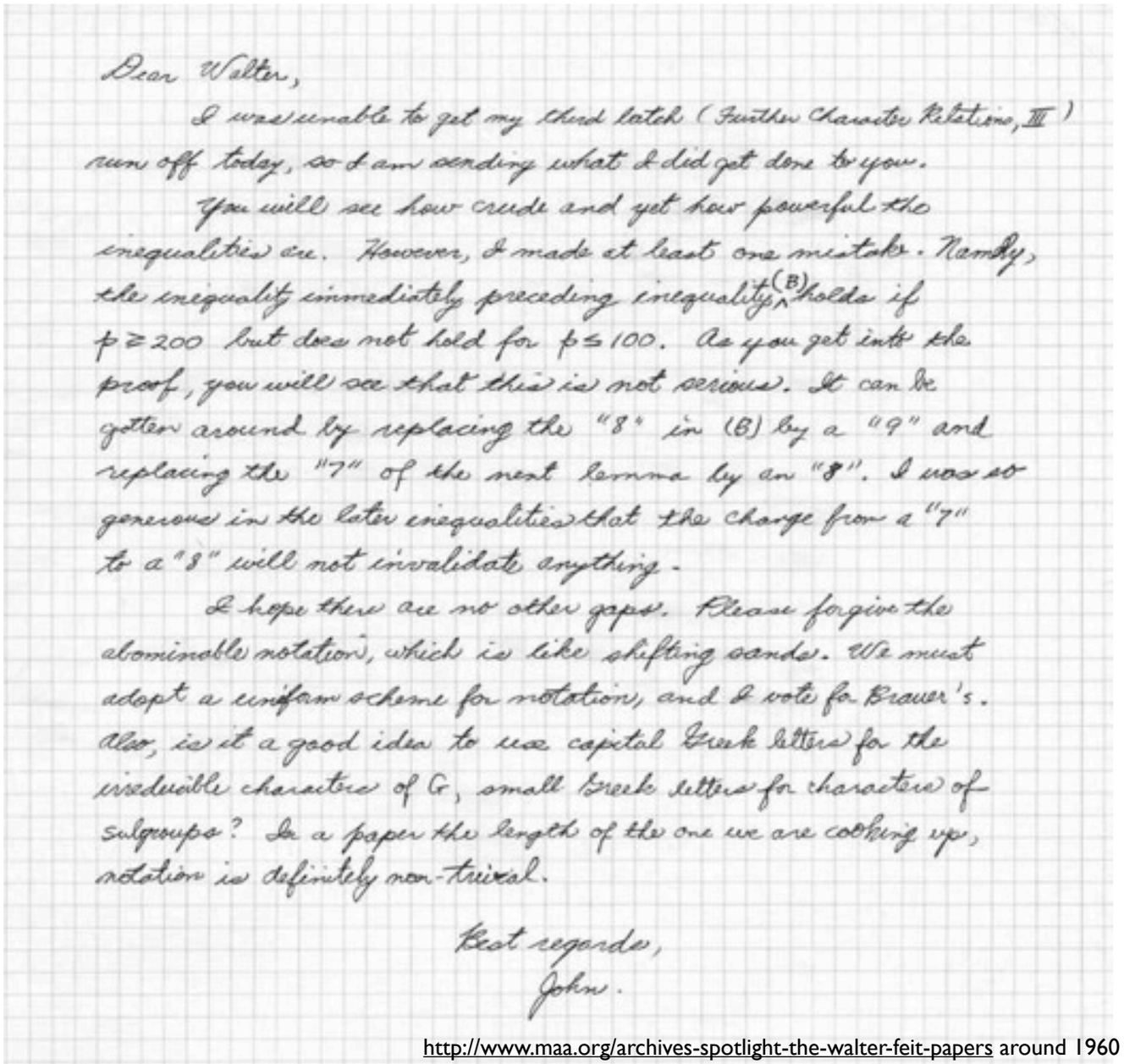
```
Variable R : real_model.
Theorem four_color : (m : (map R))
      (simple_map m) -> (map_colorable (4) m).
Proof.
    Exact (compactness_extension four_color_finite).
Qed.
```



Feit and Thompson published their famous theorem in 1963 [19].

THEOREM 2 (The Odd Order Theorem, Feit-Thompson). — *All finite groups of odd order are solvable.*

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



Dear Walter,

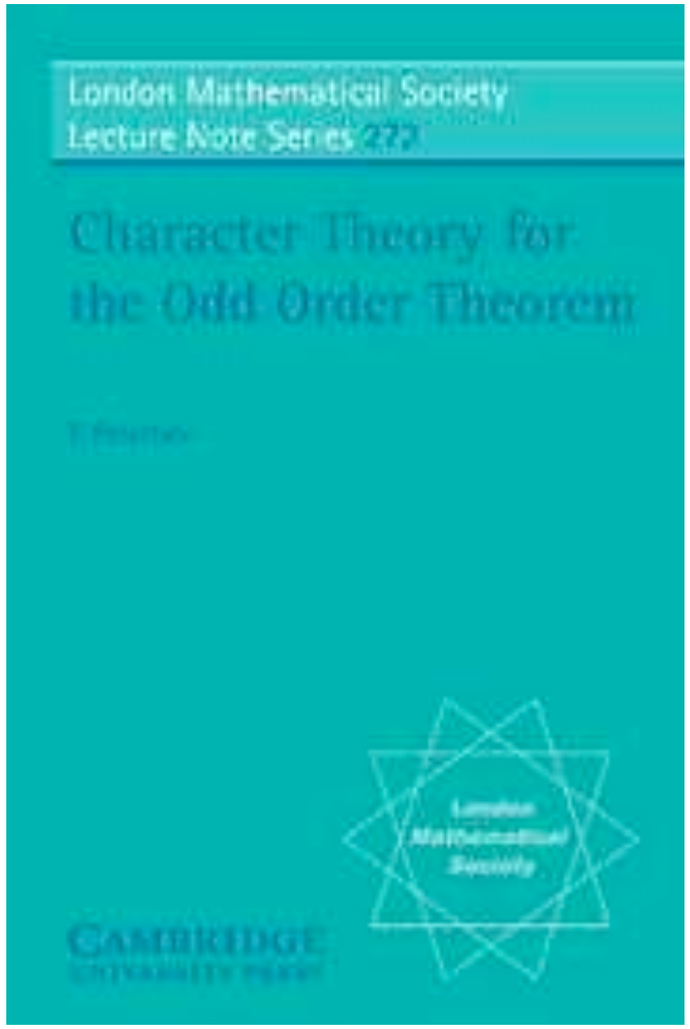
I was unable to get my third batch (Further Character Relations, III) run off today, so I am sending what I did get done to you.

You will see how crude and yet how powerful the inequalities are. However, I made at least one mistake. Namely, the inequality immediately preceding inequality (B) holds if $p \geq 200$ but does not hold for $p \leq 100$. As you get into the proof, you will see that this is not serious. It can be gotten around by replacing the "8" in (B) by a "9" and replacing the "7" of the next lemma by an "8". I was so generous in the later inequalities that the change from a "7" to a "8" will not invalidate anything.

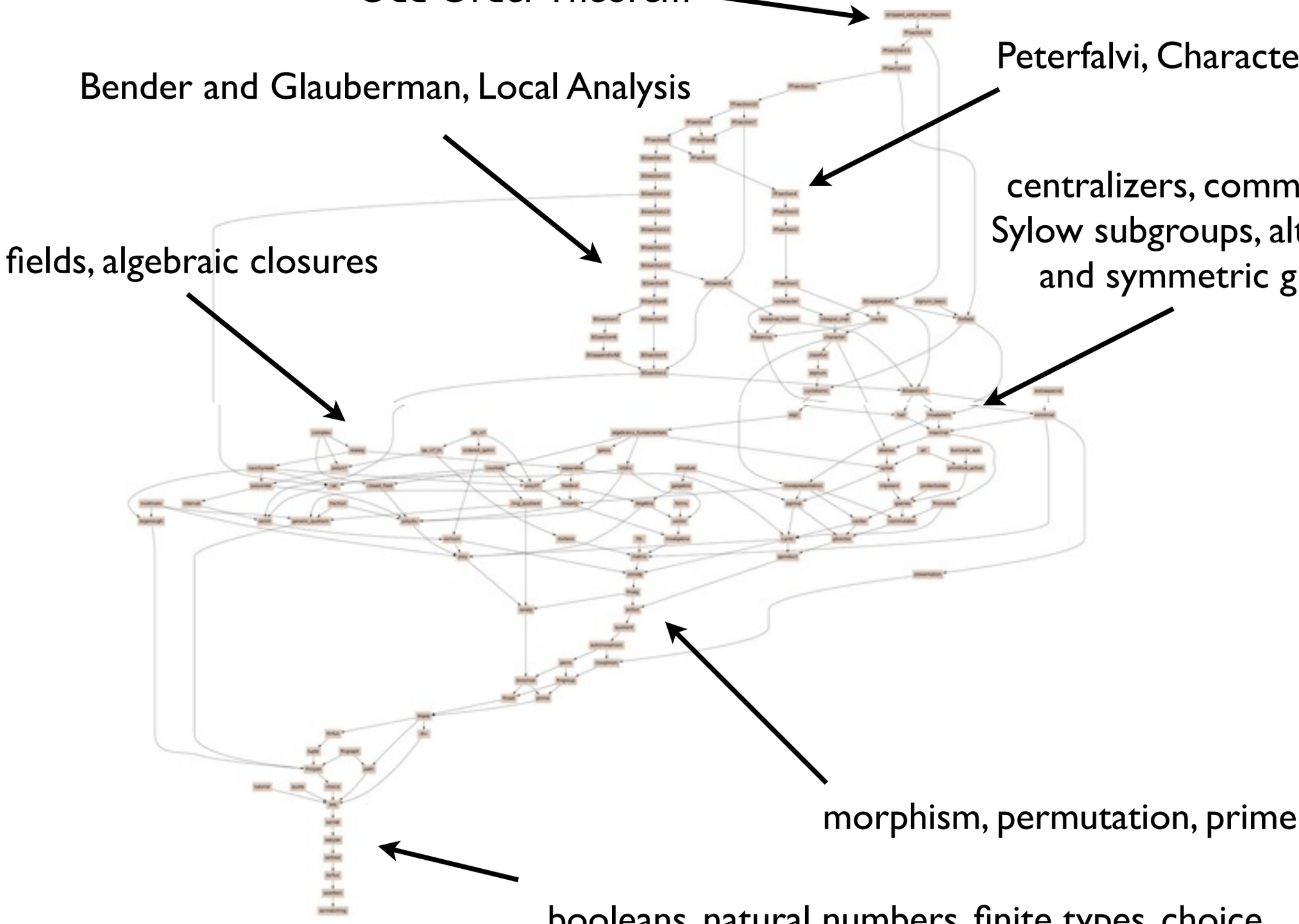
I hope there are no other gaps. Please forgive the abominable notation, which is like shifting sands. We must adopt a uniform scheme for notation, and I vote for Brauer's. Also, is it a good idea to use capital Greek letters for the irreducible characters of G, small Greek letters for characters of subgroups? In a paper the length of the one we are cooking up, notation is definitely non-trivial.

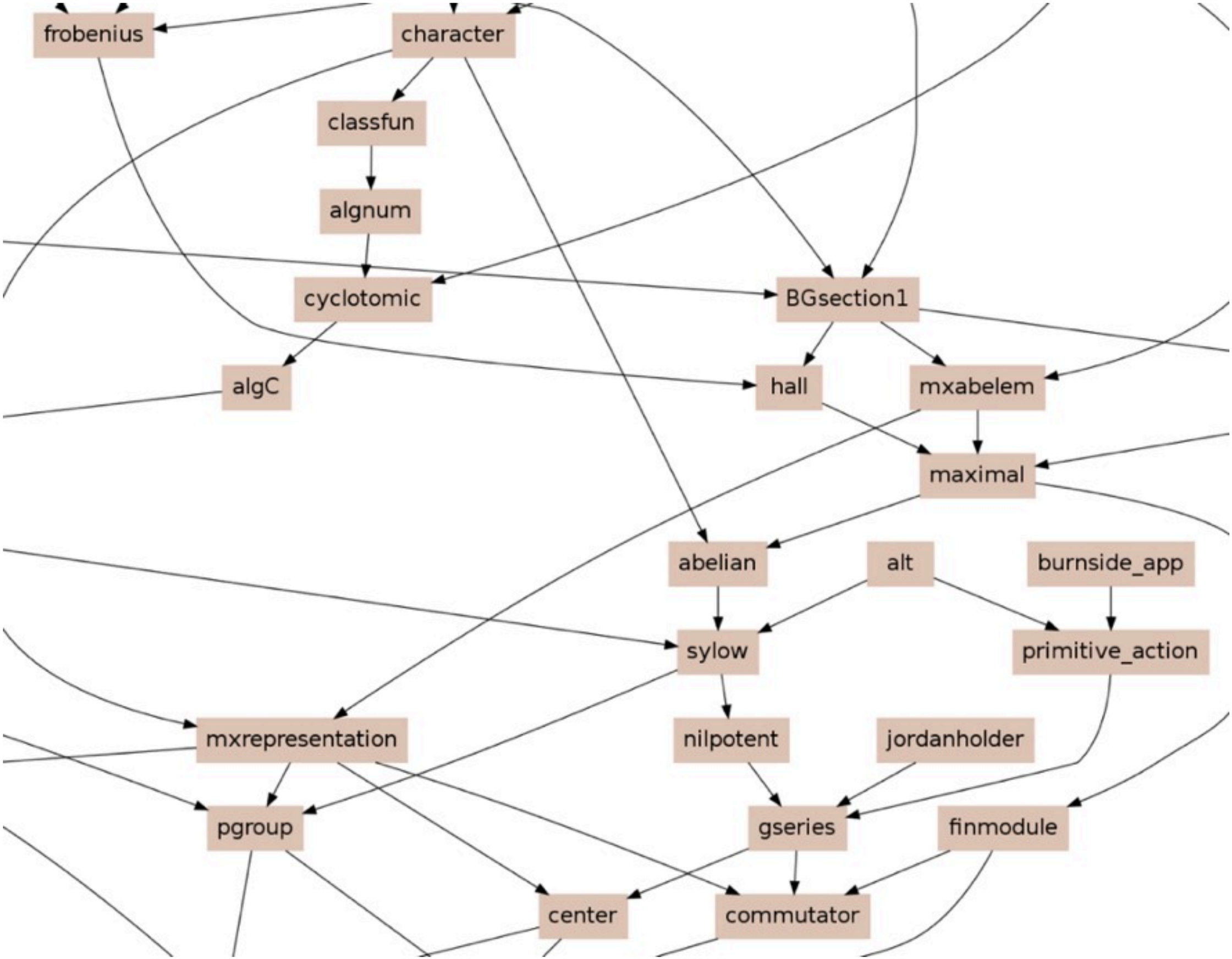
Best regards,
John.

http://www.maa.org/archives-spotlight-the-walter-feit-papers around 1960



Background material for the proof appears in textbooks *Finite Groups* [27], *Finite Group Theory* [3], and *Character Theory of Finite Groups* [37]. The necessary background includes a basic graduate-level understanding of rings, modules, linear and multilinear algebra (including direct sums, tensor products and determinants); fields, algebraic closures, and basic Galois theory; the structure theorems of Sylow and Hall; Jordan-Hölder; Wedderburn's structure theorem for semisimple algebras; representation theory with induced representations, Schur's lemma, Clifford theory, and Maschke's theorem; and character theory including Frobenius reciprocity, and orthogonality.


Odd Order Theorem
Peterfalvi, Character theory
Bender and Glauberman, Local Analysis
centralizers, commutators, Sylow subgroups, alternating and symmetric groups
fields, algebraic closures
morphism, permutation, prime
booleans, natural numbers, finite types, choice


frobenius
character
classfun
algnum
cyclotomic
BGsection1
algC
hall
mxabelem
maximal
abelian
alt
burnside_app
sylow
primitive_action
mxrepresentation
nilpotent
jordanholder
pgroup
gseries
finmodule
center
commutator

Library hall

```
(* (c) Copyright Microsoft Corporation and Inria. All rights reserved. *)
Require Import ssreflect ssrbool ssrfun eqtype ssrnat seq div fintype finset.
Require Import prime fingroup morphism automorphism quotient action gproduct.
Require Import commutator center pgroup finmodule nilpotent sylow.
Require Import abelian maximal.
```

In this files we prove the Schur-Zassenhaus splitting and transitivity theorems (under solvability assumptions), then derive P. Hall's generalization of Sylow's theorem to solvable groups and its corollaries, in particular the theory of coprime action. We develop both the theory of coprime action of a solvable group on Sylow subgroups (as in Aschbacher 18.7), and that of coprime action on Hall subgroups of a solvable group as per B & G, Proposition 1.5; however we only support external group action (as opposed to internal action by conjugation) for the latter case because it is much harder to apply in practice.

```
Set Implicit Arguments.

Import GroupScope.

Section Hall.

Implicit Type gT : finGroupType.

Theorem SchurZassenhaus_split gT (G H : {group gT}) :
  Hall G H → H <| G → [splits G, over H].

Theorem SchurZassenhaus_trans_sol gT (H K K1 : {group gT}) :
    solvable H → K \subset 'N(H) → K1 \subset H × K →
    coprime #|H| #|K| → #|K1| = #|K| →
  exists2 x, x \in H & K1 :=: K :^ x.

Lemma SchurZassenhaus_trans_actsol gT (G A B : {group gT}) :
    solvable A → A \subset 'N(G) → B \subset A <*> G →
    coprime #|G| #|A| → #|A| = #|B| →
  exists2 x, x \in G & B :=: A :^ x.

Lemma Hall_exists_subJ pi gT (G : {group gT}) :
  solvable G → exists2 H : {group gT}, pi.-Hall(G) H
                & ∀ K : {group gT}, K \subset G → pi.-group K →
                  exists2 x, x \in G & K \subset H :^ x.

End Hall.

Section HallCorollaries.

Variable gT : finGroupType.

Corollary Hall_exists pi (G : {group gT}) :
  solvable G → ∃ H : {group gT}, pi.-Hall(G) H.

Corollary Hall_trans pi (G H1 H2 : {group gT}) :
```

Computer Algebra Systems versus Algebra in Proof Assistants

This library for the Odd Order Theorem differs in a fundamental way from a computer algebra systems such as Sage, Mathematica, Maple, Gap, Singular, and Macaulay2.

Computer algebra systems are not grounded in the foundations of mathematics. Algorithms are not formally verified.

In a formal proof, every step is verified.

Prime Number Theorem

The elementary proof of the Prime Number Theorem by Erdös and Selberg was formalized in Isabelle [6]. The analytic proof by Hadamard and de la Vallée Poussin was formalized in HOL Light [33]. In the statement that follows, the symbol `&` denotes the function embedding the natural numbers into the real numbers.

```
// Prime Number Theorem:
((\n. &(CARD {p | prime p ∧ p <= n}) / (&n / log(&n)))
   ---> &1) sequentially
```

	2	3	5	7	11	13	17	19	23
29	31	37	41	43	47	53	59	61	67
71	73	79	83	89	97	101	103	107	109
113	127	131	137	139	149	151	157	163	167
173	179	181	191	193	197	199	211	223	227
229	233	239	241	251	257	263	269	271	277
281	283	293	307	311	313	317	331	337	347
349	353	359	367	373	379	383	389	397	401
409	419	421	431	433	439	443	449	457	461
463	467	479	487	491	499	503	509	521	523
541	547	557	563	569	571	577	587	593	599
601	607	613	617	619	631	641	643	647	653
659	661	673	677	683	691	701	709	719	727
733	739	743	751	757	761	769	773	787	797
809	811	821	823	827	829	839	853	857	859
863	877	881	883	887	907	911	919	929	937
941	947	953	967	971	977	983	991	997	

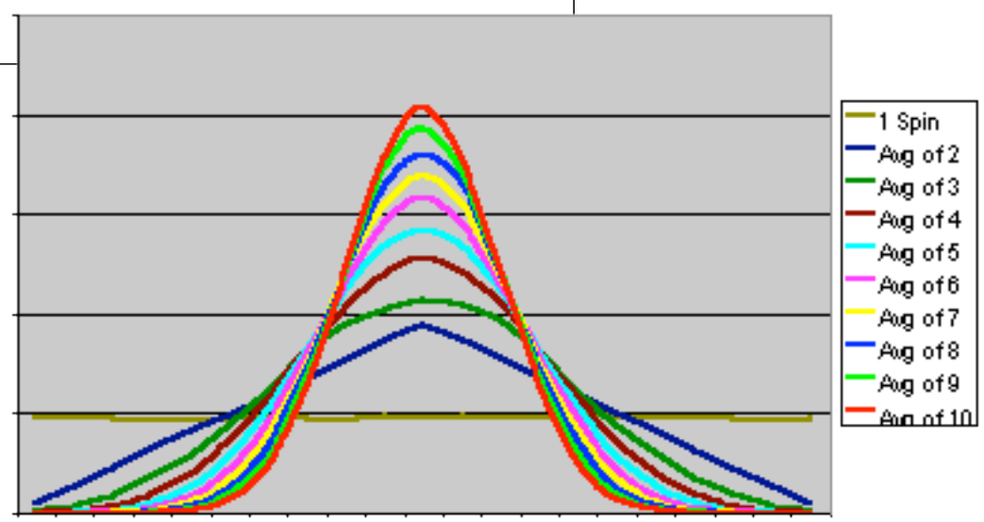
Brouwer fixed point theorem

Here is the formal statement of the Brouwer fixed point theorem, which was formalized in HOL Light by Harrison.

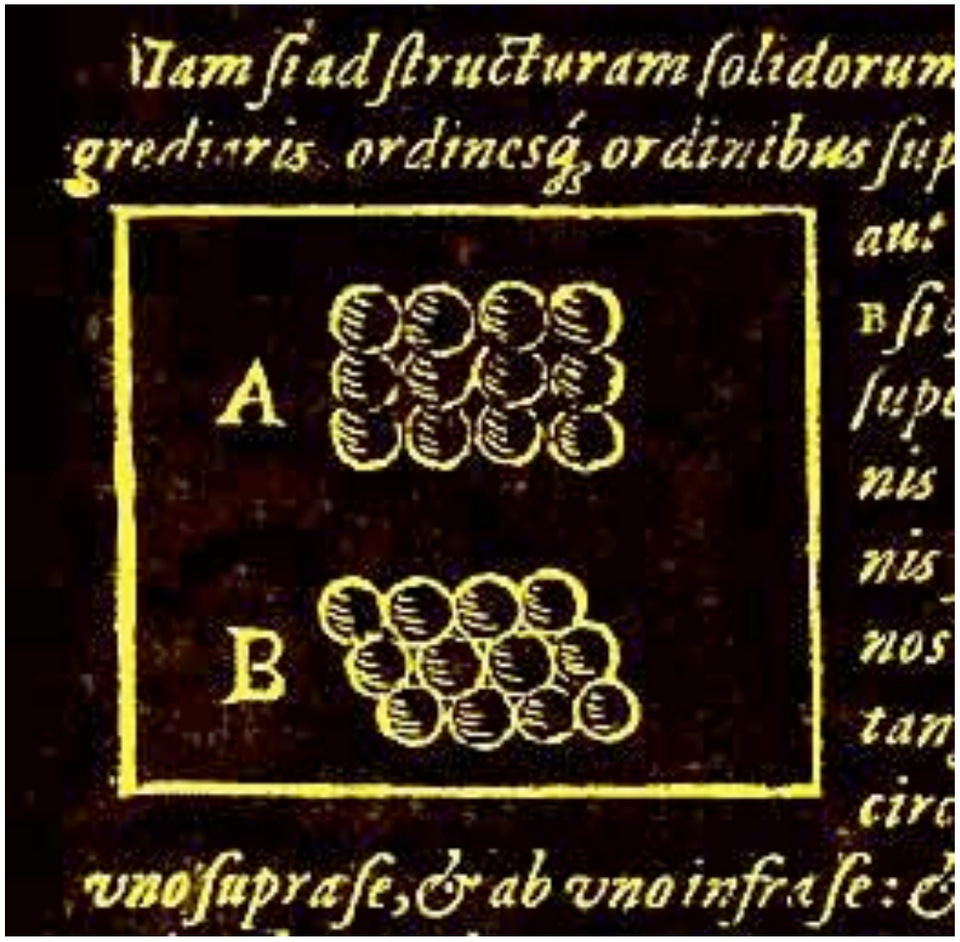
$\forall$ f:real^N $->$real^N s.
compact s $\wedge$ convex s $\wedge$ $\sim$(s = {}) $\wedge$ f continuous_on s $\wedge$ IMAGE f s SUBSET s
$\Longrightarrow$ $\exists$x. x IN s $\wedge$ f x = x

Central Limit Theorem

theorem (in prob_space) central_limit_theorem:
  fixes
    X :: "nat ⇒ 'a ⇒ real" and
    μ :: "real measure" and
    σ :: real and
    S :: "nat ⇒ 'a ⇒ real"
  assumes
    X_indep: "indep_vars (λi. borel) X UNIV" and
    X_integrable: "⋀n. integrable M (X n)" and
    X_mean_0: "⋀n. expectation (X n) = 0" and
    σ_pos: "$\sigma > 0$" and
    X_square_integrable: "⋀n. integrable M (λx. (X n x)^2)" and
    X_variance: "⋀n. variance (X n) = σ^2" and
    X_distrib: "⋀n. distr M borel (X n) = μ"
defines
  "S n ≡ λx. Σi<n. X i x"
shows
  "weak_conv_m (λn. distr M borel (λx. S n x / sqrt (n * σ^2)))
      (density lborel standard_normal_density)"



The face-centered cubic packing is “the tightest possible, so that in no other arrangement could more pellets be stuffed into the same container.” (Kepler, 1611)



The Kepler conjecture asserts that no packing of congruent balls in $\mathbb{R}^3$ can have density greater than the face-centered cubic packing. The Kepler conjecture is a theorem whose proof relies on many computer calculations [31]. The Kepler conjecture has been formalized [7] in a combination of the HOL Light and Isabelle proof assistants [32]. This formalization has been a large collaborative effort.

```
|– "g ∈ PlaneGraphs" and "tame g" shows "fgraph g ∈≃ Archive"
⊢ the_nonlinear_inequalities

⊢ import_tame_classification ∧
    the_nonlinear_inequalities ∧
    ⟹ the_kepler_conjecture

⊢ the_kepler_conjecture ⟺
     (∀V. packing V
          ⟹ (∃c. ∀r. &1 ≤ r
                ⟹ &(CARD(V ∩ ball(vec 0,r))) ≤
                     π * r^3 / sqrt(&18) + c * r^2))
```

A FORMAL PROOF OF THE KEPLER CONJECTURE

THOMAS HALES, MARK ADAMS, GERTRUD BAUER, DANG TAT DAT, JOHN HARRISON, HOANG LE TRUONG, CEZARY KALISZYK, VICTOR MAGRON, SEAN MCLAUGHLIN, NGUYEN TAT THANG, NGUYEN QUANG TRUONG, TOBIAS NIPKOW, STEVEN OBUA, JOSEPH PLESO, JASON RUTE, ALEXEY SOLOVYEV, TA THI HOAI AN, TRAN NAM TRUNG, TRIEU THI DIEP, JOSEF URBAN, VU KHAC KY, ROLAND ZUMKELLER,

Abstract. This article describes a formal proof of the Kepler conjecture on dense sphere packings in a combination of the HOL Light and Isabelle proof assistants. This paper constitutes the official published account of the now completed Flyspeck project.

We wish to acknowledge the help, support, influence, and various contributions of the following individuals: Nguyen Duc Thinh, Nguyen Duc Tam, Vu Quang Thanh, Vuong Anh Quyen, Catalin Anghel, Jeremy Avigad, Henk Barendregt, Herman Geuvers, Georges Gonthier, Daron Green, Mary Johnston, Christian Marchal, Laurel Martin, Robert Solovay, Erin Susick, Dan Synek, Nicholas Volker, Matthew Wampler-Doty, Benjamin Werner, Freek Wiedijk, Carl Witty, and Wenming Ye.

We wish to thank the following sources of institutional support: NSF grant 0503447 on the "Formal Foundations of Discrete Geometry" and NSF grant 0804189 on the "Formal Proof of the Kepler Conjecture," Microsoft Azure Research, William Benter Foundation, University of Pittsburgh, Radboud Research Facilities, Institute of Math (VAST), and VIASM.

flyspeck/counting_sphere...

GitHub, Inc. (US) https://github.com/flyspeck/flyspeck/blob/master/text_for

Search

7288 lines (7081 sloc) 267 KB

Raw Blame History

```
(* ========================================================================== *)
(* FLYSPECK - BOOK FORMALIZATION                                              *)
(* Section: Counting Spheres                                                  *)
(* Chapter: packing                                                           *)
(* Author: Thomas C. Hales                                                    *)
(* Date: 2011-06-18                                                           *)
(* ========================================================================== *)

module Counting_spheres = struct

  open Tactics_jordan;;
open Ysskqoy;;
open Hales_tactic;;

let CALC_ID_TAC = Calc_derivative.CALC_ID_TAC;;
let DIHV_SYM = Trigonometry2.DIHV_SYM;;

(* ---------------- *)


(* fat_lemma1 = Fatugpd.lemma1  *)
let fat_lemma1 = prove_by_refinement(
 `! (S:real^3->bool). packing S /\ S SUBSET ball_annulus ==> FINITE S`,
[
  (REPEAT STRIP_TAC);
  (SUBGOAL_THEN ` (S:real^3->bool) = S INTER ball (vec 0,&3 * h0)` ASSUME_TAC);
    (MATCH_MP_TAC (SET_RULE `! U V. U SUBSET ball_annulus /\ ball_annulus SUBSET V ==> (U = U INTER V)`));
    (ASM_SIMP_TAC[Pack_defs.ball_annulus;ball;cball;IN_DIFF;SUBSET;IN_ELIM_THM;Sphere.h0]);
    (REPEAT STRIP_TAC );
    (MATCH_MP_TAC (ARITH_RULE ` u <= &2 * #1.26 ==> u < &3 * #1.26`));
    BY((ASM_SIMP_TAC[]));
  (ONCE_ASM_REWRITE_TAC[]);
  BY((ASM_SIMP_TAC[Pack2.KIUMVTC]))
]);;

(* ckq_in_ball_annulus = Ckqowsa_3_points.in_ball_annulus *)
```

Aquamacs File Edit Options Tools Tuareg Fld Window Help — Mon 6:40 PM flyspeck

fusion.ml

scratch.hl | tame_defs2.hl

```
(* ========================================
(* Complete HOL kernel of types,
(*
(*       John Harrison, Universit
(*
(*           (c) Copyright, Univ
(*             (c) Copyright, Jc
(* ========================================

needs "lib.ml";;

module type Hol_kernel =
  sig
    type hol_type = private
      Tyvar of string
    | Tyapp of string *  hol_ty

    type term = private
      Var of string * hol_type
    | Const of string * hol_typ
    | Comb of term * term
    | Abs of term * term

    type thm

    val types: unit -> (string
    val get_type_arity : string
    val new_type : (string * ir
    val mk_type: (string * hol_
    val mk_vartype : string ->
    val dest_type : hol_type ->
    val dest_vartype : hol_type
    val is_type : hol_type -> b
    val is_vartype : hol_type -
    val tyvars : hol_type -> ho
    val type_subst : (hol_type
    val bool_ty : hol_type
    val aty : hol_type

    val constants : unit -> (st
    val get_const_type : string
    val new_constant : string *
    val type_of : term -> hol_t
    val alphaorder : term -> te
    val is_var : term -> bool
    val is_const : term -> bool
    val is_abs : term -> bool
    val is_comb : term -> bool
    val mk_var : string * hol_t
    val mk_const : string * (ho
    val mk_abs : term * term ->
    val mk_comb : term * term -
    val dest_var : term -> stri
    val dest_const : term -> st
    val dest_comb : term -> ter
    val dest_abs : term -> term
    val frees : term -> term li
    val freesl : term list -> t
    val freesin : term list -> term -> bool
    val vfree_in : term -> term -> bool
    val type_vars_in_term : term -> hol_type
    val variant : term list -> term -> term
    val vsubst : (term * term) list -> term
    val inst : (hol_type * hol_type) list ->
    val rand: term -> term
    val rator: term -> term
    val dest_eq: term -> term * term

    val dest_thm : thm -> term list * term
    val hyp : thm -> term list
    val concl : thm -> term
    val REFL : term -> thm
    val TRANS : thm -> thm -> thm
    val MK_COMB : thm * thm -> thm
    val ABS : term -> thm -> thm
    val BETA : term -> thm
    val ASSUME : term -> thm
    val EQ_MP : thm -> thm -> thm
    val DEDUCT_ANTISYM_RULE : thm -> thm -> t
    val INST_TYPE : (hol_type * hol_type) lis
    val INST : (term * term) list -> thm -> t
    val axioms : unit -> thm list
    val new_axiom : term -> thm
    val definitions : unit -> thm list
    val new_basic_definition : term -> thm
    val new_basic_type_definition :
          string -> string * string -> thm
end;;

(* ========================================
(* This is the implementation of those primitiv
(* ========================================

module Hol : Hol_kernel = struct

  type hol_type = Tyvar of string
                | Tyapp of string *  hol_type l

  type term = Var of string * hol_type
            | Const of string * hol_type
            | Comb of term * term
            | Abs of term * term

  type thm = Sequent of (term list * term)

(* ========================================
(* List of current type constants with their ar
(*
(* Initially we just have the boolean type and
(* constructor. Later on we add as primitive th
(* All other new types result from definitional
(* ========================================

  let the_type_constants = ref ["bool",0; "fun"
```

-:--- fusion.ml Top (1,0) SVN

shell | scratch.hl | fusion.ml

```
(* ========================================
(* Return all the defined types.
(* ========================================

  let types() = !the_type_constants

(* ========================================
(* Lookup function for type constants. Returns arit
(* ========================================

  let get_type_arity s = assoc s (!the_type_constan

(* ========================================
(* Declare a new type.
(* ========================================

  let new_type(name,arity) =
    if can get_type_arity name then
      failwith ("new_type: type "^name^" has alread
    else the_type_constants := (name,arity)::(!the_

(* ========================================
(* Basic type constructors.
(* ========================================

  let mk_type(tyop,args) =
    let arity = try get_type_arity tyop with Failur
      failwith ("mk_type: type "^tyop^" has not bee
    if arity = length args then
      Tyapp(tyop,args)
    else failwith ("mk_type: wrong number of argume

  let mk_vartype v = Tyvar(v)

(* ========================================
(* Basic type destructors.
(* ========================================

  let dest_type =
    function
        (Tyapp (s,ty)) -> s,ty
      | (Tyvar _) -> failwith "dest_type: type vari

  let dest_vartype =
    function
        (Tyapp(_,_)) -> failwith "dest_vartype: typ
      | (Tyvar s) -> s

(* ========================================
(* Basic type discriminators.
(* ========================================

  let is_type = can dest_type

  let is_vartype = can dest_vartype

(* ========================================
(* Return the type variables in a type and in a lis
```

-:--- fusion.ml 8% (88,0) SVN-83 (Tuareg F

april6 | fusion.ml

```
(* ========================================
(* Return the type variables in a type an
(* ========================================

  let rec tyvars =
      function
          (Tyapp(_,args)) -> itlist (unio
        | (Tyvar v as tv) -> [tv]

(* ========================================
(* Substitute types for type variables.
(*
(* NB: non-variables in subst list are ju
(* repeated many times), as are repetitio
(* ========================================

  let rec type_subst i ty =
    match ty with
      Tyapp(tycon,args) ->
          let args' = qmap (type_subst i)
          if args' == args then ty else T
      | _ -> rev_assocd ty i ty

  let bool_ty = Tyapp("bool",[])

  let aty = Tyvar "A"

(* ========================================
(* List of term constants and their types
(*
(* We begin with just equality (over all
(* operator is added. All other new const
(* ========================================

  let the_term_constants =
      ref ["=",Tyapp("fun",[aty;Tyapp("fun

(* ========================================
(* Return all the defined constants with
(* ========================================

  let constants() = !the_term_constants

(* ========================================
(* Gets type of constant if it succeeds.
(* ========================================

  let get_const_type s = assoc s (!the_te

(* ========================================
(* Declare a new constant.
(* ========================================

  let new_constant(name,ty) =
    if can get_const_type name then
      failwith ("new_constant: constant "
    else the_term_constants := (name,ty):
```

-:--- fusion.ml 17% (125,0) SVN-83 (Tuareg Fld A

fusion.ml

```
(* ========================================
(* Finds the type of a term (assumes it
(* ========================================

  let rec type_of tm =
    match tm with
      Var(_,ty) -> ty
    | Const(_,ty) -> ty
    | Comb(s,_) -> hd(tl(snd(dest_type(t
    | Abs(Var(_,ty),t) -> Tyapp("fun",[t

(* ========================================
(* Primitive discriminators.
(* ========================================

  let is_var = function (Var(_,_)) -> tr

  let is_const = function (Const(_,_)) -

  let is_abs = function (Abs(_,_)) -> tr

  let is_comb = function (Comb(_,_)) ->

(* ========================================
(* Primitive constructors.
(* ========================================

  let mk_var(v,ty) = Var(v,ty)

  let mk_const(name,theta) =
    let uty = try get_const_type name wi
      failwith "mk_const: not a constant
    Const(name,type_subst theta uty)

  let mk_abs(bvar,bod) =
    match bvar with
      Var(_,_) -> Abs(bvar,bod)
    | _ -> failwith "mk_abs: not a varia

  let mk_comb(f,a) =
    match type_of f with
      Tyapp("fun",[ty;_]) when Pervasive
        -> Comb(f,a)
    | _ -> failwith "mk_comb: types do r

(* ========================================
(* Primitive destructors.
(* ========================================

  let dest_var =
    function (Var(s,ty)) -> s,ty | _ ->

  let dest_const =
    function (Const(s,ty)) -> s,ty | _ -

  let dest_comb =
    function (Comb(f,x)) -> f,x | _ -> f

  let dest_abs =
```

-:--- fusion.ml 25% (199,0) SVN-83 (T

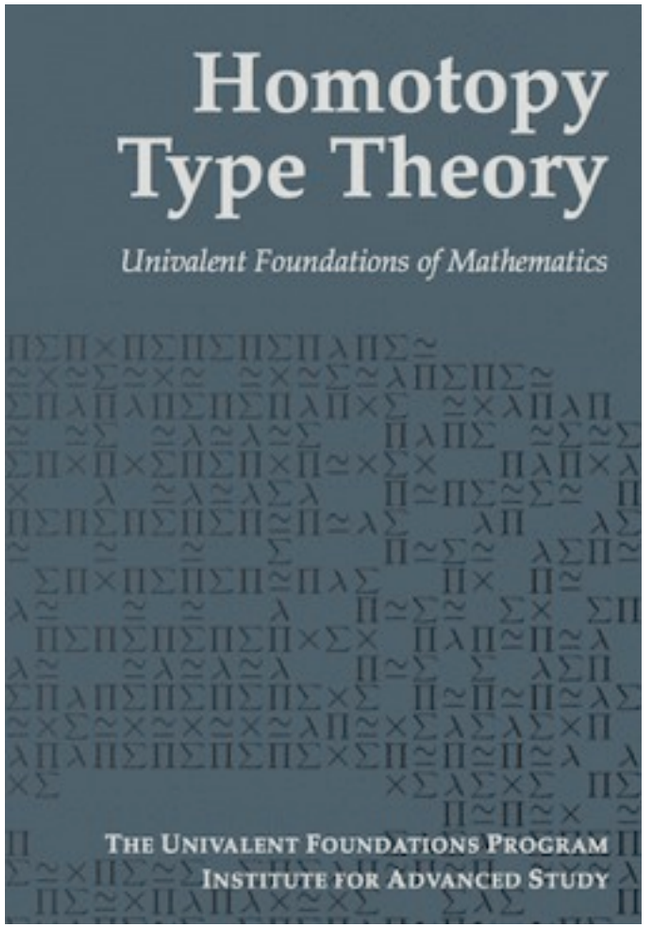
-:--- fusion.ml 35% (231,0) SVN-83

Wednesday, January 6, 2016

Grayson's code in HoTT

```
Definition ClassifyingSpace G := pointedType (Torsor G) (trivialTorsor G).
Definition E := PointedTorsor.
Definition B := ClassifyingSpace.
Definition π {G:gr} := underlyingTorsor : E G -> B G.

Lemma isconnBG (G:gr) : isconnected (B G).
Proof. intros. apply (base_connected (trivialTorsor _)).
  intros X. apply (squash_to_prop (torsor_nonempty X)). { apply propproperty. }
  intros x. apply hinhpr. exact (torsor_eqweq_to_path (triviality_isomorphism X x)).
Defined.
```



Bellare-Rogaway game style proofs in cryptography

Initialize Game C_0

100 $T[\varepsilon] \leftarrow 0^n$ Game C_1

procedure $F(M)$

110 $s \leftarrow s+1,\ \ M^s \leftarrow M$

111 $P \leftarrow \mathsf{Prefix}(M^1,\ldots,M^s),\ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

112 for $j \leftarrow p+1$ to m do

113 $\quad X \leftarrow C \oplus M_j$

114 $\quad C \xleftarrow{\$} \{0,1\}^n$

115 $\quad$ if $C \in \mathrm{image}(\pi)$ then $bad \leftarrow \mathsf{true}$, $C \leftarrow \overline{\mathrm{image}}(\pi)$

116 $\quad$ if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$, $C \leftarrow \pi[X]$

117 $\quad \pi[X] \leftarrow C$

118 $\quad T[M_{1..j}] \leftarrow C$

119 return C

Initialize Game C_2

200 $T[\varepsilon] \leftarrow 0^n$

procedure $F(M)$

210 $s \leftarrow s+1,\ \ M^s \leftarrow M$

211 $P \leftarrow \mathsf{Prefix}(M^1,\ldots,M^s),\ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

212 for $j \leftarrow p+1$ to m do

213 $\quad X \leftarrow C \oplus M_j$

214 $\quad C \xleftarrow{\$} \{0,1\}^n$

215 $\quad$ if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

216 $\quad \pi[X] \leftarrow C$

217 $\quad T[M_{1..j}] \leftarrow C$

218 return C

Initialize Game C_3

300 $T[\varepsilon] \leftarrow 0^n,\ \ \mathsf{defined} \leftarrow 0^n$

procedure $F(M)$

310 $s \leftarrow s+1,\ \ M^s \leftarrow M$

311 $P \leftarrow \mathsf{Prefix}(M^1,\ldots,M^s),\ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

312 for $j \leftarrow p+1$ to m do

313 $\quad X \leftarrow C \oplus M_j$

314 $\quad C \xleftarrow{\$} \{0,1\}^n$

315 $\quad$ if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

316 $\quad \pi[X] \leftarrow \mathsf{defined}$

317 $\quad T[M_{1..j}] \leftarrow C$

318 return C

Initialize Game C_4

400 $T[\varepsilon] \leftarrow 0^n,\ \ \mathsf{defined} \leftarrow 0^n$

procedure $F(M)$

410 $s \leftarrow s+1,\ \ M^s \leftarrow M$

411 $P \leftarrow \mathsf{Prefix}(M^1,\ldots,M^s),\ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

412 for $j \leftarrow p+1$ to m do

413 $\quad X \leftarrow C \oplus M_j$

414 $\quad$ if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

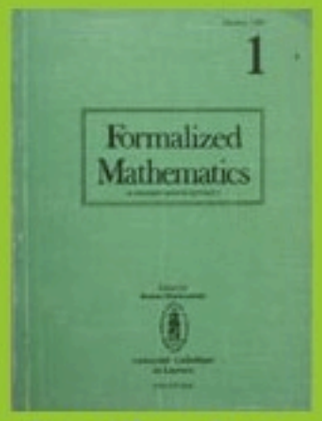
415 $\quad \pi[X] \leftarrow \mathsf{defined}$

416 $\quad C \leftarrow T[M_{1..j}] \xleftarrow{\$} \{0,1\}^n$

417 $Z^s \xleftarrow{\$} \{0,1\}^n$

418 return Z^s

game used in the analysis of the CBC MAC.



http://fm.mizar.org/

Formalized Mathematics

(a computer assisted approach)

ISSN 1426-2630 (Print)
eISSN 1898-9934 (Online)

- About this journal
- For authors
- Subscription

Since 2006, papers have been published by the de Gruyter Open

Established in 1990. In the years 1990 - 1993 published by the Université Catholique de Louvain.

FM Bibliography file
External Bibliography file

Content by Articles

Volume 23 (2015)
Articles in press: Nr.4
Number 3
Number 2
Number 1

Volume 22 (2014)
Number 4
Number 3
Special Issue-Number 2
Number 1

Volume 21 (2013)
Number 4
Number 3
Number 2
Number 1

Volume 20 (2012)
Number 4
Number 3
Number 2
Number 1

Volume 19 (2011)
Number 4
Number 3
Number 2
Number 1

FORMALIZED MATHEMATICS
Vol. **23**, No. **4**, Pages 331–347, **2015**
DOI: 10.1515/forma-2015-0027

DE GRUYTER OPEN
degruyter.com/view/j/forma

Characteristic of Rings. Prime Fields

Christoph Schwarzweller
Institute of Computer Science
University of Gdańsk
Poland

Artur Korniłowicz
Institute of Informatics
University of Białystok
Poland

Summary. The notion of the characteristic of rings and its basic properties are formalized [14], [39], [20]. Classification of prime fields in terms of isomorphisms with appropriate fields ($\mathbb{Q}$ or $\mathbb{Z}/p$) are presented. To facilitate reasonings within the field of rational numbers, values of numerators and denominators of basic operations over rationals are computed.

MSC: 03B35

Keywords: commutative algebra; characteristic of rings; prime field

MML identifier: `RING_3`, version: `8.1.04 5.34.1256`

Now we state the propositions:

(108) Let us consider a prime number p, and a field F with characteristic p. Then $\mathbb{Z}/p$ and PrimeField F are isomorphic.

(109) Let us consider a prime number p, and a strict subfield F of $\mathbb{Z}/p$. Then $F = \mathbb{Z}/p$.

(110) Let us consider a prime number p. Then PrimeField $\mathbb{Z}/p = \mathbb{Z}/p$.

(111) Let us consider a prime number p, and a field F with characteristic p. Then F includes $\mathbb{Z}/p$.

We are still far from having an automated mathematical journal referee system, but close enough to propose this as a realistic research program. Already some 10% of all papers of the Principles of Programming Languages (POPL) symposium in computer science are completely formalized [61]. Other recent research automates the translation of mathematical prose from English into a computer-parsable form with semantic content [21]. As these technologies develop, we may anticipate the day when the precise formal statements of mathematical theorems may be extracted from the prose. Once sufficiently many statements from the natural language proof can similarly be extracted, proof automation will take over, filling in the remaining details, to produce a formal proof from the natural language text.

List of long proofs

From Wikipedia, the free encyclopedia

This is a list of unusually long mathematical proofs.

As of 2011, the longest mathematical proof, measured by number of published journal pages, is the classification of finite simple groups with well over 10000 pages. There are several proofs that would be far longer than this if the details of the computer calculations they depend on were published in full.

Contents [hide]

1 Long proofs
2 Long computer calculations
3 Long proofs in mathematical logic
4 See also
5 References

Long proofs [edit]

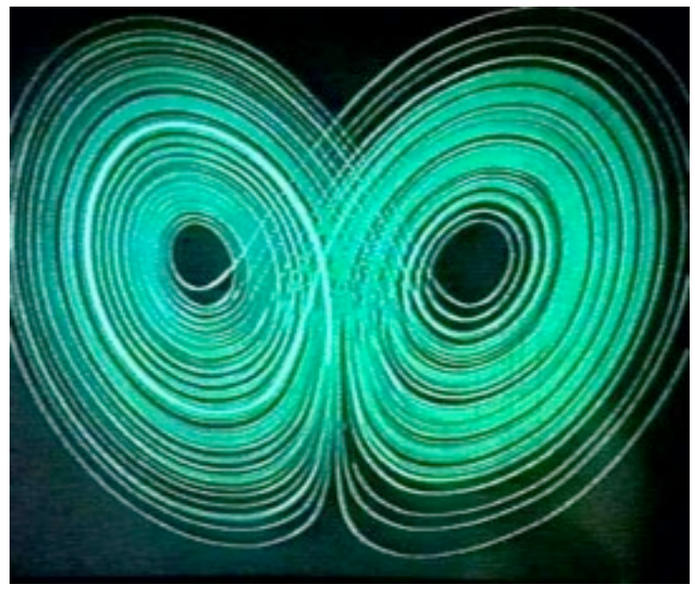
The length of unusually long proofs has increased with time. As a rough rule of thumb, 100 pages in 1900, or 200 pages in 1950, or 500 pages in 2000 is unusually long for a proof.

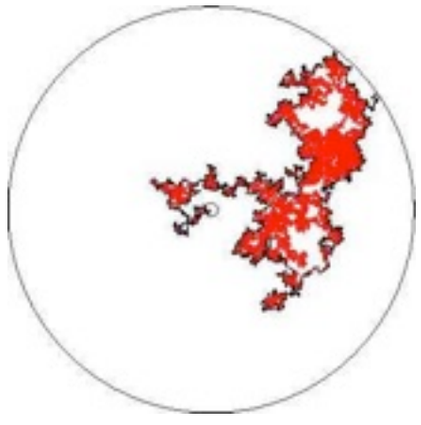
- 1799 The Abel–Ruffini theorem was nearly proved by Paolo Ruffini, but his proof, spanning 500 pages, was mostly ignored and later, in 1824, Niels Henrik Abel published a proof that required just six pages
- 1890 Killing's classification of simple complex Lie algebras, including his discovery of the exceptional Lie algebras, took 180 pages in 4 papers.
- 1894 The ruler-and-compass construction of a polygon of 65537 sides by Johann Gustav Hermes took over 200 pages.
- 1905 Lasker–Noether theorem Emanuel Lasker's original proof took 98 pages, but has since been simplified: modern proofs are less than a page long.
- 1963 Odd order theorem This was 255 pages long, which at the time was over 10 times as long as what had previously been considered a long paper in group theory.
- 1964 Resolution of singularities Hironaka's original proof was 216 pages long; it has since been simplified considerably down to about 10 or 20 pages.

+ Four-color theorem, Grothendieck, Weil conjectures, Harish-Chandra, Lafforgue, Langlands, Arthur, Almgren, Kepler conjecture, geometrization theorem, etc.

Mathematics is making the transition to computer based systems.

Birch-Swinnerton Dyer conjecture, Sato-Tate Conjecture, Lyons-Sims group of order $2^8 3^7 5^6 7^1 11^1 31^1 37^1 67^1$, original proof of the Calalan conjecture $x^m - x^n = 1$, `qWZ` proof of the Rogers-Ramunujan identities, conjectural optimal packings of tetrahedra (Chen-Engel-Glotzer), 4-color theorem, finite projective plane of order 10, Smale's 14th problem on strange attractors in the Lorenz oscillator, Mandelbrot's conjectures in fractal geometry, visualization of sphere eversions and Costa surface embeddings, the double bubble conjecture, construction of counterexamples to the Kelvin conjecture, calculation of kissing numbers (Sloane-Odlyzko), the character table for E^8 (Atlas project), Cohn-Kumar proof of the packing optimality of the Leech and E^8 packings among lattices, classification of fake projective planes, weak Goldbach, twin prime problem (2013).





Mathematics and Computer Science

Historically, ZFC set theory and type theory gave two different resolutions of Russell's paradox. These two approaches to foundations continue to this day.

mathematicians	—	computer scientists
ZFC set theory	—	type theory
classical logic	—	constructive logic

As more mathematics moves to computer platforms, these lines are becoming blurred.

- Isabelle/HOL
  - prime number theorem
  - Sel4 operating system microkernel verification
- Coq
  - Odd Order theorem
  - CompCert C compiler verification
- HOL Light
  - Kepler conjecture
  - Intel microcode verification

Trust

The formal verification of HOL in HOL does not settle the issue of trust once and for all. The reliability of the proof assistant ultimately rests on the entire computer environment in which the software operates, including the semantics of programming languages, compilers, operating systems, and hardware. These issues should be a concern of every mathematician who cares about the foundations of mathematics at a time when the practice of mathematics is gradually migrating to computers.

Compcert

Compcert is a formally verfied C compiler that was developed at INRIA According to the project page,

> Its intended use is the compilation of life-critical and mission-critical software written in C and meeting high levels of assurance.. . . What sets CompCert C apart from any other production compiler, is that it is formally verified, using machine-assisted mathematical proofs, to be exempt from miscompilation issues. In other words, the executable code it produces is proved to behave exactly as specified by the semantics of the source C program. This level of confidence in the correctness of the compilation process is unprecedented and contributes to meeting the highest levels of software assurance.

Computer scientists at the University of Utah developed software (Csmith) that automatically generates test-cases to find compiler bugs. The researchers report that their software found over 325 previously unknown software bugs in compilers. Here is their conclusion about the Compcert compiler:

> The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task. The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users.

The current working goal of researchers is to create an unbroken formally-verified chain extended from the HOL Light kernel all the way down to machine code. Most of the links in the chain have been forged. (5)

CakeML is a dialect of ML with mathematically rigorous operational semantics. According to its designers, "Our overall goal for CakeML is to provide the most secure system possible for running verified software and other programs that require a high-assurance platform" [45]. The CakeML team has improved the HOL in HOL verification [52], [51], [44]. This work closes various gaps in the verification of the OCaml implementation of HOL, such as *object magic* (a mechanism that defeats the OCaml type system) and *mutable strings* (which allow a theorem to be edited to state something different from what was proved). The HOL system can be extended by adding new definitions and new types. The formal verification now covers these extensions. It verifies the soundness of an implementation of the HOL kernel in CakeML, according to the formally specified operational semantics of CakeML.

In related work, a verified compiler has been constructed for the CakeML language [45], [58]. This brings us close to end-to-end verification of HOL Light, from its high-level logical description down to execution in machine code.

On Digital Math Libraries

We should not compromise rigorous mathematical standards as we move from paper to computer. In fact, this is an opportunity to drastically improve standards. Many computer bugs are simply slips in logical and mathematical reasoning made by programmers and software designers.

- Mathematics influences the standards of scientific discourse, in the statistical sciences, in computer science, and throughout the sciences. If we promote sloppy platforms, the entire world will be worse off.
- Bugs in computer systems can lead to disaster: Intel Pentium FDIV bug, Ariane V explosion, . . .
- Bugs and design weaknesses in cryptographic software can be exploited by adversaries: Heartbleed, Logjam, Freak bug, . . .

QED Manifesto

QED Manifesto "QED is the very tentative title of a project to build a computer system that effectively represents all important mathematical knowledge and techniques. The QED system will conform to the highest standards of mathematical rigor, including the use of strict formality in the internal representation of knowledge and the use of mechanical methods to check proofs of the correctness of all entries in the system."

"**QED Project Summary:** The aim of the QED project is to build a single, distributed, computerized repository that rigorously represents all important, established mathematical knowledge . . . This system will have benefits for mathematics, science, technology, and education."

http://mizar.org/qed/workshop94/qed-proposal.html

QED+20

In 2014, I participating in the QED+20 conference, looking back at the QED project after 20 years.

At QED+20, John Harrison made a critical historical analysis of the QED project. See https://www.youtube.com/watch?v=yDN2a6xFUtc (slides are available). Harrison criticizes the way the project got lost in balkinization and meta-concerns.

The QED manifesto has had great influence, but the dream was not realized.

A concrete proposal: mathematical FABSTRACTS (formal abstracts)

Given today's technology, it is not reasonable to ask for all proofs to be formalized. But with today's technology, it seems that it should be possible to create a formal abstract service that

- Gives a statement of the main theorem(s) of each published mathematical paper in a language that is both human and machine readable,
- Links each term in theorem statements to a precise definition of that term (again in human/machine readable form), and
- Grounds every statement and definition is the system in some foundational system for doing mathematics.

Proof Automation is the key

One way to make proof assistants more usable is to increase the amount of automation. Ideally, the human should provide the high level structure of a proof as it is done in a traditionally published paper, and the computer should use search algorithms to insert the low-level reasoning. As technology has developed, computers have become capable of providing more and more of the low-level reasoning.

There is no general decision procedure. Some decision procedures such as the Presburger algorithm for the additive first-order theory of natural numbers or quantifier elimination for real closed fields only have a limited practical value for formal proofs because they are so slow.

Buchberger's algorithm

$$
\begin{aligned}
\vdash \quad & a' = \&2 * a \wedge \ b' = a * a - \&4 * b \wedge \\
& x_2 * y_1 = x_1 \wedge \ y_2 * y_1^2 = \&1 - b * x_1^4 \wedge \\
& y_1^2 = \&1 + a * x_1^2 + b * x_1^4 \\
& \Longrightarrow y_2^2 = \&1 - a' * x_2^2 + b' * x_2^4
\end{aligned}
$$

SAT/SMT reasoning

For example, the following calculation comes up in the proof of the Odd Order theorem. Let G be a finite group, and let (β_{ij}) be a matrix of virtual characters, with each entry a linear combination $\pm\chi \pm \chi' \pm \chi''$ of three distinct irreducible characters. Assume the matrix has at least four rows and two columns. Assume the inner product relations

$$\langle \beta_{ij}, \beta_{i'j'} \rangle = \delta_{ii'} + \delta_{jj'}, \quad \text{for } (i,j) \neq (i',j'),$$

with respect to the usual inner product on class functions with an orthonormal basis consisting of irreducible characters. The conclusion is that the virtual characters in each column of the matrix have a common irreducible character as constituent with a common sign.

th3_5.smt.txt

New Open Recent Save Print Undo Redo Cut Copy Paste Search Preferences Help

scratch 1 | *shell* 2 | bourbaki_developments.tex 3 | th3_5.smt.txt 4

```
                    (and (= B112 B221) (not (= S112 S221))))
                 (or (and (= B111 B223) (not (= S111 S223)))
                    (and (= B111 B222) (not (= S111 S222)))
                    (and (= B111 B221) (not (= S111 S221)))))
              (or
                 (and (not (= B113 B223)) (not (= B113 B222))
                    (not (= B113 B221)))
                 (and (not (= B112 B223)) (not (= B112 B222))
                    (not (= B112 B221)))
                 (and (not (= B111 B223)) (not (= B111 B222))
                    (not (= B111 B221))))))))))
(not
 (and
    (or
       (and
          (or (and (= B423 B123) (= S423 S123))
             (and (= B422 B123) (= S422 S123))
             (and (= B421 B123) (= S421 S123)))
          (or (and (= B323 B123) (= S323 S123))
             (and (= B322 B123) (= S322 S123))
             (and (= B321 B123) (= S321 S123)))
          (or (and (= B223 B123) (= S223 S123))
             (and (= B222 B123) (= S222 S123))
             (and (= B221 B123) (= S221 S123)))
          (or (and (= B123 B123) (= S123 S123))
             (and (= B122 B123) (= S122 S123))
             (and (= B121 B123) (= S121 S123))))
       (and
          (or (and (= B423 B122) (= S423 S122))
             (and (= B422 B122) (= S422 S122))
             (and (= B421 B122) (= S421 S122)))
          (or (and (= B323 B122) (= S323 S122))
             (and (= B322 B122) (= S322 S122))
             (and (= B321 B122) (= S321 S122)))
          (or (and (= B223 B122) (= S223 S122))
             (and (= B222 B122) (= S222 S122))
             (and (= B221 B122) (= S221 S122)))
          (or (and (= B123 B122) (= S123 S122))
             (and (= B122 B122) (= S122 S122))
             (and (= B121 B122) (= S121 S122))))
       (and
          (or (and (= B423 B121) (= S423 S121))
             (and (= B422 B121) (= S422 S121))
             (and (= B421 B121) (= S421 S121)))
          (or (and (= B323 B121) (= S323 S121))
             (and (= B322 B121) (= S322 S121))
             (and (= B321 B121) (= S321 S121)))
          (or (and (= B223 B121) (= S223 S121))
             (and (= B222 B121) (= S222 S121))
             (and (= B221 B121) (= S221 S121)))
          (or (and (= B123 B121) (= S123 S121))
             (and (= B122 B121) (= S122 S121))
             (and (= B121 B121) (= S121 S121)))))))))
(check-sat)
(exit)
```

-:--- th3_5.smt.txt Bot (2330,0) (Text Spc Fill)

Mark set

$$
\begin{aligned}
\Delta(x_1,\ldots,x_6) &= x_1x_4(-x_1+x_2+x_3-x_4+x_5+x_6)\\
&\quad +x_2x_5(x_1-x_2+x_3+x_4-x_5+x_6)\\
&\quad +x_3x_6(x_1+x_2-x_3+x_4+x_5-x_6)\\
&\quad -x_2x_3x_4-x_1x_3x_5-x_1x_2x_6-x_4x_5x_6,\\
\Delta_4 &= \frac{\partial\Delta}{\partial x_4},\\
\mathrm{dih}_x(x_1,\ldots,x_6) &= \frac{\pi}{2}-\arctan\left(\frac{-\Delta_4(x_1,\ldots,x_6)}{\sqrt{4x_1\Delta(x_1,\ldots,x_6)}}\right),\\
\mathrm{dih}_y(y_1,\ldots,y_6) &= \mathrm{dih}_x(y_1^2,\ldots,y_6^2).
\end{aligned}
$$

If $4 \le x_i \le 6.3504$, for $i = 1, 2, 3$ and if $x_4 = 4$, and $3.01^2 \le x_i \le 3.24^2$ for $i = 5, 6$, then

$$\mathrm{dih}_x(x_1,\ldots,x_6) - \pi/2 + 0.46 < 0.$$

(431 seconds)

A *first-order formula with equality in the predicate calculus* is a formula built from variables $x, y, \ldots$, logical operations $\neg, \wedge, \Rightarrow, \ldots$, n-ary function symbols $f, g \ldots$, n-ary predicate symbols $P, Q, \ldots$ and quantified variables $\forall x, \exists y, \ldots$, according to syntactic rules that we will not spell out here. For example,

$$((\exists x.\ P(x)) \Rightarrow (\forall y.\ Q(y))) \Leftrightarrow (\forall x \forall y.\ P(x) \Rightarrow Q(y))$$

is a first-order formula. In contrast to higher-order logics discussed earlier, in a first-order formula, quantifiers over function symbols and predicate symbols are not allowed.

We are interested in algorithms that are *refutation complete*; that is, given the input of an unsatisfiable first-order formula, the algorithm outputs a proof of its unsatisfiability. A refutation complete algorithm can also produce proofs of logically valid formulas, by refuting the negation of the formula.

There are some automated theorem provers based on higher-order logic but the most prevalent practice is to translate problems from higher-order logic into first-order logic, solve them there, then translate the answers back into higher-order logic, with an automated reconstruction of a formal proof inside the proof assistant. (6)

An early example of translating proofs in this way is Harrison's MESON procedure (based on model elimination), implemented in HOL Light. In this approach, the human supplies all the relevant theorems, and the automated procedure generates the logical glue that combines the given theorems into a proof.

Paulson advocates shipping the goal together with large libraries of theorems to a first-order theorem prover, and letting it figure out which of many theorems to use in the proof [53]. The human is no longer forced to search through the libraries to find the relevant theorems. The method is called a *sledgehammer* for its ability to deliver a powerful blow and its complete lack of finesse.

In a refinement of this approach, an automated heuristic procedure selects a few hundred theorems deemed to be the most relevant and ships those to the theorem prover. For example, to prove a new trig identity, we might select other trig identities as likely to be relevant. There is an art to selecting relevant premises wisely, and machine learning algorithms can be trained to do this effectively [66], [40], [39], [2].

Lemma 1. *The convex hull of any three points in* $\mathbb{R}^3$ *is a set of measure zero.*

Proof. An automated procedure locates the following facts in the relevant libraries and combines them with the necessary logic to give a formal proof.

1. the convex hull is a subset of the affine hull;
2. the affine hull of three points in $\mathbb{R}^3$ is a set of measure zero;
3. a subset of a set of measure zero has measure zero.

Sledgehammers and machine learning algorithms have led to visible success. Fully automated procedures can prove 40% of the theorems in the Mizar math library, 47% of the HOL Light/Flyspeck libraries, with comparable rates in Isabelle [38], [41], [15]. These automation rates represent an enormous savings in human labor.

1st Conference on Artificial Intelligence and Theorem Proving

AITP 2016

April 3–6, 2016, Obergurgl, Austria

Background

Large-scale semantic processing and strong computer assistance of mathematics and science is our inevitable future. New combinations of AI and reasoning methods and tools deployed over large mathematical and scientific corpora will be instrumental to this task. The AITP conference is the forum for discussing how to get there as soon as possible, and the force driving the progress towards that.

Topics

- AI and big-data methods in theorem proving and mathematics
- Collaboration between automated and interactive theorem proving
- Common-sense reasoning and reasoning in science
- Alignment and joint processing of formal, semi-formal, and informal libraries
- Methods for large-scale computer understanding of mathematics and science
- Combinations of linguistic/learning-based and semantic/reasoning methods