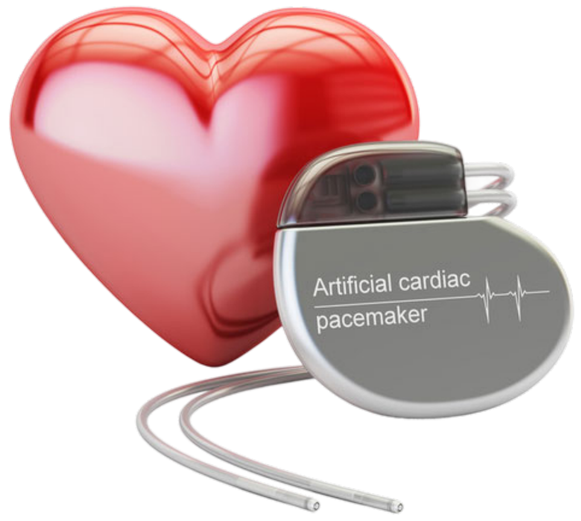
Automating Theorem Proving

Saketh Kasibatla; CSE 230; June 5, 2025

Software is pretty neat!



...but it has problems

CrowdStrike blames global IT outage on bug in checking updates

Historic crash renews focus on lack of accountability for software companies vital for commerce worldwide.

why did it take so long to fix after social platform went down?

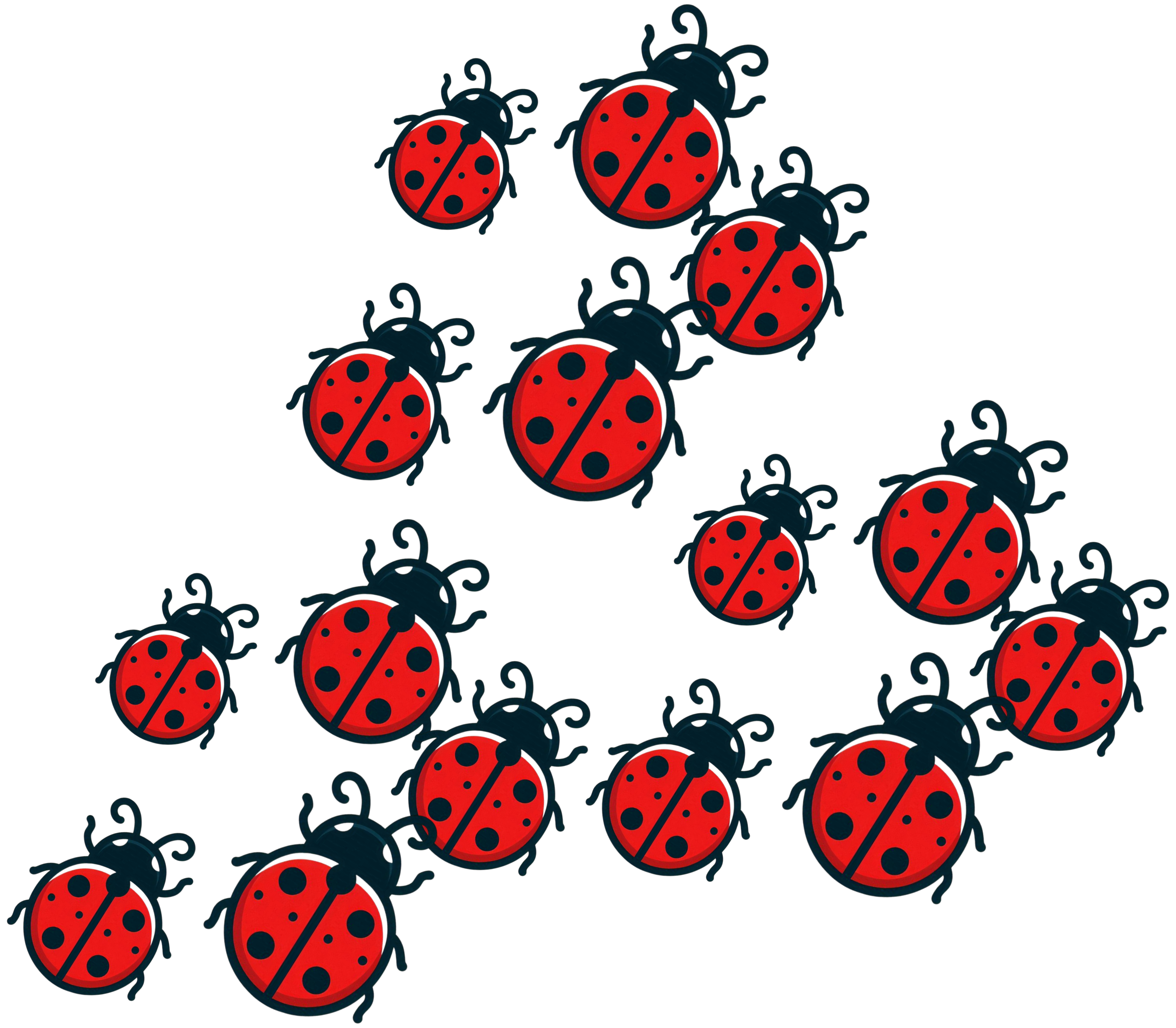
unable to access Facebook, Instagram
urs while the social media giant
services

PBS NEWS HOUR

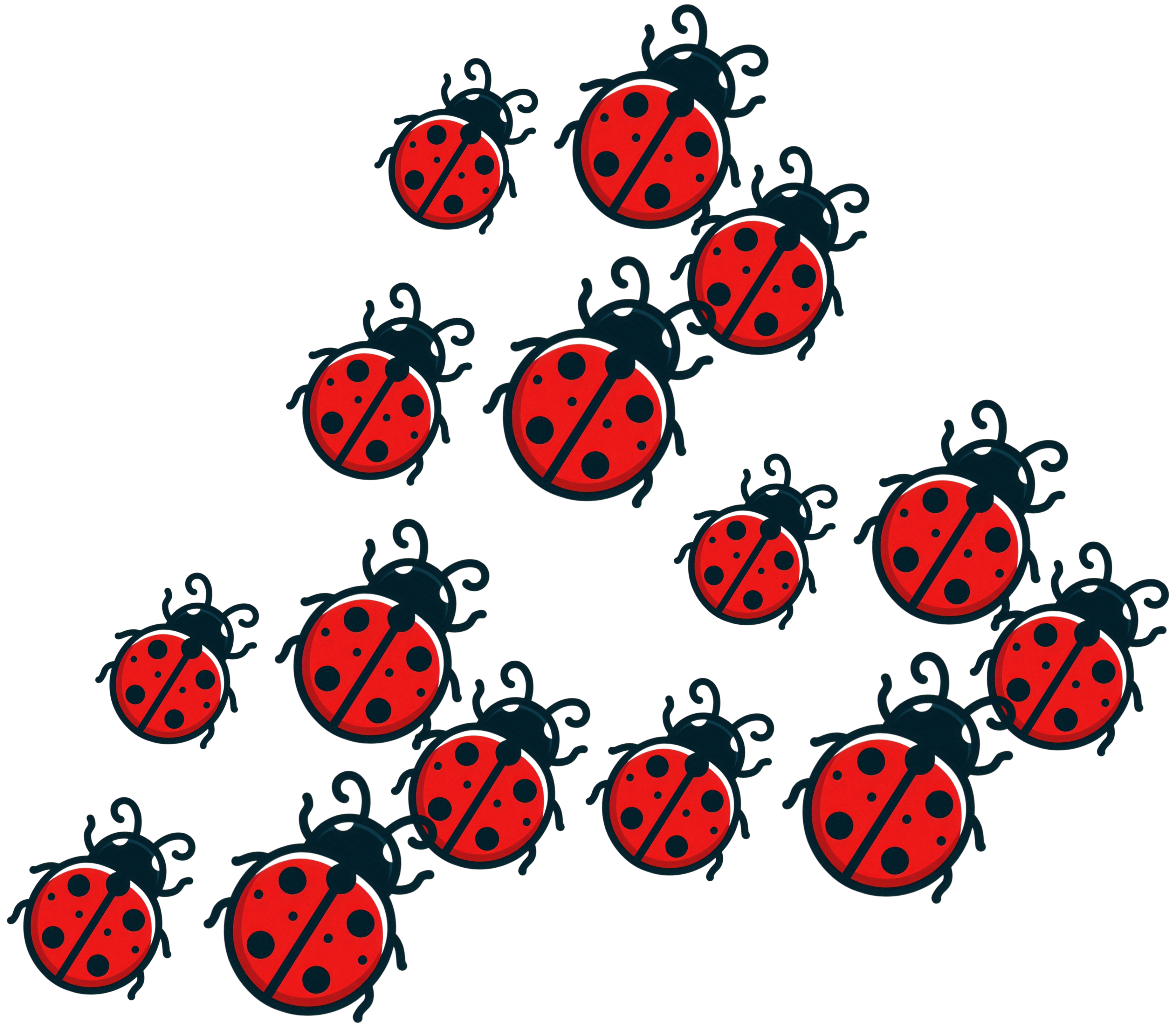
How a faulty software update sparked tech disruptions worldwide

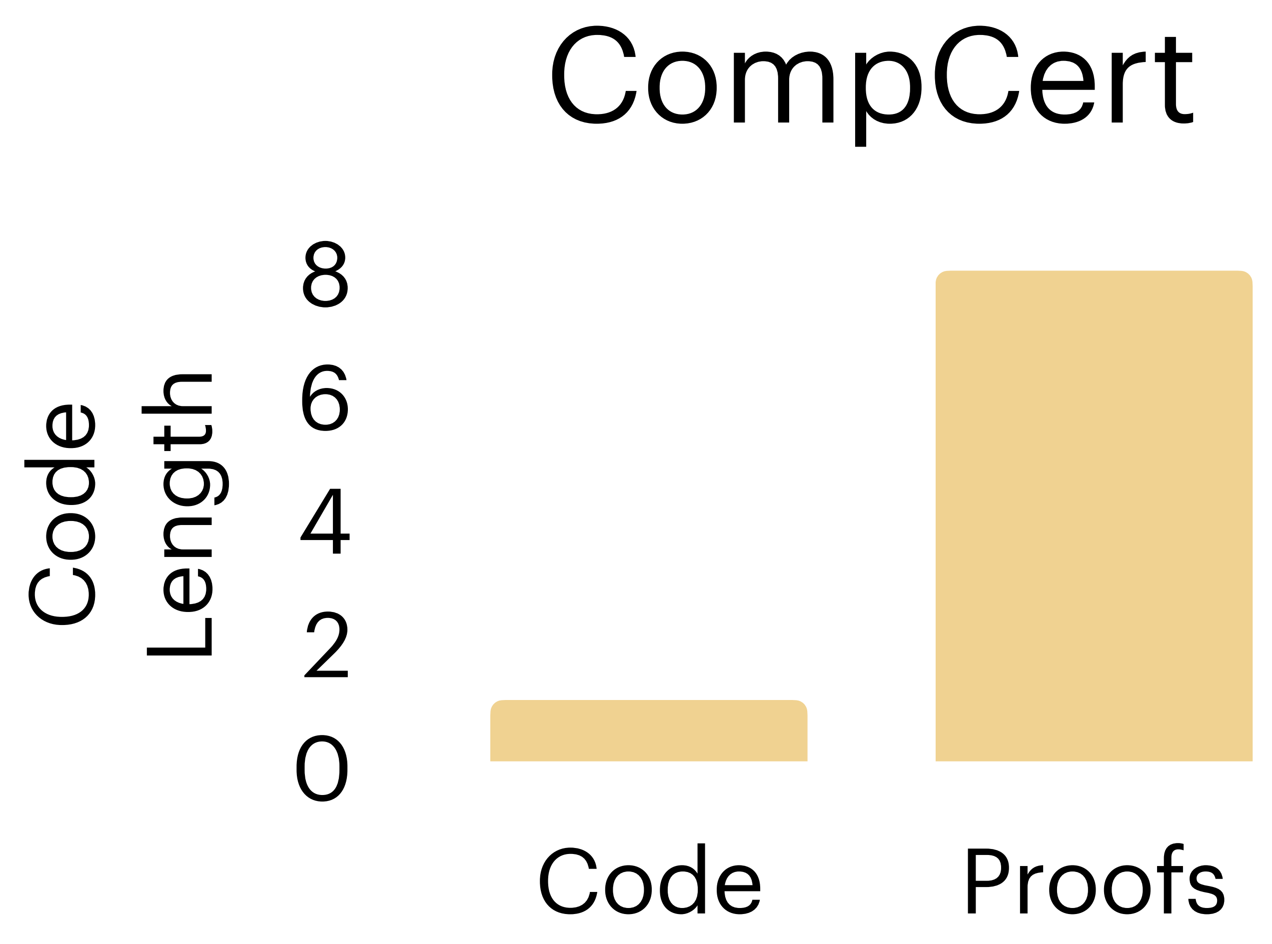
In this 2022 update report we estimate that the cost of poor software quality in the US has grown to at least $2.41 trillion[1], but not in similar proportions as seen in 2020. The accumulated software Technical Debt (TD) has grown to ~$1.52 trillion[1].

Verification and ITPs can help!



… but they're a lot of work

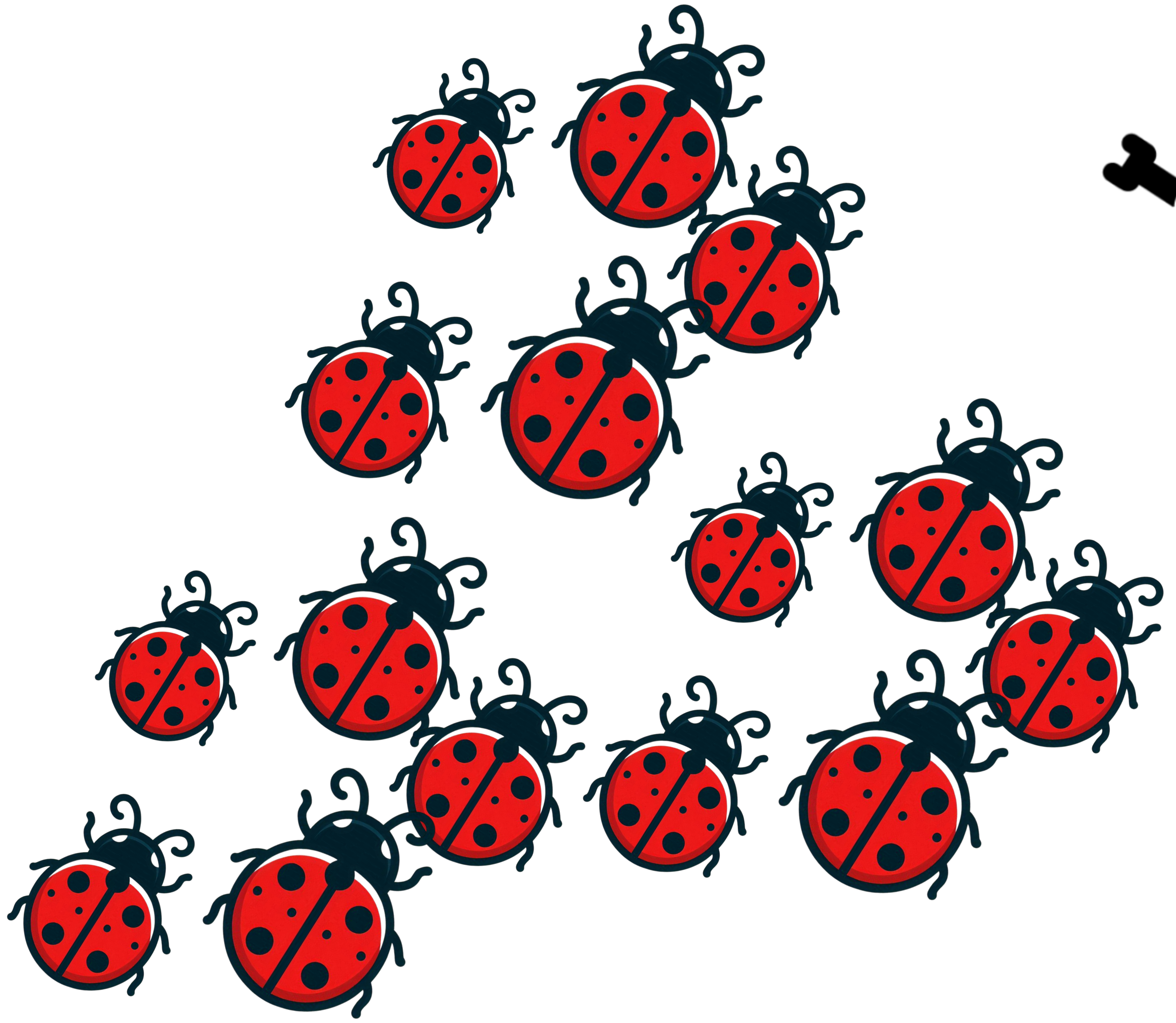


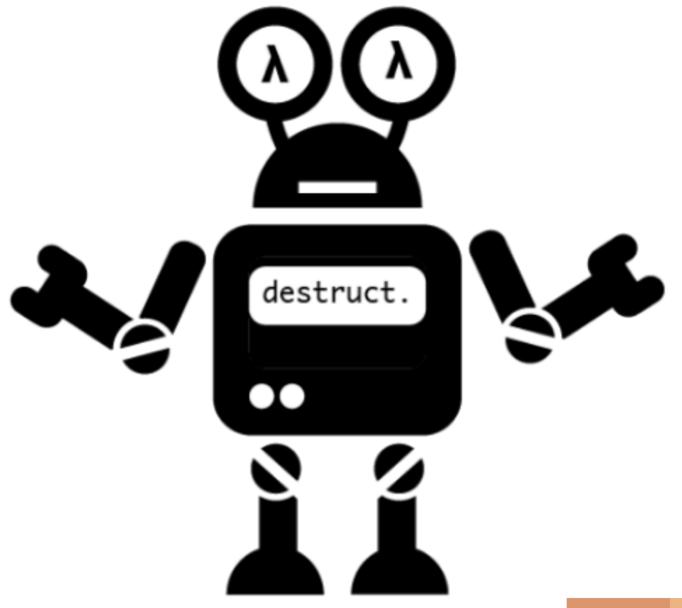


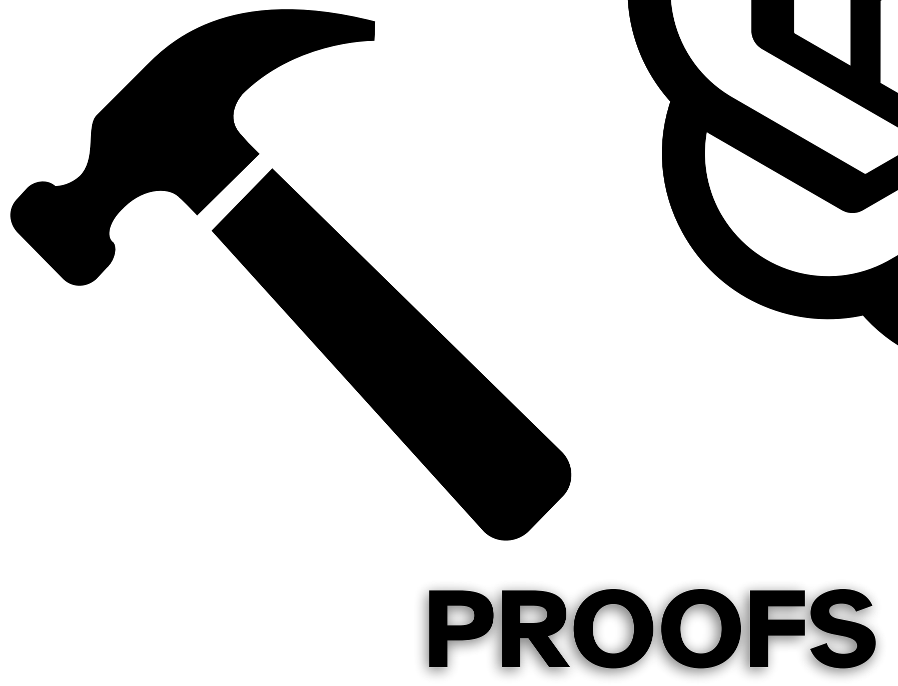
Xavier Leroy. Formal certification of a compiler back-end or: Programming a compiler with a proof assistant.

let's use automation to reduce work!

program + spec → proof







Let's verify a program!

```
Definition swap (m n : nat) : decorated :=
  <{
    {{ X = m /\ Y = n }} ->>
      {{ (X + Y) - ((X + Y) - Y) = n /\ (X + Y) - Y = m }}
    X := X + Y
      {{ X - (X - Y) = n /\ X - Y = m }};
    Y := X - Y
      {{ X - Y = n /\ Y = m }};
    X := X - Y
      {{ X = n /\ Y = m }}
  }>.

Theorem swap_valid : forall m n, outer_triple_valid (swap m n).
Proof.
```

Let's verify a program!

```
Definition swap (m n : nat) : decorated :=
  <{
    {{ X = m /\ Y = n }} ->>
      {{ (X + Y) - ((X + Y) - Y) = n /\ (X + Y) - Y = m }}
    X := X + Y
      {{ X - (X - Y) = n /\ X - Y = m }};
    Y := X - Y
      {{ X - Y = n /\ Y = m }};
    X := X - Y
      {{ X = n /\ Y = m }}
  }>.
```

```
Theorem swap_valid : forall m n, outer_triple_valid (swap m n).
Proof.
  intros m n.
  unfold outer_triple_valid. simpl.
  eapply hoare_seq.
  - eapply hoare_seq.
    + apply hoare_asgn.
    + apply hoare_asgn.
  - eapply hoare_consequence_pre.
    + apply hoare_asgn.
    + unfold "->>", assertion_sub, t_update, bassertion.
      intros. simpl in *.
      destruct H.
      rewrite H. rewrite H0. split.
      * admit.
      * admit.
Admitted.
```

What kinds of mental tasks do you do when writing a proof?

```
Definition swap (m n : nat) : decorated :=
  <{
    {{ X = m /\ Y = n }} ->>
      {{ (X + Y) - ((X + Y) - Y) = n /\ (X + Y) - Y = m }}
    X := X + Y
      {{ X - (X - Y) = n /\ X - Y = m }};
    Y := X - Y
      {{ X - Y = n /\ Y = m }};
    X := X - Y
      {{ X = n /\ Y = m }}
  }>.
```

```
Theorem swap_valid : forall m n, outer_triple_valid (swap m n).
Proof.
  intros m n.
  unfold outer_triple_valid. simpl.
  eapply hoare_seq.
  - eapply hoare_seq.
    + apply hoare_asgn.
    + apply hoare_asgn.
  - eapply hoare_consequence_pre.
    + apply hoare_asgn.
    + unfold "->>", assertion_sub, t_update, bassertion.
      intros. simpl in *.
      destruct H.
      rewrite H. rewrite H0. split.
      * admit.
      * admit.
Admitted.
```

Subproblems

premise selection - picking out useful lemmas we could apply

tactic prediction - identifying which tactics to try

proof search - searching for different proof states that get us closer to Qed

Built in tactics

auto. lia.

```
Definition swap (m n : nat) : decorated :=
  <{
    {{ X = m /\ Y = n }} ->>
      {{ (X + Y) - ((X + Y) - Y) = n /\ (X + Y) - Y = m }}
    X := X + Y
      {{ X - (X - Y) = n /\ X - Y = m }};
    Y := X - Y
      {{ X - Y = n /\ Y = m }};
    X := X - Y
      {{ X = n /\ Y = m }}
  }>.
```

```
Theorem swap_valid : forall m n, outer_triple_valid (swap m n).
Proof.
  intros m n.
  unfold outer_triple_valid. simpl.
  eapply hoare_seq.
  - eapply hoare_seq.
    + apply hoare_asgn.
    + apply hoare_asgn.
  - eapply hoare_consequence_pre.
    + apply hoare_asgn.
    + unfold "->>", assertion_sub, t_update, bassertion.
      intros. simpl in *.
      lia.
Qed.
```

auto/lia's approach

premise selection: no premises, or manually provided

tactic prediction: hard coded

auto - reflexivity, assumption, apply

lia - linear positivstellensatz refutations, cutting plane proofs, case split

search procedure: decision procedure

Domain-specific Tactics

```
Definition swap (m n : nat) : decorated :=
  <{
    {{ X = m /\ Y = n }} ->>
      {{ (X + Y) - ((X + Y) - Y) = n /\ (X + Y) - Y = m }}
    X := X + Y
      {{ X - (X - Y) = n /\ X - Y = m }};
    Y := X - Y
      {{ X - Y = n /\ Y = m }};
    X := X - Y
      {{ X = n /\ Y = m }}
  }>.
```

```
Ltac assertion_auto :=
  try auto;  (* as in example 1, above *)
  try (unfold "->>", assertion_sub, t_update;
       intros; simpl in *; lia).

Theorem swap_valid : forall m n, outer_triple_valid (swap m n).
Proof.
  intros m n.
  unfold outer_triple_valid. simpl.
  eapply hoare_seq.
  - eapply hoare_seq.
    + apply hoare_asgn.
    + apply hoare_asgn.
  - eapply hoare_consequence_pre.
    + apply hoare_asgn.
    + assertion_auto.
Qed.
```

Domain-specific solvers

```
Ltac verify := intros; apply verification_correct; verify_assertion.
Ltac verify_assertion := …

Theorem swap_valid : forall m n, outer_triple_valid (swap m n).
Proof.
  verify.
Qed.
```

Approach of domain specific solvers

premise selection: hard coded

tactic prediction: hard coded

search procedure: hard coded

Domain-specific solvers

they require encoding domain knowledge

```
Ltac verify_assertion := repeat split;
  simpl;
  unfold assert_implies;
  unfold bassertion in *; unfold beval in *; unfold aeval in *;
  unfold assertion_sub; intros;
  repeat (simpl in *;
          rewrite t_update_eq ||
          (try rewrite t_update_neq;
          [| (intro X; inversion X; fail)]));
  simpl in *;
  repeat match goal with [H : _ ∧ _ ⊢ _] ⇒
                         destruct H end;
  repeat rewrite not_true_iff_false in *;
  repeat rewrite not_false_iff_true in *;
  repeat rewrite negb_true_iff in *;
  repeat rewrite negb_false_iff in *;
  repeat rewrite eqb_eq in *;
  repeat rewrite eqb_neq in *;
  repeat rewrite leb_iff in *;
  repeat rewrite leb_iff_conv in *;
  try subst;
  simpl in *;
  repeat
    match goal with
      [st : state ⊢ _] ⇒
        match goal with
        | [H : st _ = _ ⊢ _] ⇒
            rewrite → H in *; clear H
        | [H : _ = st _ ⊢ _] ⇒
            rewrite <- H in *; clear H
        end
    end;
  try eauto;
  try lia.
```

CoqHammer and SMT Solvers

(2018)

[1] Łukasz Czajka and Cezary Kaliszyk. 2018. Hammer for Coq: Automation for Dependent Type Theory.

What is an SMT Solver?

Satisfiability Modulo Theories

SAT: is there a boolean assignment that satisfies this equation?

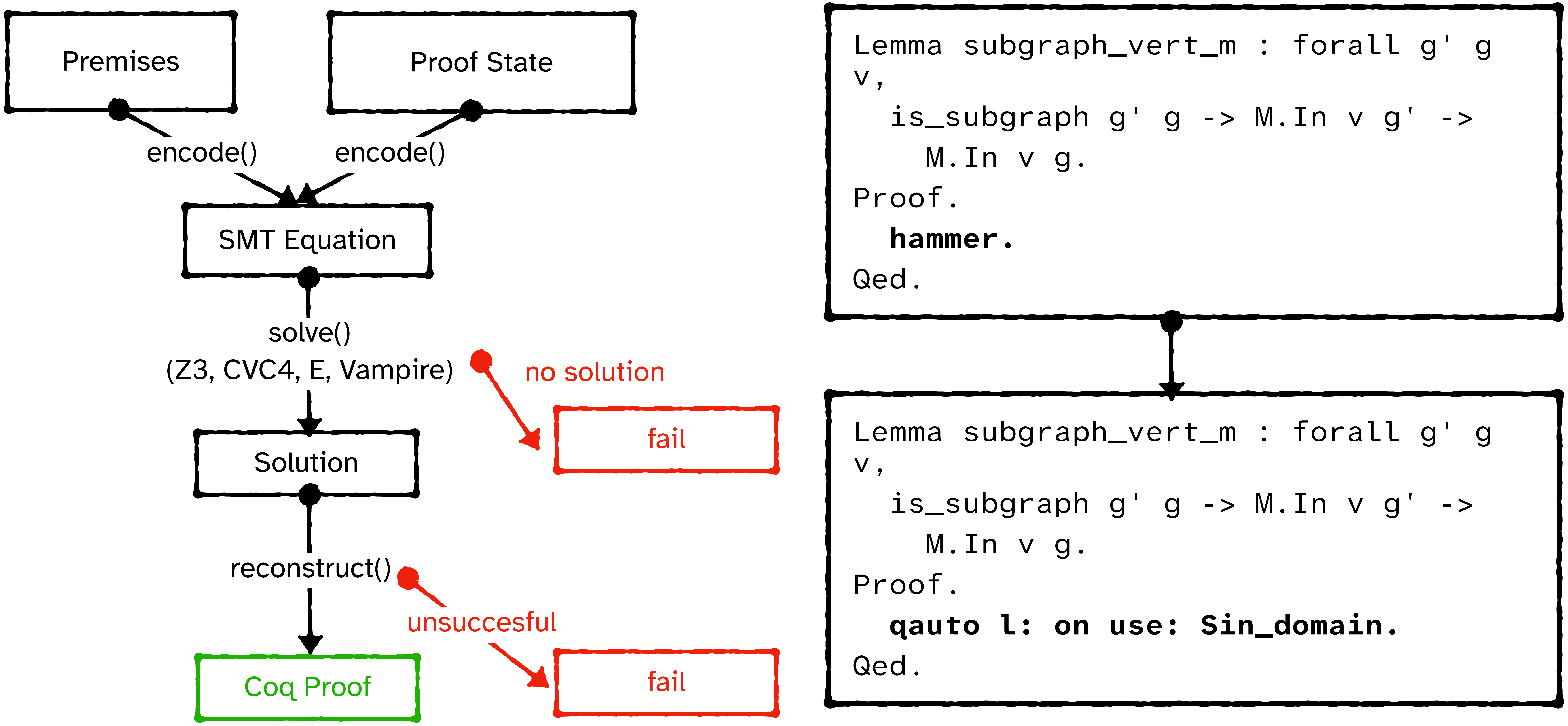
```
(serveGin \/ serveTonic) /\ (isMinor -> ~serveGin) /\ isMinor
isMinor: T; serveGin: F; serveTonic: T
```

SMT: is there an assignment within the theory that satisfies this equation?

```
(serveGin \/ serveTonic) /\ (age <= 21 -> abv = 0) /\ (age = 17)
/\ (serveGin => abv >= 40)
age: 17; abv: 0; serveGin: F; serveTonic: T
```

https://www.youtube.com/watch?v=rTOqg-f2rNM

CoqHammer


Premises
Proof State
encode()
encode()
SMT Equation
solve()
(Z3, CVC4, E, Vampire)
no solution
fail
Solution
reconstruct()
unsuccesful
fail
Coq Proof
Lemma subgraph_vert_m : forall g' g v,
is_subgraph g' g -> M.In v g' ->
M.In v g.
Proof.
hammer.
Qed.
Lemma subgraph_vert_m : forall g' g v,
is_subgraph g' g -> M.In v g' ->
M.In v g.
Proof.
qauto l: on use: Sin_domain.
Qed.

CoqHammer’s Approach

premise selection: k-nearest neighbours (k-NN)

tactic prediction: reconstruction tactics

search procedure: reconstruction tactics + SMT Solver

Performance

CoqGym - 68,501 theorems from 124 projects

proves 26.6% of theorems automatically!

CoqGym is a tough benchmark for AI tools

Proverbot9001 and Tactic-by-Tactic Search

(2020)

Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. 2020. Generating correctness proofs with neural networks.


Goal
(forall ...)
Hypothesis
(eq ...)
Last Tactic
(apply ...)
Tactic Predictor
unfold eq.
apply IHn.
intros.

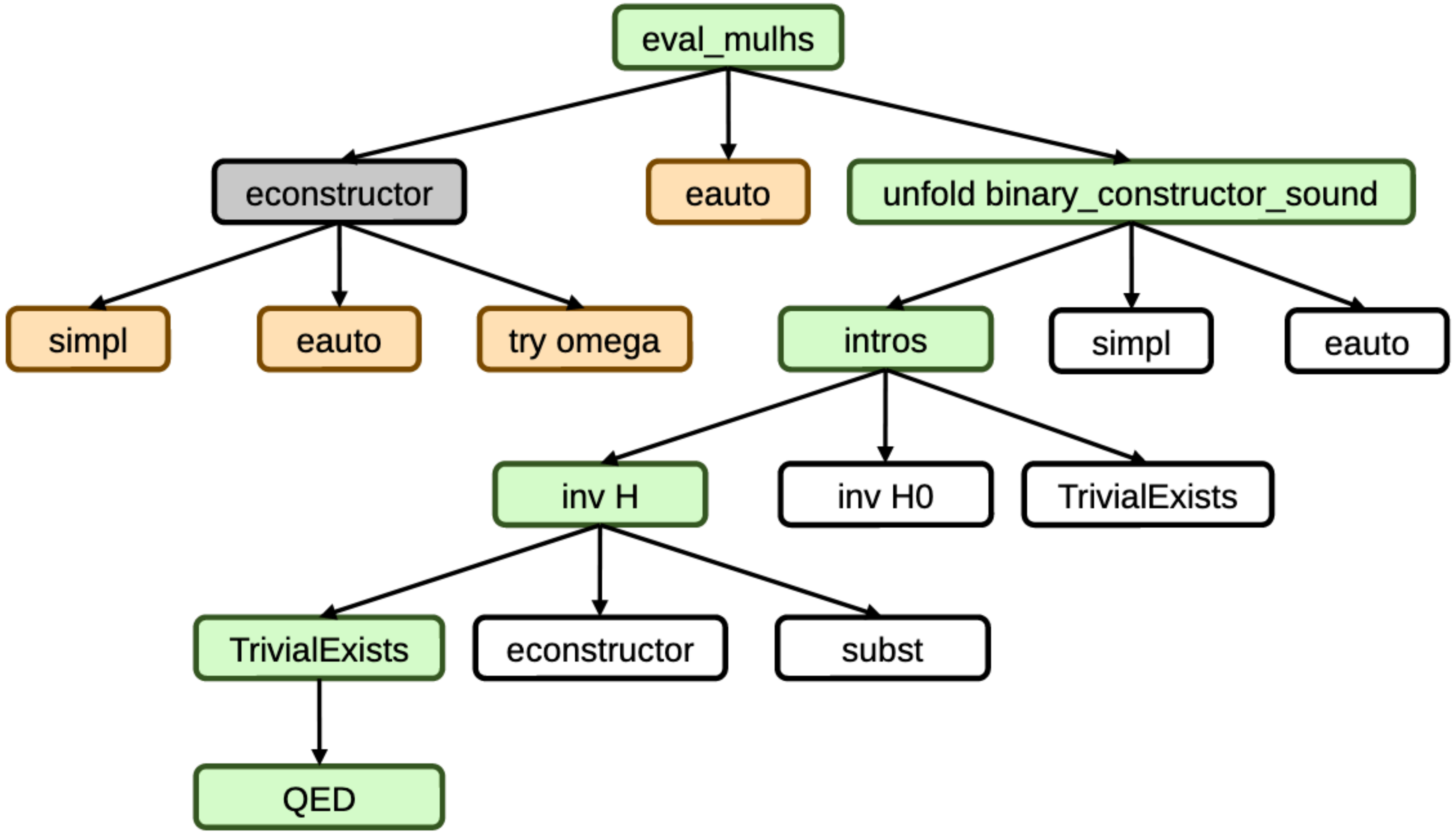
Tactic-by-Tactic Search

```
Definition binary_constructor_sound
    (constructor: expr -> expr -> expr)
    (semantics: val -> val -> val) : Prop := …

Theorem eval_mulhs:
  binary_constructor_sound mulhs Val.mulhs.
Proof.
```

Tactic-by-Tactic Search

```
Theorem eval_mulhs:
  binary_constructor_sound mulhs Val.mulhs.
Proof.
```



Proverbot Architecture

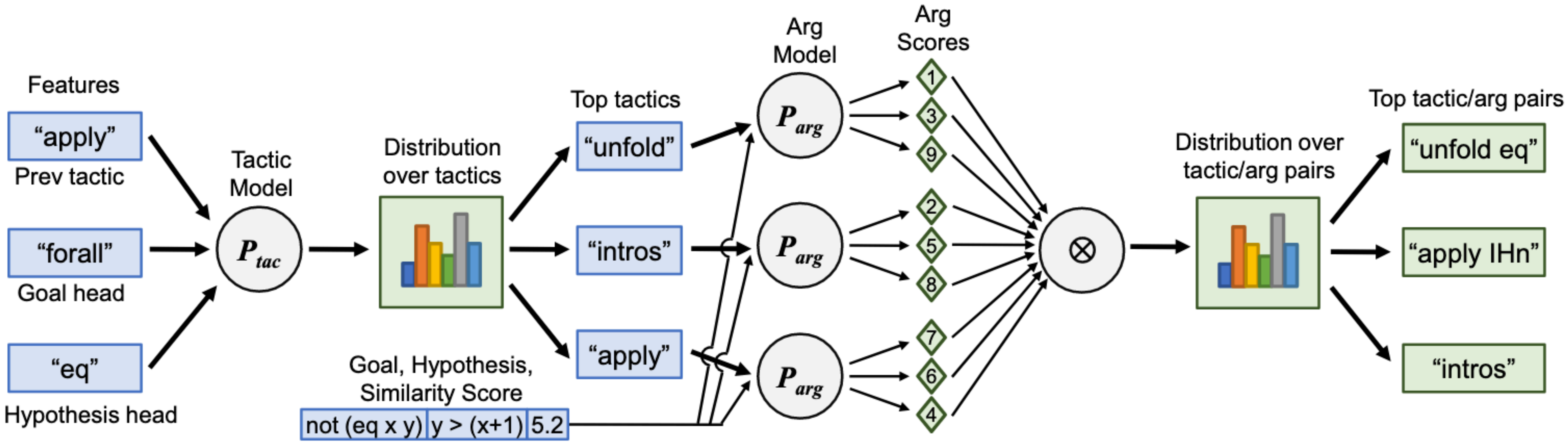
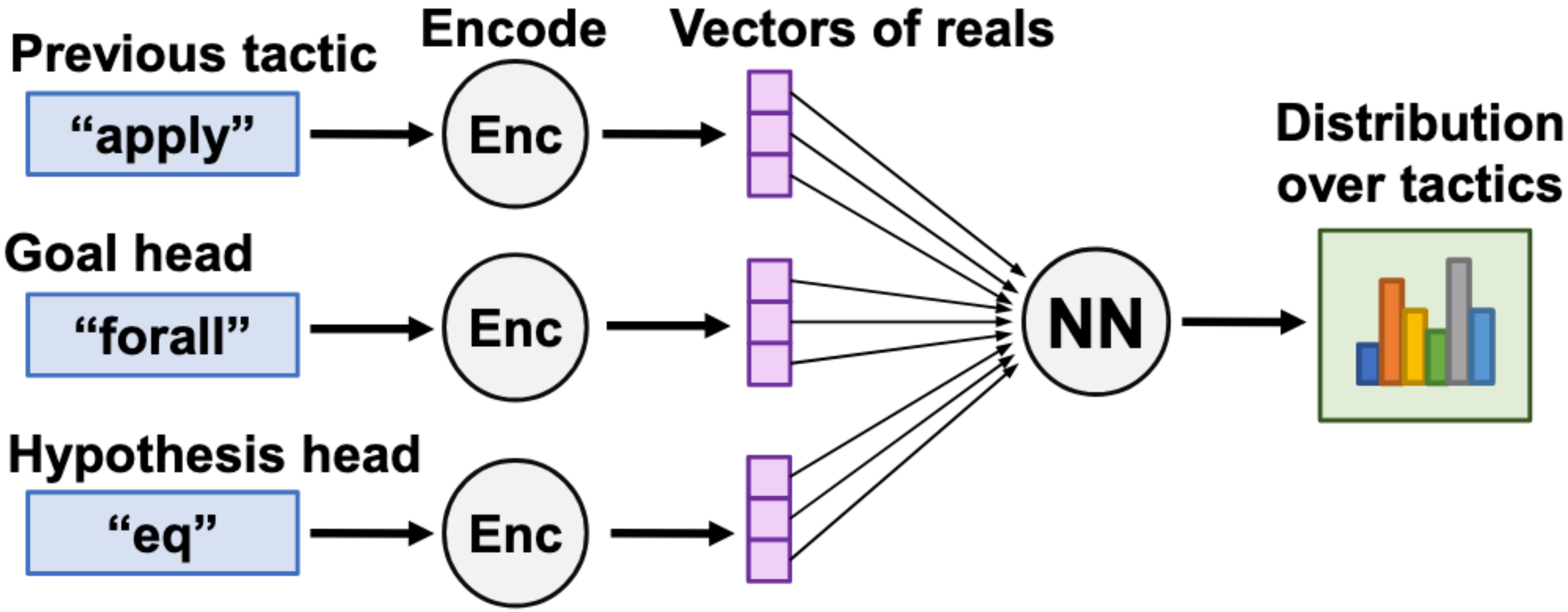


Figure 8. The overall prediction model, combining the tactic prediction and argument prediction models.

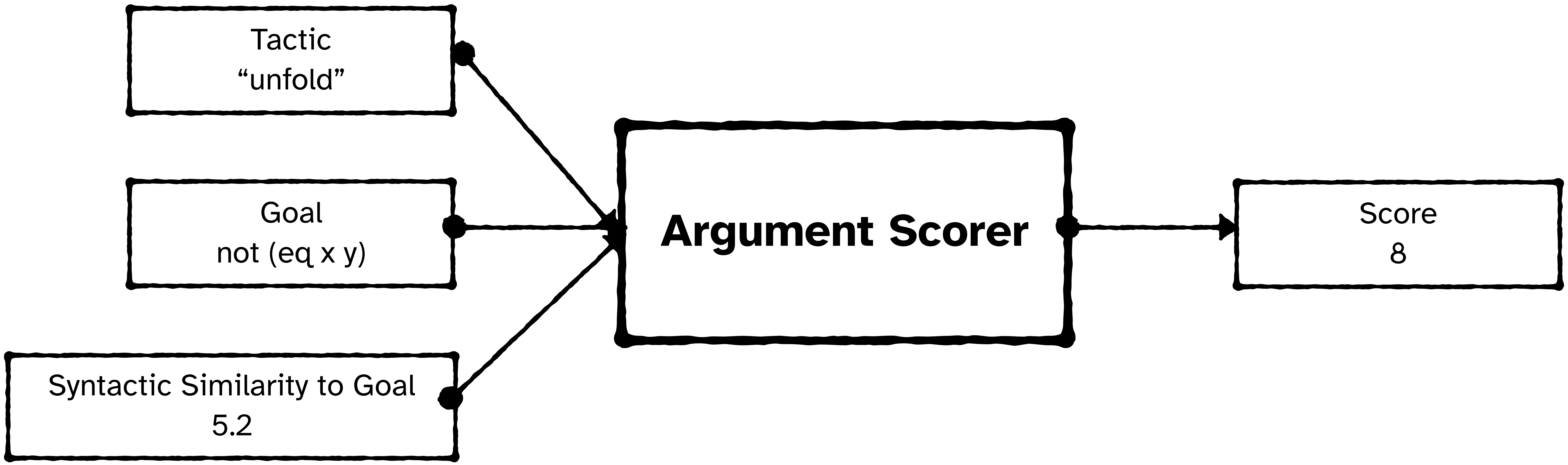
Predicting the next tactic

what are the most likely tactics to come next?



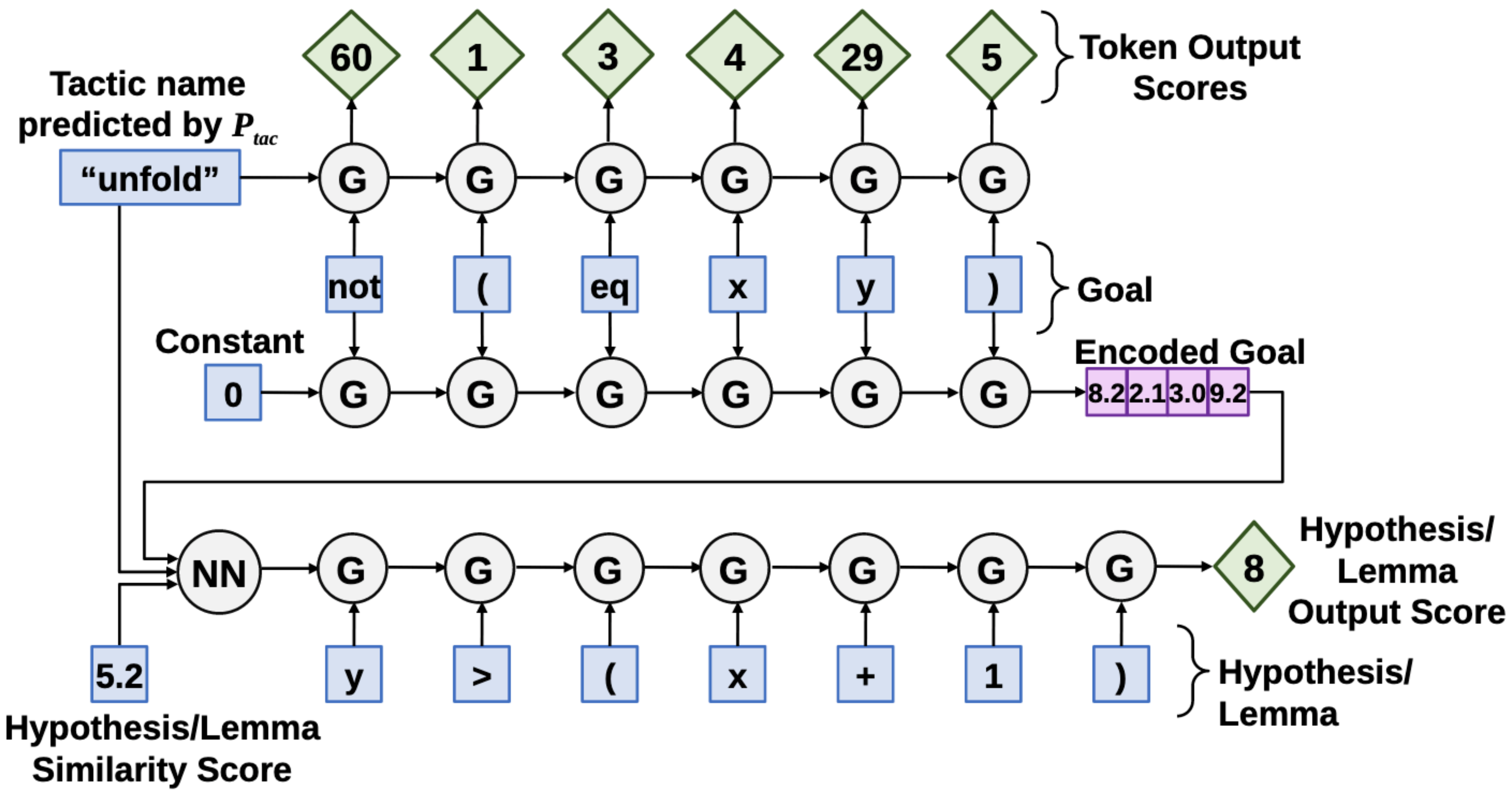
Scoring arguments

How useful is each argument for a specific tactic?



Scoring arguments

How useful is each argument for a specific tactic?



Proverbot9001's Approach

premise selection: preceding lemmas in the same file

tactic prediction: RNN-based architecture

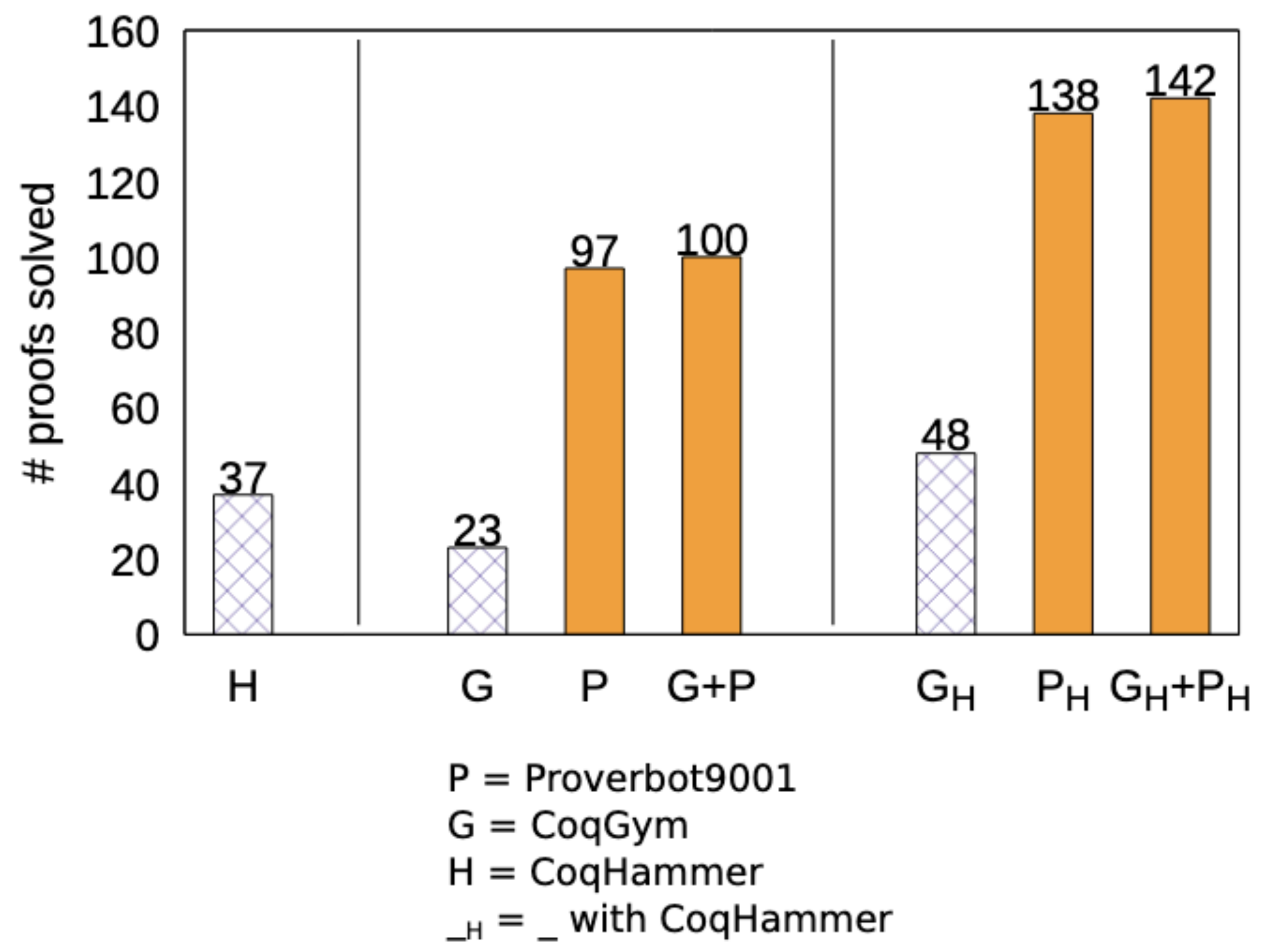
search procedure: tactic-by-tactic search (beam search)

Performance

proves 17% of CoqGym theorems automatically

proves theorems that CoqHammer could not

using both together produces even better results



Baldur and Whole Proof Synthesis

(2023)

Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. 2023. Baldur: Whole-Proof Generation and Repair with Large Language Models.

Claude
ANTHROP\C

GitHub Copilot

What about LLMs?

OpenAI
GPT - 4

Gemini

Minerva: Solving Quantitative Reasoning Problems with Language Models

Google Research

- Mostly trained on natural language
- Further training on a large mathematics corpus

Question: Assume that the variance of the first n natural numbers is 10, and the variance of the first m even natural numbers is 16. Compute $m + n$.

Model output: Let σ^2 be the variance of the first m even natural numbers, and τ^2 be the variance of the first n natural numbers. Then, we have $\tau^2 = \frac{n^2-1}{12}$ and $\sigma^2 = \frac{m^2-1}{3}$. We solve the system of equations:

$$\frac{n^2 - 1}{12} = 10$$
$$\frac{m^2 - 1}{3} = 16.$$

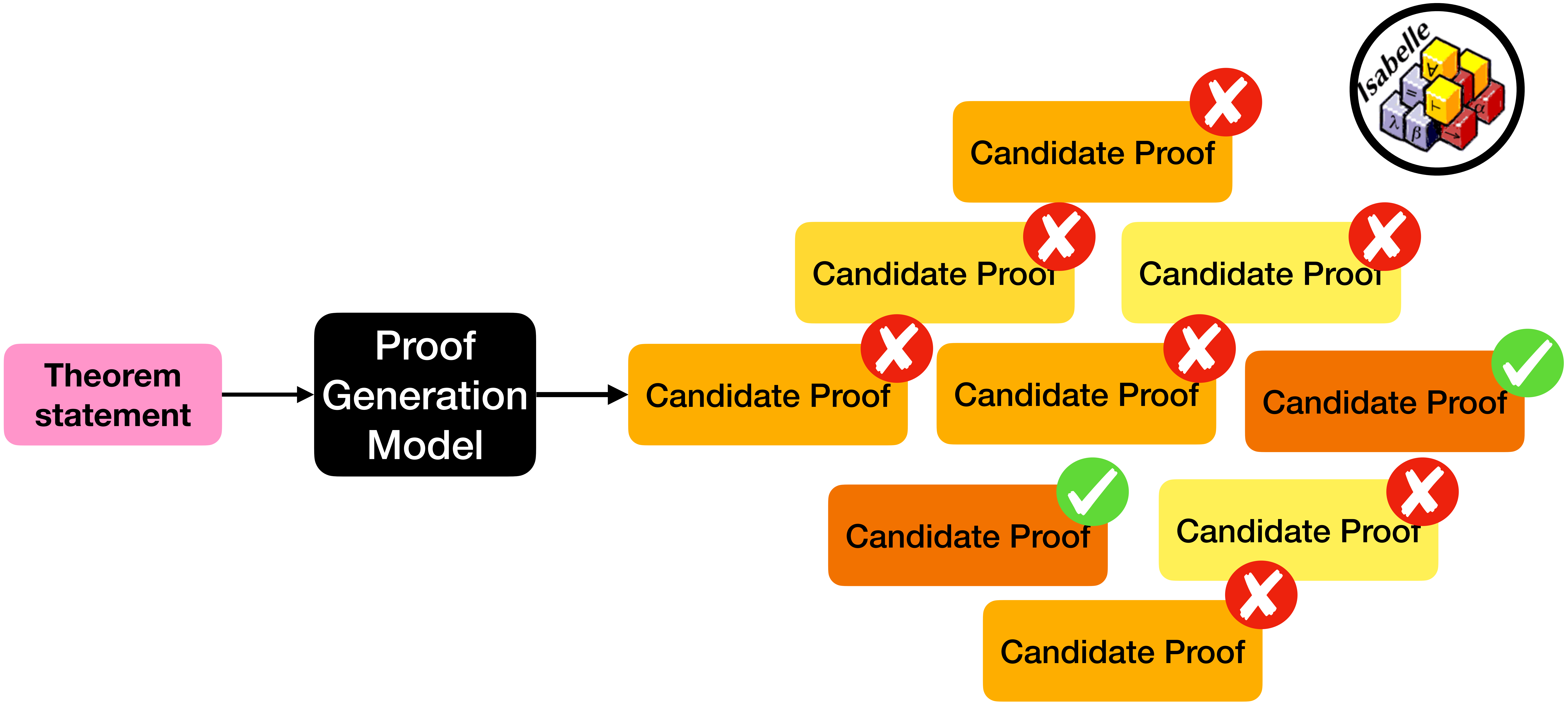
This translates into the system

$$n^2 = 121$$
$$m^2 = 48 + 1 = 49.$$

Therefore, $n = \boxed{11}$ and $m = \boxed{7}$, so $n + m = \boxed{18}$.

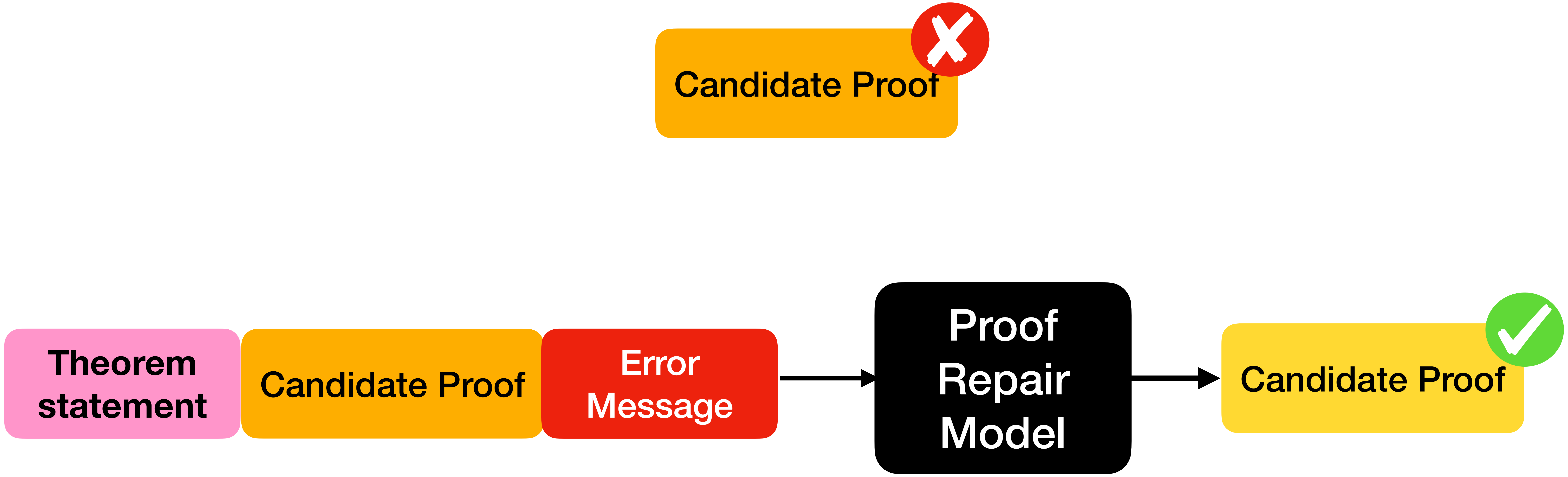
Chowdhery et al. (2022) “PaLM: Scaling Language Modeling with Pathways”
Lewkowycz et al. (2022) “Solving Quantitative Reasoning Problems with Language Models”

Baldur: Proof Generation



Temperature Sampling
Each sample = independent proof attempt

Baldur: Proof Repair



Baldur: Training Example Creation

Proof Generation training example

Source: Theorem statement → Proof Generation Model → Candidate Proof

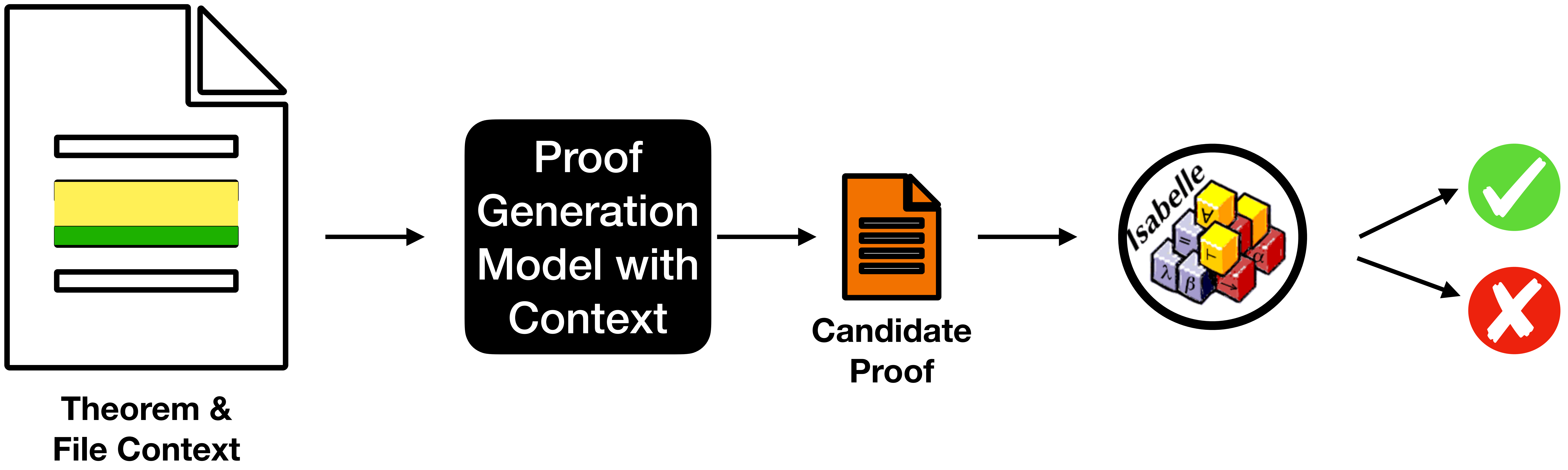
Target: Human-written Proof

Proof Repair training example

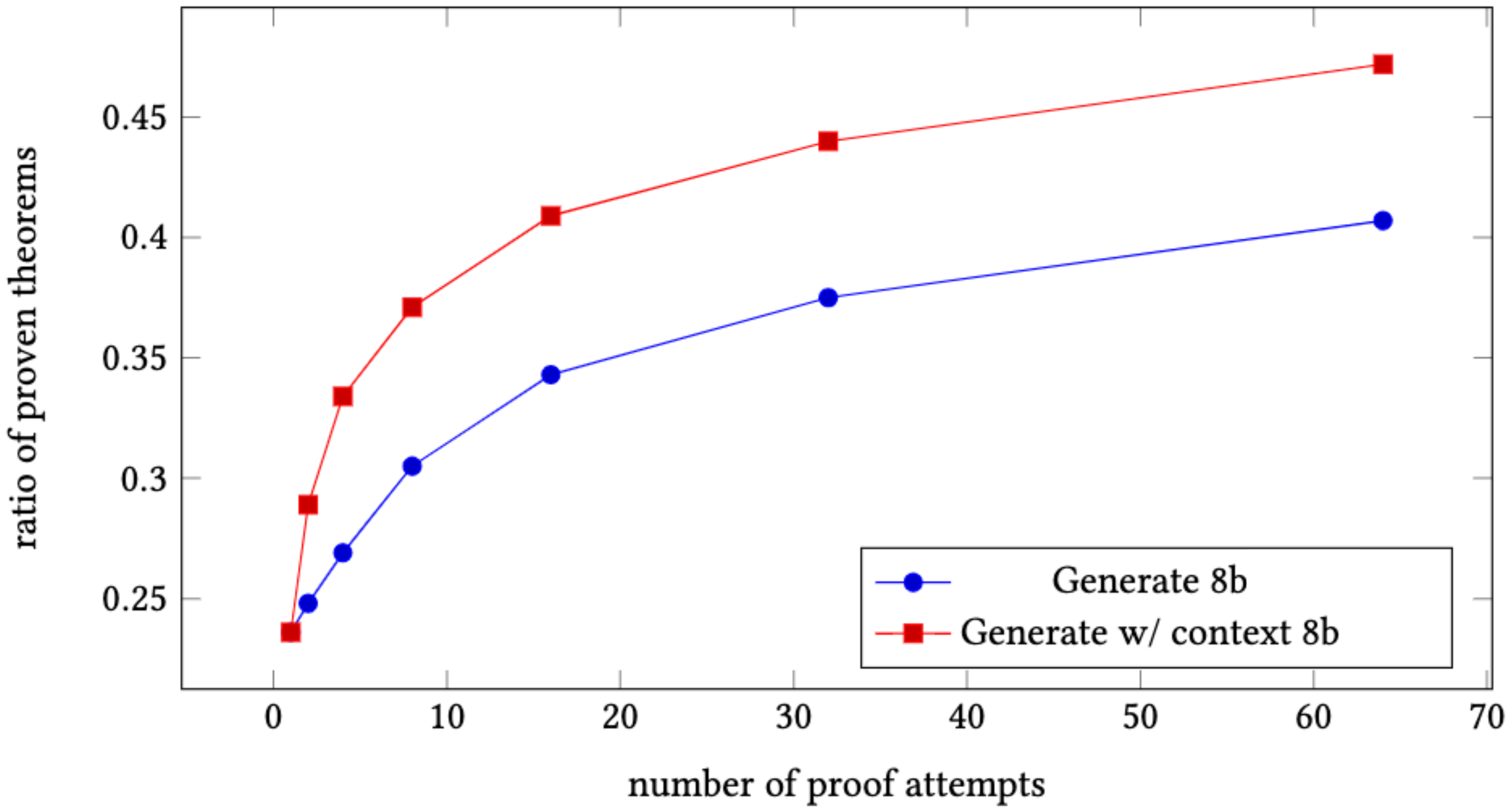
Source: Theorem statement, Incorrect Proof, Error Message

Target: Human-written Proof

Proof Generation with context

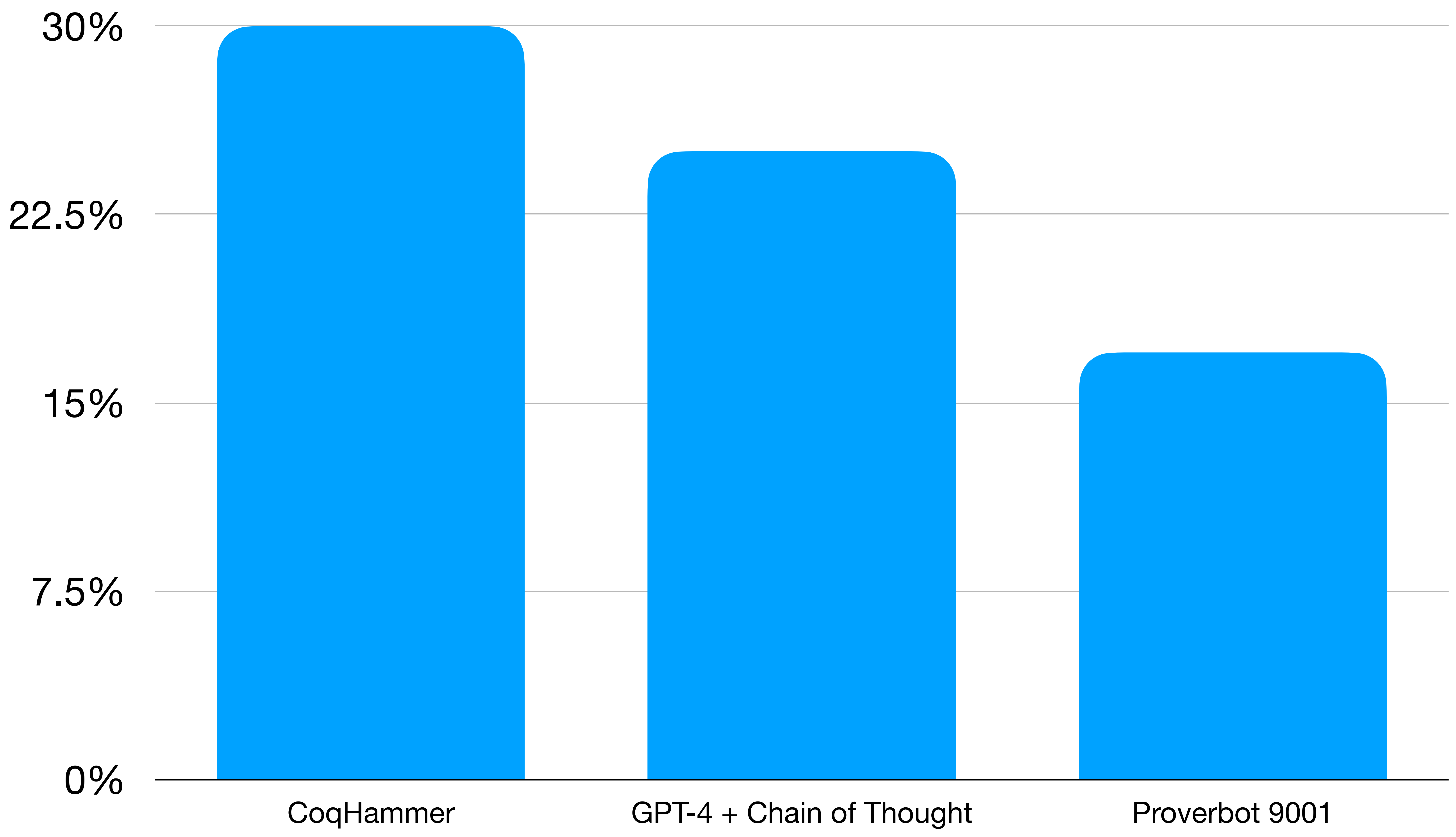


Generate with context



Proof context helps improve proof generation

LLM Performance on CoqGym



Baldur's Approach

~~*premise*~~ *context selection*: preceding lines in the same file

tactic prediction: fine-tuned LLM

search procedure: whole-proof search

Rango and Retrieval Augmentation

(2024)

Kyle Thompson et. al. 2024. Rango: Adaptive retrieval-augmented proving for automated software verification.

Our Contribution

Motivating Example

```
Theorem foo_idemp :
  forall x, 2 < x → foo x = x.
Proof.
  rewrite foo_helper.
  apply baz_idemp.
  lia.
Qed.
```

Motivating Example

```
Theorem foo_idemp :
  forall x, 2 < x → foo x = x.
Proof.
  rewrite foo_helper.
  apply baz_idemp.
  lia.
Qed.
```

```
Theorem bar_idemp :
  forall x, 2 < x → bar x = x.
Proof.
  ???
```

Motivating Example

```
Theorem foo_idemp :
  forall x, 2 < x → foo x = x.
Proof.
  rewrite foo_helper.
  apply baz_idemp.
  lia.
Qed.
```

```
Theorem bar_idemp :
  forall x, 2 < x → bar x = x.
Proof.
  rewrite bar_helper.
  apply baz_idemp.
  lia.
Qed.
```

System Components

Retrieval Augmentation + Fine-tuning + Proof Search

How do we retrieve Lemmas?

We syntactically compare the **proof state** to each **lemma declaration**

Current Proof State

```
m n p : nat
H1 : n < m
H2 : m < p
⊢ n < p
```

Available Lemmas

```
Lemma add_comm : ∀ n m : nat,
  n + m = n + n
```

✗

```
Lemma lt_trans : ∀ n m p : nat,
  n < m → m < p → n < p
```

✓

How do we retrieve Proofs?

We syntactically compare the **proof state** to each **prior proof state**

Current Proof State

```
m n p : nat
H1 : n < m
H2 : m < p
⊢ n < p
```

Prior Proof States

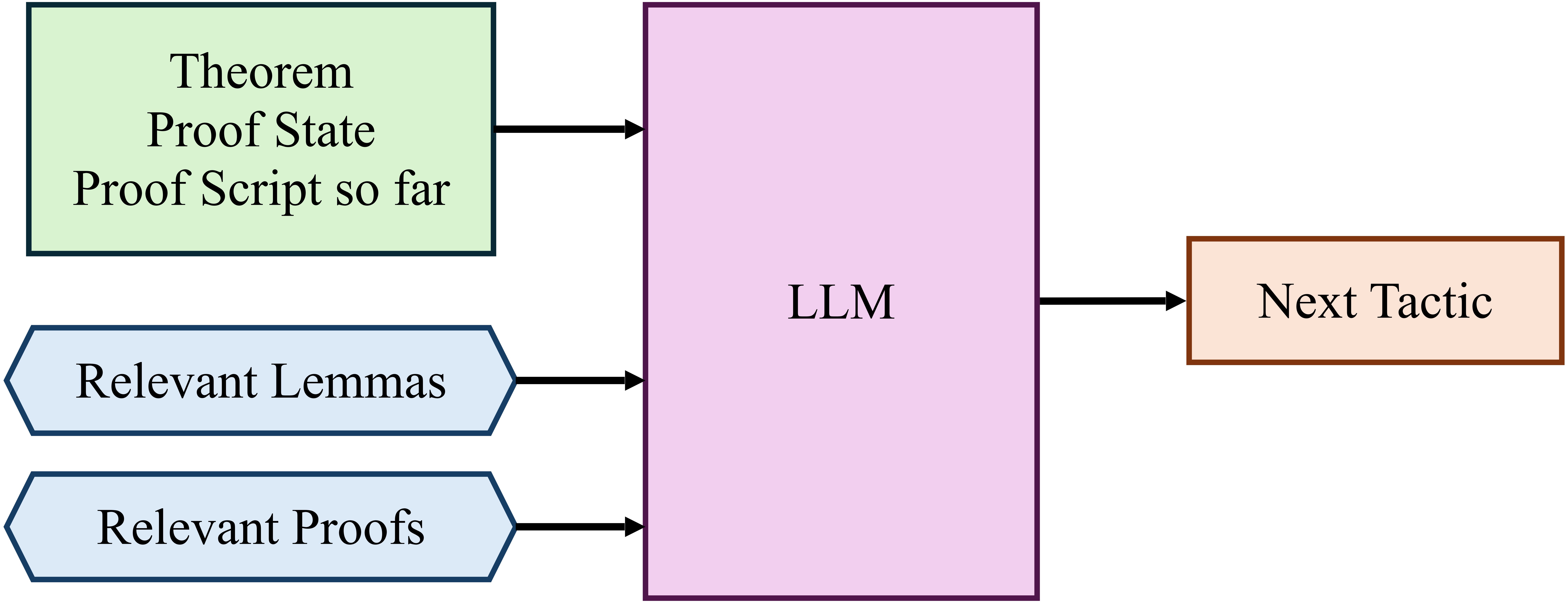
```
m n : nat
H1 : n + m = 0
⊢ n = 0
```

✗

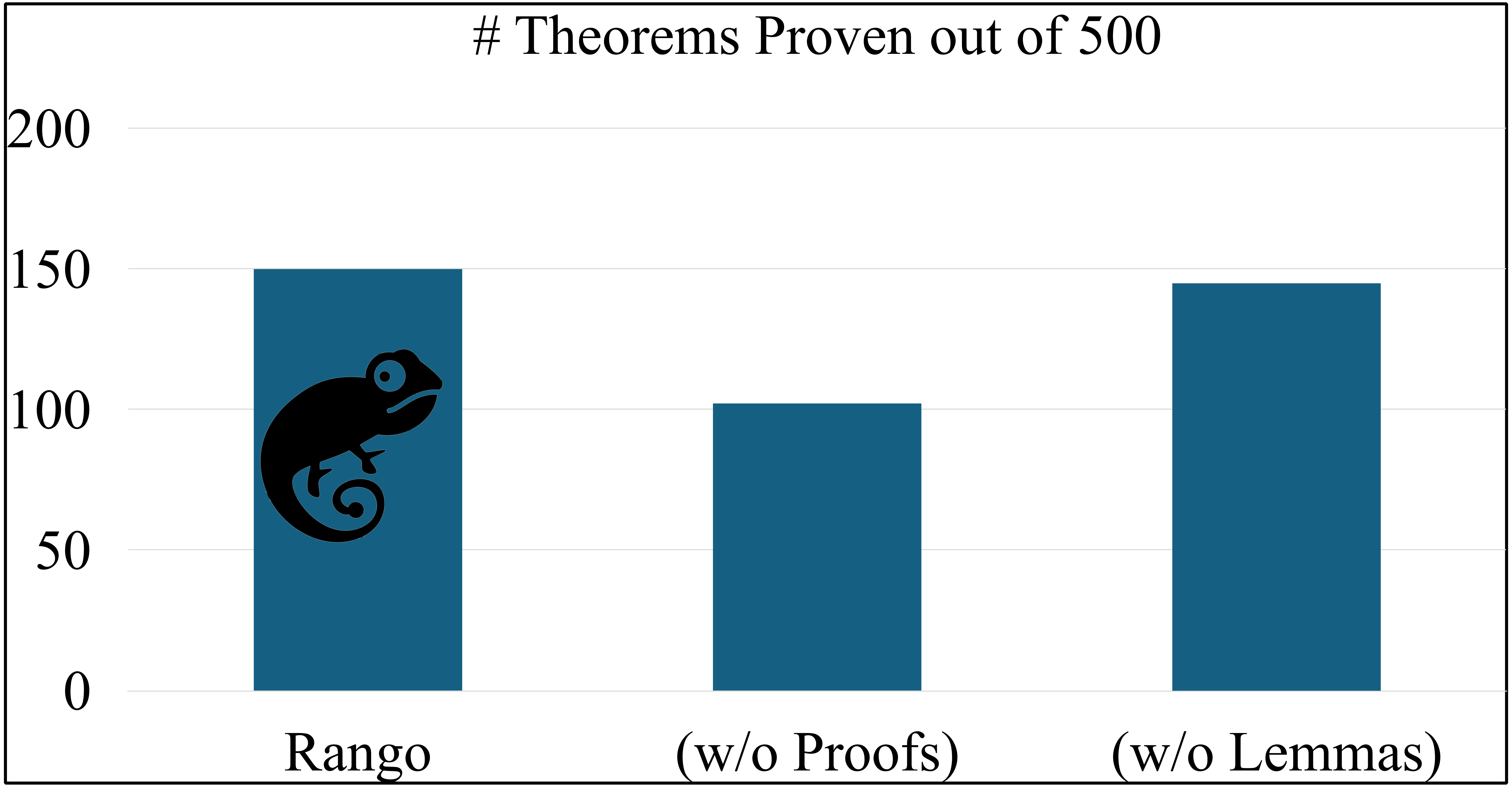
```
m n p : nat
H1 : n < m
H2 : 0 < n
⊢ 0 < m
```

✓

How Can We Make the LLM good at Rocq?

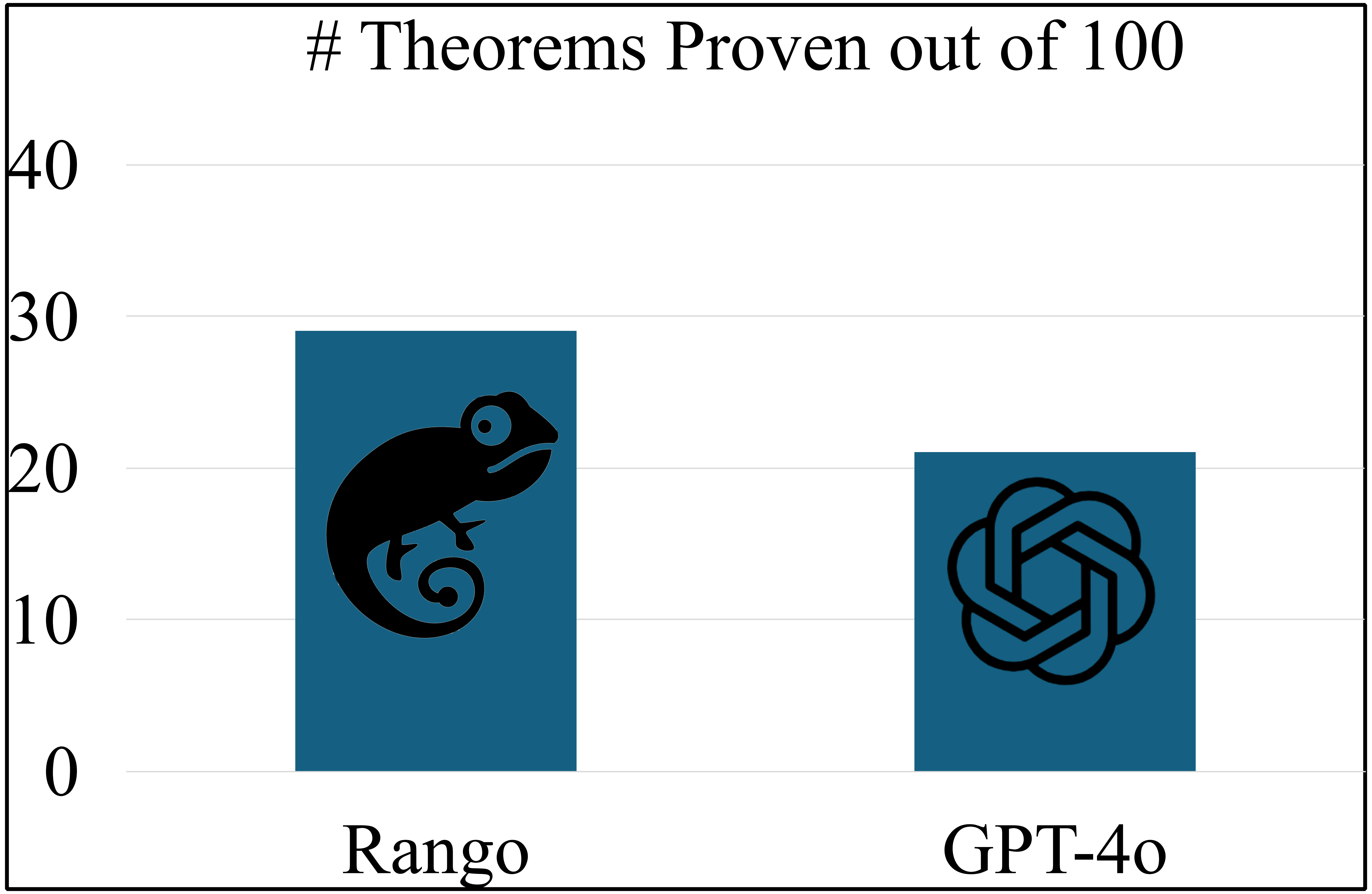


Rango Benefits Most from Similar Proofs

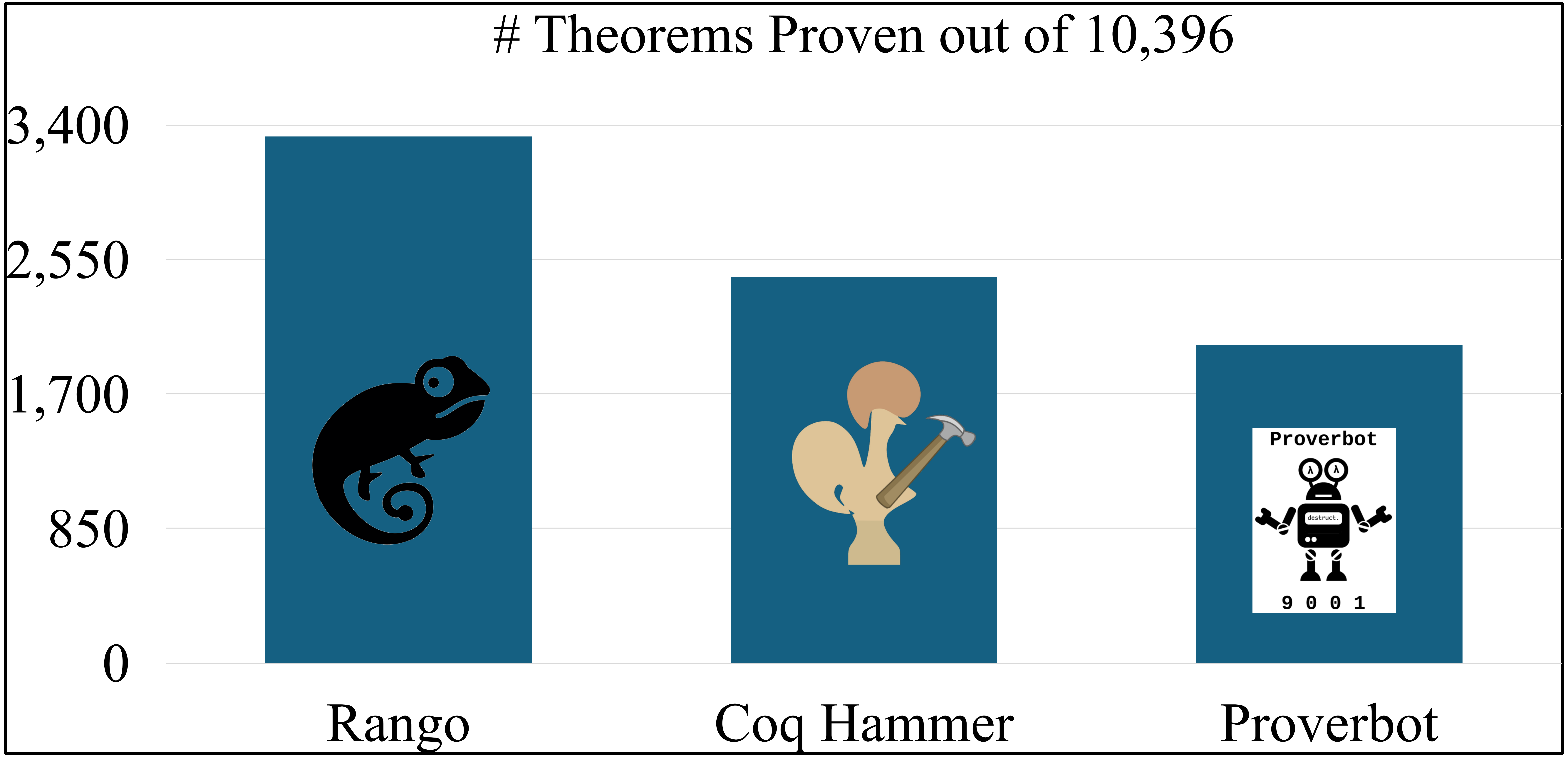


Rango Outperforms GPT-4o

At 1/400th the size!



Rango Outperforms Prior Tools



There's a ton more work in this space!

Deepseek Prover 1.5 - LLMs + Reinforcement Learning and Monte Carlo Tree Search

Cobblestone - isolates failures and recursively reprompts the LLM

LEGO-prover - maintains a growing library of helper lemmas

Saketh Kasibatla et. al. Cobblestone: A Divide-and-Conquer Approach for Automating Formal Verification.

Haiming Wang et. al. 2023. LEGO-Prover: Neural Theorem Proving with Growing Libraries. October 27, 2023.

Huajian Xin et. al. 2024. DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search.

But theorem proving is far from solved

Can we build usable tools to help people prove theorems more easily?

Can we also help humans come up with specs?

Thanks! 👍🏾

skasibatla@ucsd.edu