NEURAL INFORMATION PROCESSING SYSTEMS

Machine Learning for Theorem Proving

Kaiyu Yang Albert Q. Jiang Emily First

Caltech UNIVERSITY OF CAMBRIDGE UC San Diego

Speakers

Kaiyu Yang

Kaiyu is a postdoc at Caltech, working on machine learning for formal theorem proving

Albert Q. Jiang

Albert is a Ph.D. student at Cambridge, working on mathematical reasoning with language models

Emily First

Emily is a postdoc at UCSD, working on automatically generating proofs of software correctness

Panelists

Anima Anandkumar
Caltech

Zhangir Azerbayev
Princeton

Noah Goodman
Stanford

Alex Sanchez-Stern
UMass Amherst

Dawn Song
UC Berkeley

Sean Welleck
UW, AI2 -> CMU

Outline

Presentation (2 hours)

Panel (30 minutes)

LLMs, mathematical reasoning, code generation, verification, AI4Science, and more!

Outline

Presentation (2 hours)

- **Part I: Fundamentals**
  - What is theorem proving? Why is it important for AI?
  - Demo: a simple LLM-based prover

Panel (30 minutes)

LLMs, mathematical reasoning, code generation, verification, AI4Science, and more!

Outline

Presentation (2 hours)

- **Part I: Fundamentals**
  - What is theorem proving? Why is it important for AI?
  - Demo: a simple LLM-based prover

- **Part II: Advanced topics**
  - Recent work and open problems
  - Machine learning, mathematics, and natural language
  - Machine learning for software verification

Panel (30 minutes)

LLMs, mathematical reasoning, code generation, verification, AI4Science, and more!

Outline

Presentation (2 hours)

- **Part I: Fundamentals**
  - What is theorem proving? Why is it important for AI?
  - Demo: a simple LLM-based prover

- **Part II: Advanced topics**
  - Recent work and open problems
  - Machine learning, mathematics, and natural language
  - Machine learning for software verification

Panel (30 minutes)

LLMs, mathematical reasoning, code generation, verification, AI4Science, and more!

Teaser: LLMs as Copilots for Theorem Proving

Formal Theorem Proving

Theorem

Proof

Formal Theorem Proving

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

- Theorems/proofs represented formally as programs

Formal Theorem Proving

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Leroy et al., "CompCert - A Formally Verified Optimizing Compiler", 2016]

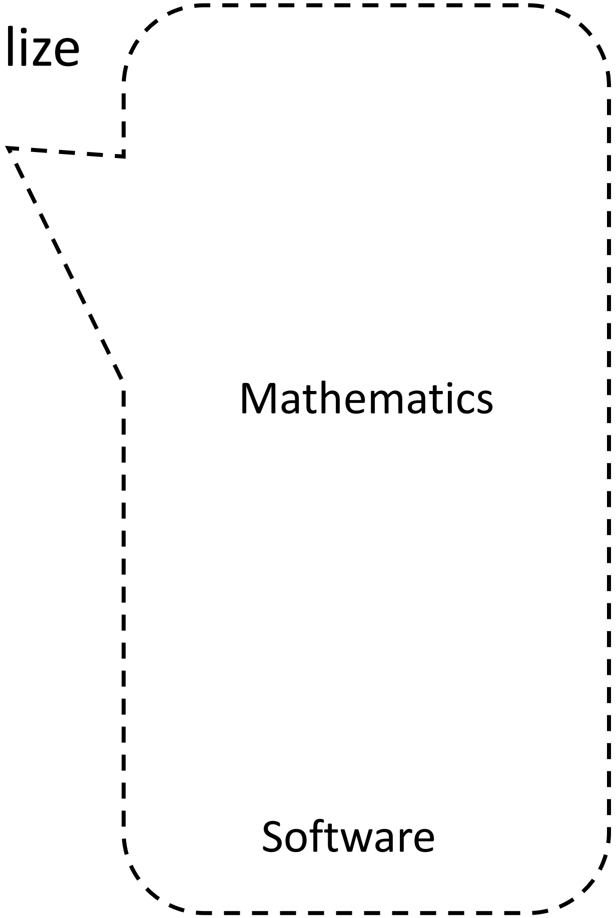
Formalize

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```



- Theorems/proofs represented formally as programs

Formal Theorem Proving

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Leroy et al., "CompCert - A Formally Verified Optimizing Compiler", 2016]

Formalize

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

Mathematics

Software

- Theorems/proofs represented formally as programs
- Proofs can be checked easily

Formal Theorem Proving

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Leroy et al., "CompCert - A Formally Verified Optimizing Compiler", 2016]

Formalize

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

✓ ✗

Mathematics

Software

- Theorems/proofs represented formally as programs
- Proofs can be checked easily

Why is Theorem Proving Important for AI?

The Era of Large Language Models (LLMs)

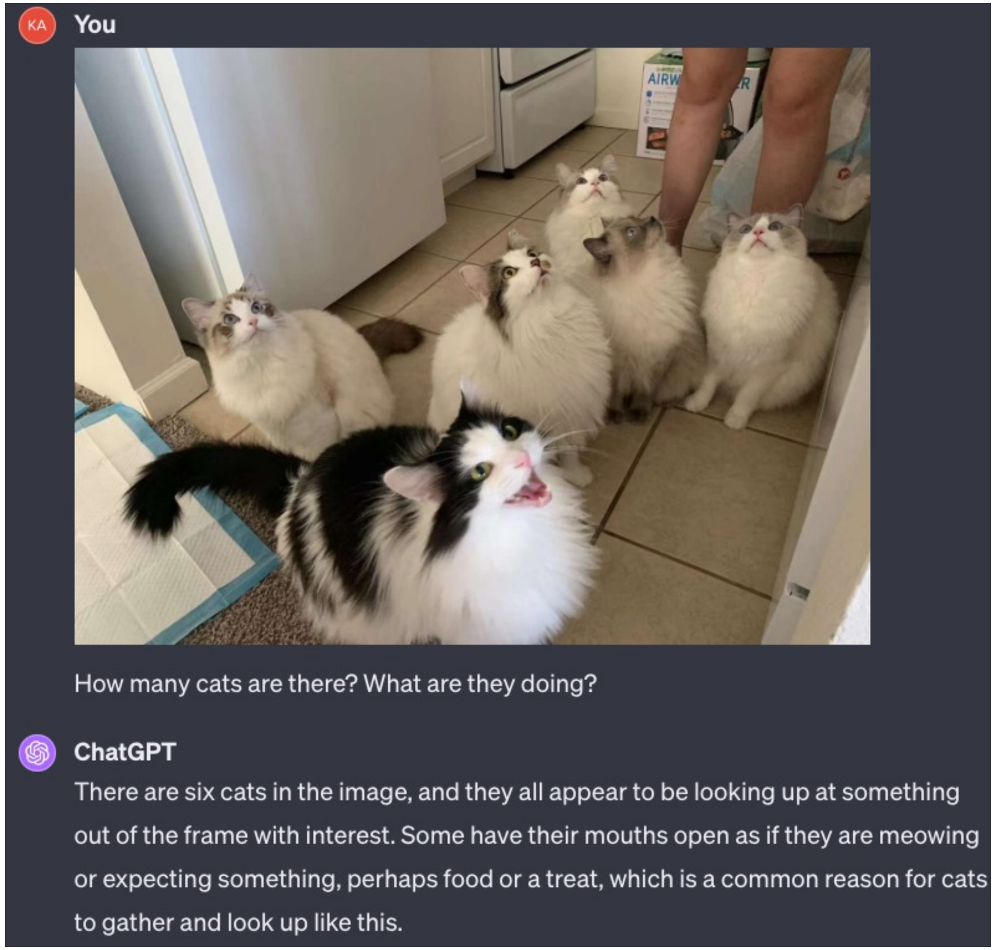
GPT-4

Claude

GitHub Copilot

[Ma et al., Eureka, 2023]

[Wang et al., Voyager, 2023]



Theorem Proving and LLMs

Theorem proving

Mathematical reasoning
with LLMs

Code generation
with LLMs

Theorem Proving and LLMs

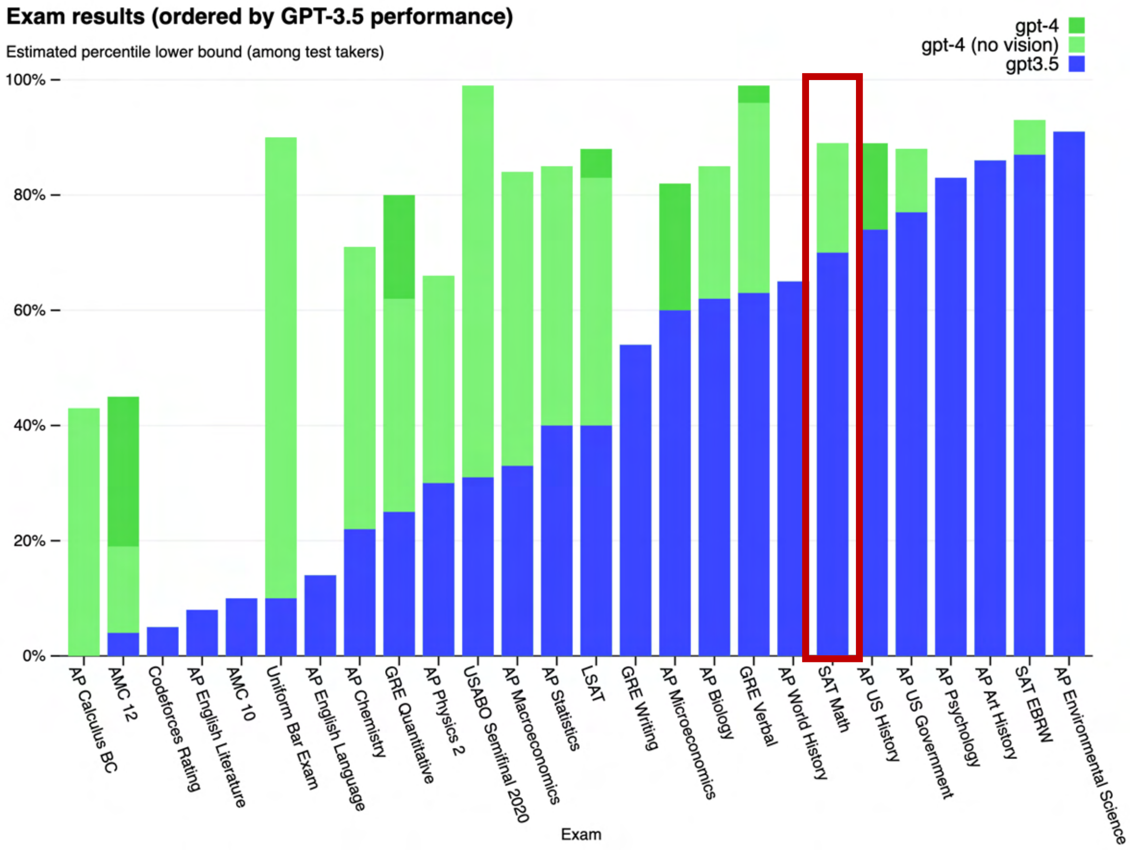
Theorem proving

Mathematical reasoning
with LLMs

Code generation
with LLMs

Mathematical Reasoning with LLMs

- GPT-4 scored 89th percentile on SAT Math



Mathematical Reasoning with LLMs

- GPT-4 scored 89th percentile on SAT Math
- Specialized math LLMs: Minerva, MetaMath, WizardMath, MAmmoTH, Llemma

Question: For every $a, b, b \neq a$ prove that

$$\frac{a^2+b^2}{2} > \left(\frac{a+b}{2}\right)^2.$$

Model output:

$$\frac{a^2+b^2}{2} > \left(\frac{a+b}{2}\right)^2$$

$$\iff \frac{a^2+b^2}{2} > \frac{a^2+b^2+2ab}{4}$$

$$\iff a^2+b^2 > \frac{a^2+b^2+2ab}{2}$$

$$\iff 2a^2+2b^2 > a^2+b^2+2ab$$

$$\iff a^2+b^2 > 2ab$$

$$\iff a^2+b^2-2ab > 0$$

$$\iff (a-b)^2 > 0$$

which is true, because the square of a real number is positive.

[Lewkowycz et al., **Minerva**, 2022]

Input:
Let $f(r) = \sum_{j=2}^{2008} \frac{1}{j^r} = \frac{1}{2^r} + \frac{1}{3^r} + \cdots + \frac{1}{2008^r}$. Find $\sum_{k=2}^{\infty} f(k)$.

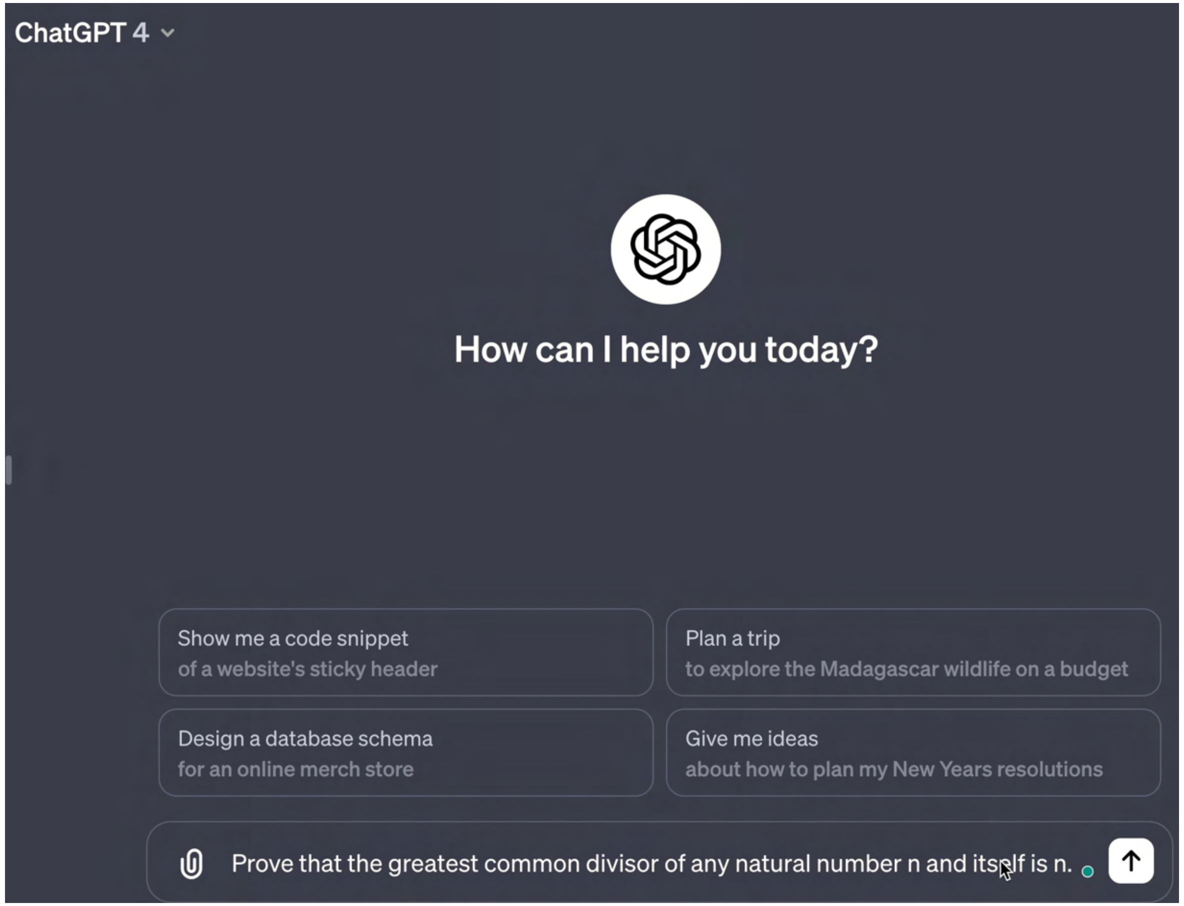
LLEMMA 34B solution:
We have

$$\begin{aligned}
\sum_{k=2}^{\infty} f(k) &= \sum_{k=2}^{\infty}\sum_{j=2}^{2008} \frac{1}{j^k} = \sum_{j=2}^{2008}\sum_{k=2}^{\infty} \frac{1}{j^k} = \sum_{j=2}^{2008} \frac{1}{j^2}\sum_{k=0}^{\infty} \frac{1}{j^k} = \sum_{j=2}^{2008} \frac{1}{j^2}\frac{1}{1-\frac{1}{j}} \\
&= \sum_{j=2}^{2008} \frac{1}{j(j-1)} = \sum_{j=2}^{2008} \left(\frac{1}{j-1} - \frac{1}{j}\right) \\
&= \left(\frac{1}{1} - \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{1}{3}\right) + \cdots + \left(\frac{1}{2007} - \frac{1}{2008}\right) \\
&= 1 - \frac{1}{2008} \\
&= \boxed{\frac{2007}{2008}}.
\end{aligned}$$

Final Answer: The final answer is $\frac{2007}{2008}$.

[Azerbayev et al., **Llemma**, 2023]

Informal vs. Formal Mathematical Reasoning




```
theorem gcd_self (n : ℕ) : gcd n n = n
```

```
cases n
· unfold gcd
  rfl
· unfold gcd
  rw [mod_self]
  unfold gcd
  rfl
```

```
cases n
· simp [gcd]
· rw [mod_self]
  unfold gcd
  rfl
```

Important for LLMs to tackle advanced mathematics

- Grounded in environments that can provide feedback
- Simple and rigorous evaluation: formal proofs can be checked (no hallucination)

Informal ⟷ **Formal**

Checking Mathematical Proofs is Hard for Humans

Quantamagazine

Titans of Mathematics Clash Over Epic Proof of ABC Conjecture

Two mathematicians have found what they say is a hole at the heart of a proof that has convulsed the mathematics community for nearly six years.

Theorem Proving and LLMs

Theorem proving

Mathematical reasoning
with LLMs

Code generation
with LLMs

Theorem Proving and LLMs

Theorem proving

Mathematical reasoning with LLMs

Code generation with LLMs

Code Generation with LLMs

```python
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

Passing a few testing examples ≠ correctness

Code Generation with LLMs

```
In [3]: gcd(-10, -5)
```

What if x and y are negative?

```python
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

GitHub Copilot

Passing a few testing examples ≠ correctness

Code Generation with LLMs

```python
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

GitHub Copilot

```
In [3]: gcd(-10, -5)
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
Cell In[3], line 1
----> 1 gcd(-10, -5)

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14         return x
     15 if x < y:
---> 16         return gcd(x, y % x)
     17 return gcd(x % y, y)

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14         return x
     15 if x < y:
---> 16         return gcd(x, y % x)
     17 return gcd(x % y, y)

    [... skipping similar frames: gcd at line 16 (2981 times)]

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14         return x
     15 if x < y:
---> 16         return gcd(x, y % x)
     17 return gcd(x % y, y)

File ~/LeanDojo/tmp.py:11, in gcd(x, y)
      2 def gcd (x : int, y : int) -> int:
      3     """Compute the greatest common divisor of ``x`` and ``y``.
      4     >>> gcd(10, 5)
      5     5
   (...)
      9     4
     10     """
---> 11     if x == 0:
     12         return y
     13     if y == 0:

RecursionError: maximum recursion depth exceeded in comparison
```

Passing a few testing examples ≠ correctness

How Can We Trust AI-Generated Code?

Freethink

GitHub CEO says Copilot will write 80% of code "sooner than later"

Theorem Proving for Verified Code Generation

- Generate code + formal specification (theorem) + formal proof

[Sun and Sheng et al., "Clover: Closed-Loop Verifiable Code Generation", 2023]

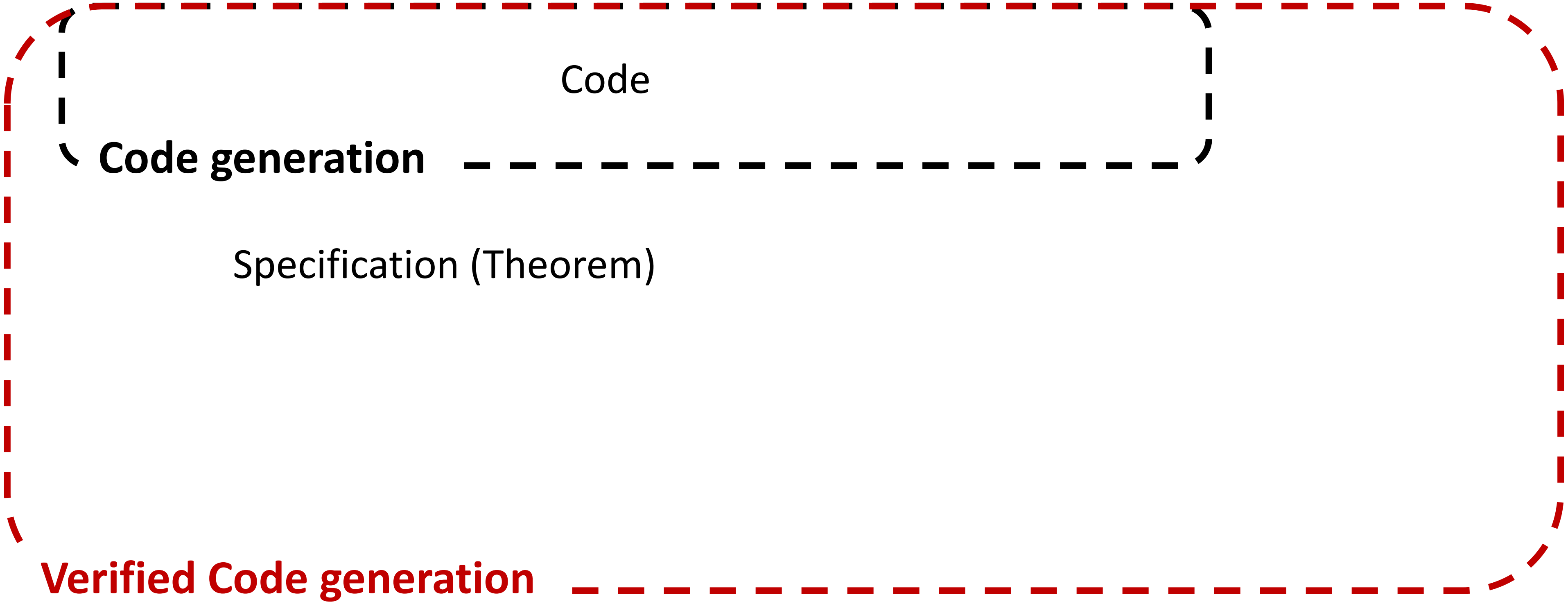
Code

Code generation

Theorem Proving for Verified Code Generation

- Generate code + formal specification (theorem) + formal proof

[Sun and Sheng et al., "Clover: Closed-Loop Verifiable Code Generation", 2023]



Theorem Proving for Verified Code Generation

- Generate code + formal specification (theorem) + formal proof

[Sun and Sheng et al., "Clover: Closed-Loop Verifiable Code Generation", 2023]

Code

Code generation

Specification (Theorem)

Proof

Verified Code generation

Theorem Proving and LLMs: Takeaways

Theorem proving

Mathematical reasoning with LLMs

Code generation with LLMs

- Elementary math -> advanced math
- Verified code generation
- Feedback & evaluation at scale: AI mathematicians/programmers

How to Prove Theorems (with Machine Learning)?

Proof Assistants (Interactive Theorem Provers)

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

⬇

Proof

```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

Proof Assistants (Interactive Theorem Provers)

Humans + Proof assistants

IDEs for writing formal proofs

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

Examples of Proof Assistants

Isabelle
[Nipkow et al., 2002]

- Large formal libraries: ~250K proofs

Coq
[Barras et al., 1997]

- >100K proofs in different repos
- Popular for software verification, e.g., CompCert [Leroy et al., 2016]

Lean
[de Moura et al., 2015]

- ~100K proofs in Mathlib
- Liquid tensor experiment [Commelin, 2022]
- Polynomial Freiman-Ruzsa conjecture (led by Terence Tao)

Examples of Proof Assistants

Isabelle
[Nipkow et al., 2002]

- Large formal libraries: ~250K proofs

[First et al., Baldur, 2023]
[Jiang et al., Thor, 2022]
[Mikuła et al., Magnushammer, 2023]
[Jiang et al., DSP, 2023]
[Wu et al., Autoformalization, 2022]
[Li et al., IsarStep, 2021]
[Wang and Xin et al., LEGO-Prover, 2023]

Coq
[Barras et al., 1997]

- >100K proofs in different repos
- Popular for software verification, e.g., CompCert [Leroy et al., 2016]

[Huang et al., GamePad, 2018]
[Yang and Deng, CoqGym, 2019]
[Sivaraman, et al., Lemma Synthesis, 2022]
[Sanchez-Stern et al., Proverbot9001, 2020]
[Ringer et al., REPLica, 2020]
[Sanchez-Stern and First et al., Passport, 2023]

Lean
[de Moura et al., 2015]

- ~100K proofs in Mathlib
- Liquid tensor experiment [Commelin, 2022]
- Polynomial Freiman-Ruzsa conjecture (led by Terence Tao)

[Han et al., PACT, 2022]
[Polu et al., 2023]
[Lample et al., HTPS 2022]
[Want et al., DT-Solver, 2023]
[Yang et al., LeanDojo, 2023]
[Thakur et al., COPRA, 2023]

Examples of Proof Assistants

Isabelle

[Nipkow et al., 2002]

- Large formal libraries: ~250K proofs

[First et al., Baldur, 2023]
[Jiang et al., Thor, 2022]
[Mikuła et al., Magnushammer, 2023]
[Jiang et al., DSP, 2023]
[Wu et al., Autoformalization, 2022]
[Li et al., IsarStep, 2021]
[Wang and Xin et al., LEGO-Prover, 2023]

Coq

[Barras et al., 1997]

- >100K proofs in different repos
- Popular for software verification, e.g., CompCert [Leroy et al., 2016]

[Huang et al., GamePad, 2018]
[Yang and Deng, CoqGym, 2019]
[Sivaraman, et al., Lemma Synthesis, 2022]
[Sanchez-Stern et al., Proverbot9001, 2020]
[Ringer et al., REPLica, 2020]
[Sanchez-Stern and First et al., Passport, 2023]

Lean

[de Moura et al., 2015]

- ~100K proofs in Mathlib
- Liquid tensor experiment [Commelin, 2022]
- Polynomial Freiman-Ruzsa conjecture (led by Terence Tao)

[Han et al., PACT, 2022]
[Polu et al., 2023]
[Lample et al., HTPS 2022]
[Want et al., DT-Solver, 2023]
[Yang et al., LeanDojo, 2023]
[Thakur et al., COPRA, 2023]

Proving Theorems Using Language Models

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  search_proof
```

Proving Theorems Using Language Models

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  search_proof
```

⬇

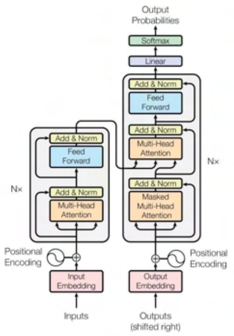
```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Proving Theorems Using Language Models

Input: Theorem

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b
```

```
intro a b c
rw [Nat.add_right_comm
```



[Vaswani et al., Transformer, 2017]

Output: Proof

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Proof state

⊢ ∀ (a b c : ℕ), a + b + c = a + c + b

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Proof state

Tactic

⊢ ∀ (a b c : ℕ), a + b + c = a + c + b

intro a b c

a b c: ℕ
⊢ a + b + c = a + c + b

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Proof state

Tactic

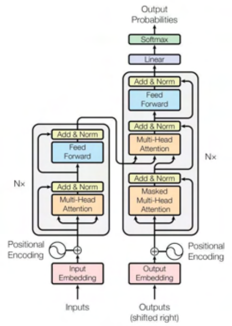
⊢ ∀ (a b c : ℕ), a + b + c = a + c + b $\xrightarrow{\text{intro a b c}}$ a b c: ℕ
⊢ a + b + c = a + c + b $\xrightarrow{\text{rw [Nat.add\_right\_comm]}}$ ✓

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Tactic generator

Input: Proof state



Output: Tactic

⊢ ∀ (a b c : ℕ), a + b + c = a + c + b

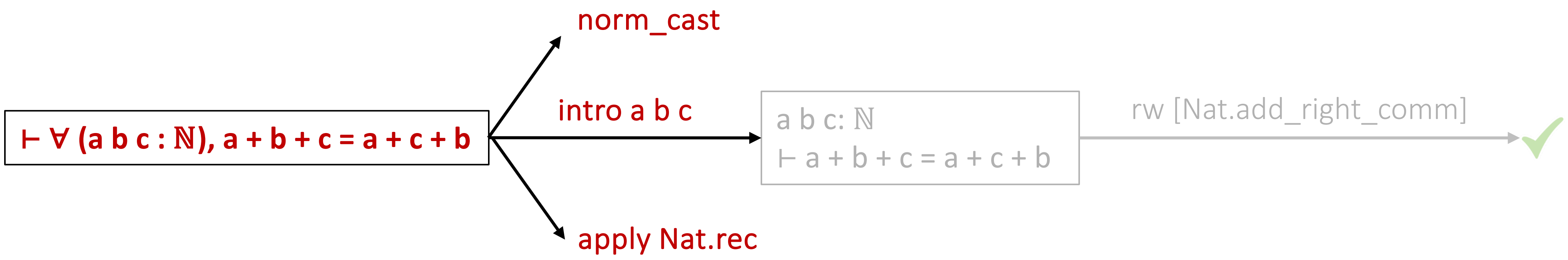
intro a b c

a b c: ℕ
⊢ a + b + c = a + c + b

rw [Nat.add_right_comm]

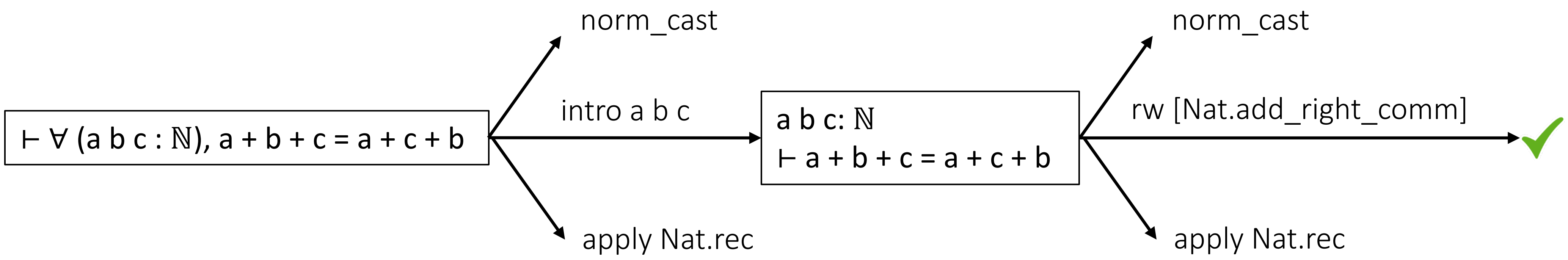
Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```



Searching for Proofs

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

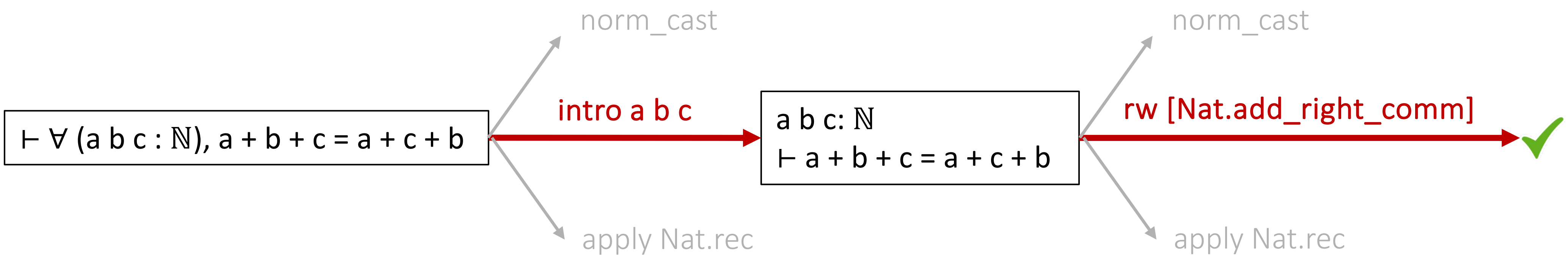


Searching for Proofs

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Classical proof search algorithms

- Depth first search (DFS)
- Breadth first search (BFS)
- …

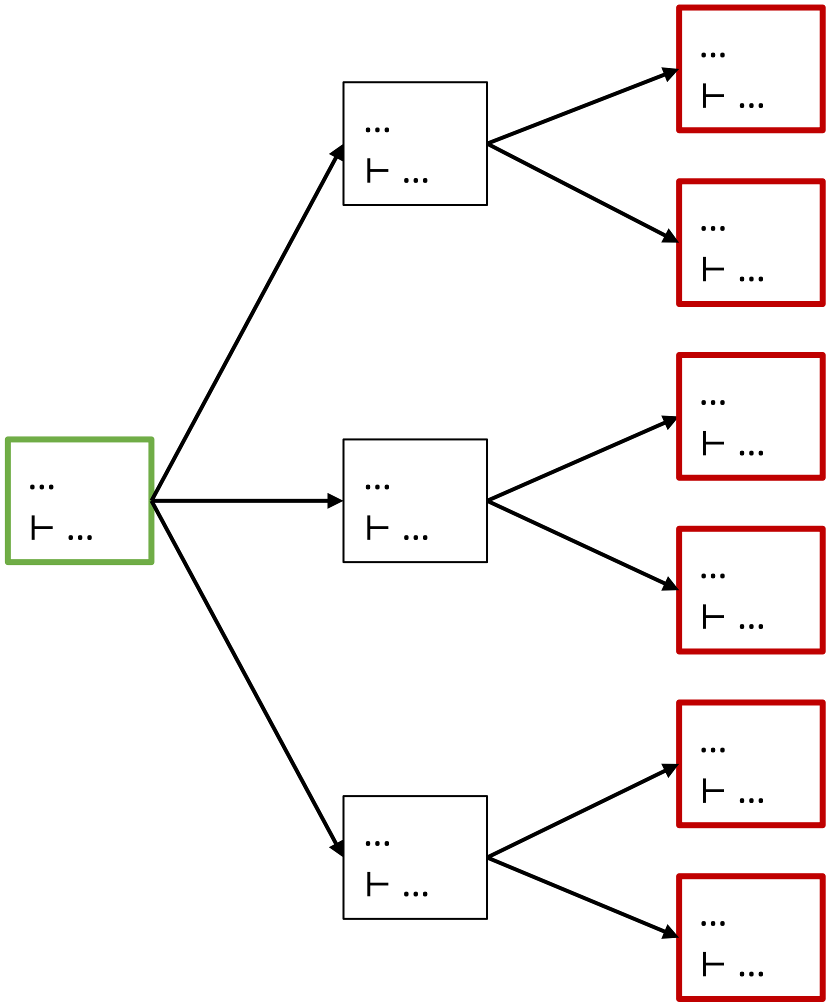


Demo:
A Simple Theorem Prover Using Language Models

Improving the Simple Prover

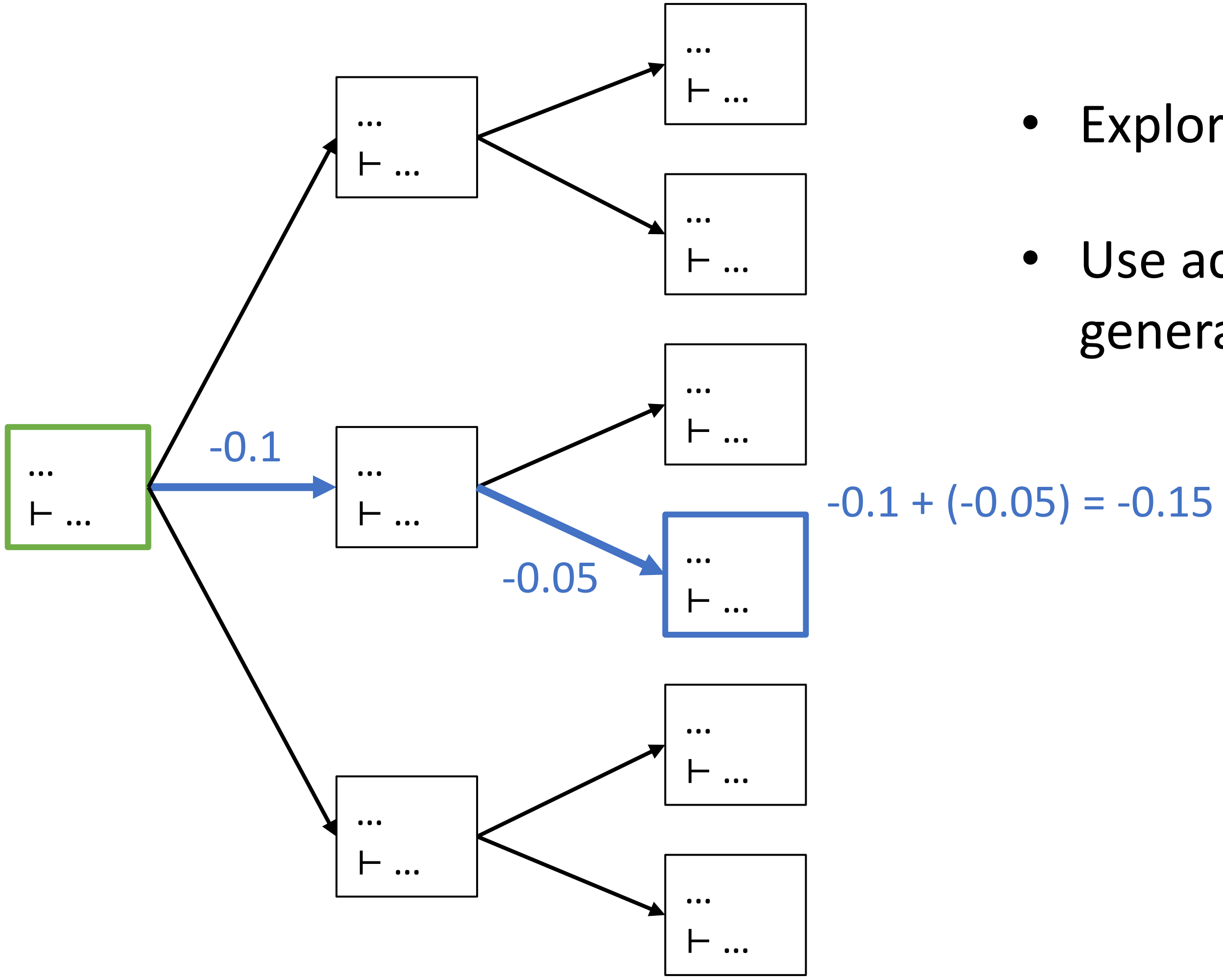
- Proof search
- Premise selection

Best First Search



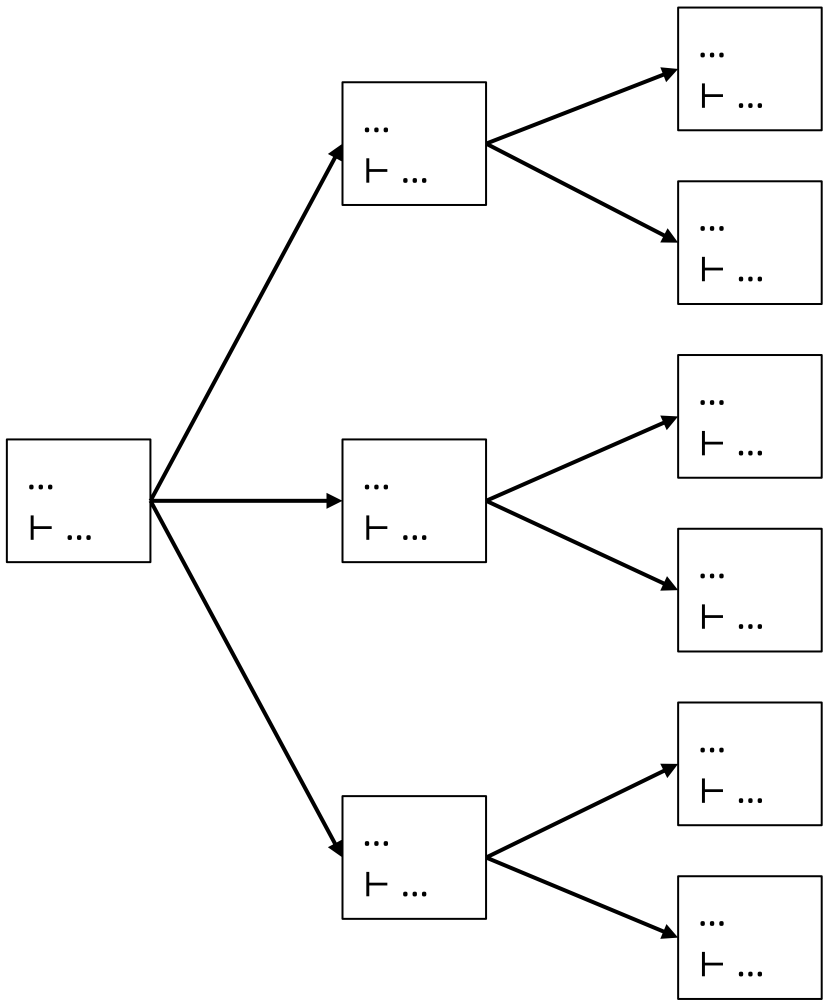
- Explore the most promising node
- Use accumulated scores from the tactic generator to rank the nodes

Best First Search



- Explore the most promising node
- Use accumulated scores from the tactic generator to rank the nodes

Best First Search



- Explore the most promising node
- Use accumulated scores from the tactic generator to rank the nodes
- **Simple and widely used**

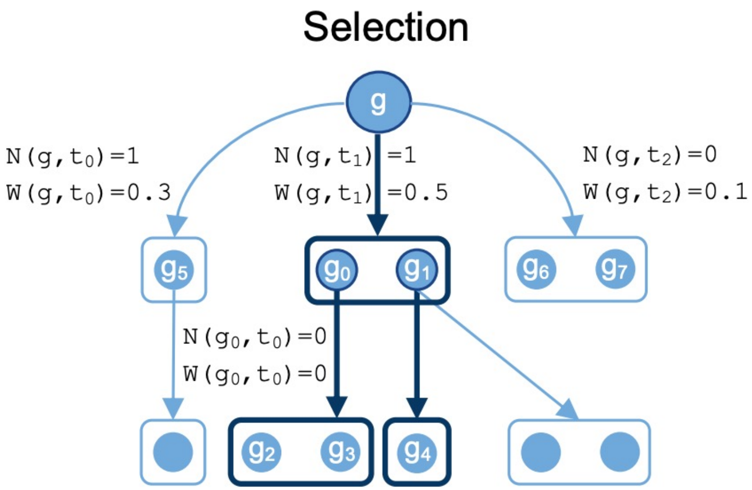
[Han et al., PACT, ICLR 2022]
[Polu et al., ICLR 2023]
[Jiang et al., Thor, NeurIPS 2022]
[Yang et al., LeanDojo, NeurIPS 2023]

Hyper Tree Proof Search

- Inspired by Monte Carlo Tree Search (MCTS)
- Update visit counts and estimated values for each node

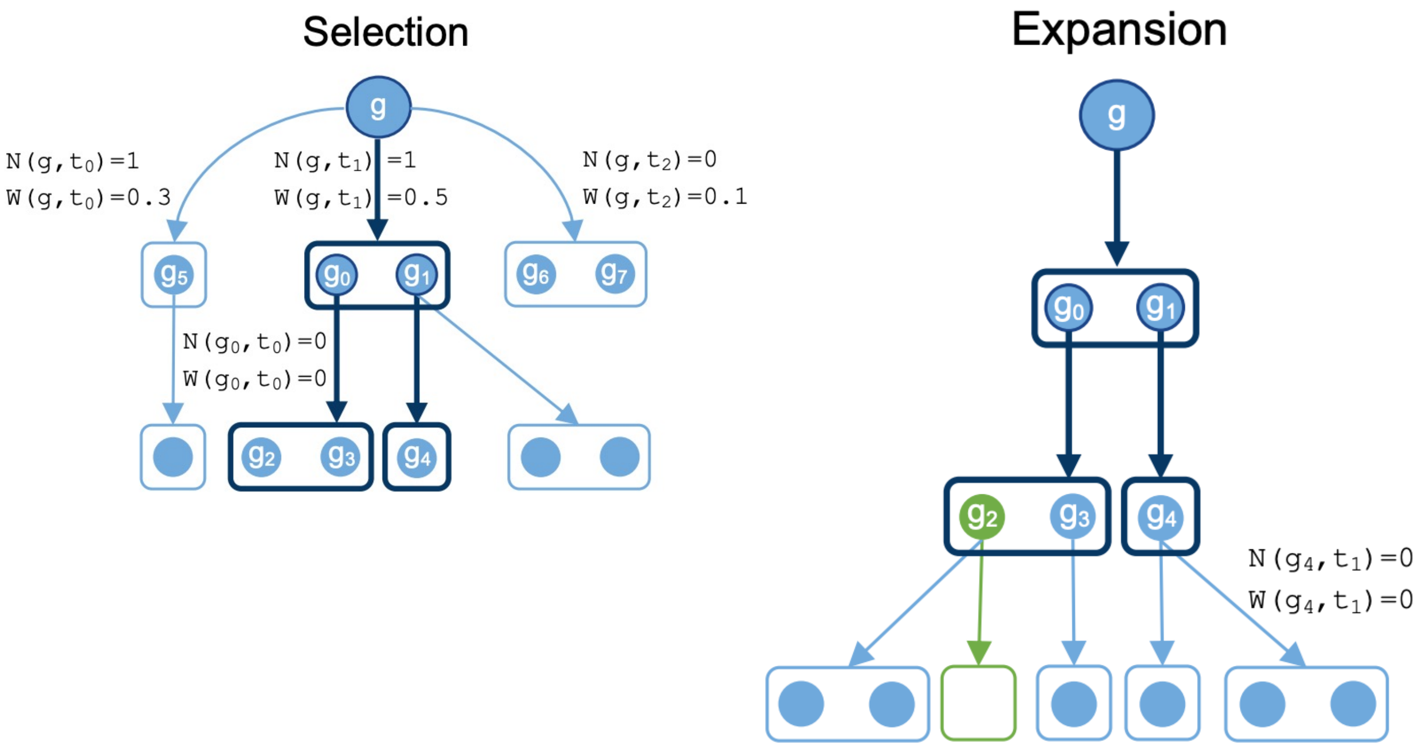
Hyper Tree Proof Search

- Inspired by Monte Carlo Tree Search (MCTS)
- Update visit counts and estimated values for each node



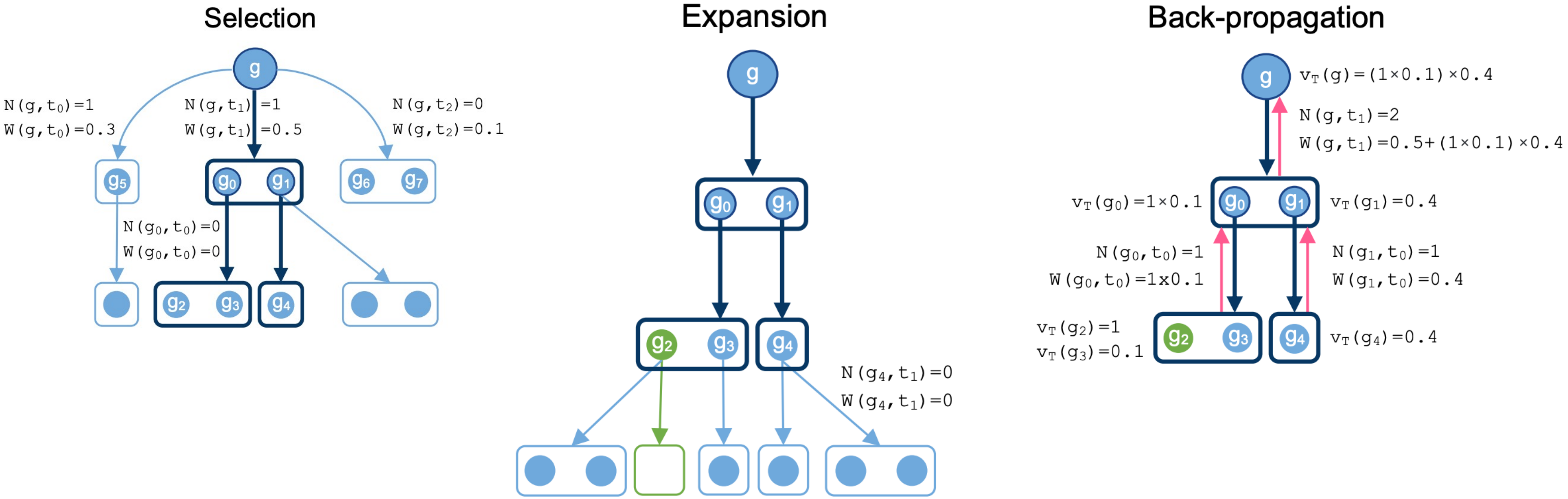
Hyper Tree Proof Search

- Inspired by Monte Carlo Tree Search (MCTS)
- Update visit counts and estimated values for each node



Hyper Tree Proof Search

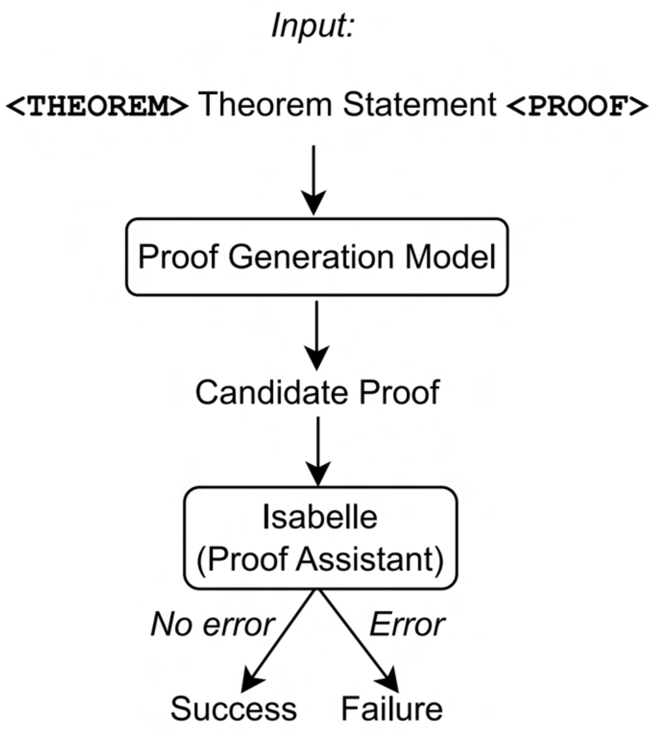
- Inspired by Monte Carlo Tree Search (MCTS)
- Update visit counts and estimated values for each node



[First et al., "Baldur: Whole-Proof Generation and Repair with Large Language Models", FSE 2023]

Is Proof Search Really Necessary?

- Baldur: It’s possible to build state-of-the-art provers without search
- 6B and 62B models finetuned from Minerva on Isabelle proofs



Premise Selection

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

- Premise selection: A key challenge in theorem proving

Premise Selection

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

- Premise selection: A key challenge in theorem proving
- Studied as a separate task w/o theorem proving

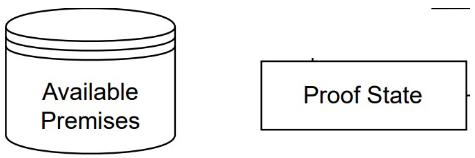
[Irving et al., DeepMath, NeurIPS 2016]
[Wang et al., "Premise Selection for Theorem Proving by Deep Graph Embedding", NeurIPS 2017]

- Recent work integrate premise selection into theorem proving

[Mikuła et al., "Magnushammer: A Transformer-based Approach to Premise Selection", 2023]
[Yang et al., "LeanDojo: Theorem Proving with Retrieval-Augmented Language Models", NeurIPS 2023]

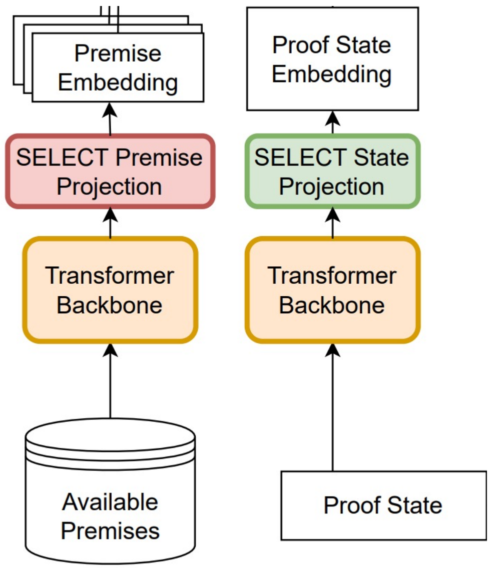
Magnushammer

- Premises selected by Transformer + a simple symbolic prover



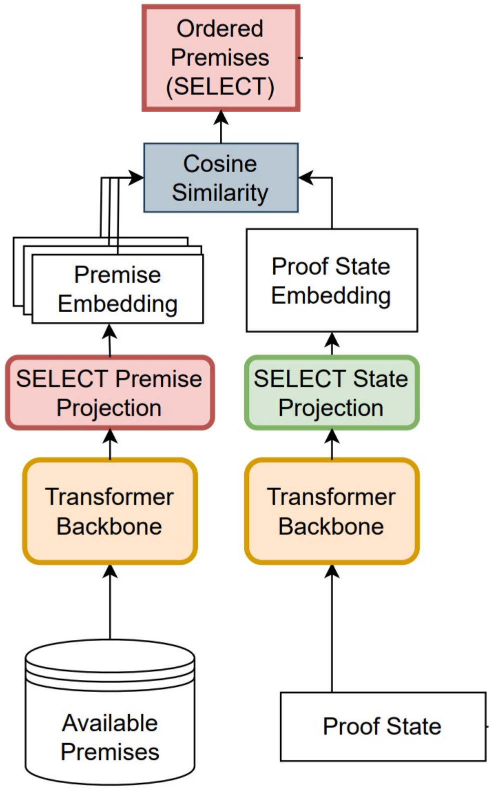
Magnushammer

- Premises selected by Transformer + a simple symbolic prover



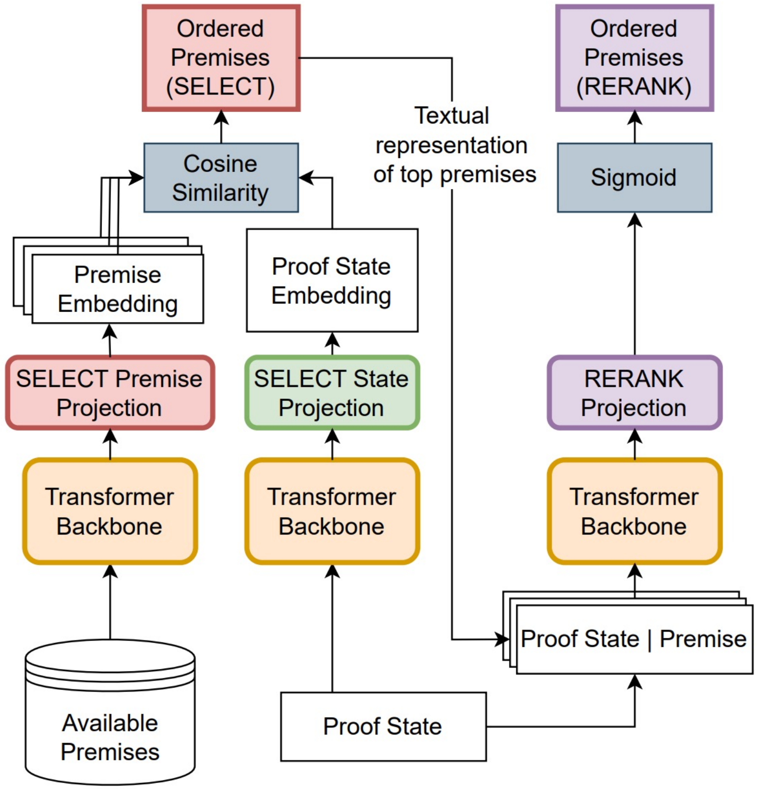
Magnushammer

- Premises selected by Transformer + a simple symbolic prover



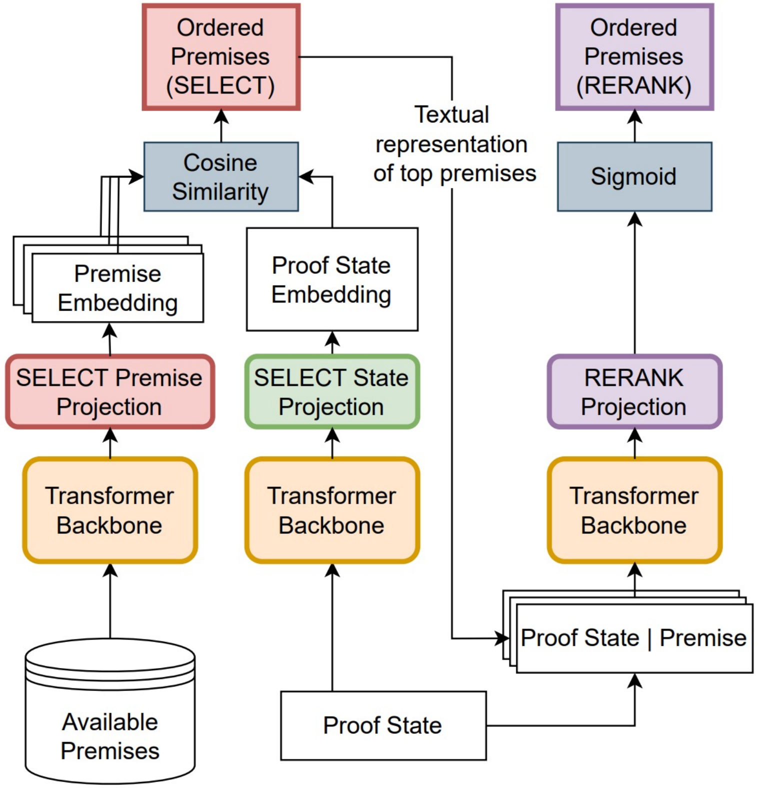
Magnushammer

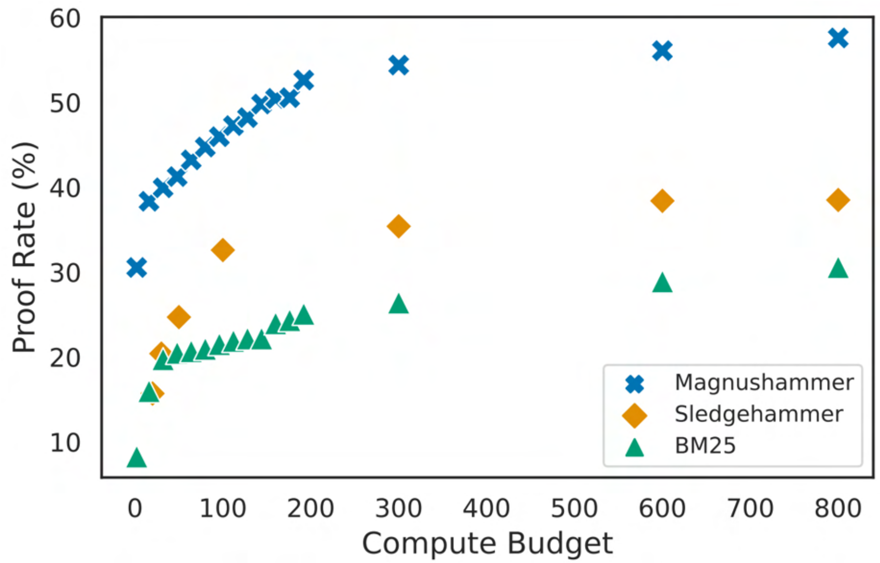
- Premises selected by Transformer + a simple symbolic prover



Magnushammer

- Premises selected by Transformer + a simple symbolic prover





[Yang et al., "LeanDojo: Theorem Proving with Retrieval-Augmented Language Models", NeurIPS 2023]

ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises

State

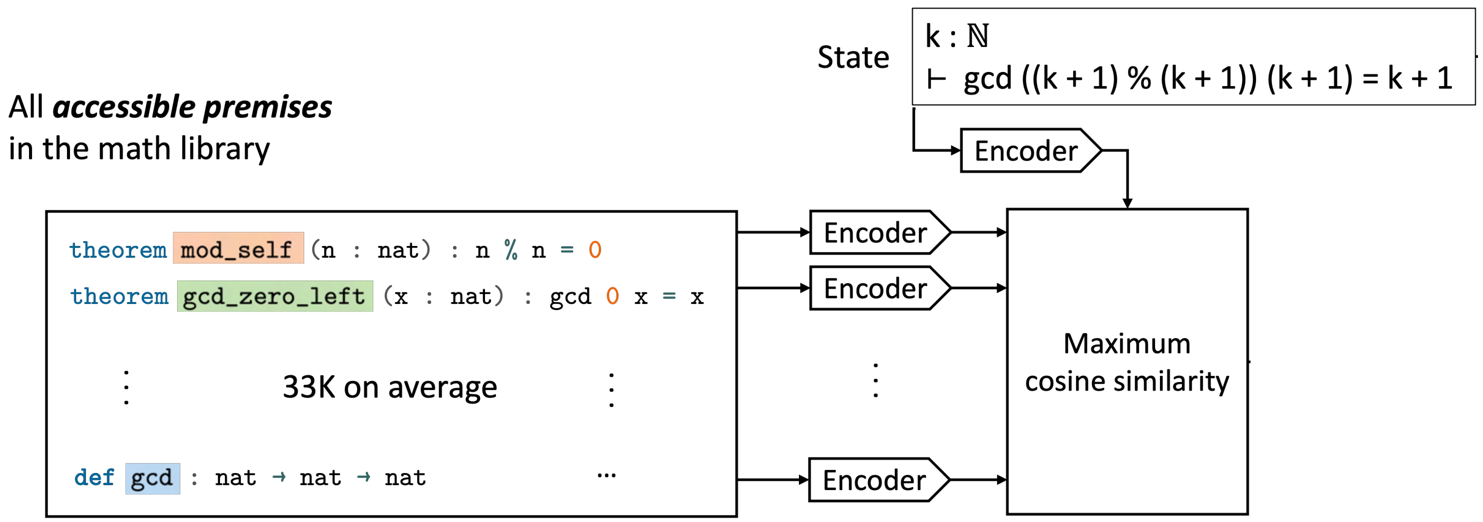
$k : \mathbb{N}$
$\vdash \text{gcd}\ ((k + 1)\ \%\ (k + 1))\ (k + 1) = k + 1$

All ***accessible premises***
in the math library

```
theorem mod_self (n : nat) : n % n = 0
theorem gcd_zero_left (x : nat) : gcd 0 x = x

  ⋮            33K on average            ⋮

def gcd : nat → nat → nat                   …
```

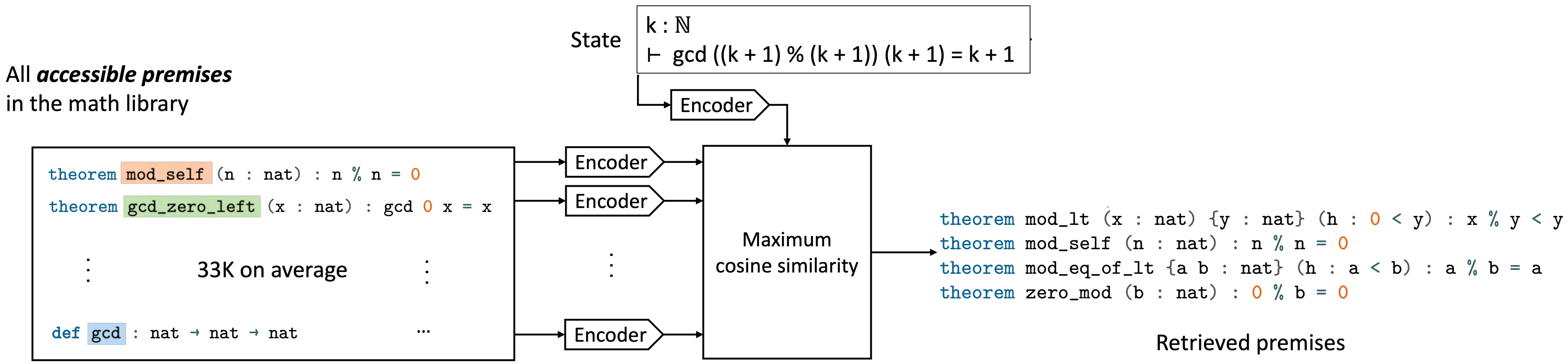
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises



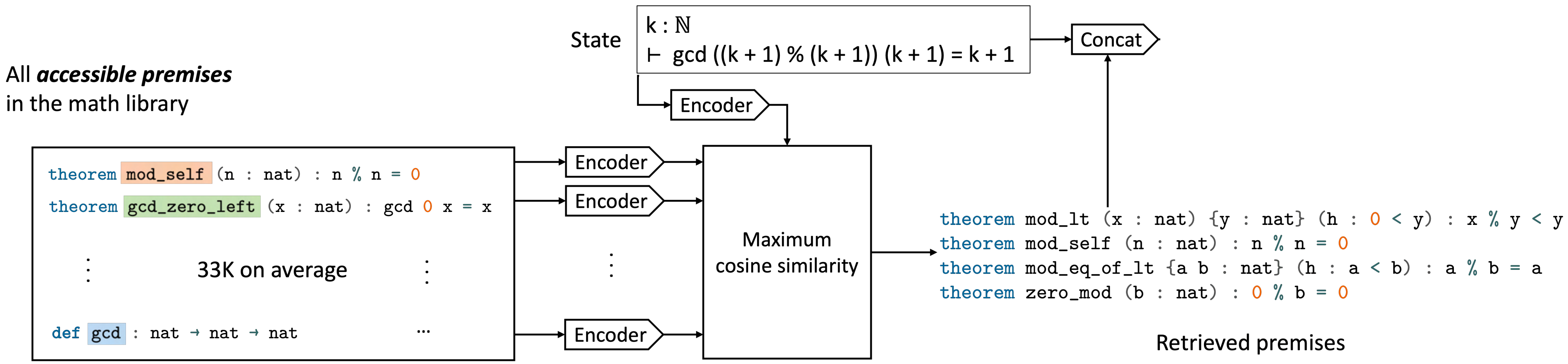
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises



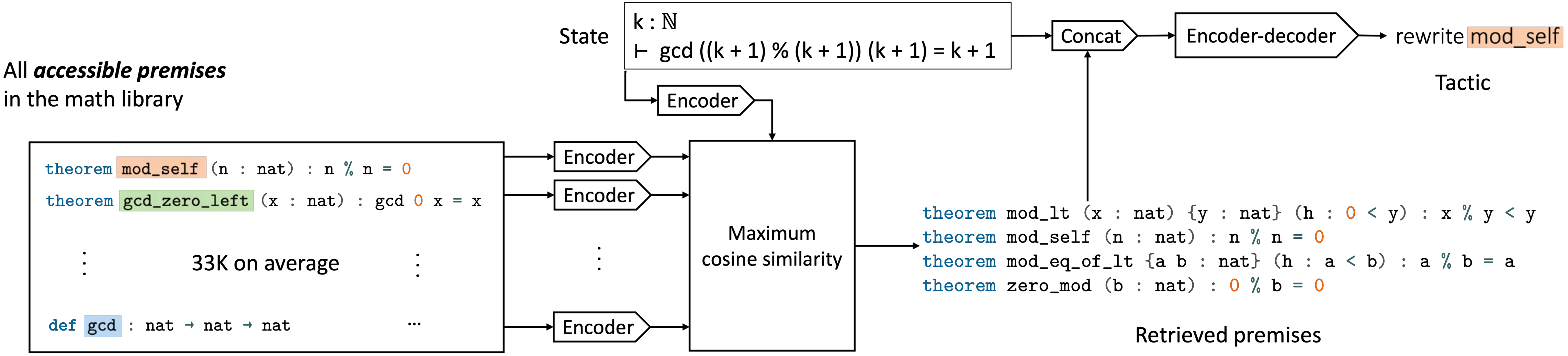
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises
- Retrieved premises are concatenated with the state and used for **tactic generation**

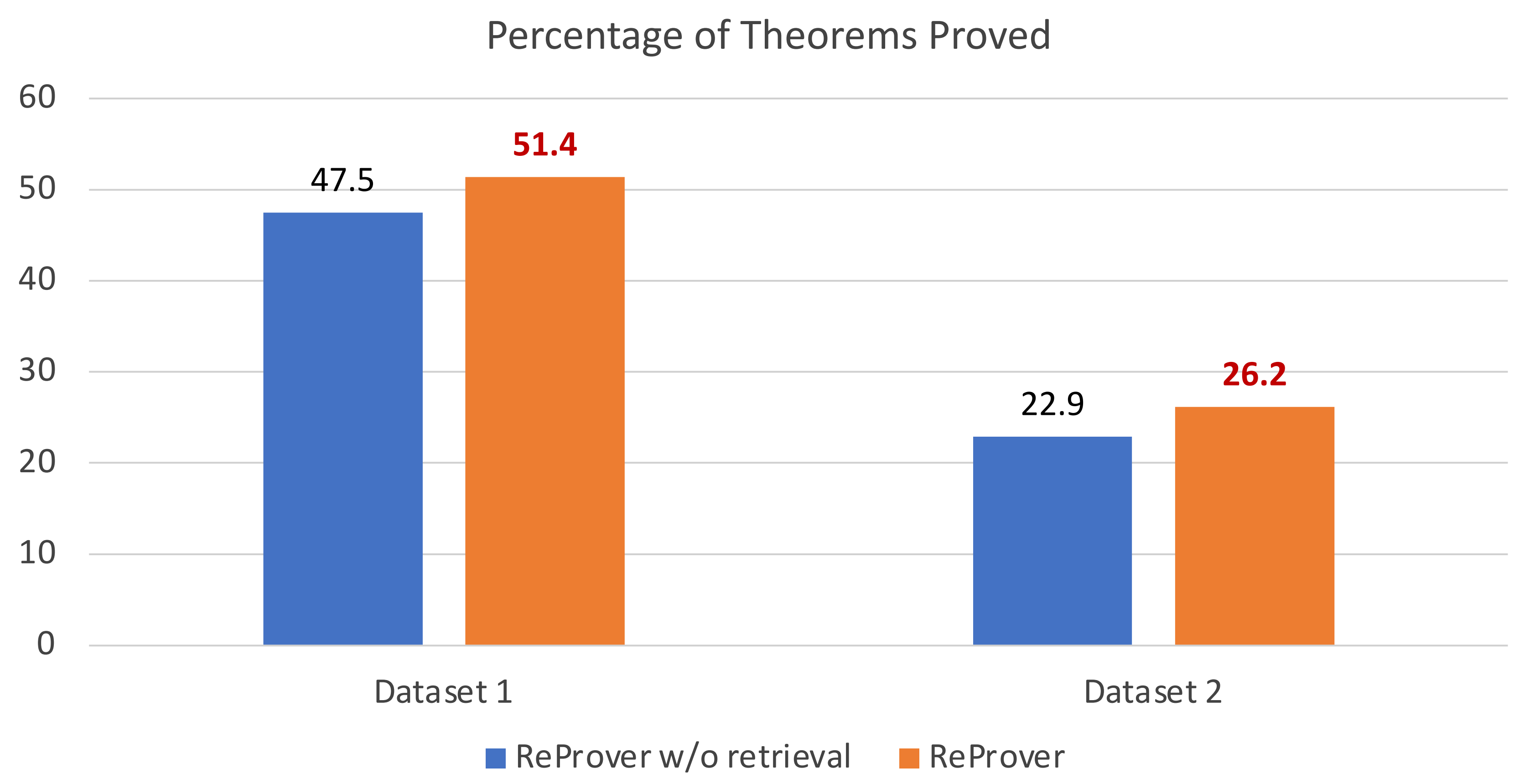


ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises
- Retrieved premises are concatenated with the state and used for **tactic generation**



Premise Retrieval Improves Theorem Proving

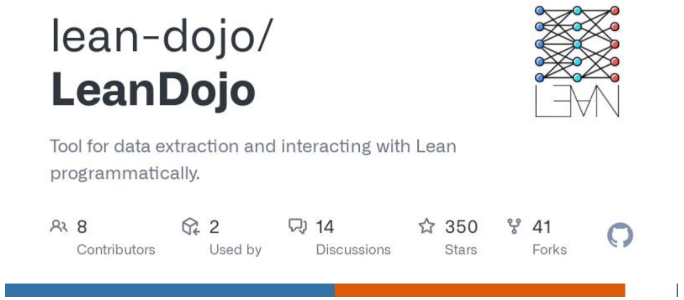


Recap

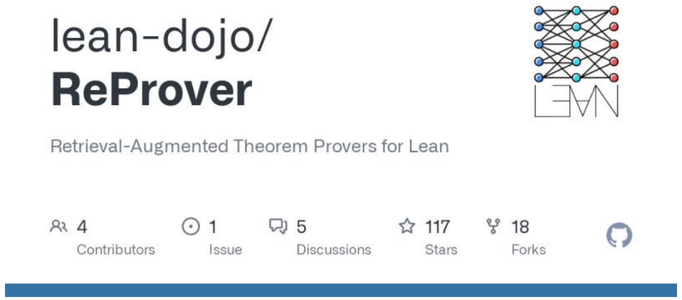
- Theorem proving can help LLMs understand mathematics and generate verifiable code
- LLMs for theorem proving
  - Tactic generator: state -> tactics
  - Proof search: tactics -> proof

Slides, demos, etc. will be available at:
machine-learning-for-theorem-proving.github.io

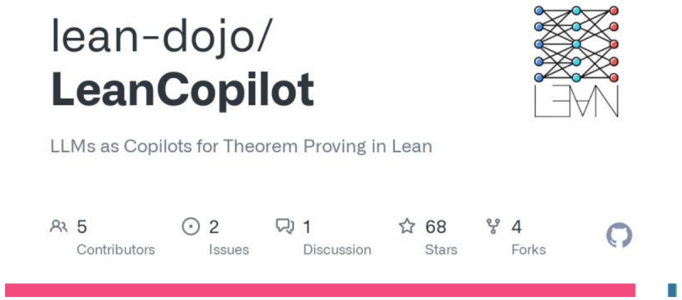
Open-Source Tools



Extract data & interact with Lean



Training & evaluation



Use LLMs in proof assistants

- Isabelle: PISA
- Coq: GamePad, CoqGym, Proverbot9001
- Lean: LLMStep, lean-gym
- Others: HOList, INT

Related Events @ NeurIPS 2023

LeanDojo

- Oral: 10 AM Tuesday
- Poster: 10:45 AM Tuesday

MATH-AI Workshop

- Friday, Room 217-219
- Posters of Lean Copilot and other interesting works!

Outline

Presentation (2 hours)

- **Part I: Fundamentals**
  - What is theorem proving? Why is it important for AI?
  - Demo: a simple LLM-based prover
- **Part II: Advanced topics**
  - Recent work and open problems
  - Machine learning, mathematics, and natural language
  - Machine learning for software verification

Panel (30 minutes)

LLMs, mathematical reasoning, code generation, verification, AI4Science, and more!

Outline

Presentation (2 hours)

- **Part I: Fundamentals**
  - What is theorem proving? Why is it important for AI?
  - Demo: a simple LLM-based prover
- **Part II: Advanced topics**
  - Recent work and open problems
  - Machine learning, mathematics, and natural language
  - Machine learning for software verification

Panel (30 minutes)

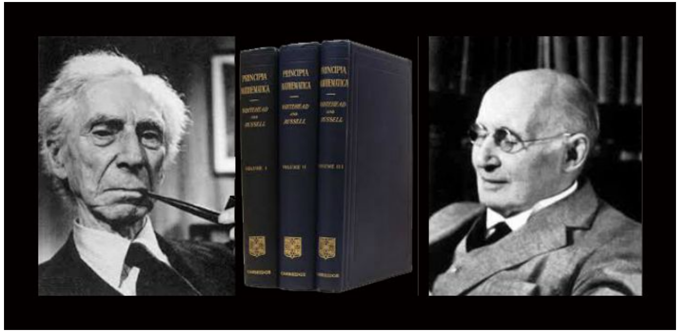
LLMs, mathematical reasoning, code generation, verification, AI4Science, and more!

Guiding Formal Maths with Informal Maths

Albert Q. Jiang, University of Cambridge

What is formal mathematics

Principia Mathematica
Russell and Whitehead



Simple theorems
about **simple** objects:

1 + 1 = 2

Kepler Conjecture
Hales

Complex theorems
about **simple** objects:

Optimal packing of
spheres

Liquid Tensor Experiment
Scholze and Commelin

Complex theorems about
complex objects:

Theorem about condensed
real vector spaces

1910 **2015** **2022**

Formal mathematics ***in real time***

Formalised in 3 weeks!

On a conjecture of Marton

W. T. Gowers, Ben Green, Freddie Manners, Terence Tao

Polynomial Freiman-Ruzsa conjecture




2023

What is a proof assistant?

- Has some logical/type-theory basis, with axioms, rules and theorems
- Proving a theorem:
  - Iteratively applying rules of the formal system to transform the goal
  - Until it becomes trivial

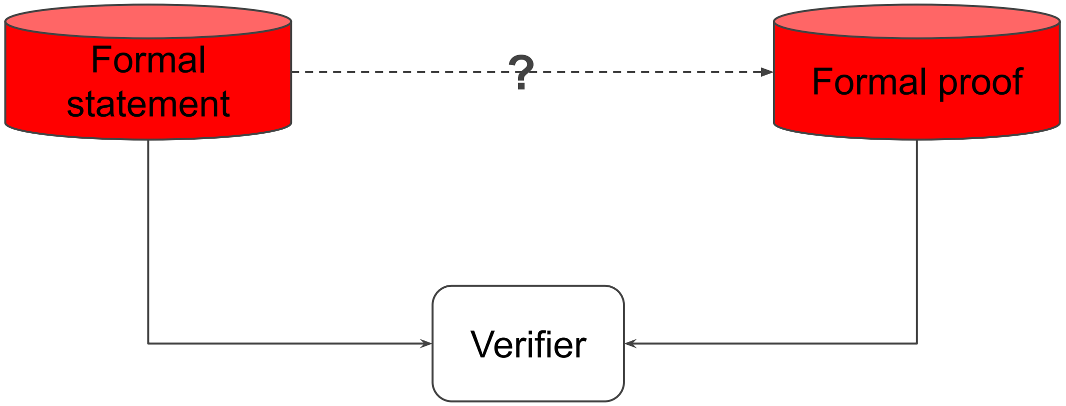
Example of a proof in Lean

```
example (m n k : ℕ) (h₀ : n ≤ m) : n + k ≤ m + k := begin
  induction k,
  {
    exact h₀
  },
  {
   rw nat.succ_le_succ_iff,
   exact k_ih
  }
end
```

First subgoal : $\mathrm{n} + 0 \leq m + 0$

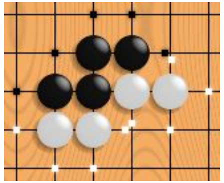
Second subgoal :
$\mathrm{n} + k \leq m + k \Rightarrow n + k + 1 \leq m + k + 1$

The goal of automated formal theorem proving

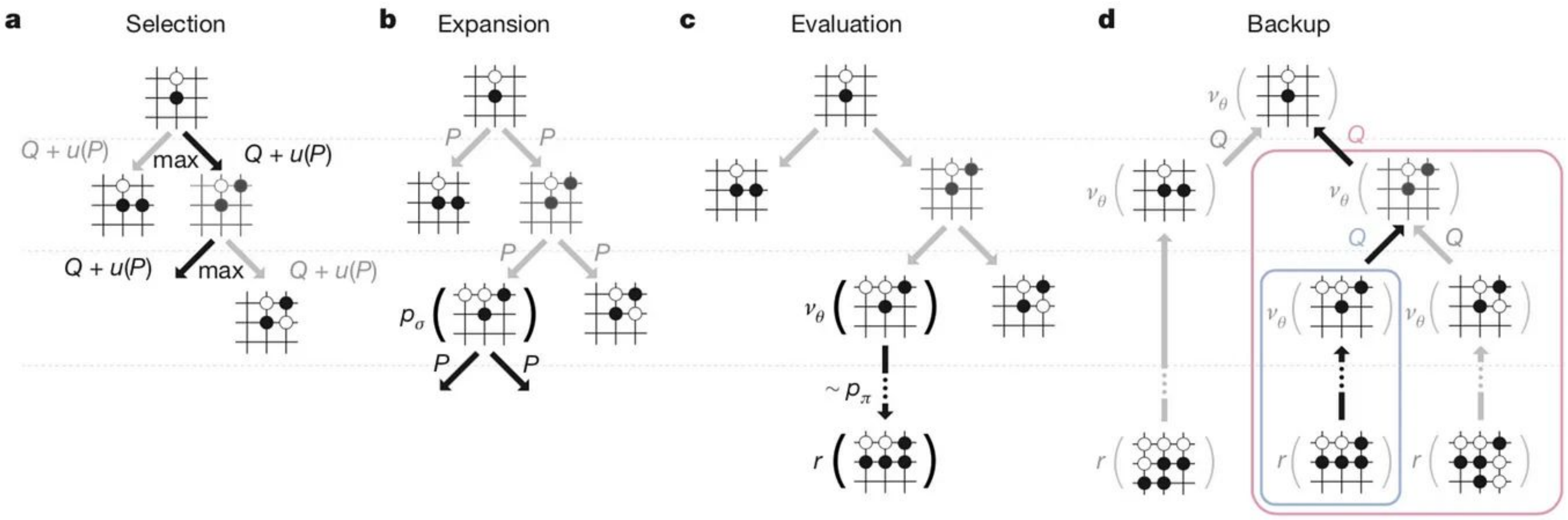


How can machine learning come in?

Components of a Markov Decision Process

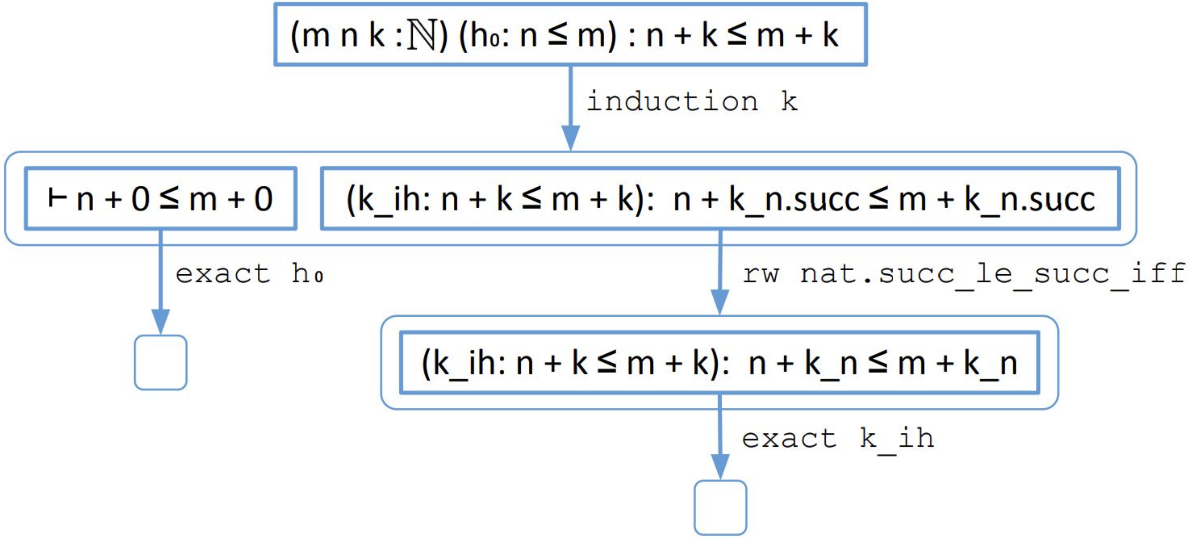
	state	action	reward	transition
Go	A board position 	Place a stone 	1, 0, or -1 at end of the game	
Theorem Proving	Goal (s) to prove `⊢ 1+1 = 2`	Use a tactic `Eq.refl _`	1 for QED, 0 for failure	

How can machine learning come in? (cont.)



Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." nature 529.7587 (2016): 484-489.

How can machine learning come in? (cont.)



Lample, Guillaume, et al. "Hypertree proof search for neural theorem proving." Advances in Neural Information Processing Systems 35 (2022): 26337-26349.

But there’s an important aspect of mathematics

Mathematics is mostly written in natural language and not utilised by machine learning at this point!

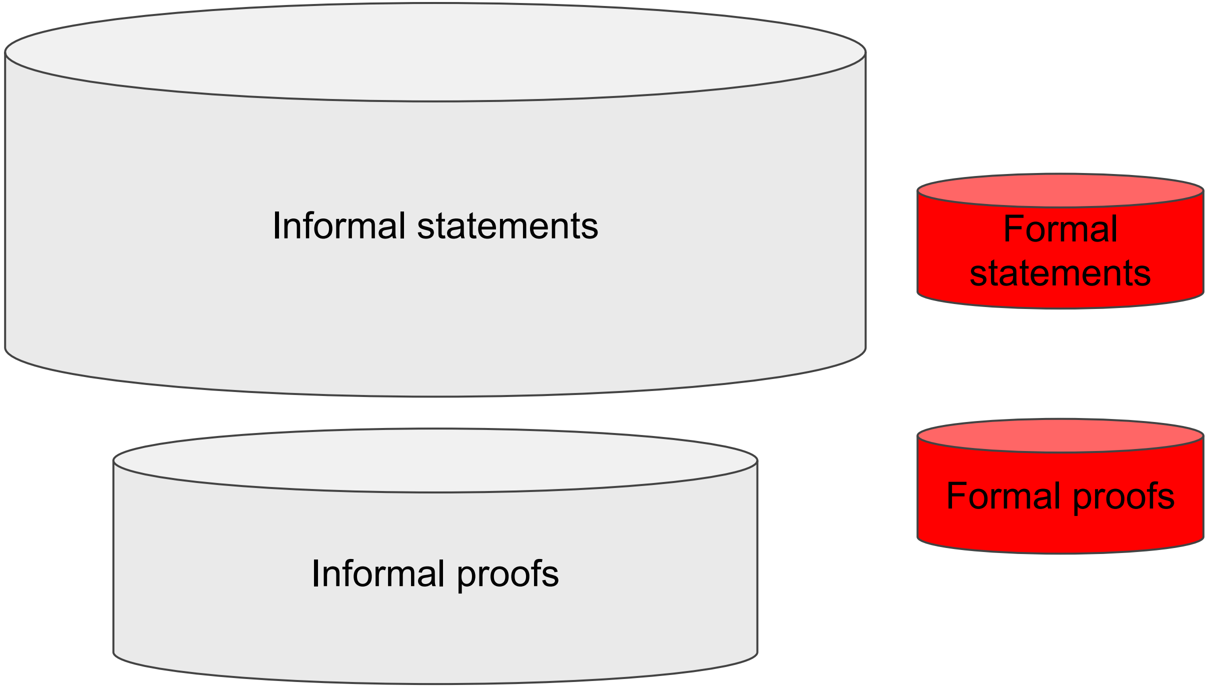
Annals of Mathematics, **142** (1995), 443–551

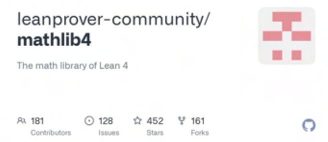
Modular elliptic curves and Fermat’s Last Theorem

By Andrew Wiles*

Not yet fully formalised on a computer!

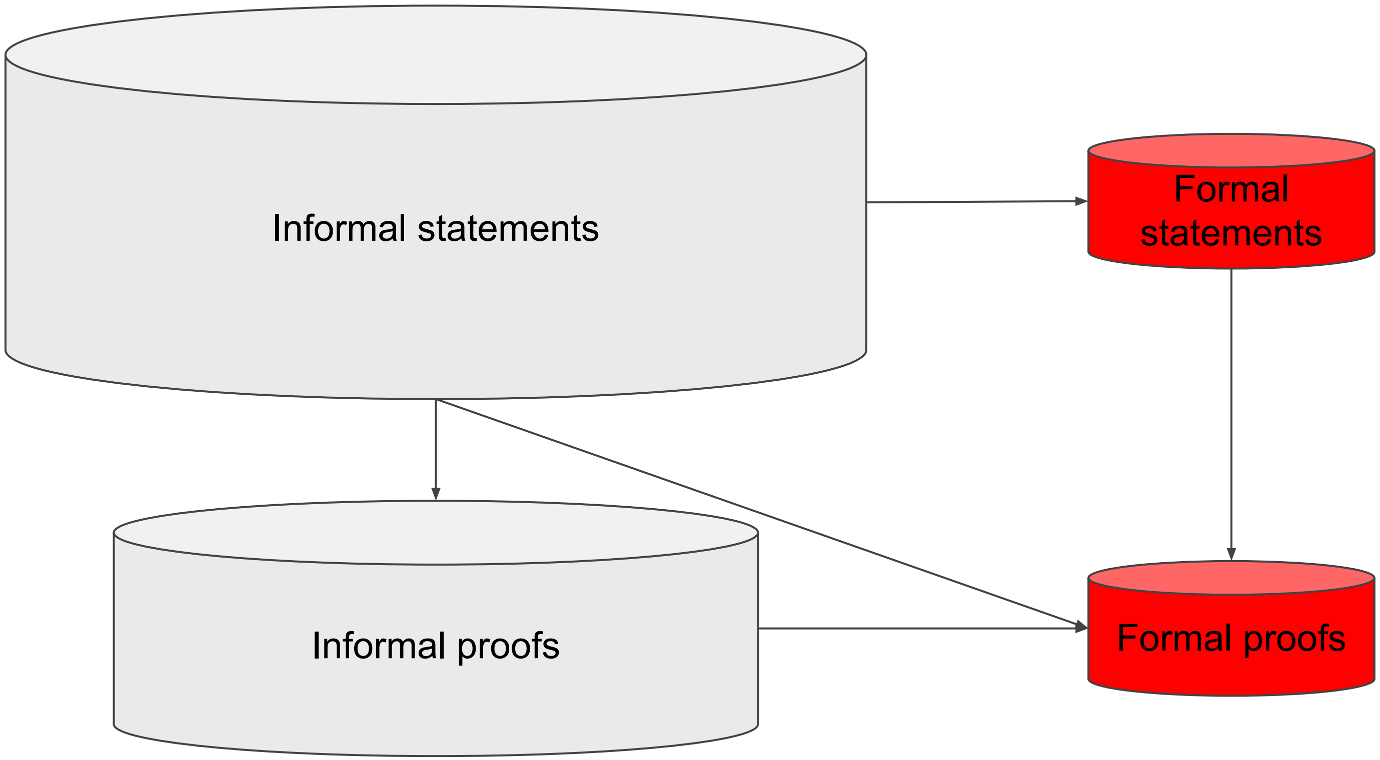
What we have


Informal statements
Informal proofs
Formal statements
Formal proofs

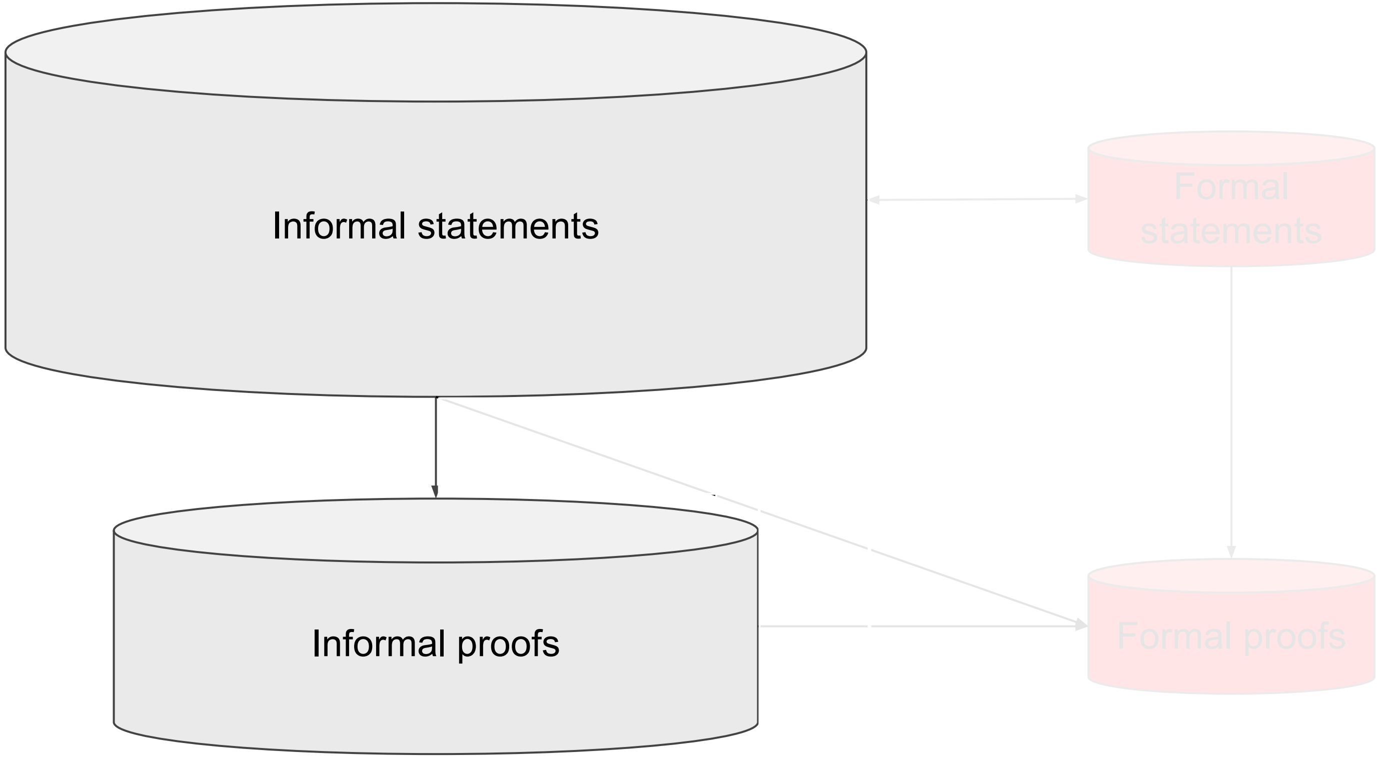

leanprover-community/
mathlib4
The math library of Lean 4
181 Contributors
128 Issues
452 Stars
161 Forks


jrh13/hol-light
The HOL Light theorem prover
12 Contributors
44 Issues
352 Stars
73 Forks

The goal of this presentation



Making better informal reasoners

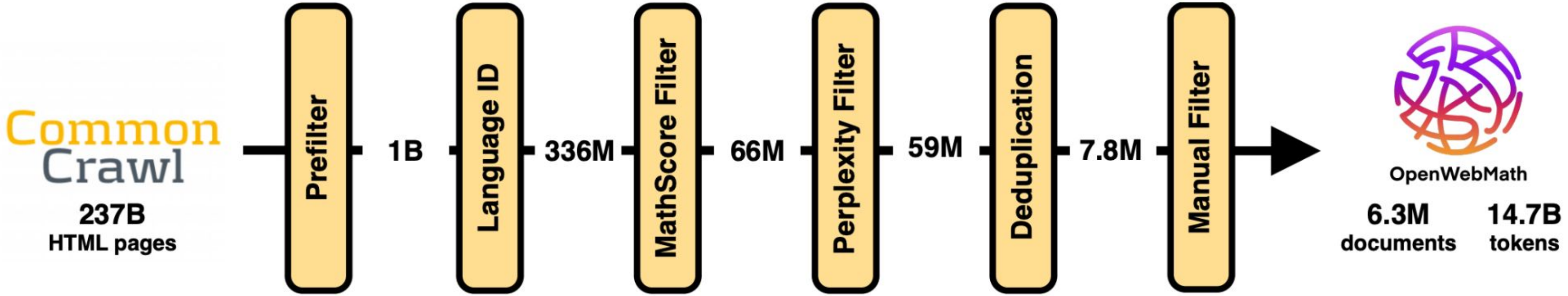


Make better informal reasoners

1. Find high-quality mathematical content online

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<script type="text/javascript" src="https://cdn.mathjax.org/mathjax/latest/MathJax.js?
config=TeX-AMS_HTML"></script>
</head>
<body>

<p>
 This is a paragraph with inline math.
 \( f \left( x \right) = 3x^2 + 3x + 3 \)
 You should see a quadratic function before this sentence.
</p>

</body>
</html>
```

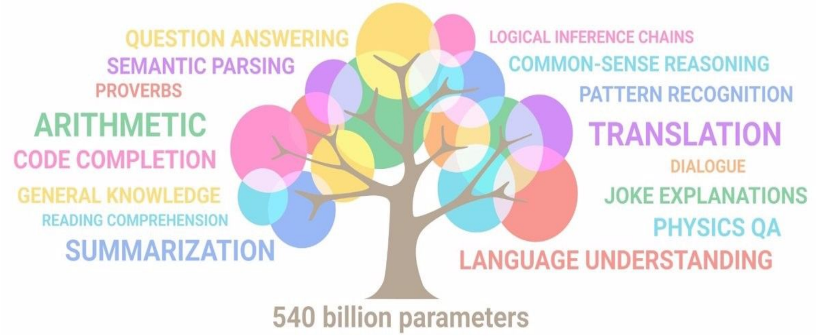
Data collection: OpenWebMath



Paster, Keiran, et al. "OpenWebMath: An Open Dataset of High-Quality Mathematical Web Text." *arXiv preprint arXiv:2310.06786* (2023).

Make better informal reasoners (cont.)

2. Fine-tune or continue pretraining a strong base language model on it

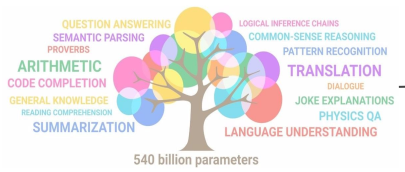


PaLM

CodeLlama

Lewkowycz, Aitor, et al. "Solving quantitative reasoning problems with language models." *Advances in Neural Information Processing Systems* 35 (2022): 3843-3857.
Azerbayev, Zhangir, et al. "Llemma: An open language model for mathematics." *arXiv preprint arXiv:2310.10631* (2023).

Specialising models on informal maths



PaLM
(closed)

Google's secret sauce mathematical dataset 🪄 (closed)

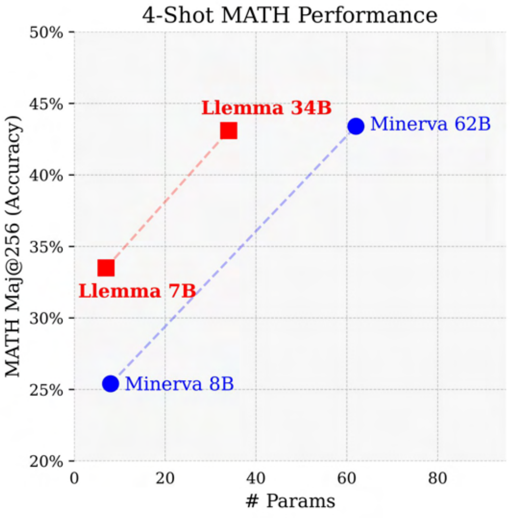
Minerva
(closed)

CodeLlama
(open)

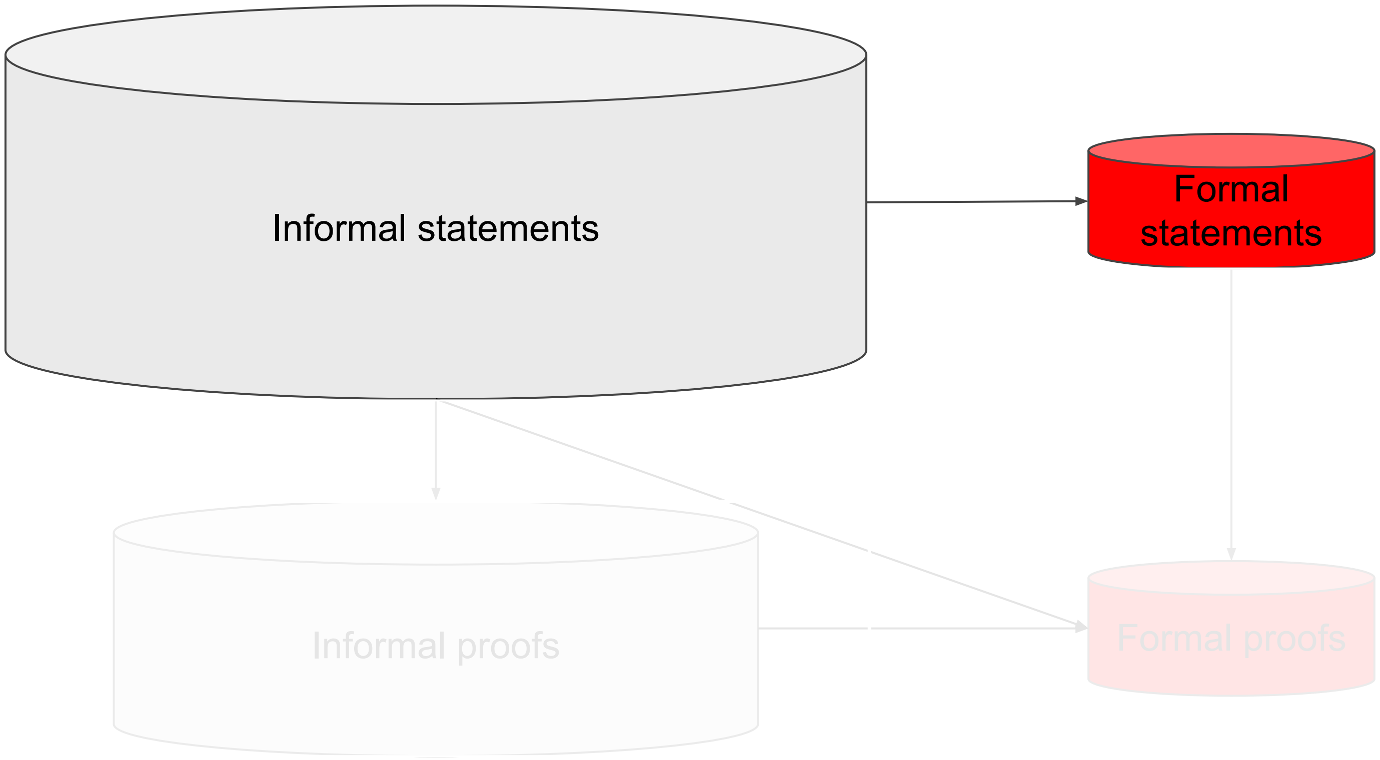
ProofPile 2 = OpenWebMath + Code + Arxiv (open)

LLEMMA

(open)

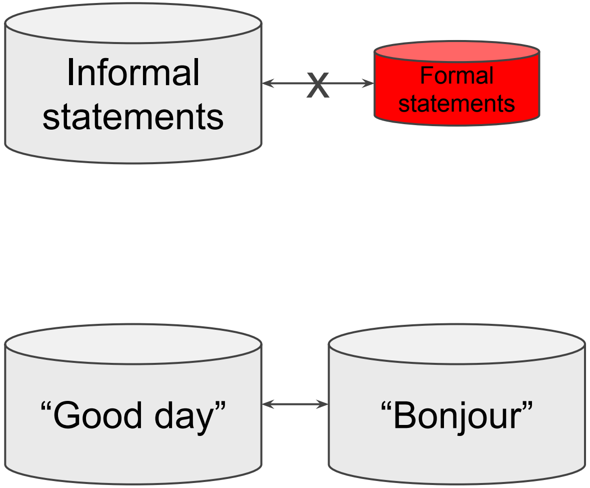


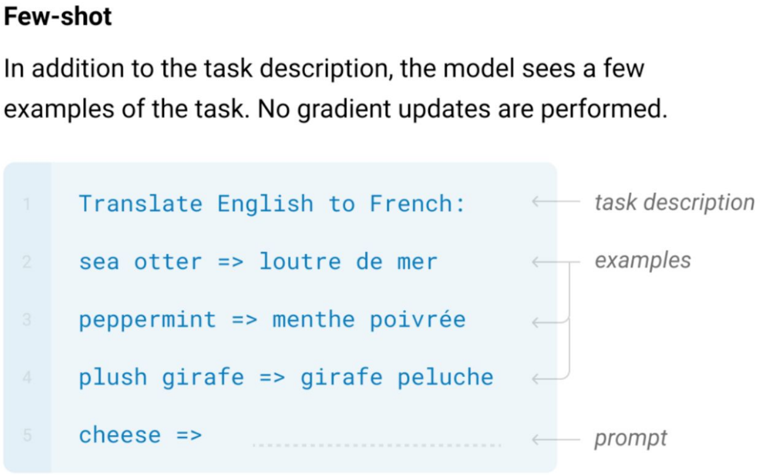
Turning informal data into formal ones



Turning informal data into formal data

Autoformalization with large language models





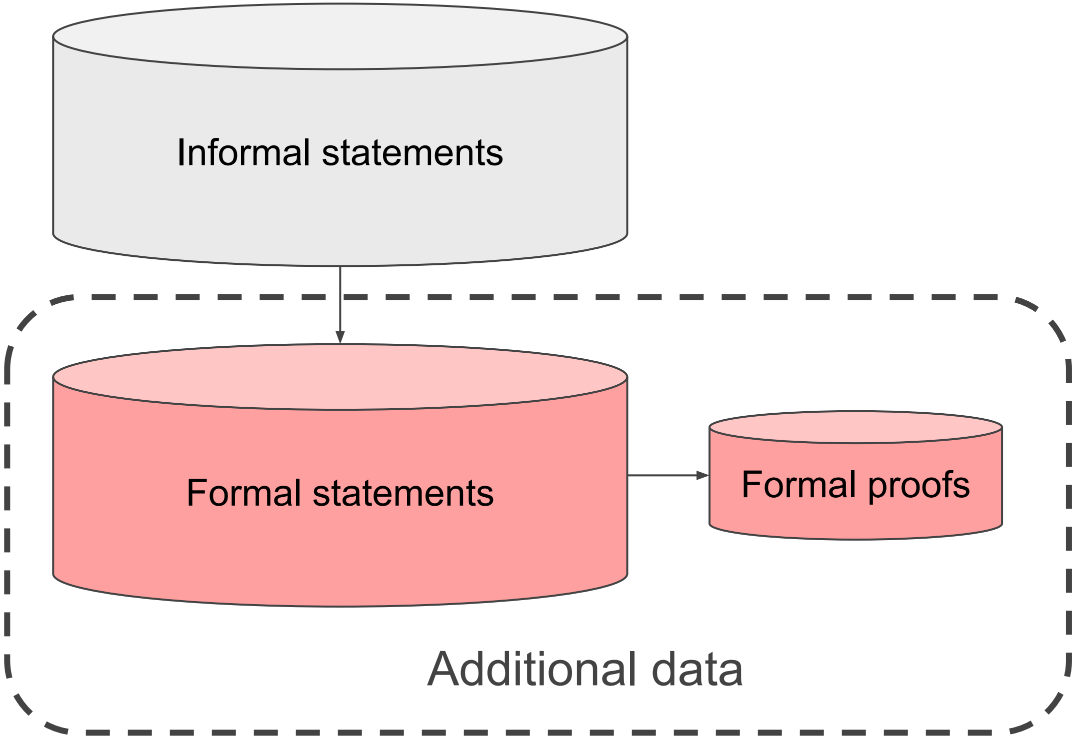
Wu, Yuhuai, et al. "Autoformalization with large language models." *Advances in Neural Information Processing Systems* 35 (2022): 32353-32368.

Does this work?

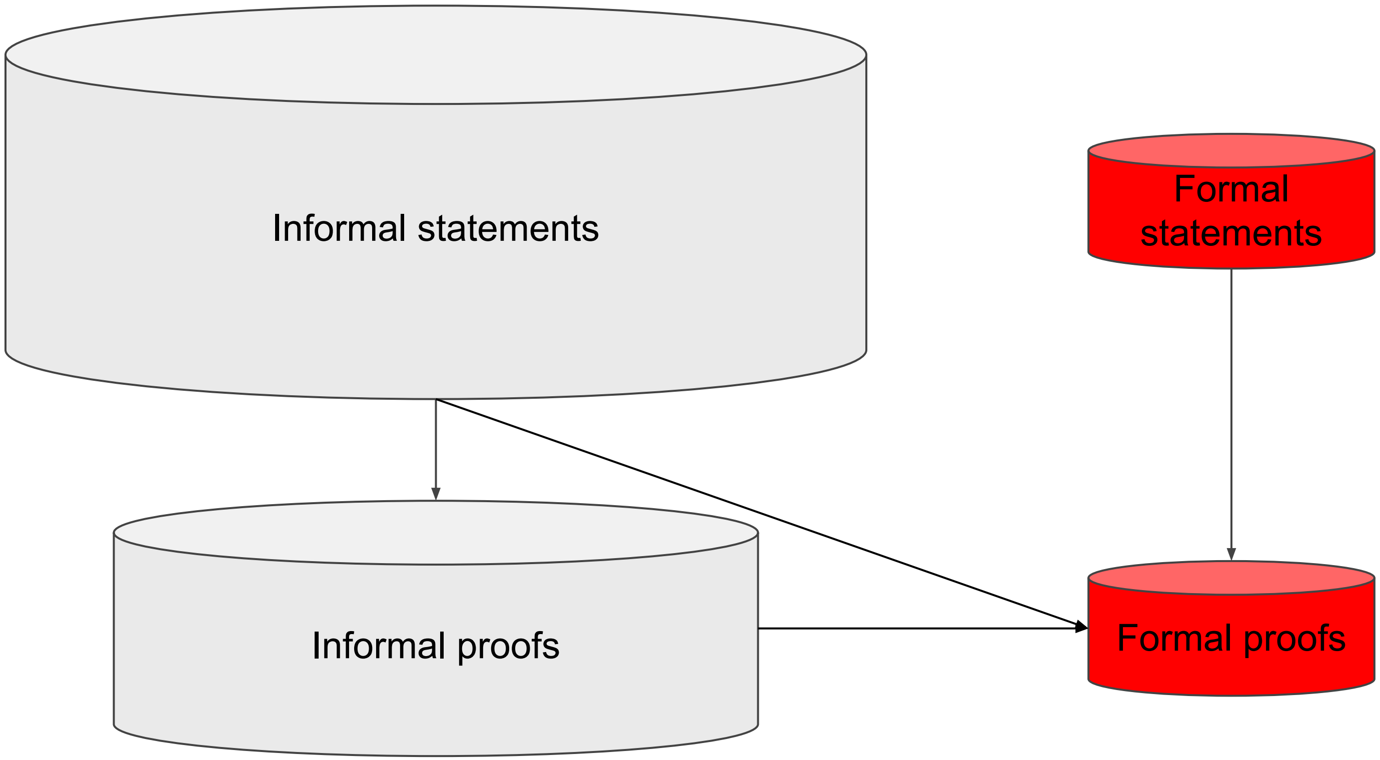
Yes, to an extent.

Manually examined 150 informal → formal statement translations. Correctness rate is 25%.

Drawback: we don't automatically know which translations are right.



Put everything together



Alignment challenge: different levels of reasoning

Informal Statement: Show that for any real number a, $10a \leq 28a^2 + 1$.

Informal Proof:
It suffices to show $0 \leq 28a^2 - 10a + 1$. First, consider completing the square for $28a^2 - 10a$ and observe that $\left(a - \frac{5}{28}\right)^2 = a^2 - \frac{10}{28}a + (5/28)^2$. Since $0 \leq \left(a - \frac{5}{28}\right)^2$, we get $0 \leq a^2 - \frac{10}{28}a + (5/28)^2$. Multiplying by 28 and simplifying gives $0 \leq 28a^2 - 10a + (25/28)$. Since $25/28 < 1$, the result follows.

Formal Statement:

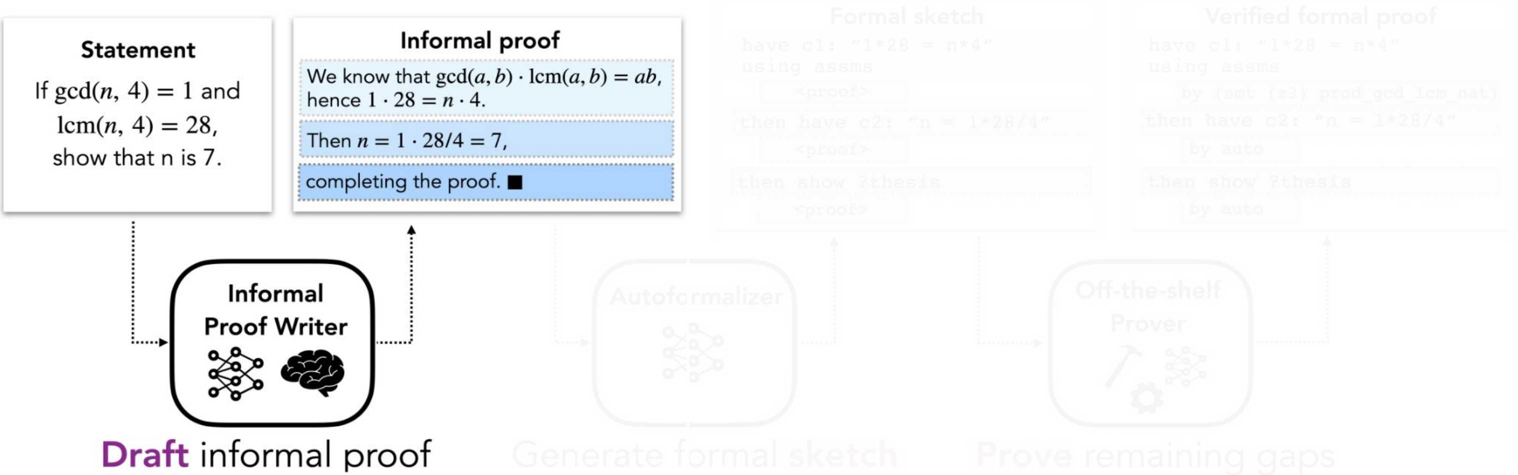
```
theorem algebra_binomnegdiscrineq_10alt28asqp1:
  fixes a :: real
  shows "10 * a ≤ 28 * a^2 + 1"
```

Formal Proof Sketch:

```
proof - (* it suffices to show 0 <= 28a^2 - 10a + 1 *)
  have c0: "0 ≤ 28a^2 - 10a + 1"
  proof - (* observe that (a - (5/28))^2 = a^2 - (10/28)a + (5/28)^2 *)
    have c1: "(a - (5/28))^2 = a^2 - 10/28a + (5/28)^2" <···>
    (* we get 0 <= a^2 - (10/28)a + (5/28)^2 *)
    have c2: "0 ≤ a^2 - 10/28a + (5/28)^2" using c1 <···>
    (* Multiplying by 28 and simplifying gives 0 <= 28a^2 - 10a + (25/28) *)
    have c3: "0 ≤ 28a^2 - 10a + 28((5/28)^2)" using c2 <···>
    have c4: "0 ≤ 28a^2 - 10a + 28((5/28)*(5/28))" using c3 <···>
    have c5: "0 ≤ 28a^2 - 10a + (25/28)" using c4 <···>
    (* Since 25/28 < 1, the result follows. *)
    show ?thesis using c5 <···>
  qed
  show ?thesis <···>
qed
```

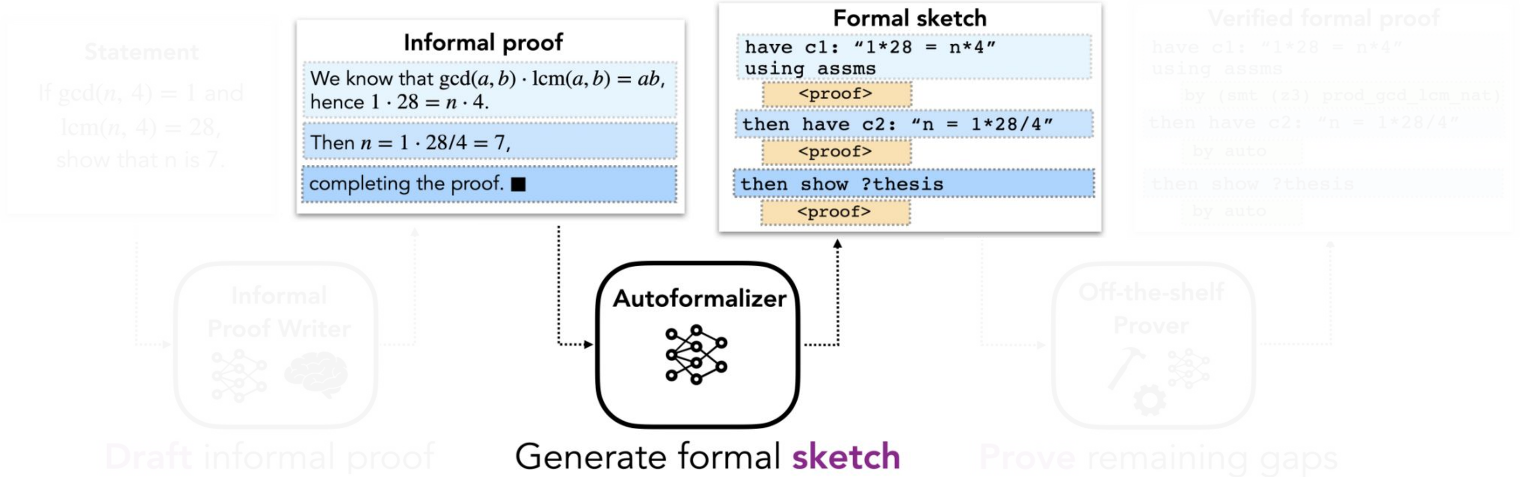
- **Solution: translate into proof sketches**
- Formal proof sketches encapsulate the high-level ideas of the proof.
- They are better aligned with the informal proofs
- We copy segments of the informal proof as in-line comments to create (even) better alignment.

Getting formal proofs: Draft, Sketch, and Prove



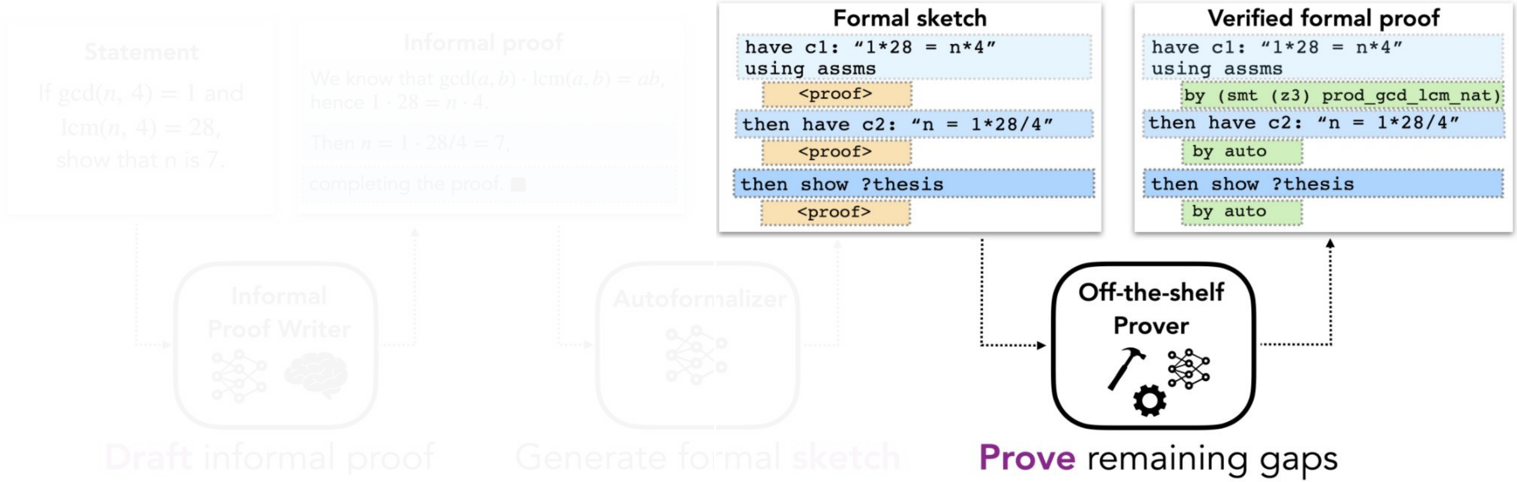
Jiang, Albert Q., et al. "Draft, sketch, and prove: Guiding formal theorem provers with informal proofs." *arXiv preprint arXiv:2210.12283* (2022).

Getting formal proofs: Draft, **Sketch**, and Prove



Jiang, Albert Q., et al. "Draft, sketch, and prove: Guiding formal theorem provers with informal proofs." *arXiv preprint arXiv:2210.12283* (2022).

Getting formal proofs: Draft, Sketch, and **Prove**



Jiang, Albert Q., et al. "Draft, sketch, and prove: Guiding formal theorem provers with informal proofs." *arXiv preprint arXiv:2210.12283* (2022).

Benchmark

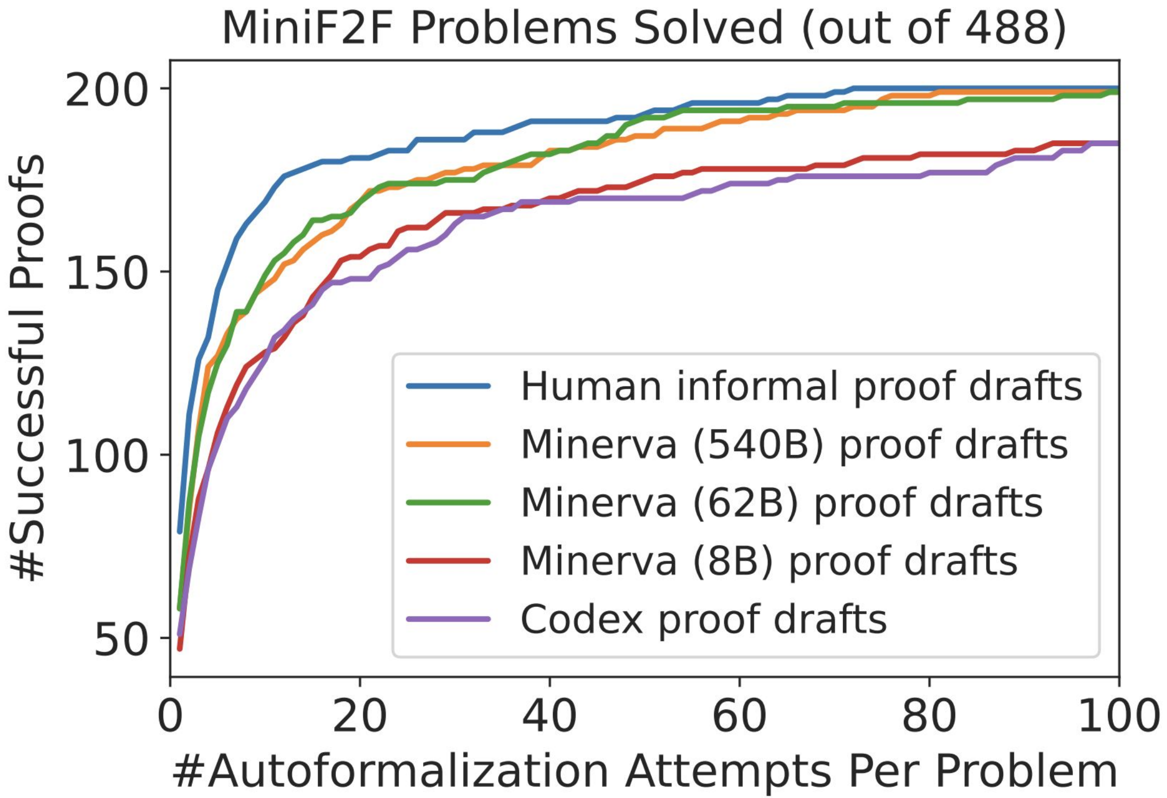
- MiniF2F = Reference benchmark developed by OpenAI

- Formalized problems from olympiads (IMO, AIME, AMC), high-schools and undergraduate math classes

- Valid / Test splits:
  - 488 problems
  - Metamath / Isabelle / Lean / Hol-light

			Test Set	Validation Set
TOTAL			244	244
IMO			20	20
AIME			15	15
AMC			45	45
MATH	Algebra	Level 5	14	14
		Level 4	14	14
		Level 3	14	14
		Level 2	14	14
		Level 1	14	14
	Number Theory	Level 5	16	16
		Level 4	11	11
		Level 3	11	11
		Level 2	11	11
		Level 1	11	11
CUSTOM	Algebra		18	18
	Number Theory		8	8
	Induction		8	8

Results



MiniF2F Problems Solved (out of 488)
#Successful Proofs
200
150
100
50
0
20
40
60
80
100
#Autoformalization Attempts Per Problem
Human informal proof drafts
Minerva (540B) proof drafts
Minerva (62B) proof drafts
Minerva (8B) proof drafts
Codex proof drafts

Takeaways

- Machine learning methods for formal mathematics should not discard informal mathematics
    - That’s where (almost) all the data is!

- LLMs gave us the opportunity to realistically convert informal maths to formal maths
    - But the detailed implementation needs careful thought

Machine Learning for *Formal Software Verification*

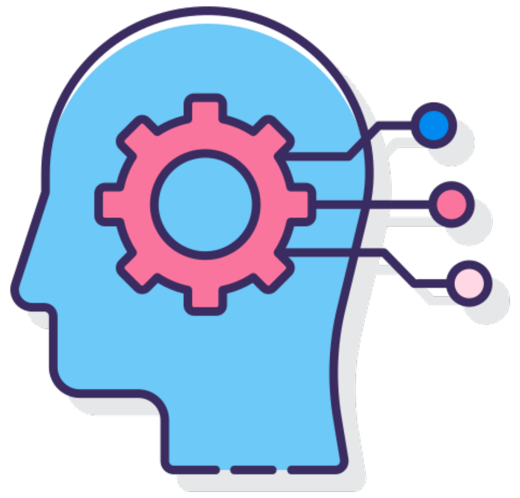
Emily First, Albert Q Jiang, Kaiyu Yang

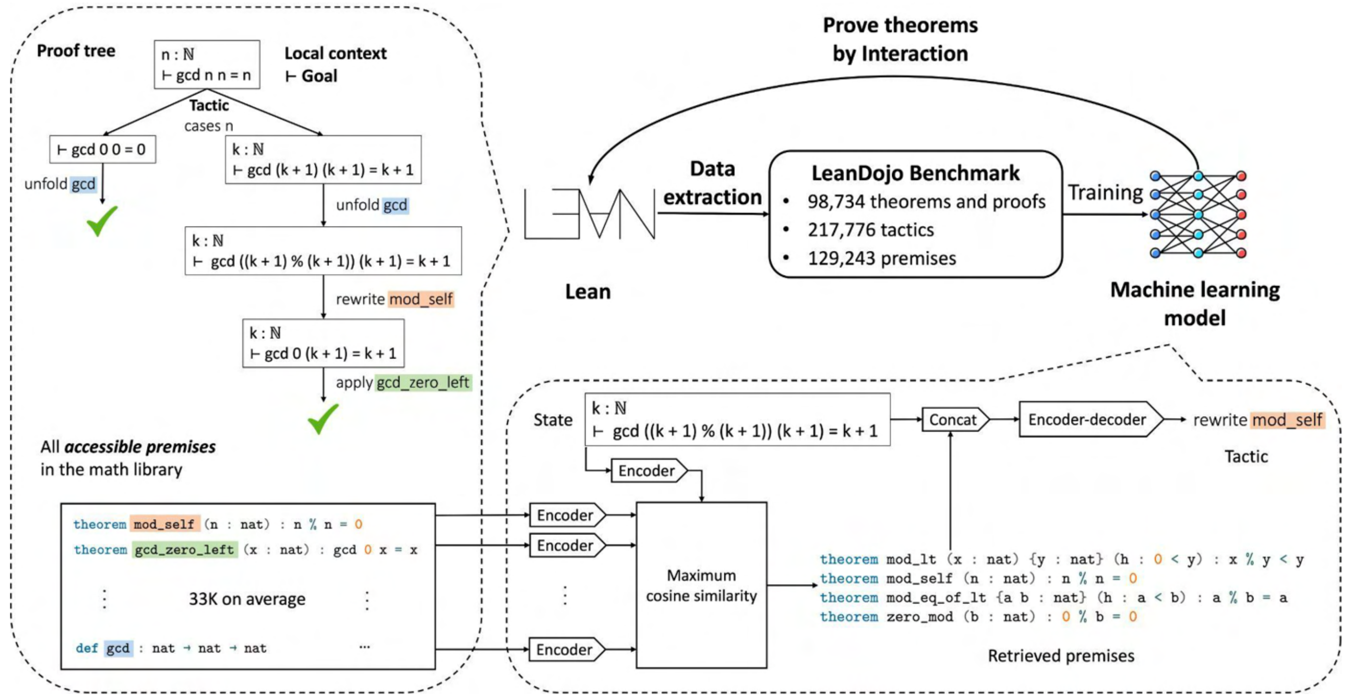
NeurIPS Tutorial on Machine Learning for Theorem Proving
December 11, 2023

Quick Recap

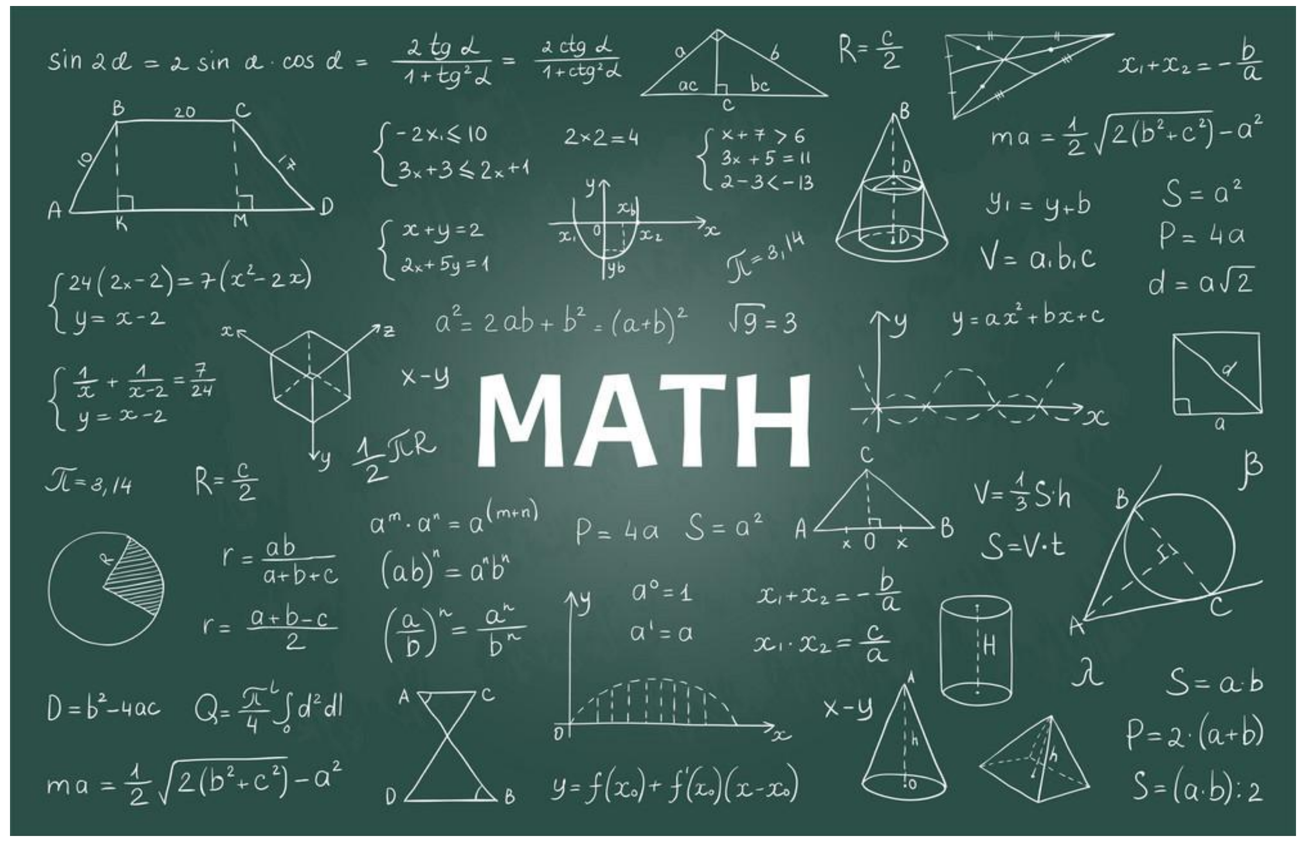
- Proof assistants

- Machine learning methods for theorem proving



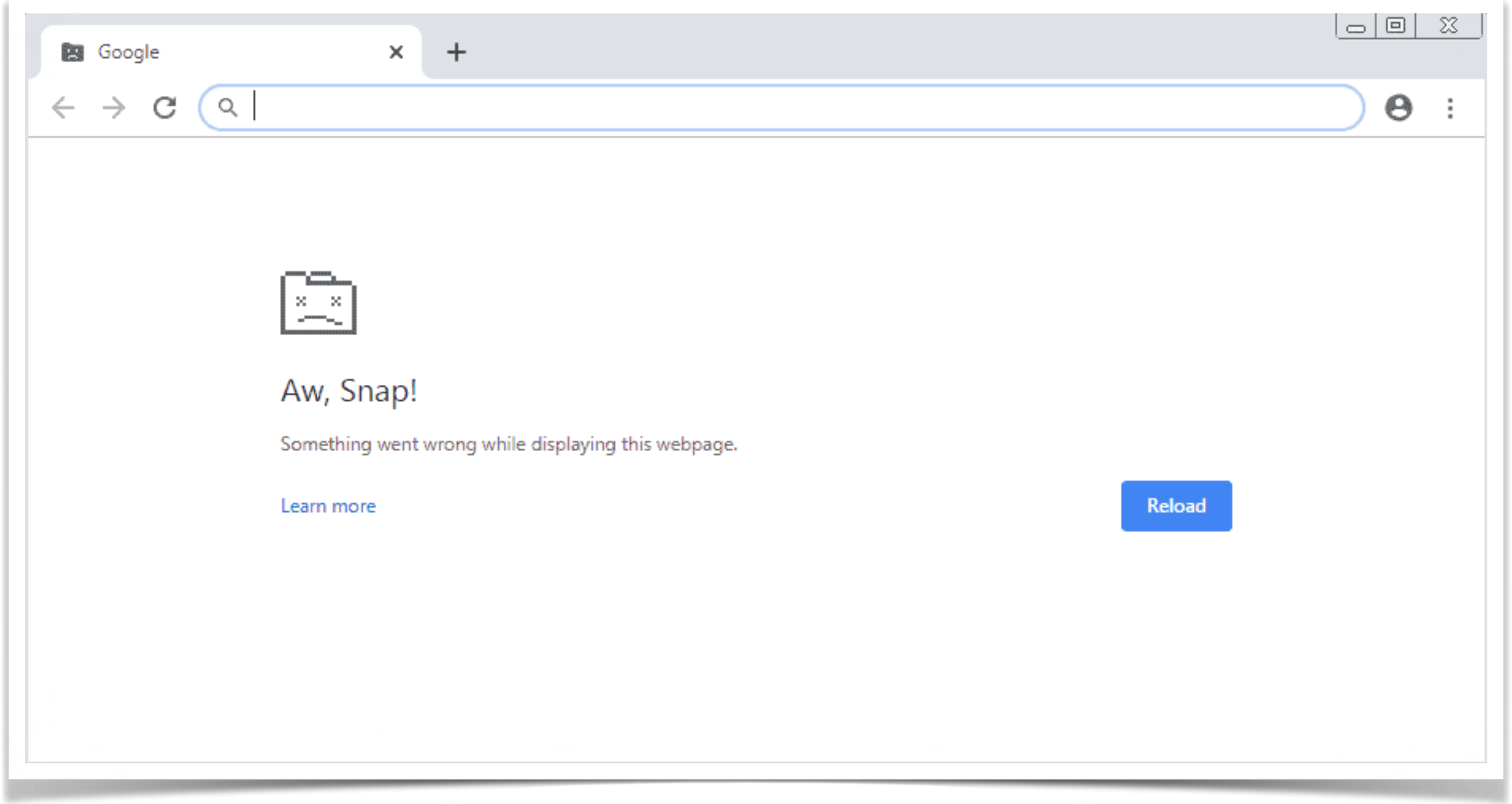


- Formalizing and proving mathematics



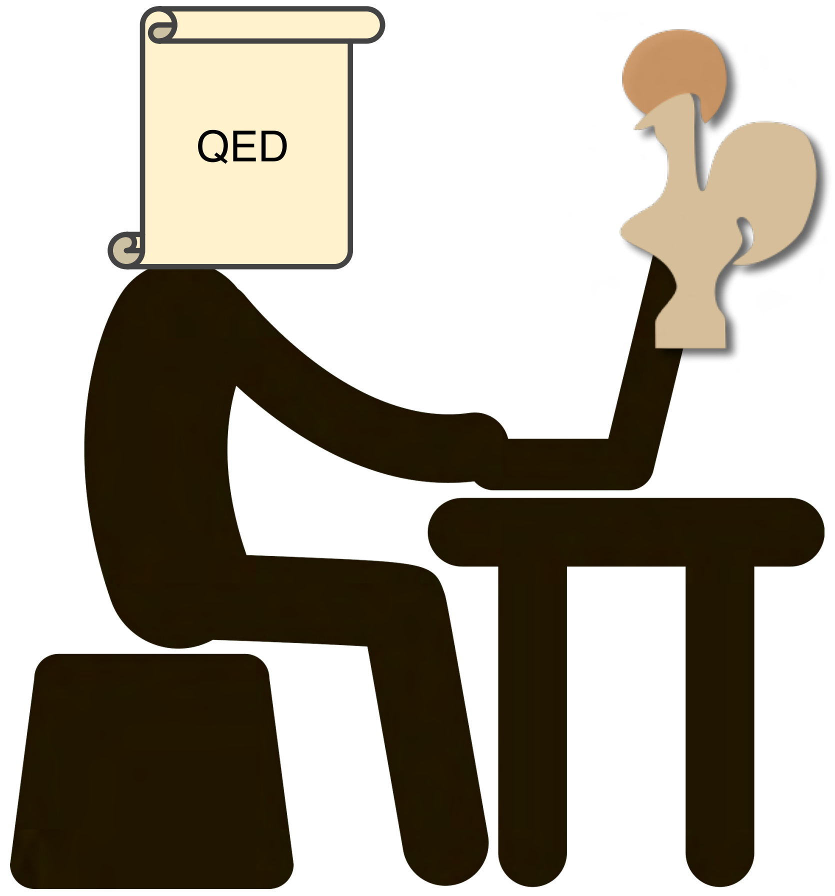
Why should you care about formal software verification?

Software Bugs Matter



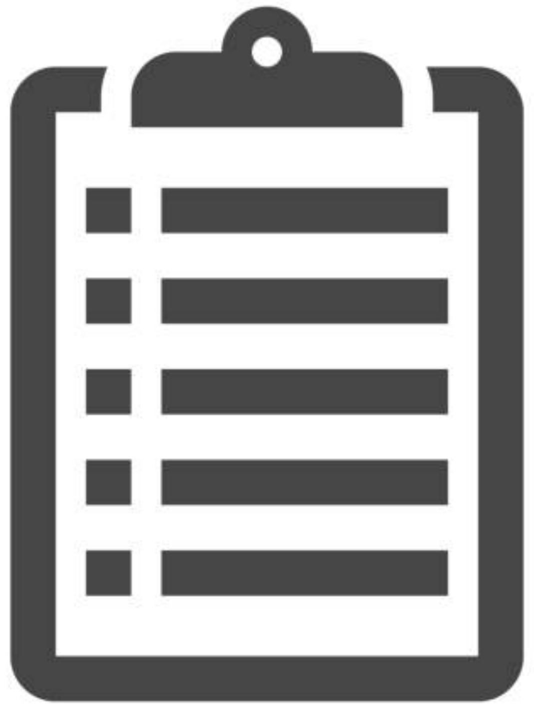
In 2020, CISQ estimated that software failures cost the economy **$1.56 trillion dollars** annually

Formal Software Verification

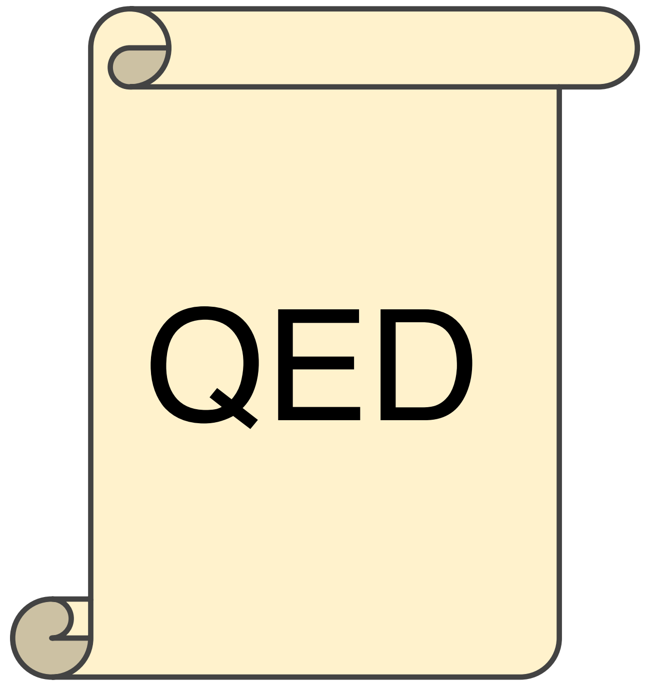


Proof Engineer

Specifications



Program implementation

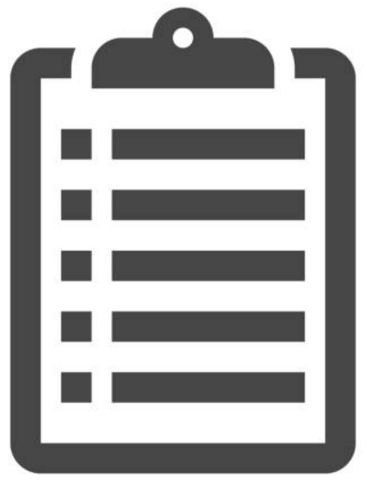


Mathematical proofs

- Think about the desired & actual behavior of the program
- Perhaps finding & fixing bugs in the process
- Make explicit which parts of the system are trusted
- Decrease the burden of trust as more of the system is verified

Ringer et al. (2020) “QED at Large: A Survey of Engineering of Formally Verified Software”

Software Development Life Cycle

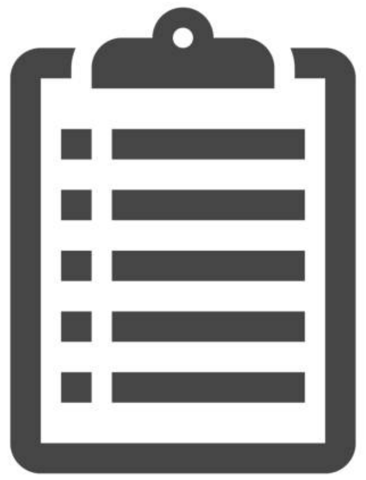


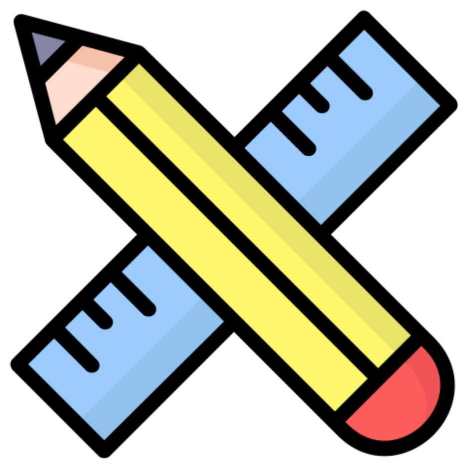
Requirements

The length of a reversed list is the same as the length of the original list

```
Theorem len_rev_unchanged:
forall (A: Type) (l: list A),
length (rev l) = length l.
```

Software Development Life Cycle





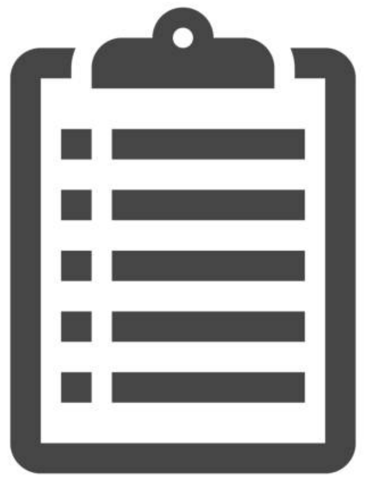
Requirements → Design

Helper Lemmas!

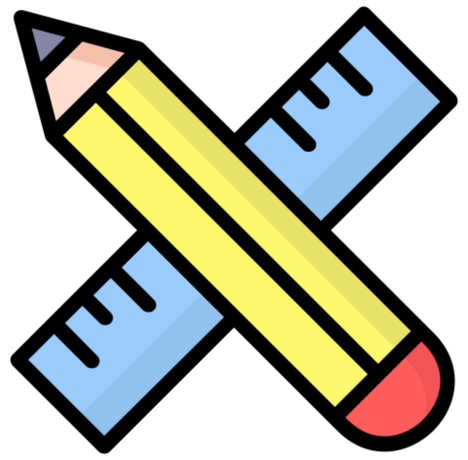
```
Theorem len_rev_unchanged:
forall (A: Type) (l: list A),
 length (rev l) = length l.
```

```
Lemma app_length : forall l l' : list A,
 length (l++l') = length l + length l'.
```

Software Development Life Cycle



Requirements →



Design →

Implementation

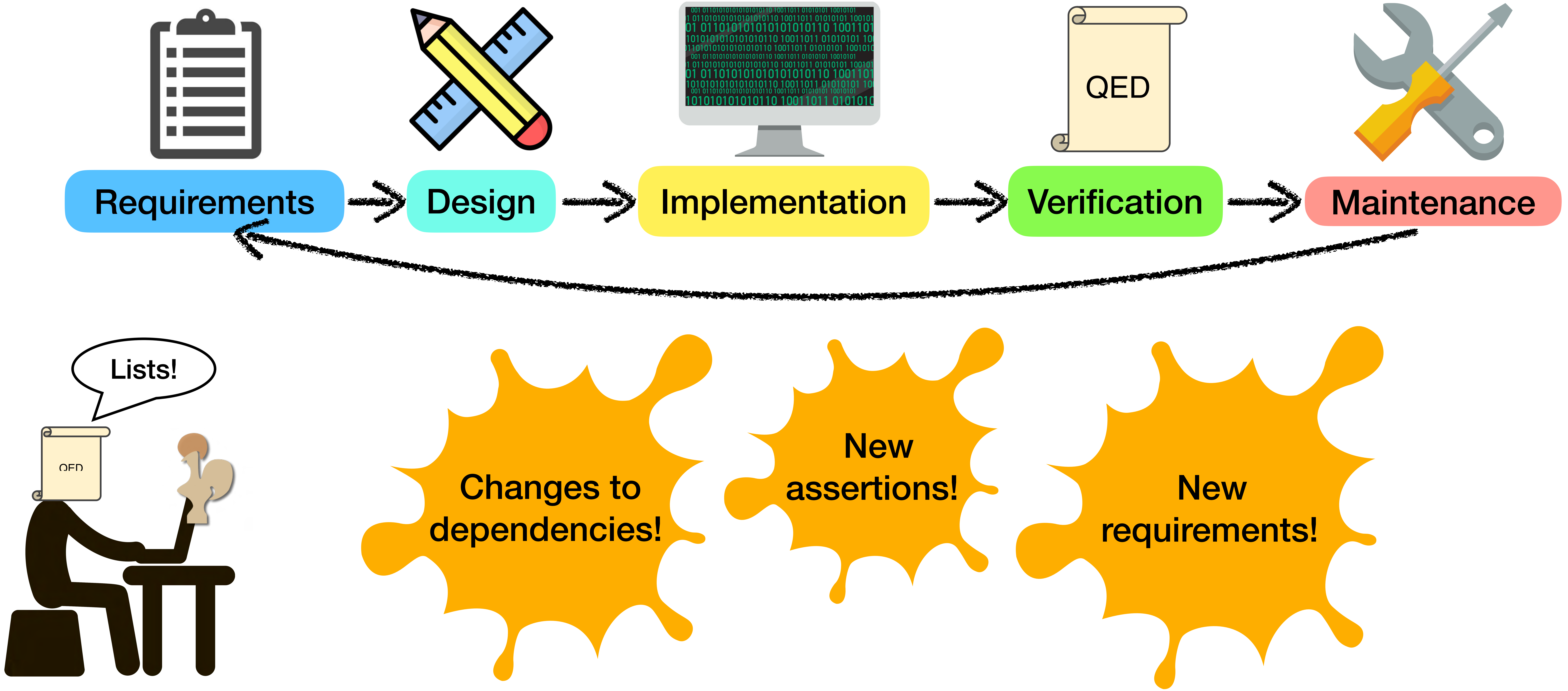

```
Fixpoint rev (l:list A) : list A :=
    match l with
      | [] => []
      | x :: l' => rev l' ++ [x]
    end.
```

Software Development Life Cycle

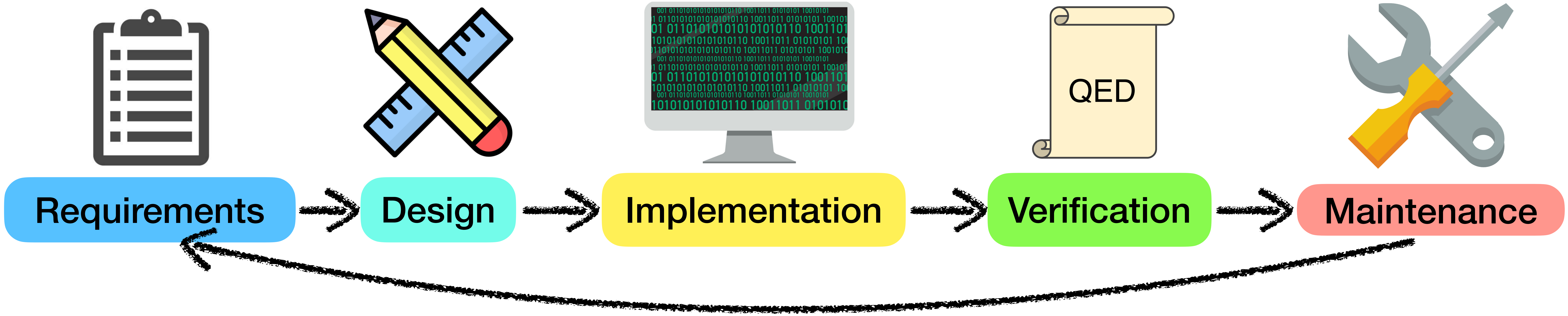
Requirements → Design → Implementation → Verification


```
Proof.
    induction l.
    - auto.
    - assert (H: rev (a :: l) = (rev l) ++ [a]) by auto.
      rewrite H.
      simpl.
      rewrite app_length.
      simpl.
      rewrite IHl.
      rewrite PeanoNat.Nat.add_1_r.
      reflexivity.
Qed.
```

Software Development Life Cycle



Software Development Life Cycle



Does anyone actually do this?

Formal Software Verification: real-world examples

AbsInt/CompCert

The CompCert formally-verified C compiler

CERTIKOS

CAKEML

A Verified Implementation of ML

seL4

Quark : A Web Browser with a Formally Verified Kernel

University of California, San Diego
Computer Science and Engineering

Funded by NSF Award 1228967

Formal Software Verification: real companies do it

AIRBUS

BEDROCK
Systems Inc

galois

CERTORA

android

aws

Formal Verification: can produce better quality software

(2011)

Csmith

AbsInt/CompCert

The CompCert formally-verified C compiler

GCC

LLVM

COMPILER INFRASTRUCTURE

CompCert was the only one for which Csmith could not find bugs!

Yang et al (2011) “Finding and Understanding Bugs in C Compilers”

Prohibitively difficult

Verified software requires a lot of time and a lot of proofs in relation to code

Formal Certification of a Compiler Back-end
or: Programming a Compiler with a Proof Assistant

Xavier Leroy
INRIA Rocquencourt
Xavier.Leroy@inria.fr

Proof is about 8 times bigger than the compiler code

3 person years of work

Comprehensive Formal Verification of an OS Microkernel

GERWIN KLEIN, JUNE ANDRONICK, KEVIN ELPHINSTONE, TOBY MURRAY, THOMAS SEWELL, RAFAL KOLANSKI, and GERNOT HEISER, NICTA and UNSW, Sydney, Australia

11 person years of work

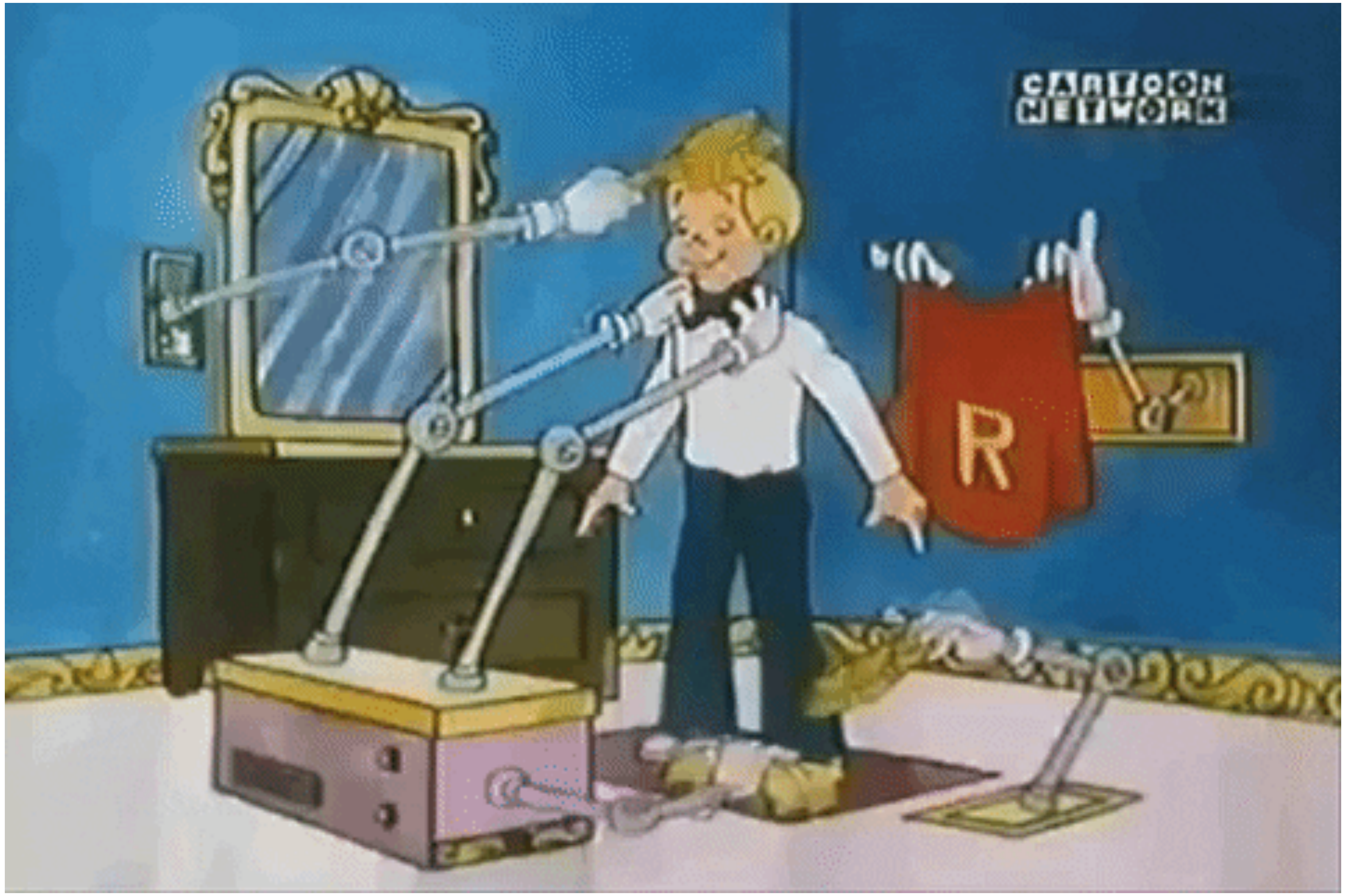
Virtually all software that ships today is unverified.

AbsInt/CompCert

The CompCert formally-verified C compiler

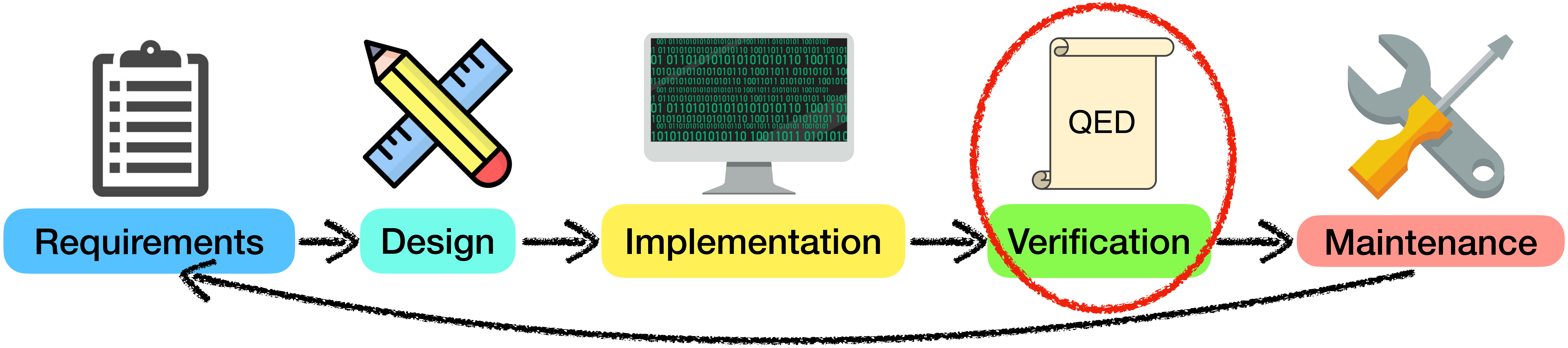
seL4

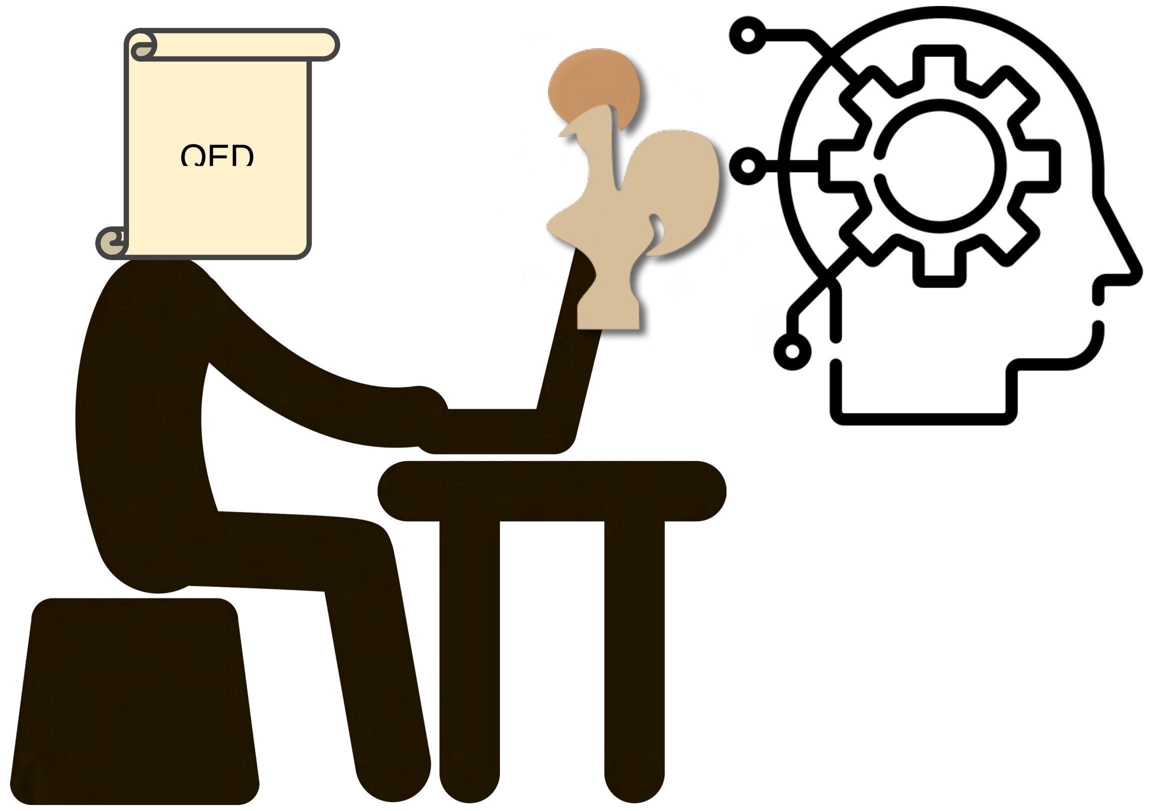
How do programmers deal with hard things?



Automation!

Software Development Life Cycle





- Automating the process using ML
- Work that has been done with an eye towards ML approaches
- Parts of the process that are largely untouched — opportunities!

QED

Constraint-solver based proof automation

CoqHammer

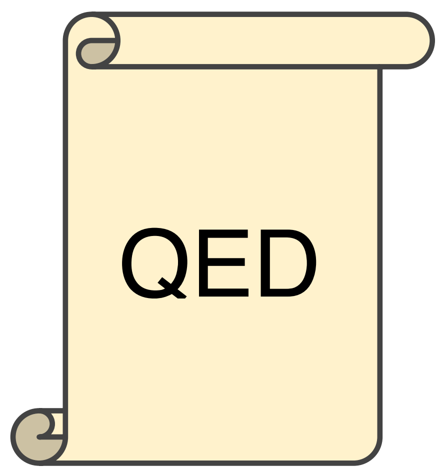
Z3

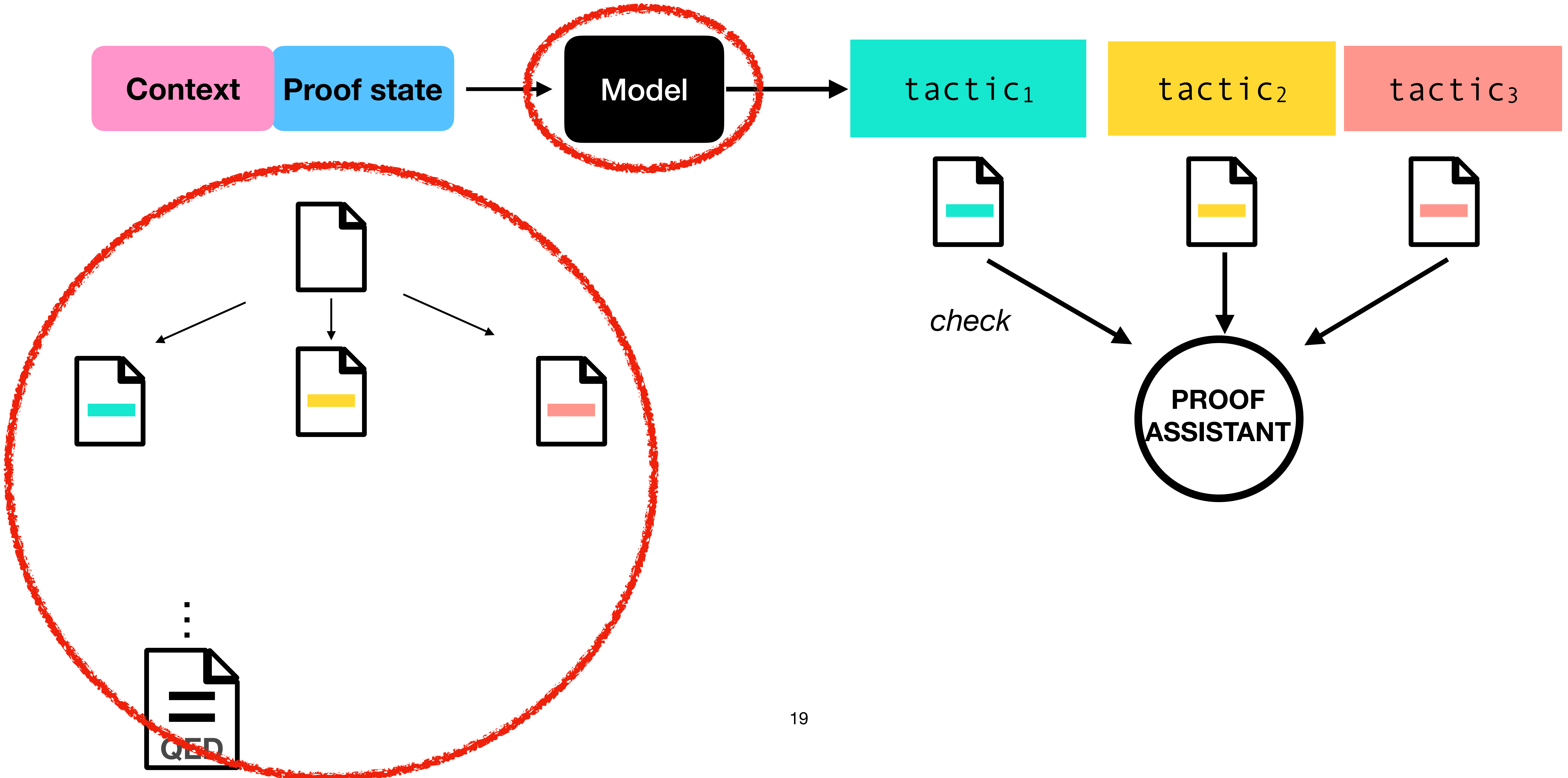
Dafny

- Restricted by precomputed facts
- Cannot perform induction
- Struggle with higher-order logic

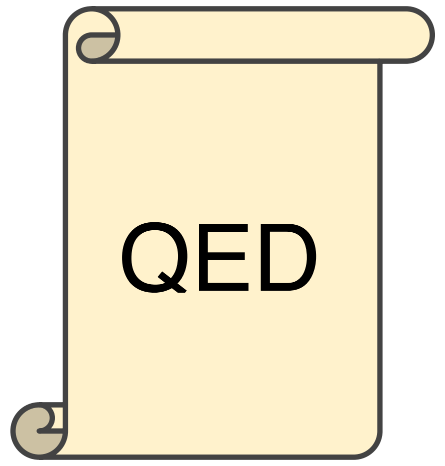
Complementary to machine learning techniques!

Machine Learning: proof synthesis





Machine Learning: proof synthesis



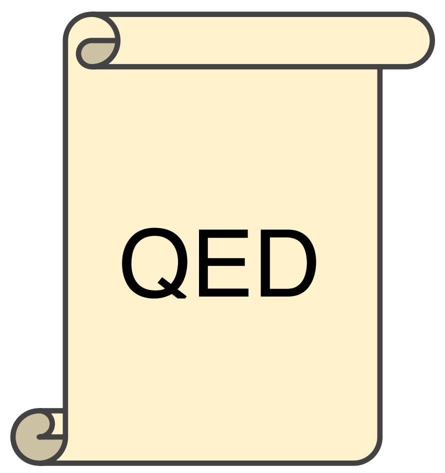
How well does this work for proofs of software correctness?

Succeeds at most <u>**30%**</u> of the time

Are only “easy” proofs being synthesized?

Failing proofs means that your code is not verified!

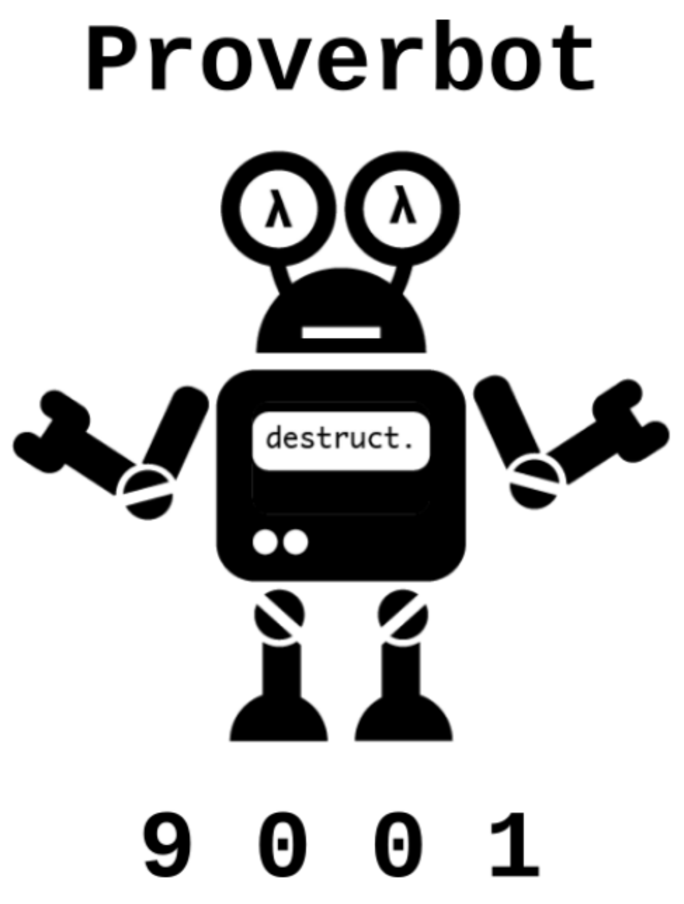
Need methods for debugging and recovering from proof search failures



Proofster

Enter a Coq theorem to prove, or select an example from the drop-down menu

Enter your own theorem

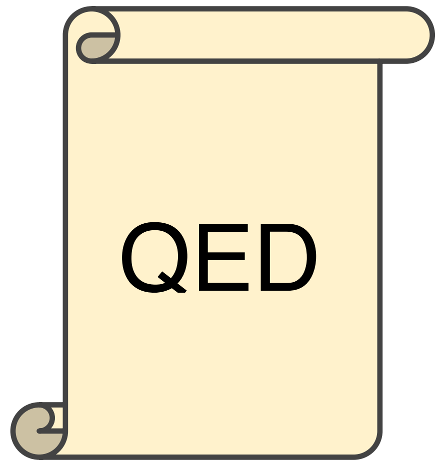


Following the theorem statement, start the proof with “Proof.” and “Admitted.”
Proofster will attempt to replace “Admitted.” with a Coq proof.

Proofster it!

Agrawal et al (2023) “Proofster: Automated Formal Verification”

Proofster



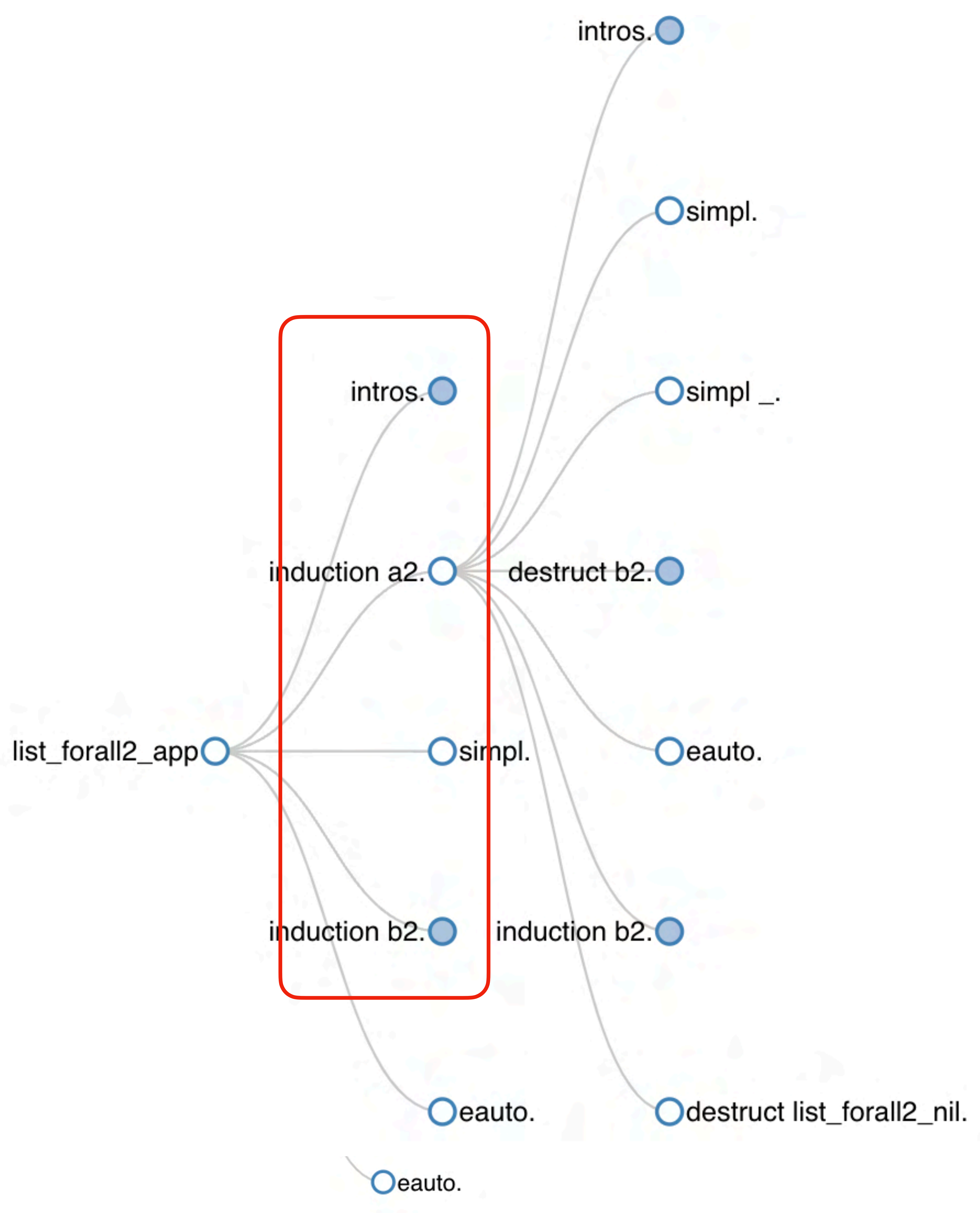
Enter a Coq theorem to prove, or select an example from the drop-down menu

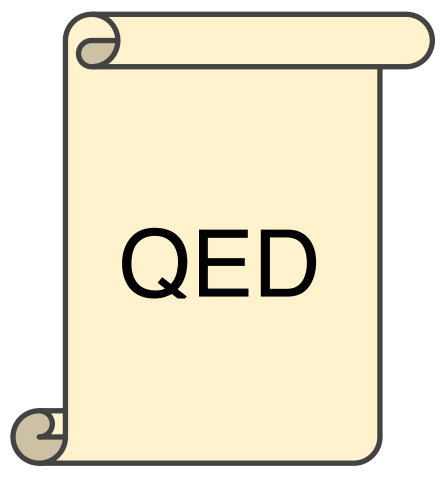
list_forall2_app: If property P holds on corresponding pairs from lists a1, b1 and a2, b2, P also holds on pairs from a1a2, b1b2

```
Require Export List.
Variable A: Type.
Variable B: Type.
Variable P: A -> B -> Prop.

Inductive list_forall2: list A -> list B -> Prop :=
  | list_forall2_nil:
      list_forall2 nil nil
  | list_forall2_cons:
      forall a1 al b1 bl,
      P a1 b1 ->
      list_forall2 al bl ->
      list_forall2 (a1 :: al) (b1 :: bl).

Theorem list_forall2_app:
  forall a2 b2 a1 b1,
  list_forall2 a1 b1 -> list_forall2 a2 b2 ->
  list_forall2 (a1 ++ a2) (b1 ++ b2).
Proof.
Admitted.
```

Following the theorem statement, start the proof with “Proof.” and “Admitted.”
Proofster will attempt to replace “Admitted.” with a Coq proof.

Proofster it!





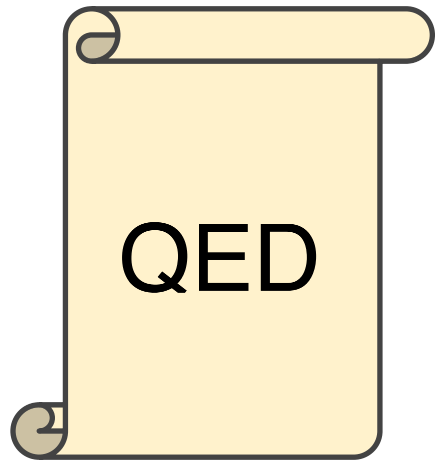
Sorry, I couldn't synthesize a proof of this theorem for you.

list_forall2_app with induction hint

QED

```
Require Export List.
Variable A: Type.
Variable B: Type.
Variable P: A -> B -> Prop.

Inductive list_forall2: list A -> list B -> Prop :=
  | list_forall2_nil:
      list_forall2 nil nil
  | list_forall2_cons:
      forall a1 al b1 bl,
      P a1 b1 ->
      list_forall2 al bl ->
      list_forall2 (a1 :: al) (b1 :: bl).

Theorem list_forall2_app:
  forall a2 b2 a1 b1,
  list_forall2 a1 b1 -> list_forall2 a2 b2 ->
  list_forall2 (a1 ++ a2) (b1 ++ b2).
Proof.
induction 1.
Admitted.
```

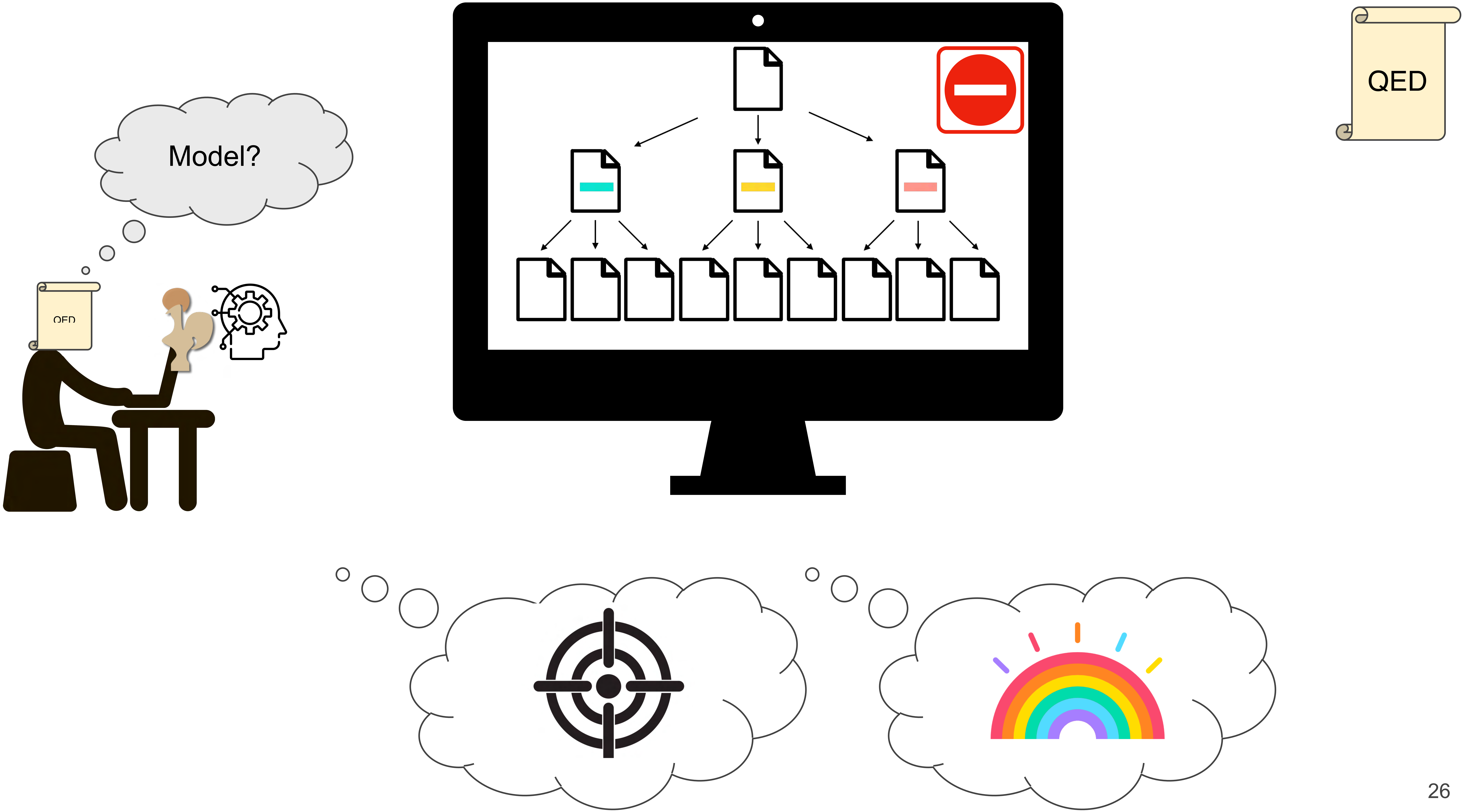
PRoofster




```
Require Export List.
Variable A: Type.
Variable B: Type.
Variable P: A → B → Prop.

Inductive list_forall2: list A → list B → Prop :=
  | list_forall2_nil:
      list_forall2 nil nil
  | list_forall2_cons:
```

Visualization of the proof search tree could help programmer understand why search failed

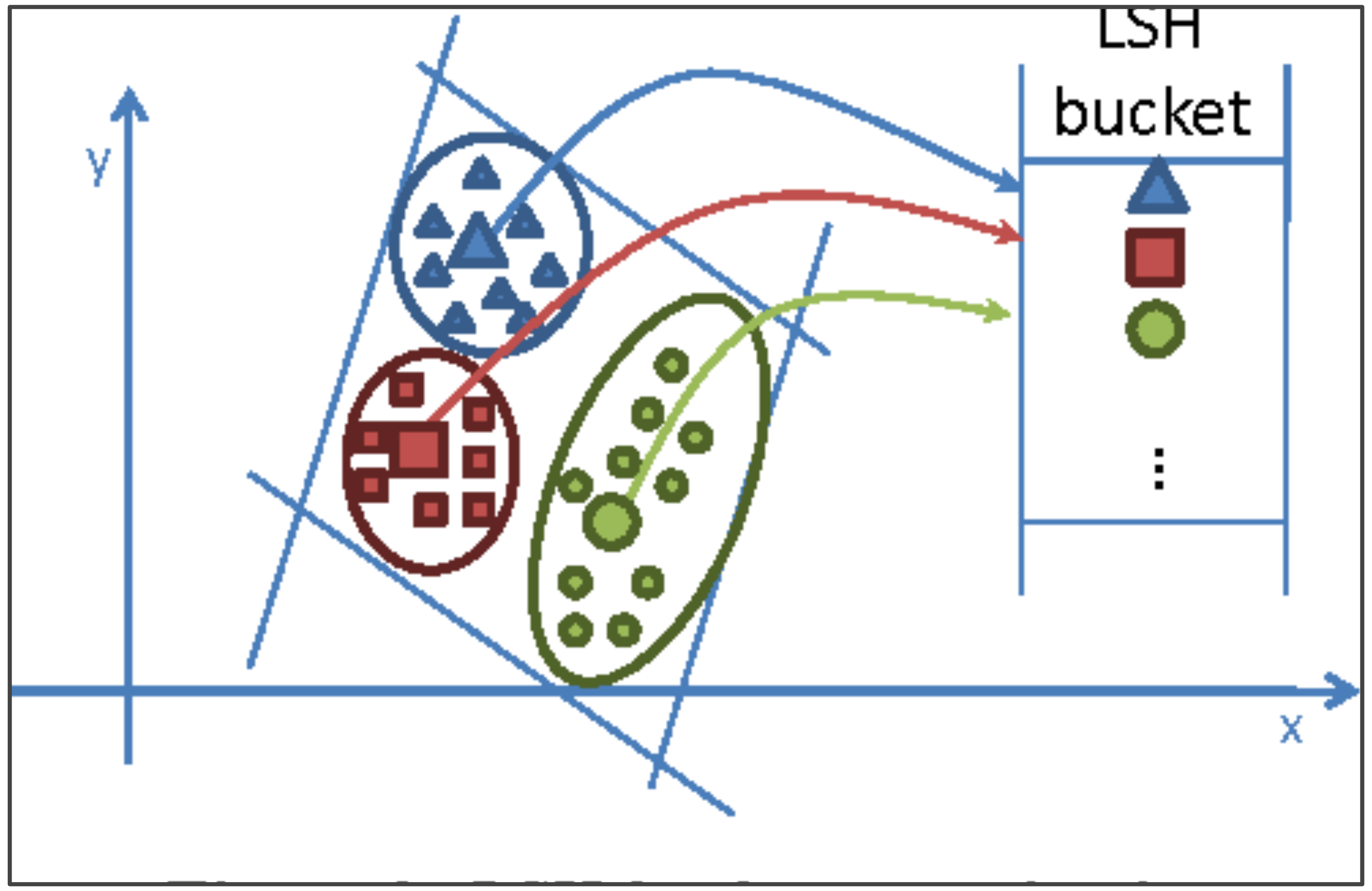
```
    forall a2 b2 a1 b1,
    list_forall2 a1 b1 → list_forall2 a2 b2 →
    list_forall2 (a1 ++ a2) (b1 ++ b2).
Proof.
induction 1.
simpl.
intros.
eauto.
intros.
econstructor.
eauto.
eauto.
Qed.
```


Model?
QED
QED

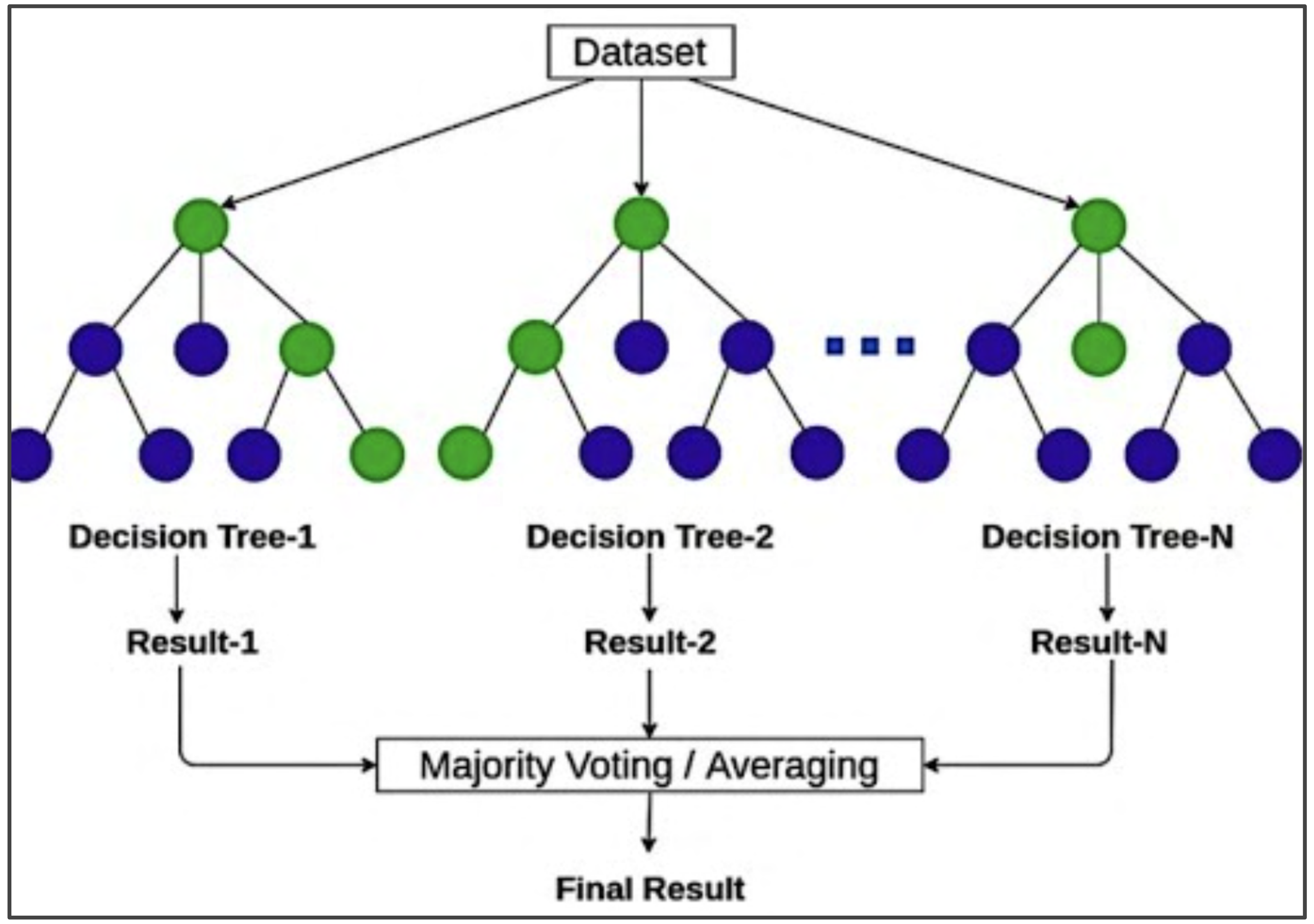
Online Learning

Tactician

QED



Locality sensitive hashing (LSH) forests for online approximate k-NN

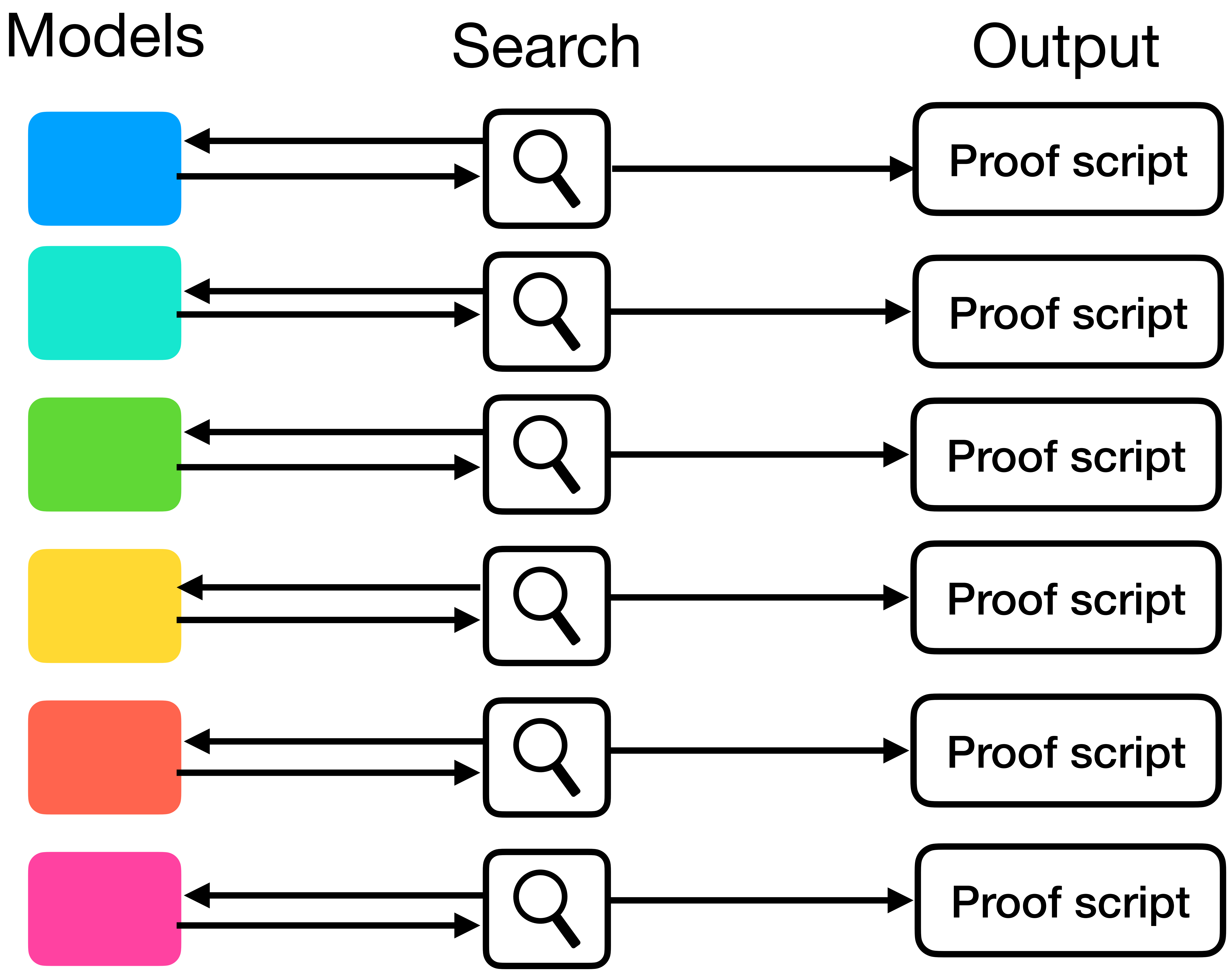


Online random forests

Zhang et al (2021) "Online Machine Learning Techniques for Coq: A Comparison"

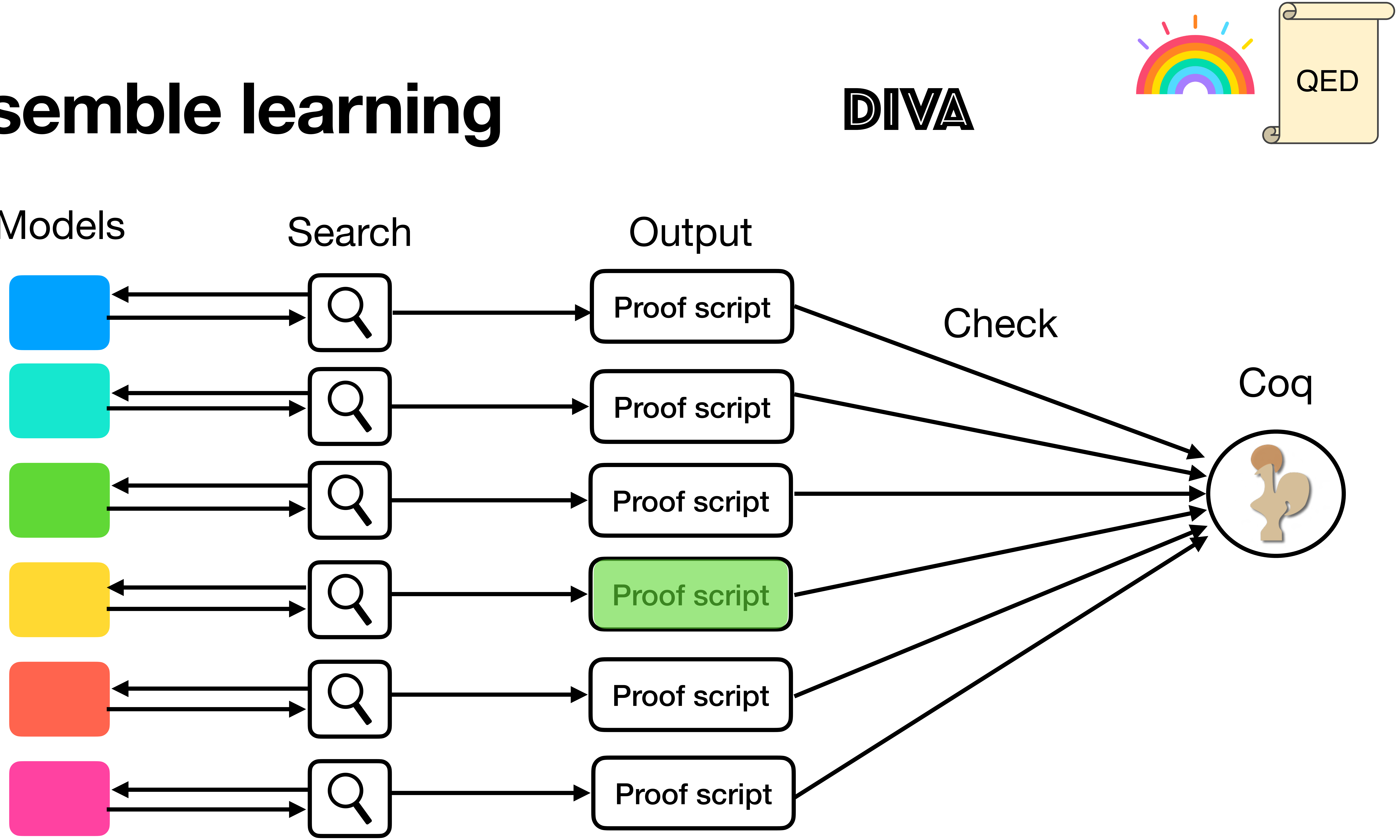
Ensemble learning

DIVA



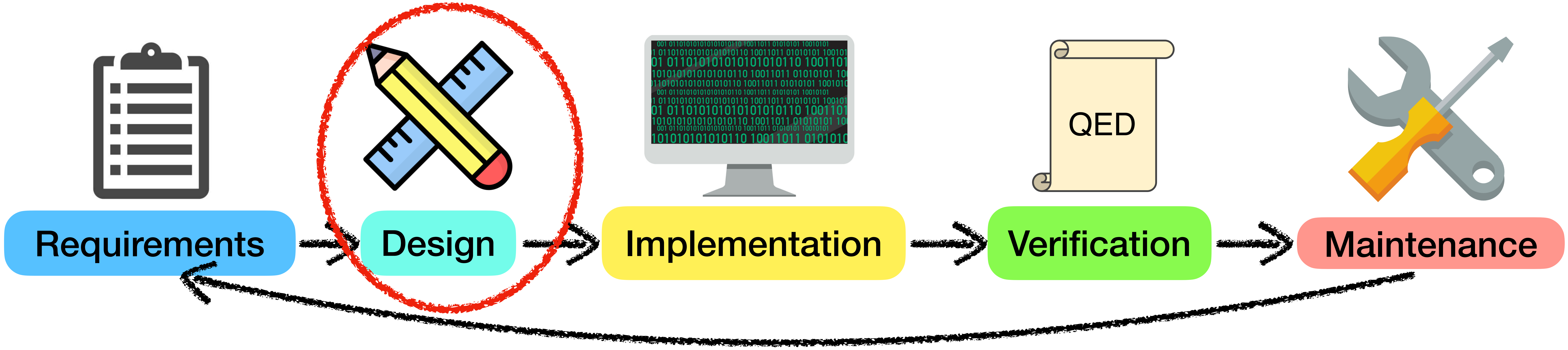
First et al (2022) “Diversity-Driven Automated Formal Verification”

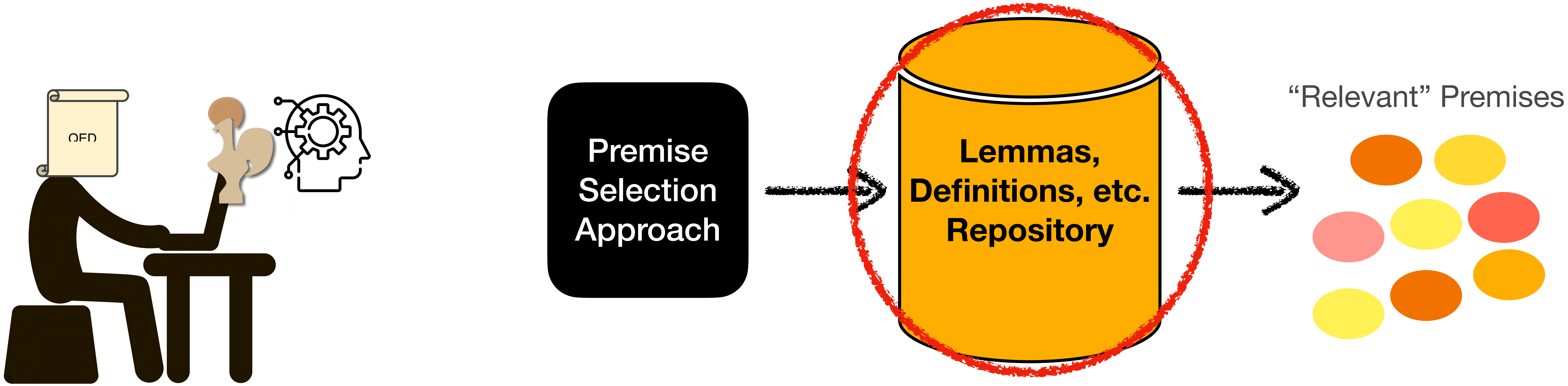
Ensemble learning



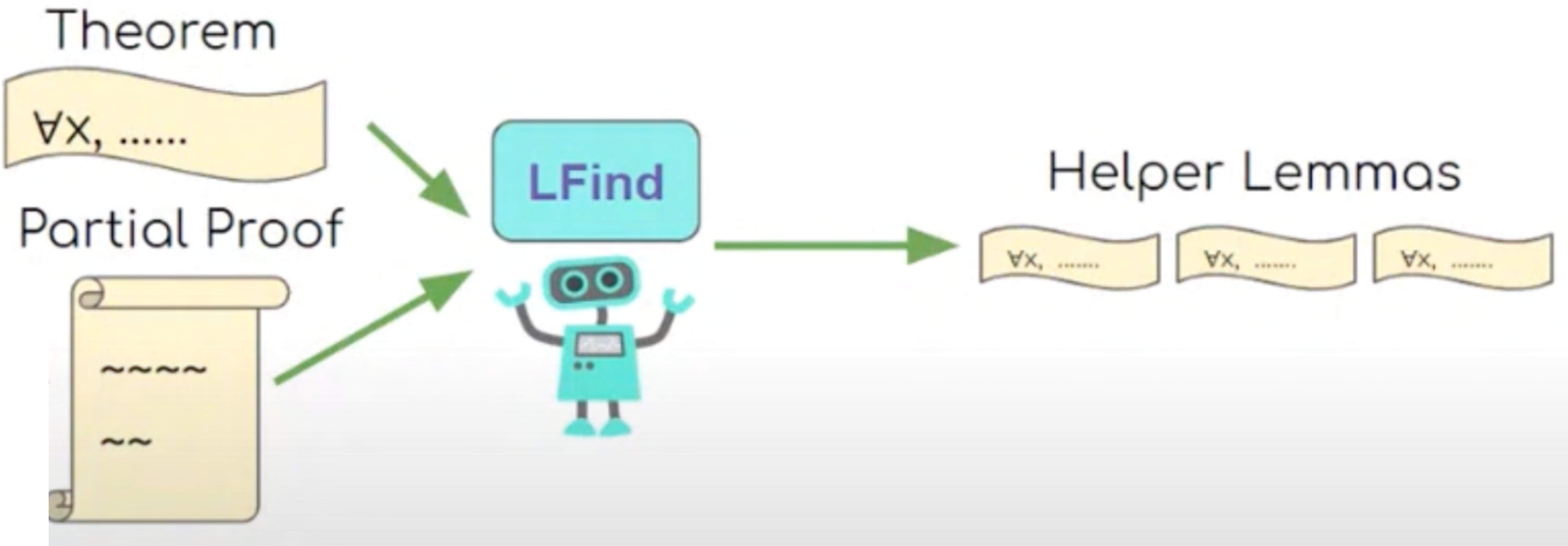
First et al (2022) “Diversity-Driven Automated Formal Verification”

Software Development Life Cycle



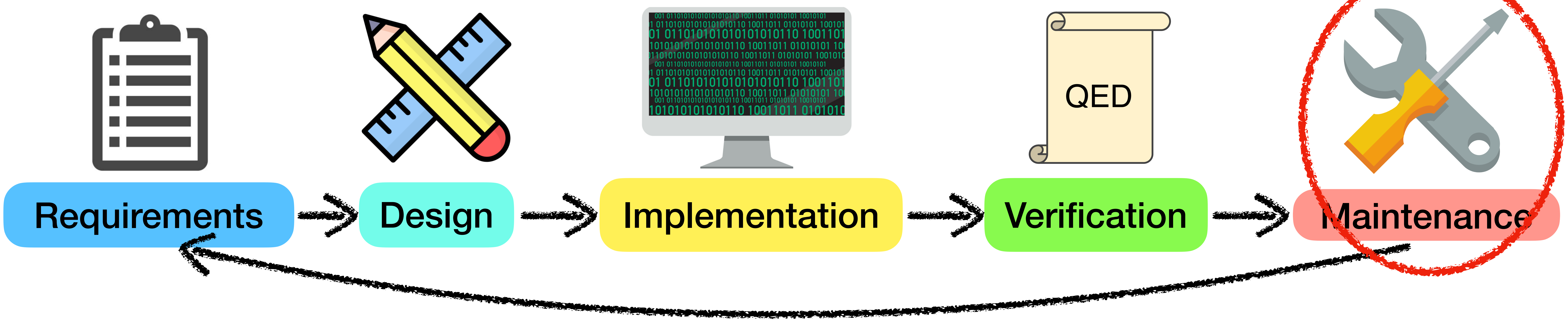


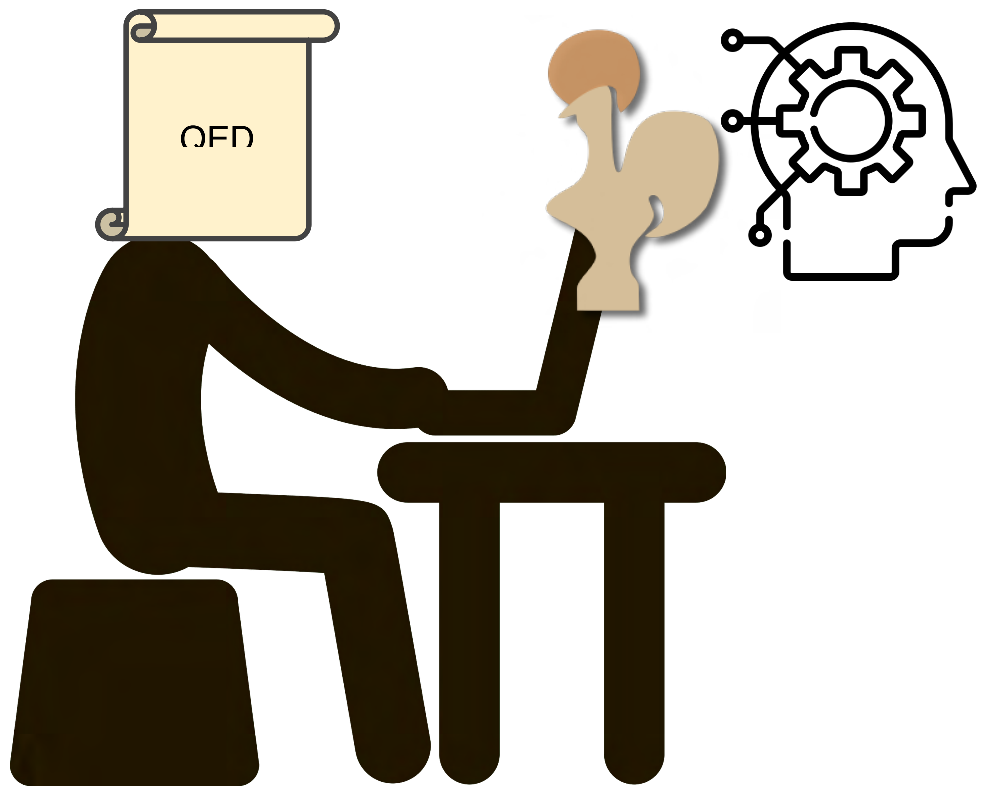
Could potentially be improved using machine learning!



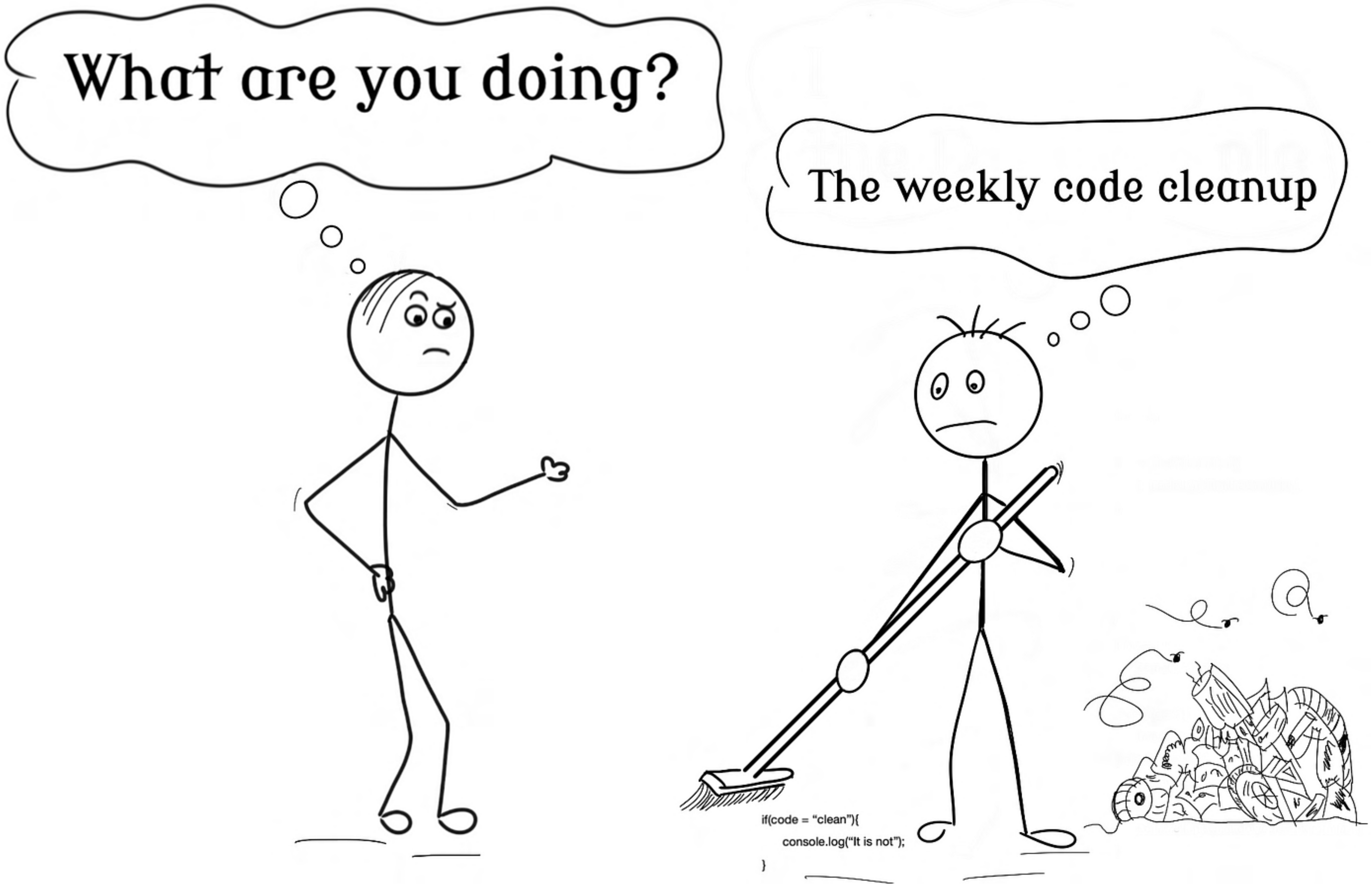
Sivaraman et al (2022) “Data-Driven Lemma Synthesis for Interactive Proofs”

Software Development Life Cycle





Software Refactoring



```
(roosterize) pynie@pynie-ThinkPad-T470:~/roosterize-demo/fcsl-pcm$ ~/projects/roosterize/bin/roosterize \
suggest_naming --file=$PWD/finmap/finmap.v
== Analyzed 110 lemma names, 8 (7.3%) conform to the learned naming conventions.
==========
== 21 can be improved and here are Roosterize's suggestions:
Line 851: fcatsK => eq_fcat (likelihood: 0.45)
Line 822: fcatC => eq_fcat (likelihood: 0.44)
Line 862: fcatKs => eq_fcat (likelihood: 0.43)
Line 1178: zi
Line 1118: m
Line 1258: z
Line 769: disjC => eq_disj (likelihood: 0.30)
Line 962: mapf_disj => eq_map (likelihood: 0.29)
Line 526: fcats0 => fcat_nil (likelihood: 0.28)
Line 1273: zunit_disj => disj_zip (likelihood: 0.27)
Line 1186: zip_supp => eq_zip (likelihood: 0.27)
Line 937: mapf_ins => map_ins (likelihood: 0.26)
Line 525: fcat0s => fcat_nil (likelihood: 0.25)
Line 443: seqof_ins => path_ordP (likelihood: 0.24)
```

LLMs would likely help even more!

RNNs to learn and suggest ***lemma names***

Nie et al (2021) “Roosterize: Suggesting Lemma Names for Coq Verification Projects Using Deep Learning”

```
Lemma sec_left_sum_tree (X Y:Set) (p : WFT X):
  forall (A : X -> X -> Prop), SecureBy A p ->
  SecureBy (left_sum_lift A) (left_sum_tree Y p).
induction p.
  intros A Zsec.
  simpl in *. intros v w x y z.
  destruct x; (repeat (auto; firstorder)).
  destruct v; (repeat (auto; firstorder)).
  destruct w; (repeat (auto; firstorder)).
  destruct v; (
  destruct w; (
  intros. simpl
  eapply sec_strengthen. Focus 2. apply H. apply H0.
  intros. destruct x0; repeat (auto; firstorder).
         destruct y; repeat (auto; firstorder).
  simpl in *. intro x.
  destruct x; repeat (auto; firstorder).
  eapply sec_strengthen. Focus 2. apply H. apply H0.
  intros. destruct x0; repeat (auto;firstorder).
         destruct y0; repeat (auto;firstorder).
Defined.
```

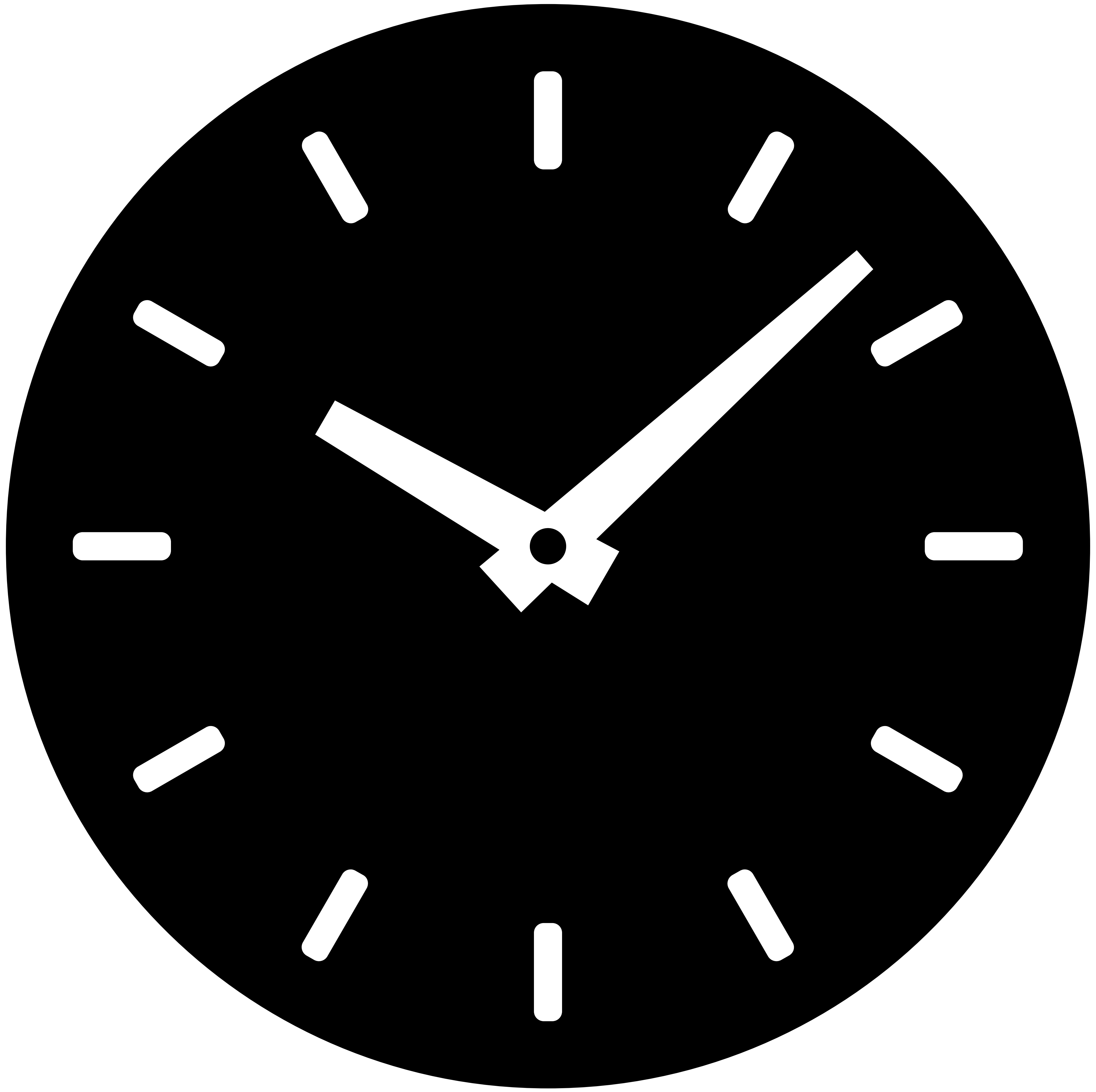
```
Lemma list_find_app_Some l1 l2 i x :
  list_find P (l1 ++ l2) = Some (i,x) ↔
    list_find P l1 = Some (i,x) ∨
    length l1 ≤ i ∧ list_find P l1 = None ∧ list_find P l2 = Some (i - length l1,x).
Proof.
  split.
  - intros ([?|[??]]%lookup_app_Some&?&Hleast)%list_find_Some.
    + left. apply list_find_Some; eauto using lookup_app_l_Some.
    + right. split; [lia|]. split.
      { apply list_find_None, Forall_lookup. intros j z ??.
        assert (i < length l1) by eauto using lookup_lt_Some.

      by rewrite lookup_app_r, minus_plus by lia.
  - intros [(?&?&Hleast)%list_find_Some|(?&Hl1&(?&?&Hleast)%list_find_Some)].
    + apply list_find_Some. split_and!; [by auto using lookup_app_l_Some..|].
      assert (i < length l1) by eauto using lookup_lt_Some.
      intros j y ?%lookup_app_Some; naive_solver eauto with lia.
    + rewrite list_find_Some, lookup_app_Some. split_and!; [by auto..|].
      intros j y [?|?]%lookup_app_Some ?; [|naive_solver auto with lia].
      by eapply (Forall_lookup_1 (not ∘ P) l1); [by apply list_find_None|..].
Qed.
```

LLMs would likely help even more!

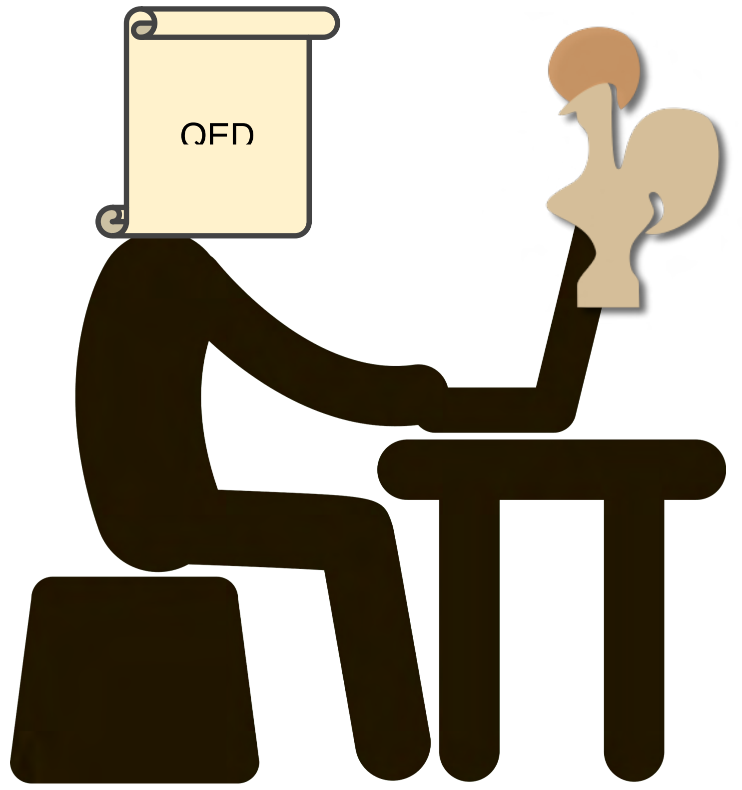
RNNs and N-grams to learn and suggest *space formatting*

Nie et al (2020) “Learning to Format Coq Code Using Language Models”

Software Evolution



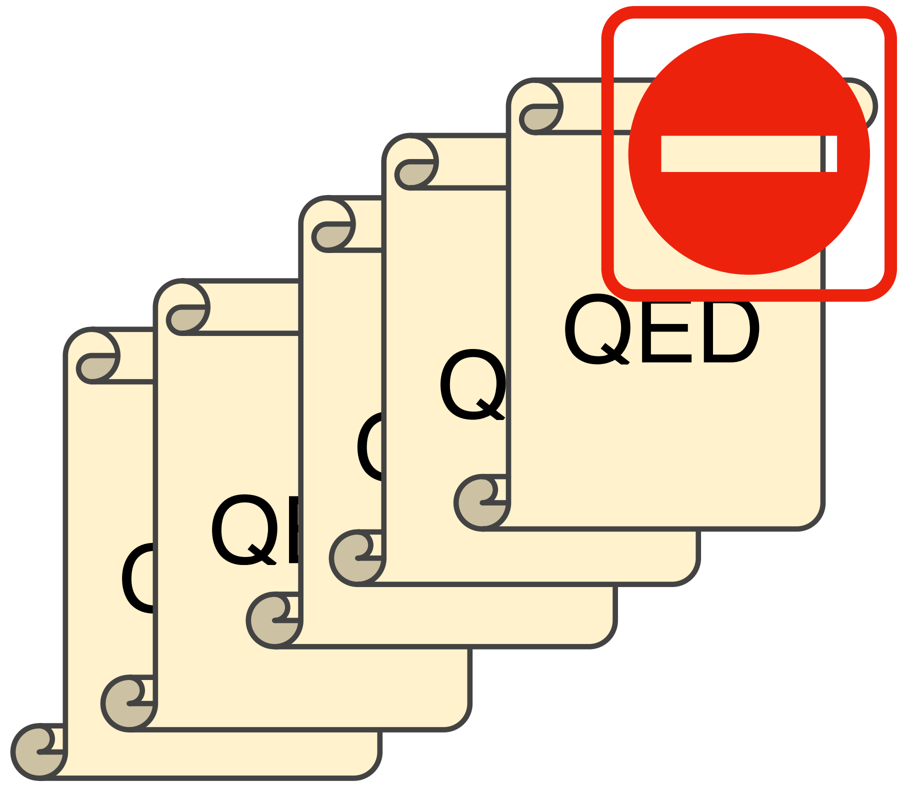
Change to dependency!




```
Lemma proc_rspec_crash_refines_op T (p : proc C_Op T)
  (rec : proc C_Op unit) spec (op : A_Op T) :
  (forall sA sC,
-   absr sA sC tt -> proc_rspec c_sem p rec (refine_spec spec sA)) ->
-  (forall sA sC, absr sA sC tt -> (spec sA).(pre)) ->
+   absr sA (Val sC tt) -> proc_rspec c_sem p rec (refine_spec spec sA)) ->
+  (forall sA sC, absr sA (Val sC tt) -> (spec sA).(pre)) ->
  (forall sA sC sA' v,
-   absr sA' sC tt ->
+   absr sA' (Val sC tt) ->
    (spec sA).(post) sA' v -> (op_spec a_sem op sA).(post) sA' v) ->
  (forall sA sC sA' v,
-   absr sA sC tt ->
+   absr sA (Val sC tt) ->
    (spec sA).(alternate) sA' v -> (op_spec a_sem op sA).(alternate) sA' v) ->
  crash_refines absr c_sem p rec (a_sem.(step) op)
    (a_sem.(crash_step) + (a_sem.(step) op;; a_sem.(crash_step))).
```

Need to change 10+ lemmas and definitions

Not just *tedious* — can be quite *challenging* even for experts!

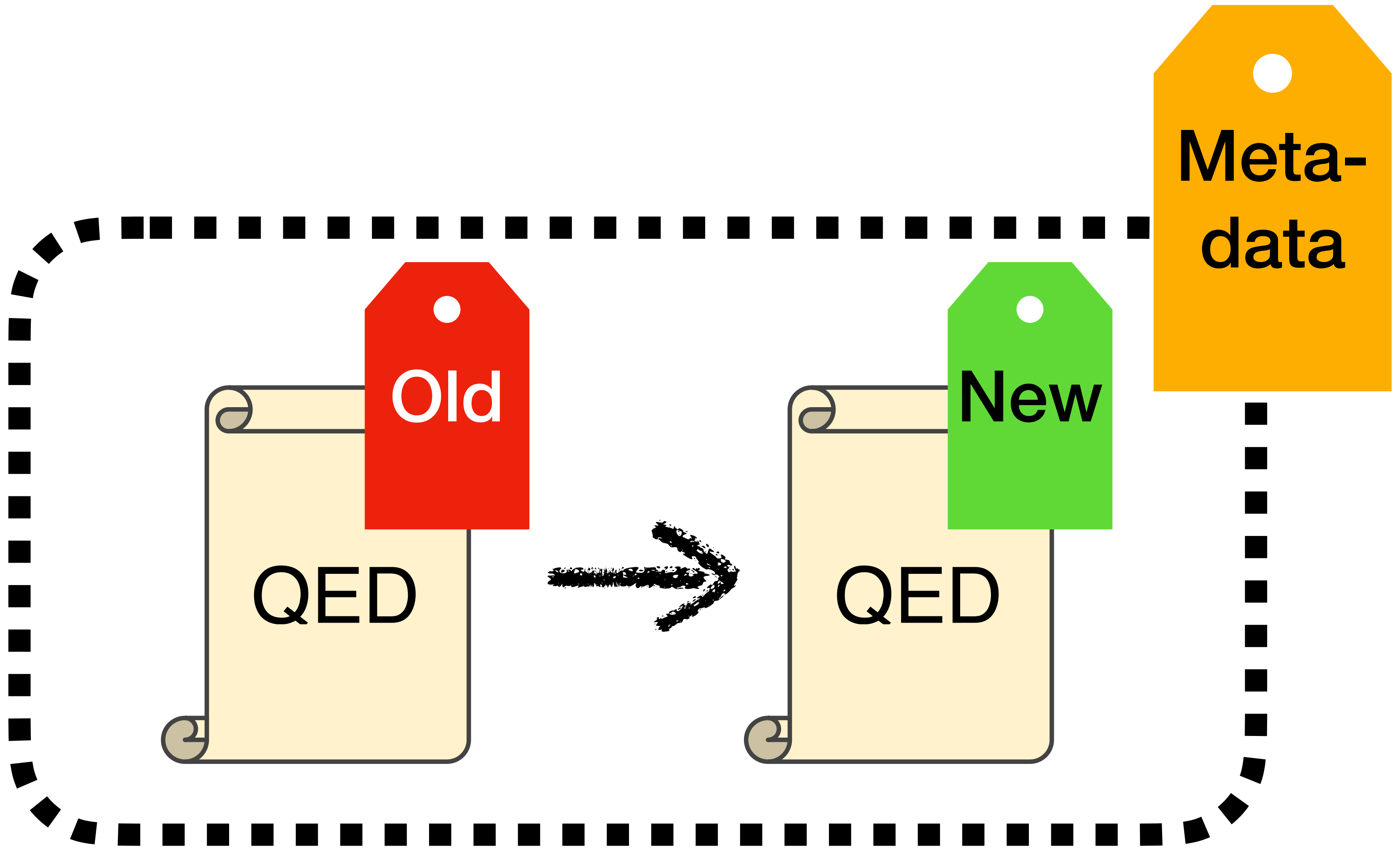


5 broken proofs!

Only able to fix 1 proof!

Ringer et al (2020) “REPLica: REPL instrumentation for Coq Analysis”

Proof repair across commits dataset: PRISM

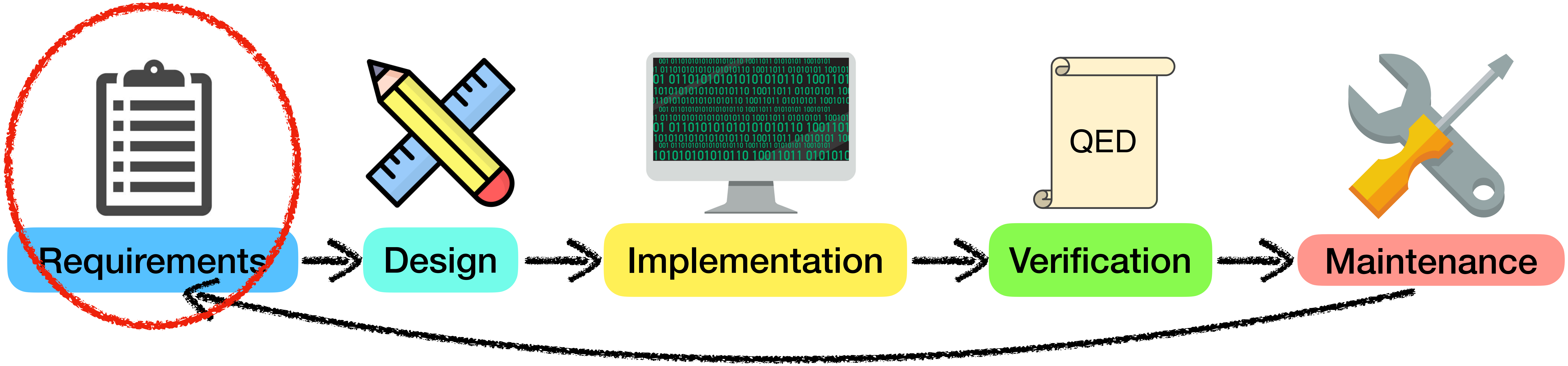


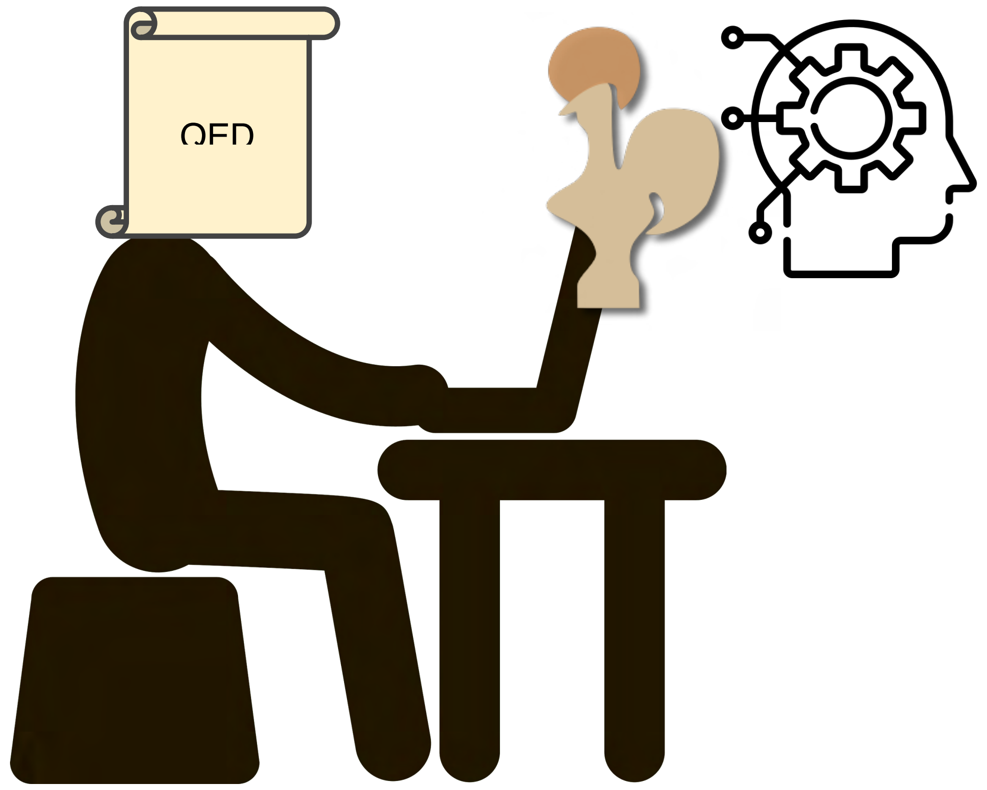
Important for considering large-scale ramifications of changes

Can be used by ML researchers!

Reichel et al (2023) “Proof Repair Infrastructure for Supervised Models: Building a Large Proof Repair Dataset”

Software Development Life Cycle





Autoformalization

Cunningham et al (2022) “Towards Autoformalization of Mathematics and Code Correctness: Experiments with Elementary Proofs”

LaTeX

Theorem. *Consider the following series of commands such that*

```
S := 3;
S := 3 + S * Z;
S := 1 + S * Z
```

Allow $Z = y$, for any natural number y, ahead of running this code then $S = 3 \times y^2 + 3 \times y + 1$ after the set of instructions has executed.

Proof. By application of usual Hoare logic:

$$\{Z = y\}$$

```
S := 3;
```

$$\{Z = y \wedge S = 3\}$$

```
S :=
```

$$\{Z = y \wedge S =$$

```
S :
```

$$\{Z = y \wedge S = 3 \times y^2 + 3 \times y + 1\}$$

Hence, this program is shown to be correct. □

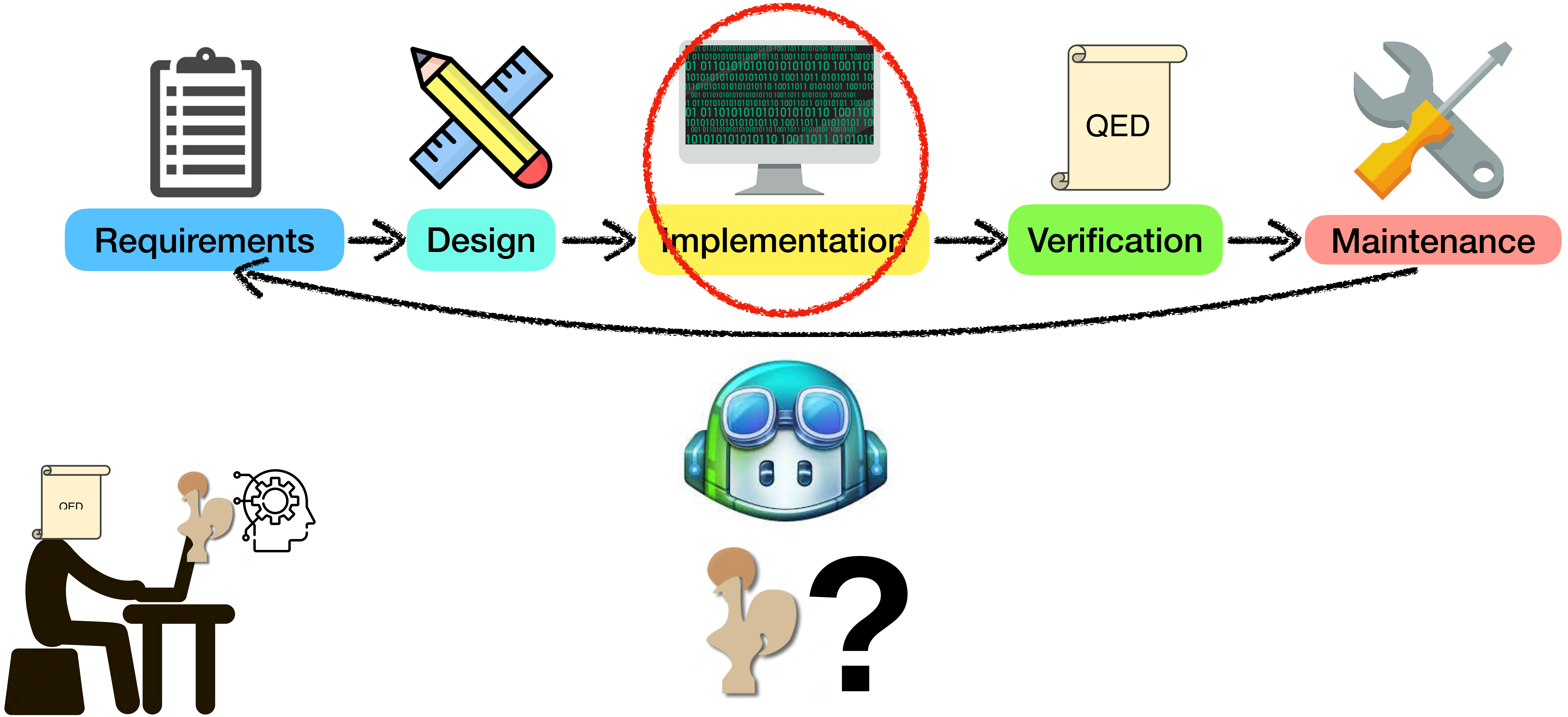
```
Require Import String.
From PLF Require Import Imp.
From PLF Require Import Hoare.
Theorem poly_code_correct:
  forall y : nat,
  {{ Z = y }}
  S := 3;
  S := 3 + S * Z;
  S := 1 + S * Z
  {{ S = 3 * y ^ 2 + 3 * y + 1 }}.
Proof.
  intros.
  apply hoare_seq with
    (Q := (
      (Z = y /\ S = 3)
    )%assertion).
  apply hoare_seq with
  + 3)
  y^2 + 3 * y + 1)
    )%assertion).
  all: eapply hoare_consequence_pre;
  try (apply hoare_asgn || assn_auto'').
Qed.
```

Autoformalization techniques may be useful for verifying code!

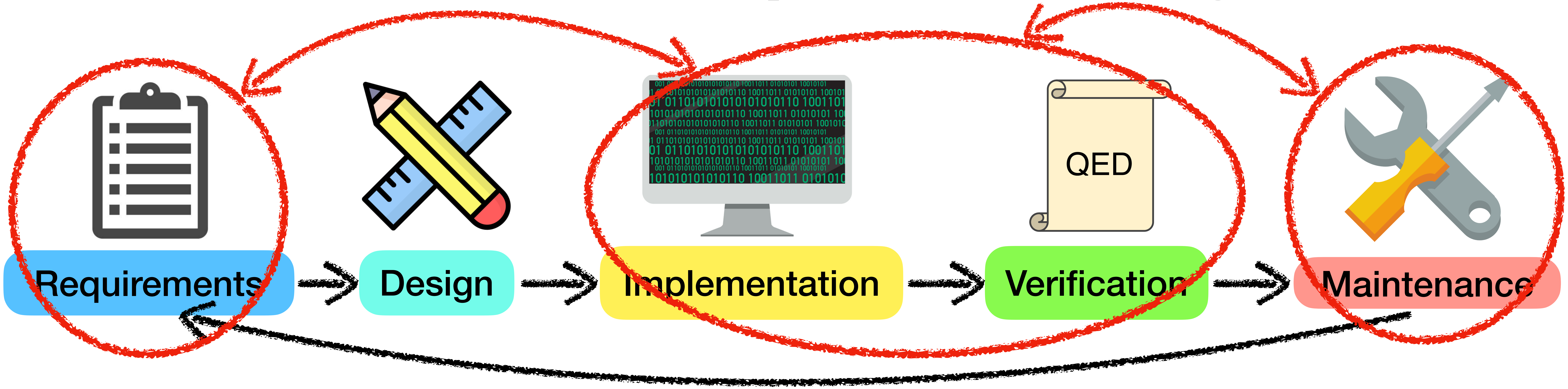
QED

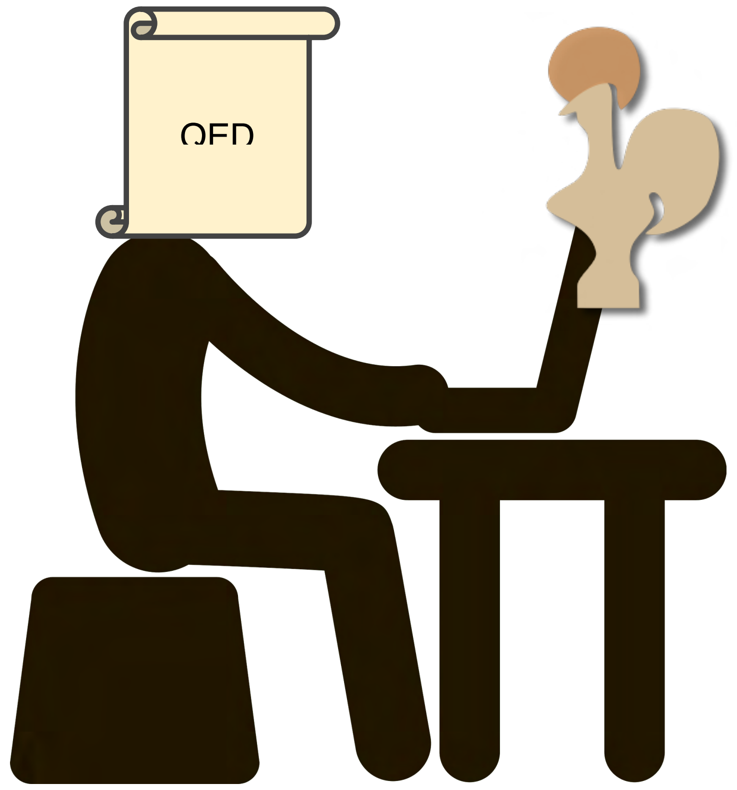
Cunningham et al (2022) "Towards Autoformalization of Mathematics and Code Correctness: Experiments with Elementary Proofs"

Software Development Life Cycle



Software Development Life Cycle

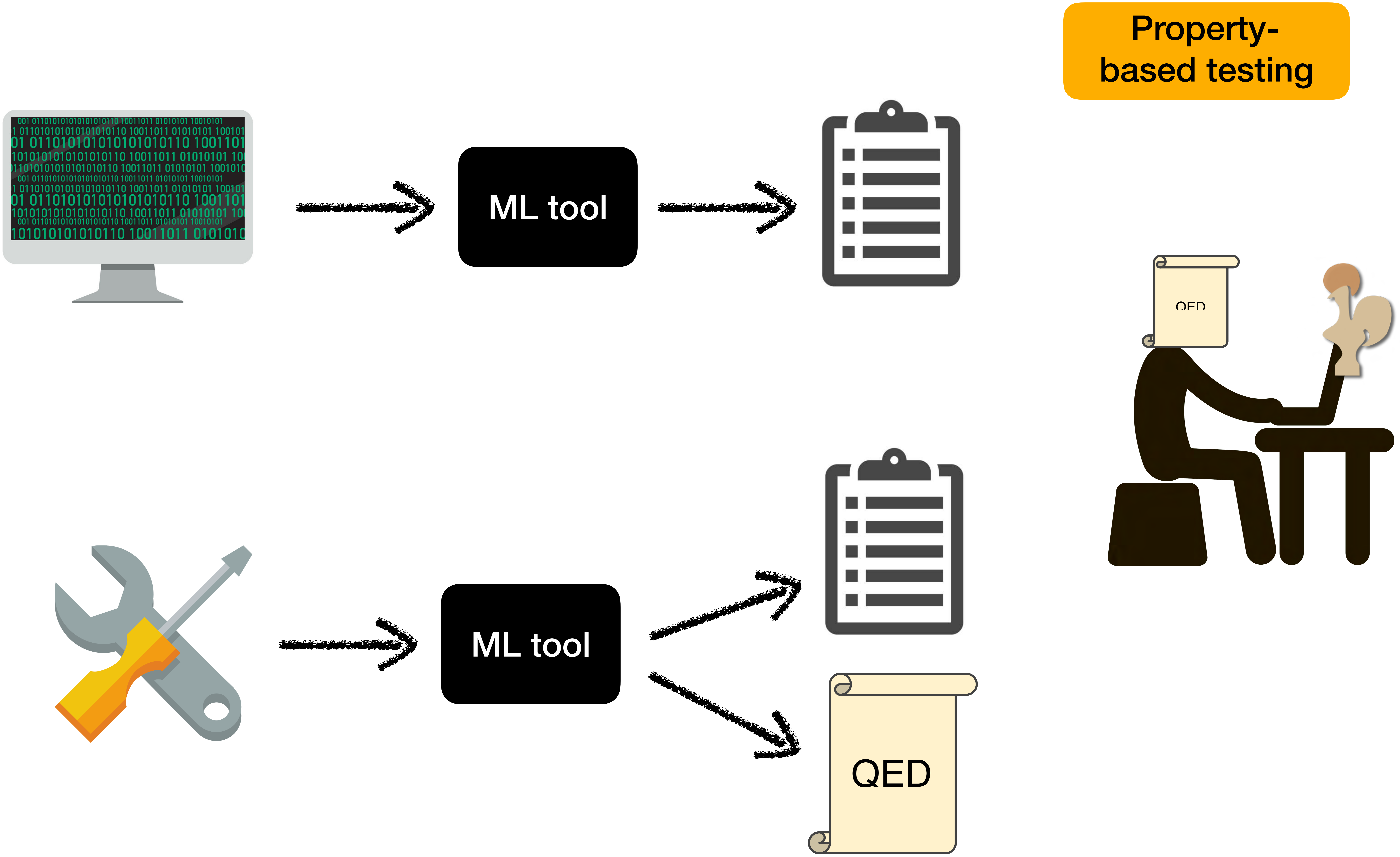




Need to carefully consider the process when developing ML-based tools

Ringer et al (2020) “REPLica: REPL instrumentation for Coq Analysis”

Aspirational ML-based tools



How about an LLM?

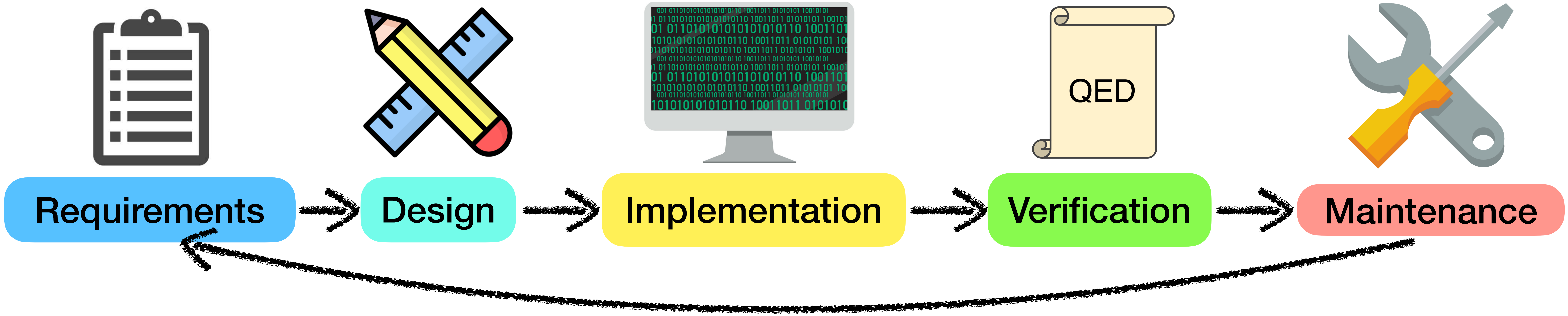
LLMs produce good answers

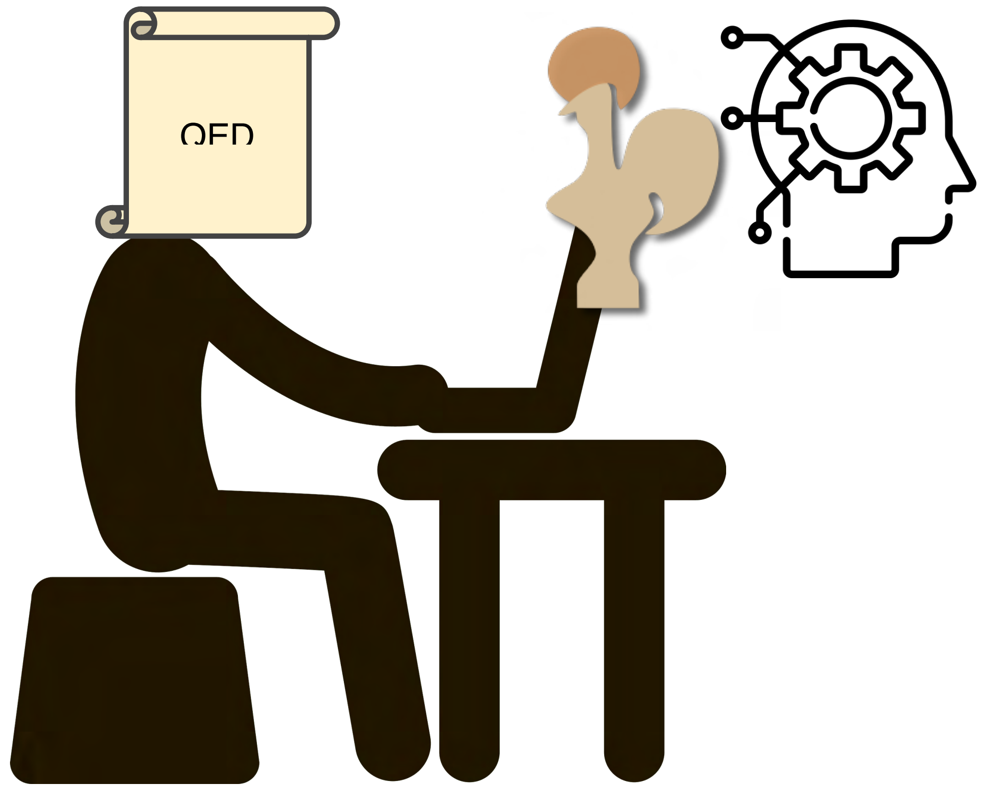
LLMs produce convincing wrong answers

Proof assistant is an oracle

Theorem proving is potentially a power domain for LLM use

Takeaways





- Current research in ML for formal software verification has only just scratched the surface!
- Need more consideration of the software development process
- Will lead to more usable tools for practitioners and adoption of techniques

Panel moderator

Zhangir Azerbayev