Proofs with Computers: Introduction

Nima Rasekh

Universität Greifswald

11.04.2025

Overview

- **Goal:** To learn how to formulate and program mathematical and logical proofs with computers.
- **Plan:**
  - First half: Basics of Lean.
  - Second half: We will decide later!
- **Today:**
  - Overview and rules for the lecture
  - Projects
  - History
  - Background
  - Logic
- **Starting next week:** Formalization in Lean

Logistik

1. Everyone needs access to a laptop!
2. First steps: Installation of VS Code, Lean and Git and signing up in GitHub.
3. If there are any problems: **Ask!**

GitHub Page

[Demonstration]

Projects

- The grade is determined by a project.
- A project is a formalization and a presentation.
- Everyone who does a project passes.[1]
- Exercises are optional, but **strongly** recommended.
- Everyone who needs a grade: Notify me by **02.05.2025**! Otherwise, you cannot pass.
- Not sure? Still notify me!

[1] Terms and Conditions apply.

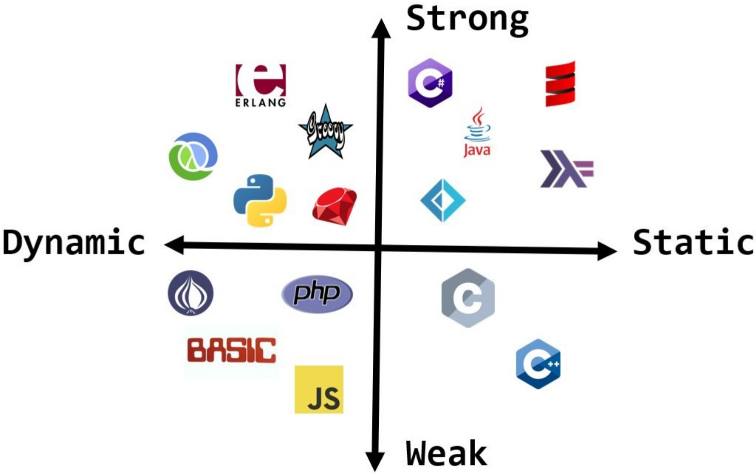
What is the Benefit of Computers?

- Computers check inputs
- Computers automate processes
- Computers manage data

Application of computers in proving programs and mathematics is called:

Formalization!

Programming Languages



Dynamic vs. Static Programming Languages

Demonstration

Where does Formalization come from?

1. First steps in the 60s and 70s with **Automath** and **Logic for Computable Functions**
2. In the 80s and 90s, we have the first languages that are still relevant: **Coq**, **Isabelle**, **Agda**, **PVS**
3. Since the 2000s:
   - New languages: **Metamath**, **Lean**, ...
   - Formalization of interesting problems: **Four color theorem** (2005), **Kepler conjecture** (2014), ...
4. **Why now?** User friendlier, AI, ...

Formalization in Programming

```
From iris.algebra Require Export view.
From iris.algebra Require Import proofmode_classes big_op.
From iris.prelude Require Import options.

(** The authoritative camera with fractional authoritative elements *)
(** The authoritative camera has 2 types of elements: the authoritative
element [●{q} a] and the fragment [◯ b] (of which there can be several). To
enable sharing of the authoritative element [●{q} a], it is equiped with a
fraction [q]. Updates are only possible with the full authoritative element
[● a] (syntax for [●{1} a]]), while fractional authoritative elements have
agreement, i.e., [✓ (●{p1} a1 · ●{p2} a2) → a1 ≡ a2]. *)

(** * Definition of the view relation *)
(** The authoritative camera is obtained by instantiating the view camera. *)
Definition auth_view_rel_raw {A : ucmra} (n : nat) (a b : A) : Prop :=
  b ≼{n} a ∧ ✓{n} a.
Lemma auth_view_rel_raw_mono (A : ucmra) n1 n2 (a1 a2 b1 b2 : A) :
  auth_view_rel_raw n1 a1 b1 →
  a1 ≡{n2}≡ a2 →
  b2 ≼{n2} b1 →
  n2 ≤ n1 →
```

(a) **Coq** Iris

```
%-------------------------------------------------------------------------------
% criteria_3D.pvs
% ACCoRD v.1.0
%-------------------------------------------------------------------------------

criteria_3D[D,H:posreal] : THEORY
BEGIN

  IMPORTING vertical_criterion[D,H],
            horizontal_criterion_line[D]%,

  sp        : VAR Sp_vect3 % 3-D separation
  s,v,nv,
  nvo,nvi   : VAR Vect3
  vo,vi     : VAR Nzv2_vect3
  epsh,epsv : VAR Sign

  criterion_3D?(sp,v,epsh,epsv)(nv) : bool =
    (horizontal_sep?(sp) AND horizontal_criterion?(sp,epsh)(nv)) OR
    (vertical_criterion?(epsv)(sp,v)(nv) AND
     (horizontal_los?(sp) OR
```

(b) **PVS** in NASA

```
{-# OPTIONS --without-K #-}
-- https://upload.wikimedia.org/wikipedia/commons/e/e2/SHA-1.svg
-- http://www.faqs.org/rfcs/rfc3174.html
open import Data.Nat.Base using (ℕ; zero; suc; _+_; _∸_)
import Data.Vec as V
open V using (Vec; []; _∷_)
open import Function using (_∘_)
open import FunUniverse.Core hiding (_,_)
open import Data.Fin using (Fin; zero; suc; #_; inject+; raise) renaming (toℕ to Fin▹ℕ)

open import Solver.Linear

module sha1 where

Endo : Set → Set
Endo A = A → A

module FunSHA1
  {t}
  {T : Set t}
  {funU : FunUniverse T}
```

(c) **Agda** in Cryptography & Blockchain

```
inductive Tp where
| u (size : Nat)
| i (size : Nat)
| bi -- BigInt
| bool
| unit
| str (size: U 32)
| fmtStr (size : U 32) (argTps : List Tp)
| field
| slice (element : Tp)
| array (element: Tp) (size: U 32)
| tuple (name : Option String) (fields : List Tp)
| ref (tp : Tp)
| fn (argTps : List Tp) (outTp : Tp)

mutual

def tpsDecEq (a b : List Tp) : Decidable (a = b) := match a, b with
| [], [] => isTrue rfl
| [], _ :: _ => isFalse (by simp)
| _ :: _, [] => isFalse (by simp)
```

(d) **Lean** in Cryptography

Formalization in Mathematics

```
Lemma eqm_genus : genus G = genus G'.
Proof.
rewrite {2}/genus /Euler_lhs -eqm_gcomp; have [e n f Eenf Ee En Ef] := eqGG'.
by rewrite /Euler_rhs /= -(eq_fcard Ee) -(eq_fcard En) -(eq_fcard Ef).
Qed.

Lemma eqm_planar : planar G = planar G'.
Proof. by rewrite {2}/planar -eqm_genus. Qed.

End EqualHypermap.

Notation "x '=m' y" := (eqm x y) (at level 70, no associativity).

Lemma eqm_sym G G': G =m G' -> G' =m G.
Proof. by case: G => d e n f EG [] *; apply: EqHypermap => x; auto. Qed.

Lemma dual_inv G : dual (dual G) =m G.
Proof.
case: G (@edgeI G) (@nodeI G) (@faceI G) => d e n f /= eK eI nI fI.
by apply: EqHypermap; apply: finv_inv.
Qed.
```

(a) Four color theorem in **Coq**

```
let float_pow_pos_lo pp x_tm n =
  let rec pow n =
    match n with
      | 0 -> INST[x_tm, x_var_real] float_pow0_lo
      | 1 -> INST[x_tm, x_var_real] float_pow1_lo
      | 2 ->
          let mul_lo = float_mul_lo pp x_tm x_tm in
          let lo_tm = lhand (concl mul_lo) in
          let th0 = INST[x_tm, x_var_real; lo_tm, lo_var_real] float_pow2_lo in
            MY_PROVE_HYP mul_lo th0
     | _ ->
          let _ = assert (n > 2) in
          if (n land 1) = 0 then
            (* even *)
            let t_th = pow (n lsr 1) in
              float_pow_pos_double_lo pp x_tm t_th
          else
            (* odd *)
            let t_th = pow (n - 1) in
              float_pow_pos_suc_lo pp x_tm t_th
  in
```

(b) Kepler conjecture in **Isabelle**

```
π₄S³≃ℤ/2ℤ-direct : GroupEquiv (π 4 S³) (ℤGroup/ 2)
π₄S³≃ℤ/2ℤ-direct = DirectProof.BrunerieGroupEquiv


-- This direct proof allows us to define a much simplified version of
-- the Brunerie number:
β' : ℤ
β' = fst DirectProof.computer η₃'

-- This number computes definitionally to -2 in a few seconds!
β'≡-2 : β' ≡ -2
β'≡-2 = refl

-- As a sanity check we have proved (commented as typechecking is quite slow):
-- β'Spec : GroupEquiv (π 4 S³) (ℤGroup/ abs β')
-- β'Spec = DirectProof.BrunerieGroupEquiv'

-- Combining all of this gives us the desired equivalence of groups by
-- computation as conjectured in Brunerie's thesis:
π₄S³≃ℤ/2ℤ-computation : GroupEquiv (π 4 S³) (ℤGroup/ 2)
π₄S³≃ℤ/2ℤ-computation = DirectProof.BrunerieGroupEquiv''
```

(c) Homotopy groups in **Cubical Agda**

```

/-- Theorem 9.4 in [Analytic] for weak bounded exactness -/
theorem thm94_weak' :
  ∀ m : ℕ, ∃ (k K : ℝ≥0) (hk : fact (1 ≤ k)) (c₀ : ℝ≥0),
  ∀ (S : Profinite) (V : SemiNormedGroup.{u}) [normed_with_aut r V],
    ((BD.data.system κ r V r').obj (op $ of r' ((Lbar.functor.{0 0} r').obj S)))
      .is_weak_bounded_exact k K m c₀ :=
begin
  intro m,
  obtain ⟨k, K, hk, H⟩ := thm95''.profinite BD r r' κ m,
  obtain ⟨c₀, H⟩ := H ℤ,
  use [k, K, hk, c₀],
  introsI S V hV,
  specialize H S V,
  let i := (BD.data.system κ r V r').map_iso (HomZ_iso (of r' $ (Lbar.functor.{0 0} r').obj
  refine H.of_iso i.symm _,
  intros c n,
  rw ← system_of_complexes.apply_hom_eq_hom_apply,
  apply SemiNormedGroup.iso_isometry_of_norm_noninc;
  apply breen_deligne.data.complex.map_norm_noninc
end
```

(d) Liquid Tensor Experiment in **Lean**

[Demonstration]

Lean

1. We use Lean (Lean 4).
2. Lean is a functional programming language and proof assistant.
3. Lean was developed in 2013 by Leonardo de Moura (first at Microsoft, now at Amazon).
4. Lean 4 was released in 2021 and should not change anymore.
5. Structurally, it resembles Coq.
6. Mathematics in Lean is in MathLib Library!
7. **Relatively** easy to use and learn.
8. But there **are** many other options!

Lean Online

[Demonstration]

Logic from a different Perspective

- In classical mathematics, there is **Mathematics** and **Logic**.
- *Let* $f : S \to T$ *be* a **bijective function** of **sets**. *Then* f *is* **injective**.
- **Mathematics** vs. *Logic*!

$\Rightarrow$ We need math and logic in one place!

Proofs as Elements

[Demonstration]

Examples

Classical	**What we want**
Prop P	P : Prop
Formula for Set X	Function f: X $\to$ Prop
Proof for prop P	σ : P
Proof for $P \to Q$	Function $P \to Q$
x = y	(x = y) not empty

Proof for prop

$$\exists S \forall X \forall f, g \colon S \to X (f = g)$$

A set S and a proof f = g	A pair (S, π) $\pi \colon (X, f, g) \to (f = g)$

Examples in Lean

Demonstration