

Proofs with Computers: Introduction

Nima Rasekh

Universität Greifswald



11.04.2025

- **Goal:** To learn how to formulate and program mathematical and logical proofs with computers.
- **Today:**
 - Overview and rules for the lecture
 - Projects
 - History
 - Background
 - Logic
- **Starting next week:** Formalization in Lean

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

[Demonstration](#)

- The grade is determined by a project.
- A project is a formalization and a presentation.
- Everyone who does a project passes.
- 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!

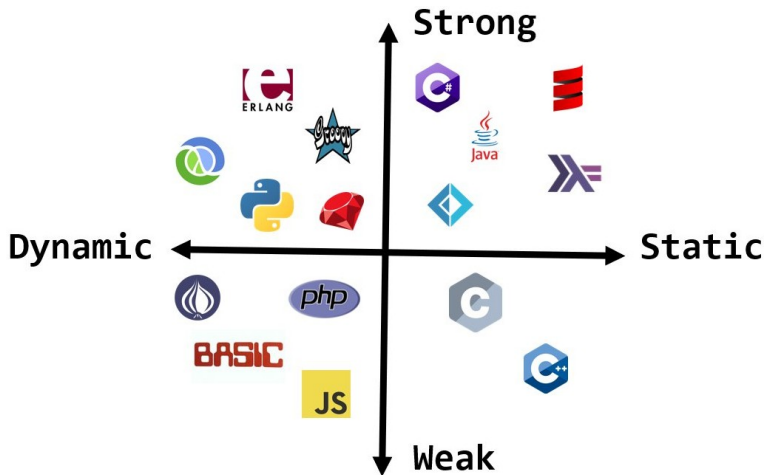
What is the Benefit of Computers?

- 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

```
1 From iris.algebra Require Export view.
2 From iris.algebra Require Import proofmode.classes big.op.
3 From iris.prelude Require Import options.
4
5 (** The authoritative camera with fractional authoritative elements *)
6 (** The authoritative camera has 2 types of elements: the authoritative
7  element  $[*q]$  a) and the fragment  $[ \square b ]$  (of which there can be several). To
8  enable sharing of the authoritative element  $[*q]$  a), it is equipped with a
9  fraction  $[q]$ . Updates are only possible with the full authoritative element
10  $[* a]$  (syntax for  $[* (1) a]$ )), while fractional authoritative elements have
11 agreement, i.e.,  $[ \surd (* (p1) a1 \cdot (* (p2) a2) \rightarrow a1 = a2 ]$ . *)
12
13 (** * Definition of the view relation *)
14 (** The authoritative camera is obtained by instantiating the view camera. *)
15 Definition auth_view_rel_raw {A : ucmra} {m : nat} {s b : A} : Prop :=
16   b <{m} s &  $\surd$ {s} s.
17 Lemma auth_view_rel_raw_mono (A : ucmra) n1 n2 (a1 a2 b1 b2 : A) :
18   auth_view_rel_raw n1 a1 b1  $\rightarrow$ 
19   a1  $\leq$  {n2} s a2  $\rightarrow$ 
20   b2 <{n2} b1  $\rightarrow$ 
21   n2  $\leq$  n1  $\rightarrow$ 
```

(a) Coq Iris

```
1 [-# OPTIONS --without-K -#]
2 -- https://upload.wikimedia.org/wikipedia/commons/e/e2/DNA-1.svg
3 -- http://www.fqas.org/rfcs/rfcs174.html
4 open import Data.Nat.Base using (N; zero; suc; _+_ _*_ _)
5 import Data.Vec as V
6 open V using (Vec; []; _::_)
7 open import Function using (_<_)
8 open import FunInverse.Core hiding (_<_)
9 open import Data.Fin using (Fin; zero; suc; #; inject+; raise) renaming (toN to FinN)
10
11 open import Solver.Linear
12
13 module sha1 where
14
15 Endo : Set  $\rightarrow$  Set
16 Endo A = A  $\rightarrow$  A
17
18 module FunSHA1
19   (t)
20   [T : Set t]
21   {funU : FunInverse T}
```

(c) Agda in Cryptography & Blockchain

```
1 %-----
2 % criteria_3D.pvs
3 % ACCORD v.1.8
4 %-----
5
6 criteria_3D[D,H:posreal] : THEORY
7 BEGIN
8
9   IMPORTING vertical_criterion[D,H],
10   horizontal_criterion_line[D]%,
11
12   sp      : VAR Sp_vect3 % 3-D separation
13   s,v,nv, : VAR Vec3
14   nev,nv1 : VAR Vec3
15   vo,v1    : VAR Nzv2_vect3
16   ephs,epsv : VAR Sign
17
18   criterion_3D?(sp,v,ephs,epsv)(nv) : bool =
19     (horizontal_sep?(sp) AND horizontal_criterion?(sp,ephs)(nv)) OR
20     (vertical_criterion?(epsv)(sp,v)(nv) AND
21      horizontal_lost?(sp) OR
```

(b) PVS in NASA

```
18 Inductive Tp where
19   | s (size : Nat)
20   | i (size : Nat)
21   | bi -- BigInt
22   | bool
23   | unit
24   | str (size : U 32)
25   | fstr (size : U 32) (argTps : List Tp)
26   | field
27   | slice (element : Tp)
28   | array (element : Tp) (size : U 32)
29   | tuple (name : Option String) (fields : List Tp)
30   | ref (tp : Tp)
31   | fn (argTps : List Tp) (outTp : Tp)
32
33 mutual
34
35 def tpsDecEq (a b : List Tp) : Decidable (a = b) := match a, b with
36 | [], [] => isTrue rfl
37 | [], _ :: _ => isFalse (by simp)
38 | _ :: _, [] => isFalse (by simp)
```

(d) Lean in Cryptography

Formalization in Mathematics

```
455 Lemma egn_genus : genus G = genus G'.
456 Proof.
457   rewrite (2)/genus/Euler_lba -egn_gcomp; have [e n f Eanf Ee En Ef] := eqG6'.
458   by rewrite /Euler_rhs /- (eq_fcard Ee) - (eq_fcard En) - (eq_fcard Ef).
459   Qed.
460
461 Lemma egn_planar : planar G = planar G'.
462 Proof. by rewrite (2)/planar -egn_genus. Qed.
463
464 End EqualityHypermap.
465
466 Notation "x '≡' y" := (eqn x y) (at level 70, no associativity).
467
468 Lemma egn_syn G G' : G ≡ G' => G' ≡ G.
469 Proof. by case: G => d e n f E0 [*]; apply: Eghypermap => x; auto. Qed.
470
471 Lemma dual_inv G : dual (dual G) ≡ G.
472 Proof.
473   case: G (@edge1 G) (@model G) (@face1 G) => d e n f /- sK sI sI fI.
474   by apply: Eghypermap; apply: fmv_inv.
475   Qed.
```

(a) Four color theorem in Coq

```
119 s1~Z/ZF-direct : GroupEquiv (n 4 S1) (ZBgroup 2)
120 s1~Z/ZF-direct = DirectProof.BrunerieGroupEquiv
121
122 -- This direct proof allows us to define a much simplified version of
123 -- the Brunerie number:
124 b1 : Z
125 b1 = fct DirectProof.computer q1'
126
127 -- This number computes definitionally to -2 in a few seconds!
128 b1~2 : b1 = -2
129 b1~2 = refl
130
131 -- As a sanity check we have proved (commented as typechecking is quite slow):
132 b1Spec : GroupEquiv (n 4 S1) (ZBgroup/ abs b1)
133 b1Spec = DirectProof.BrunerieGroupEquiv'
134
135 -- Combining all of this gives us the desired equivalence of groups by
136 -- computation as conjectured in Brunerie's thesis:
137 s1~Z/ZF-computation : GroupEquiv (n 4 S1) (ZBgroup/ 2)
138 s1~Z/ZF-computation = DirectProof.BrunerieGroupEquiv'
```

(c) Homotopy groups in Cubical Agda

```
237 let float_pow_pos_lo pp x_tm n =
238   let rec pow n =
239     match n with
240     | 0 -> INST[x_tm, x_var_real] float_pow_lo
241     | 1 -> INST[x_tm, x_var_real] float_pow_lo
242     | 2 ->
243       let mul_lo = float_mul_lo pp x_tm x_tm in
244       let lo_tm = lhand (concl mul_lo) in
245       let th0 = INST[x_tm, x_var_real; lo_tm, lo_var_real] float_pow2_lo in
246       MY_PROVE_HYP mul_lo th0
247   | _ ->
248     let _ = assert (n > 2) in
249     if (n land 1) = 0 then
250       (* even *)
251       let t_th = pow (n lsr 1) in
252       float_pow_pos_double_lo pp x_tm t_th
253     else
254       (* odd *)
255       let t_th = pow (n - 1) in
256       float_pow_pos_suc_lo pp x_tm t_th
257   in
```

(b) Kepler conjecture in Isabelle

```
115
116 /-- Theorem 9.4 in [Analytic] for weak bounded exactness -/
117 theorem thm94_weak' :
118   ∀ m : ℕ, ∃ (K K : ℝ≥0) (hk : fact (1 ≤ K)) (c₄ : ℝ≥0),
119   ∀ (S : Profinite) (V : SemiNormedGroup.{u}) [normed_with_aut r V],
120   ((BD.data.system x r V r').obj (op $ of r') ((Lbar.functor.{0 0} r').obj S)))
121   .is_weak_bounded_exact K K m c₄ :=
122   begin
123     intro m,
124     obtain (K, K, hk, h0) := thm94''.profinite BD r r' K m,
125     obtain (c₄, h0) := H_Z,
126     use [K, K, hk, c₄],
127     intro1 S V NV,
128     specialize H S V,
129     let i := (BD.data.system x r V r').map_iso (Hom2_iso (of r' $ (Lbar.functor.{0 0} r')).obj !
130       refine H.of_iso i.symm _),
131     intros c n,
132     rw - system.of_complexes.apply hom_eq_hom.apply,
133     apply SemiNormedGroup.iso_isometry_of_norm_noninc;
134     apply green_deline.data.complex_exp_norm_noninc
135   end
```

(d) Liquid Tensor Experiment in Lean

Demonstration

- 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!

[Demonstration](#)

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

\Rightarrow We need math and logic in one place!

Demonstration

| Classical | What we want |
|---|---|
| Prop P Formula for Set X Proof for prop P Proof for $P \rightarrow Q$ $x = y$ | $P : \text{Prop}$ Function $f: X \rightarrow \text{Prop}$ $\sigma : P$ Function $P \rightarrow Q$ $(x = y)$ not empty |

Proof for prop

$$\exists S \forall X \forall f, g: S \rightarrow X (f = g)$$

| | |
|----------------------------------|---|
| A set S and a proof $f = g$ | A pair (S, π) $\pi: (X, f, g) \rightarrow (f = g)$ |
|----------------------------------|---|

[Demonstration](#)