Beweise mit Computern: Einführung

Nima Rasekh

Universität Greifswald

11.04.2025

Übersicht

- **Ziel:** Zu lernen wie man mathematische und logische Beweise mit Computern formulieren und programmieren kann.
- **Plan:**
  - Erste Hälfte: Grundlagen von Lean.
  - Zweite Hälfte: Entscheiden wir später!
- **Heute:**
  - Übersicht und Regeln für die Vorlesung
  - Projekte
  - Geschichte
  - Hintergrund
  - Logik
- **Ab übernächste Woche:** Formalisierung in Lean

Logistik

1. Jeder muss Zugang zu einem Laptop haben!
2. Erste Schritte: Installation von VS Code, Lean und Git und Anmeldung in GitHub
3. Falls es Probleme gibt: **Fragen!**

GitHub Seite

[Demonstration]

Projekte

- Die Note wird durch Projekte bestimmt.
- Ein Projekt ist eine Formalisierung und eine Präsentation.
- Jeder der ein Projekt macht besteht.[1]
- Übungen sind optional, aber **dringend** empfohlen.
- Jeder, der eine Note braucht: Bis zum **02.05.2025** melden! Ansonsten kann man nicht bestehen.
- Nicht sicher? Dennoch melden!

[1] Terms and Conditions apply.

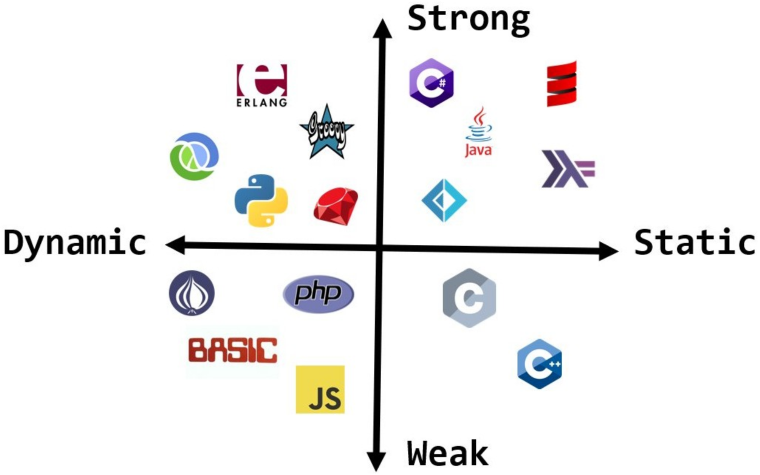
Was ist der Vorteil von Computern?

- Computer überprüfen Eingaben
- Computer automatisieren Prozesse
- Computer managen Daten

Anwendung von Computern in der Beweisführung von Programmen und Mathematik heißt:

Formalisierung!

Programmiersprachen



Dynamische vs. Statische Programmiersprachen

Demonstration

Woher kommt Formalisierung?

1. Erste Schritte in den 60ern und 70ern mit **Automath** und **Logic for Computable Functions**
2. In den 80ern und 90ern gibt es die ersten Sprachen die breitere Anwendung finden: **Coq**, **Isabelle**, **Agda**, **PVS**
3. Seit den 2000ern:
    - Neue Sprachen: **Metamath**, **Lean**, ...
    - Formalisierung von interessanten Problemen: **Vier-Farben-Satz** (2005), **Keplervermutung** (2014), ...
4. **Warum jetzt?** Benutzerfreundlicher, KI, ...

Formalisierung im Programmieren

```coq
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

```agda
{-# 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 Kryptographie & Blockchain

```lean
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 Kryptographie

```
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) Vier-Farben-Satz 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) Keplervermutung 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) Homotopiegruppen 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**

100 Probleme

Demonstration

Lean

1. Wir benutzen Lean (Lean 4).
2. Lean ist eine funktionale Programmiersprache und ein Beweisassistent.
3. Lean wurde in 2013 von Leonardo de Moura (erst in Microsoft, jetzt in Amazon) entwickelt.
4. Lean 4 wurde in 2021 veröffentlicht und sollte stabil sein.
5. Von der Struktur ähnelt es Coq
6. Mathematik in Lean ist in MathLib Library!
7. **Relativ** einfach zu benutzen und zu lernen.
8. Aber es **gibt** viele andere Optionen!

Lean Online

Demonstration

Logik neu denken

- In der klassischen Mathematik gibt es **Mathe** und **Logik**.
- *Es sei* $f : S \to T$ eine **bijektive Abbildung** von **Mengen**. *Dann ist* f **injektiv**.
- **Mathe** vs. *Logik*!

$\Rightarrow$ Wir brauchen Mathe und Logik an einem Ort!

Beweise als Elemente

Demonstration

Beispiele

Klassisch	Was wir wollen
Satz P	P : Prop
Formel für Menge X	Abbildung f: X $\to$ Prop
Beweis für Satz P	$\sigma : P$
Beweis für $P \to Q$	Abbildung $P \to Q$
x = y	(x = y) ist nicht leer

Beweis für Satz

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

Eine Menge S und ein Beweis f = g	Ein Paar (S, π) $\pi \colon (X, f, g) \to (f = g)$

Beispiel in Lean

Demonstration