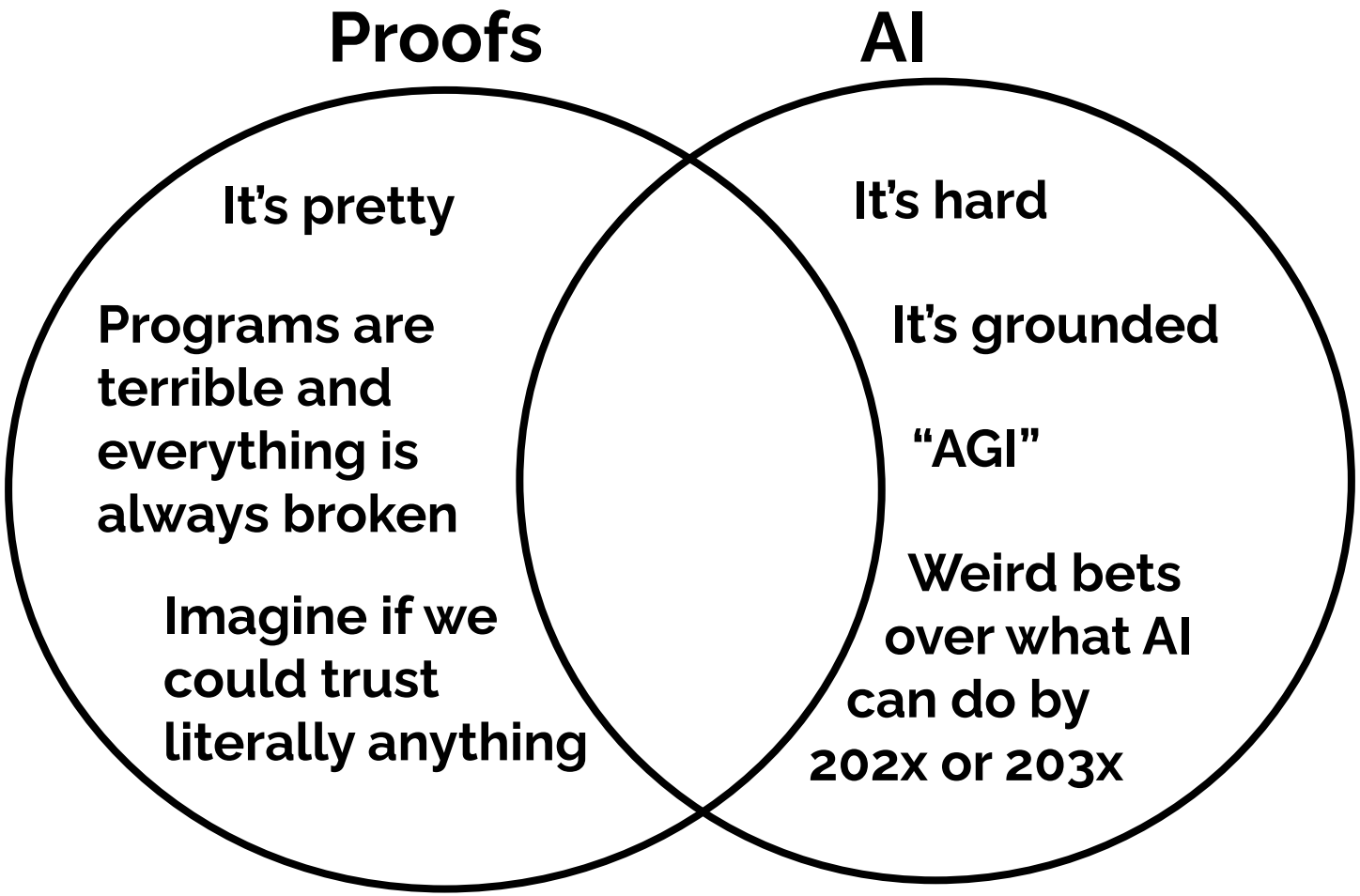
Concrete Problems in Proof Automation

Talia Ringer
UIUC (currently also visiting at Google)

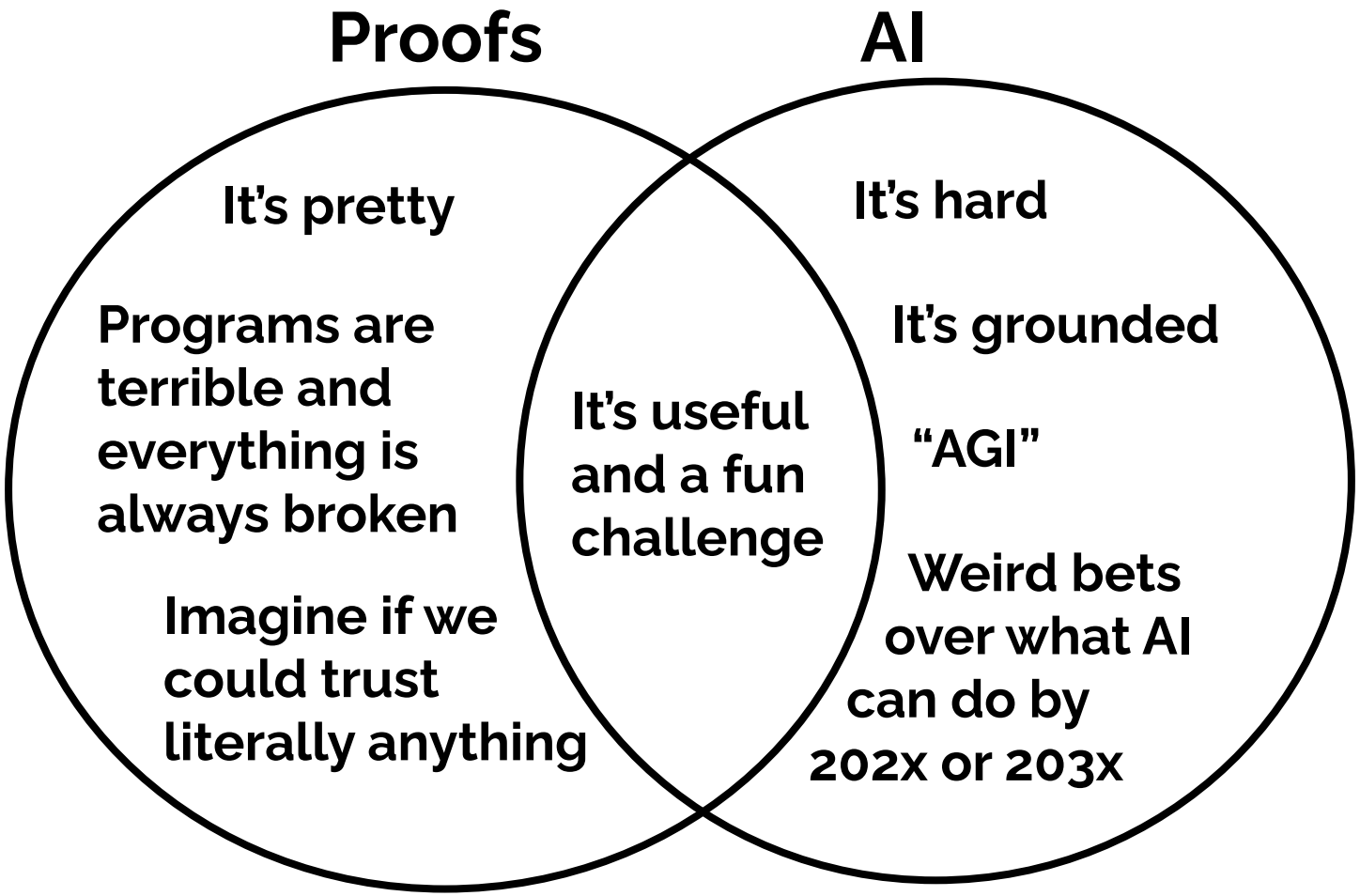
The long version of this talk from AITP 2022 is on my YouTube!

AI for Proofs Renaissance

AI for Proofs



AI for Proofs



Most AI for proofs:

1. tactic prediction,
2. synthesis,
3. autoformalization,
4. premise selection, and
5. concept alignment.

Don't stop doing:

1. tactic prediction,
2. synthesis,
3. autoformalization,
4. premise selection, and
5. concept alignment.

But …

Do lots more:

1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

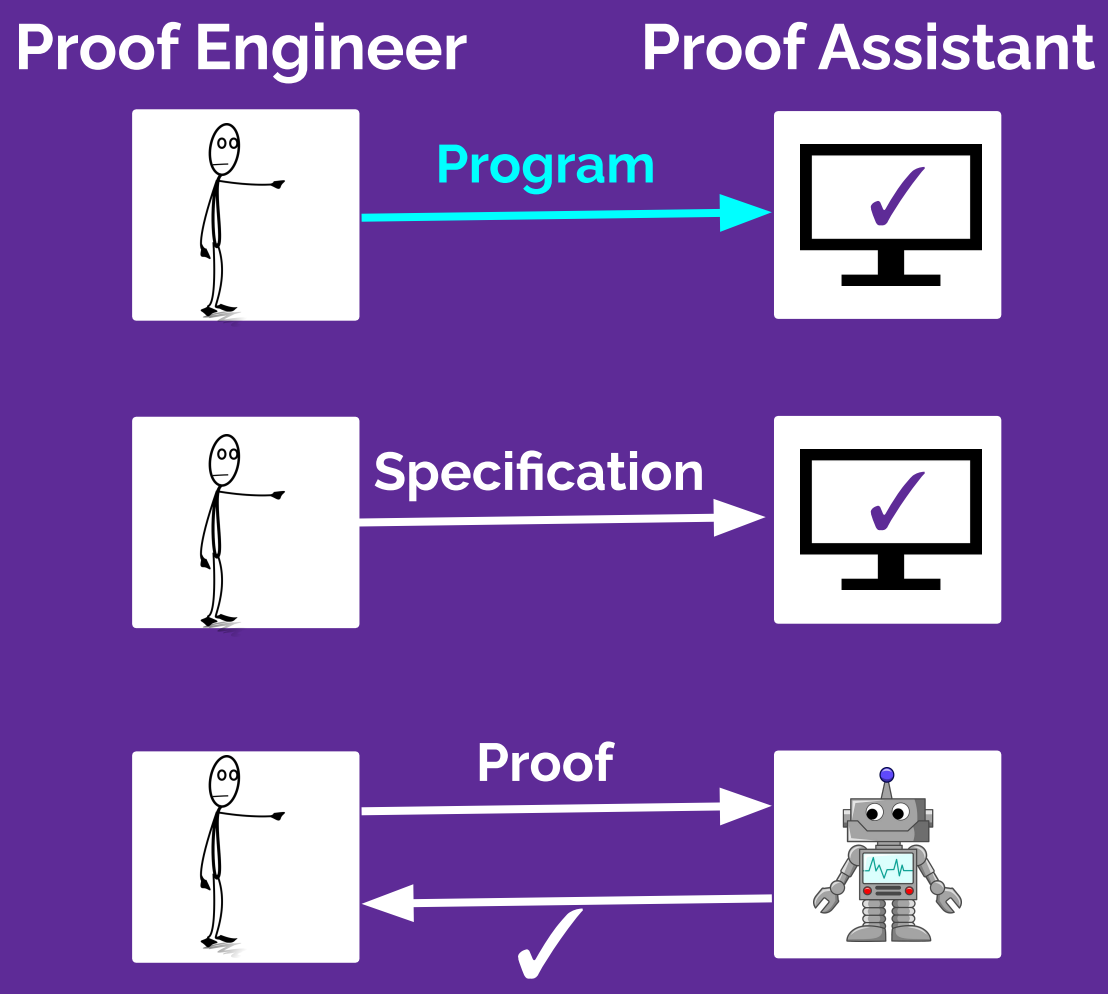
You won't regret it!

Do lots more:

1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

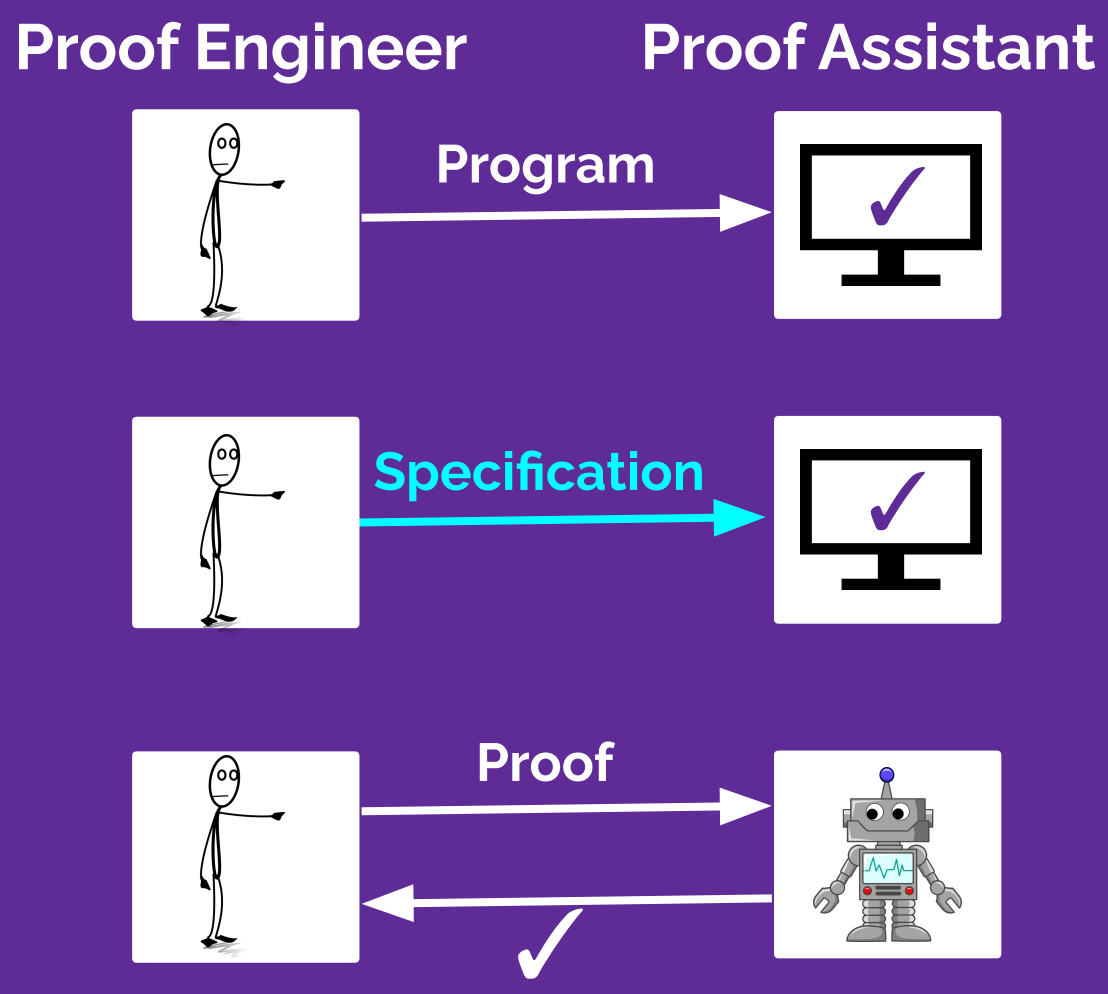
You won't regret it!

Myth



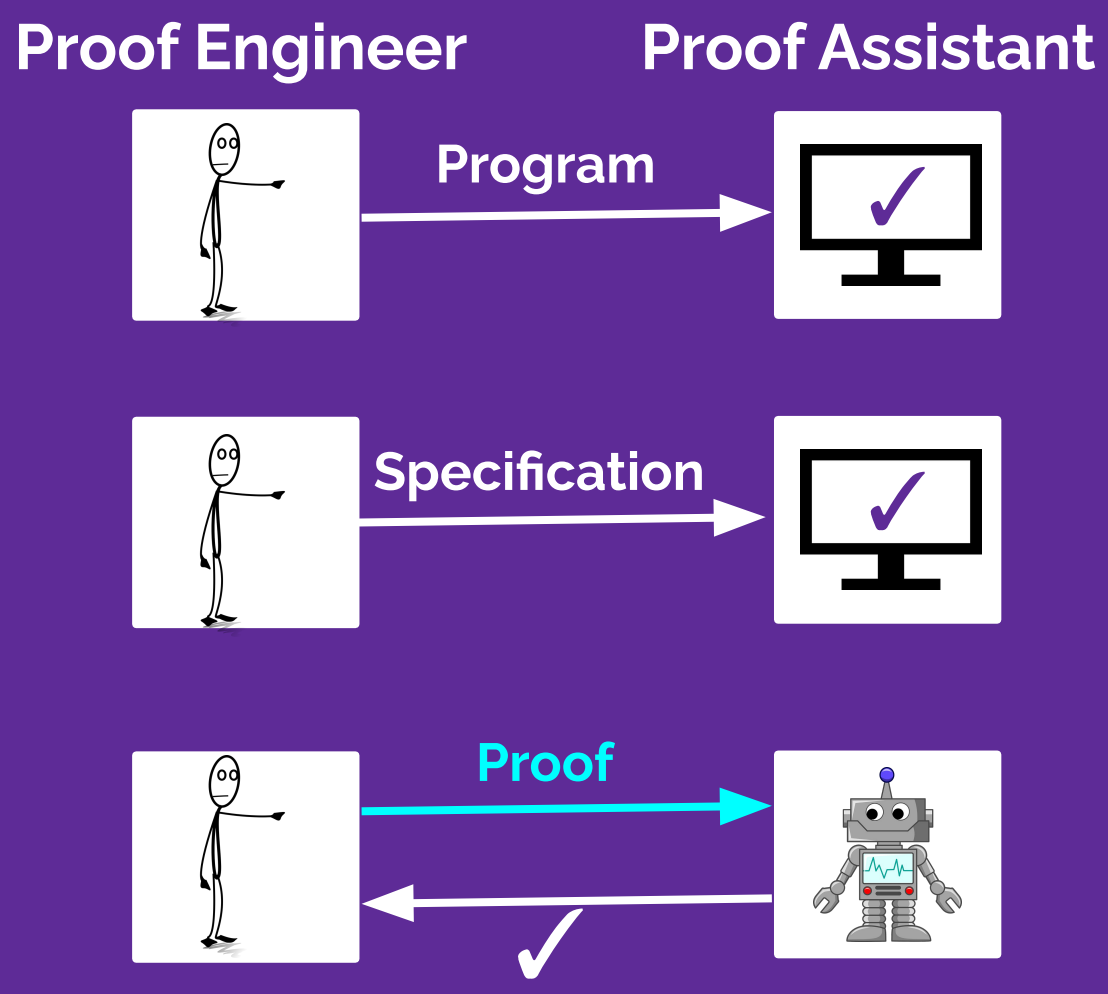
Conjecture Testing (Task 1 of 5)

Myth



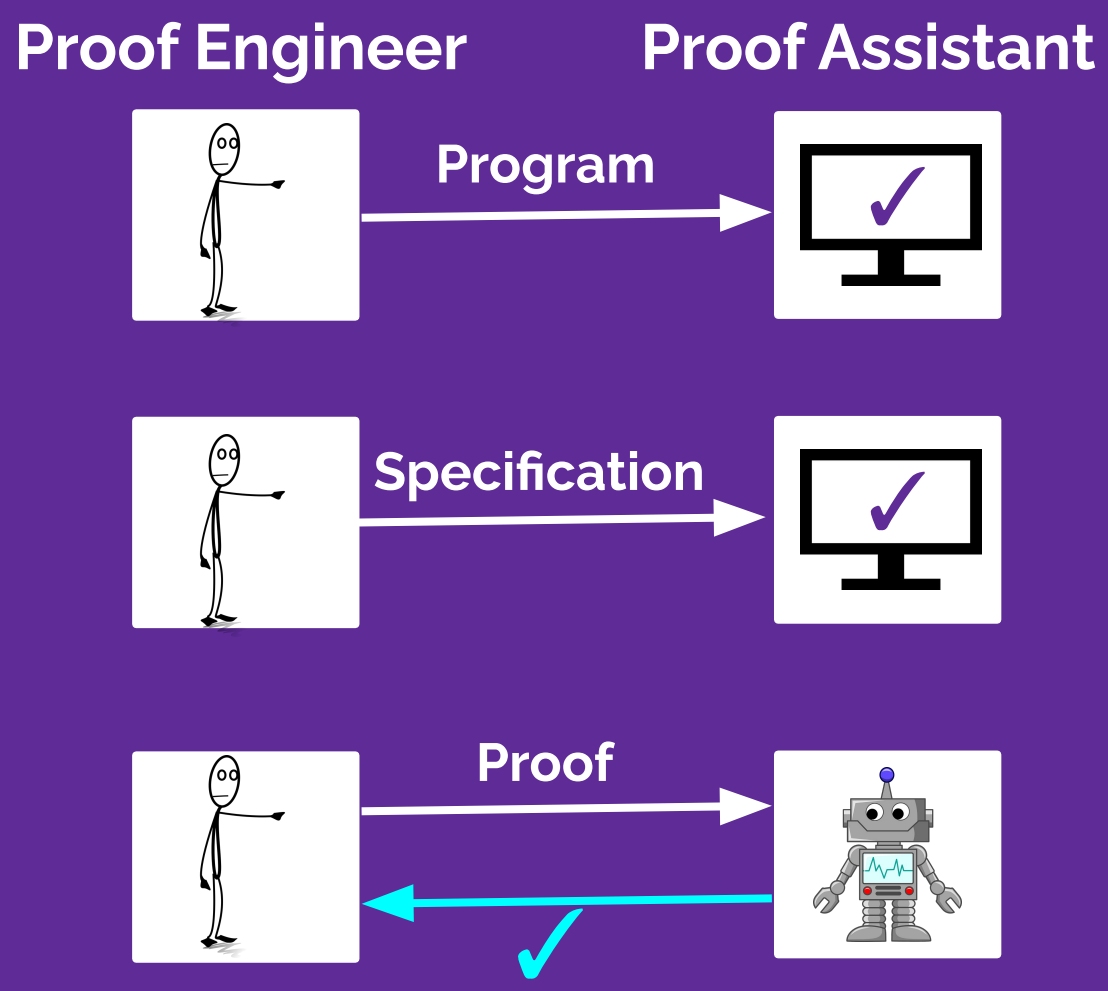
Conjecture Testing (Task 1 of 5)

Myth



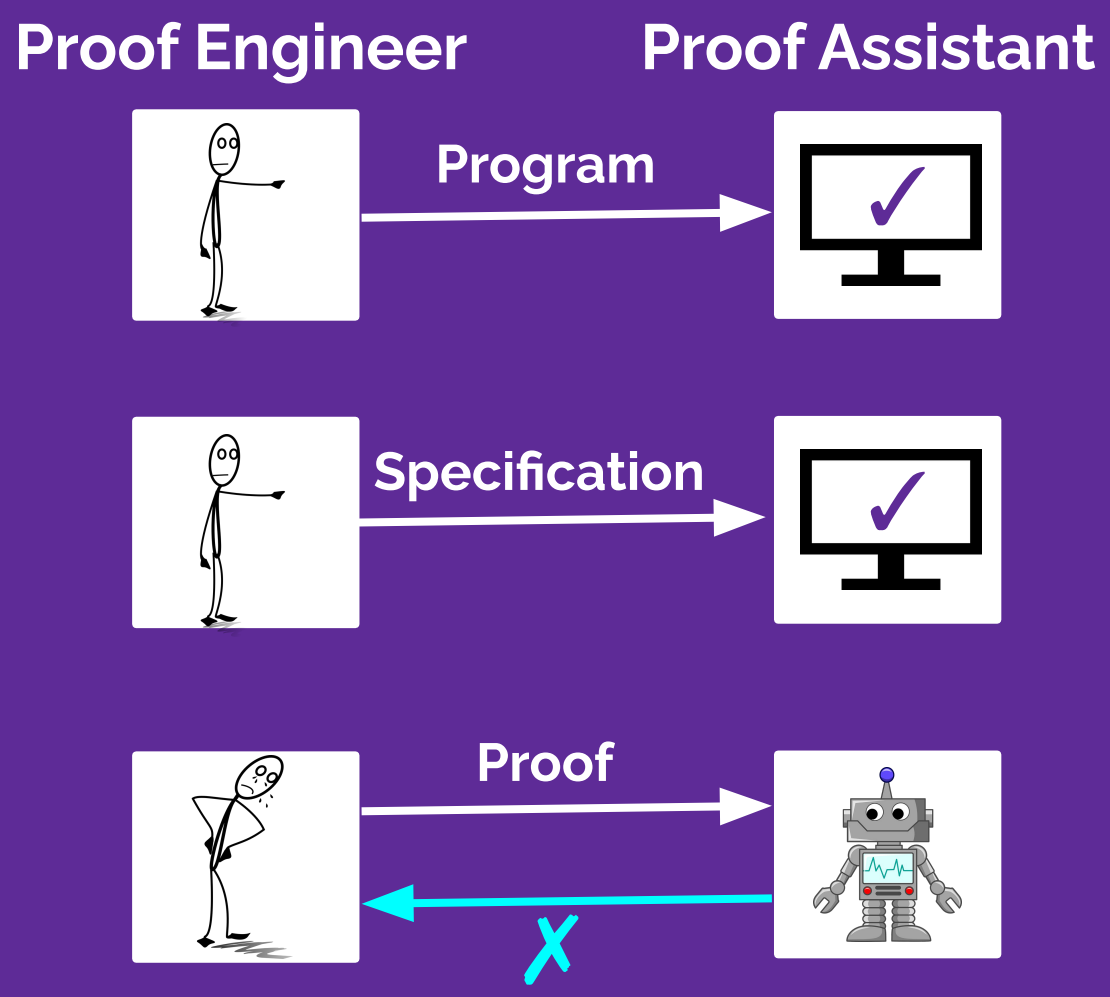
Conjecture Testing (Task 1 of 5)

Myth



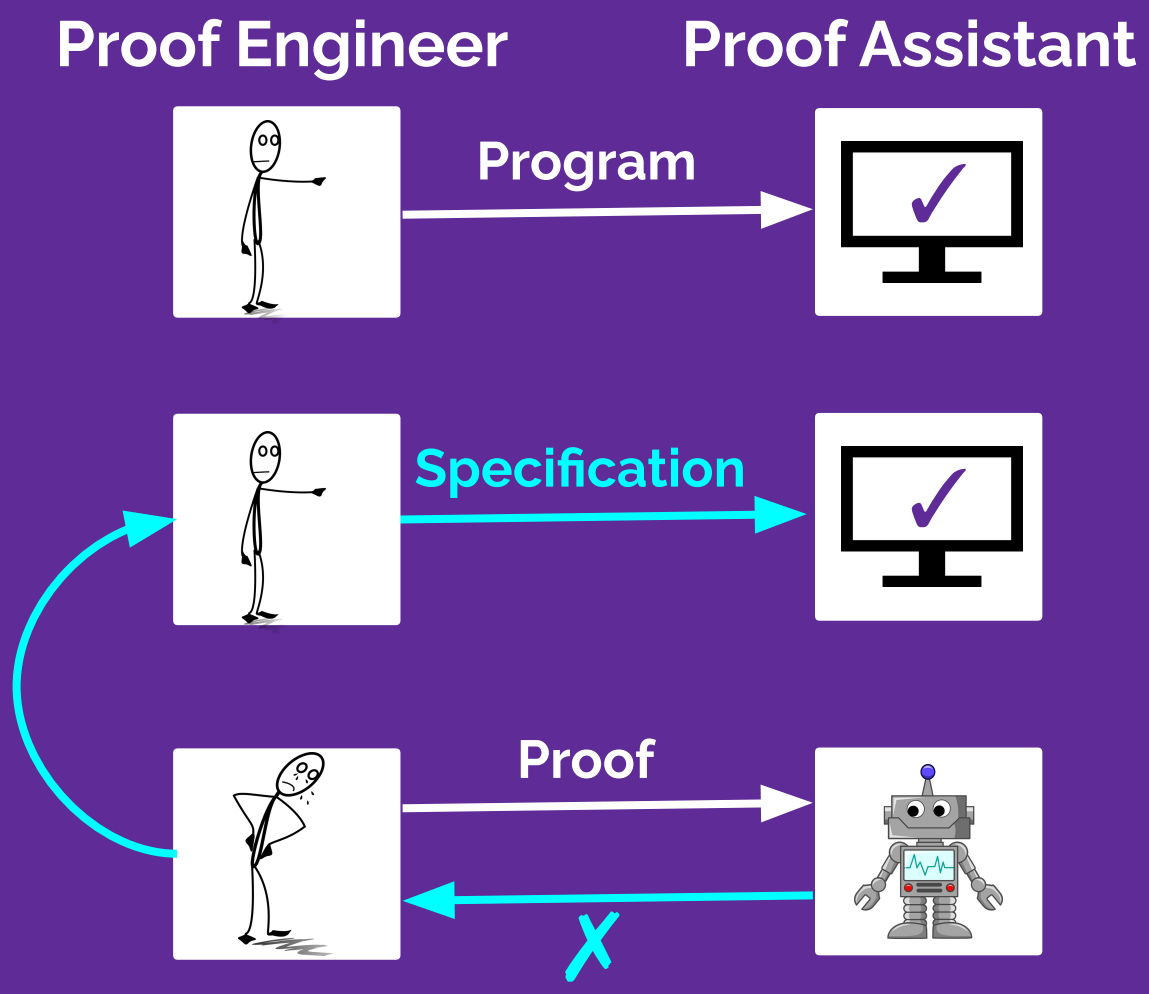
Conjecture Testing (Task 1 of 5)

Reality



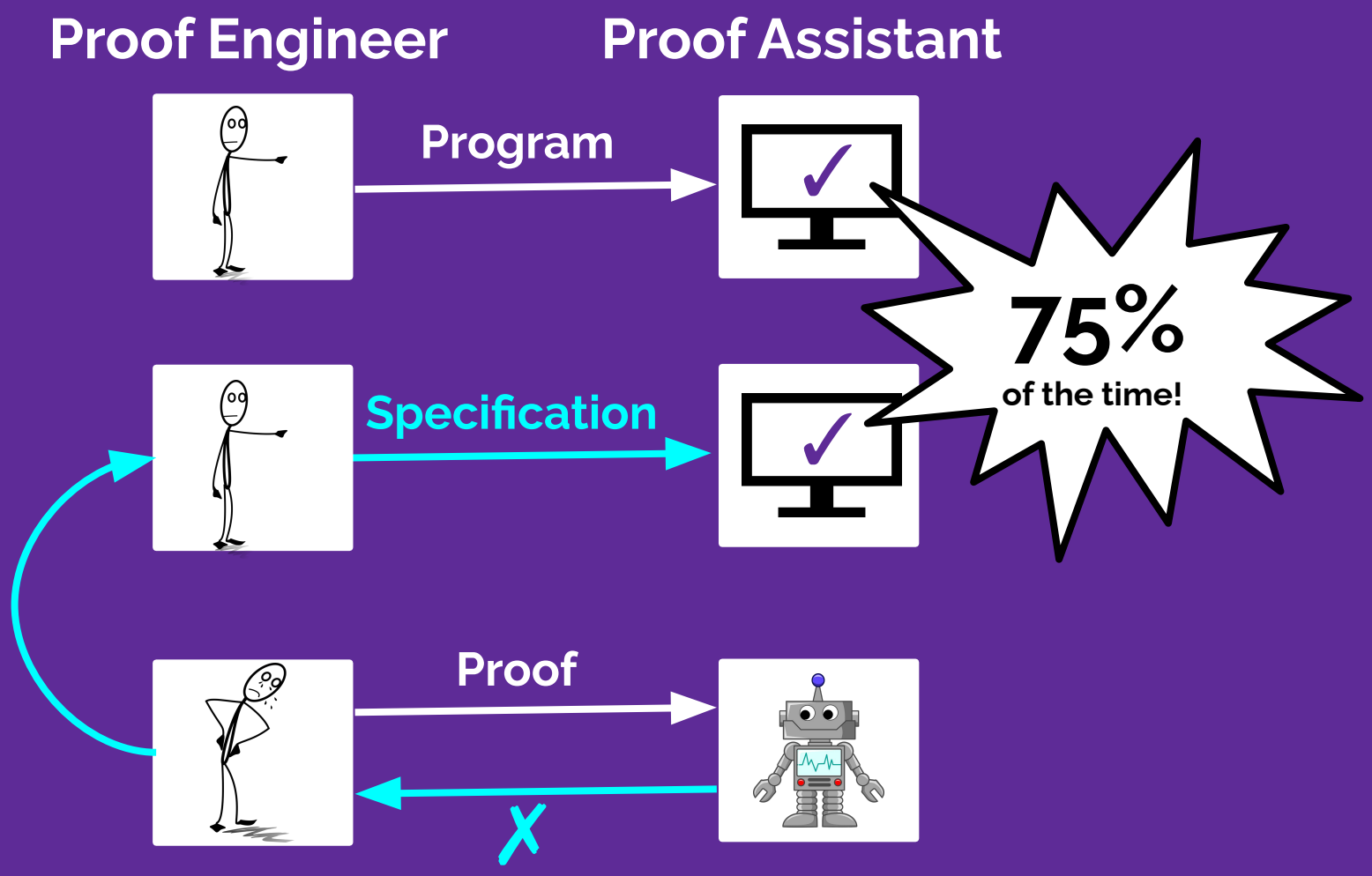
Conjecture Testing (Task 1 of 5)

Reality



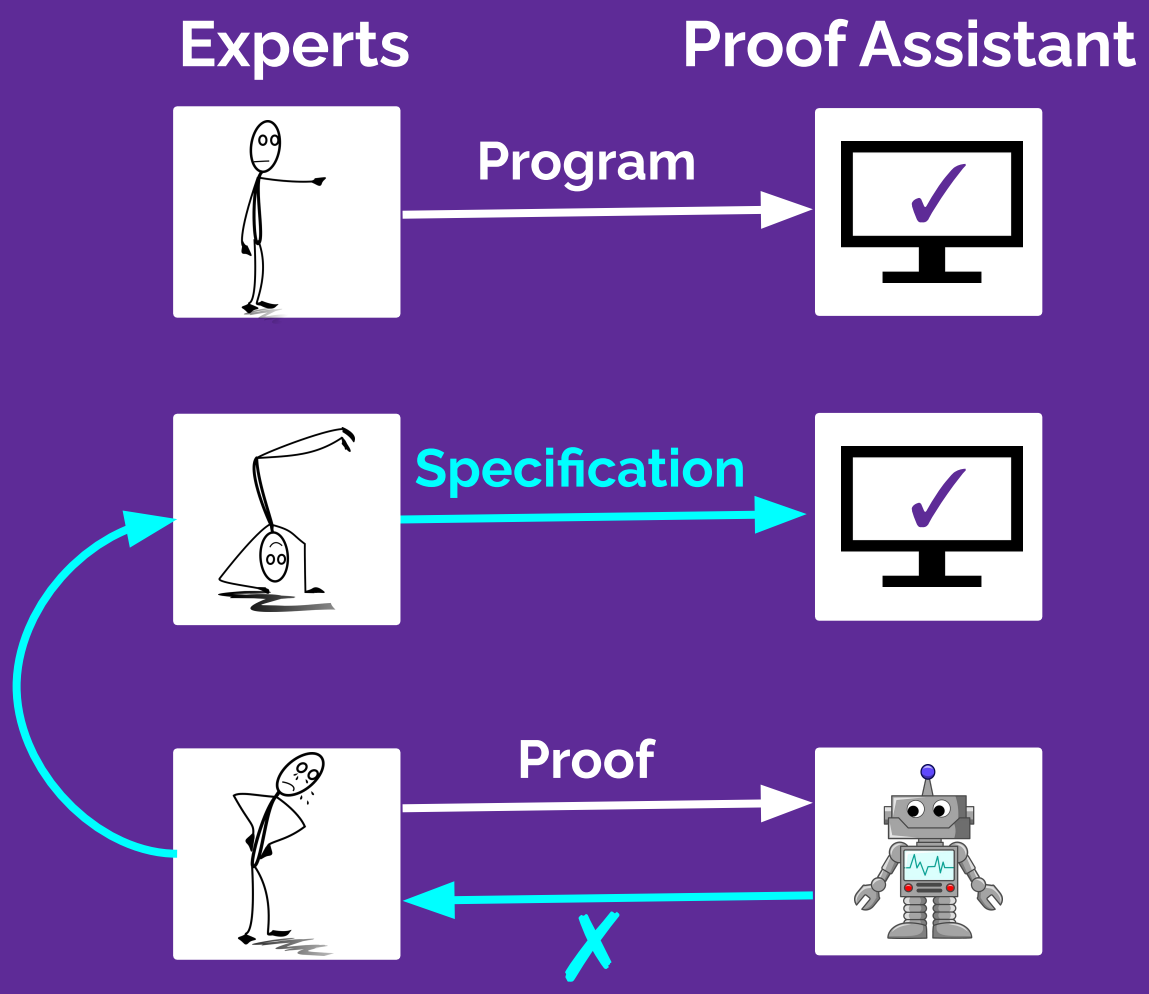
Conjecture Testing (Task 1 of 5)

Reality



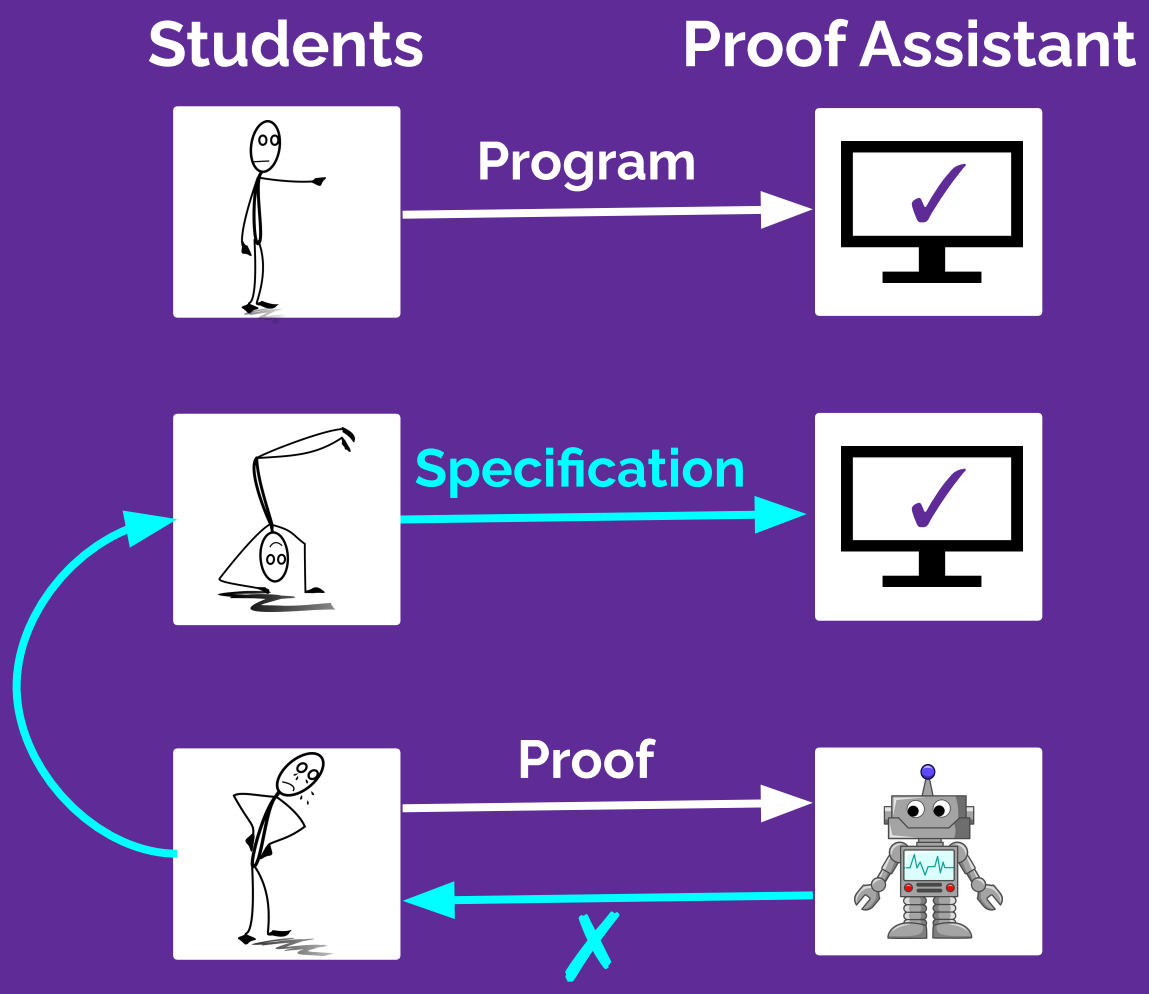
Conjecture Testing (Task 1 of 5)

It's Hard



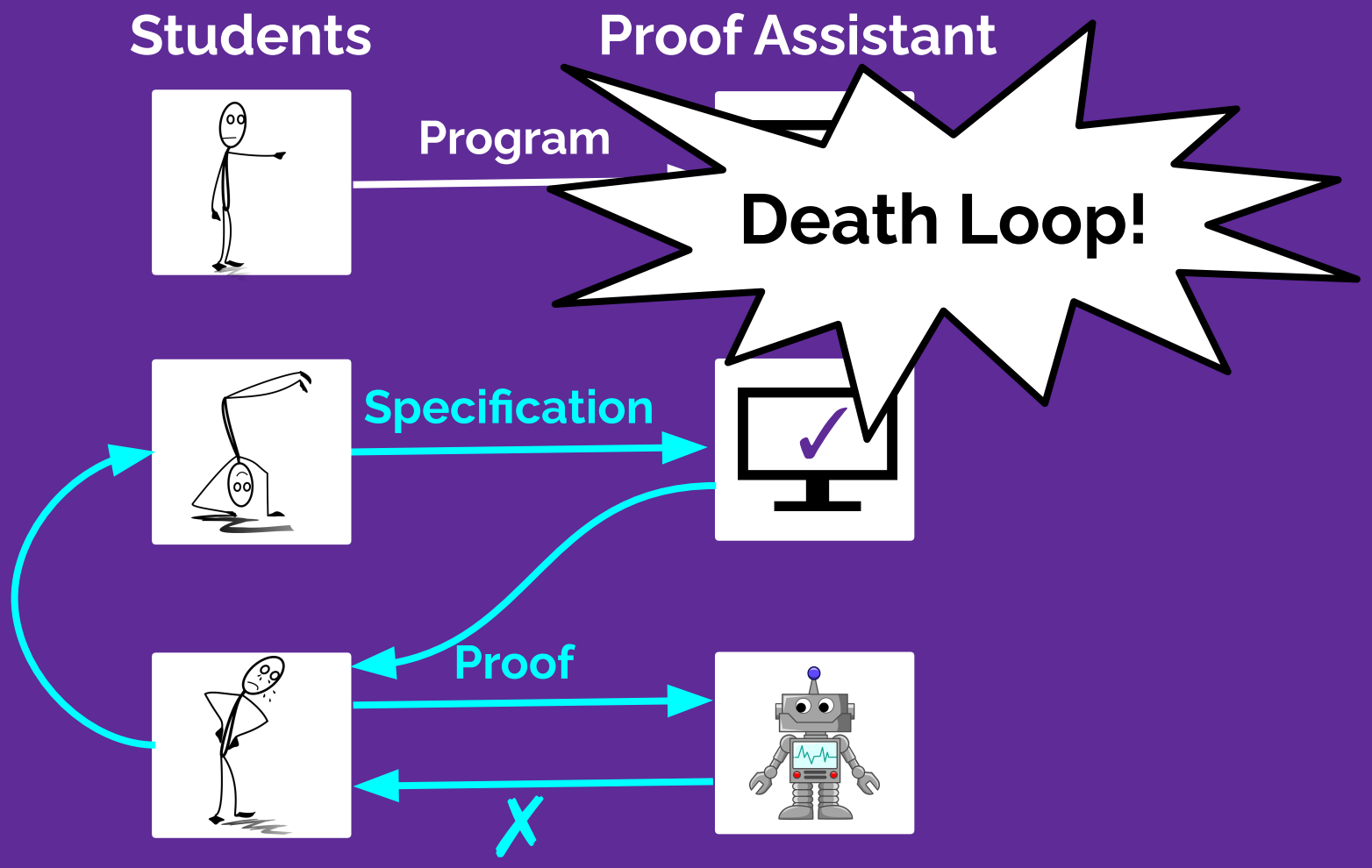
Conjecture Testing (Task 1 of 5)

It's Hard



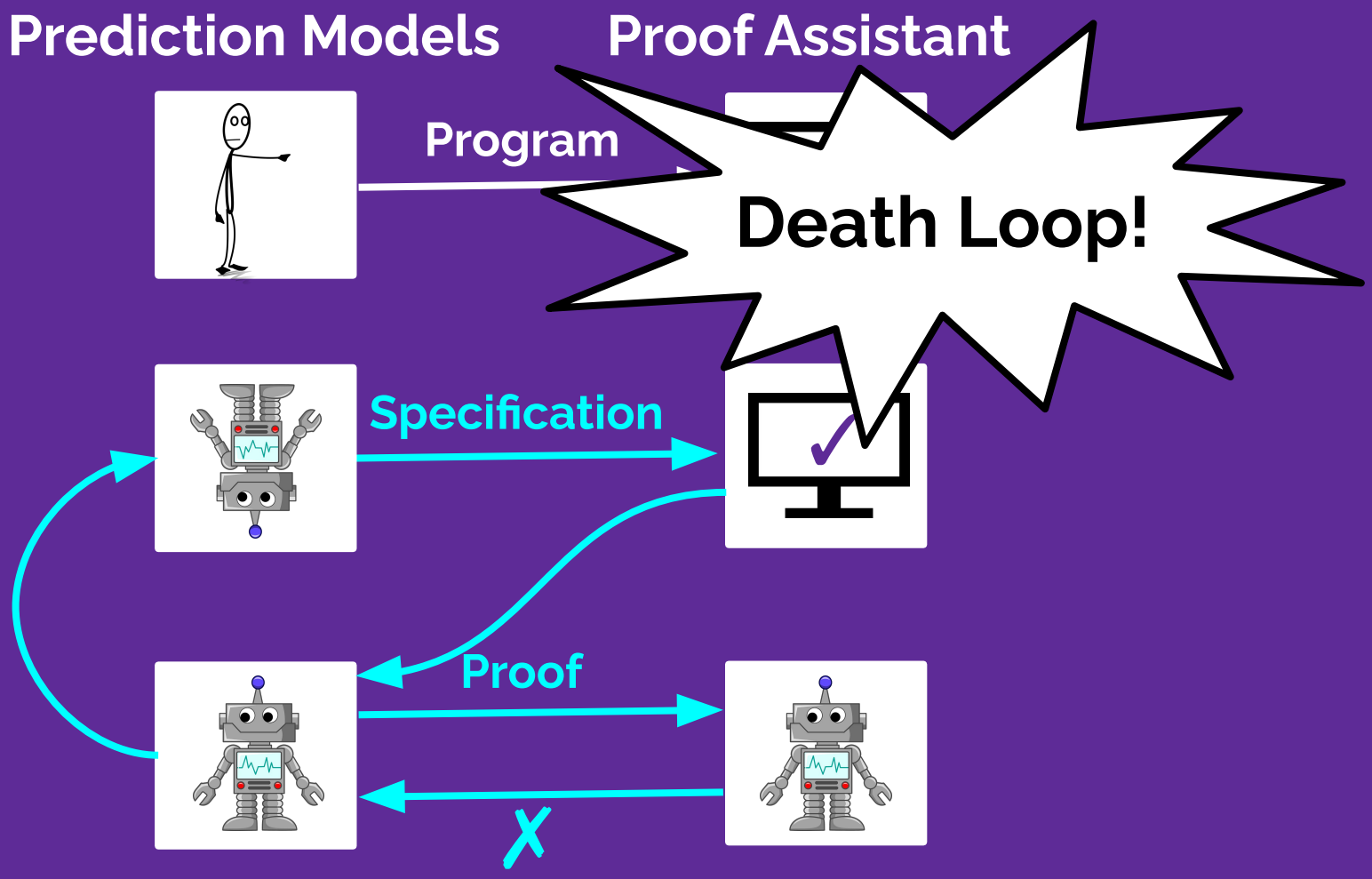
Conjecture Testing (Task 1 of 5)

It's Hard



Conjecture Testing (Task 1 of 5)

It's Hard

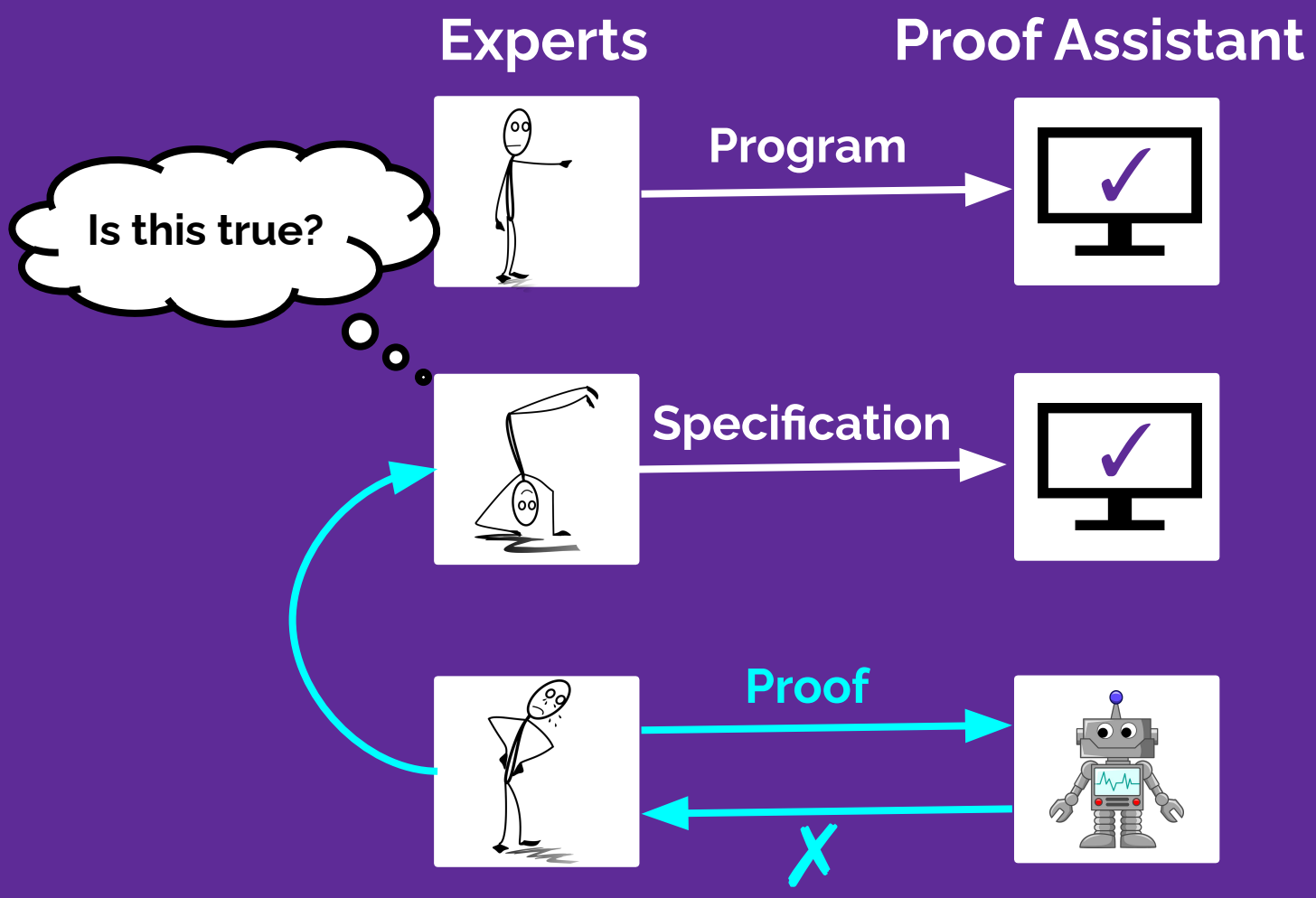


Conjecture Testing (Task 1 of 5)

The death loop
is a symptom of
gaming proofs.

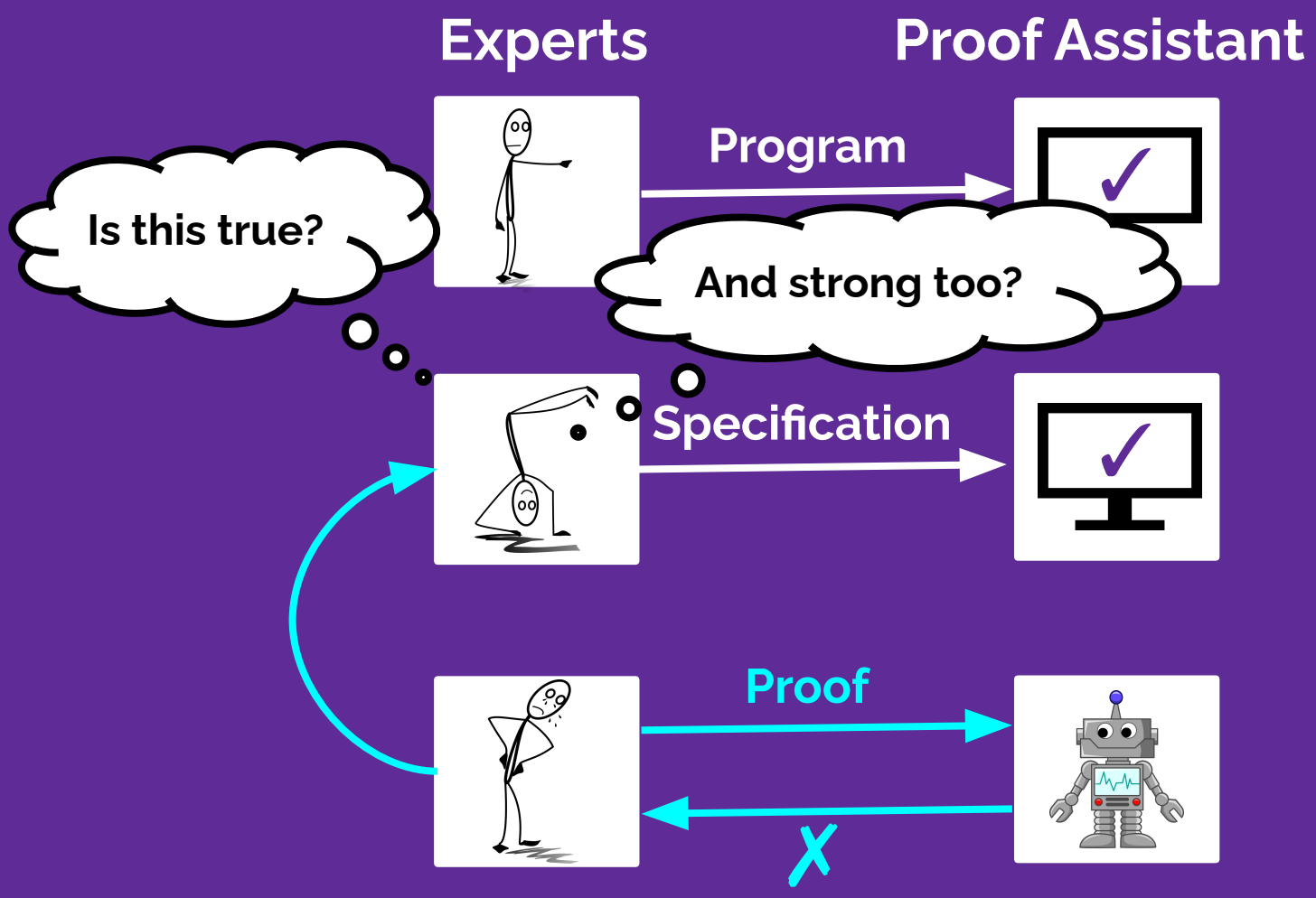
Conjecture Testing (Task 1 of 5)

What Experts Do



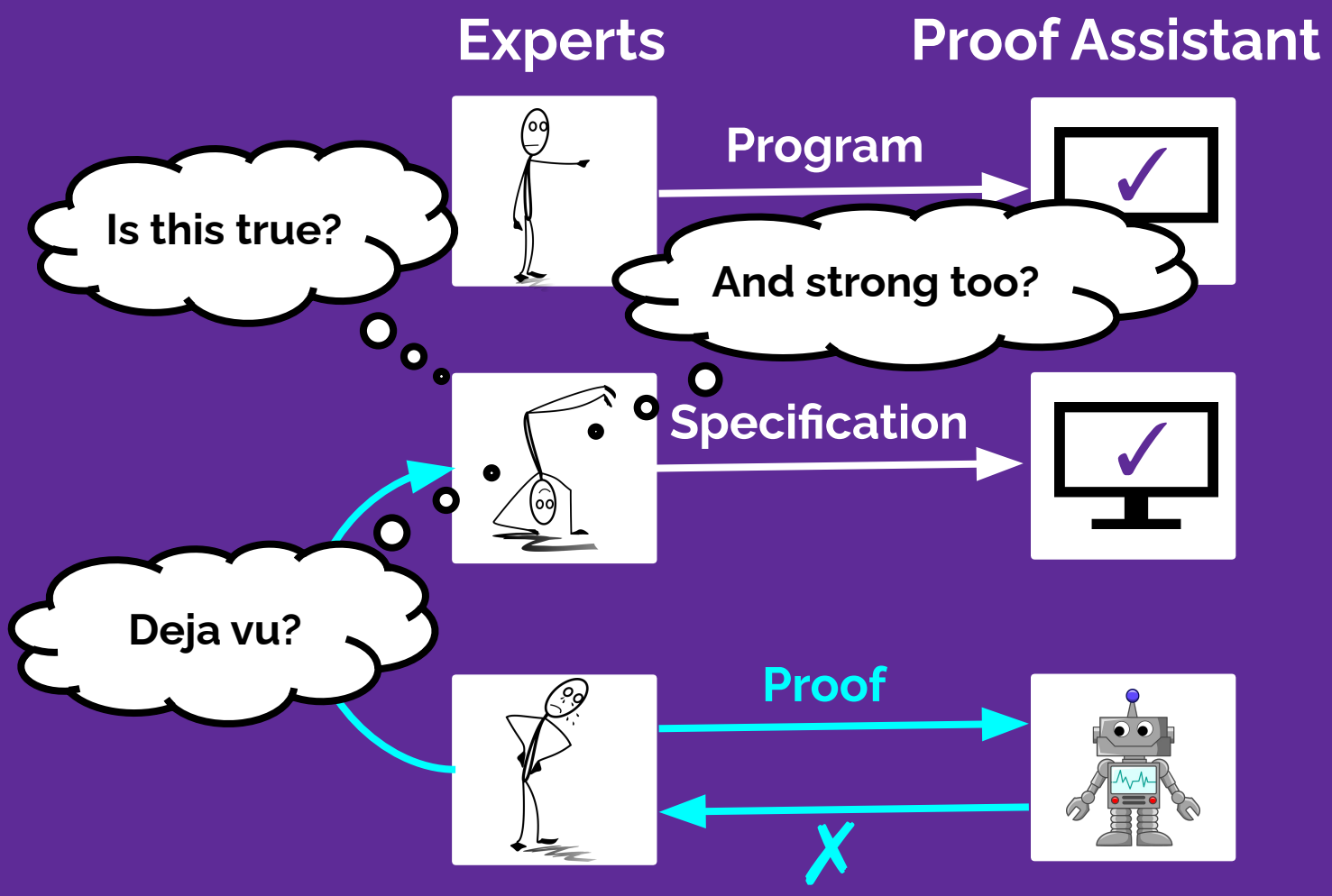
Conjecture Testing (Task 1 of 5)

What Experts Do



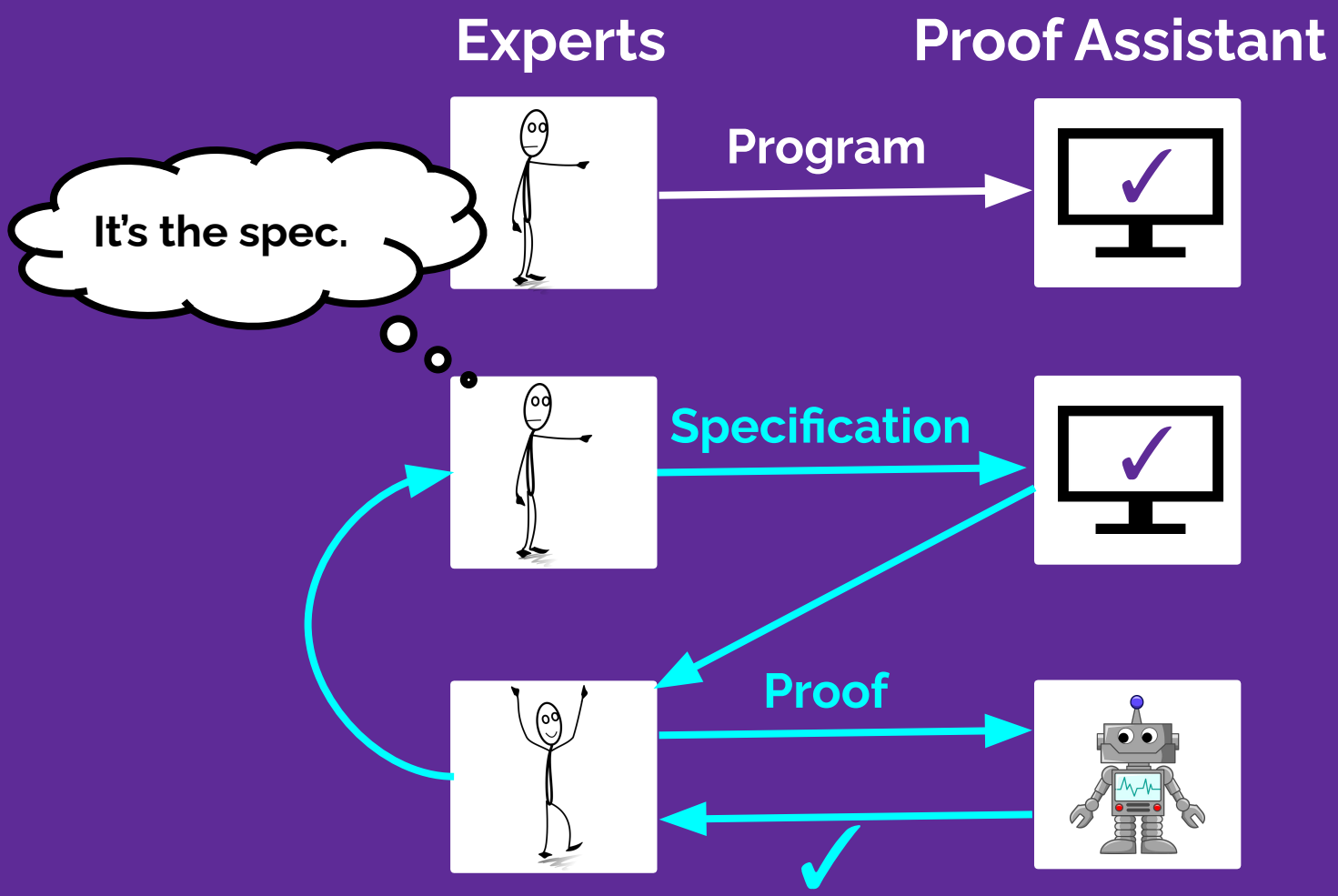
Conjecture Testing (Task 1 of 5)

What Experts Do



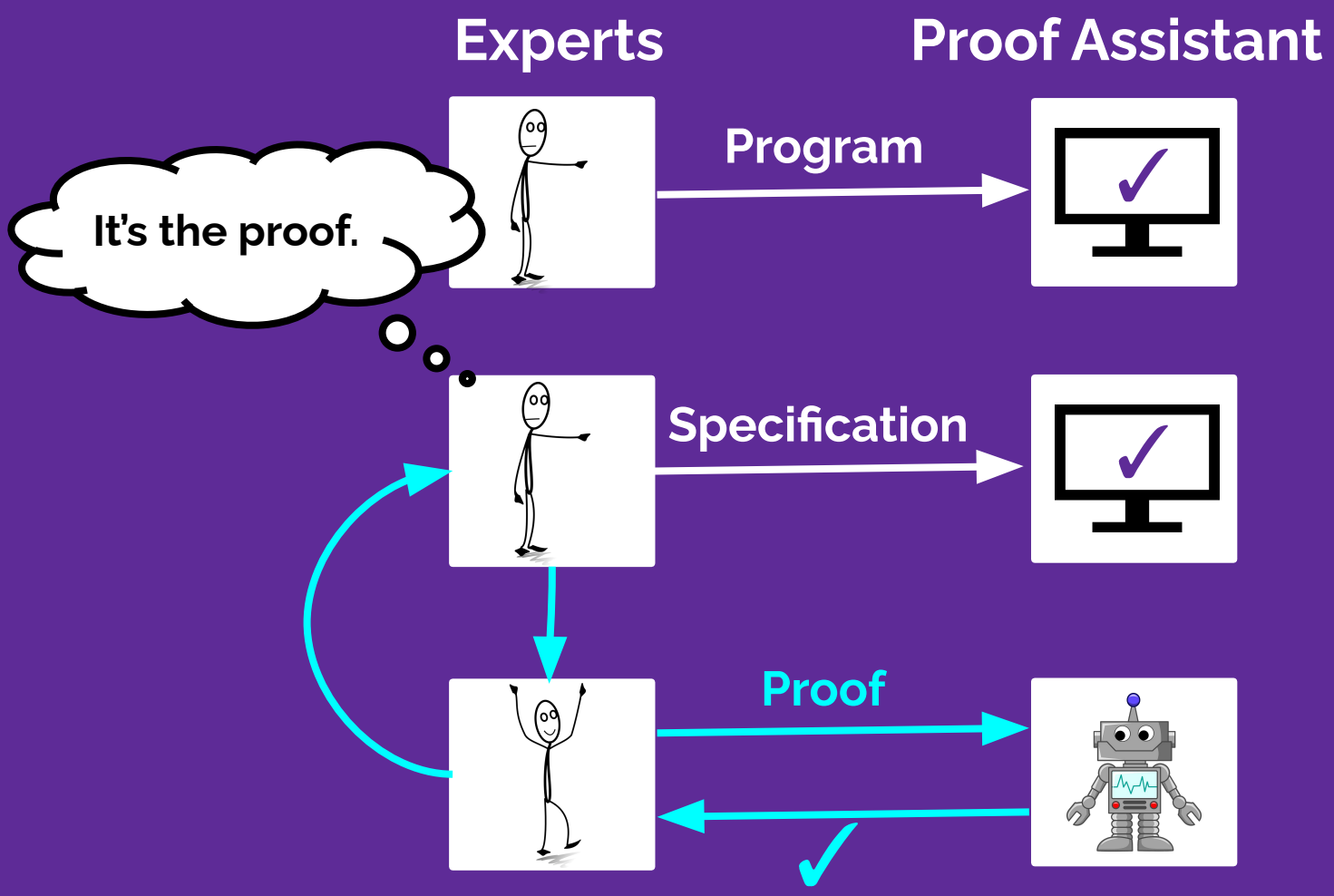
Conjecture Testing (Task 1 of 5)

What Experts Do



Conjecture Testing (Task 1 of 5)

What Experts Do



Conjecture Testing (Task 1 of 5)

Good automation ought to answer these questions.

Conjecture Testing (Task 1 of 5)

What's Hard

Static proof data mostly contains true conjectures, plus getting symbolic tools to play nicely with ML is hard but needed.

Conjecture Testing (Task 1 of 5)

Ideas 💡

More granular data via instrumentation, capturing the development process?

Conjecture Testing (Task 1 of 5)

Ideas 💡

Synthetic data with false propositions?

Conjecture Testing (Task 1 of 5)

Ideas 💡

Combine ML with or train for property-based testing or counterexample generation?

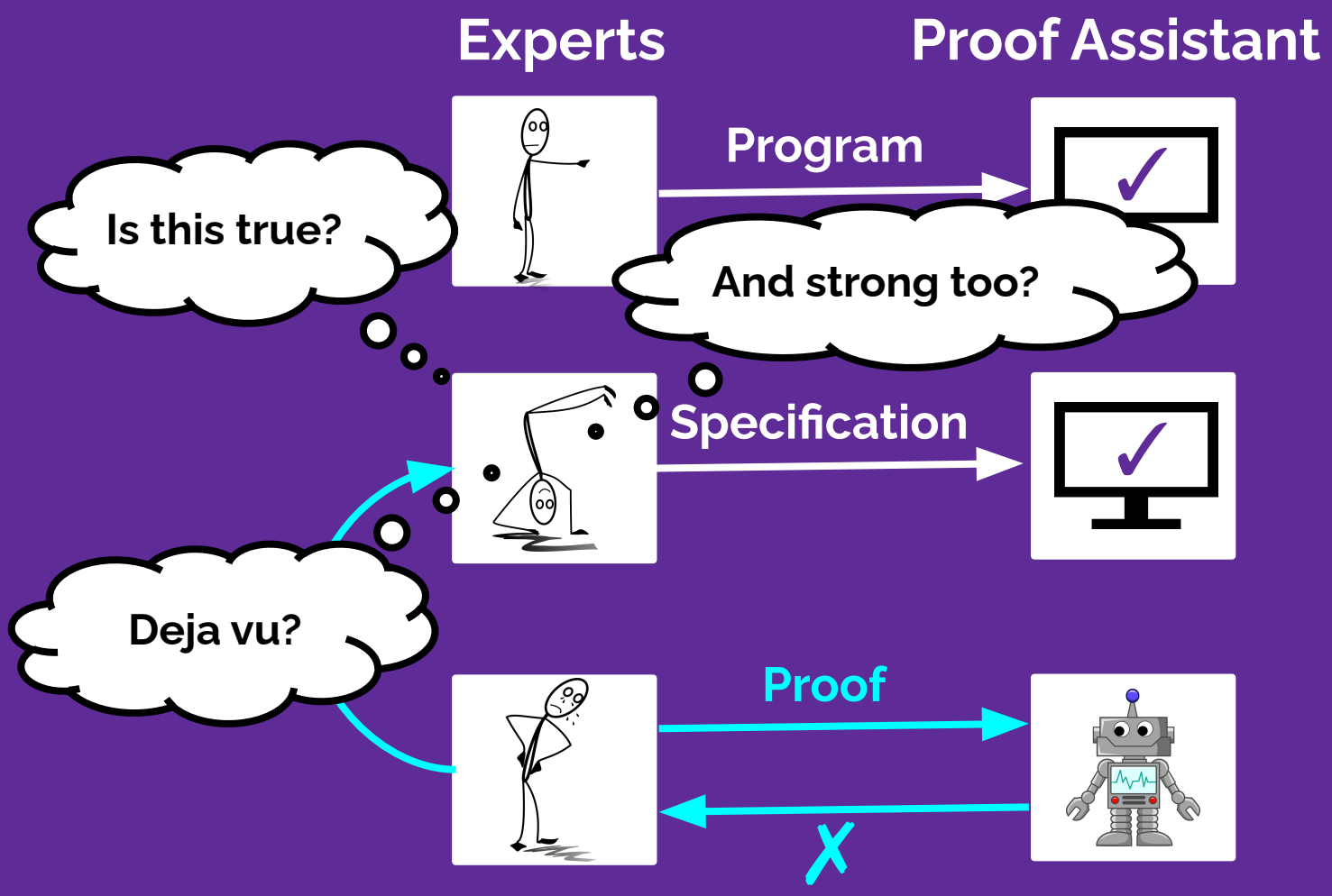
Conjecture Testing (Task 1 of 5)

Ideas 💡

Your ideas here!

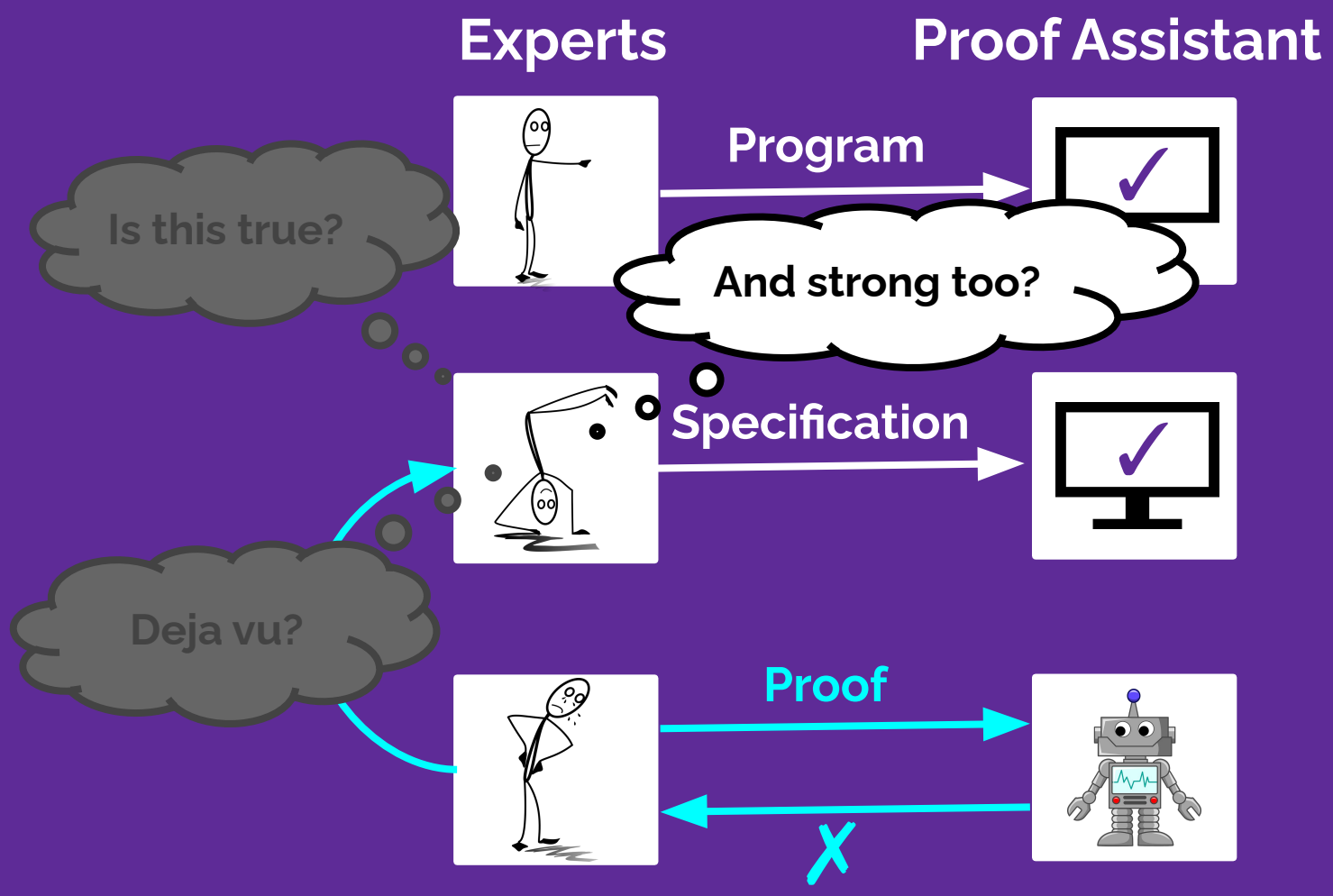
Conjecture Testing (Task 1 of 5)

What Experts Do



Conjecture Testing (Task 1 of 5)

What Experts Do



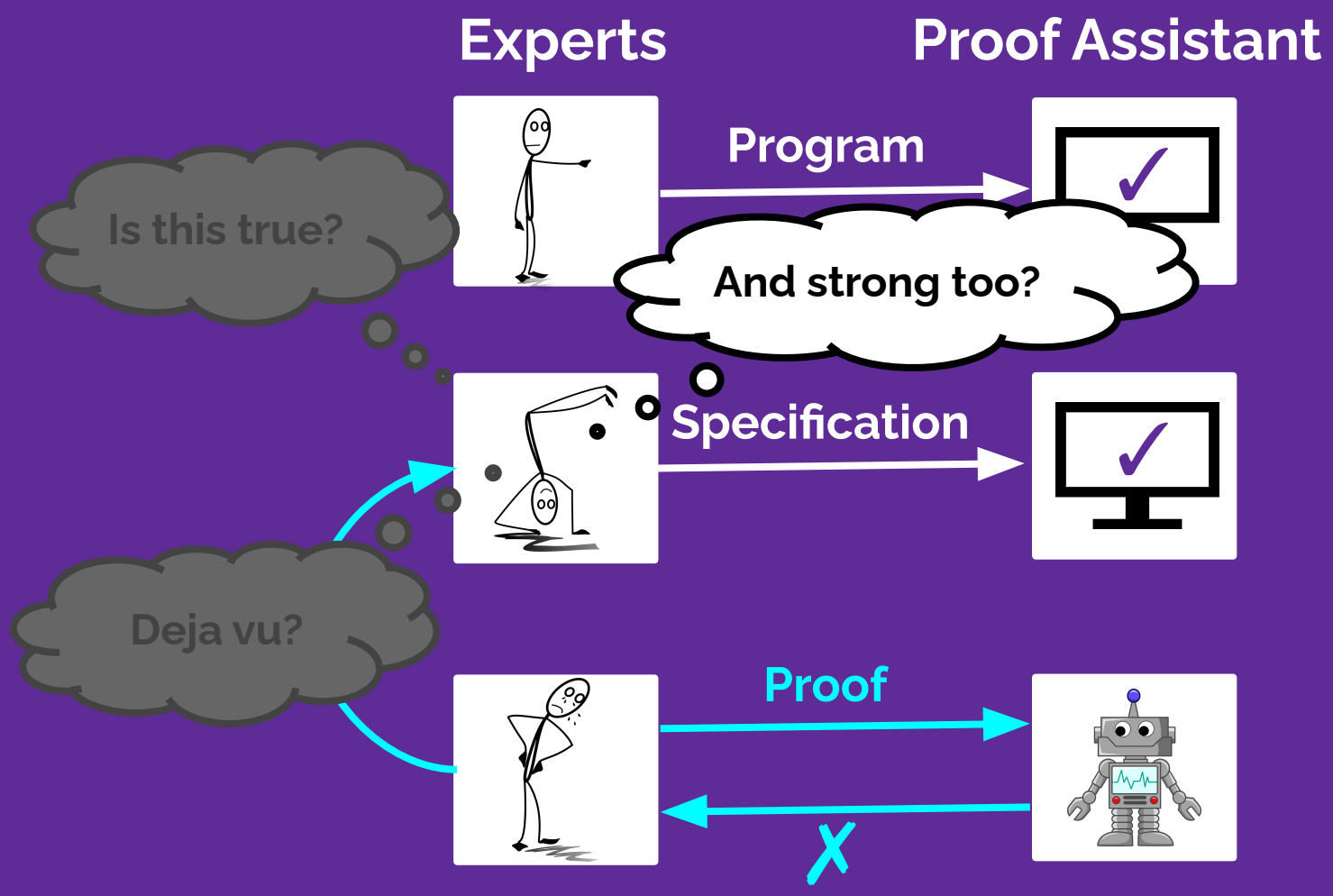
Conjecture Testing (Task 1 of 5)

Do lots more:

1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

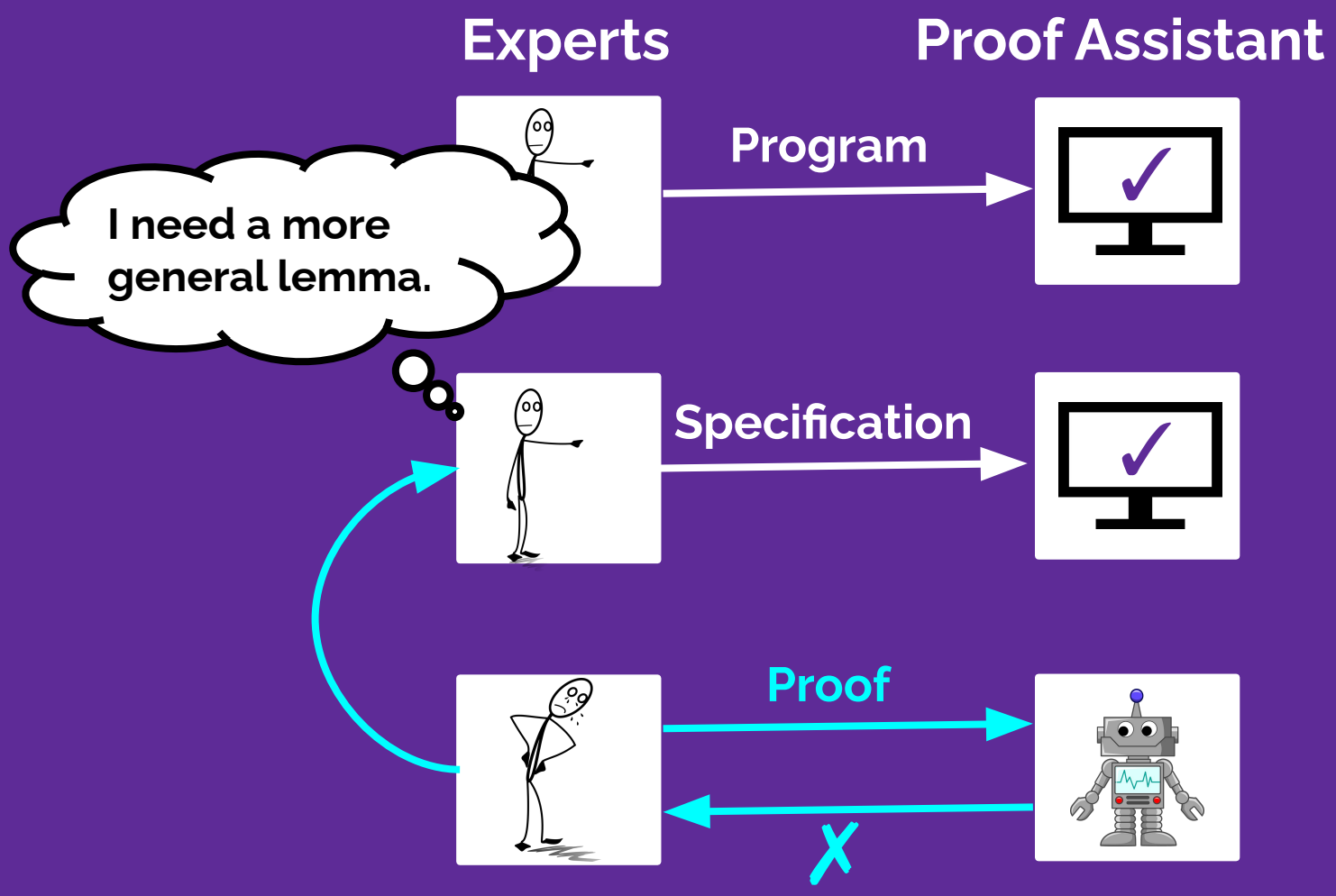
You won't regret it!

Not Just Useful—Essential



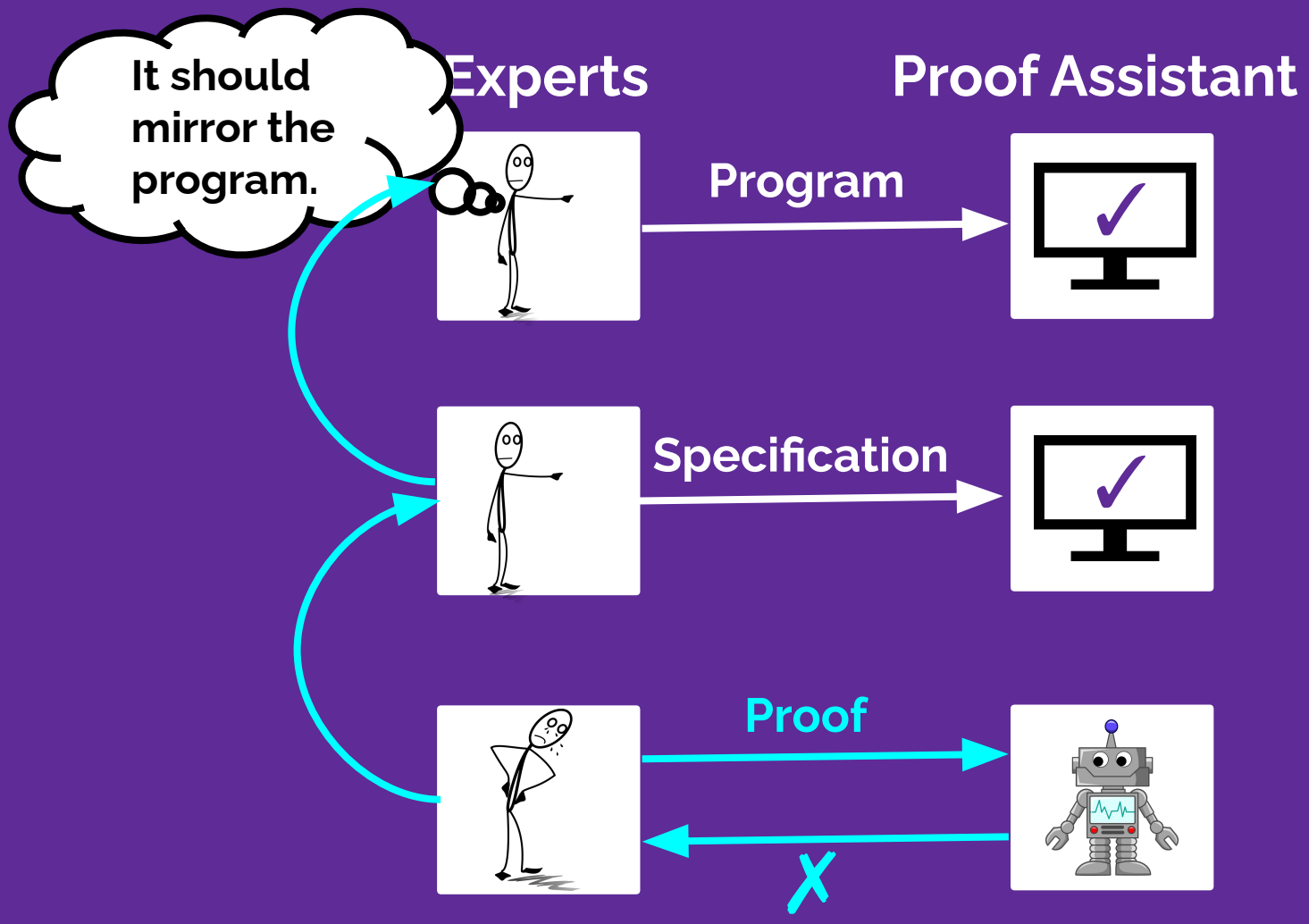
Lemma Discovery (Task 2 of 5)

Not Just Useful—Essential



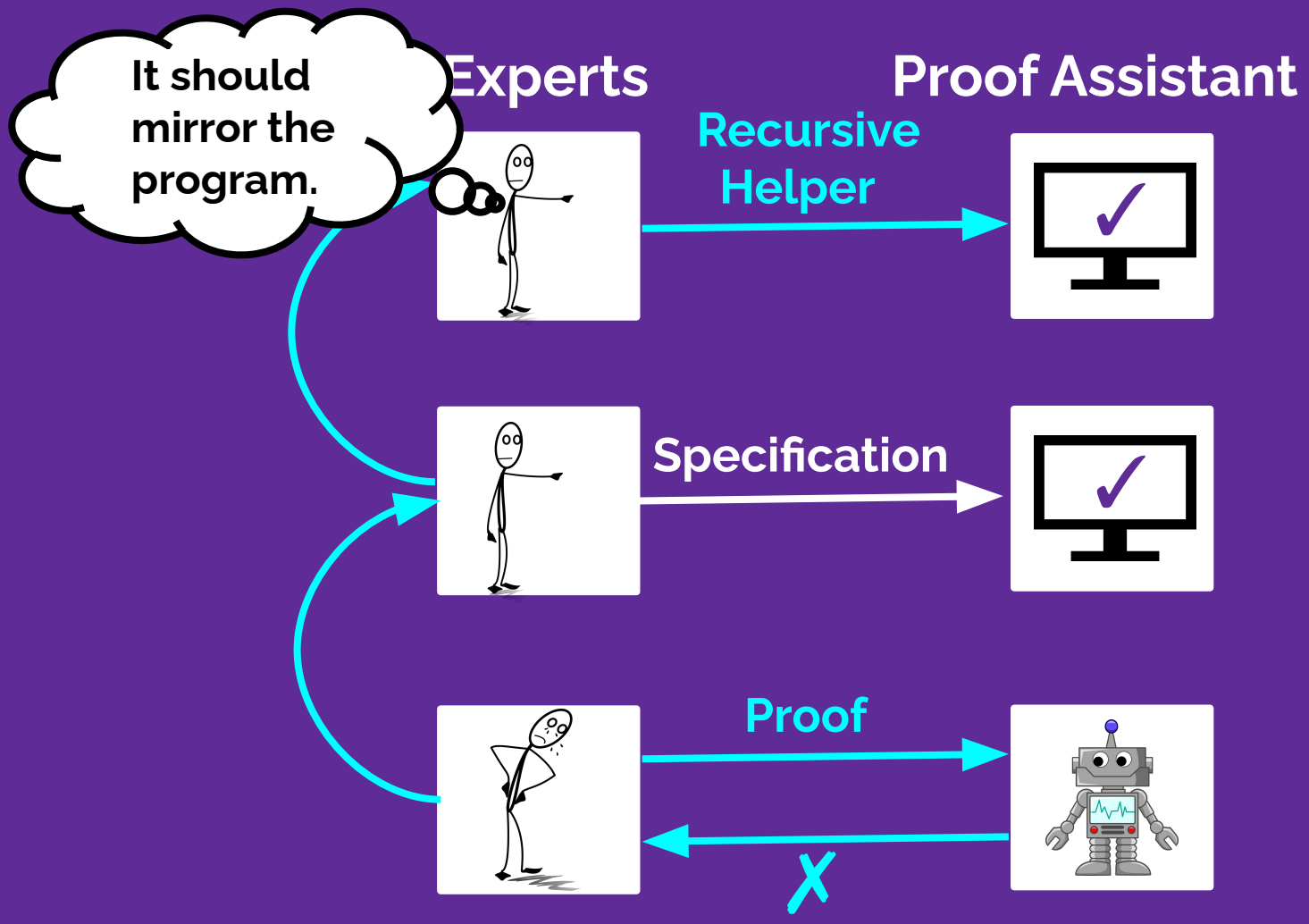
Lemma Discovery (Task 2 of 5)

Not Just Useful—Essential



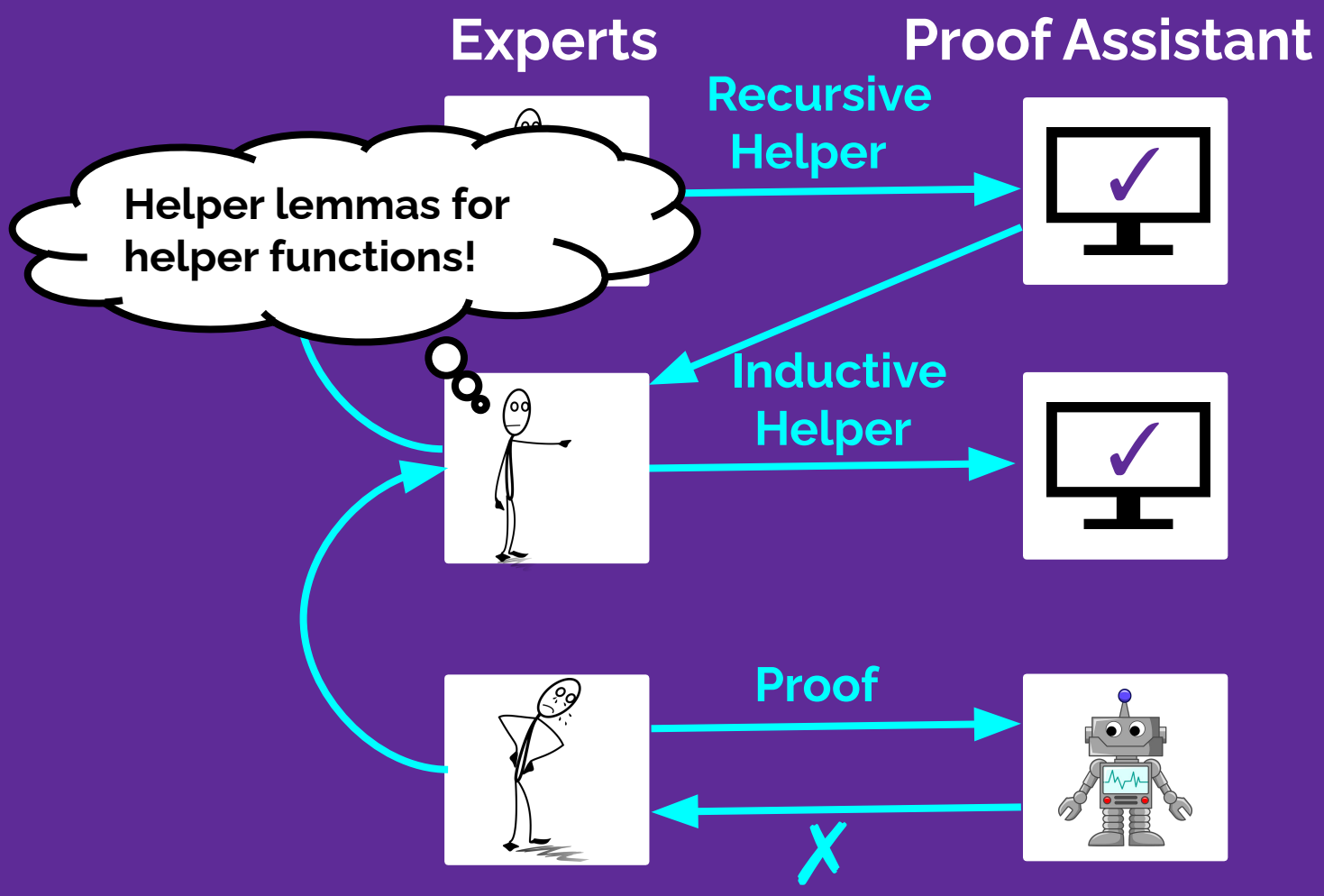
Lemma Discovery (Task 2 of 5)

Not Just Useful—Essential



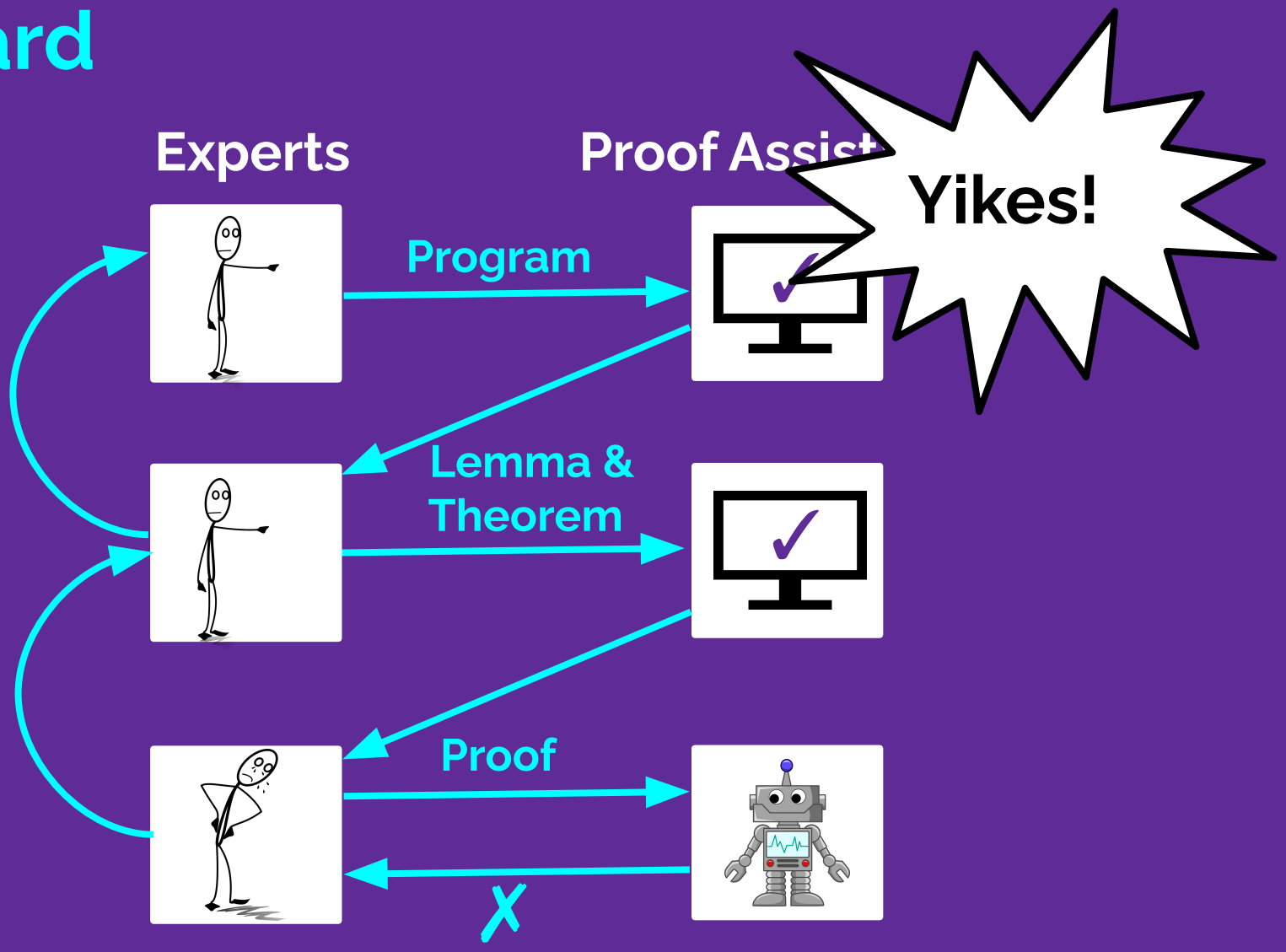
Lemma Discovery (Task 2 of 5)

Not Just Useful—Essential



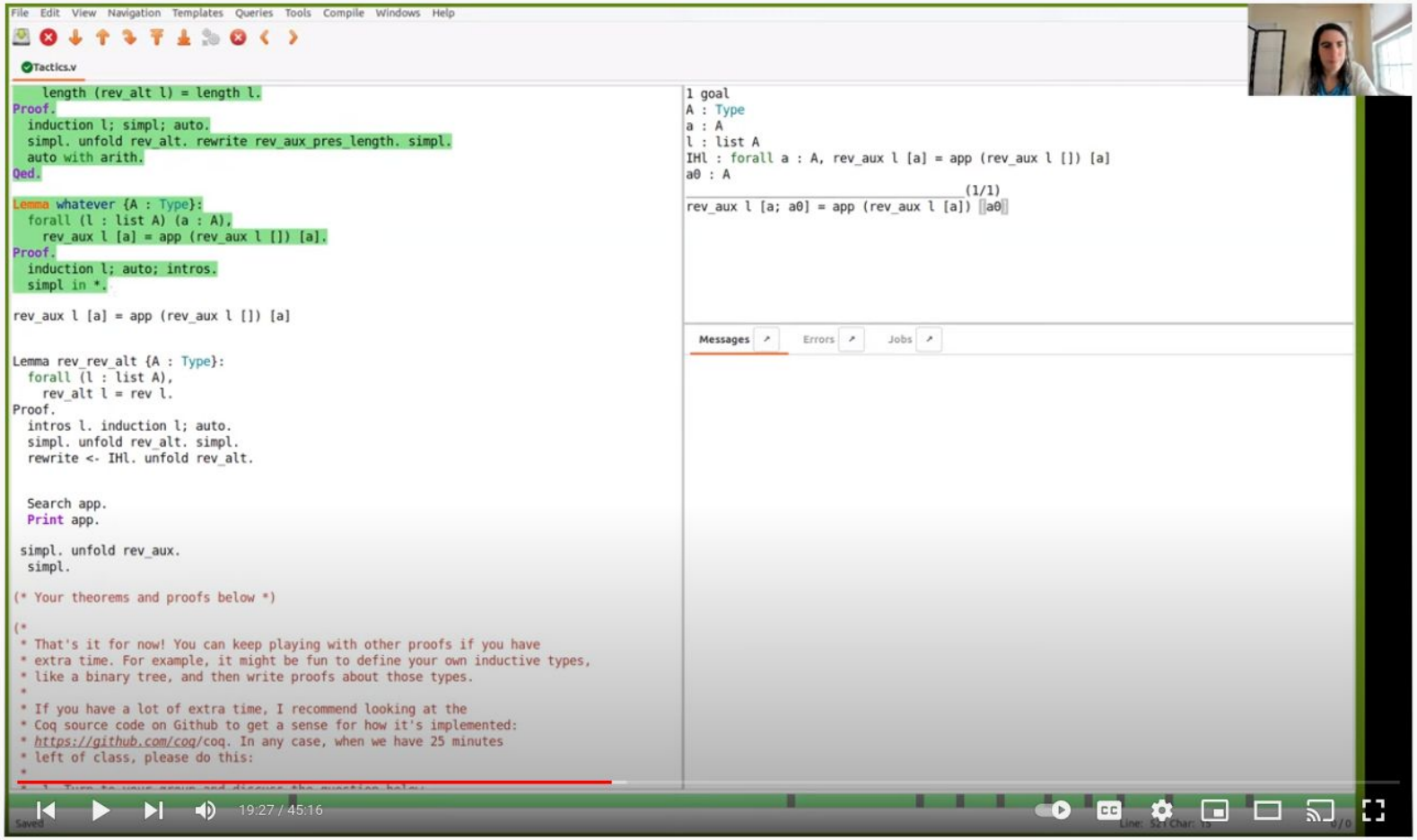
Lemma Discovery (Task 2 of 5)

It's Hard



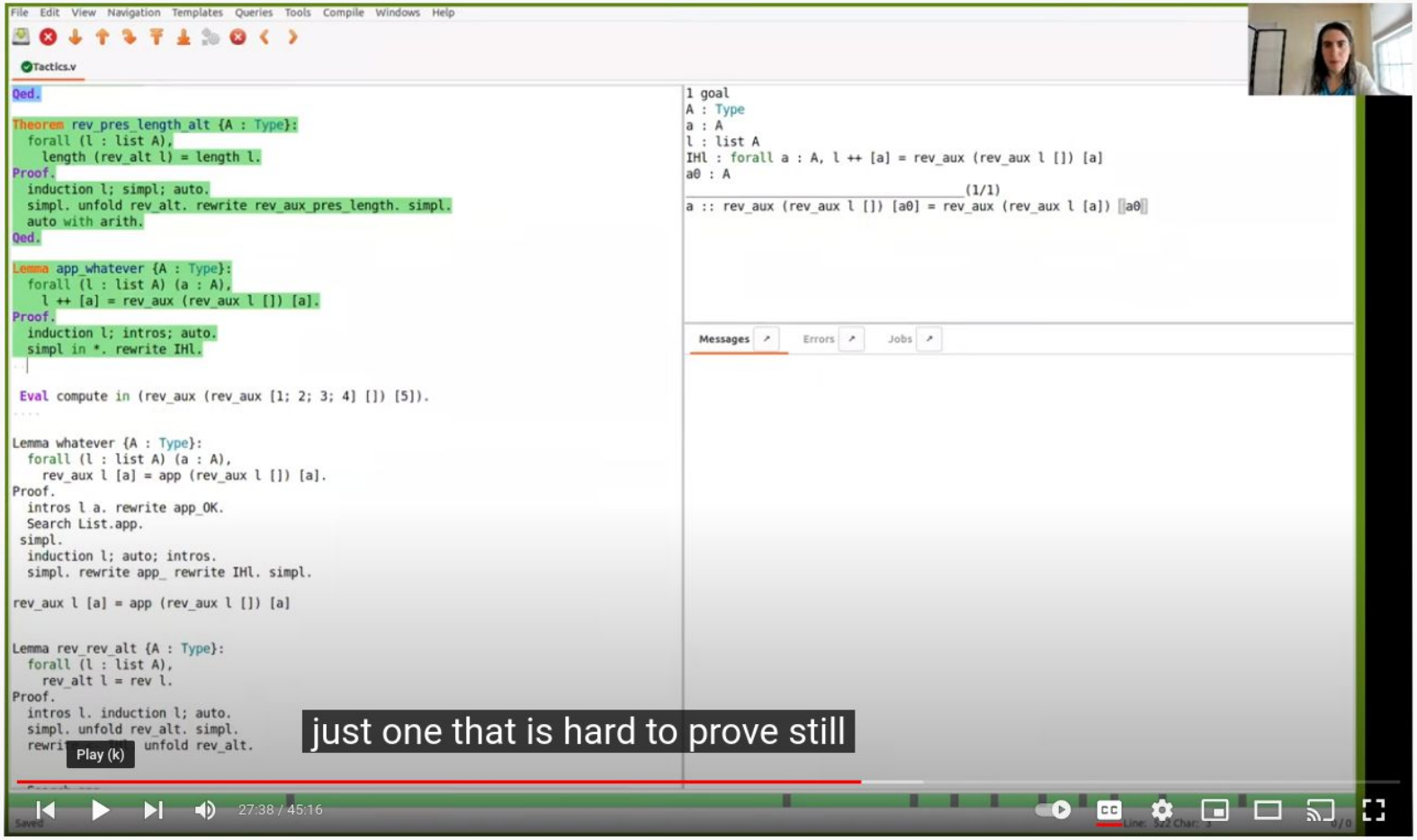
Lemma Discovery (Task 2 of 5)

It's Hard



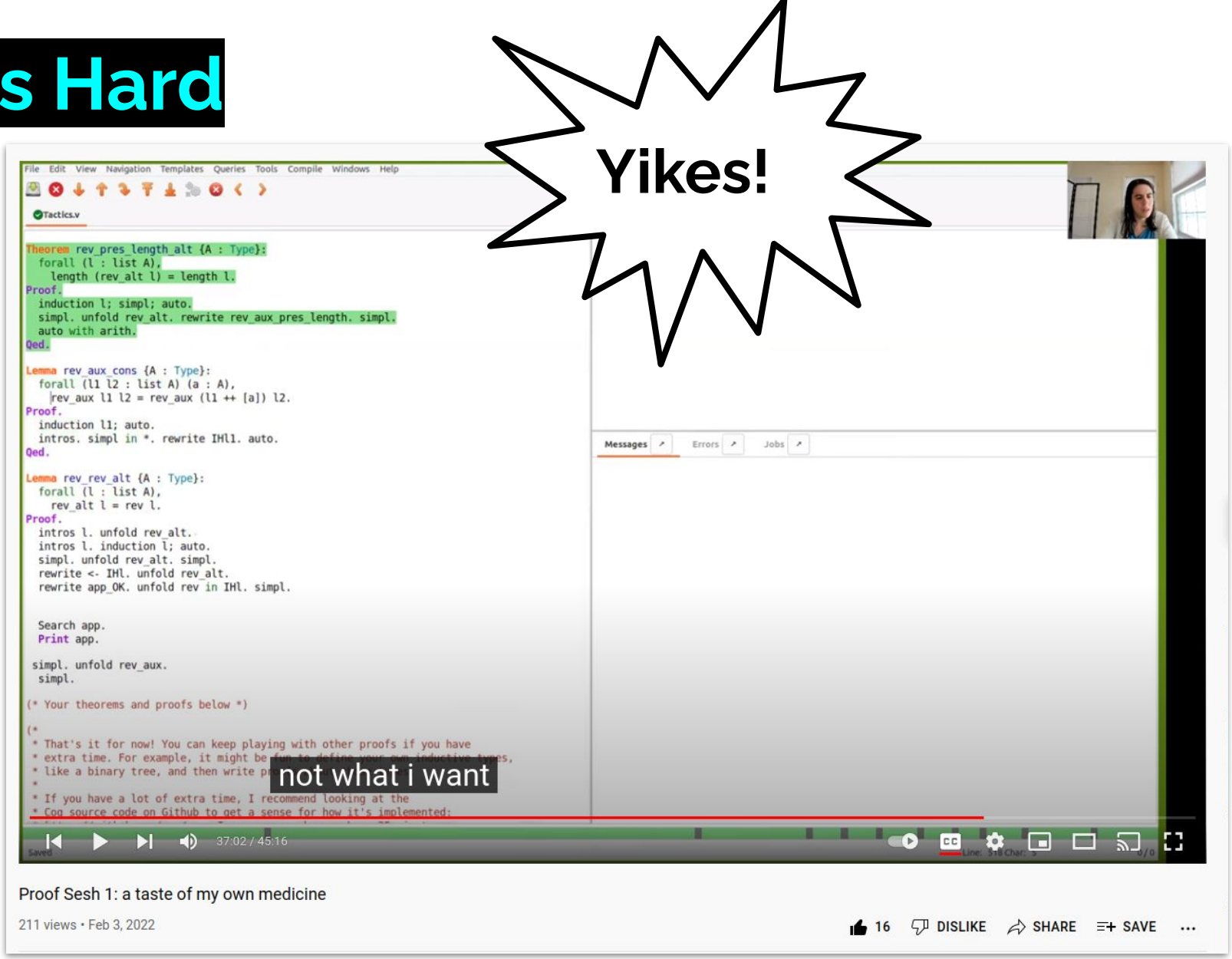
Lemma Discovery (Task 2 of 5)

It's Hard



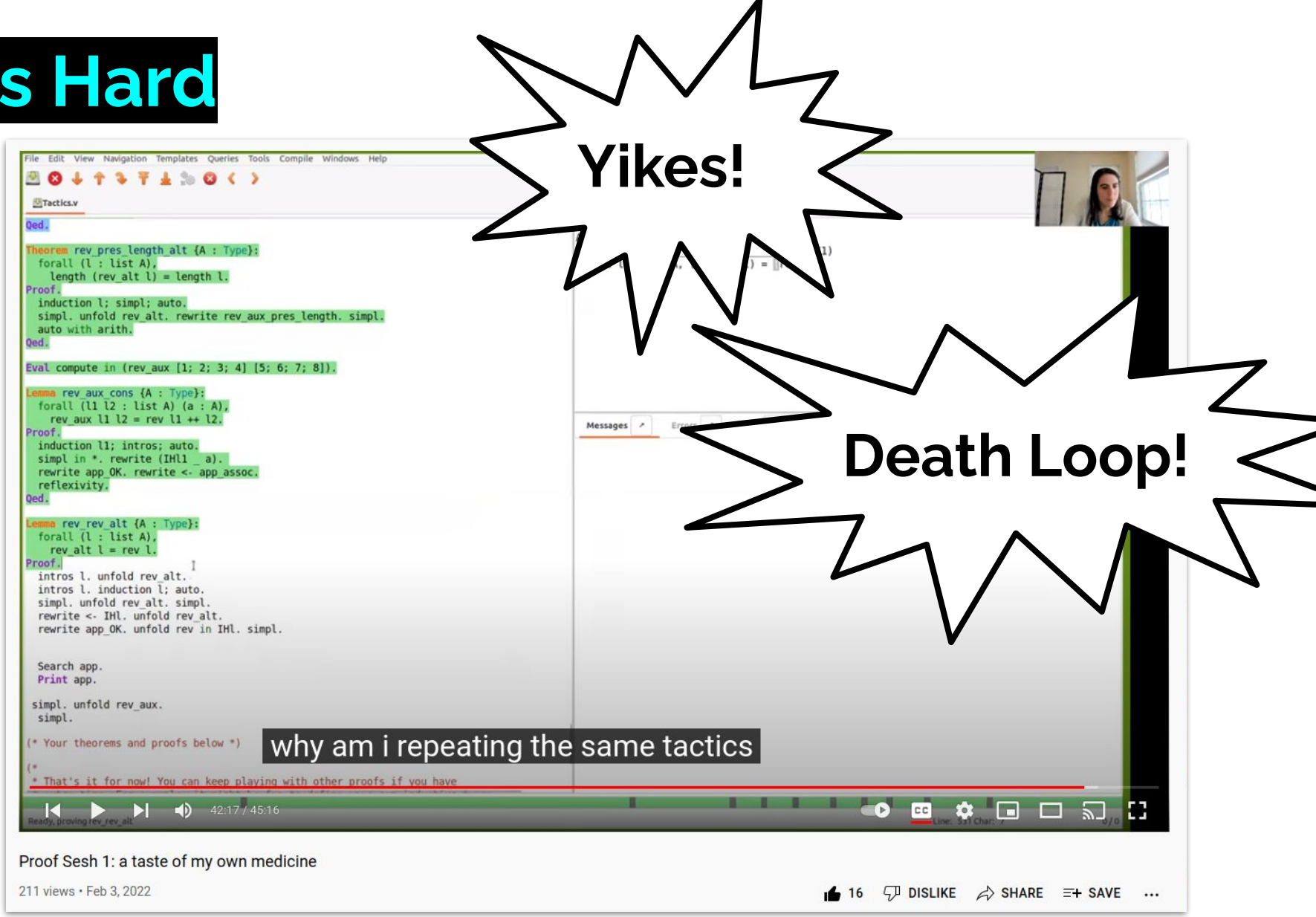
Lemma Discovery (Task 2 of 5)

It's Hard



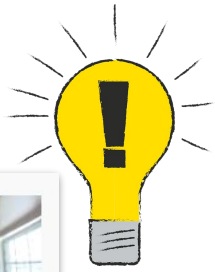
Lemma Discovery (Task 2 of 5)

It's Hard



Lemma Discovery (Task 2 of 5)

It's Hard



File Edit View Navigation Templates Queries Tools Compile Windows Help

Tactics.v

```
Qed.

Theorem rev_pres_length_alt {A : Type}:
  forall (l : list A),
    length (rev_alt l) = length l.
Proof.
  induction l; simpl; auto.
  simpl. unfold rev_alt. rewrite rev_aux_pres_length. simpl.
  auto with arith.
Qed.

Eval compute in (rev_aux [1; 2; 3; 4] [5; 6; 7; 8]).

Lemma rev_aux_app {A : Type}:
  forall (l1 l2 : list A),
    rev_aux l1 l2 = rev l1 ++ l2.
Proof.
  induction l1; intros; auto.
  simpl in *. rewrite IHl1.
  rewrite app_OK. rewrite <- app_assoc.
  reflexivity.
Qed.

Lemma rev_rev_alt {A : Type}:
  forall (l : list A),
    rev_alt l = rev l.
Proof.
  intros l. unfold rev_alt. rewrite unfold rev.
 induction l; auto.
  simpl. unfold rev_alt. simpl.
  rewrite <- IHl. rewrite

 unfold rev_alt.
  rewrite app_OK. unfold rev in IHl. simpl.


  Search app.
  Print app.

 simpl. unfold rev_aux.
  simpl.
(* Your and proofs below *)
```

```
1 goal
A : Type
a : A
l1 : list A
IHl1 : forall l2 : list A, rev_aux l1 l2 = rev l1 ++ l2
l2 : list A
____________________________________(1/1)
rev_aux l1 (a :: l2) = app (rev l1) [a] ++ l2
```

Messages Errors Jobs

no wonder i just made it extra hard for no reason

Play (k)

43:57 / 45:16

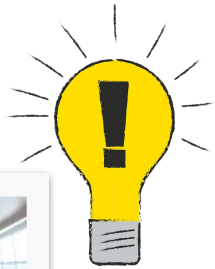
Proof Sesh 1: a taste of my own medicine

211 views • Feb 3, 2022

16 DISLIKE SHARE SAVE ...

Lemma Discovery (Task 2 of 5)

It's Hard



File Edit View Navigation Templates Queries Tools Compile Windows Help

Tactics.v

```
Qed.

Theorem rev_pres_length_alt {A : Type}:
  forall (l : list A),
    length (rev_alt l) = length l.
Proof.
  induction l; simpl; auto.
  simpl. unfold rev_alt. rewrite rev_aux_pres_length. simpl.
  auto with arith.
Qed.

Eval compute in (rev_aux [1; 2; 3; 4] [5; 6; 7; 8]).

Lemma rev_aux_app {A : Type}:
  forall (l1 l2 : list A),
    rev_aux l1 l2 = rev l1 ++ l2.
Proof.
  induction l1; intros; auto.
  simpl in *. rewrite IHl1.
  rewrite app_OK. rewrite <- app_assoc.
  reflexivity.
Qed.

Lemma rev_rev_alt {A : Type}:
  forall (l : list A),
    rev_alt l = rev l.
Proof.
  intros l. unfold rev_alt.
  rewrite rev_aux_app. apply List.app_nil_r.
Qed.

(*
 * That's it for now! You can keep playing with other proofs if you have
 * extra time. For example, it might be fun to define your own inductive types,
 * like a binary tree, and then write proofs about those types.
 *
 * If you have a lot of extra time, I recommend looking at the
 * Coq source code on Github to get a sense for how it's implemented:
 * https://github.com/coq/coq. In any case, when we have 25 minutes
 * left of class, please do this:
 *
 *  1.  r group and discuss the question below.
 *  2.  nswer—just one answer for your group, clearly indicating
 *      all members of the group. (If you are not here, and are working alone,
```

Messages Errors Jobs

Pause (k)

44:45 / 45:16

Proof Sesh 1: a taste of my own medicine

211 views • Feb 3, 2022

16 DISLIKE SHARE SAVE ...

Lemma Discovery (Task 2 of 5)

Good automation ought to help us find these lemmas.

Lemma Discovery (Task 2 of 5)

What's Hard

Static data already has the **perfect lemmas**, but hides the **discovery process**.

Lemma Discovery (Task 2 of 5)

Ideas 💡

Learn to predict those perfect lemmas anyways?

Lemma Discovery (Task 2 of 5)

Ideas 💡

Learn to predict lemmas that mirror the structure of the definitions and programs?

Lemma Discovery (Task 2 of 5)

Ideas 💡 💡

More granular data via instrumentation, capturing the development process?

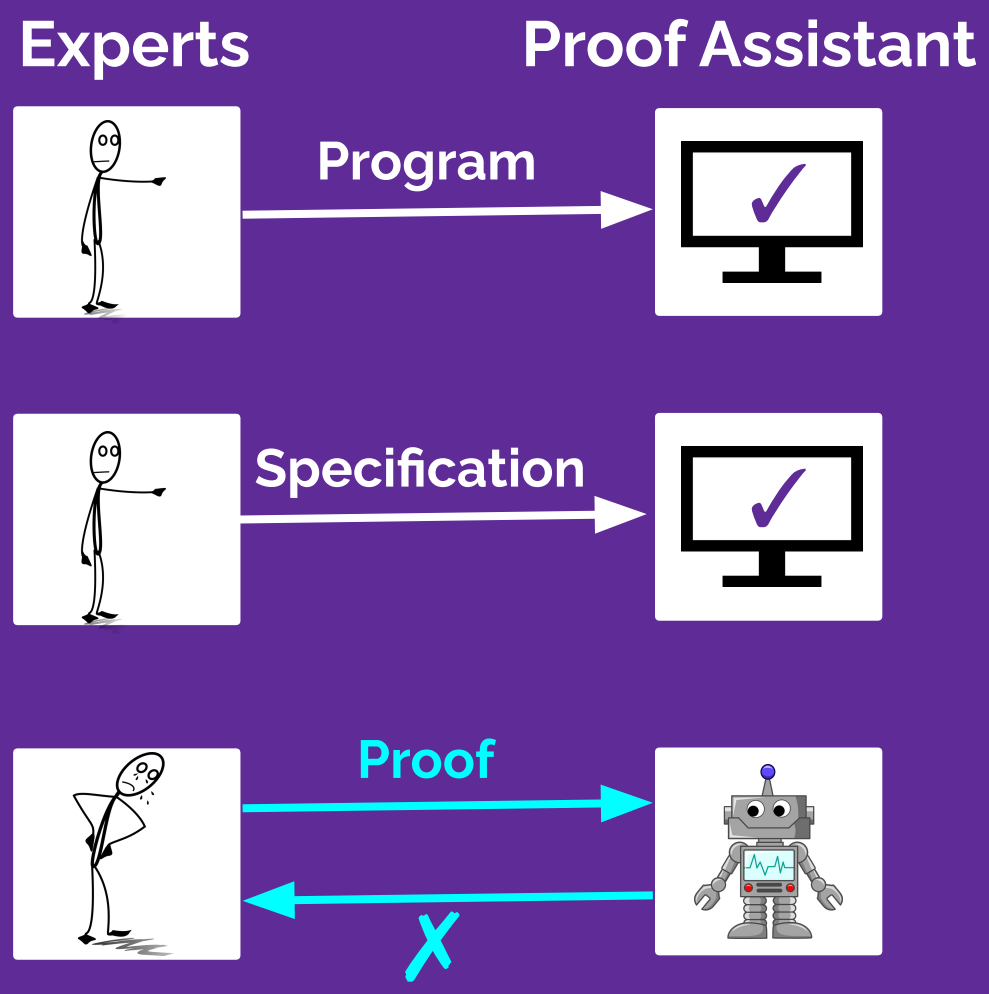
Lemma Discovery (Task 2 of 5)

Ideas 💡

Your ideas here!

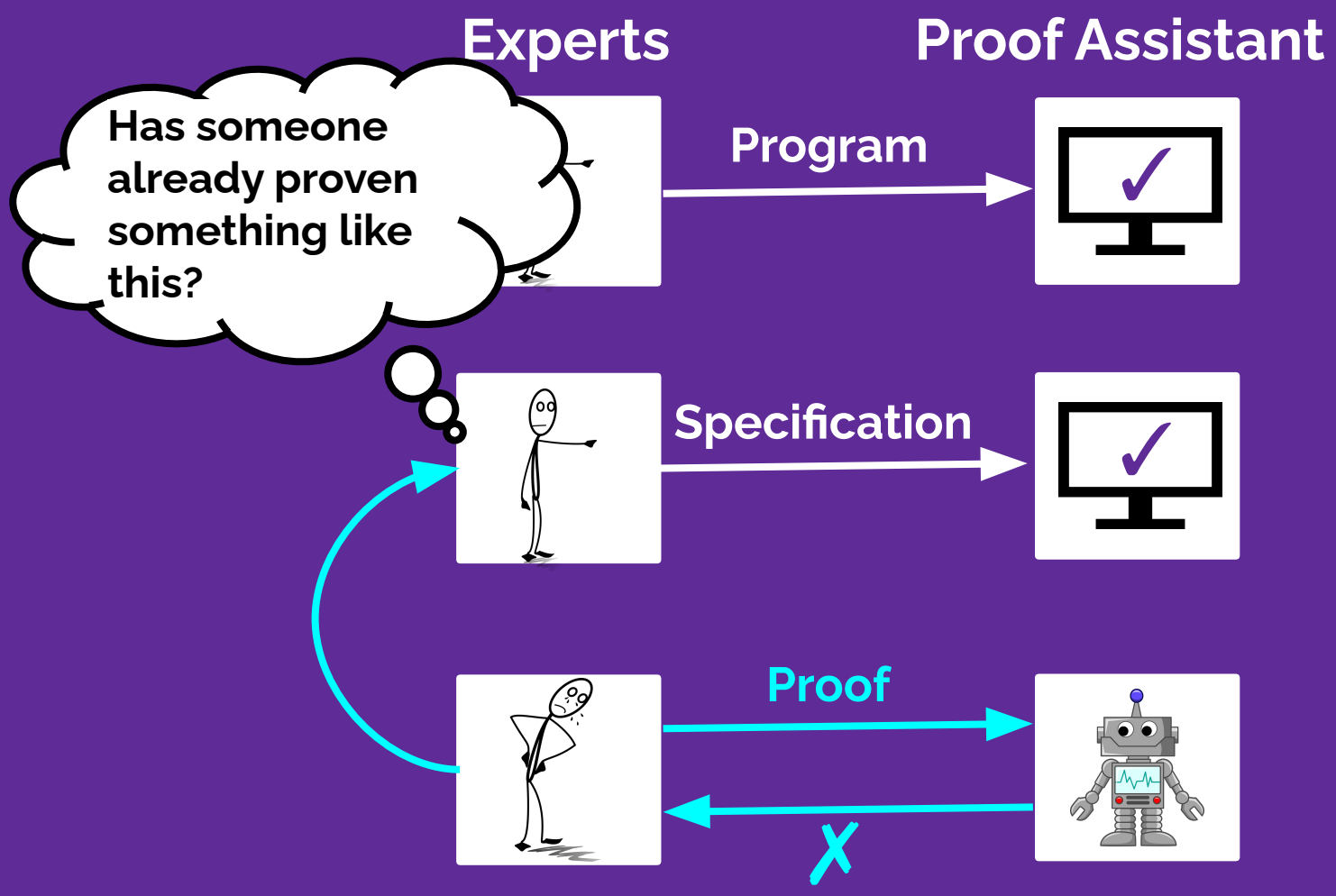
Lemma Discovery (Task 2 of 5)

Experts are Mere Mortals



Lemma Discovery (Task 2 of 5)

Experts are Mere Mortals



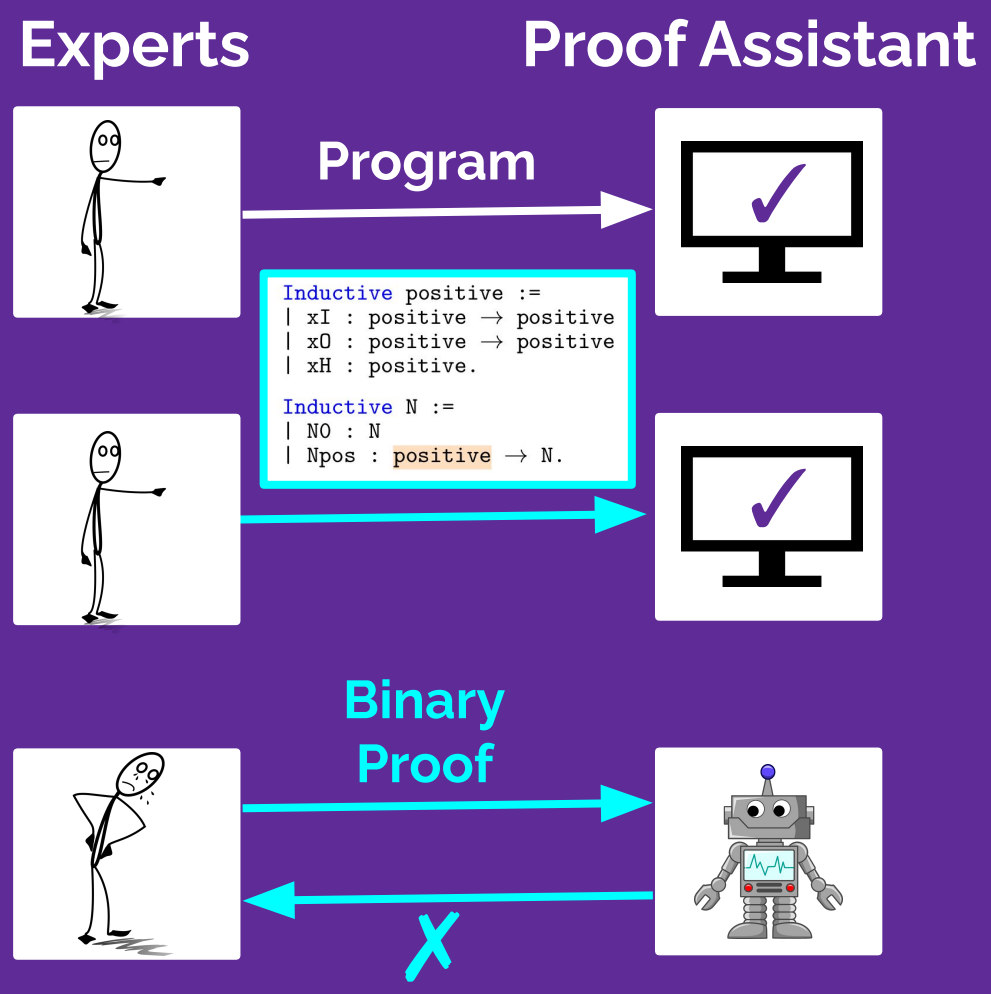
Lemma Discovery (Task 2 of 5)

Do lots more:

1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

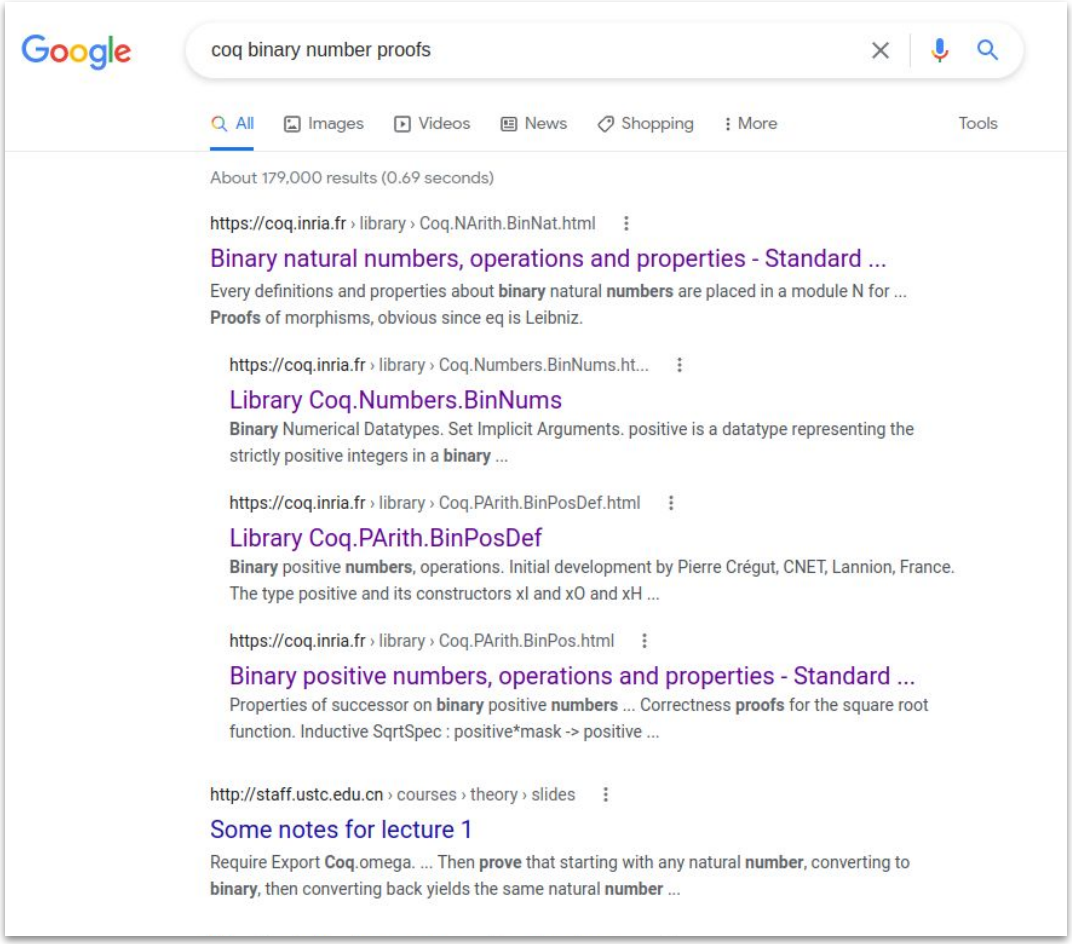
You won't regret it!

Experts are Mere Mortals



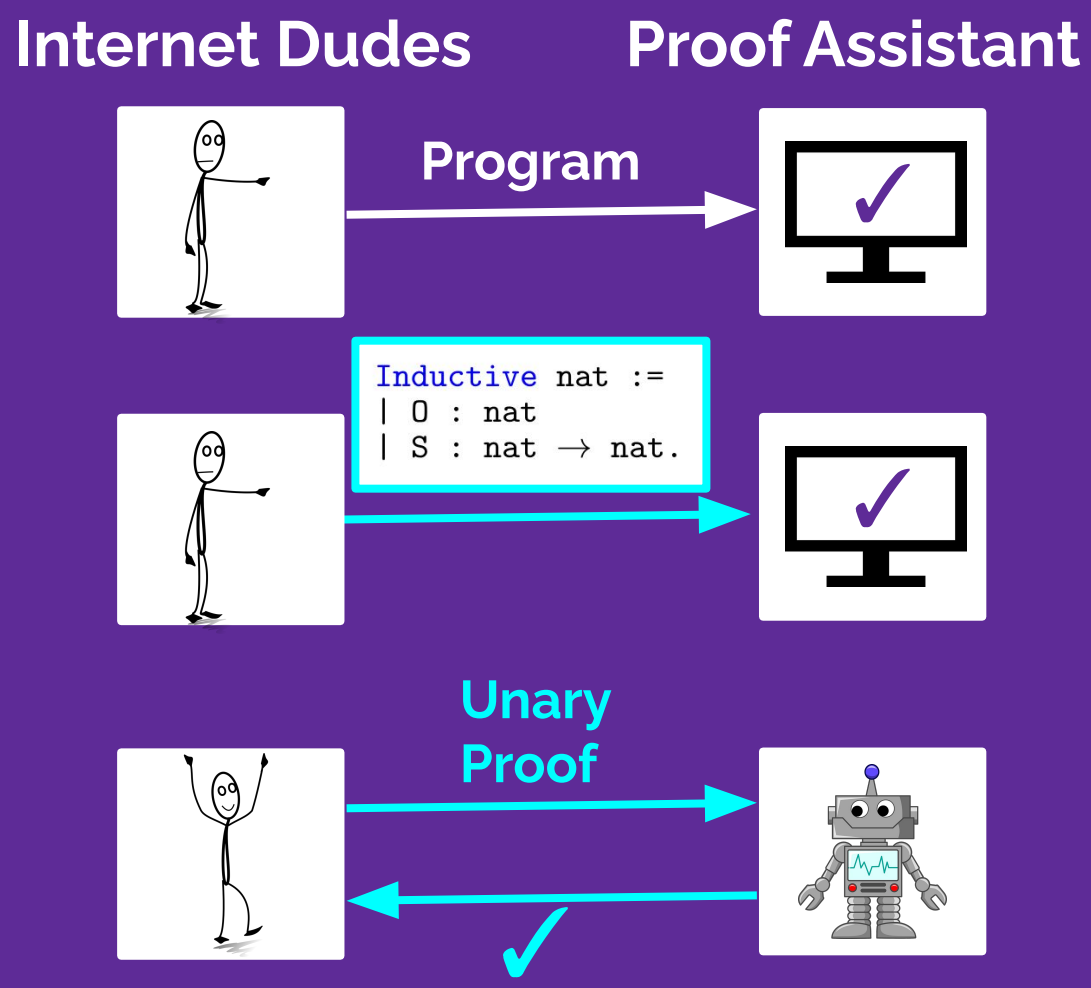
Relation Discovery (Task 3 of 5)

Experts are Mere Mortals



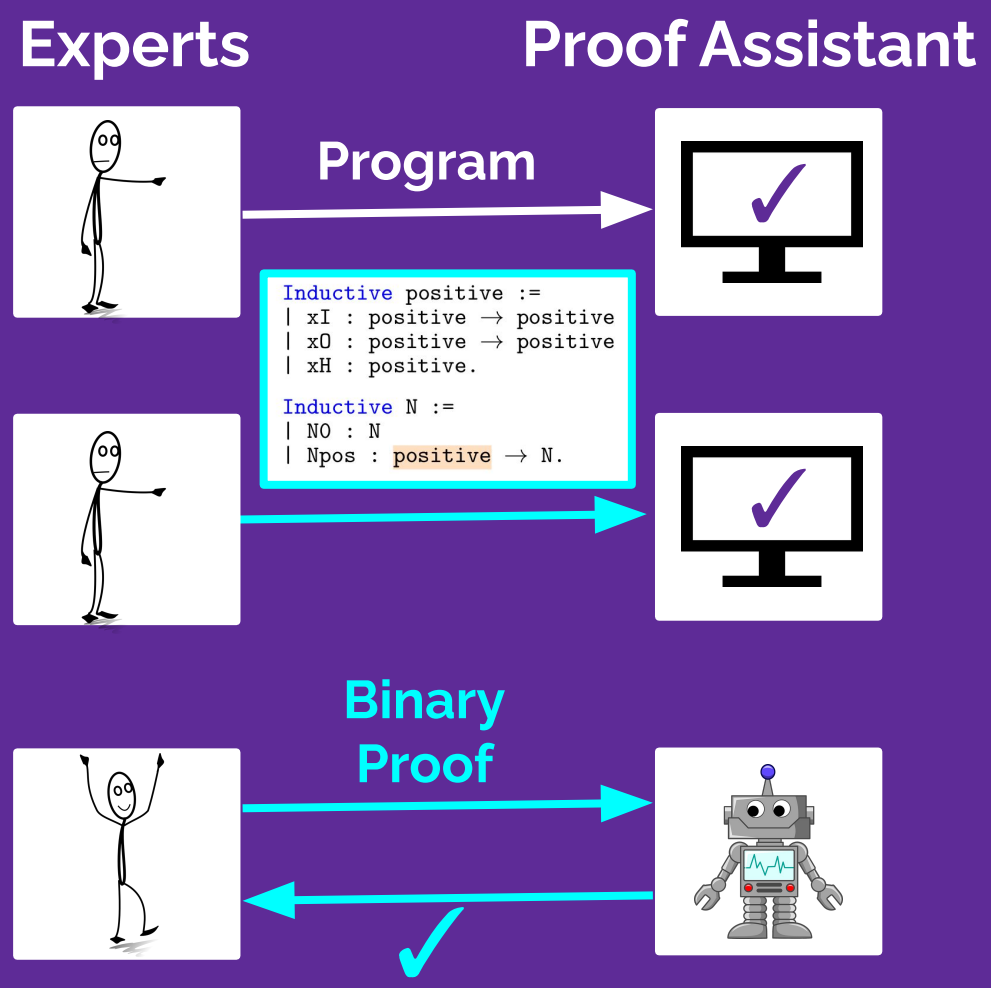
Relation Discovery (Task 3 of 5)

Close Enough?



Relation Discovery (Task 3 of 5)

Close Enough?



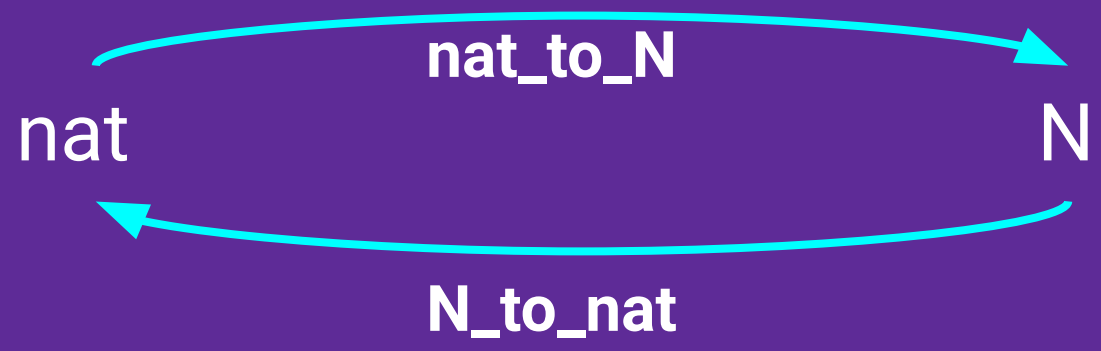
Relation Discovery (Task 3 of 5)

Close Enough!

```
Inductive nat :=
| O : nat
| S : nat → nat.
```

```
Inductive positive :=
| xI : positive → positive
| xO : positive → positive
| xH : positive.

Inductive N :=
| N0 : N
| Npos : positive → N.
```



Relation Discovery (Task 3 of 5)

It's Hard

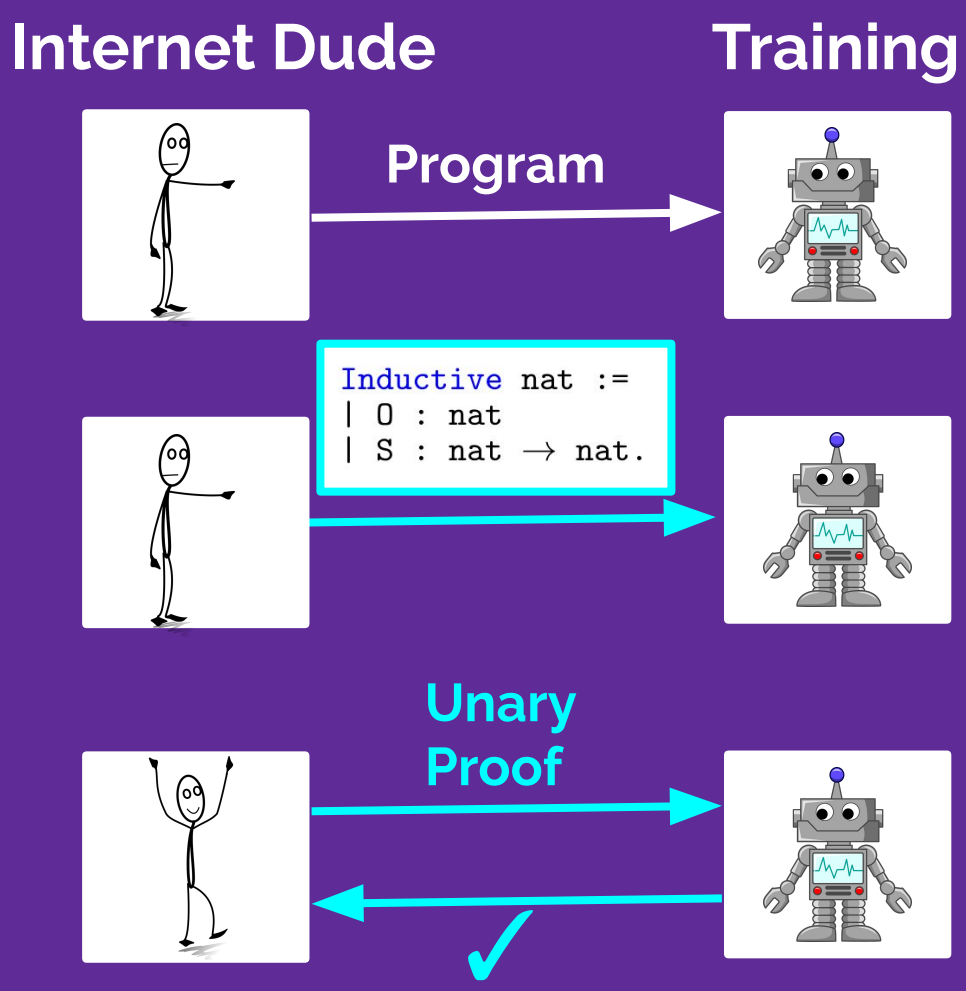

```
Inductive nat :=
| O : nat
| S : nat → nat.
```

```
Inductive positive :=
| xI : positive → positive
| xO : positive → positive
| xH : positive.

Inductive N :=
| N0 : N
| Npos : positive → N.
```

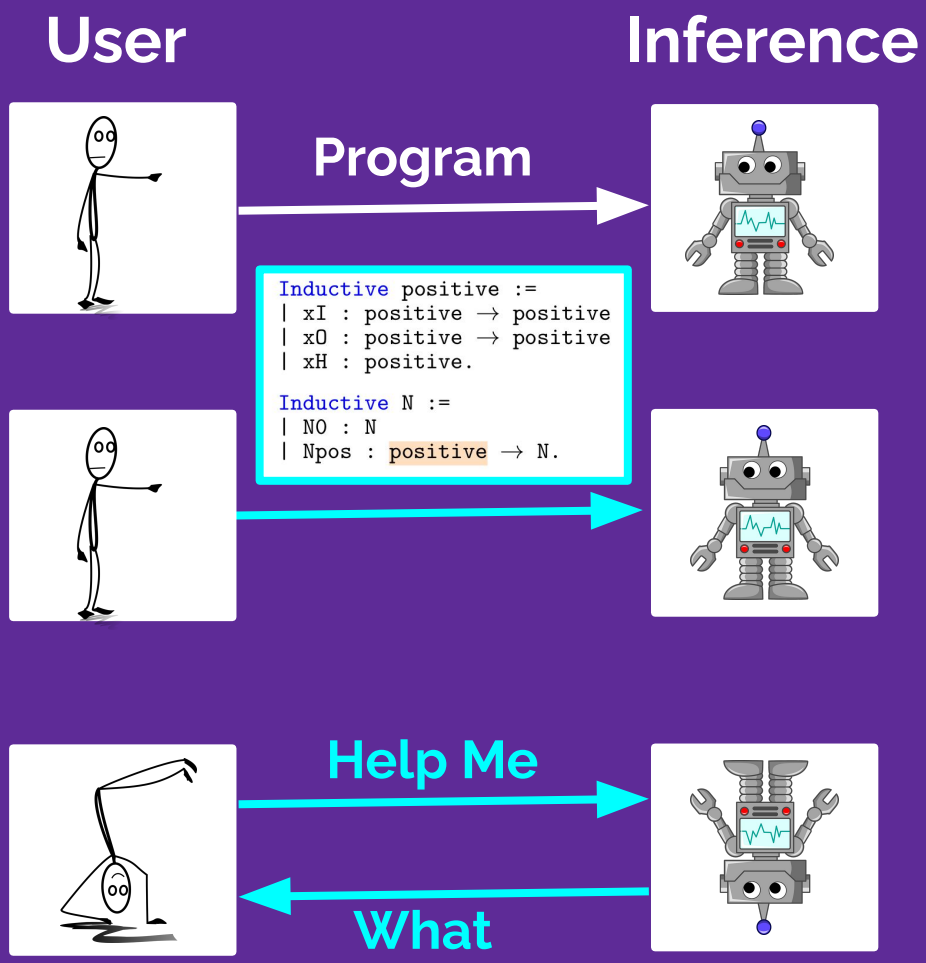
Relation Discovery (Task 3 of 5)

It's Hard



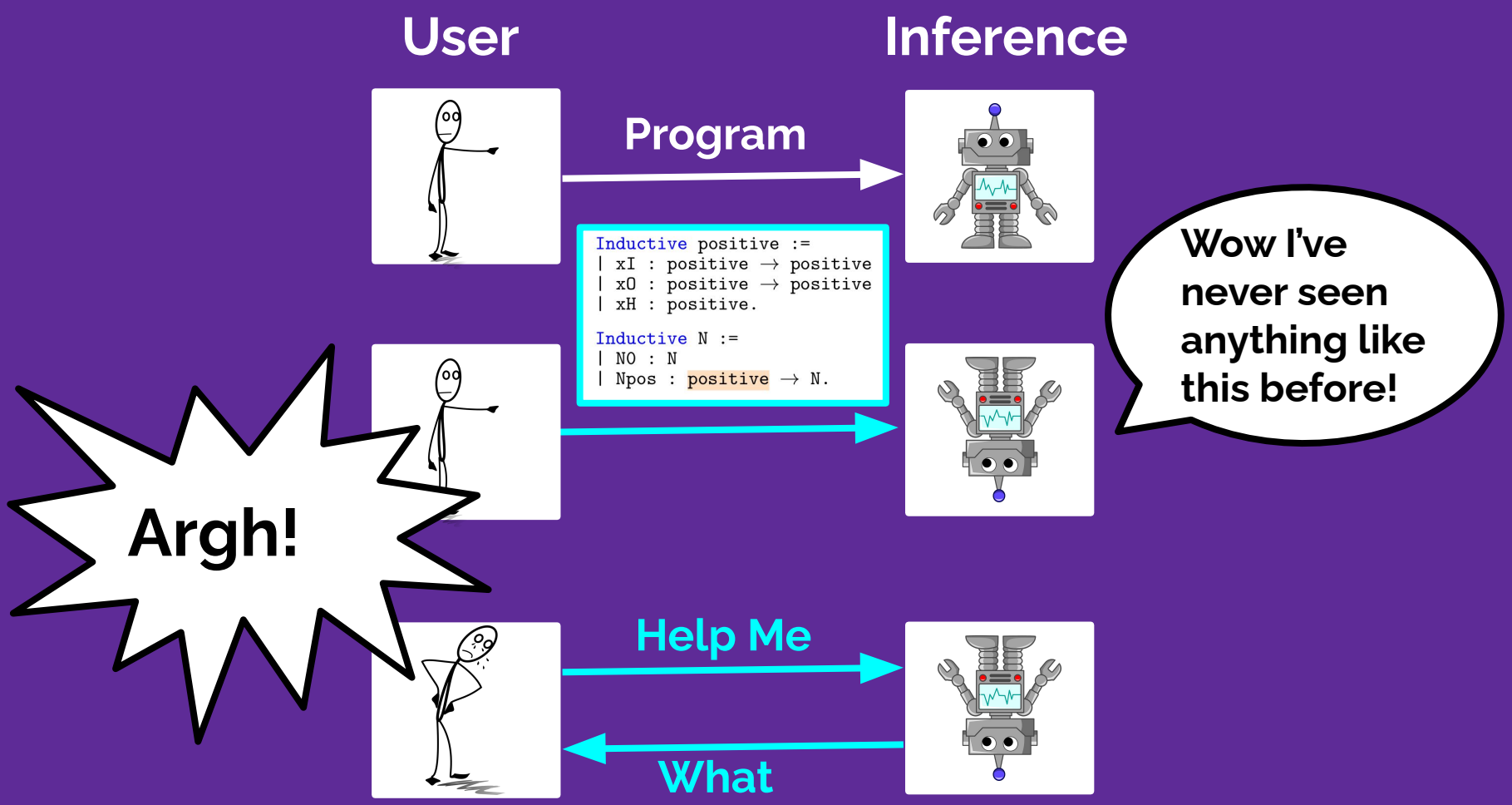
Relation Discovery (Task 3 of 5)

It's Hard



Relation Discovery (Task 3 of 5)

It's Hard



Relation Discovery (Task 3 of 5)

AlphaGo lost a match
because of this kind of thing.

Relation Discovery (Task 3 of 5)

In the world of proofs, small changes can break everything.

Relation Discovery (Task 3 of 5)

Good automation ought to discover general relations.

Relation Discovery (Task 3 of 5)

What's Hard

These relations are often not explicitly stated, and often lack reasonable syntactic proxies.

Relation Discovery (Task 3 of 5)

Ideas 💡

Force the user to write a few example functions or proofs over the new datatype?

Relation Discovery (Task 3 of 5)

Ideas 💡

Semantically embed
types and their relations?

Relation Discovery (Task 3 of 5)

Ideas 💡

Allow the model to play with the datatypes?

Relation Discovery (Task 3 of 5)

Ideas 💡 💡

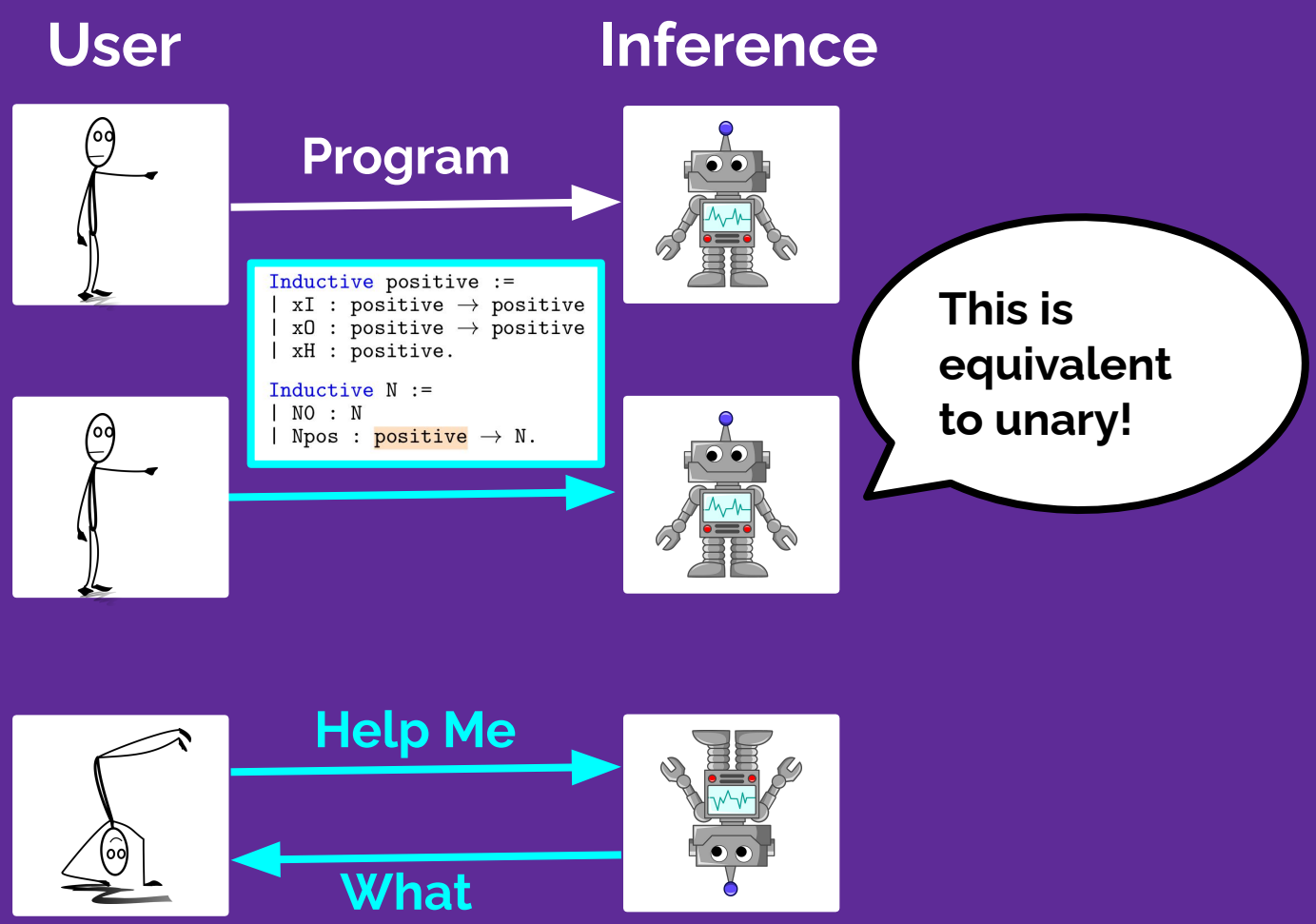
Combine with
property-based testing?

Relation Discovery (Task 3 of 5)

Ideas 💡

Your ideas here!

So What?



Relation Discovery (Task 3 of 5)

Do lots more:

1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

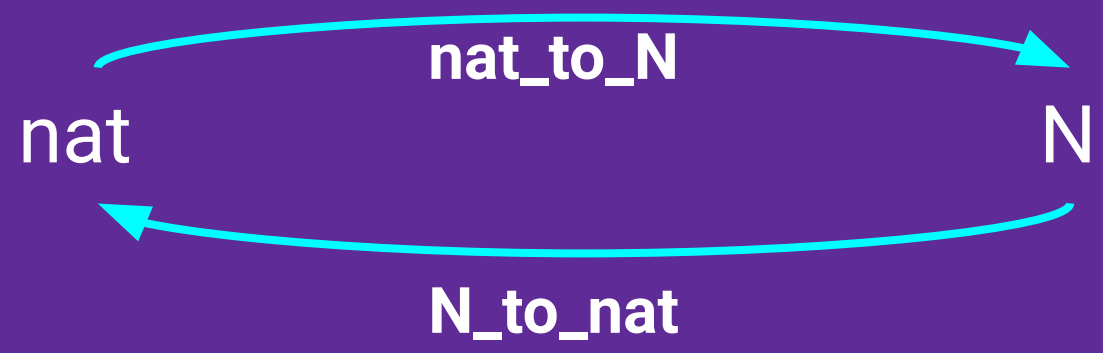
You won't regret it!

So Far

```
Inductive nat :=
| O : nat
| S : nat → nat.
```

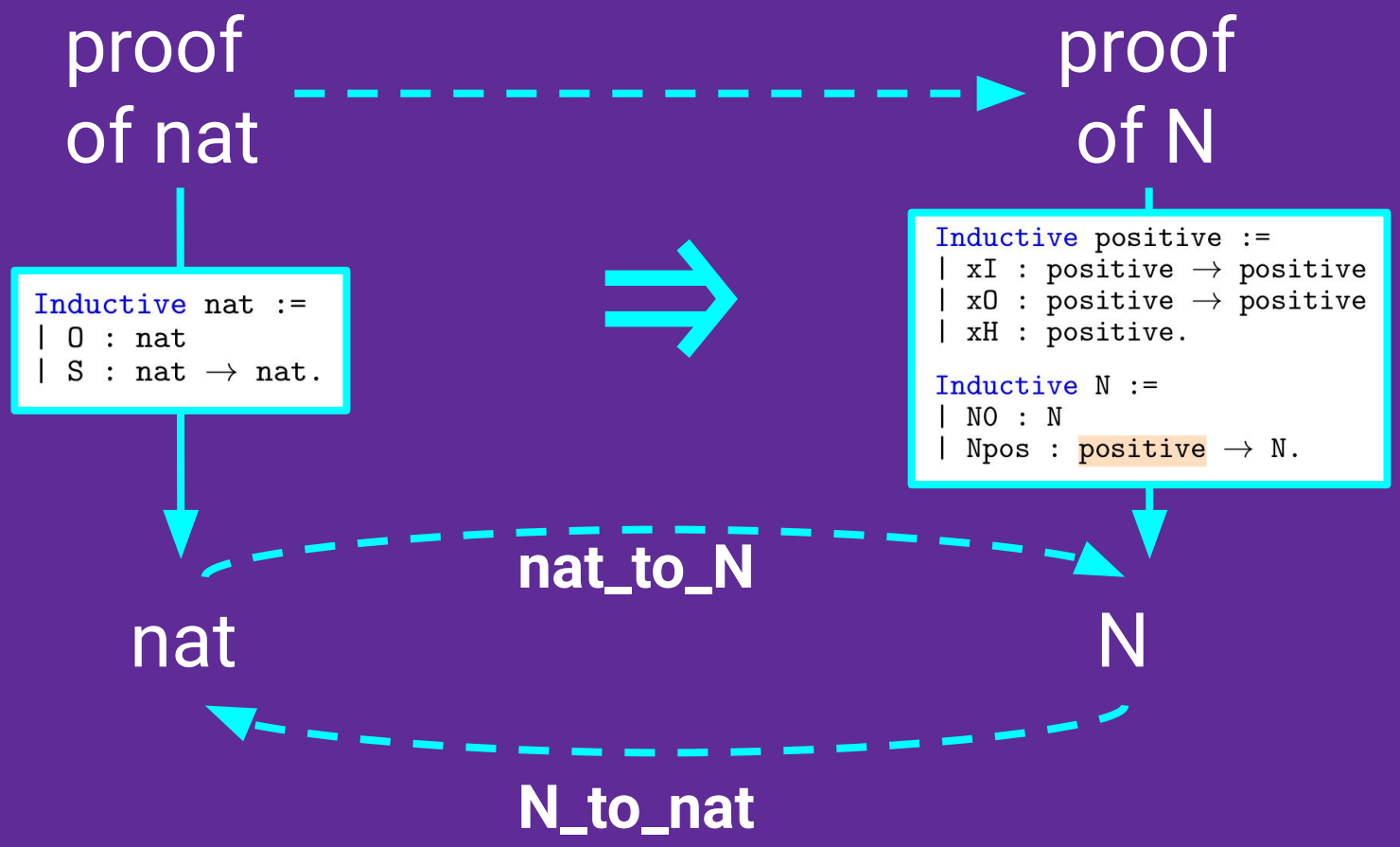
```
Inductive positive :=
| xI : positive → positive
| xO : positive → positive
| xH : positive.

Inductive N :=
| N0 : N
| Npos : positive → N.
```



Proof Reuse & Repair (Task 4 of 5)

So Far



Proof Reuse & Repair (Task 4 of 5)

So Far

proof
of N

```
Inductive positive :=
| xI : positive → positive
| xO : positive → positive
| xH : positive.

Inductive N :=
| N0 : N
| Npos : positive → N.
```

N

Proof Reuse & Repair (Task 4 of 5)

It's Hard

- Still too manual
- Just equivalences
- Only experts can extend
- No human-friendly proof scripts

Proof Reuse & Repair (Task 4 of 5)

Good automation ought to adapt proofs to change.

Proof Reuse & Repair (Task 4 of 5)

Imagine

- Only the necessary human input
- Any kind of relation
- Not gated by experts
- Human-friendly proof scripts

Proof Reuse & Repair (Task 4 of 5)

What's Hard

Most data sources hide atomic edits, plus literally nobody knows how to solve this yet for some classes of changes.

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

Type theory work to better understand classes of changes, like quotient equivalences.

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡 💡 💡

More granular data via instrumentation, capturing the development process?

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

Train model to break down less granular repairs?

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

More datasets capturing synthetic repair data?

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

More datasets capturing existing public repair data?

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

More datasets!

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

Allow the model to play with and learn from symbolic proof repair tools?

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

Embed repairs across equivalences as paths in higher-dimensional spaces, a la cubical type theory (so that proof repair becomes path finding)?

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

Use e-graphs to compress knowledge about equivalences?

Proof Reuse & Repair (Task 4 of 5)

Ideas 💡

Your ideas here!

Proof Reuse & Repair (Task 4 of 5)

New Frontiers

Even cooler: crawl proof corpora and construct a knowledge graph.

Proof Reuse & Repair (Task 4 of 5)

Even cooler: combine with something like a search engine.

Proof Reuse & Repair (Task 4 of 5)

Voevodsky's vision (IAS Memorial):

Imagine "mathematicians around the world could collaborate by depositing proofs and constructions in the computer, and … it would be up to the computer to locate the equivalence between formulations and [to] transport the constructions from one context to another."

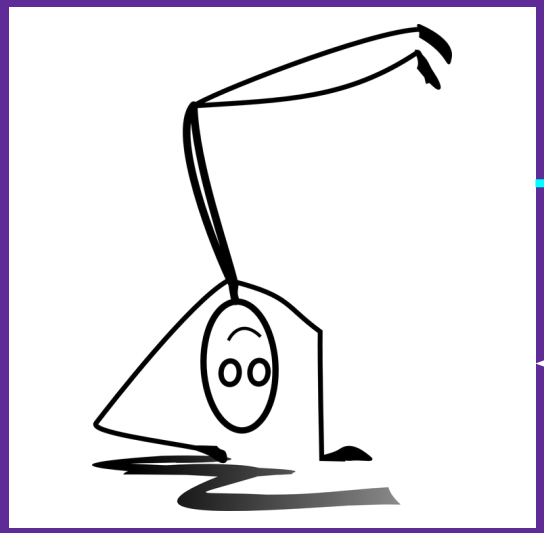
Proof Reuse & Repair (Task 4 of 5)

Do lots more:

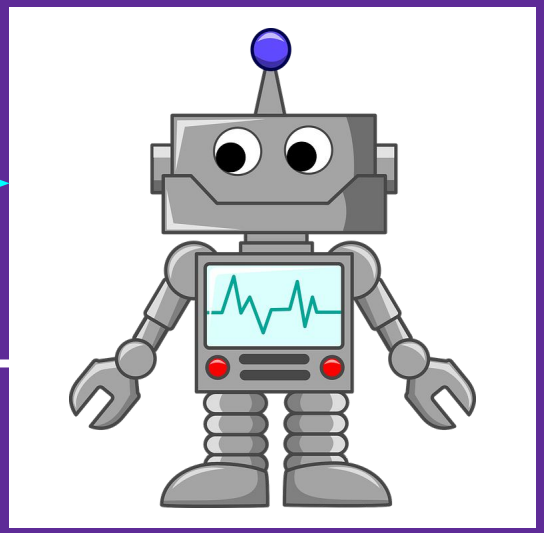
1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

You won't regret it!

So Far



Search

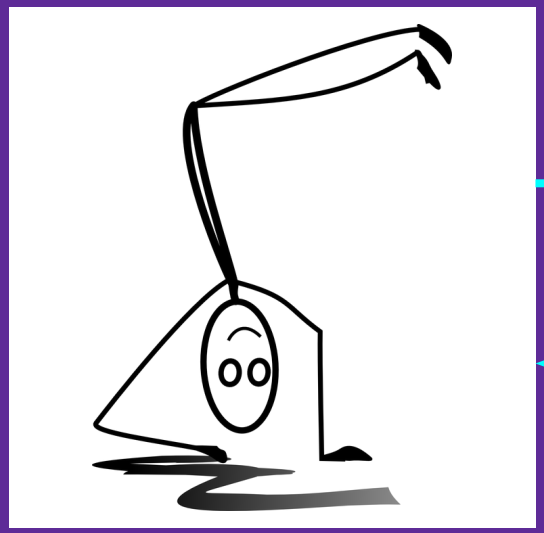


Proof Engineer

Proof Assistant

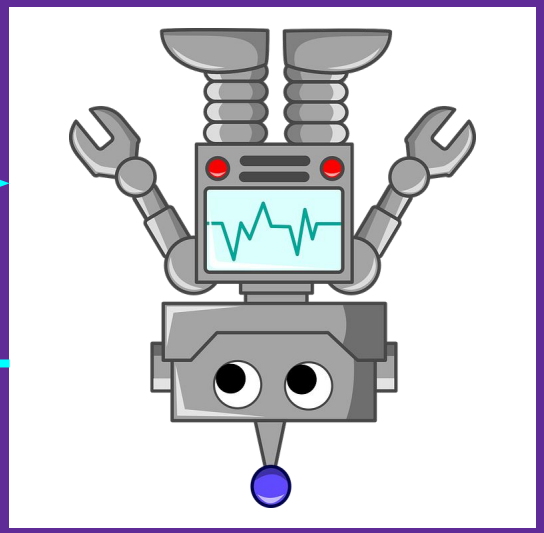
Semantic Search (Task 5 of 5)

So Far



Search →

← Godspeed



Proof Engineer

Proof Assistant

Semantic Search (Task 5 of 5)

So Far

Search is the #1 complaint in my proof automation class.

Semantic Search (Task 5 of 5)

Imagine

- Online
- Semantics-aware
- Context-and-goal-aware
- Mixed natural and formal queries

Semantic Search (Task 5 of 5)

What's Hard

We need to develop an internet of proofs.

Semantic Search (Task 5 of 5)

Ideas 💡

Build the internet of proofs:
an online, shared, updating, cross-language database of all of the proof data we have.

Semantic Search (Task 5 of 5)

Let's bring this all together.

Imagine:
A Proof Search Engine

A Proof Search Engine

- Online
- Semantics-aware
- Context-and-goal-aware
- Mixed natural and formal queries

A Proof Search Engine

- Refine natural language to specs
- Create or discover helper lemmas
- Find and adapt existing proofs
- Discover relevant proof strategies

A Proof Search Engine

It'll draw on:

1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

A Proof Search Engine

It'll help us with:

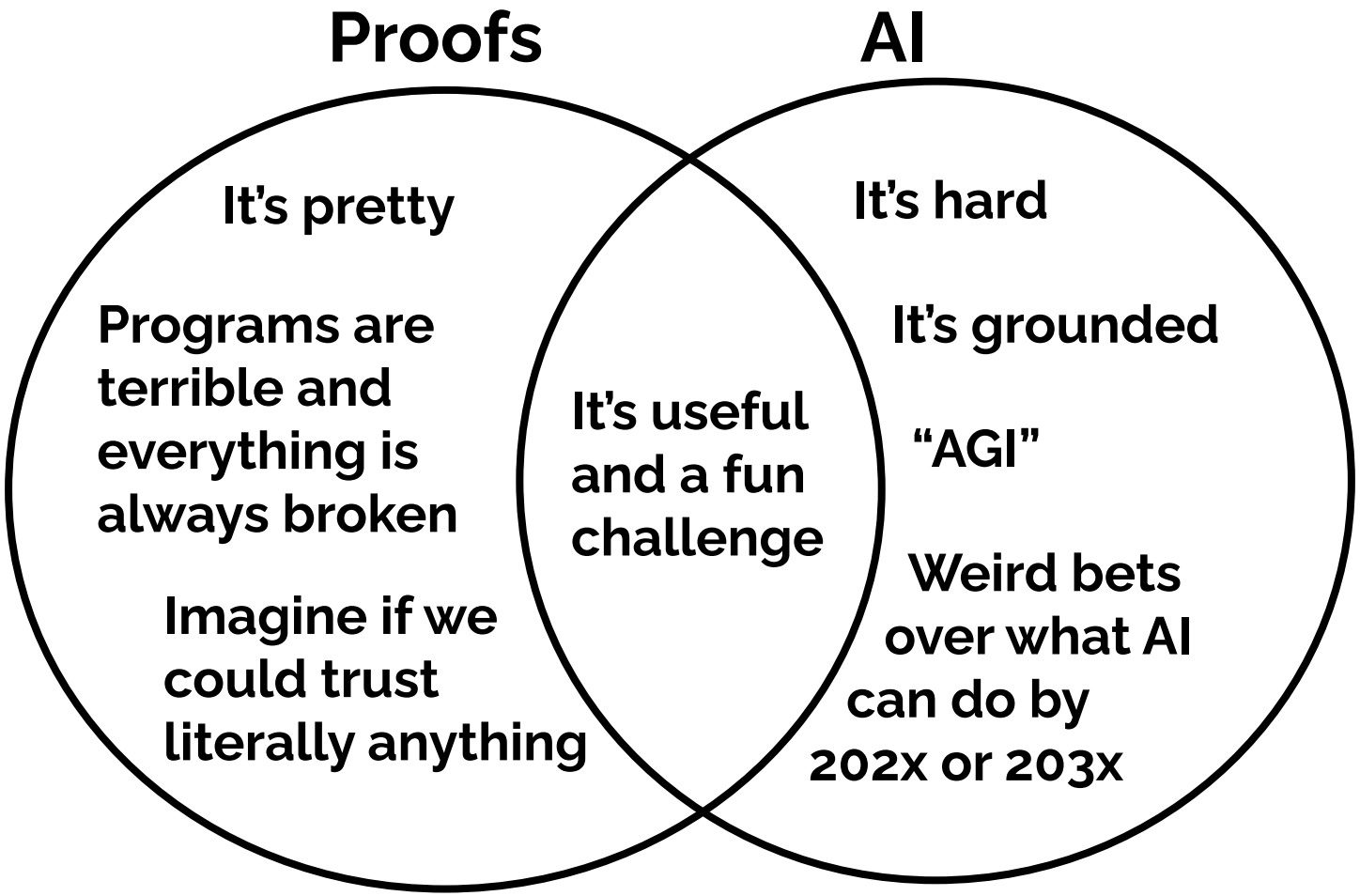
1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, **and**
5. semantic search.

A Proof Search Engine

It'll help us with:

1. tactic prediction,
2. synthesis,
3. autoformalization,
4. premise selection, and
5. concept alignment.

No Matter the Motivation



Natural allies with a common cause.

Let's do lots more:

1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

We won't regret it!