

Concrete Problems in Proof Automation

Talia Ringer

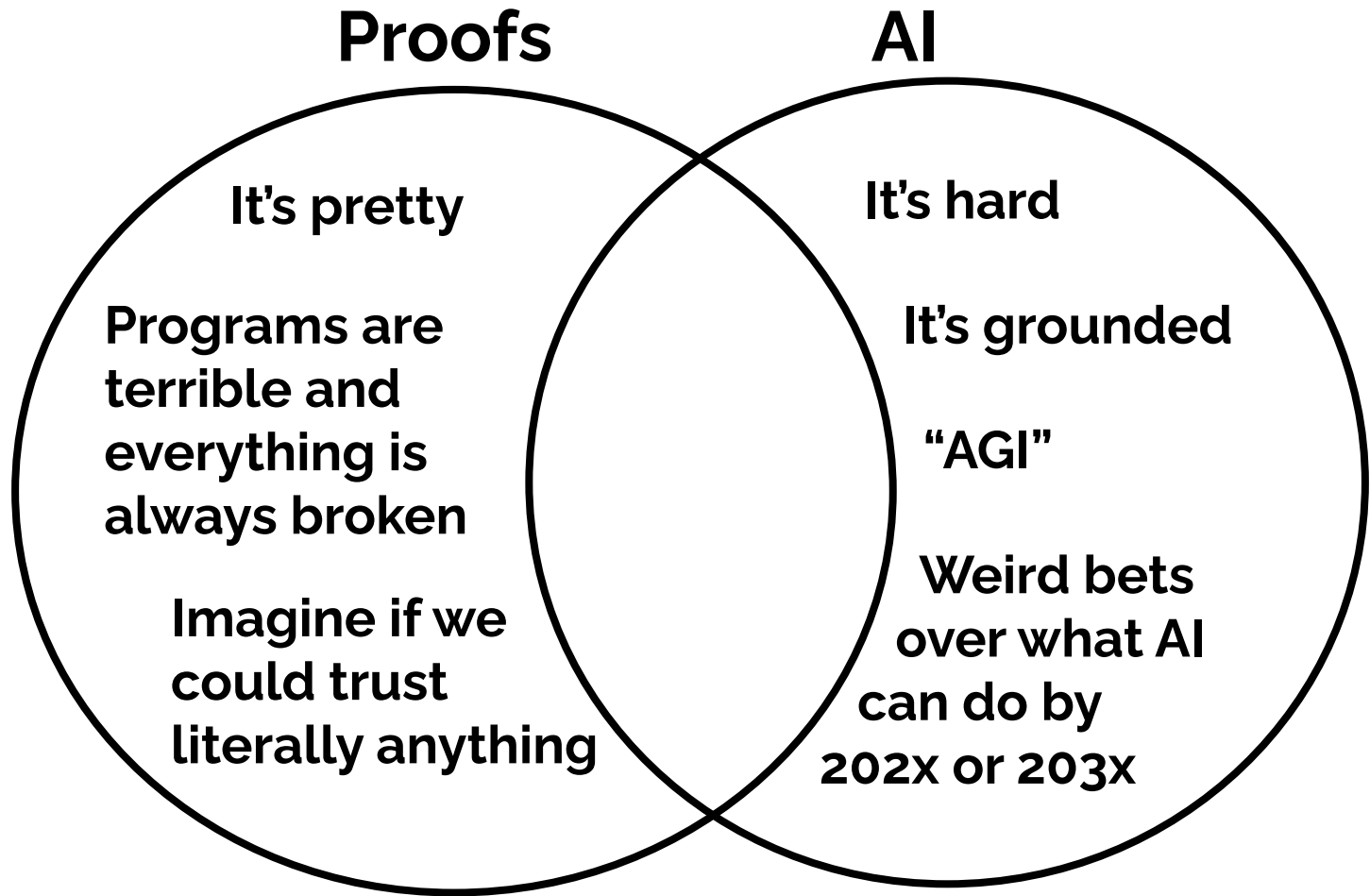
UIUC (currently also visiting at Google)



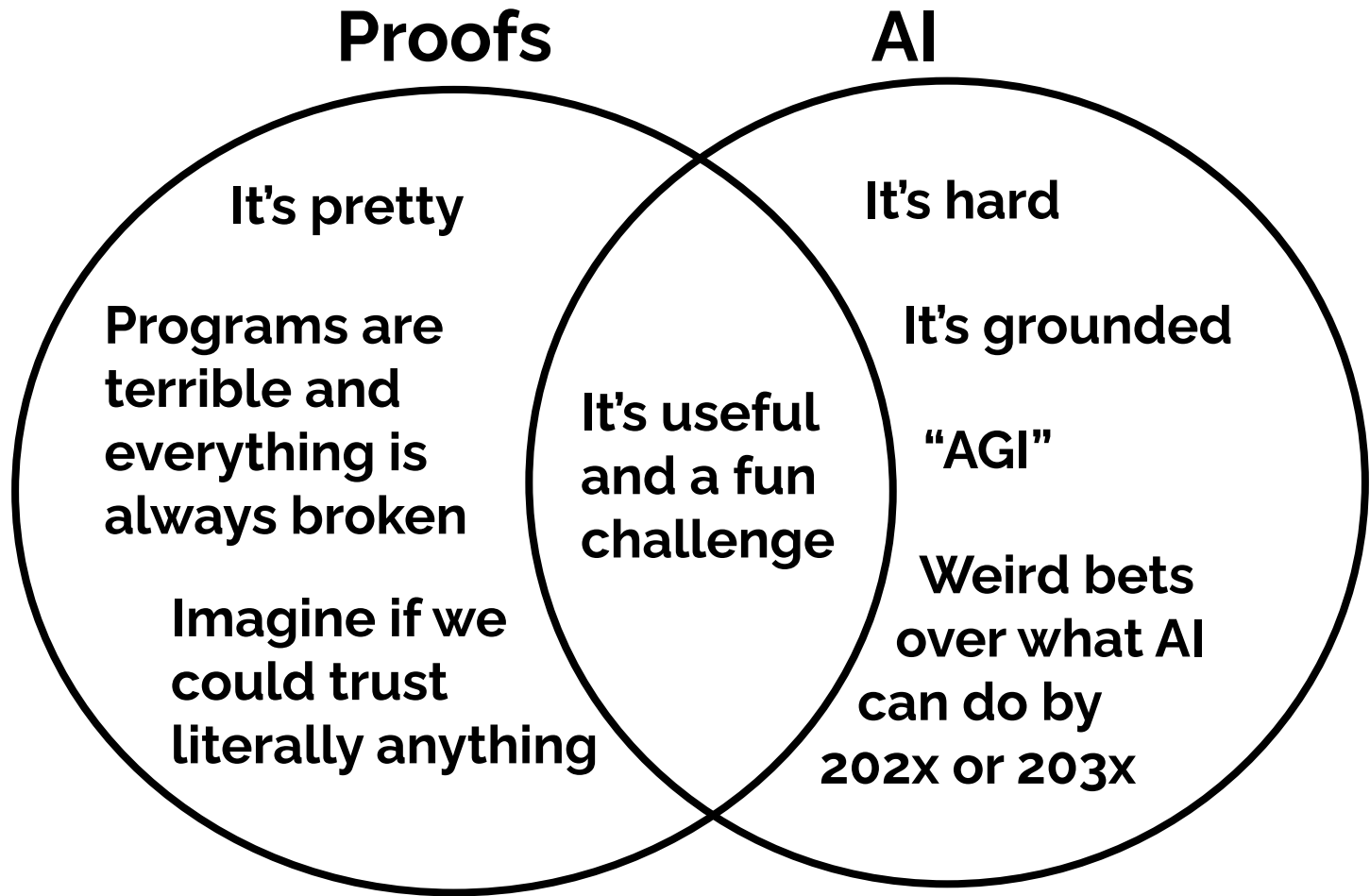
The **long version** of this
talk from AITP 2022
is on my **YouTube!**

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:

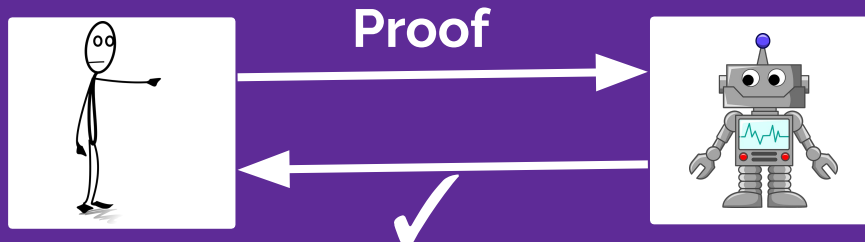
- 1. conjecture testing,**
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

You won't regret it!

Myth

Proof Engineer

Proof Assistant

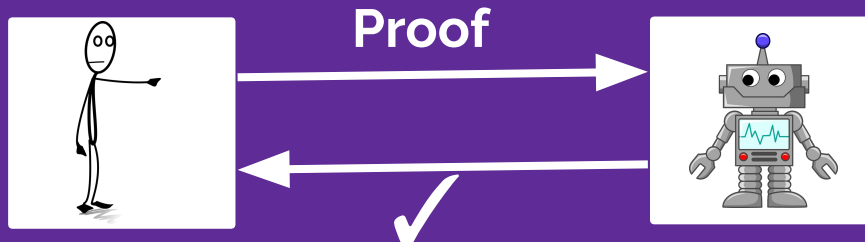


Conjecture Testing (Task 1 of 5)

Myth

Proof Engineer

Proof Assistant

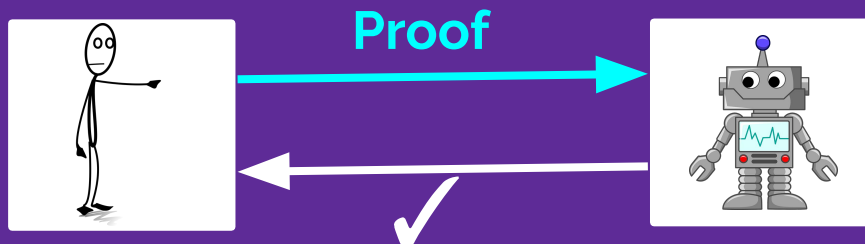


Conjecture Testing (Task 1 of 5)

Myth

Proof Engineer

Proof Assistant

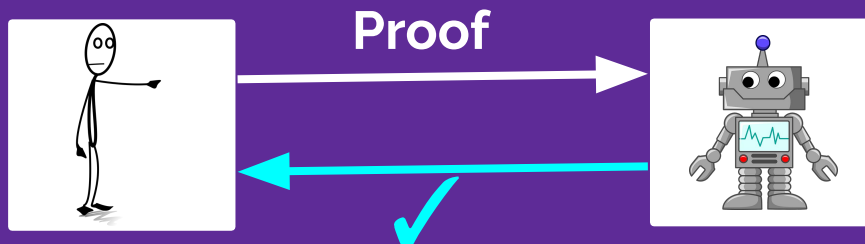


Conjecture Testing (Task 1 of 5)

Myth

Proof Engineer

Proof Assistant

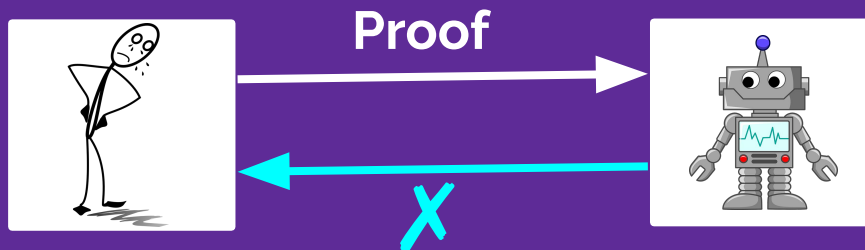


Conjecture Testing (Task 1 of 5)

Reality

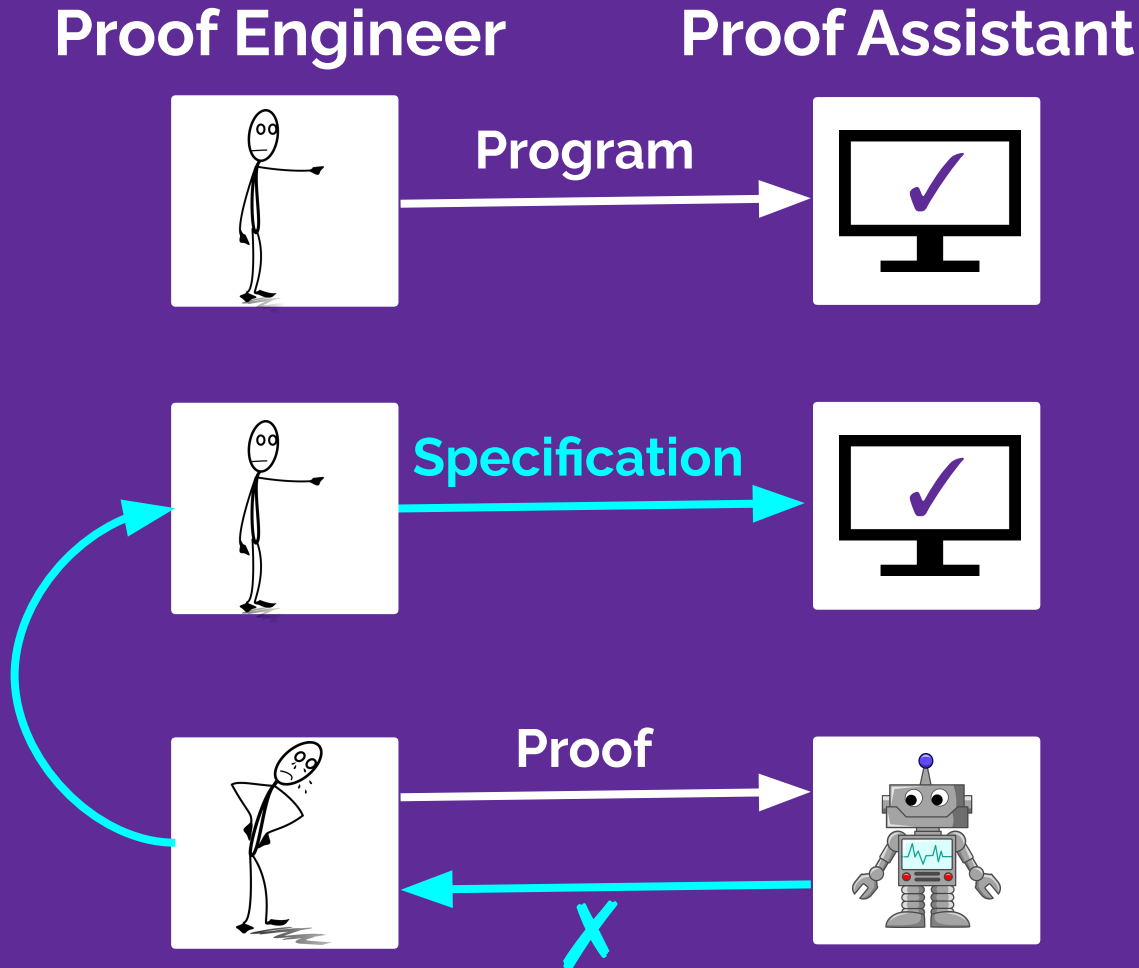
Proof Engineer

Proof Assistant



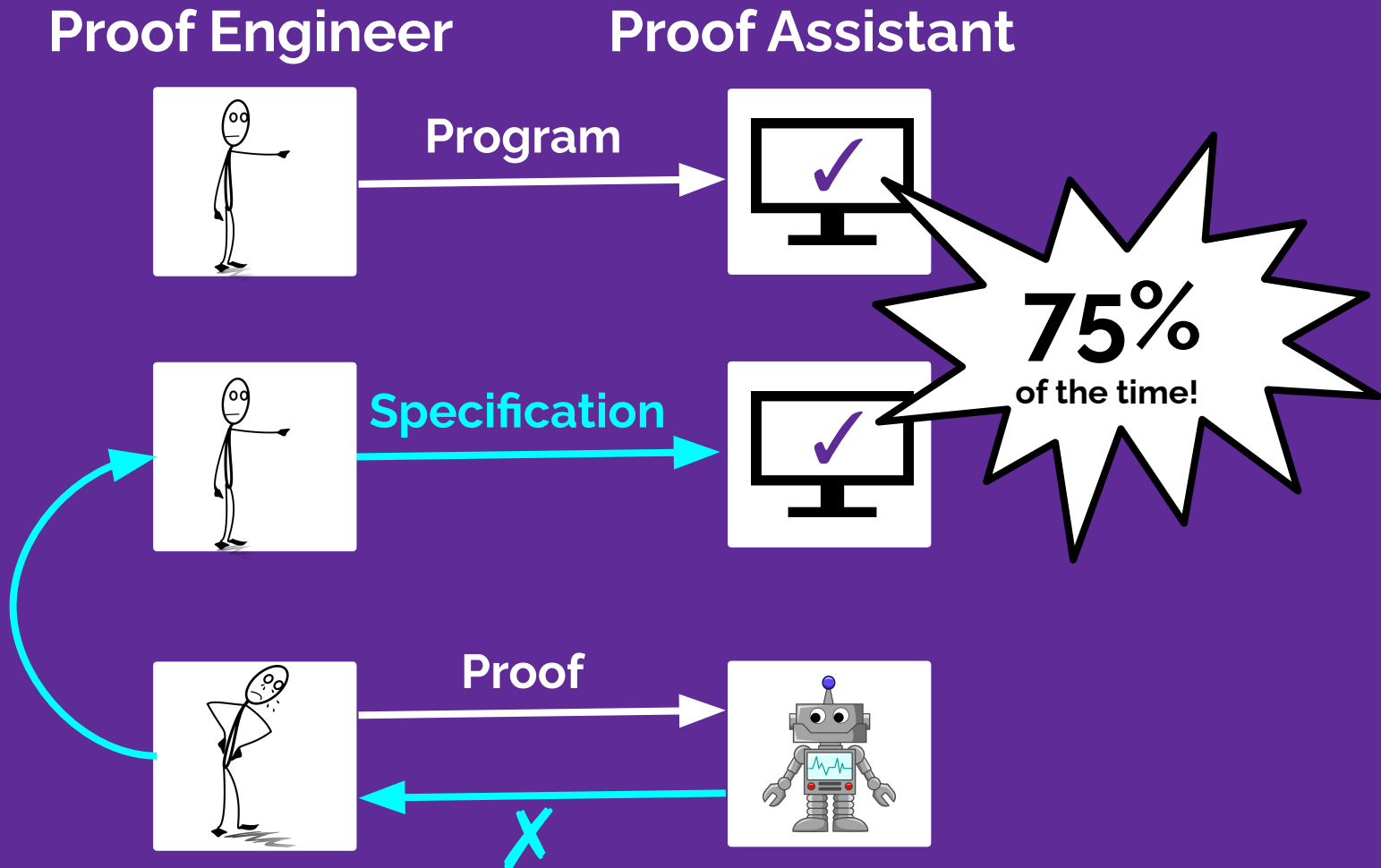
Conjecture Testing (Task 1 of 5)

Reality



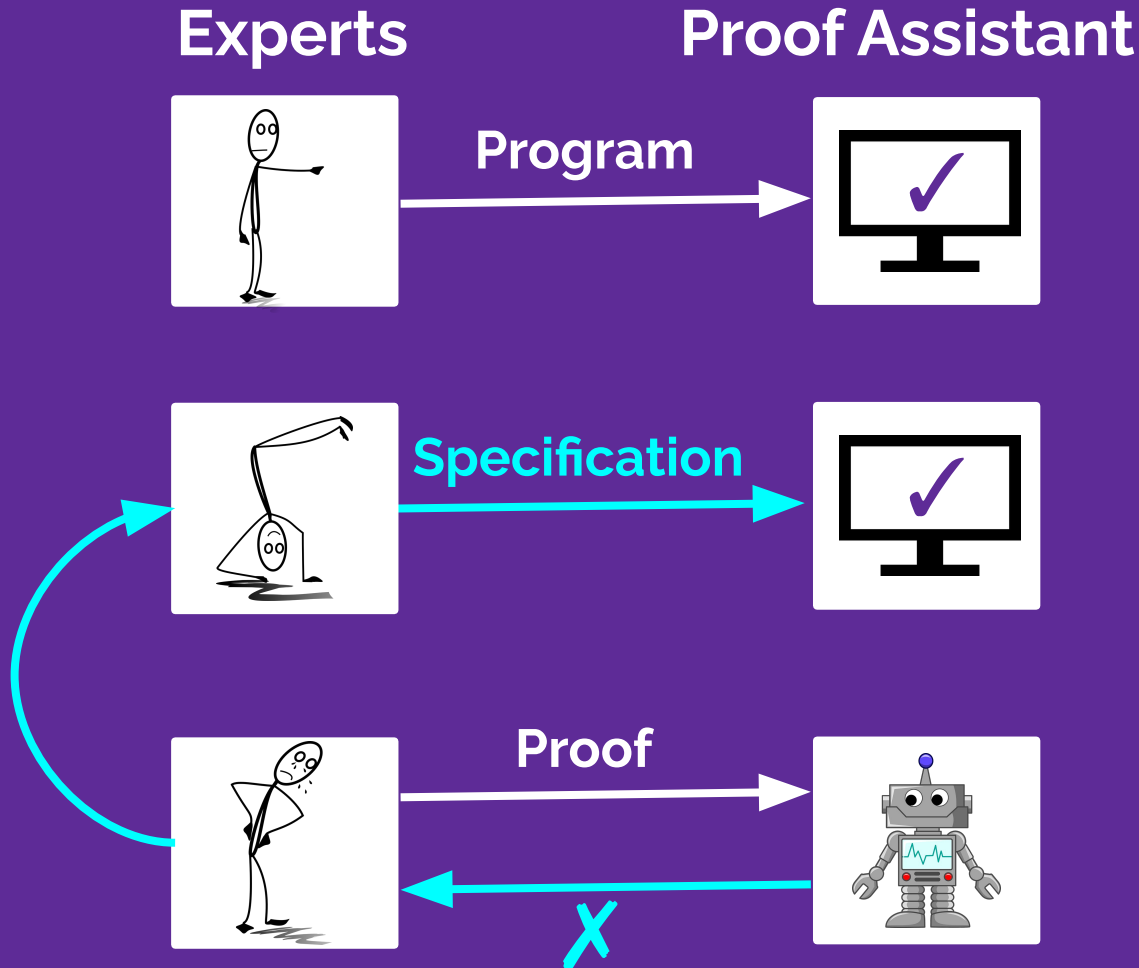
Conjecture Testing (Task 1 of 5)

Reality



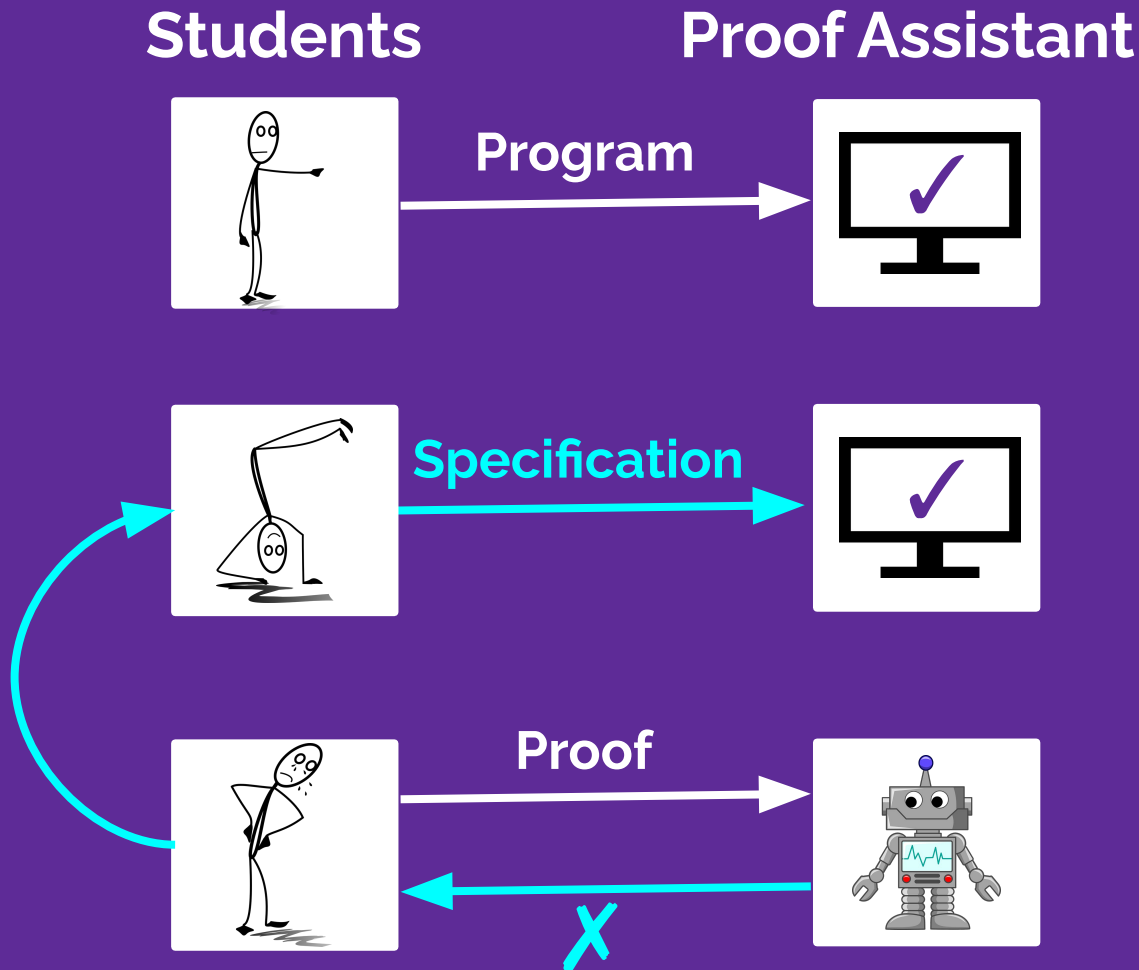
Conjecture Testing (Task 1 of 5)

It's Hard



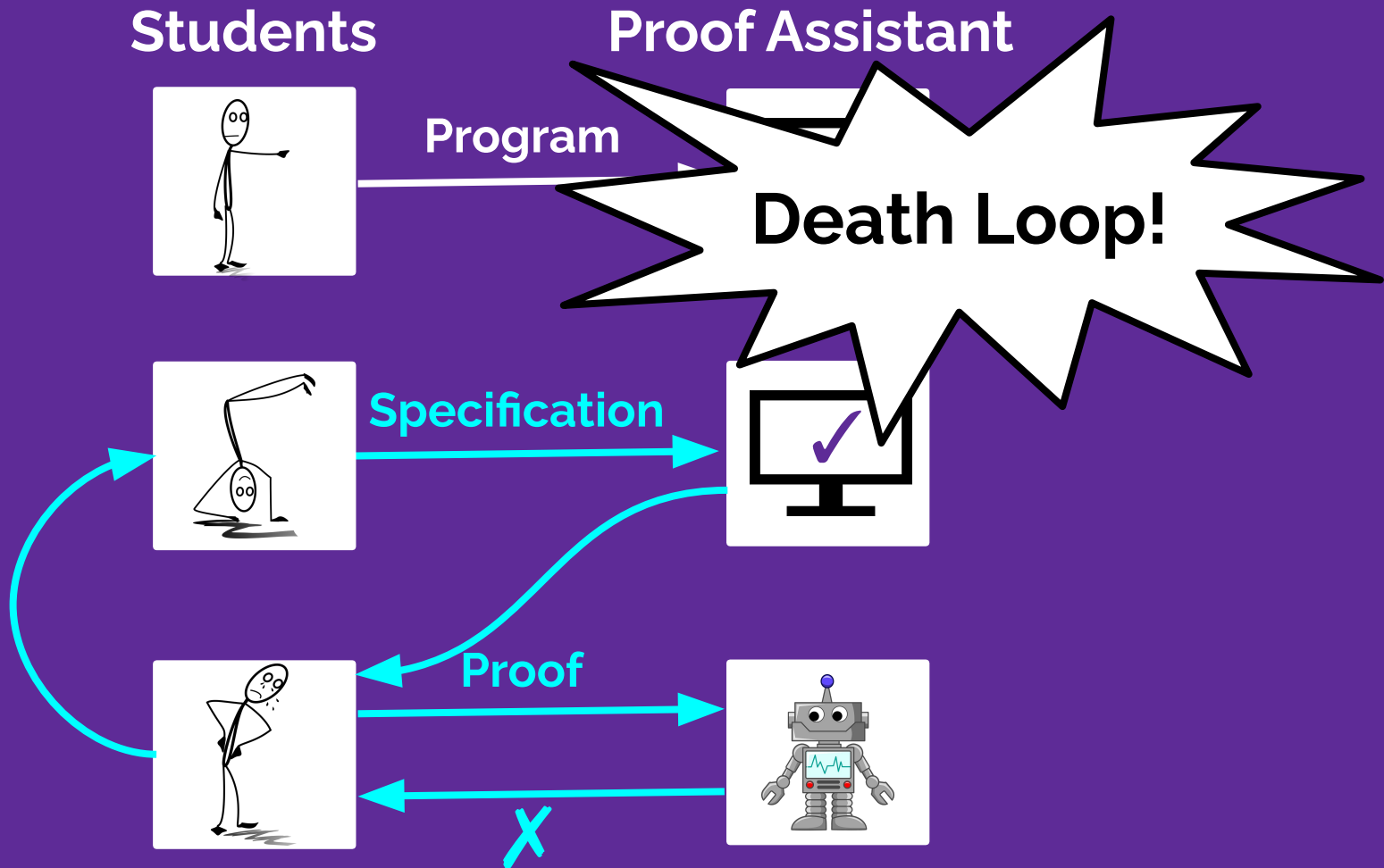
Conjecture Testing (Task 1 of 5)

It's Hard



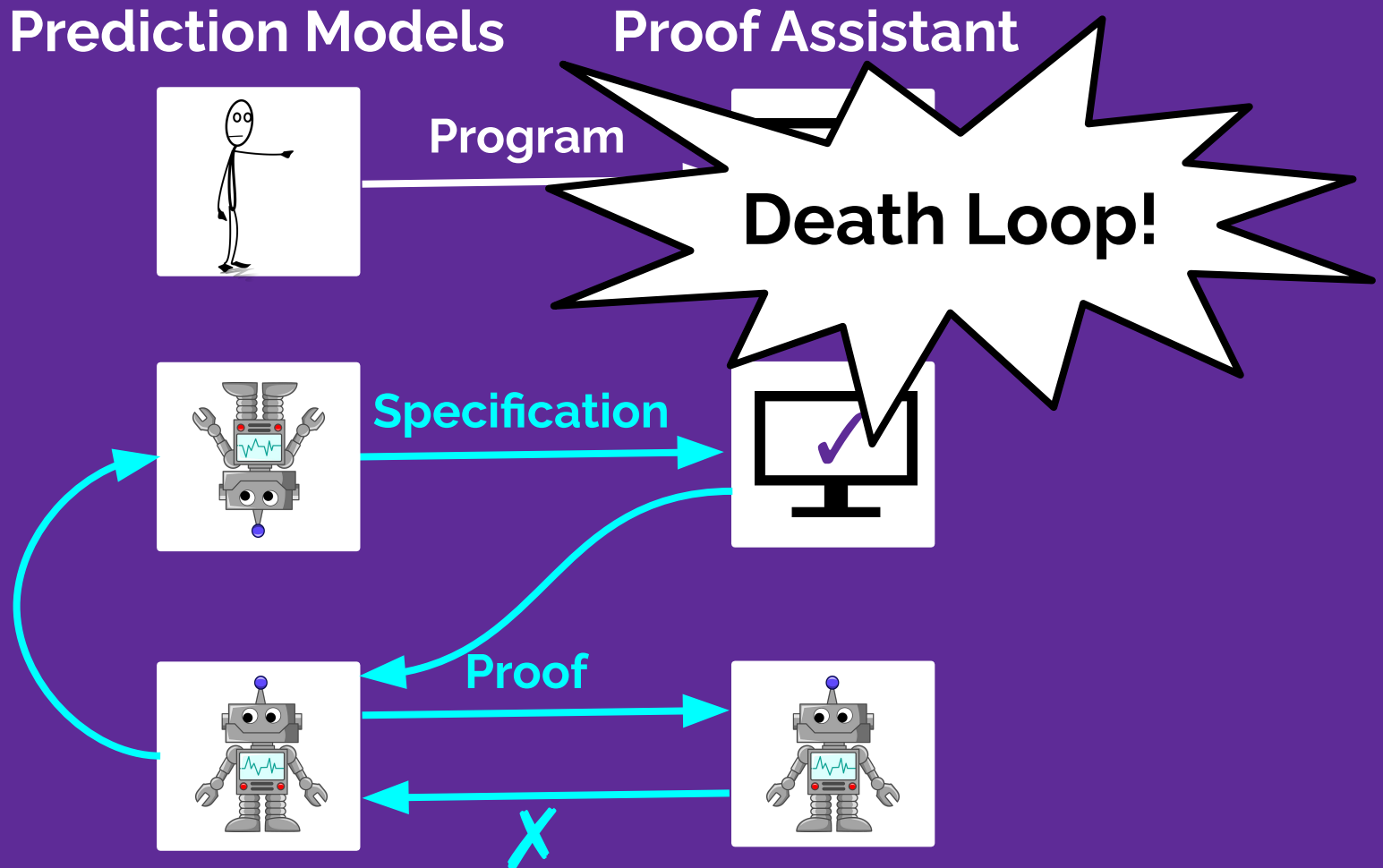
Conjecture Testing (Task 1 of 5)

It's Hard



Conjecture Testing (Task 1 of 5)

It's Hard

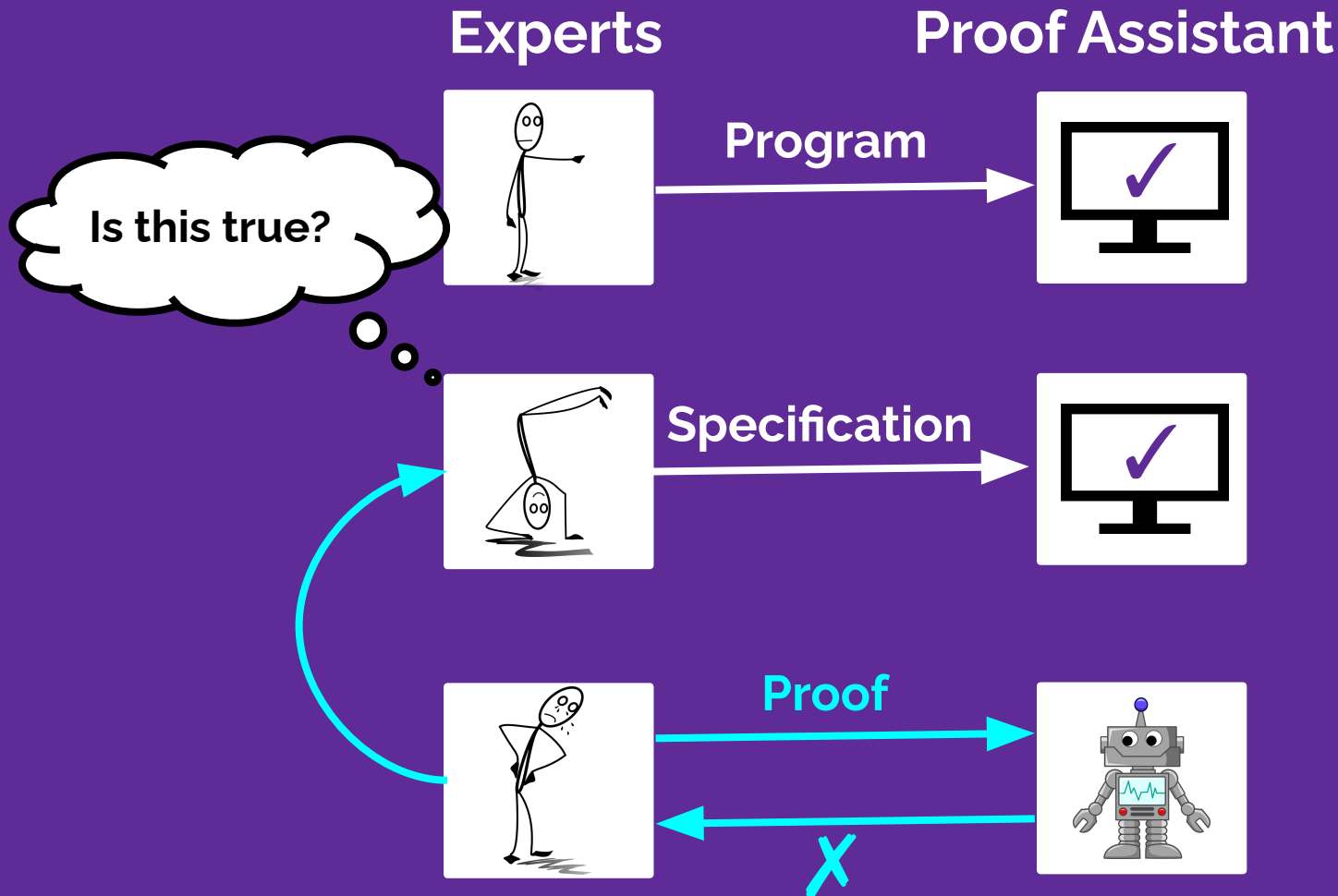


Conjecture Testing (Task 1 of 5)

The **death loop**
is a symptom of
gaming proofs.

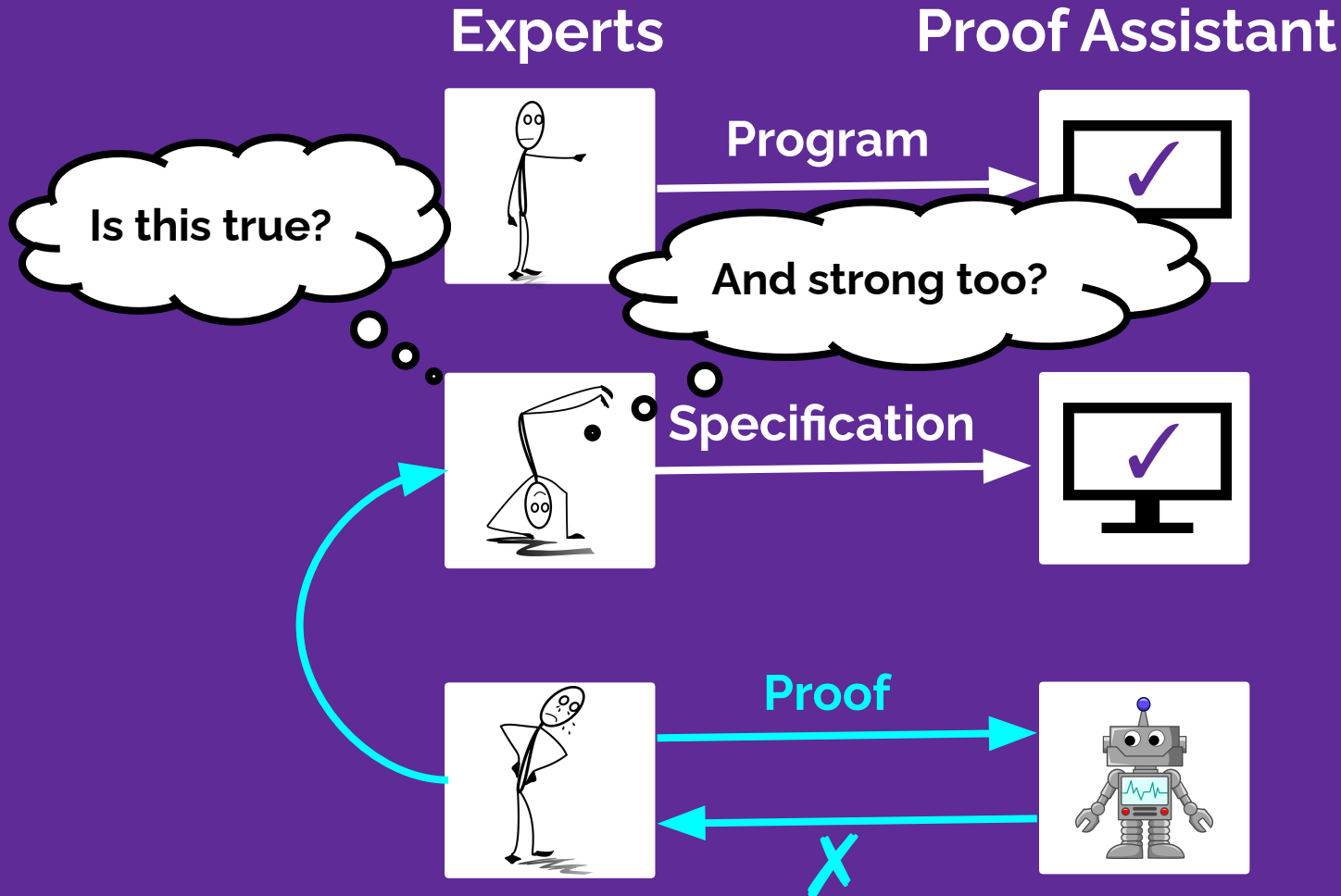
Conjecture Testing (Task 1 of 5)

What Experts Do



Conjecture Testing (Task 1 of 5)

What Experts Do

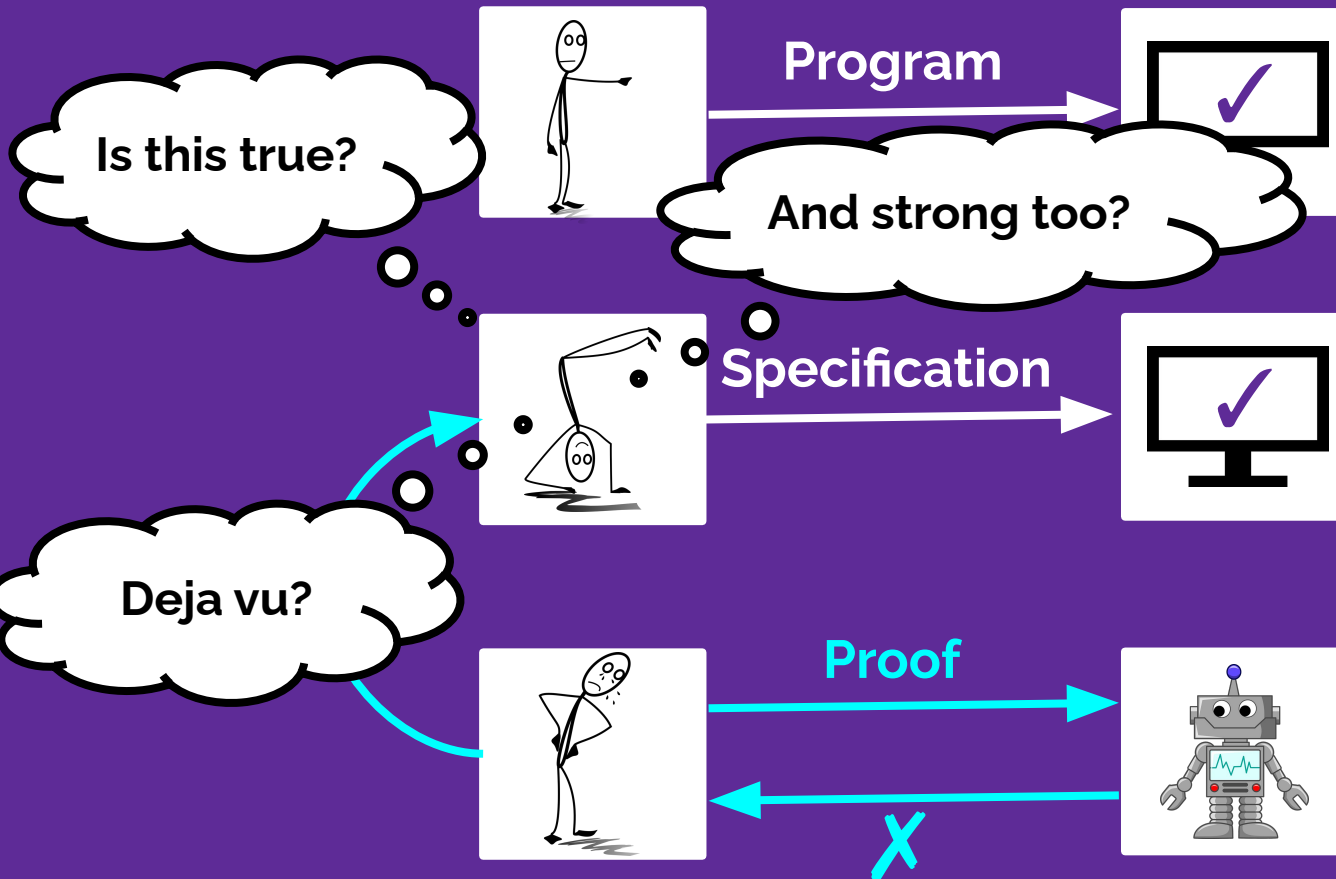


Conjecture Testing (Task 1 of 5)

What Experts Do

Experts

Proof Assistant



Conjecture Testing (Task 1 of 5)

What Experts Do

Experts

Proof Assistant

It's the spec.

Program

Specification

Proof

Conjecture Testing (Task 1 of 5)

What Experts Do

Experts

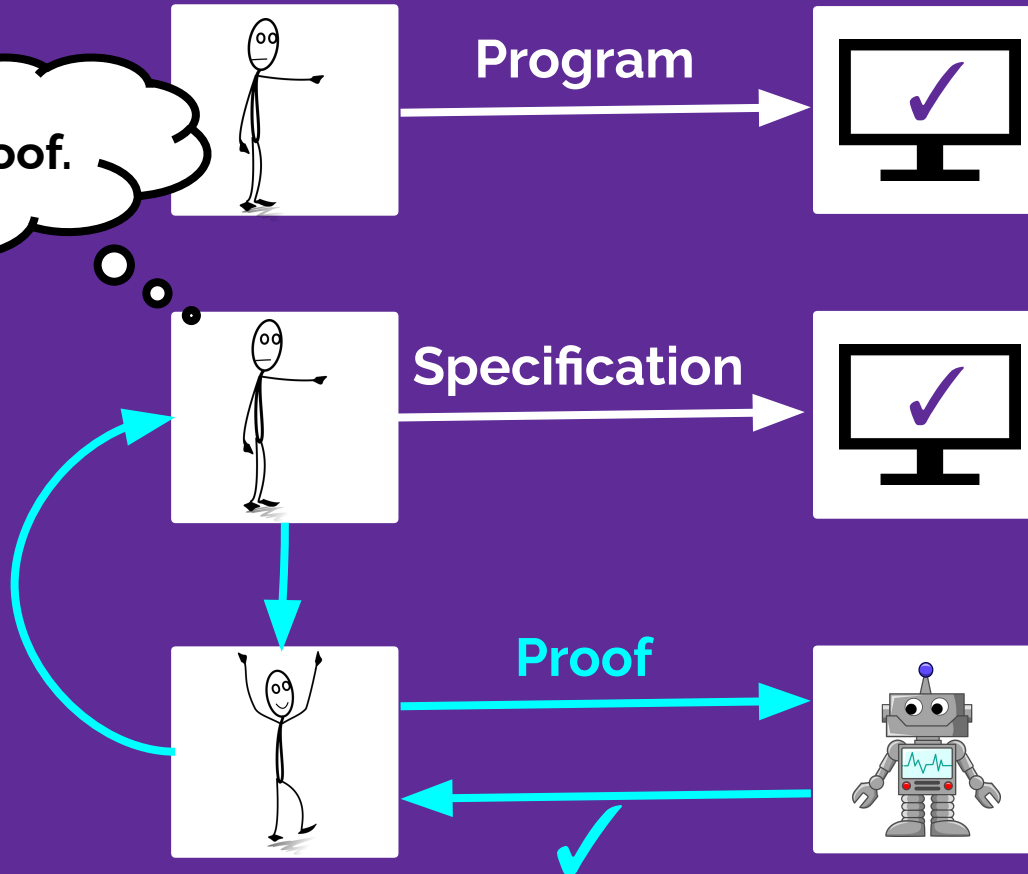
Proof Assistant

It's the proof.

Program

Specification

Proof



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 

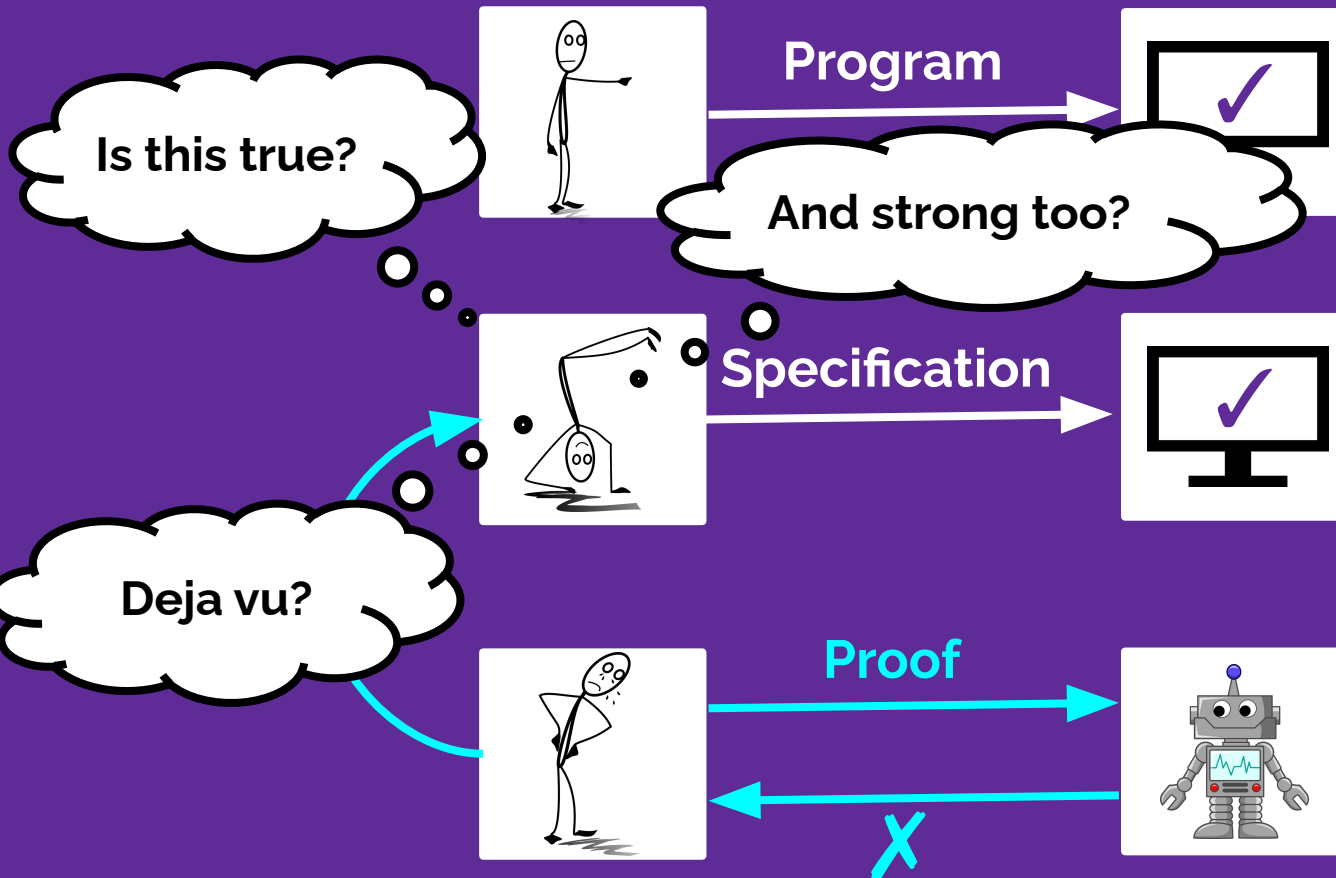
Your ideas here!

Conjecture Testing (Task 1 of 5)

What Experts Do

Experts

Proof Assistant

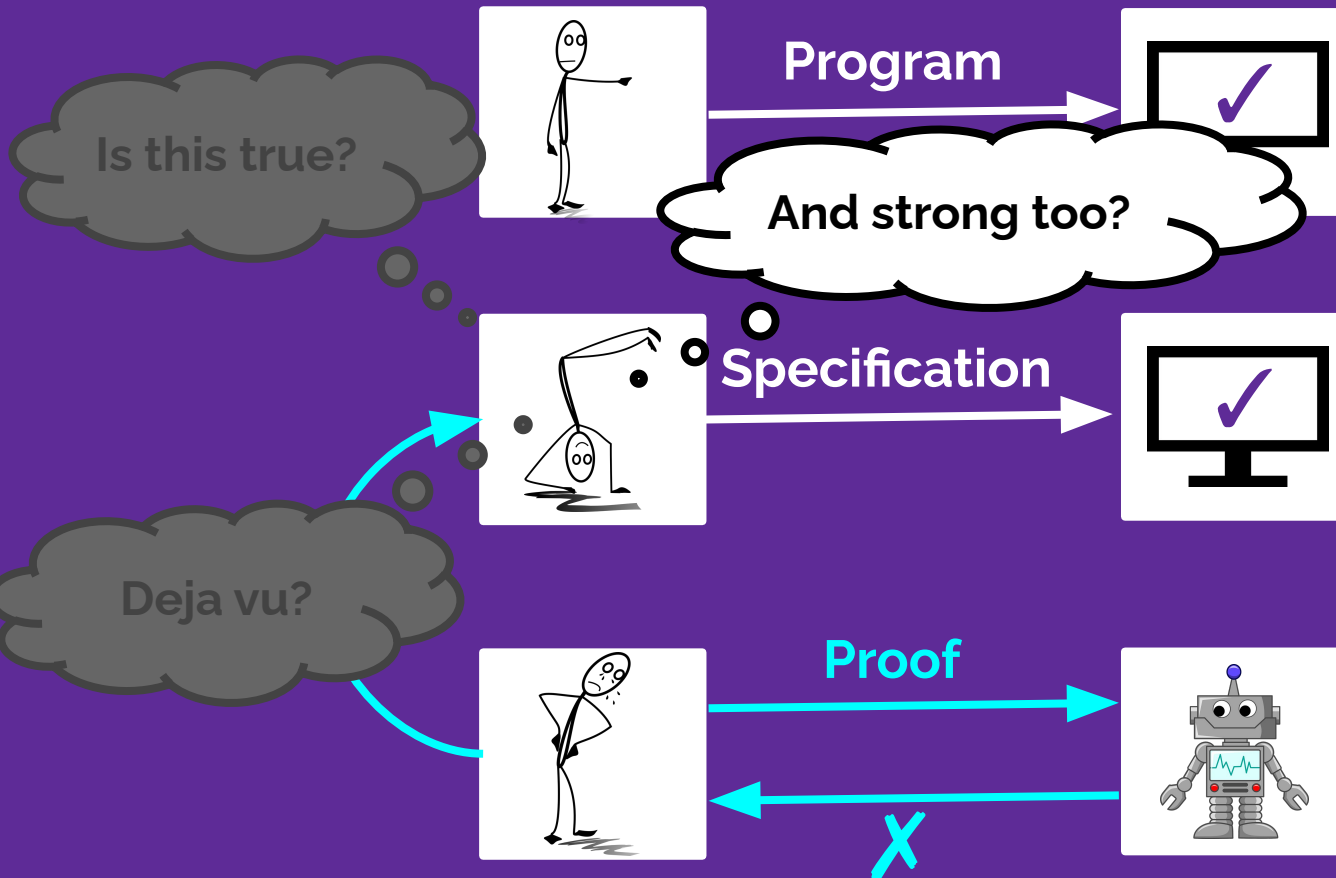


Conjecture Testing (Task 1 of 5)

What Experts Do

Experts

Proof Assistant



Conjecture Testing (Task 1 of 5)

Do lots more:

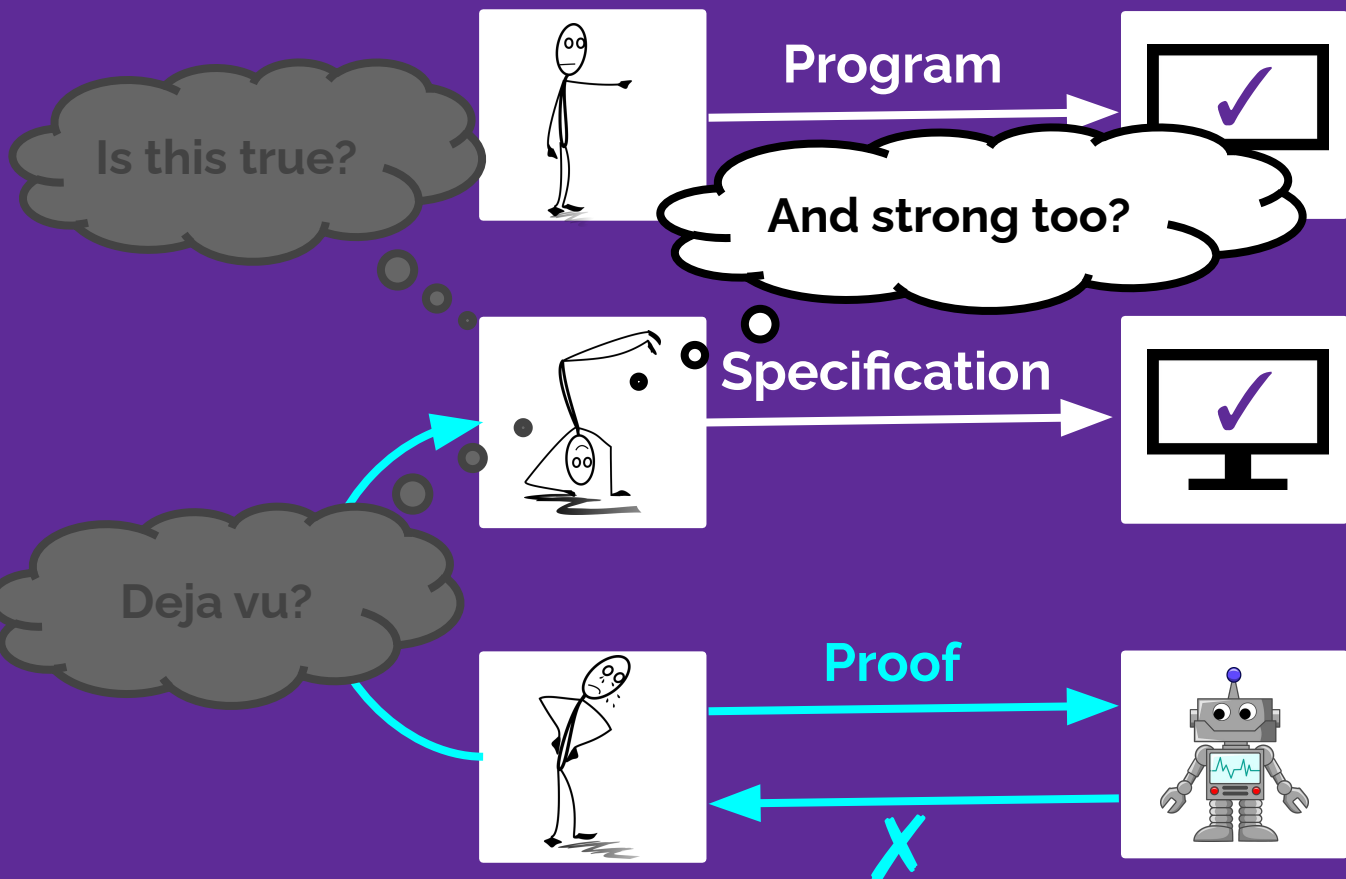
1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

You won't regret it!

Not Just Useful—Essential

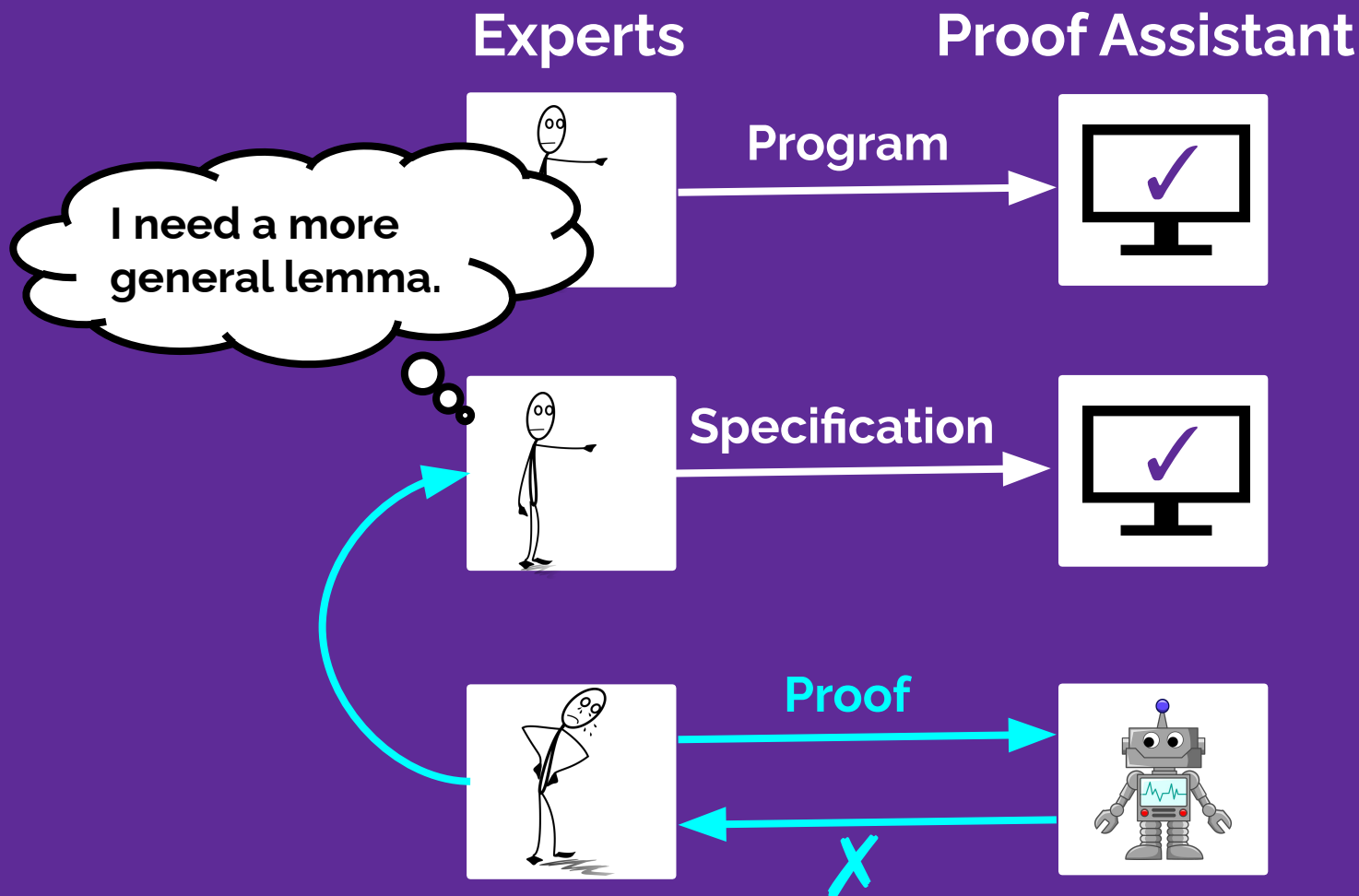
Experts

Proof Assistant



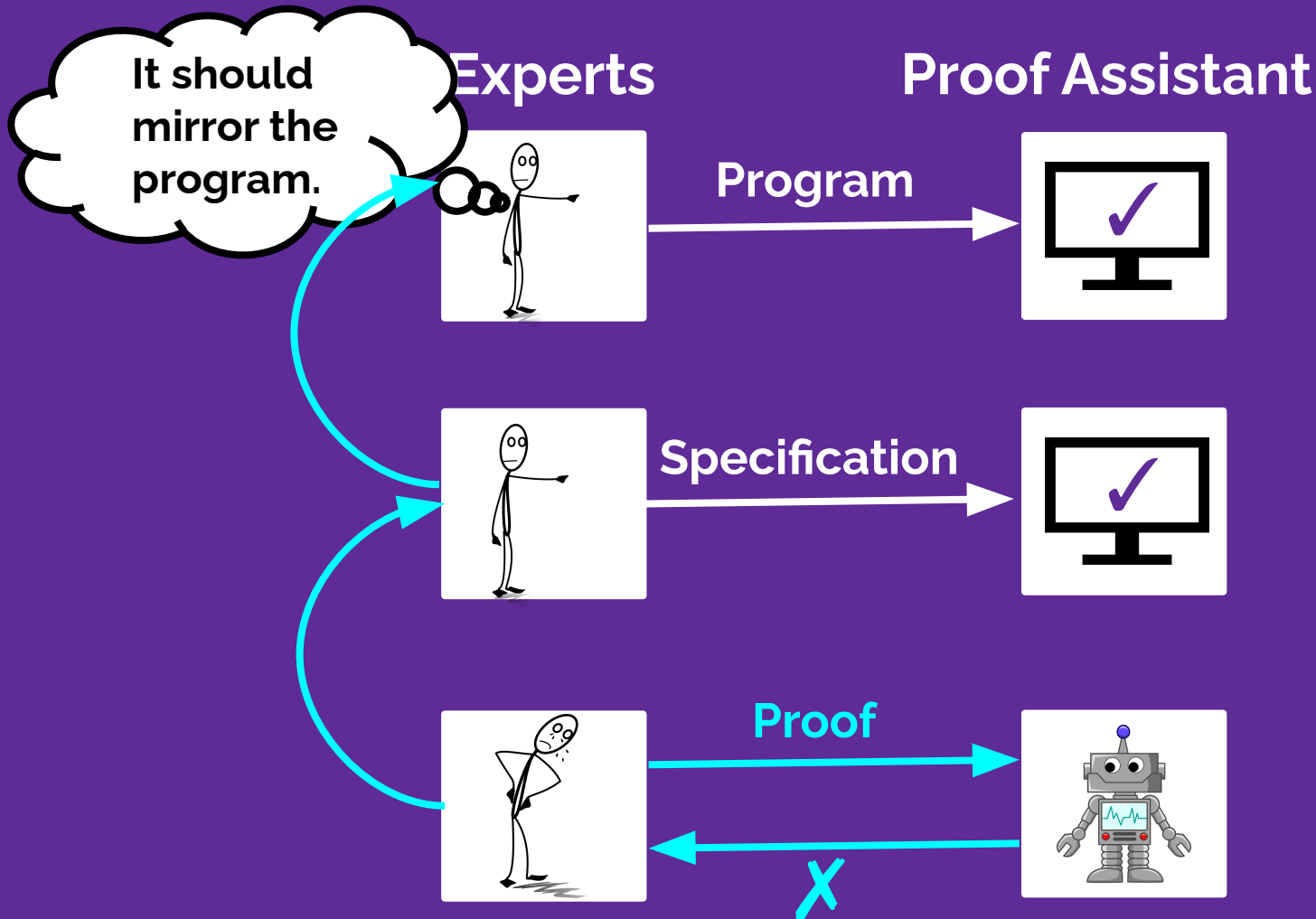
Lemma Discovery (Task 2 of 5)

Not Just Useful—Essential



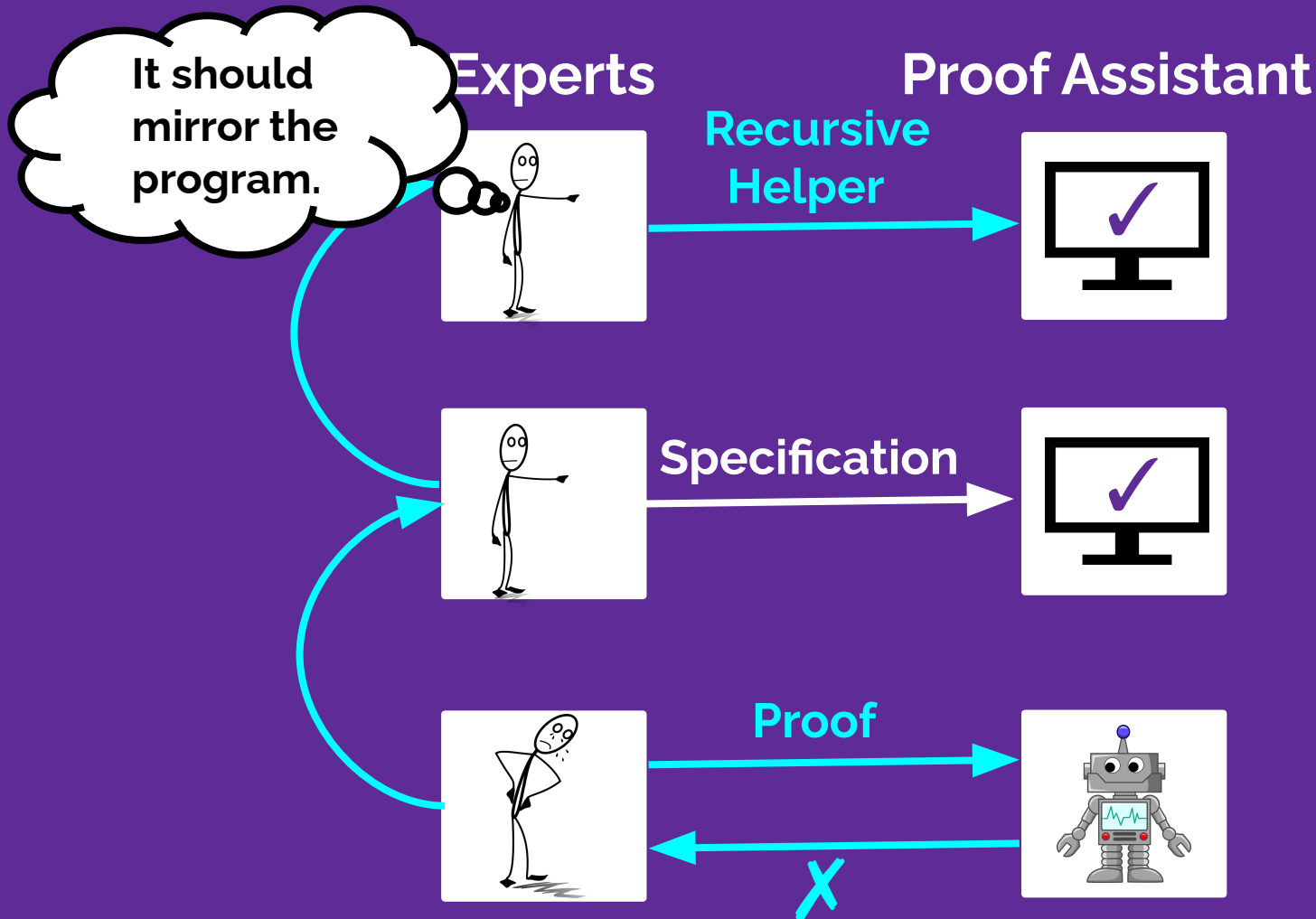
Lemma Discovery (Task 2 of 5)

Not Just Useful—Essential



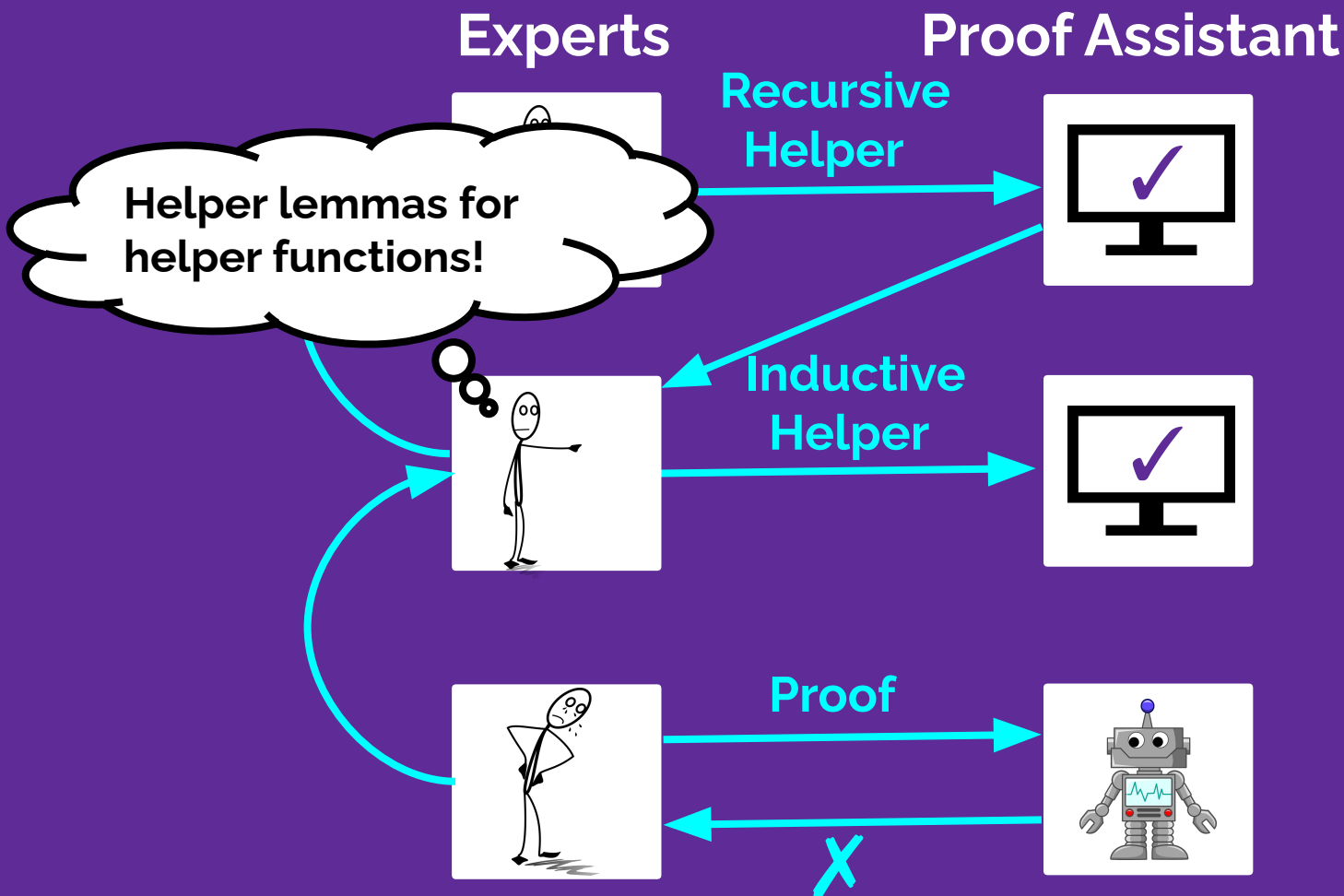
Lemma Discovery (Task 2 of 5)

Not Just Useful—Essential



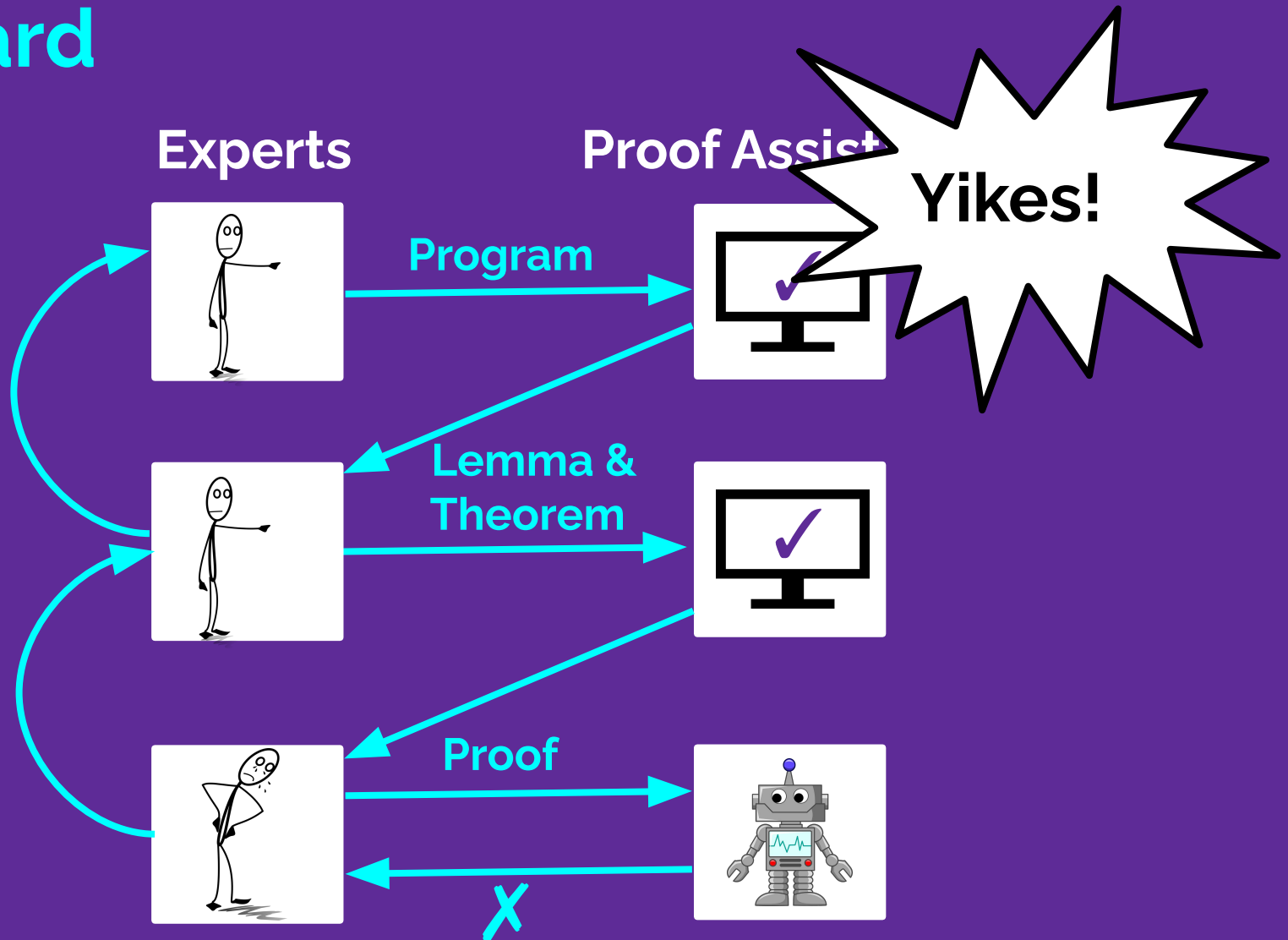
Lemma Discovery (Task 2 of 5)

Not Just Useful—Essential



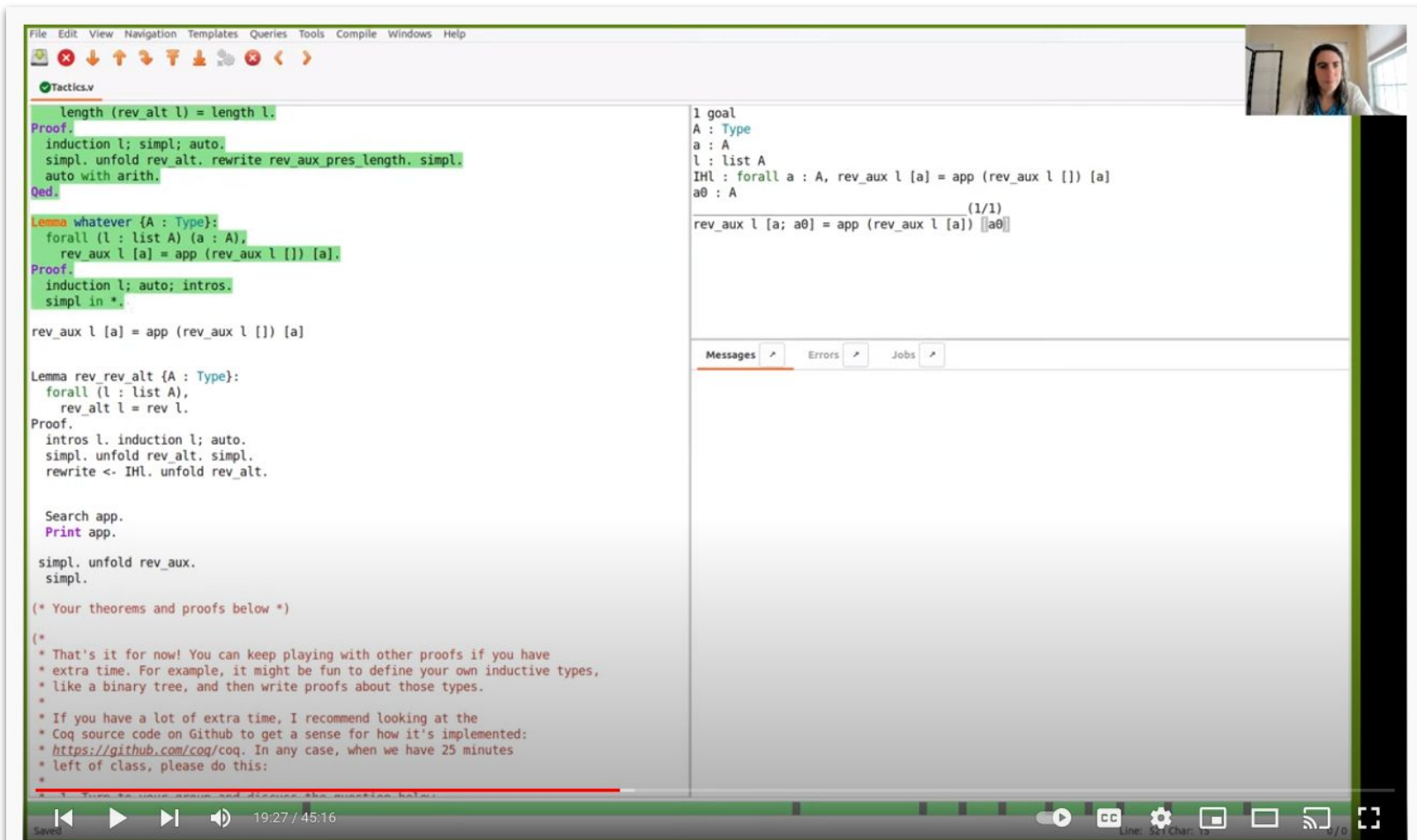
Lemma Discovery (Task 2 of 5)

It's Hard



Lemma Discovery (Task 2 of 5)

It's Hard



```
File Edit View Navigation Templates Queries Tools Compile Windows Help
Tactics.v
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.

Lemma whatever (A : Type):
forall (l : list A) (a : A),
  rev_aux l [a] = app (rev_aux l []) [a].
Proof.
induction l; auto; intros.
simpl in *.
rev_aux l [a] = app (rev_aux l []) [a]

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

Search app.
Print app.

simpl. unfold rev_aux.
simpl.

(* Your theorems and proofs below *)

(*
* 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 goal
A : Type
a : A
l : list A
IHL : forall a : A, rev_aux l [a] = app (rev_aux l []) [a]
a0 : A

(1/1)
rev_aux l [a; a0] = app (rev_aux l [a]) [a0]

Messages Errors Jobs

19:27 / 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

The screenshot shows a video player interface. The main content is a Lean 4 proof session. The code editor on the left contains the following code:

```
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.  
  
Lemma app_whatever {A : Type}:  
  forall (l : list A) (a : A),  
    l ++ [a] = rev_aux (rev_aux l []) [a].  
Proof.  
  induction l; intros; auto.  
  simpl in *. rewrite IHL.  
  
Eval compute in (rev_aux (rev_aux [1; 2; 3; 4] []) [5]).  
.....  
  
Lemma whatever {A : Type}:  
  forall (l : list A) (a : A),  
    rev_aux l [a] = app (rev_aux l []) [a].  
Proof.  
  intros l a. rewrite app_OK.  
  Search List.app.  
  simpl.  
  induction l; auto; intros.  
  simpl. rewrite app_rewrite IHL. simpl.  
  
rev_aux l [a] = app (rev_aux l []) [a]  
  
Lemma rev_rev_alt {A : Type}:  
  forall (l : list A),  
    rev_alt l = rev l.  
Proof.  
  intros l. induction l; auto.  
  simpl. unfold rev_alt. simpl.  
  rewr
```

The goal window on the right shows the current goal:

```
1 goal  
A : Type  
a : A  
l : list A  
IHL : forall a : A, l ++ [a] = rev_aux (rev_aux l []) [a]  
a0 : A  
-----  
a :: rev_aux (rev_aux l []) [a0] = rev_aux (rev_aux l [a]) [a0]
```

A text overlay in the center of the video reads: "just one that is hard to prove still".

The video player interface includes a progress bar at 27:38 / 45:16 and social media icons for likes, dislikes, shares, and saves.

Lemma Discovery (Task 2 of 5)

It's Hard



Yikes!



File Edit View Navigation Templates Queries Tools Compile Windows Help

Tactics.v

```
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.
```

```
Lemma rev_aux_cons (A : Type):  
  forall (l1 l2 : list A) (a : A),  
    rev_aux l1 l2 = rev_aux (l1 ++ [a]) l2.  
Proof.  
  induction l1; auto.  
  intros. simpl in *. rewrite IHl1. auto.  
Qed.
```

```
Lemma rev_rev_alt (A : Type):  
  forall (l : list A),  
    rev_alt l = rev l.  
Proof.  
  intros l. unfold rev alt.  
  intros l. induction l; auto.  
  simpl. unfold rev alt. simpl.  
  rewrite <- IHL. unfold rev alt.  
  rewrite app_OK. unfold rev in IHL. simpl.
```

```
Search app.  
Print app.
```

```
simpl. unfold rev_aux.  
simpl.
```

```
(* Your theorems and proofs below *)
```

```
(*  
 * 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 them.  
 *)  
 * If you have a lot of extra time, I recommend looking at the  
 * Con source code on Github to get a sense for how it's implemented.
```

not what i want

Messages Errors Jobs

37:02 / 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

Yikes!

Death Loop!

The screenshot shows a video player interface. The main content is a Lean 4 proof script in a text editor. The script includes several theorems and lemmas with their proofs. A dark grey box with white text is overlaid on the bottom of the script, containing the text "why am i repeating the same tactics". The video player controls at the bottom show a progress bar at 42:17 / 45:16. Below the video player, the video title "Proof Sesh 1: a taste of my own medicine" and view count "211 views • Feb 3, 2022" are visible. On the right side, there are icons for likes (16), dislikes, share, and save.

```
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_cons (A : Type):
  forall (l1 l2 : list A) (a : A),
    rev_aux l1 l2 = rev l1 ++ l2.
Proof.
  induction l1; intros; auto.
  simpl in *. rewrite (IHL1 a).
  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.
  intros l. induction l; auto.
  simpl. unfold rev_alt. simpl.
  rewrite <- IHL. unfold rev_alt.
  rewrite app_OK. unfold rev in IHL. simpl.

Search app.
Print app.

simpl. unfold rev_aux.
simpl.

(* Your theorems and proofs below *)
(*
* That's it for now! You can keep playing with other proofs if you have
```

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 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 Play (k) 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
$$\frac{}{\text{rev_aux l1 (a :: l2) = app (rev l1) [a] ++ l2}} \text{(1/1)}$$

Messages Errors Jobs

43:57 / 45:16

Proof Sesh 1: a taste of my own medicine

211 views • Feb 3, 2022

16 DISLIKE SHARE SAVE

no wonder i just made it extra hard for no reason

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. Pause (k) your group and discuss the question below.
 * 2. Answer—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

44:45 / 45:16

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?**

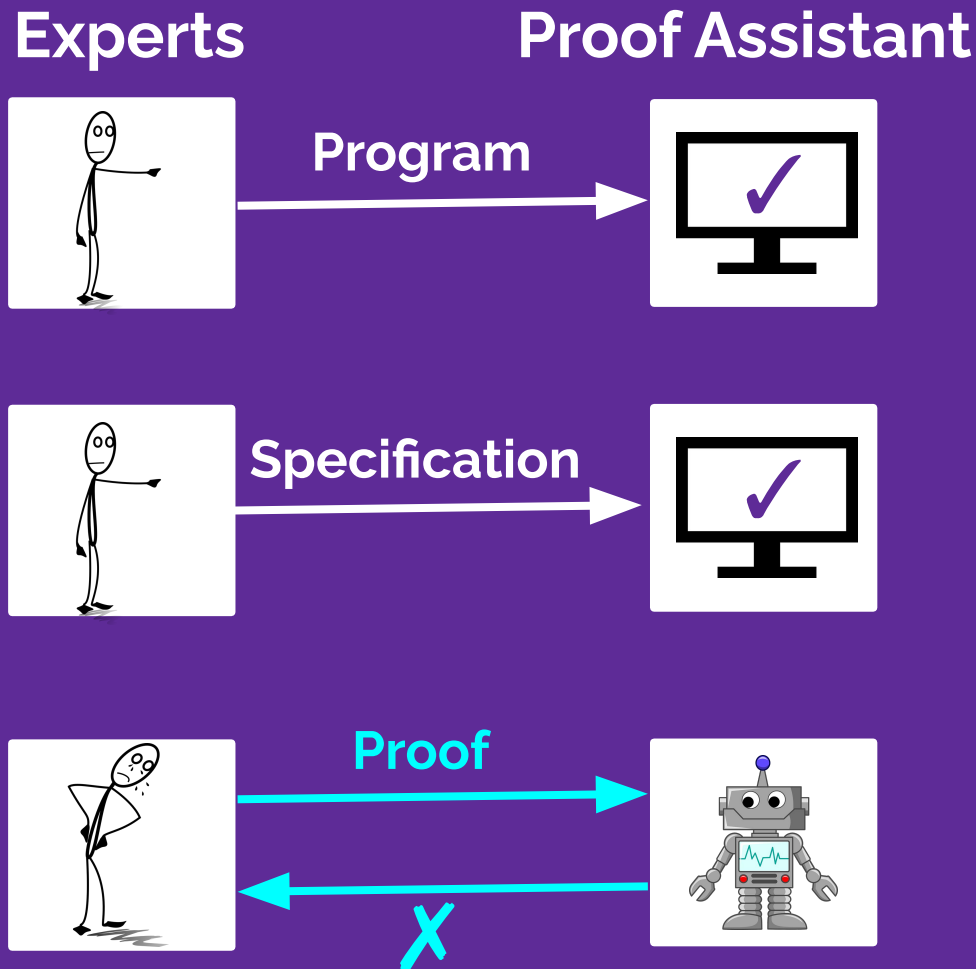
Lemma Discovery (Task 2 of 5)

Ideas 

Your ideas here!

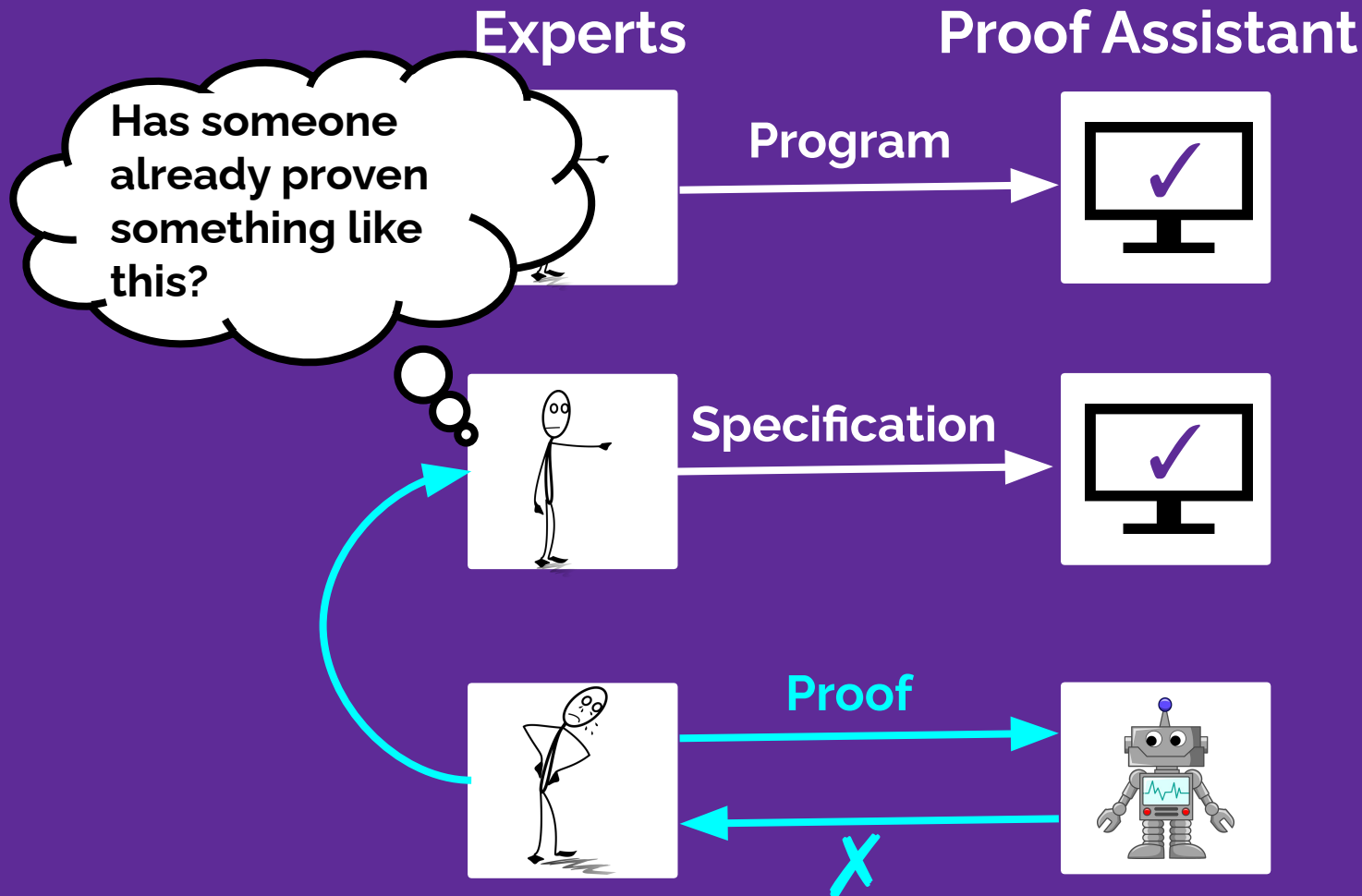
Lemma Discovery (Task 2 of 5)

Experts are Mere Mortals



Lemma Discovery (Task 2 of 5)

Experts are Mere Mortals



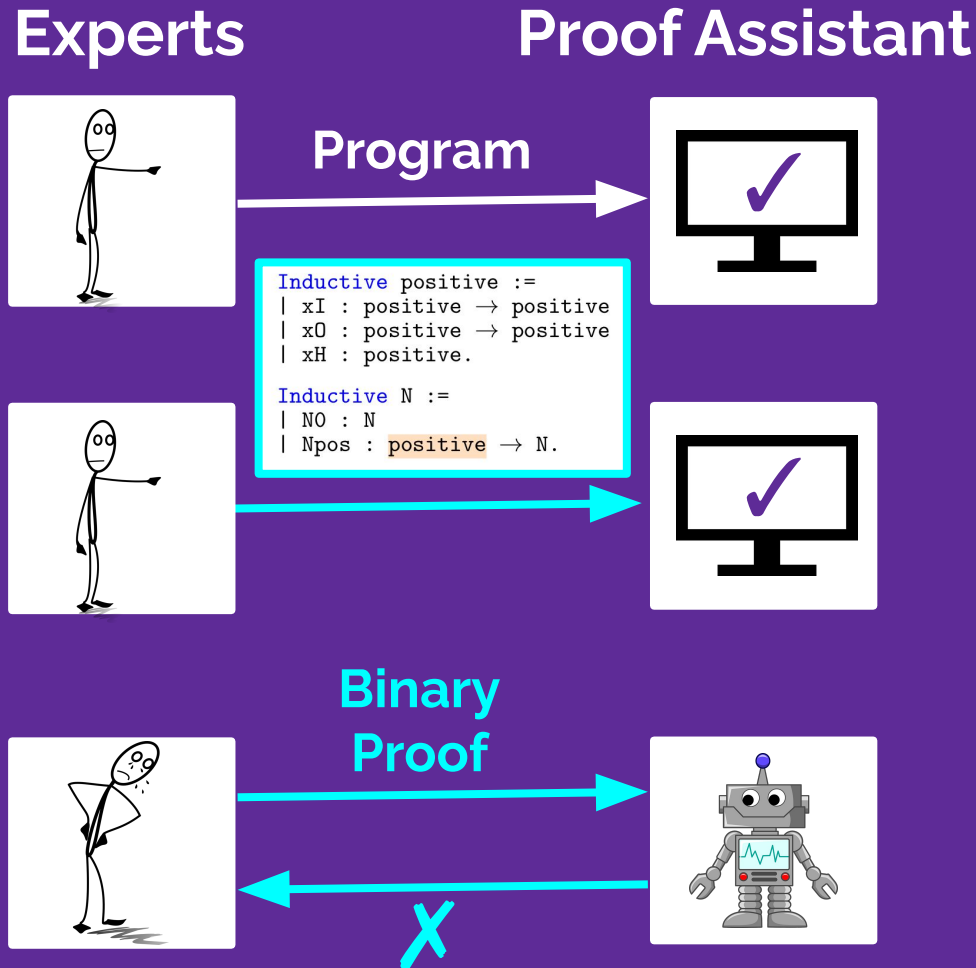
Lemma Discovery (Task 2 of 5)

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

Google search results for "coq binary number proofs".

Search bar: coq binary number proofs

Navigation: All, Images, Videos, News, Shopping, More, Tools

About 179,000 results (0.69 seconds)

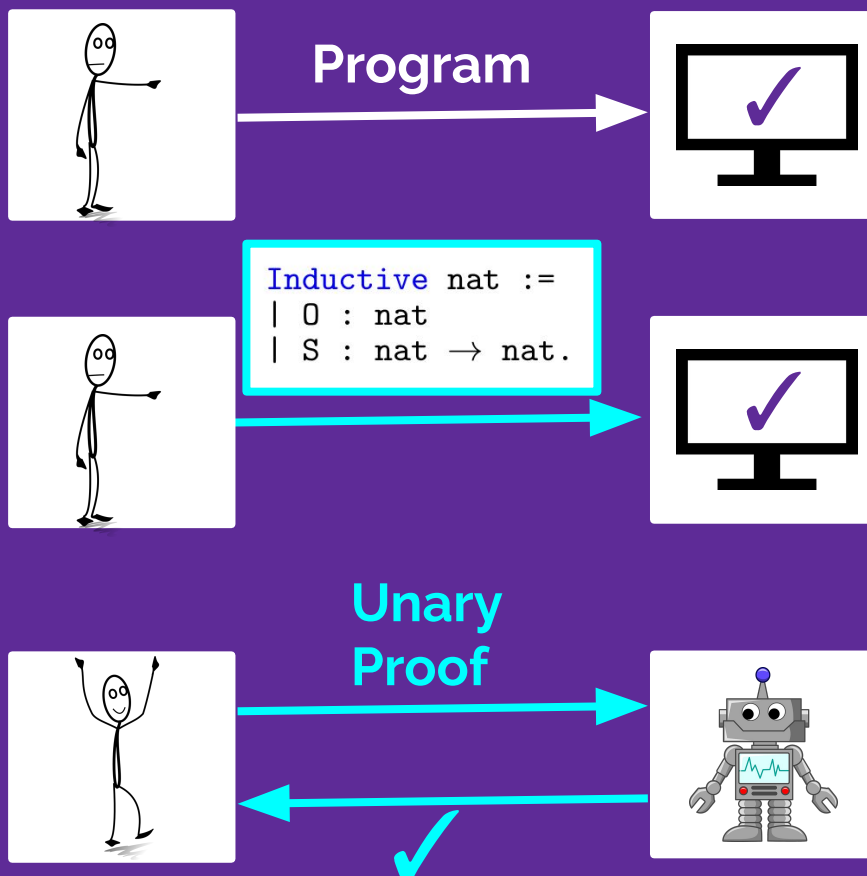
- <https://coq.inria.fr/library/Coq.NArith.BinNat.html>
Binary natural numbers, operations and properties - Standard ...
Every definitions and properties about **binary** natural **numbers** are placed in a module N for ...
Proofs of morphisms, obvious since eq is Leibniz.
- <https://coq.inria.fr/library/Coq.Numbers.BinNums.html>
Library Coq.Numbers.BinNums
Binary Numerical Datatypes. Set Implicit Arguments. positive is a datatype representing the strictly positive integers in a **binary** ...
- <https://coq.inria.fr/library/Coq.PArith.BinPosDef.html>
Library Coq.PArith.BinPosDef
Binary positive **numbers**, operations. Initial development by Pierre Crégut, CNET, Lannion, France. The type positive and its constructors x1 and x0 and xH ...
- <https://coq.inria.fr/library/Coq.PArith.BinPos.html>
Binary positive numbers, operations and properties - Standard ...
Properties of successor on **binary** positive **numbers** ... Correctness **proofs** for the square root function. Inductive SqrtSpec : positive*mask -> positive ...
- <http://staff.ustc.edu.cn/courses/theory/slides>
Some notes for lecture 1
Require Export Coq.omega. ... Then **prove** that starting with any natural **number**, converting to **binary**, then converting back yields the same natural **number** ...

Relation Discovery (Task 3 of 5)

Close Enough?

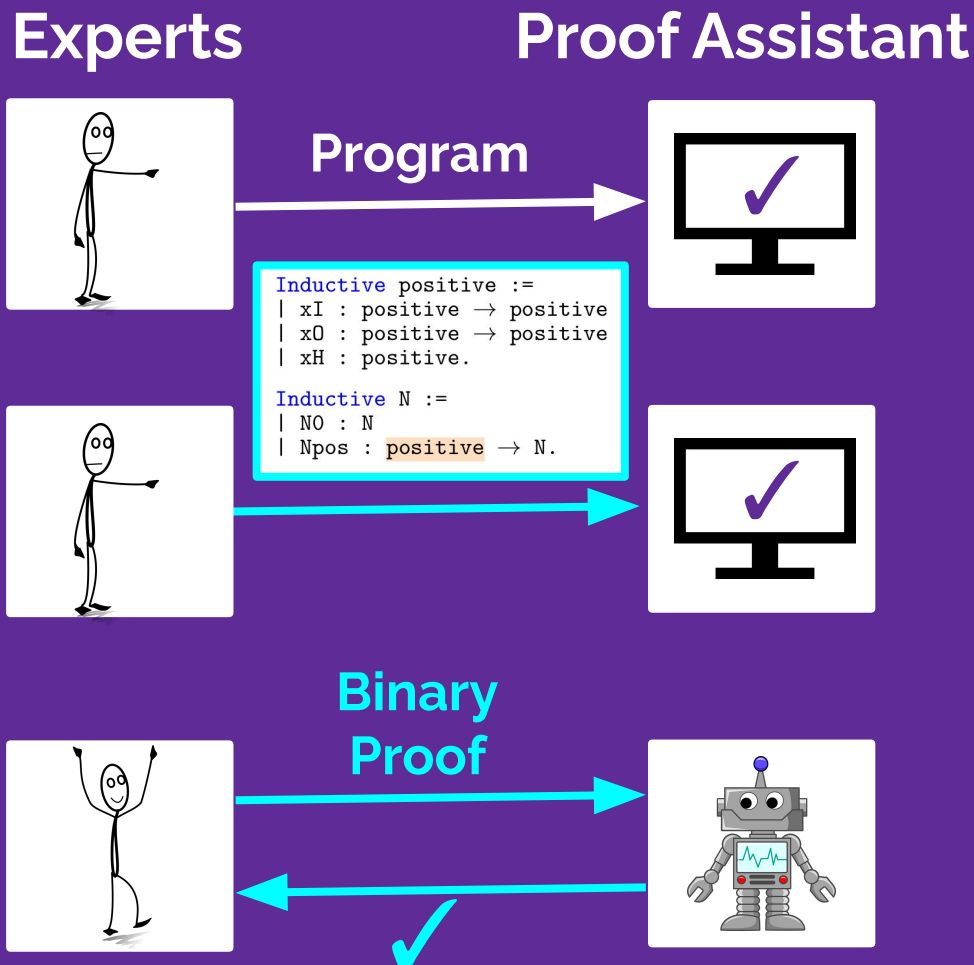
Internet Dudes

Proof Assistant



Relation Discovery (Task 3 of 5)

Close Enough?



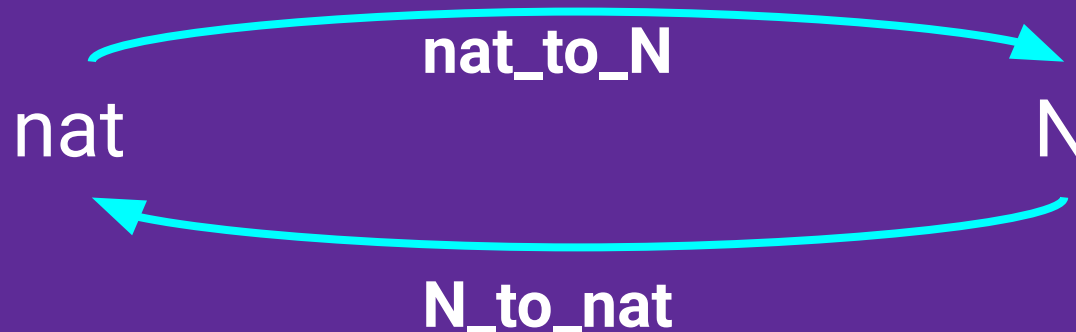
Relation Discovery (Task 3 of 5)

Close Enough!

```
Inductive nat :=  
| 0 : nat  
| S : nat → nat.
```

```
Inductive positive :=  
| xI : positive → positive  
| x0 : positive → positive  
| xH : positive.
```

```
Inductive N :=  
| NO : N  
| Npos : positive → N.
```



Relation Discovery (Task 3 of 5)

It's Hard



```
Inductive nat :=  
| 0 : nat  
| S : nat → nat.
```

```
Inductive positive :=  
| xI : positive → positive  
| x0 : positive → positive  
| xH : positive.
```

```
Inductive N :=  
| NO : N  
| Npos : positive → N.
```

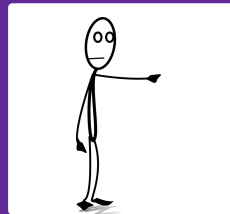


Relation Discovery (Task 3 of 5)

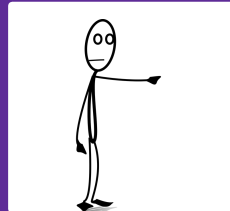
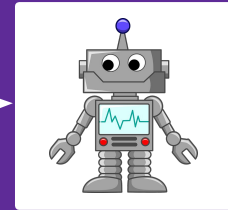
It's Hard

Internet Dude

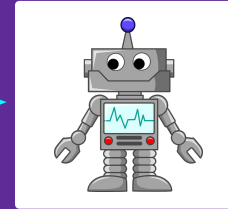
Training



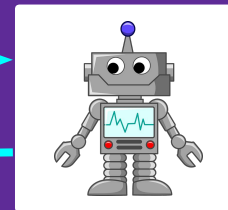
Program



```
Inductive nat :=  
| 0 : nat  
| S : nat → nat.
```

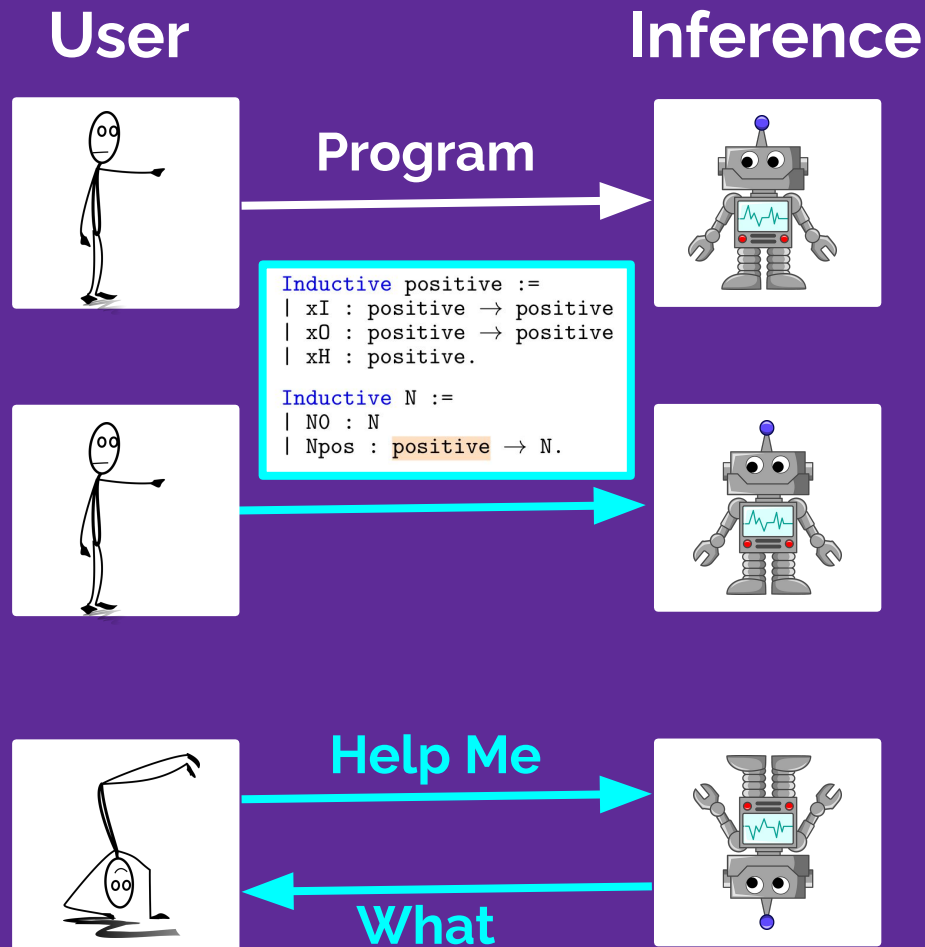


Unary
Proof



Relation Discovery (Task 3 of 5)

It's Hard

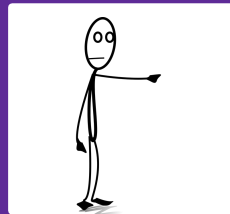


Relation Discovery (Task 3 of 5)

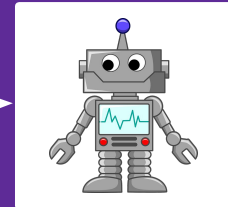
It's Hard

User

Inference

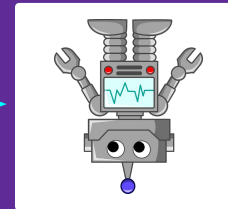
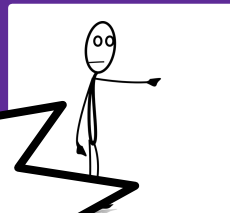


Program



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

```
Inductive N :=  
| NO : N  
| Npos : positive → N.
```

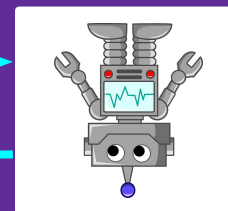


Wow I've never seen anything like this before!

Argh!



Help Me



What

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)

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

Ideas 

**Semantically embed
types and their relations?**

Relation Discovery (Task 3 of 5)

Allow the model to **play**
with the datatypes?

Relation Discovery (Task 3 of 5)

Ideas  

Combine with
property-based testing?

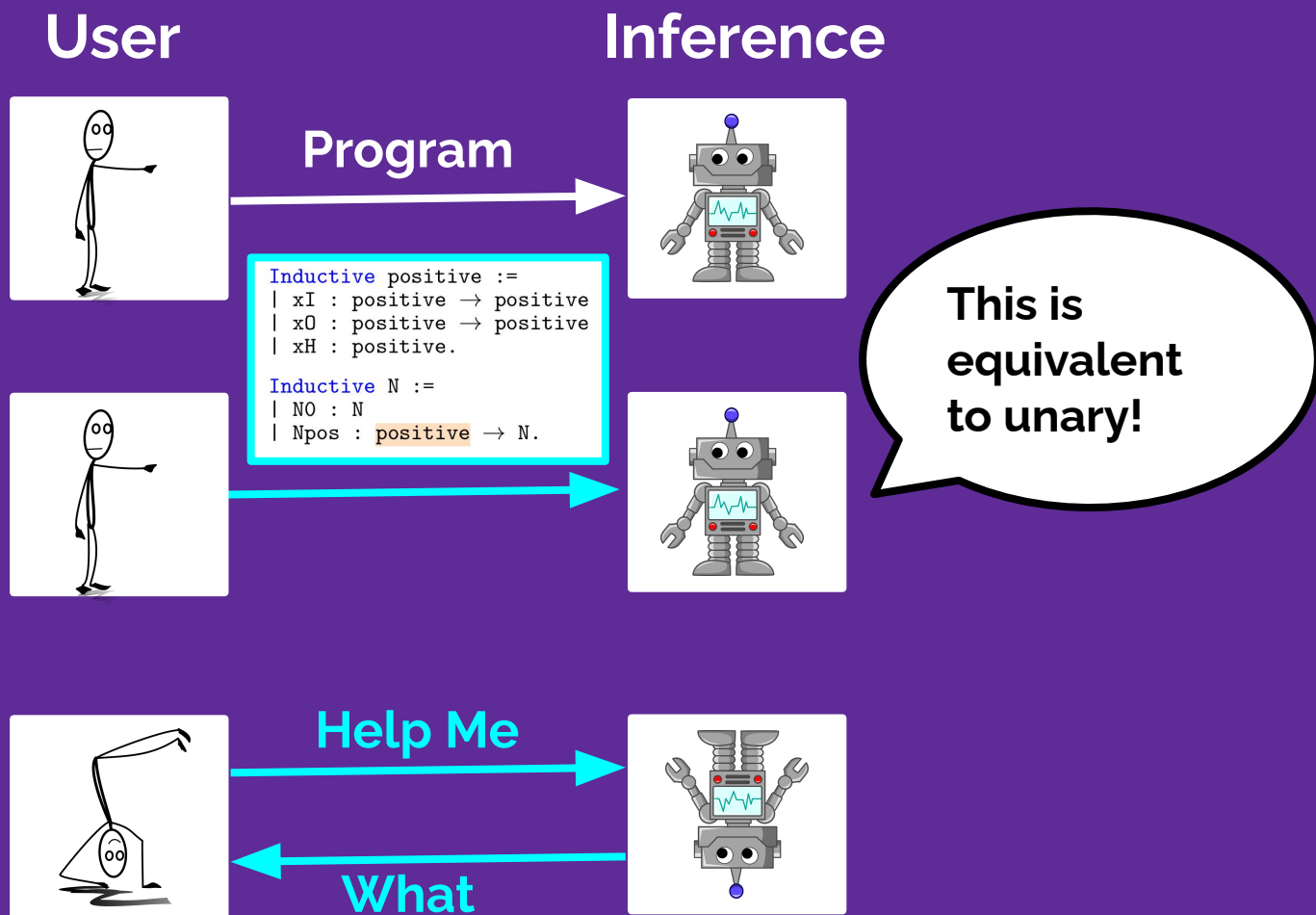
Relation Discovery (Task 3 of 5)

Ideas 

Your ideas here!

Relation Discovery (Task 3 of 5)

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.

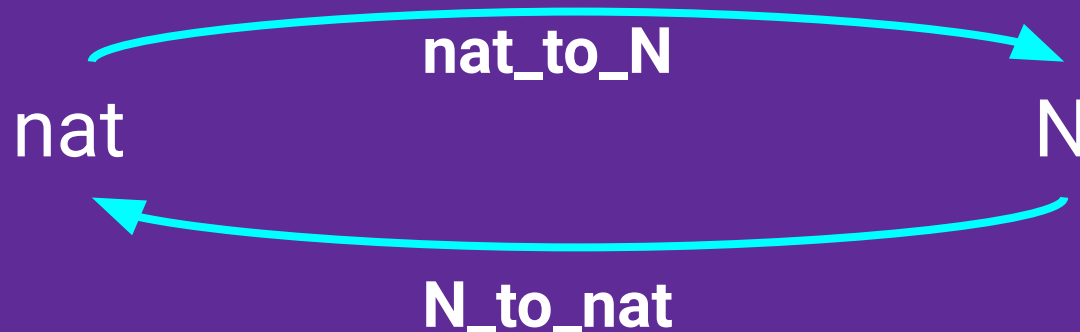
You won't regret it!

So Far

```
Inductive nat :=  
| 0 : nat  
| S : nat → nat.
```

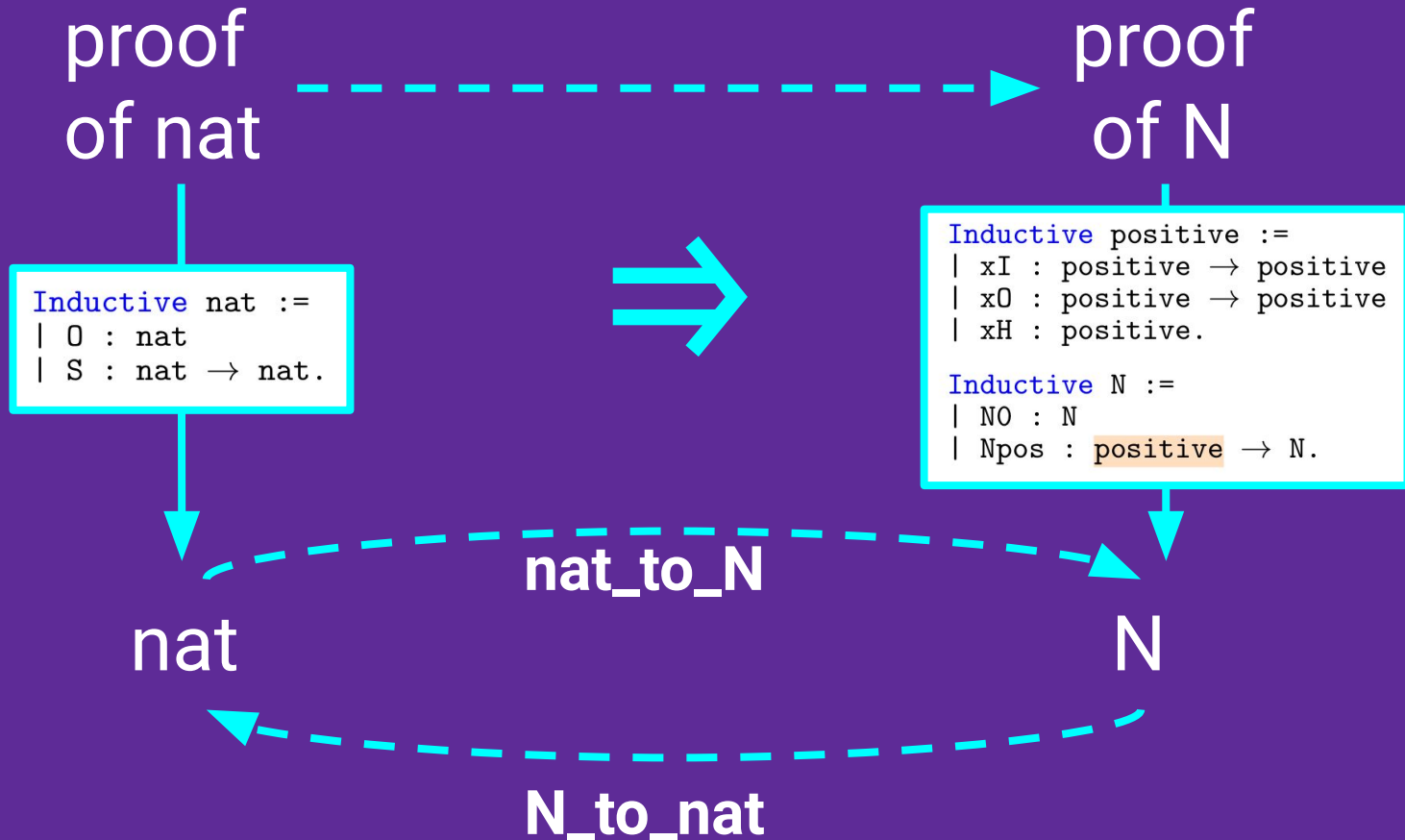
```
Inductive positive :=  
| xI : positive → positive  
| x0 : positive → positive  
| xH : positive.
```

```
Inductive N :=  
| NO : 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 :=  
| NO : 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)

New Frontiers

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.”

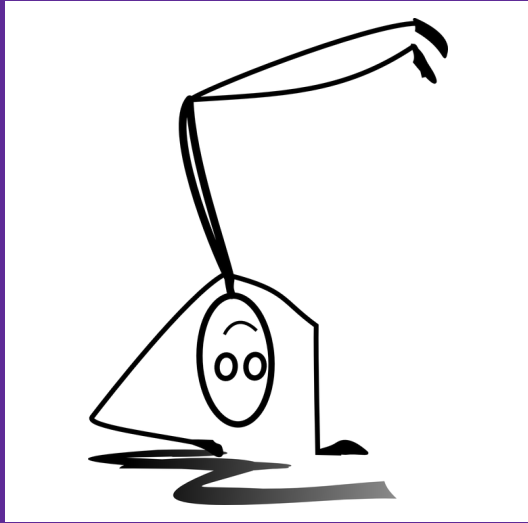
Proof Reuse & Repair (Task 4 of 5)

Do lots more:

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

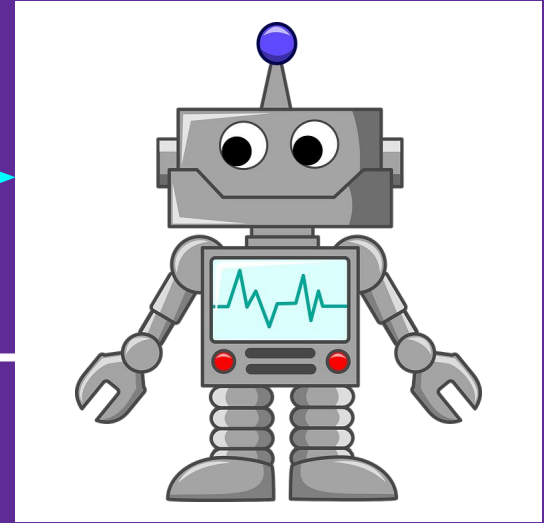
You won't regret it!

So Far



Proof Engineer

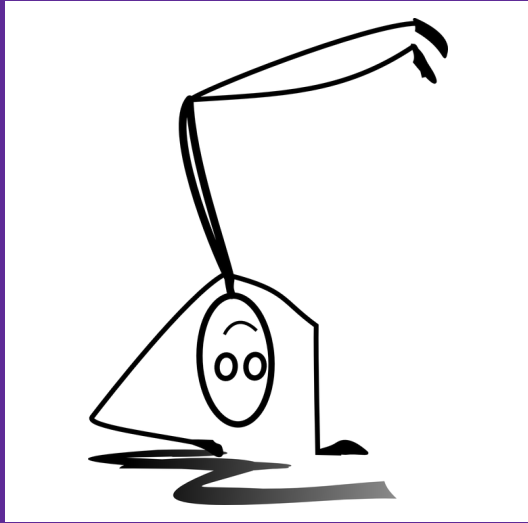
Search



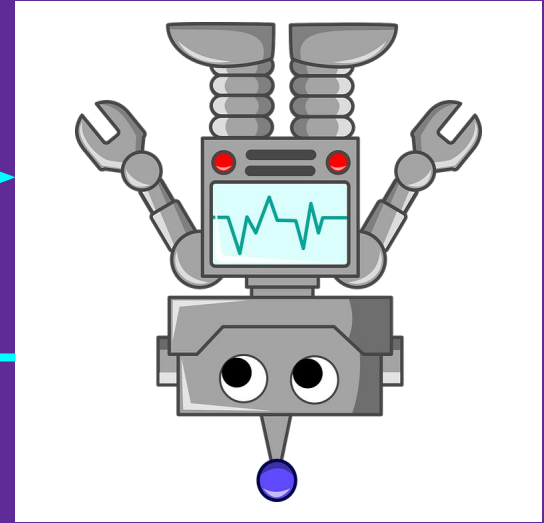
Proof Assistant

Semantic Search (Task 5 of 5)

So Far



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:

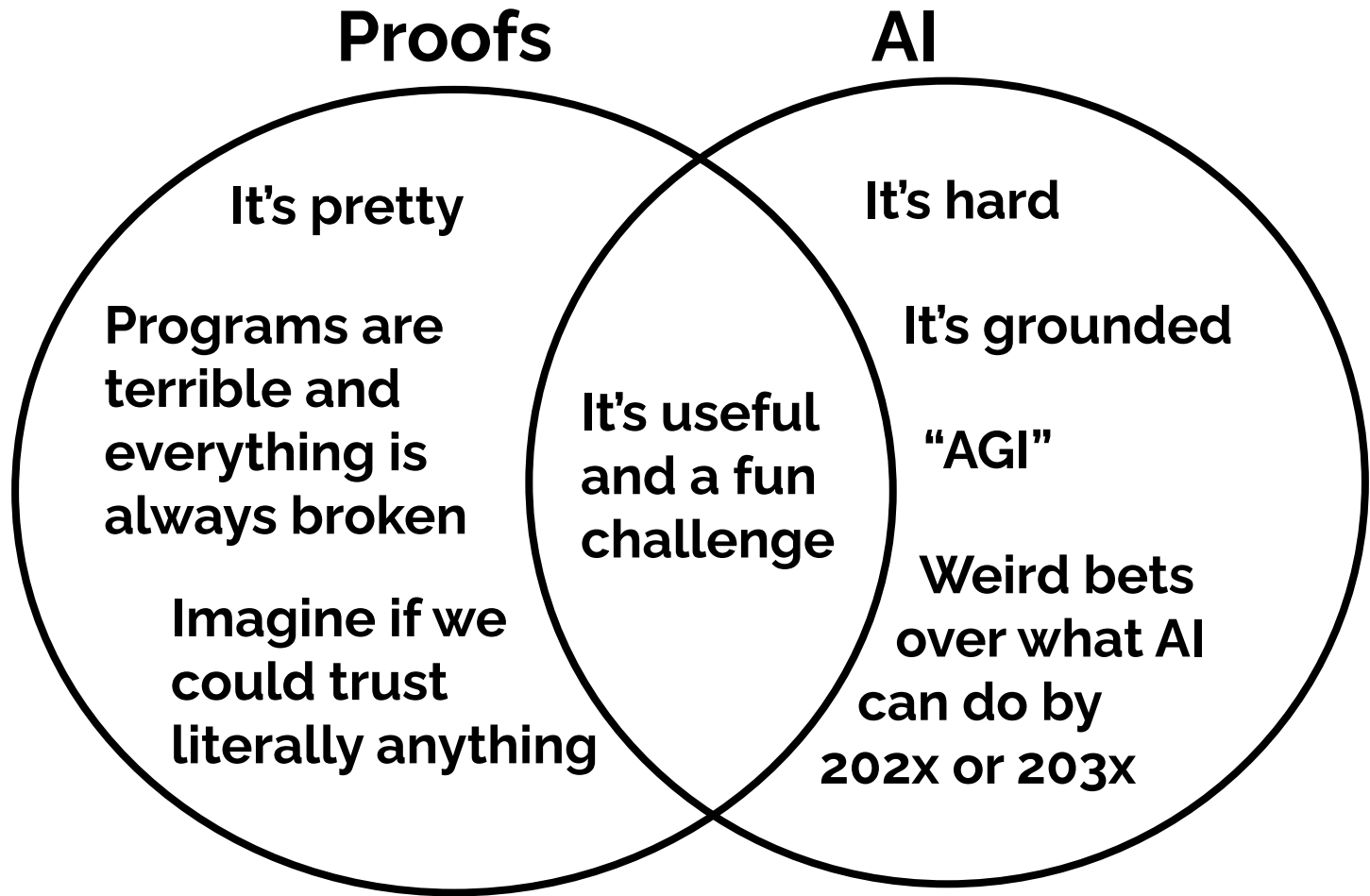
1. conjecture testing,
2. lemma discovery,
3. relation discovery,
4. proof reuse & repair, and
5. semantic search.

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!