

Beweise mit Computern: Einführung

Nima Rasekh

Universität Greifswald



11.04.2025

- **Ziel:** Zu lernen wie man mathematische und logische Beweise mit Computern formulieren und programmieren kann.
- **Heute:**
 - Übersicht und Regeln für die Vorlesung
 - Projekte
 - Geschichte
 - Hintergrund
 - Logik
- **Ab übernächste Woche:** Formalisierung in Lean

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

[Demonstration](#)

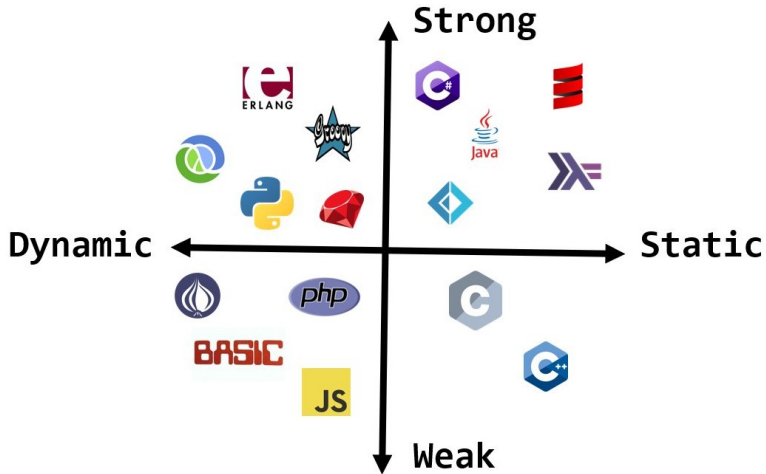
- Die Note wird durch Projekte bestimmt.
- Ein Projekt ist eine Formalisierung und eine Präsentation.
- Jeder der ein Projekt macht besteht.
- Übungen sind optional, aber **dringend** empfohlen.
- Jeder, der eine Note braucht: Bis zum **02.05.2025** melden!
Ansonsten kann man nicht bestehen.
- Nicht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

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!



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

```

2  From Iris.algebra Require Export view.
3  From Iris.algebra Require Import proofmode_classes big_op.
4  From Iris.prelude Require Import options.

5  (** The authoritative camera with fractional authoritative elements *)
6  (** The authoritative camera has 2 types of elements: the authoritative
7  element  $\llbracket a \rrbracket$  and the fragment  $\llbracket b \rrbracket$  (of which there can be several). To
8  enable sharing of the authoritative element  $\llbracket a \rrbracket$ , it is equipped with a
9  fraction  $\llbracket a \rrbracket$ . Updates are only possible with the full authoritative element
10  $\llbracket a \rrbracket$  (syntax for  $\llbracket a \rrbracket$ ), while fractional authoritative elements have
11 agreed, i.e.,  $\llbracket a \rrbracket \cdot \llbracket a_1 \rrbracket \leq \llbracket a_2 \rrbracket \rightarrow a_1 \leq a_2$ .)

12
13 (** * Definition of the view relation *)
14
15 (** The authoritative camera is obtained by instantiating the view camera. *)
16
17 Definition auth_view_rel_raw (A : ucmra) (n : nat) (a b : A) : Prop :=
18   b  $\leq$  n? a  $\wedge$   $\neg$  b? a.
19
20 Lemma auth_view_rel_raw_mono (A : ucmra) n1 n2 (a1 a2 b1 b2 : A) :
21   auth_view_rel_raw n1 a1 b1  $\wedge$ 
22   a1  $\leq$  n2? a2  $\wedge$ 
23   a2  $\leq$  (n2) b1  $\rightarrow$ 
24   b2  $\leq$  n1  $\wedge$ 

```

(a) Coq Iris

```

1  [-# OPTIONS -without-K-A-]
2  -- https://upload.wikimedia.org/wikipedia/commons/e/e2/SHA-1.svg
3  -- http://www.faqs.org/rfcs/rfc3174.html
4
5  open import Data.Nat.Base using (N; zero; suc; _+_ _+_ _)
6  import Data.Vec as V
7
8  open V using (Vec; []; _::_)
9
10 open import Function using (_>_)
11
12 open import Data.Fin using (Fin; zero; suc; #_; inject; raise) renaming (toN to FinN)
13
14
15
16
17
18 open import Solver.Linear
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

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

```

2 %-----
3 % criteria_3D.pvs
4 % ACCORD v.1.0
5 %-----
6
7 criteria_3D[D,H:posreal] : THEORY
8 BEGIN
9
10   IMPORTING vertical_criterion[D,H],
11         horizontal_criterion_line[D],
12
13   sp : VAR Sp_vect3 % 3-D separation
14   s,v,w :
15   mvo,mvi : VAR Vect3
16   vo,v1 : VAR Nnz2_vect3
17   spsh,spov : VAR Sign
18
19   criterion_3D?(sp,v,spsh,spov)(nv) : bool =
20     (horizontal_seg?(sp) AND horizontal_criterion?(sp,spsh)(nv)) OR
21     (vertical_criterion?(spov)(sp,v)(nv) AND
22       (horizontal_los?(sp) OR

```

(b) PVS in NASA

```

18   inductive Tp where
19   | u (size : Nat)
20   | i (size : Nat)
21   | bi -- BigInt
22   | bool
23   | unit
24   | str (size : U 32)
25   | fwtStr (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 postDecEq (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 Kryptographie

Formalisierung in der Mathematik

```
455 Lemma eqm_genus : genus G = genus G'.
456 Proof.
457   rewrite (2)/genus /Euler_lha -eqm_gcomp; have [e n f Eanf Ee En Ef] := eqG6'.
458   by rewrite /Euler_rhs /~ -(eq_fcand Ee) -(eq_fcand En) -(eq_fcand Ef).
459   Qed.
460
461 Lemma eqm_planar : planar G = planar G'.
462 Proof. by rewrite (2)/planar -eqm_genus. Qed.
463
464 End EqualityHypermap.
465
466 Notation "x '==>' y" := (eqm x y) (at level 70, no associativity).
467
468 Lemma eqm_syn G G'; G ==> G' -> G' ==> G.
469 Proof. by case: G ==> d e n f E0 []; apply: EqHypermap ==> 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 /~ nK e1 n1 f1.
474   by apply: EqHypermap; apply: finv_inv.
475   Qed.
```

(a) Vier-Farben-Satz in Coq

```
119 M5~Z/ZF-direct : GroupEquiv (n 4 S3) (ZGroup/ 2)
120 M5~Z/ZF-direct = DirectProof.BrunerieGroupEquiv
121
122 -- This direct proof allows us to define a much simplified version of
123 -- the Brunerie number:
124 B' : Z
125 B' = fval DirectProof.computer q'
126
127 -- This number computes definitionally to -2 in a few seconds!
128 B'' : Z
129 B'' = -2
130 B'' = refl
131
132 -- As a sanity check we have proved (commented as typechecking is quite slow):
133 B' ==> B'' : GroupEquiv (n 4 S3) (ZGroup/ abs B')
134 B' ==> B'' : DirectProof.BrunerieGroupEquiv
135
136 -- Combining all of this gives us the desired equivalence of groups by
137 -- computation as conjectured in Brunerie's thesis:
138 M5~Z/ZF-computation : GroupEquiv (n 4 S3) (ZGroup/ 2)
139 M5~Z/ZF-computation = DirectProof.BrunerieGroupEquiv'
```

(c) Homotopiegruppen 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_pow0_lo
241     | 1 -> INST[x_tm, x_var_real] float_pow1_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) Keplervermutung in Isabelle

```
115
116 /-- Theorem 9.4 in [Analytic] for weak bounded exactness -/
117 theorem th94_weak' :
118   ∀ M : N, ∃ (K K' : ℝ≥0) (hk : fact (1 ≤ K)) (Cv : 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 Cv :=
122   begin
123     intro M,
124     obtain (K, K', M0) := th95''.profinite BD r r' K M,
125     obtain (Cv, M0) := H_Z,
126     use [K, K', hk, Cv],
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 i)
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.map_norm_noninc
135   end
```

(d) Liquid Tensor Experiment in Lean

Demonstration

- 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!

[Demonstration](#)

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

⇒ Wir brauchen Mathe und Logik an einem Ort!

Demonstration

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

Beweis für Satz

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

Eine Menge S und ein Beweis $f = g$	Ein Paar (S, π) $\pi: (X, f, g) \rightarrow (f = g)$
--	---

Demonstration