

Language Models for Formal Proof

Talia Ringer

UIUC Computer Science

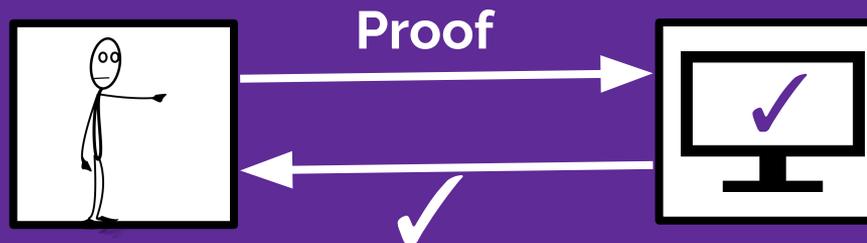


Proof Assistants

Proof Assistants

Proof Engineer

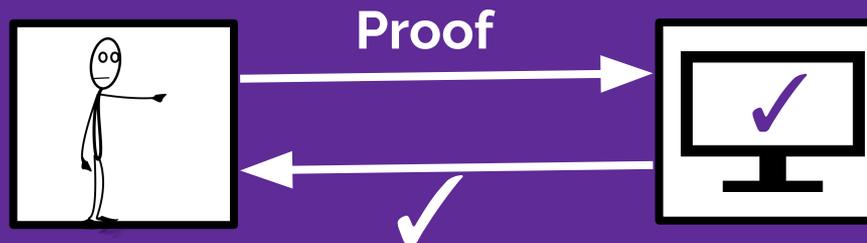
Proof Assistant



Proof Assistants

Proof Engineer

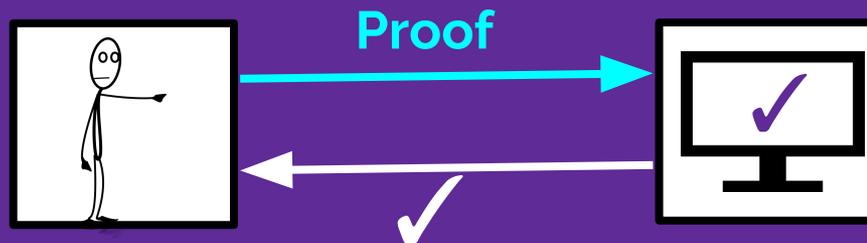
Proof Assistant



Proof Assistants

Proof Engineer

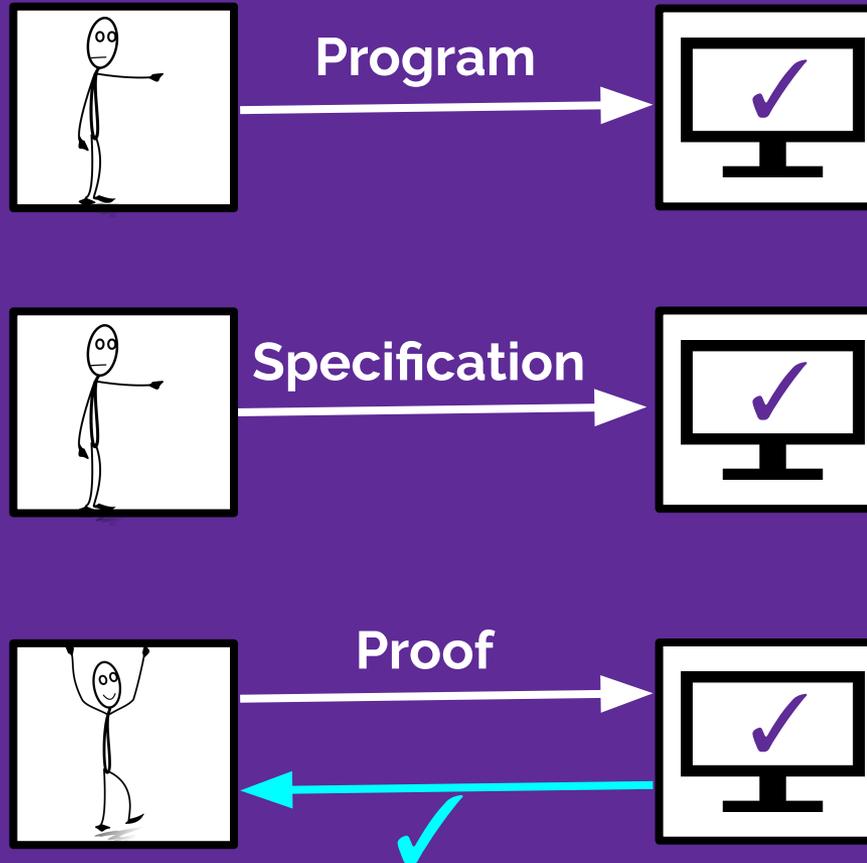
Proof Assistant



Proof Assistants

Proof Engineer

Proof Assistant

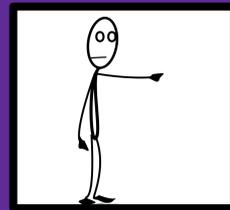


Then vs. Now

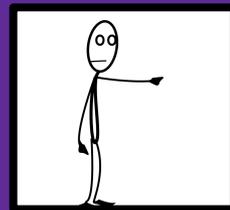
Then vs. Now

Proof Engineer

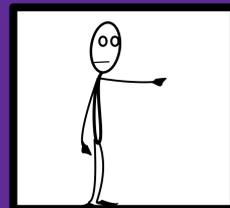
Proof Assistant



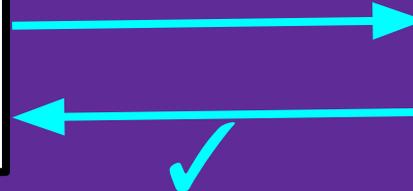
Program



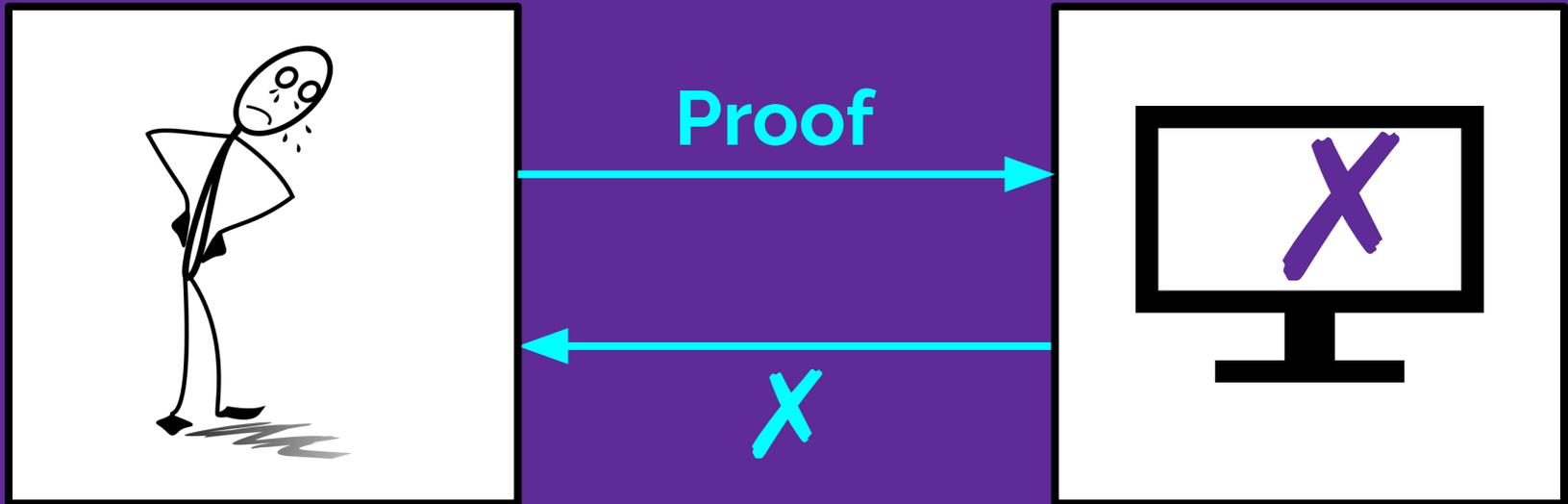
Specification



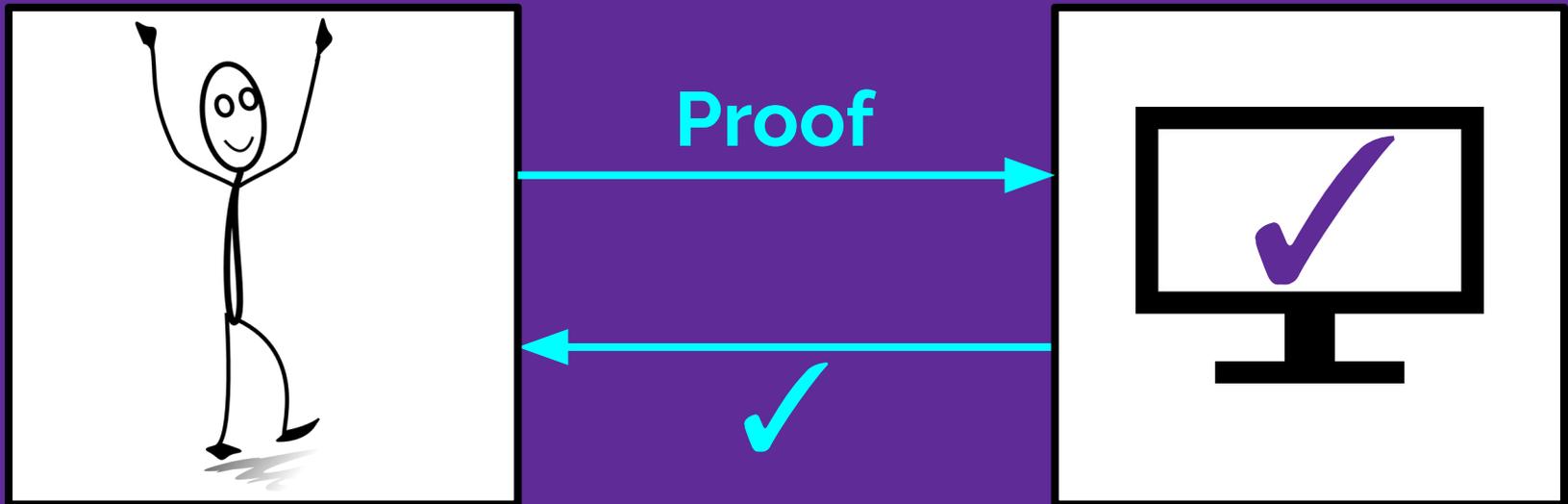
Proof



Then vs. Now



Then vs. Now



Then vs. Now

```
51 char *EXP_COM[] = {"...", "..."
52 char *RV_TIME[] = {"...", "..."
53
54
55 int summary(void *barg, void *arg)
56 {
57     char *str = (char *)arg;
58     st_board *board = (st_board *)barg;
59     int ret = 0;
60     char *ptr_shuttercounter = "...
```

Compilers

```
1. Sep 15:53
0. Sep 2015 bin -> usr/bin
19. Sep 09:31 boot
21. Sep 15:50 dev
19. Sep 09:32 etc
7 30. Sep 2015 home
34 30. Sep 2015 lib -> usr/lib
96 23. Jul 10:01 lib64 -> usr/lib
996 1. Aug 22:45 lost+found
16 21. Sep 2015 mat
4096 12. Aug 15:37 opt
560 21. Sep 08:15 private -> /home/encrypted
7 30. Sep 2015 proc
4096 21. Sep 15:50 root
4096 30. Sep 2015 run
1 300 21. Sep 15:51 sbin -> usr/bin
8 4096 12. Aug 15:45 srv
1 4096 23. Jul 10:25 usr
1 4096 21. Sep 15:54 var
```

File Systems



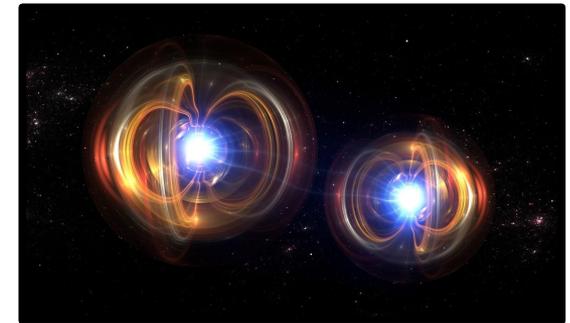
Web Browsers



Machine Learning Systems



Operating Systems



Quantum Optimizers

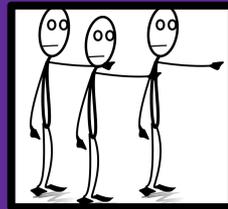
Talia Ringer, Karl Palmskog, Ilya Sergey, Milos Gligoric and Zachary Tatlock (2019), QED at Large: A Survey of Engineering of Formally Verified Software, Foundations and Trends in Programming Languages: Vol. 5, No. 2-3, pp 102–281.

Then vs. Now

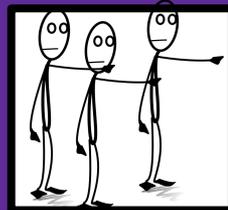
20+ person-years
~1,000,000 LOP

Proof Engineers

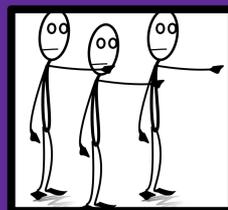
Proof Assistant



seL4



Confidentiality
& Integrity



Proof



Proof automation makes it easier to develop and maintain **verified systems** using **proof assistants**.

Traditional automation:

- + predictable
- + dependable
- + understandable
- limited in scope
- takes expertise to extend

Language models:

- *unpredictable*
- *not* dependable
- *not* understandable
- + *not very* limited in scope
- + takes *little* expertise to extend

Best of both worlds?

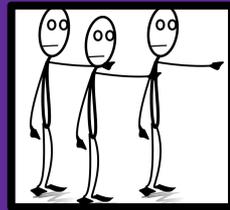
- + predictable
- + dependable
- + understandable
- + *not very* limited in scope
- + takes *little* expertise to extend

Now vs. Future

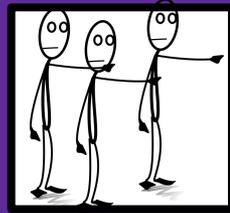
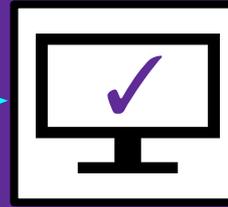
Not that much
work, lots of help?

Proof Engineers

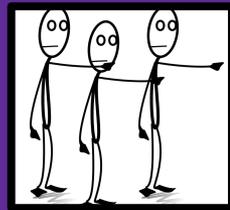
Proof Assistant



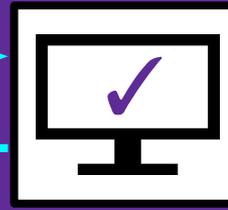
seL4



Confidentiality
& Integrity



Proof

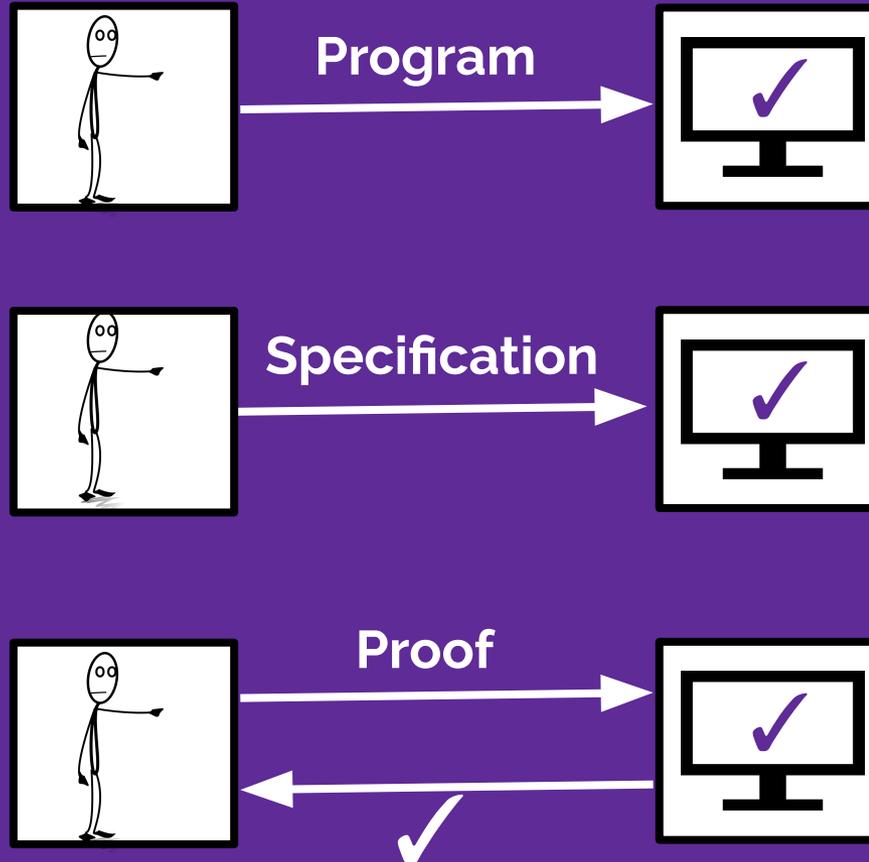


- 1. Proof Assistants**
- 2. Traditional Automation**
- 3. LM-Based Automation**
- 4. Best of Both Worlds**
- 5. Opportunities**

- 1. Proof Assistants**
- 2. Traditional Automation**
- 3. LM-Based Automation**
- 4. Best of Both Worlds**
- 5. Opportunities**

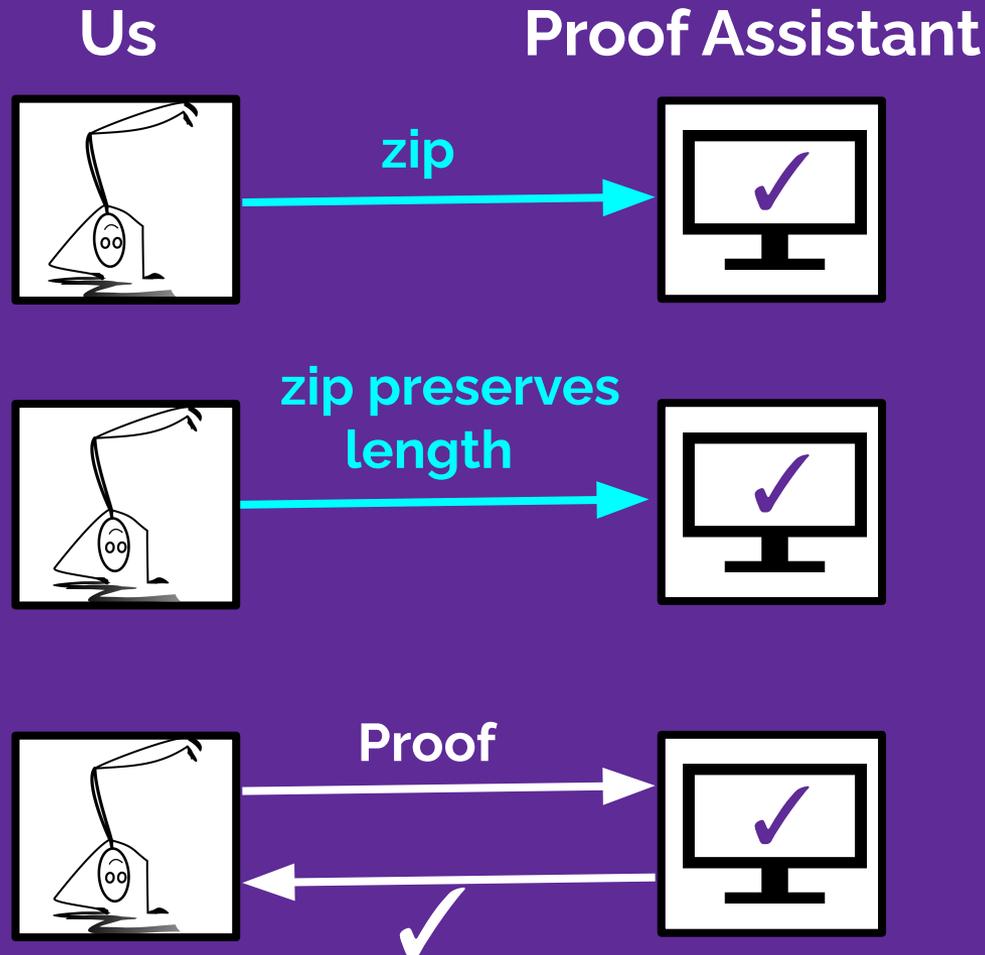
Proof Engineer

Proof Assistant



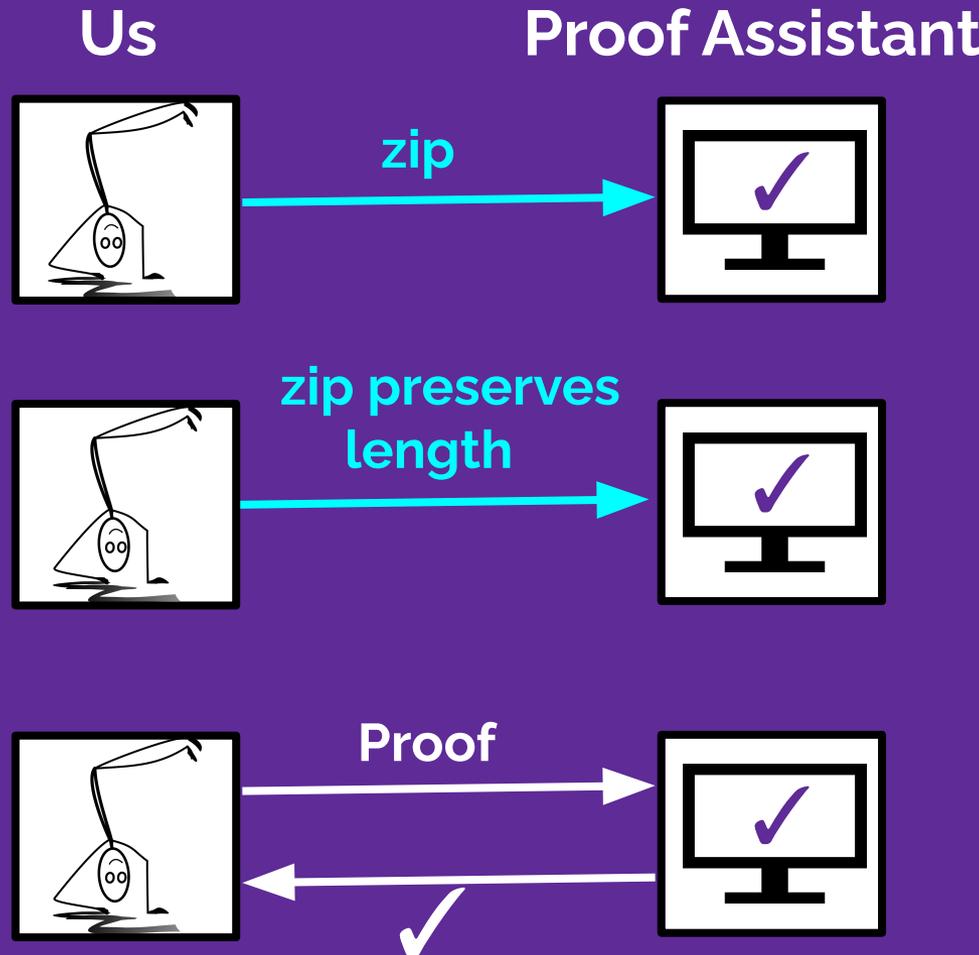
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



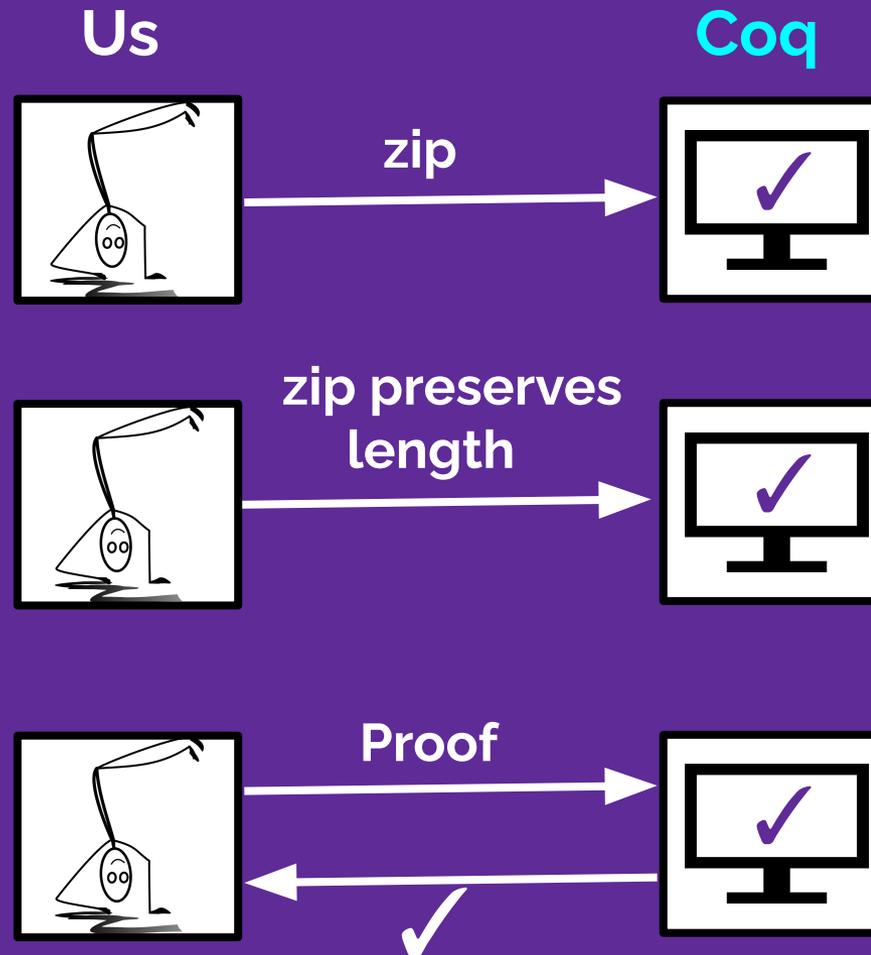
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



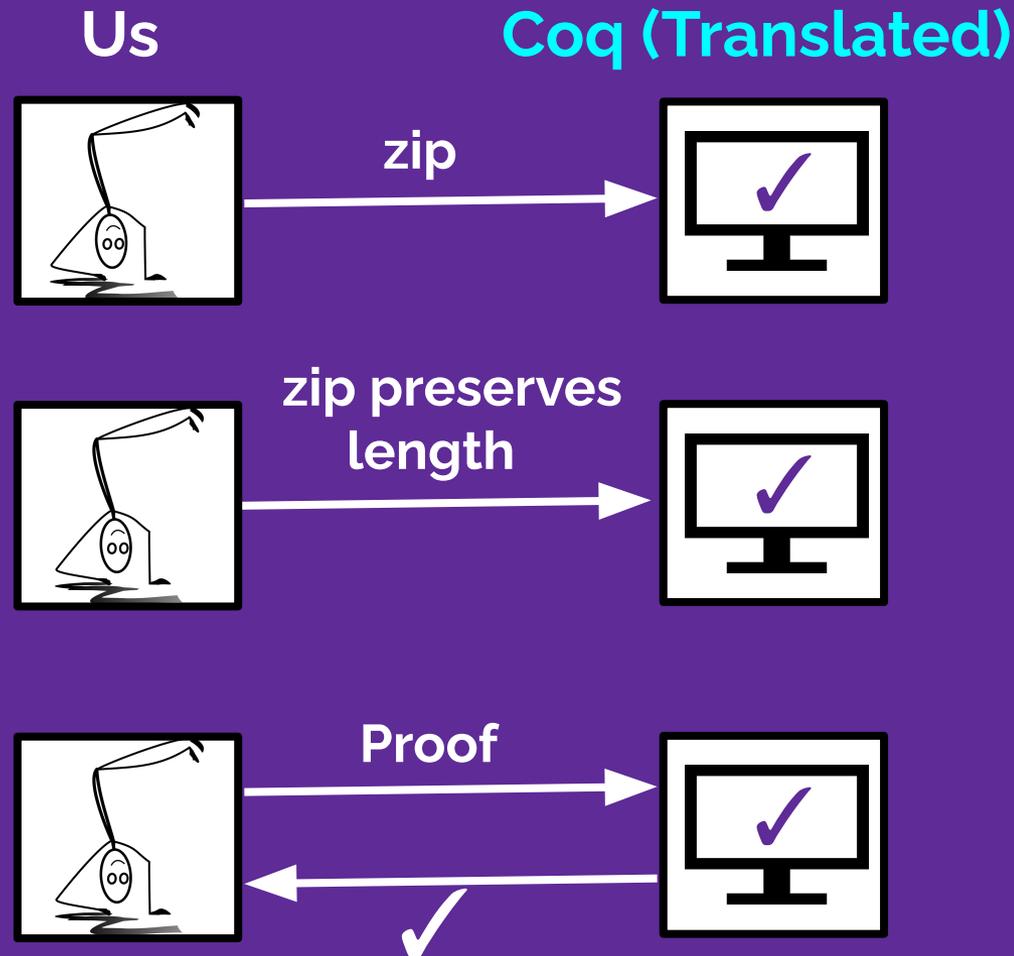
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



Proof Assistants (Part 1 of 5)

List Zip Preserves Length



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

list $\langle T \rangle :=$
| [] : list $\langle T \rangle$
| cons : $T \rightarrow$ list $\langle T \rangle \rightarrow$ list $\langle T \rangle$



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

`list <T> :=`
| `[]` : `list <T>`
| `cons` : `T` \rightarrow `list <T>` \rightarrow `list <T>`

Proof Assistants (Part 1 of 5)

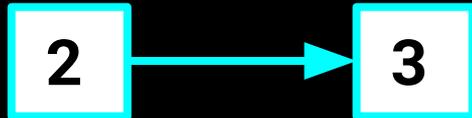
List Zip Preserves Length

```
list <T> :=  
| [] : list <T>  
| cons : T → list <T> → list <T>
```

List Zip Preserves Length

list $\langle T \rangle$:=
| [] : list $\langle T \rangle$
| cons : $T \rightarrow$ list $\langle T \rangle \rightarrow$ list $\langle T \rangle$

1



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

list $\langle T \rangle$:=
| [] : list $\langle T \rangle$
| **cons** : $T \rightarrow$ list $\langle T \rangle \rightarrow$ **list** $\langle T \rangle$



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
length <T> (l : list <T>) : nat :=  
  if l = [] then  
    0  
  else  
    1 + length (tail l)
```



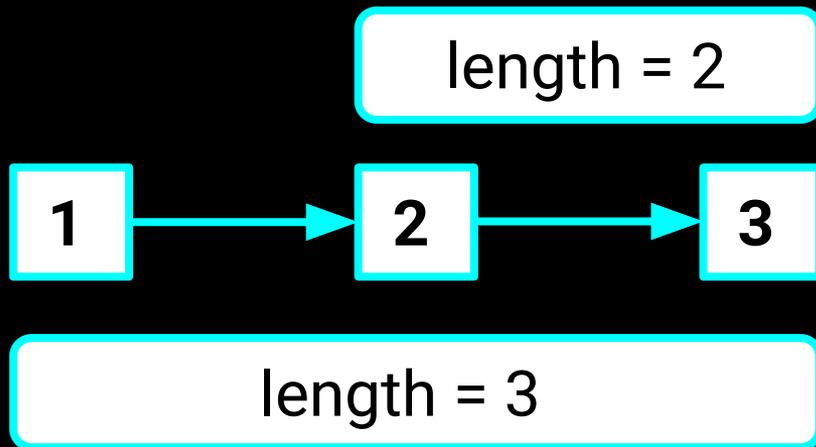
Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
length <T> (l : list <T>) : nat :=  
  if l = [] then  
    0  
  else  
    1 + length (tail l)
```

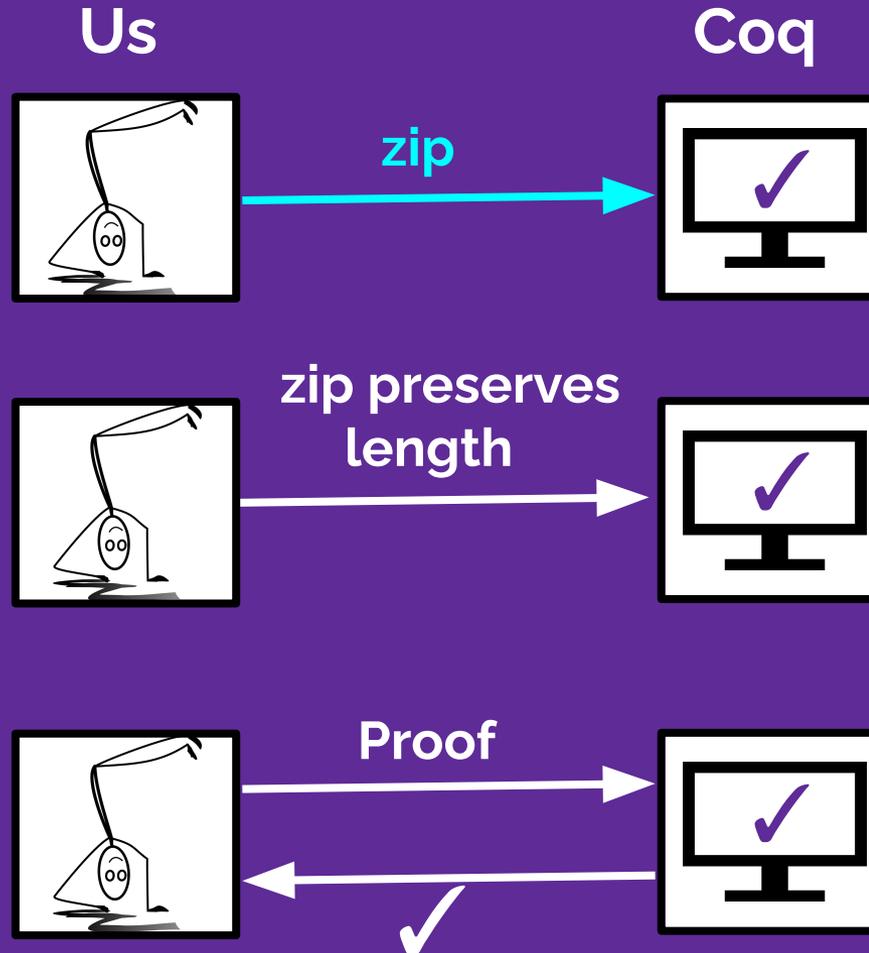
List Zip Preserves Length

```
length <T> (l : list <T>) : nat :=  
  if l = [] then  
    0  
  else  
    1 + length (tail l)
```



Proof Assistants (Part 1 of 5)

List Zip Preserves Length



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

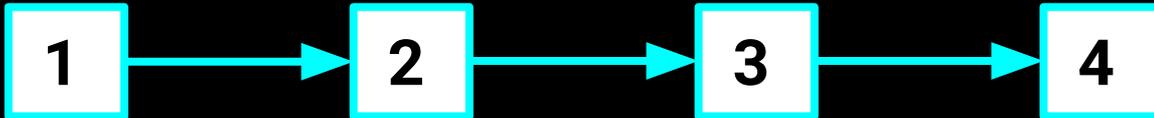
```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```

List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```

List Zip Preserves Length

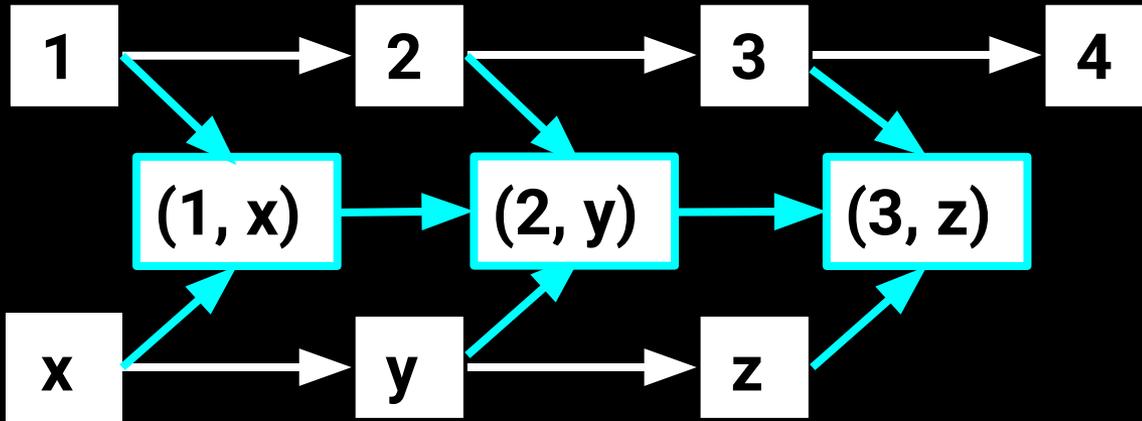
```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

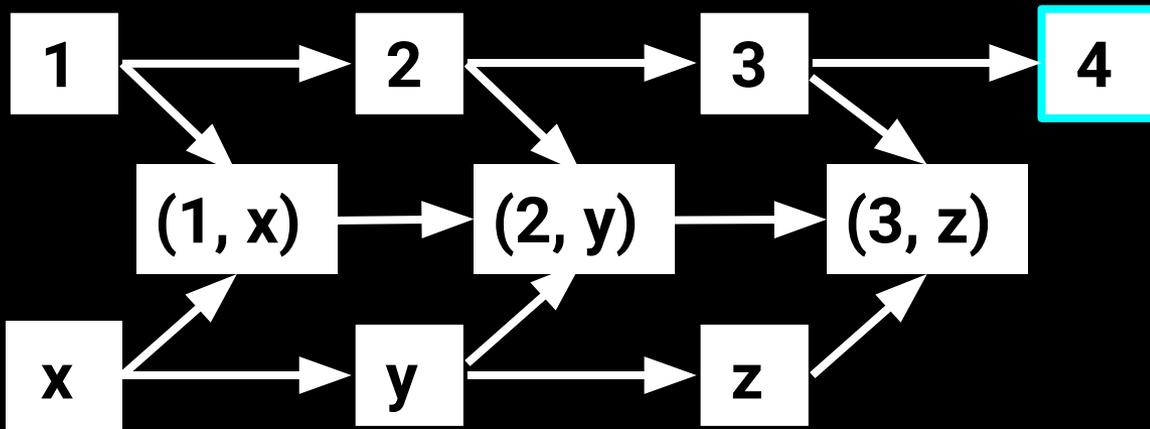
```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

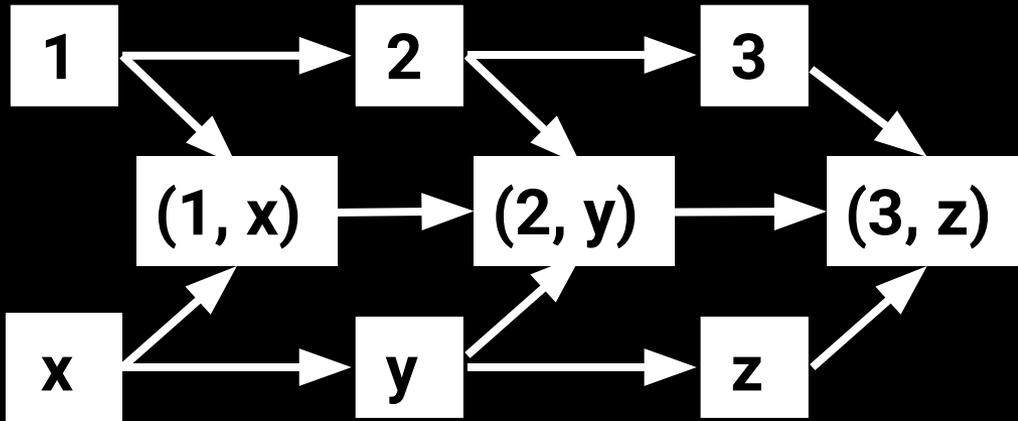
```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



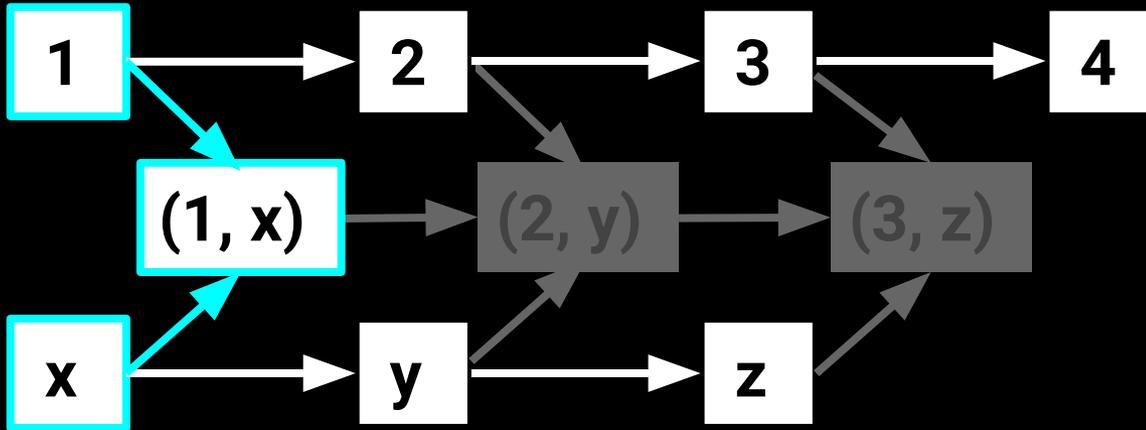
Proof Assistants (Part 1 of 5)

List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```

List Zip Preserves Length

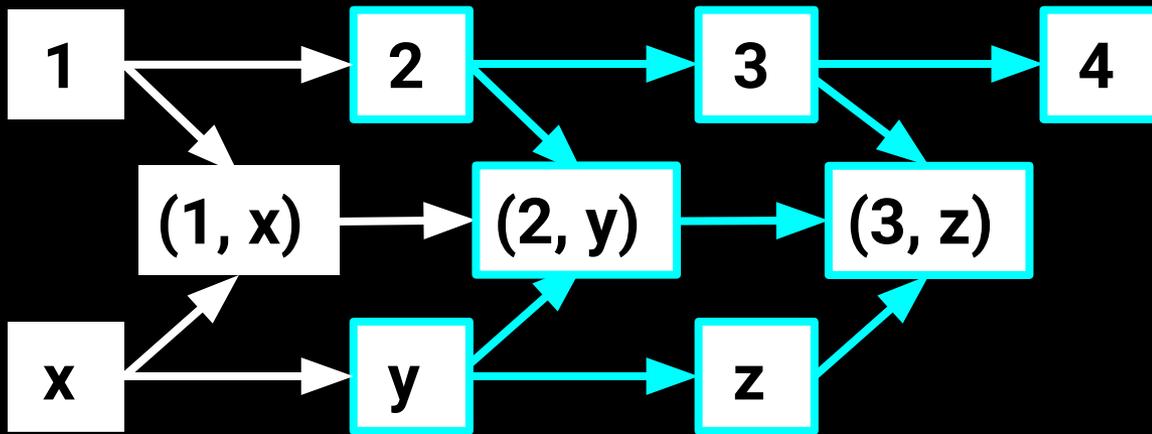
```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



Proof Assistants (Part 1 of 5)

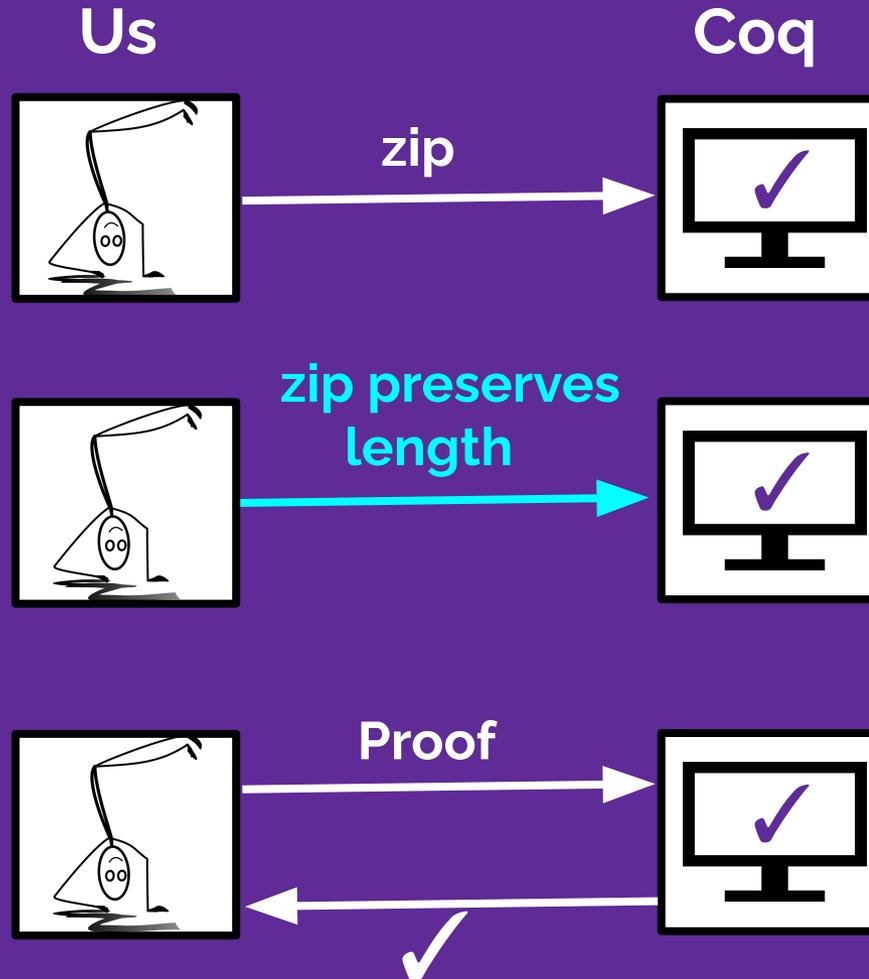
List Zip Preserves Length

```
zip <A, B> (l1 : list <A>) (l2 : list <B>) : list <(A, B)> :=  
  if l1 = [] or l2 = [] then  
    []  
  else  
    (head l1, head l2) :: (zip (tail l1) (tail l2))
```



Proof Assistants (Part 1 of 5)

List Zip Preserves Length



Proof Assistants (Part 1 of 5)

List Zip Preserves Length

Theorem zip_preserves_length :

$\forall \langle A, B \rangle (l1 : \text{list } \langle A \rangle) (l2 : \text{list } \langle B \rangle),$
length l1 = length l2 \rightarrow
length (zip l1 l2) = length l1.

List Zip Preserves Length

Theorem `zip_preserves_length` :

$\forall \langle A, B \rangle (l1 : \text{list } \langle A \rangle) (l2 : \text{list } \langle B \rangle),$

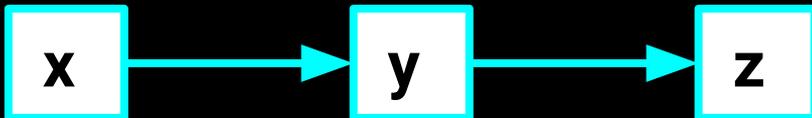
length l1 = length l2 \rightarrow

`length (zip l1 l2) = length l1`.

length = 3



length = 3

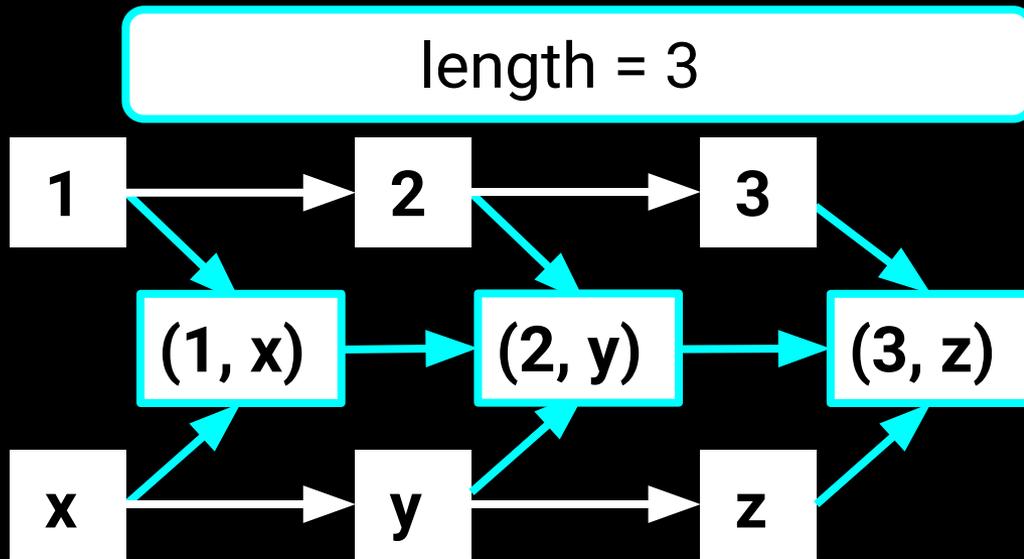


Proof Assistants (Part 1 of 5)

List Zip Preserves Length

Theorem `zip_preserves_length` :

$\forall \langle A, B \rangle (l1 : \text{list } \langle A \rangle) (l2 : \text{list } \langle B \rangle),$
 $\text{length } l1 = \text{length } l2 \rightarrow$
 $\text{length } (\text{zip } l1 \ l2) = \text{length } l1.$



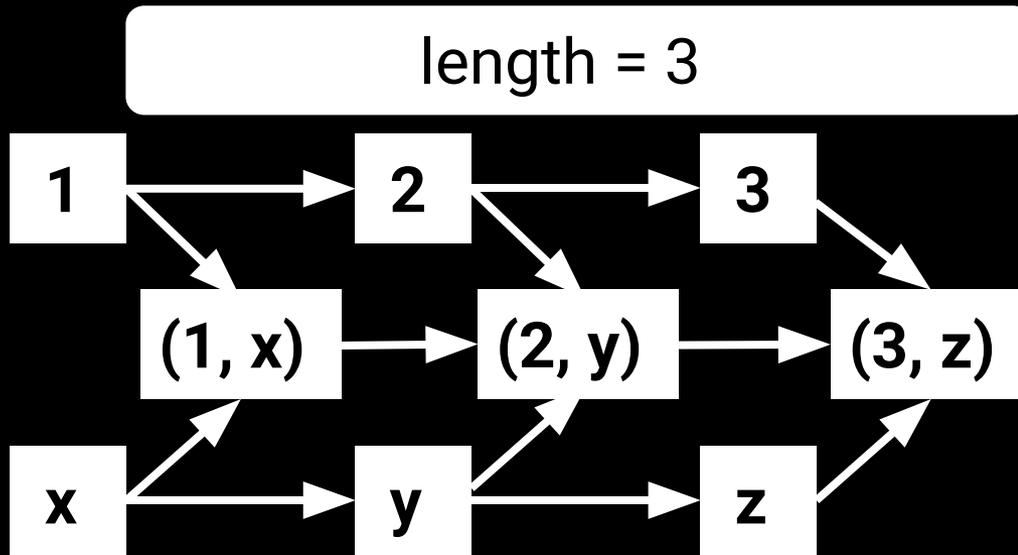
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



Theorem zip_preserves_length :

$\forall \langle A, B \rangle (l1 : \text{list } \langle A \rangle) (l2 : \text{list } \langle B \rangle),$
length l1 = length l2 \rightarrow
length (zip l1 l2) = length l1.



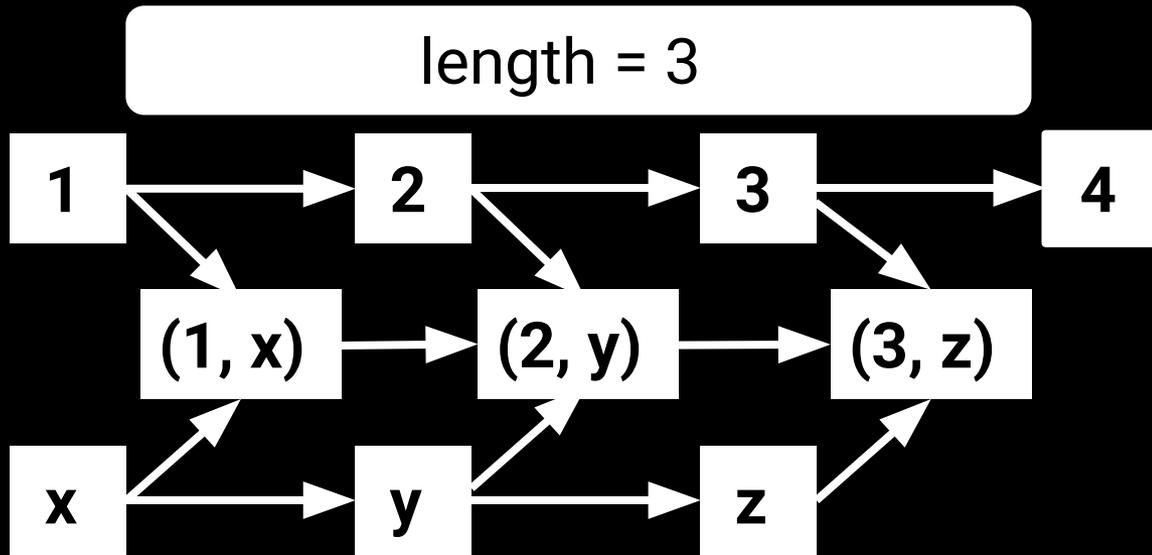
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



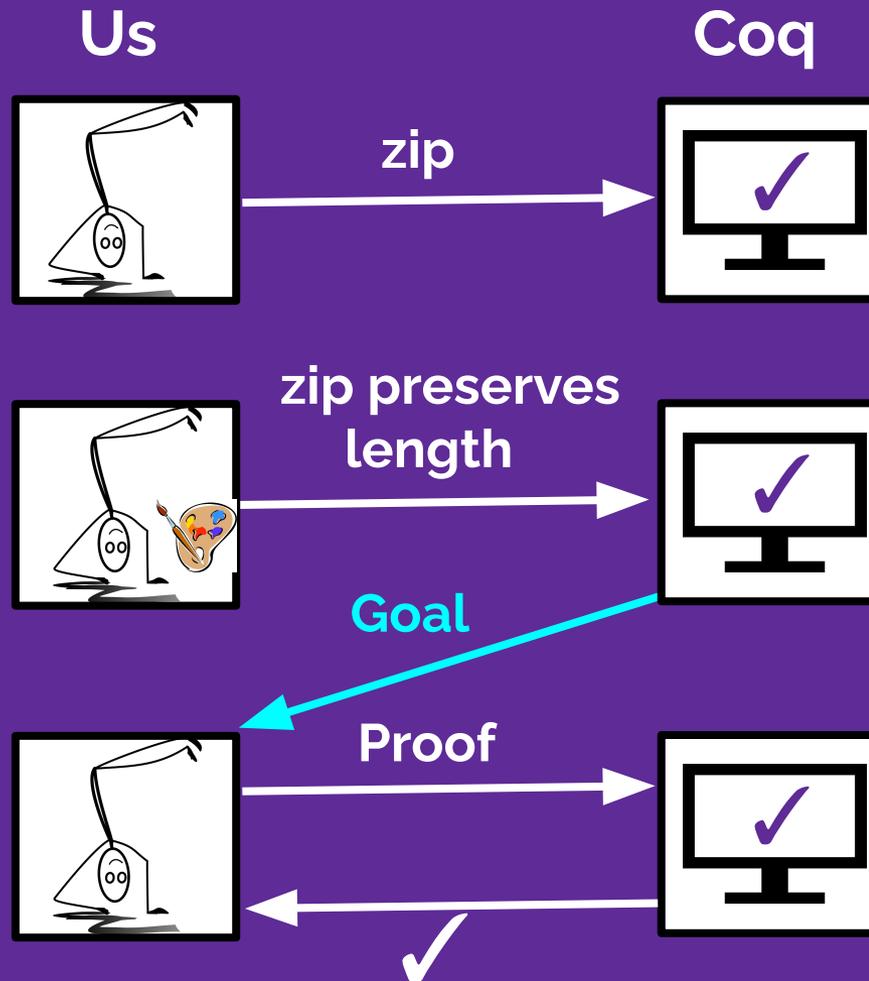
Theorem `zip_preserves_length` :

$\forall \langle A, B \rangle (l1 : \text{list } \langle A \rangle) (l2 : \text{list } \langle B \rangle),$
 $\text{length } (\text{zip } l1 \ l2) = \text{min } (\text{length } l1) \ (\text{length } l2).$



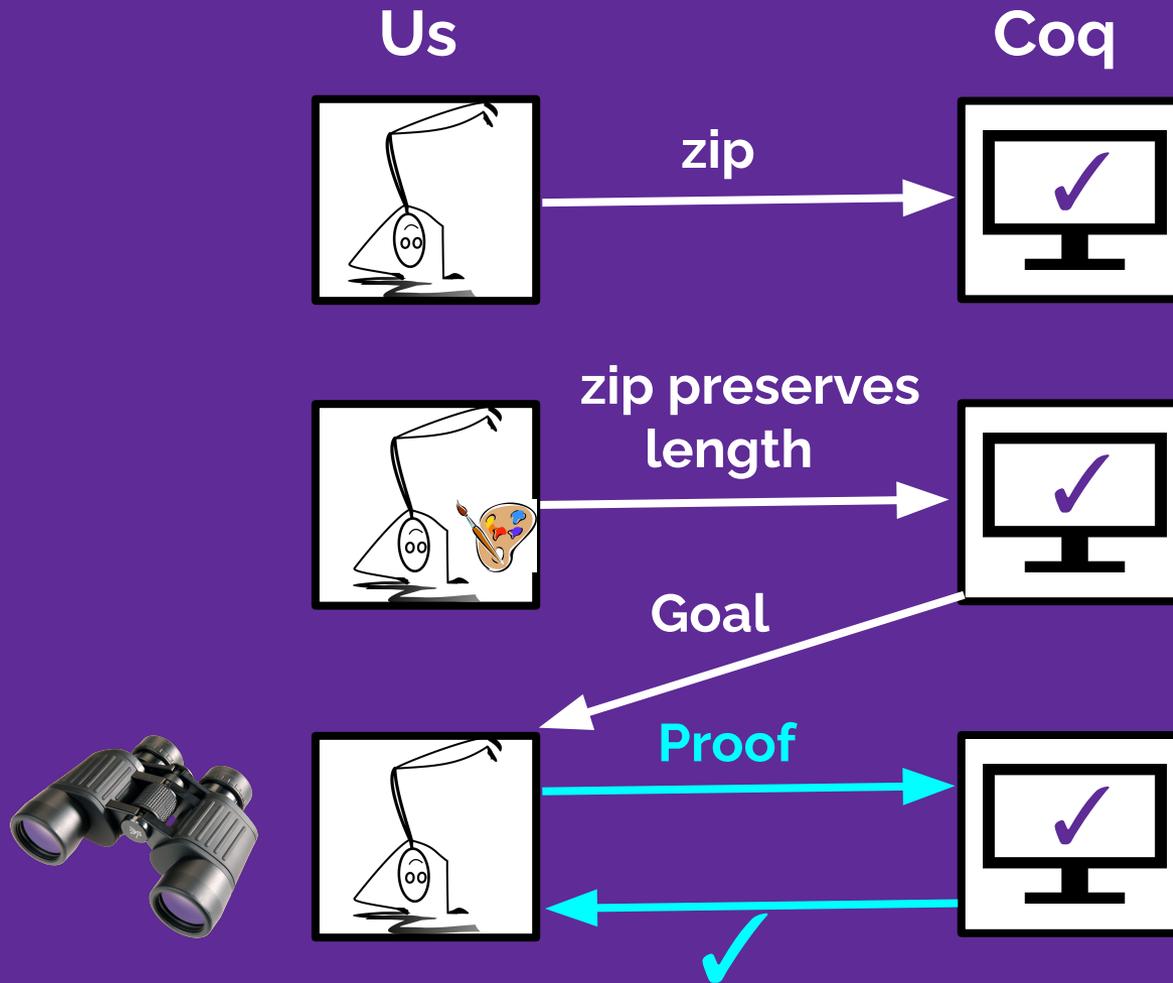
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



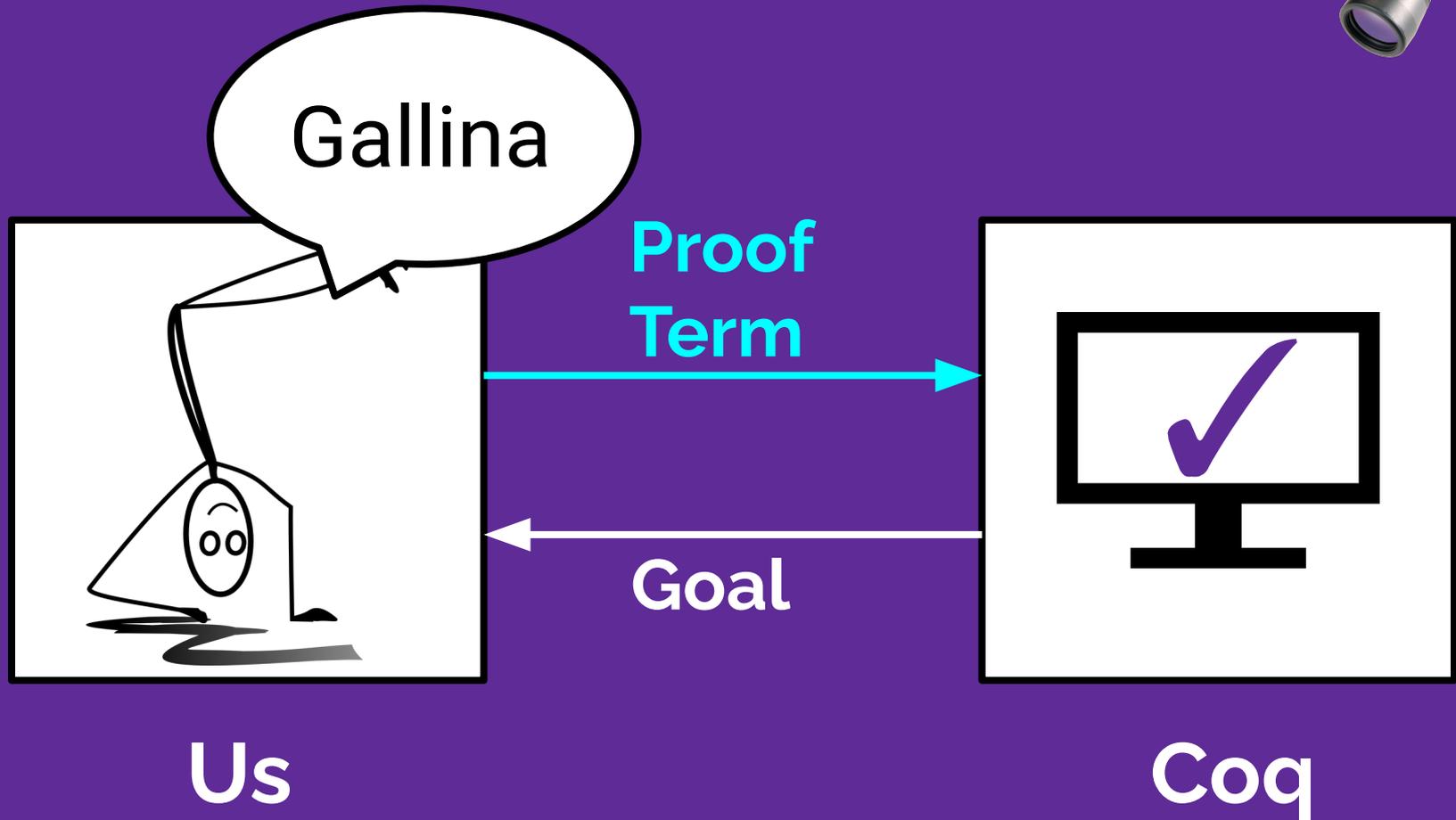
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



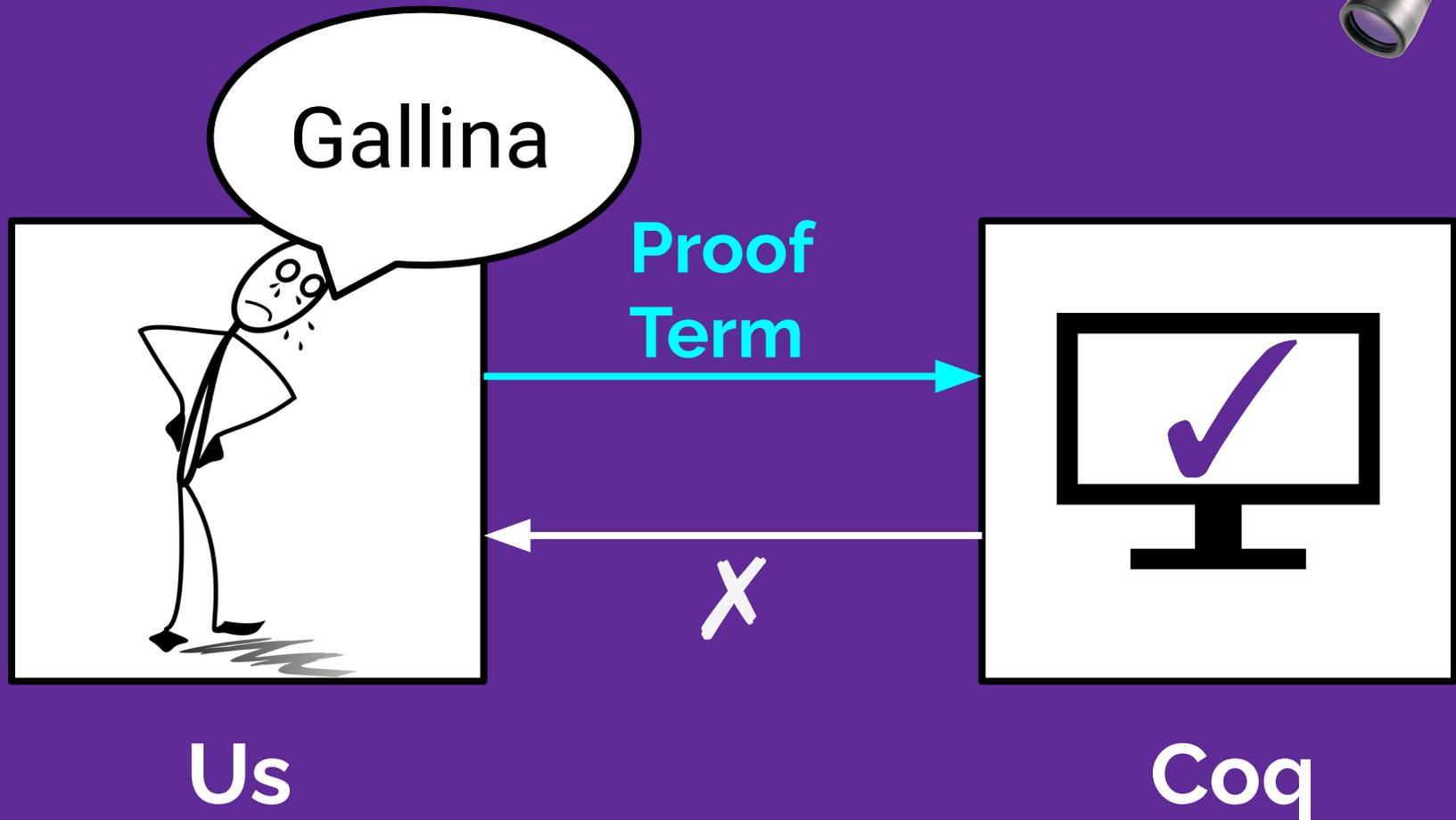
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



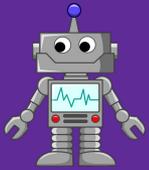
Proof Assistants (Part 1 of 5)

List Zip Preserves Length



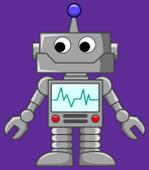
Proof Assistants (Part 1 of 5)

1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. Opportunities



Proof Automation

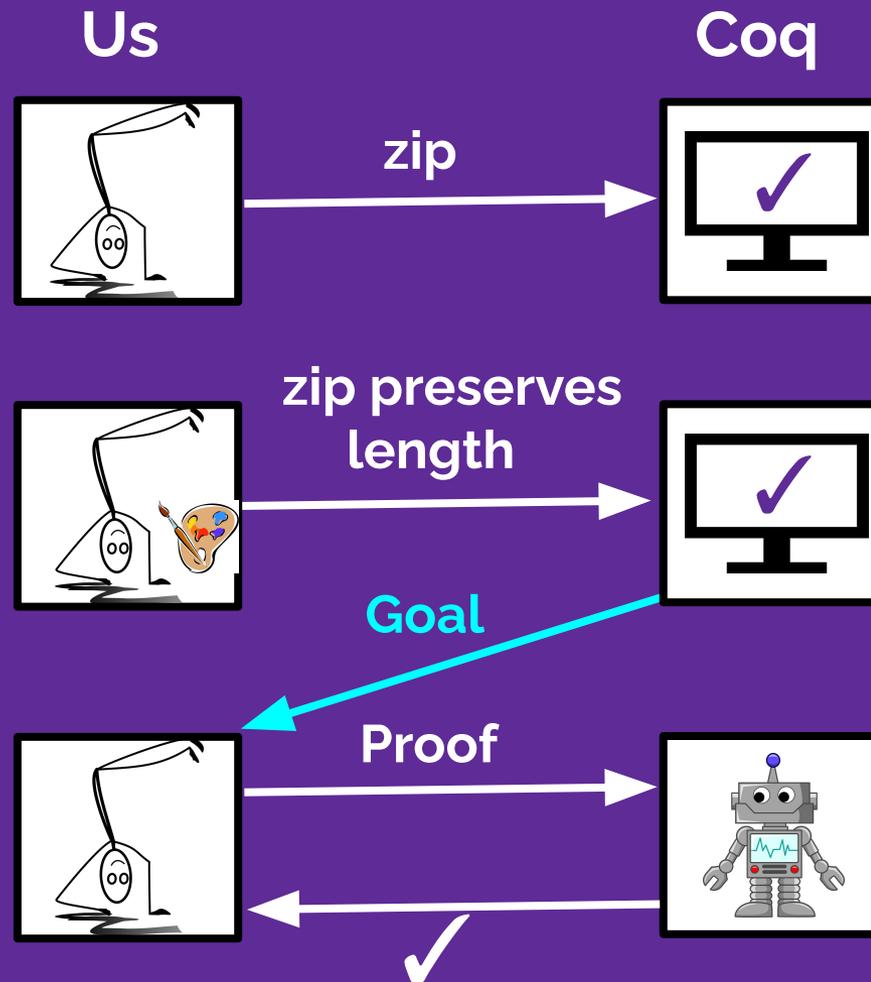
Traditional Automation (Part 2 of 5)



Proof Automation*

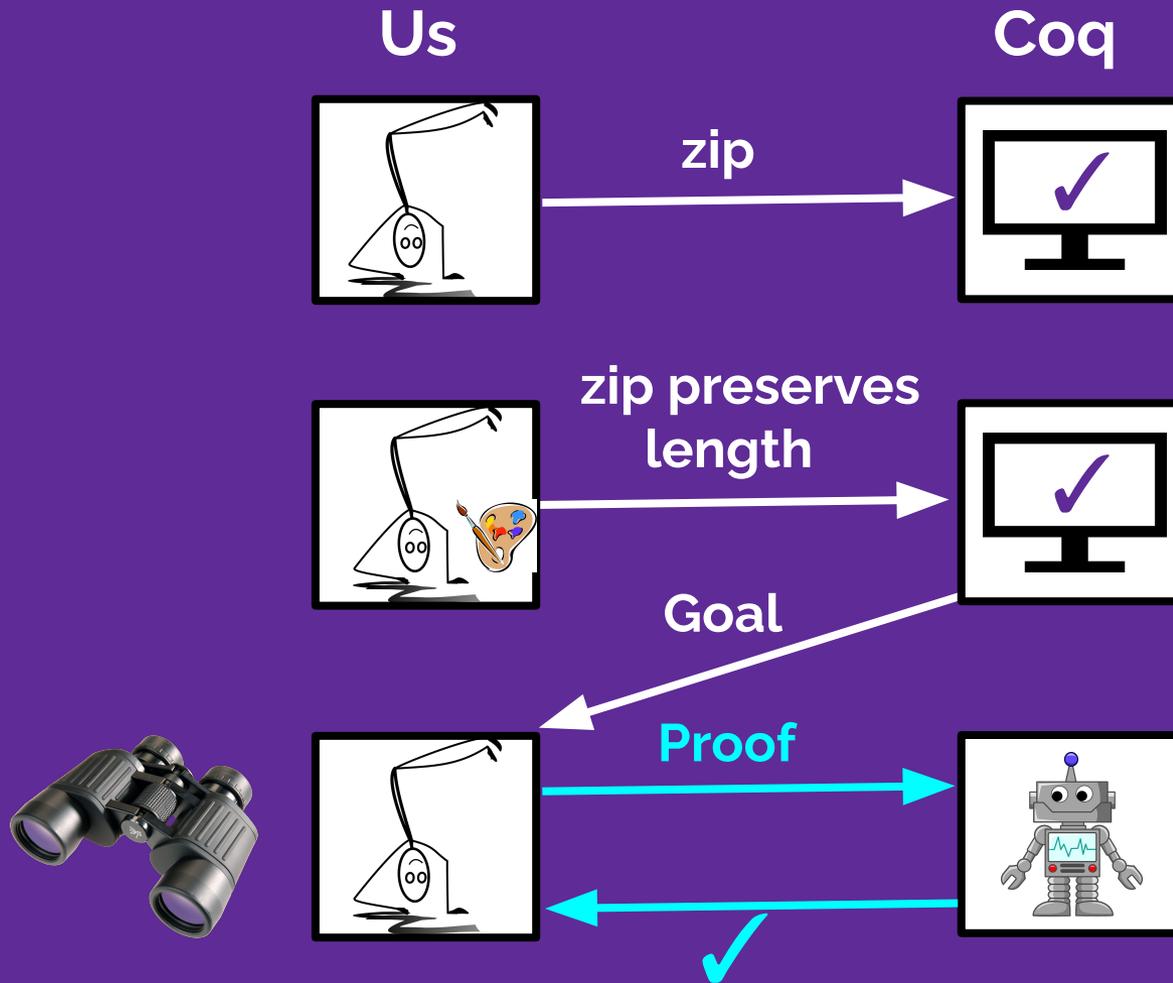
Traditional Automation (Part 2 of 5)

Proof Automation*



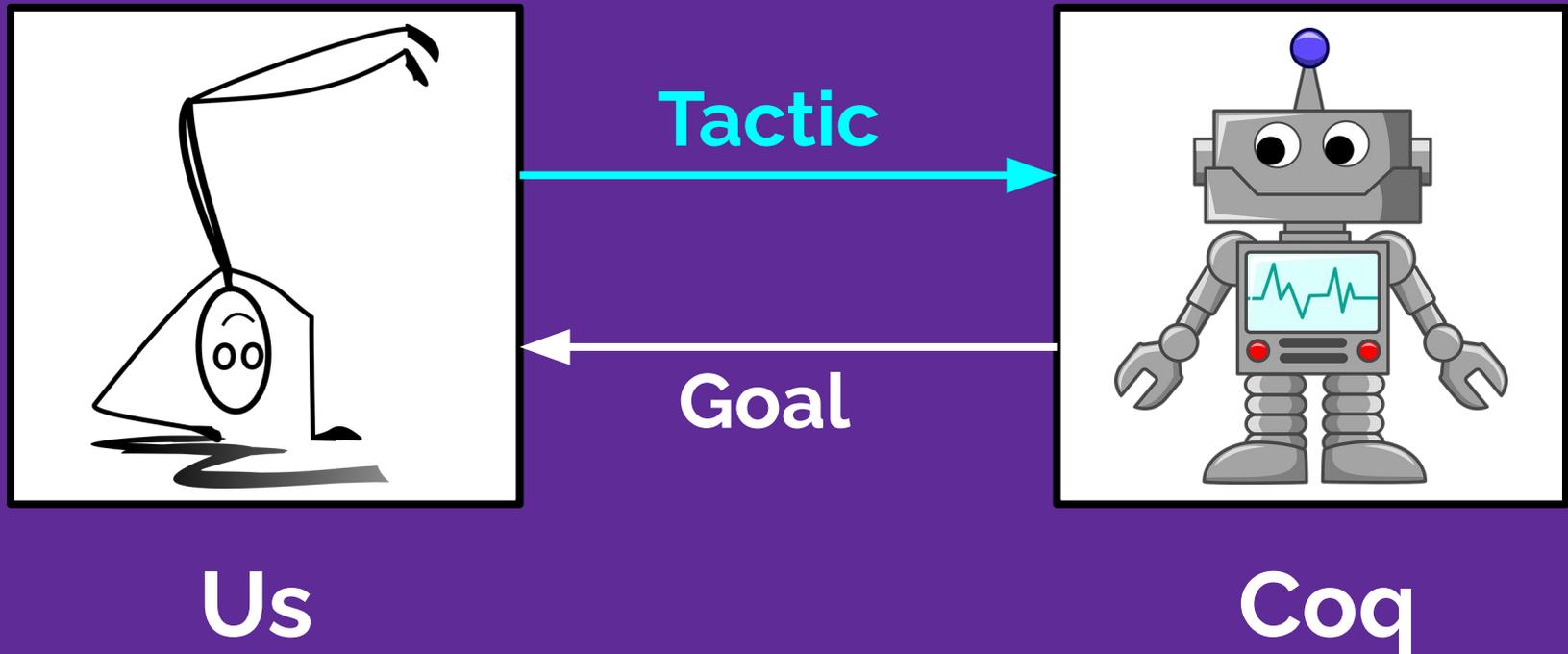
Traditional Automation (Part 2 of 5)

Proof Automation*



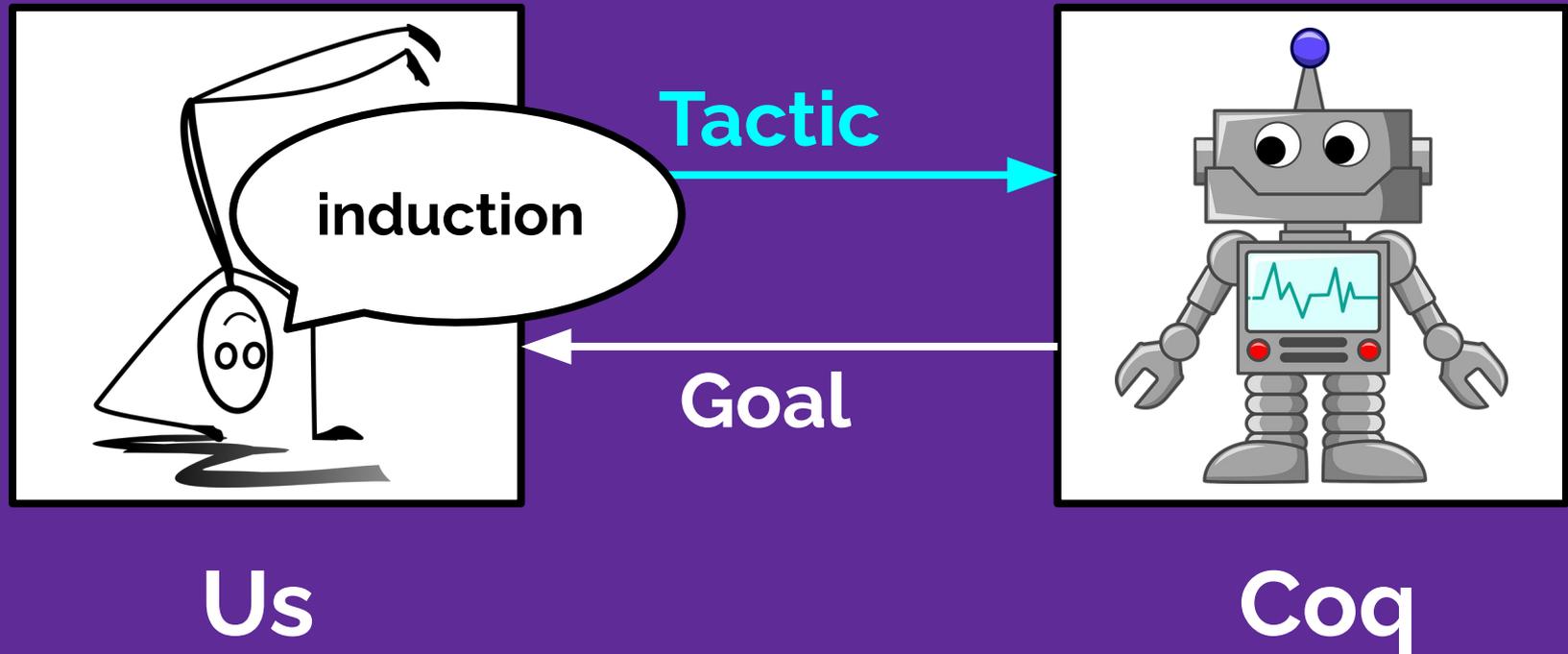
Traditional Automation (Part 2 of 5)

Proof Automation*



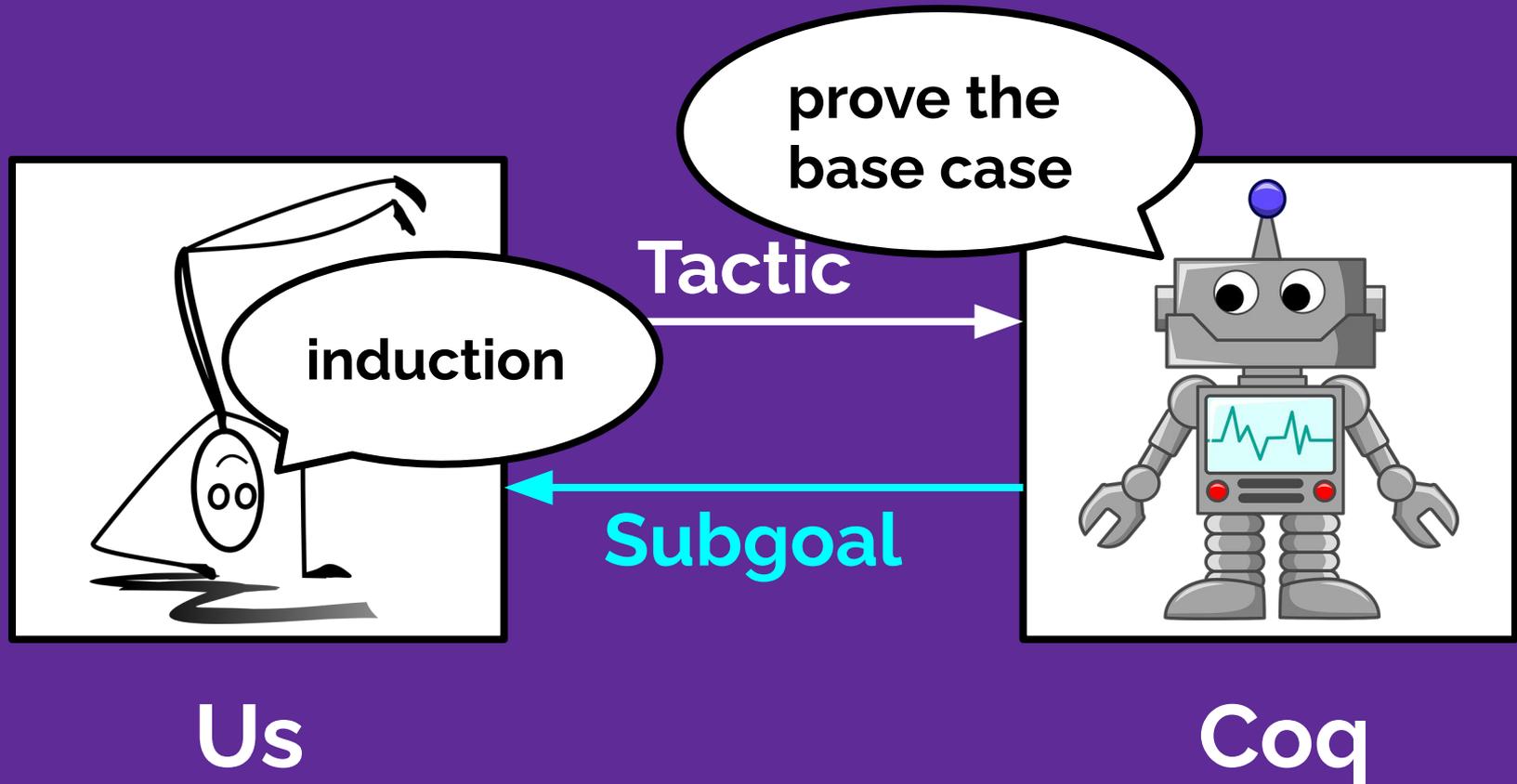
Traditional Automation (Part 2 of 5)

Proof Automation*



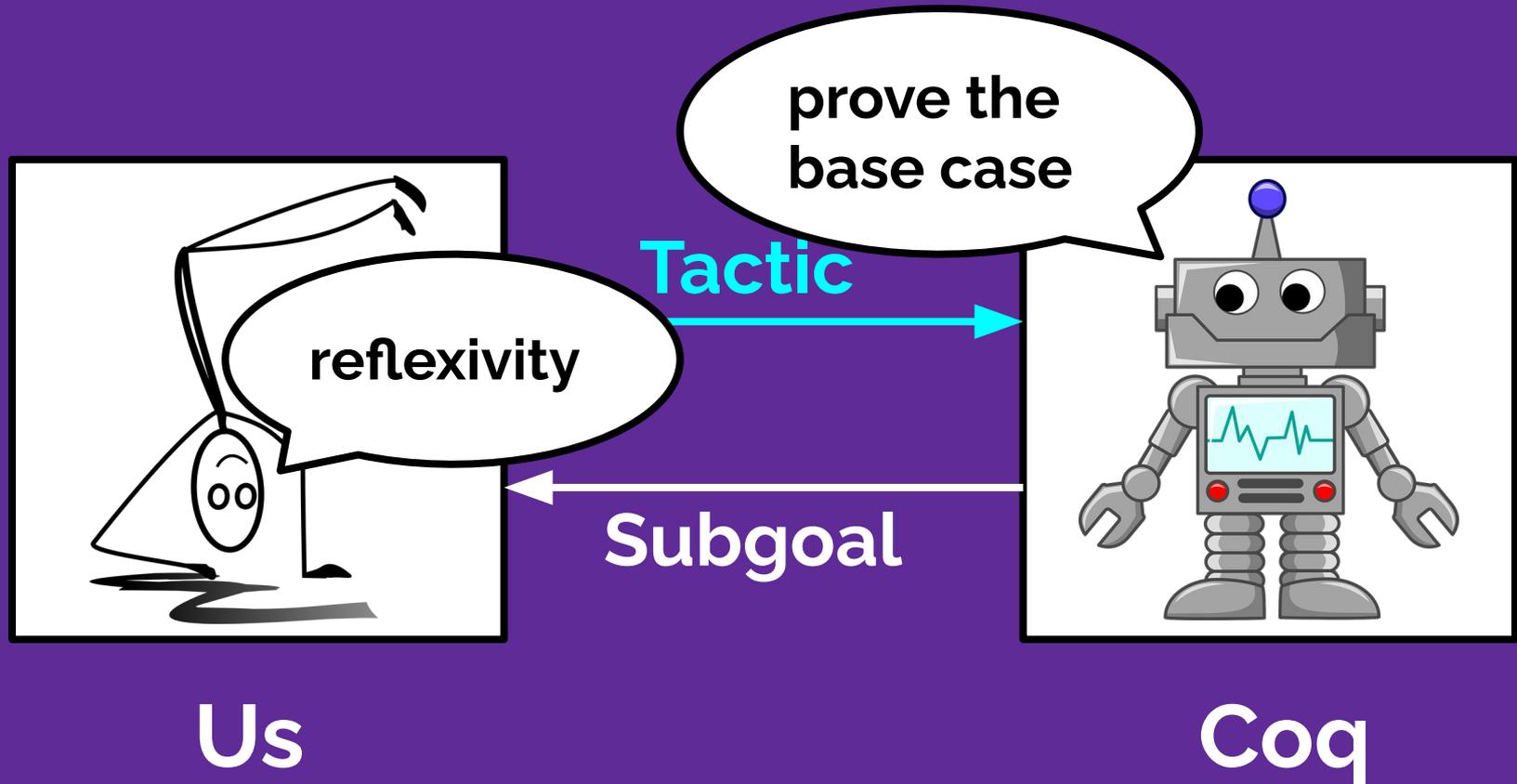
Traditional Automation (Part 2 of 5)

Proof Automation*



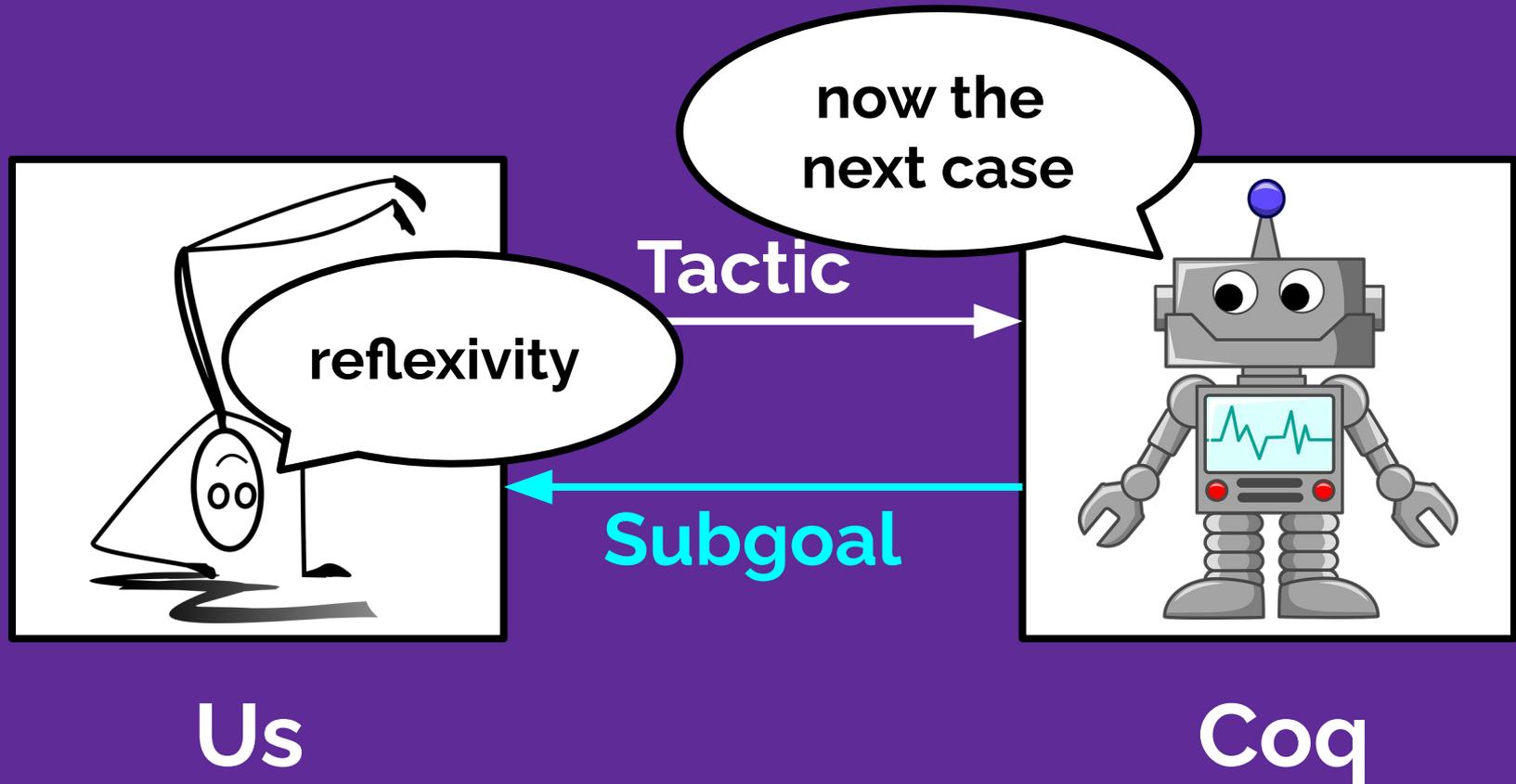
Traditional Automation (Part 2 of 5)

Proof Automation*



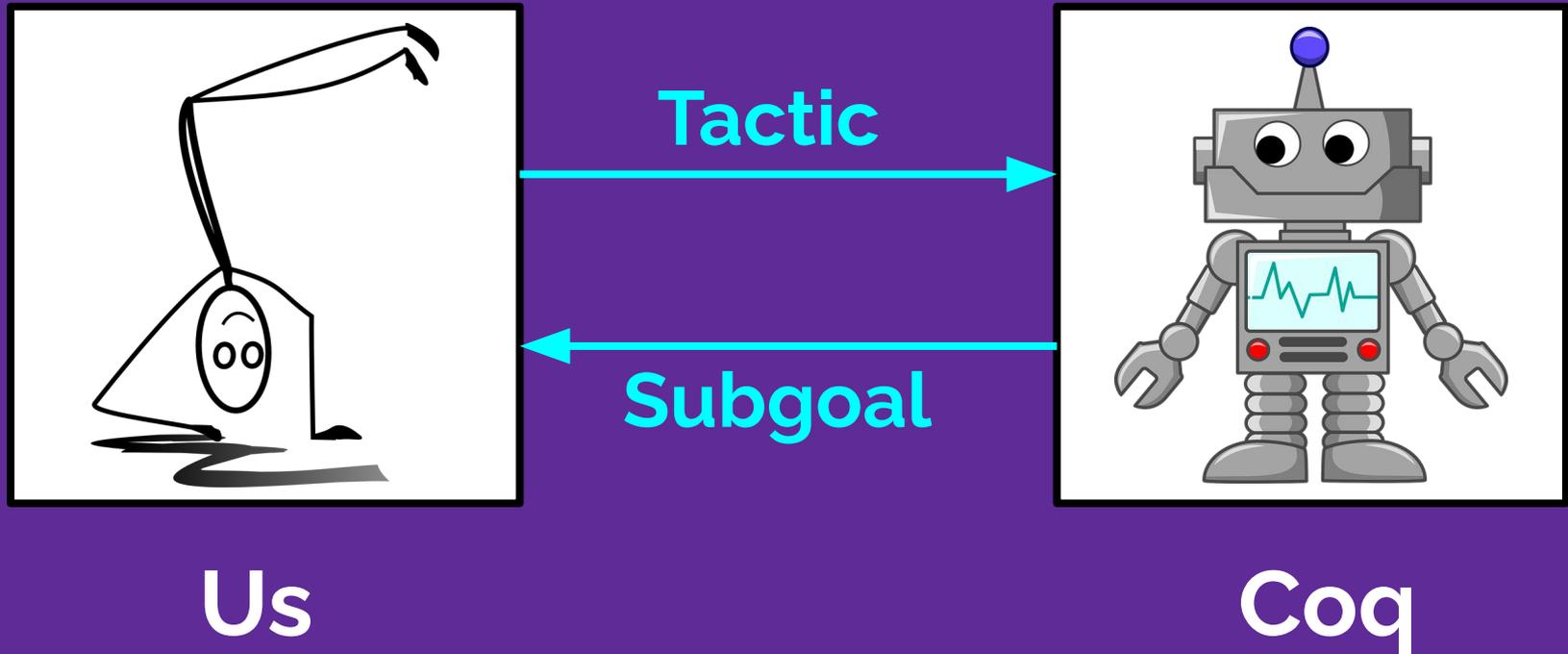
Traditional Automation (Part 2 of 5)

Proof Automation*



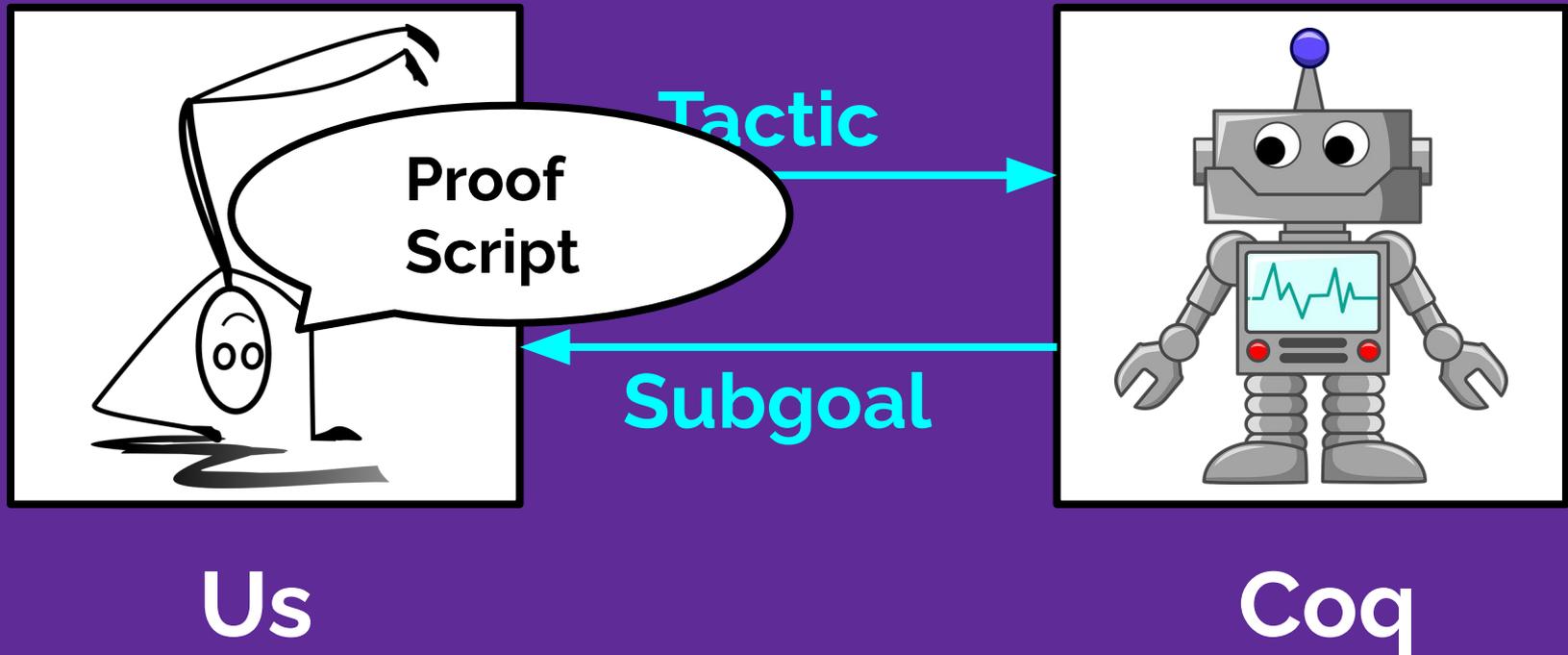
Traditional Automation (Part 2 of 5)

Proof Automation*



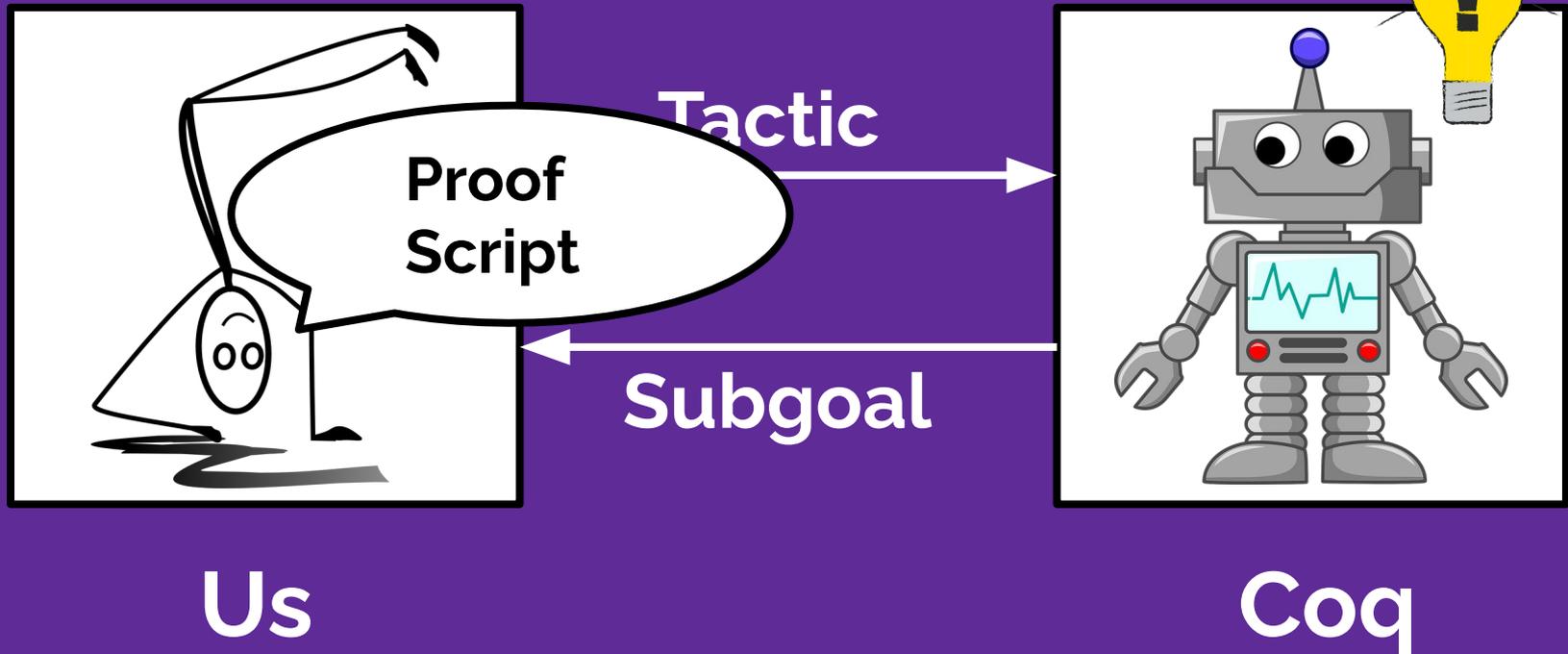
Traditional Automation (Part 2 of 5)

Proof Automation*



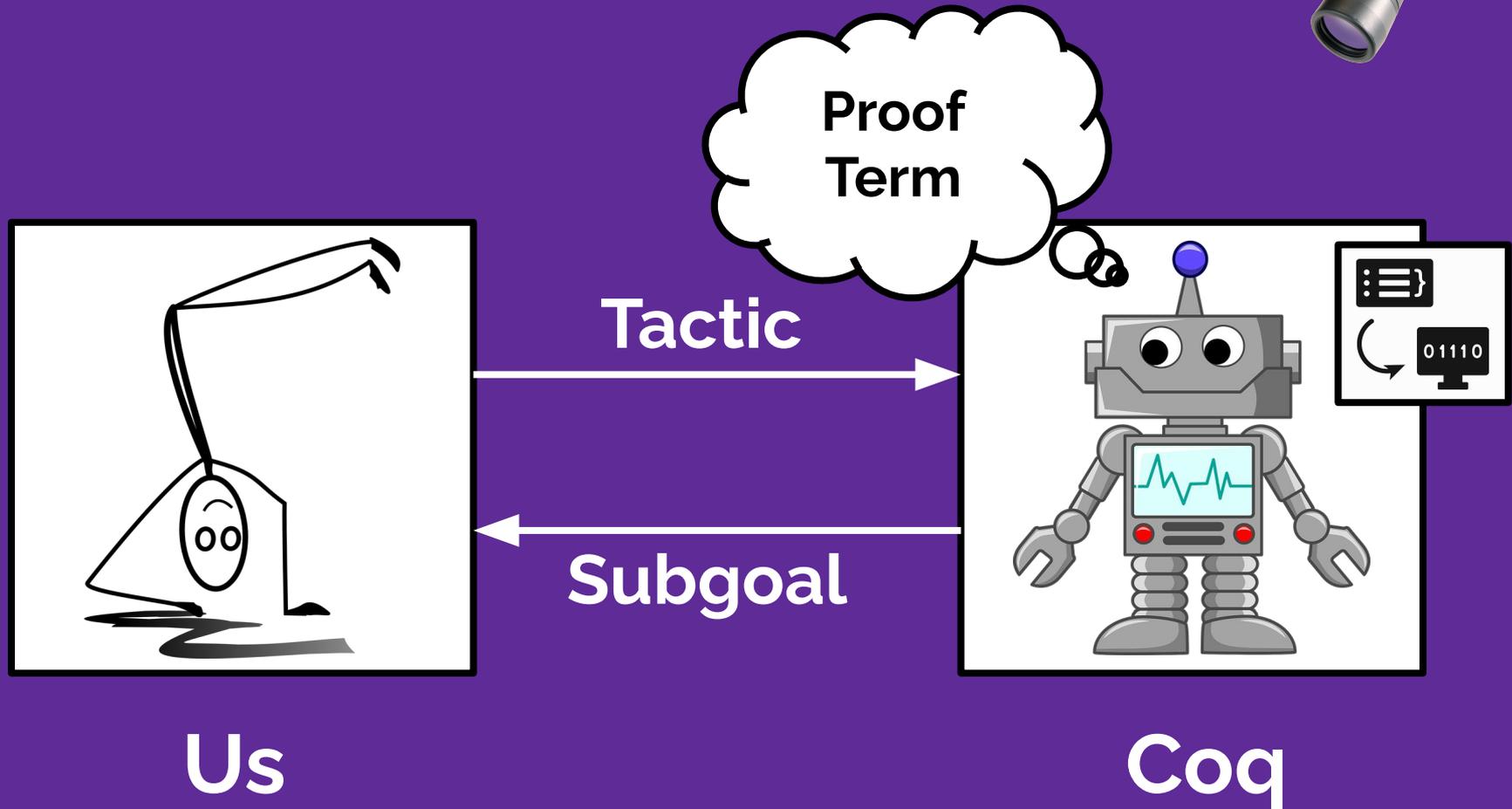
Traditional Automation (Part 2 of 5)

Proof Automation*



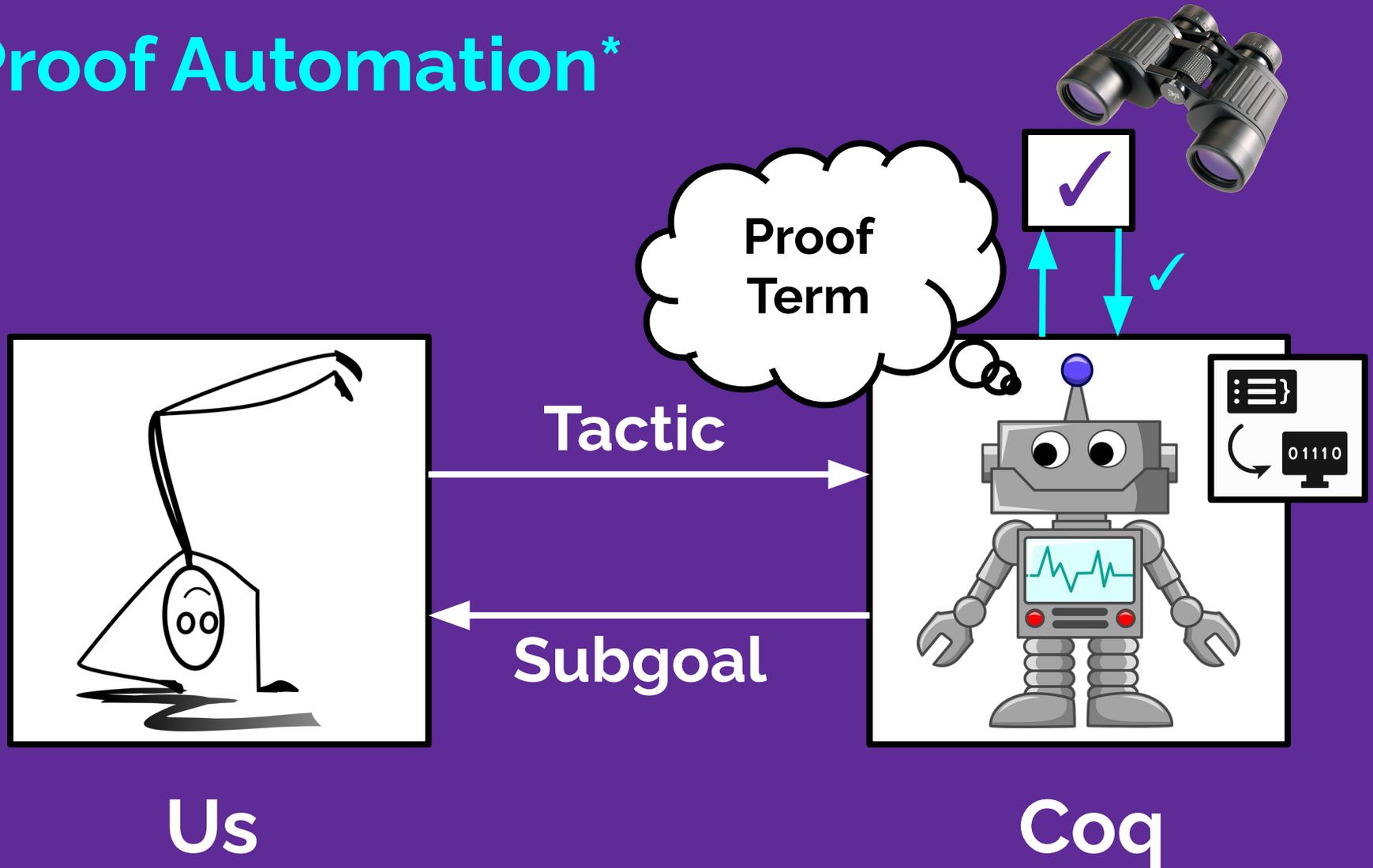
Traditional Automation (Part 2 of 5)

Proof Automation*



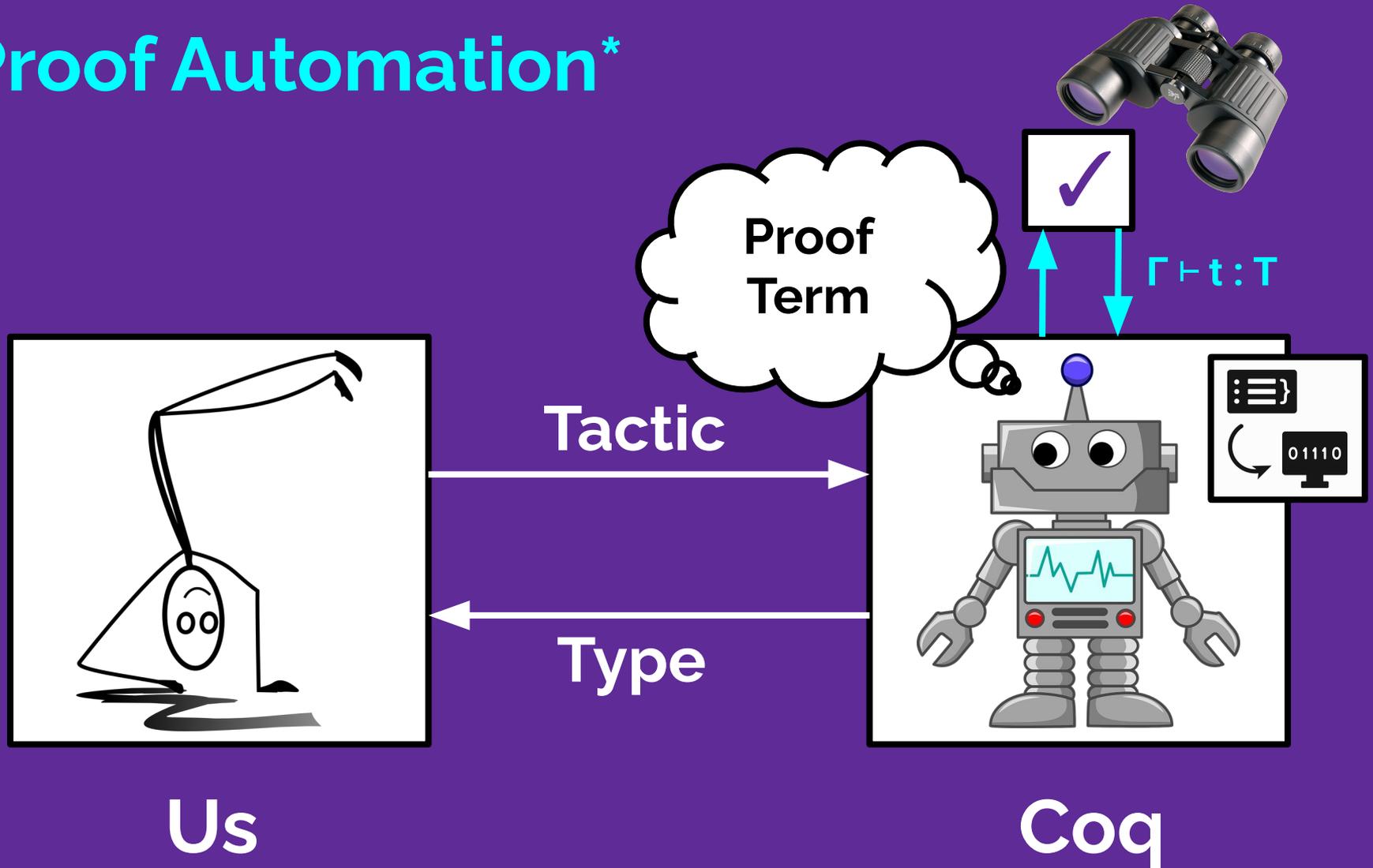
Traditional Automation (Part 2 of 5)

Proof Automation*



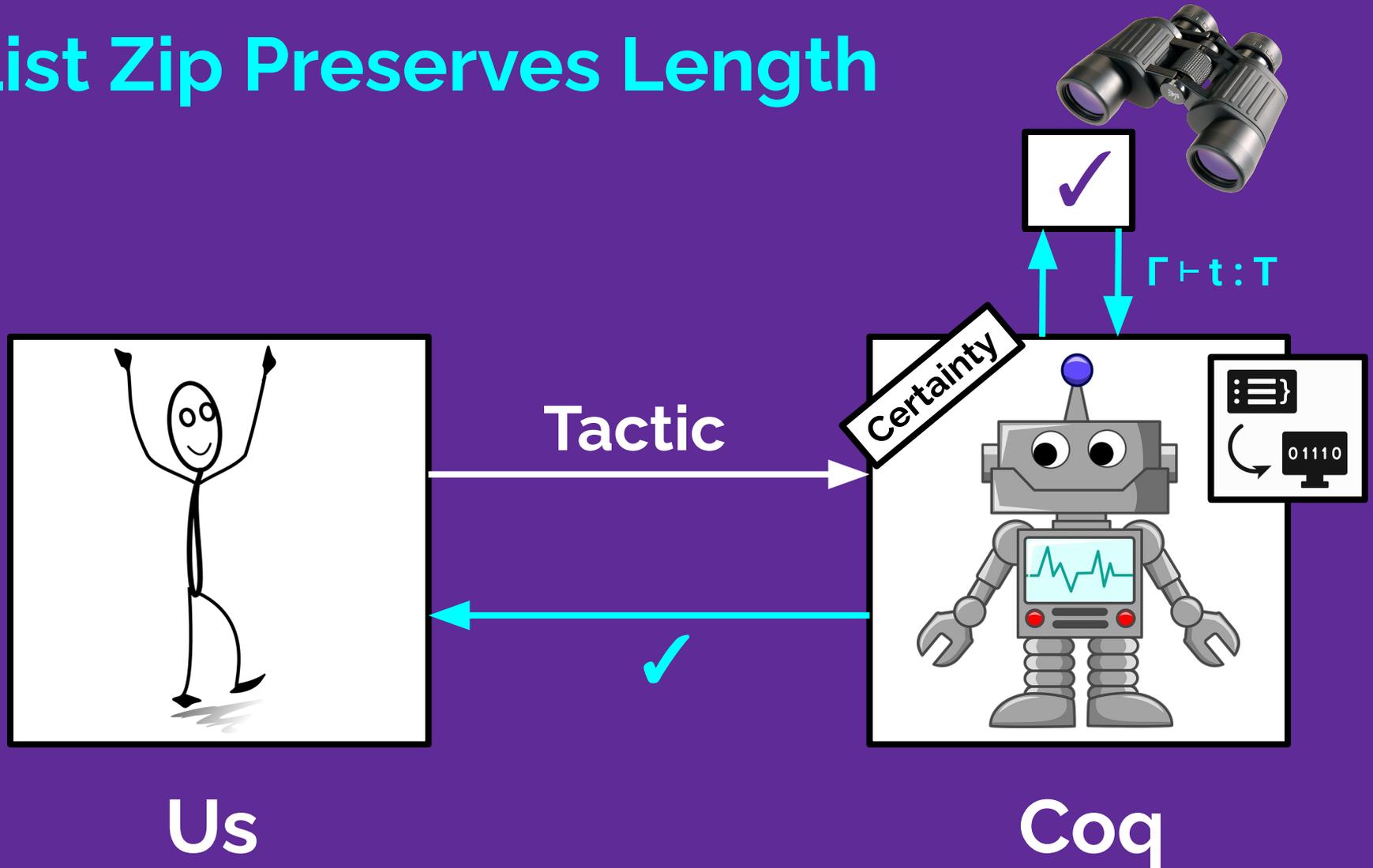
Traditional Automation (Part 2 of 5)

Proof Automation*



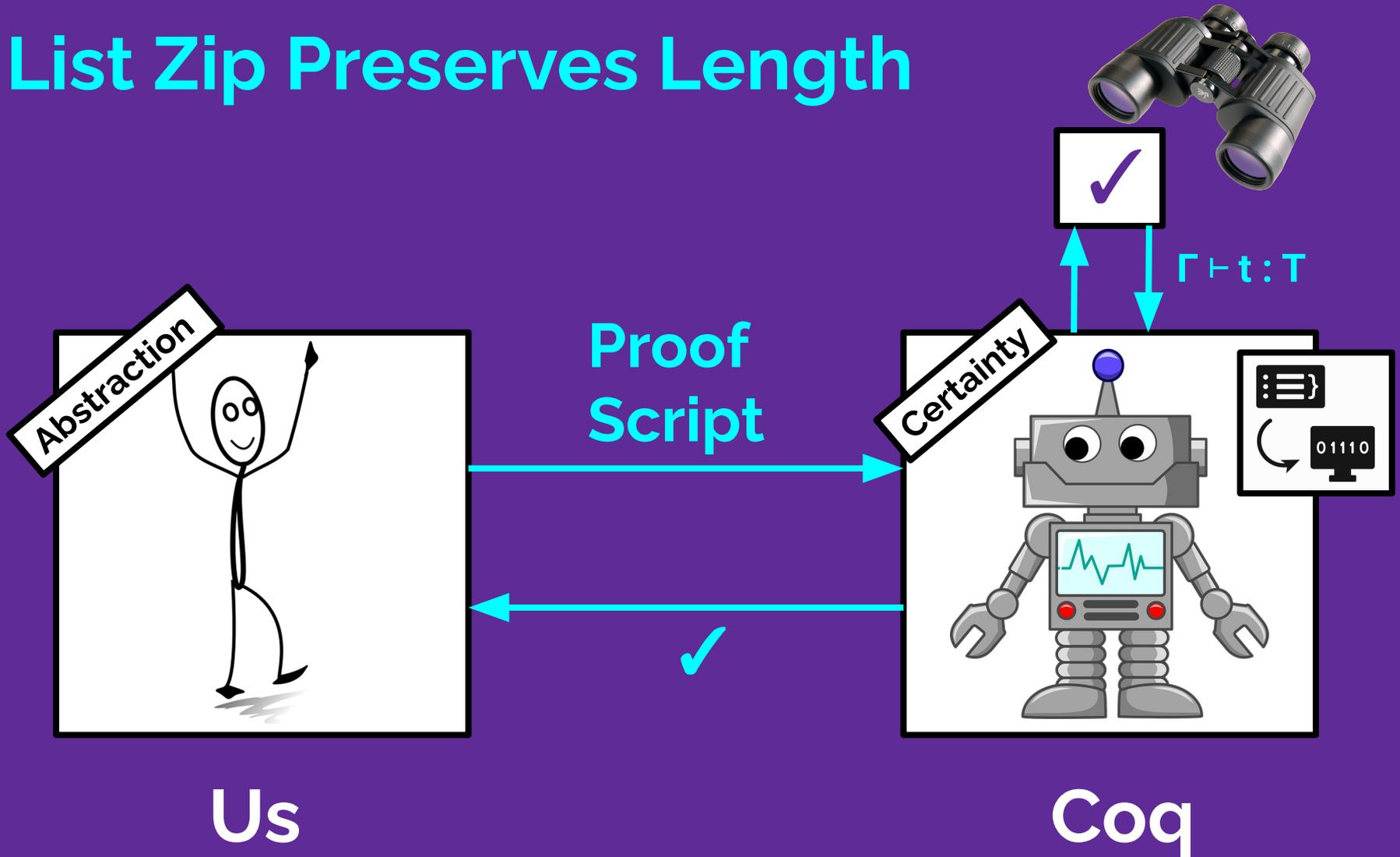
Traditional Automation (Part 2 of 5)

List Zip Preserves Length



Traditional Automation (Part 2 of 5)

List Zip Preserves Length



Traditional Automation (Part 2 of 5)

List Zip Preserves Length

```
intros T1 T2 l1. induction l11 as [|t1 tl1 IHtl1].  
- auto.2  
- intros l2. induction l23 as [|t2 tl2 IHtl2].  
  + intros H. auto.4  
  + intros H. simpl. rewrite IHtl1; auto.5
```

Abstraction



```
fun (T1 T2 : Type) (l1 : list T1) (l2 : list T2) =>  
  list_rect1 (fun (l1 : list T1) => ...)  
    (fun (l2 : list T2) _ => eq_refl)2  
    (fun (t1 : T1) (tl1 : list T1) (IHtl1 : ...) (l2 : list T2) =>  
      list_rect3 (fun (l2 : list T2) => ...)  
        (fun (H : ...) => eq_sym H)4  
        (fun (t2 : T2) (tl2 : list T2) (IHtl2 : ...) =>  
          fun (H : ...) => eq_rect_r ... eq_refl (IHtl1 ...) )5  
      l23)  
  l11  
  l2.
```

Certainty

List Zip Preserves Length

```
intros T1 T2 l1. induction l11 as [|t1 tl1 IHtl1].  
- auto.2  
- intros l2. induction l23 as [|t2 tl2 IHtl2].  
+ intros H. auto.4  
+ intros H. simpl. rewrite IHtl1; auto.5
```

Abstraction

Induction => Induction Principles



```
fun (T1 T2 : Type) (l1 : list T1) (l2 : list T2) =>  
  list_rect1 (fun (l1 : list T1) => ...)  
  (fun (l2 : list T2) _ => eq_refl)2  
  (fun (t1 : T1) (tl1 : list T1) (IHtl1 : ...) (l2 : list T2) =>  
    list_rect3 (fun (l2 : list T2) => ...)  
    (fun (H : ...) => eq_sym H)4  
    (fun (t2 : T2) (tl2 : list T2) (IHtl2 : ...) =>  
      fun (H : ...) => eq_rect_r ... eq_refl (IHtl1 ...) )5  
    l23)  
  l11  
  l2.
```

Certainty

Traditional Automation (Part 2 of 5)

List Zip Preserves Length

```
intros T1 T2 l1. induction l11 as [|t1 tl1 IHtl1].  
- auto.2  
- intros l2. induction l23 as [|t2 tl2 IHtl2].  
+ intros H. auto.4  
+ intros H. simpl. rewrite IHtl1; auto.5
```

Abstraction

Induction => Induction Principles



```
fun (T1 T2 : Type) (l1 : list T1) (l2 : list T2) =>  
  list_rect1 (fun (l1 : list T1) => ...)  
  (fun (l2 : list T2) _ => eq_refl)2  
  (fun (t1 : T1) (tl1 : list T1) (IHtl1 : ...) (l2 : list T2) =>  
    list_rect3 (fun (l2 : list T2) => ...)  
    (fun (H : ...) => eq_sym H)4  
    (fun (t2 : T2) (tl2 : list T2) (IHtl2 : ...) =>  
      fun (H : ...) => eq_rect_r ... eq_refl (IHtl1 ...)5)  
    l23)
```

¹
l1.
l2.

Certainty

Traditional Automation (Part 2 of 5)

Kinds of Automation

Tactic languages

Reflection

Custom tactics

Custom proof modes

Proof procedures

Plugins

Proof repair

Hammers

Traditional Automation (Part 2 of 5)

Kinds of Automation

Tactic languages

Reflection

Custom tactics

Custom proof modes

Proof procedures

Plugins

Proof repair

Hammers

Traditional Automation (Part 2 of 5)



**This automation can do
basically anything, yet still
preserve correctness.**

Traditional Automation (Part 2 of 5)



De Bruijn Criterion

Traditional Automation (Part 2 of 5)



Checking the Proof

Producing the Proof

Traditional Automation (Part 2 of 5)



Checking the Proof

Producing the Proof

Traditional Automation (Part 2 of 5)



Checking the Proof

Search Procedures

Producing the Proof

Traditional Automation (Part 2 of 5)



Checking the Proof

Search Procedures

Domain-Specific Heuristics

Producing the Proof

Traditional Automation (Part 2 of 5)



Checking the Proof

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof

Traditional Automation (Part 2 of 5)



Checking the Proof

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof ChatGPT



Traditional Automation (Part 2 of 5)



Checking the Proof

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof ChatGPT

Traditional Automation (Part 2 of 5)



Checking the Proof

Small & Human-Readable Logic Checker

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof ChatGPT

Traditional Automation (Part 2 of 5)



Checking the Proof

Small Logical Kernel

Search Procedures

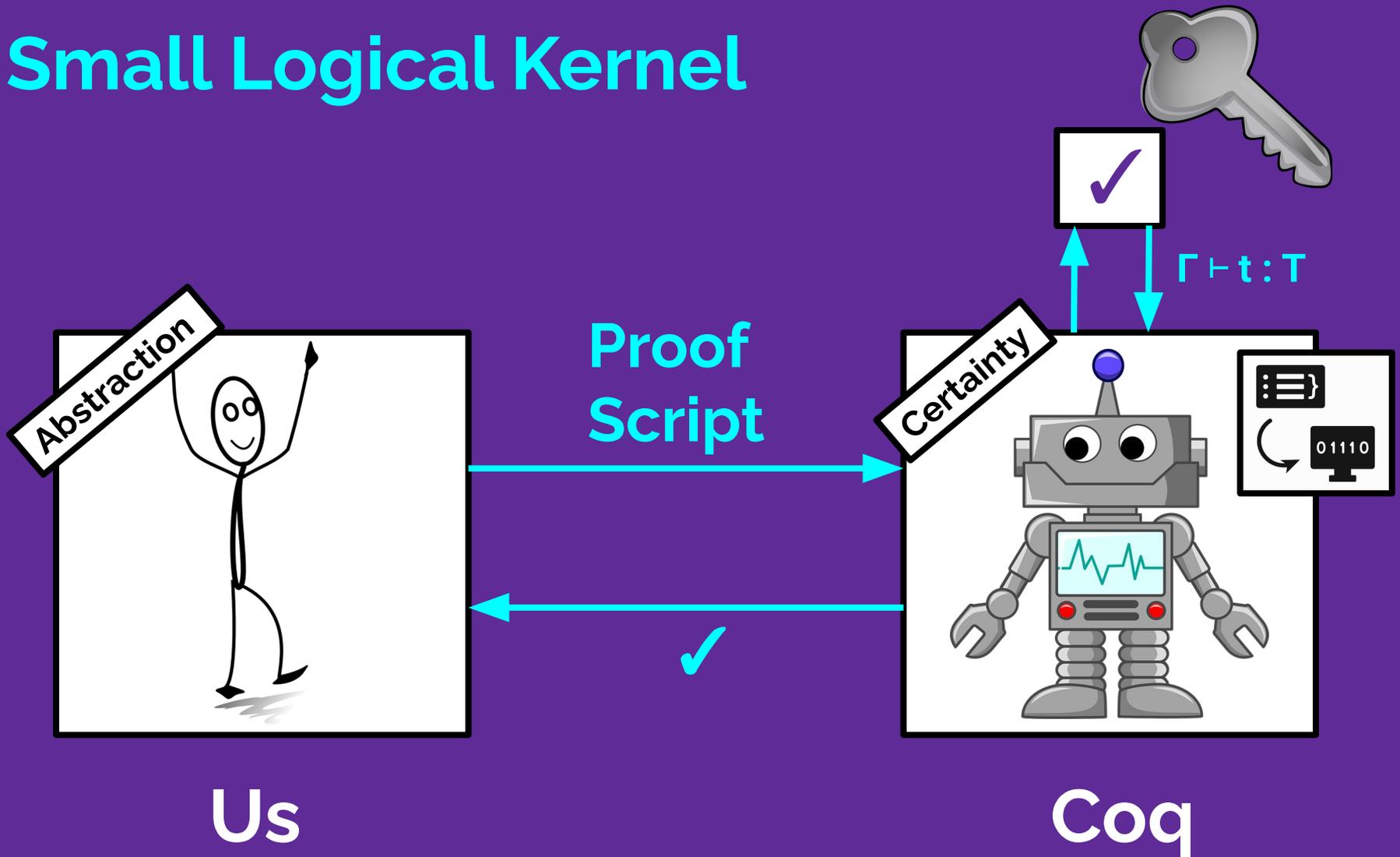
Domain-Specific Heuristics

Proof Transformations

Producing the Proof ChatGPT

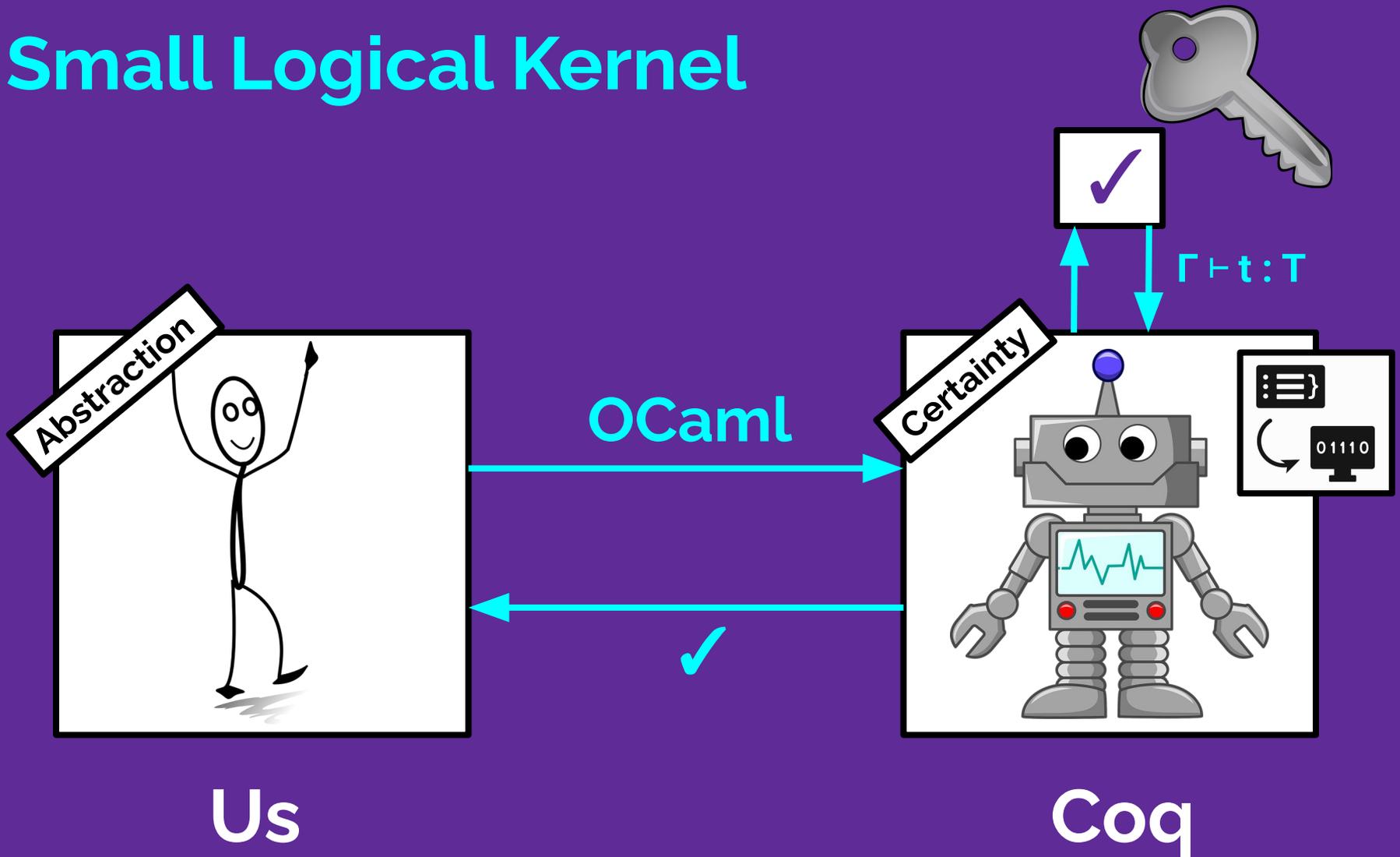
Traditional Automation (Part 2 of 5)

Small Logical Kernel



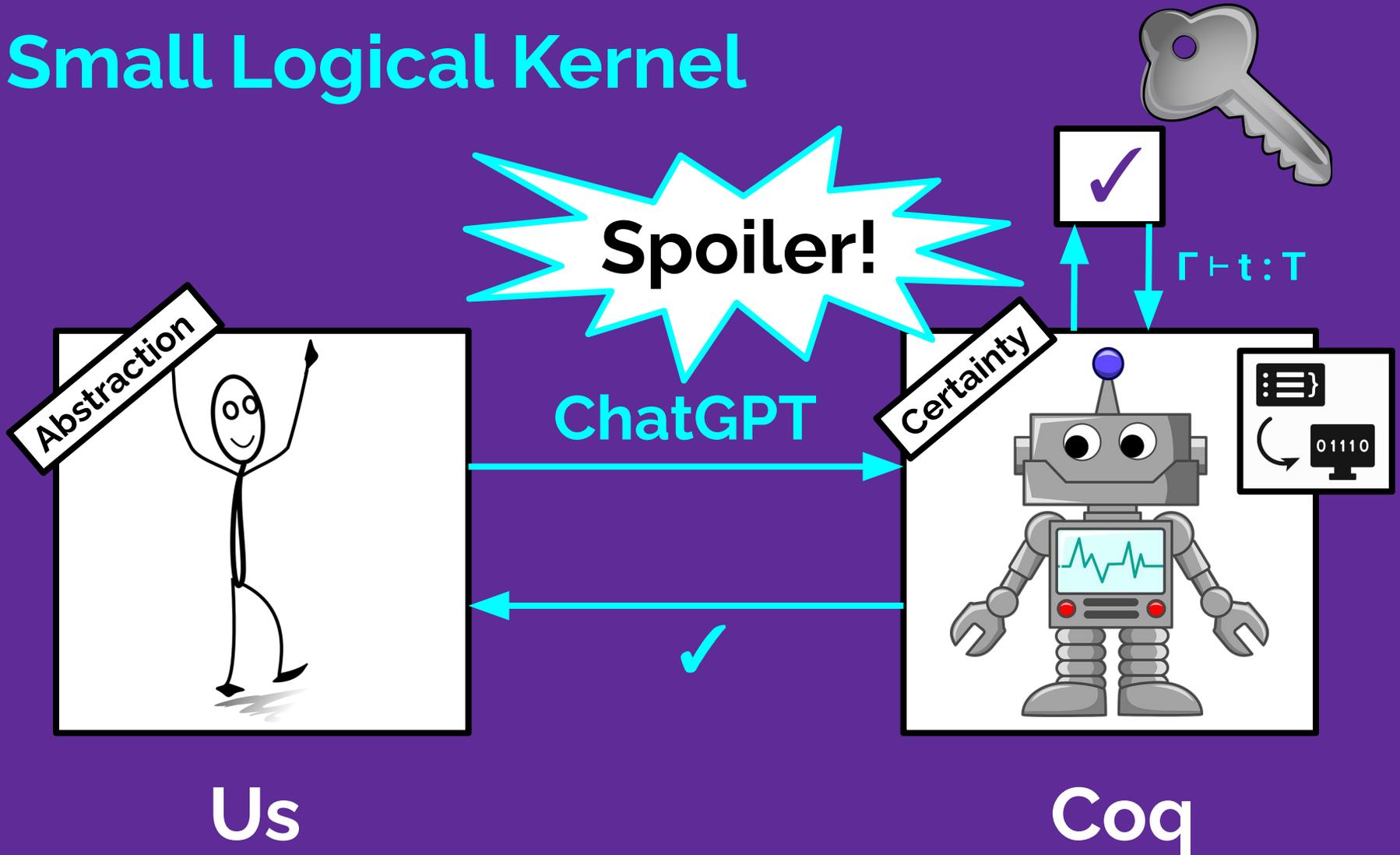
Traditional Automation (Part 2 of 5)

Small Logical Kernel



Traditional Automation (Part 2 of 5)

Small Logical Kernel



Traditional Automation (Part 2 of 5)



With **de Bruijn**, as long as you don't touch the **kernel**, your automation is **safe**.



With **de Bruijn**, as long as you don't touch the **kernel**, your automation is **safe**.*

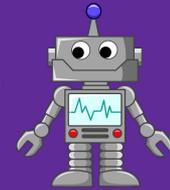
(If your specification is OK, your kernel has no bugs, and you don't introduce axioms)

Traditional Automation (Part 2 of 5)



With **de Bruijn**, as long as you don't touch the **kernel**, your automation is **safe**.*

The kernel and specification are the core **trusted** pieces, vetted by **humans**.



Traditional Automation (Part 2 of 5)

Traditional automation:

- + predictable
- + dependable
- + understandable
- limited in scope
- takes expertise to extend

Traditional proof repair:

- + predictable
- + dependable
- + understandable
- limited in scope
- takes expertise to extend

Proof Repair

Ornaments for Proof Reuse in Coq

Talia Ringer

University of Washington, USA
tringer@cs.washington.edu

Nathaniel Yazdani

University of Washington, USA
nyazdani@cs.washington.edu

John Leo

Halfaya Research, USA
leo@halfaya.org

Dan Grossman

University of Washington, USA
djg@cs.washington.edu

Abstract

Ornaments express relations between inductive types with the same inductive structure. We implement fully automatic proof reuse for a particular class of ornaments in Coq. We show how such a tool can give programmers the rewards of using indexed inductive types without paying away many of the costs. The plugin works directly on Coq code; it is the first tool to support proof reuse for a non-embedded dependently typed language. It is also the first tool to support proof reuse for ornaments: To lift a function or proof, the user must provide only the source code of the function, type, and the source function or proof. For ornaments, our approach produces proof reuse in Coq.

ITP 2019

PROOF REPAIR

Talia Ringer

Chair of the Supervisory Committee:
Dan Grossman
Computer Science & Engineering

The days of verifying only toy programs are long gone. The last two decades have marked a new era of verification at scale, with guarantees to large and critical systems—an era of proof engineering. Proof engineering is for verified systems what software engineering is for unverified systems. Still, while proof engineering—about both development and maintenance—is about both development and maintenance, engineering technologies so far have focused on development. It comes to *maintaining* these systems, proof engineering behind software engineering.

This thesis describes a proof repair tool for interactive theorem provers. It provides a way for engineers to interactively guide the machine-checked proof. When a proof about the system, traditional proof from scratch. Proof repair, automation: it determines how the system information to help fix the broken proof. Proof repair in this thesis works by algorithms with program transformation and the transformations operate on proofs called *proof terms*. Thanks to differencing and the transformation results in dependent type theory. For formalizes univalent transport from novel transformations over equality.

This approach is realized inside the Coq proof assistant. Case studies show use that this proof repair tool suite on real proof developments.

PhD Thesis

Adapting Proof Automation to Adapt Proofs

Talia Ringer
University of Washington, USA

John Leo
Halfaya Research, USA

Nathaniel Yazdani
University of Washington, USA

Dan Grossman
University of Washington, USA

Abstract

We extend proof automation in an interactive theorem prover to analyze *changes* in specifications and proofs. Our approach leverages the history of changes to specifications and proofs to search for a patch that can be applied to other specifications and proofs that need to change in analogous ways.

the search
This in turn
manually

Despite
ants is broken, even a small change to a definition can break many dependent proofs. This is a major

CPP 2018



Proof Repair across Type Equivalences

Talia Ringer
University of Washington
USA
tringer@cs.washington.edu

RanDair Porter
University of Washington
USA
randair@uw.edu

Nathaniel Yazdani
Northeastern University
USA
nyazdani.n@husky.neu.edu

John Leo
Halfaya Research
USA
leo@halfaya.org

Dan Grossman
University of Washington
USA
djg@cs.washington.edu

Abstract

We describe a new approach to automatically repairing broken proofs in the Coq proof assistant in response to changes in types. Our approach combines a configurable proof term transformation with a decompiler from proof terms to suggested tactic scripts. The proof term transformation implements transport across equivalences in a way that removes references to the old version of the changed type and does not rely on axioms beyond those Coq assumes.

We have implemented this approach in PUMPKIN Pi, an extension to the PUMPKIN PATCH Coq plugin suite for proof repair. We demonstrate PUMPKIN Pi's flexibility on eight case studies, including supporting a benchmark from a user study, easing development with dependent types, testing functions and proofs between unary and binary functions, and supporting an industrial proof between Coq and other verification languages.

PLDI 2021

1 Introduction

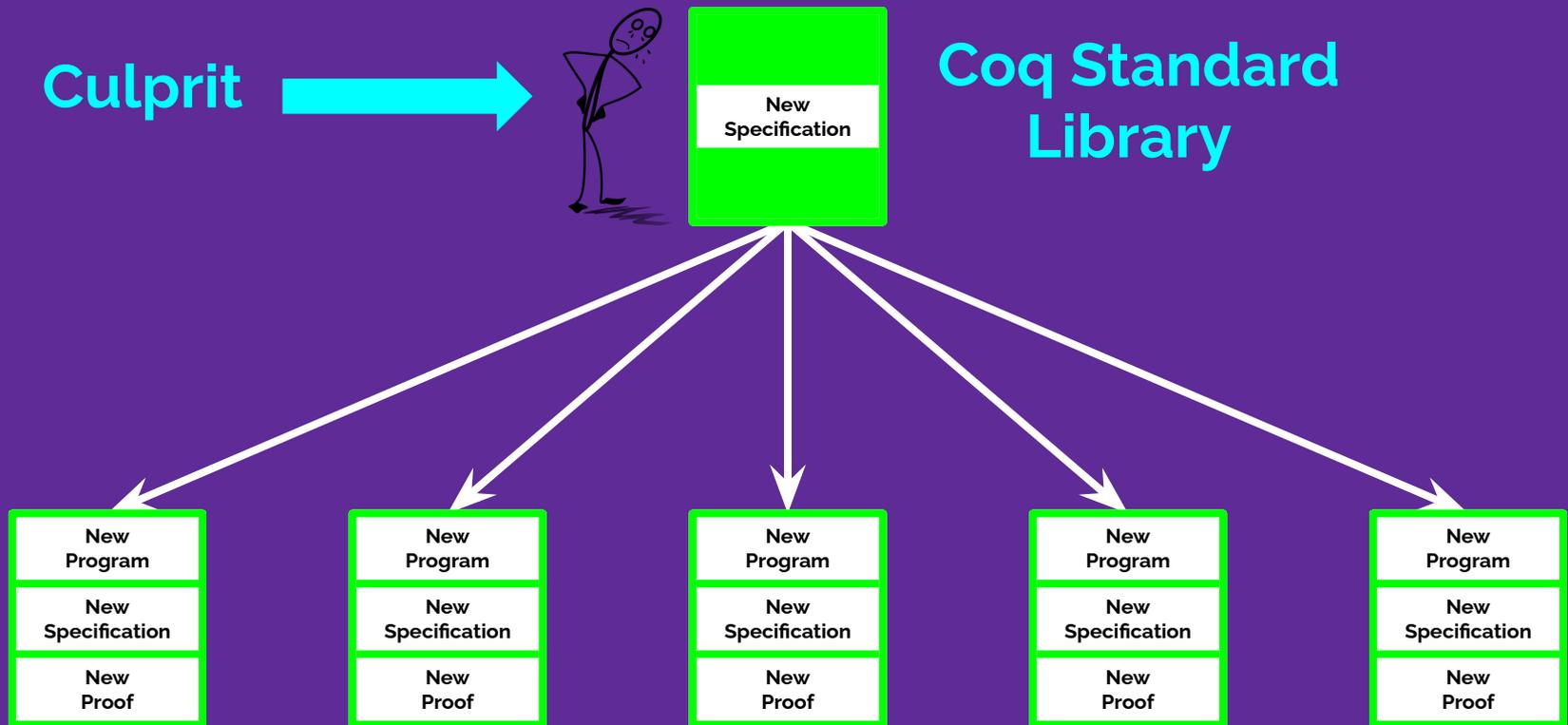
Program verification with interactive theorem provers has come a long way since its inception, especially when it comes to the scale of programs that can be verified. The seL4 [21] verified operating system kernel, for example, is the effort of a team of proof engineers spanning more than a million lines of proof, costing over 20 person-years. Given a famous 1977 critique of verification [12] (emphasis ours):

A sufficiently fanatical researcher might be willing to devote two or three years to verifying a significant piece of software if he could be assured that the software would remain stable.

we could argue that, over 40 years, either verification has become more fanatical, or all has changed (emphasis still ours): programs need to be maintained and modified. There is no reason to believe that

Traditional Automation (Part 2 of 5)

Proof Repair



Traditional Automation (Part 2 of 5)

Proof Repair

You have **changed** a **datatype**, and now the **standard library is broken!**

Traditional Automation (Part 2 of 5)

Proof Repair

451 functions & proofs,
25 seconds



You have **changed** a
datatype, and now the
standard library is broken!

Traditional Automation (Part 2 of 5)

Proof Repair



```
list <T> :=  
  | [] : list <T>  
  | cons : T → list <T> → list <T>
```

(* Repair all 451 functions & proofs: *)
Repair Module Old.list New.list in StdLib.

Proof Repair



```
list <T> :=  
  | cons : T → list <T> → list <T>  
  | [] : list <T>
```

(* Repair all 451 functions & proofs: *)

Repair Module Old.list **New.list** in StdLib.

Proof Repair

```
list <T> :=  
  | cons : T → list <T> → list <T>  
  | [] : list <T>
```

(* Repair all 451 functions & proofs: *)
Repair Module Old.list **New.list** in StdLib.



Traditional proof repair:

- + predictable
- + dependable
- + understandable
- limited in scope
- takes expertise to extend

Proof Repair – Predictable

PUMPKIN Pi supports
any **change** described
by a **type equivalence**.

The Univalent Foundations Program. 2013. **Homotopy
Type Theory: Univalent Foundations of Mathematics**.
Institute for Advanced Study.

Traditional Automation (Part 2 of 5)

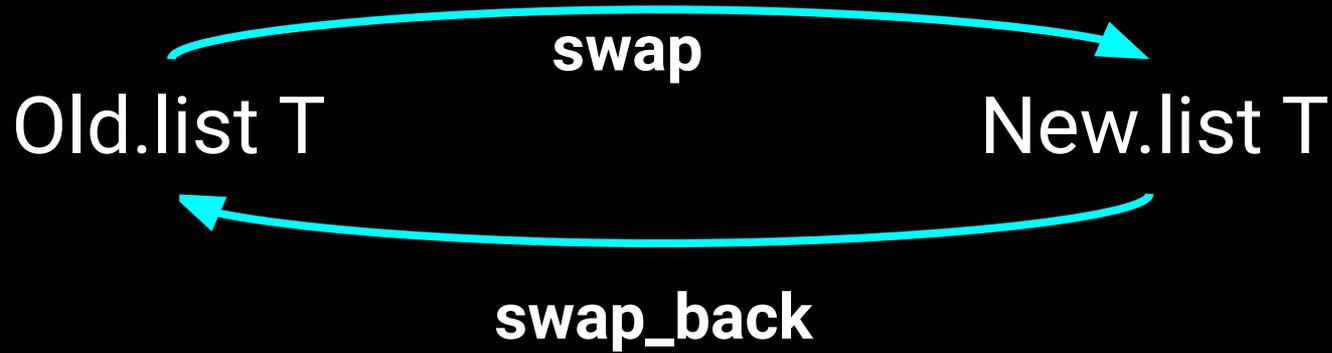
Proof Repair – Predictable

PUMPKIN Pi supports
any **change** described
by a **type equivalence**.

The Univalent Foundations Program. 2013. **Homotopy Type Theory: Univalent Foundations of Mathematics**.
Institute for Advanced Study.

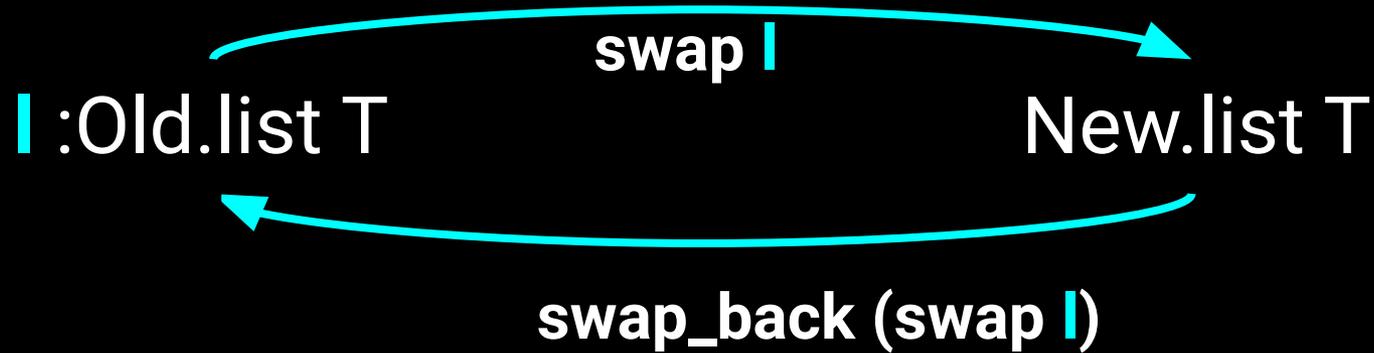
Traditional Automation (Part 2 of 5)

Equivalences



Traditional Automation (Part 2 of 5)

Equivalences



Traditional Automation (Part 2 of 5)

Equivalences

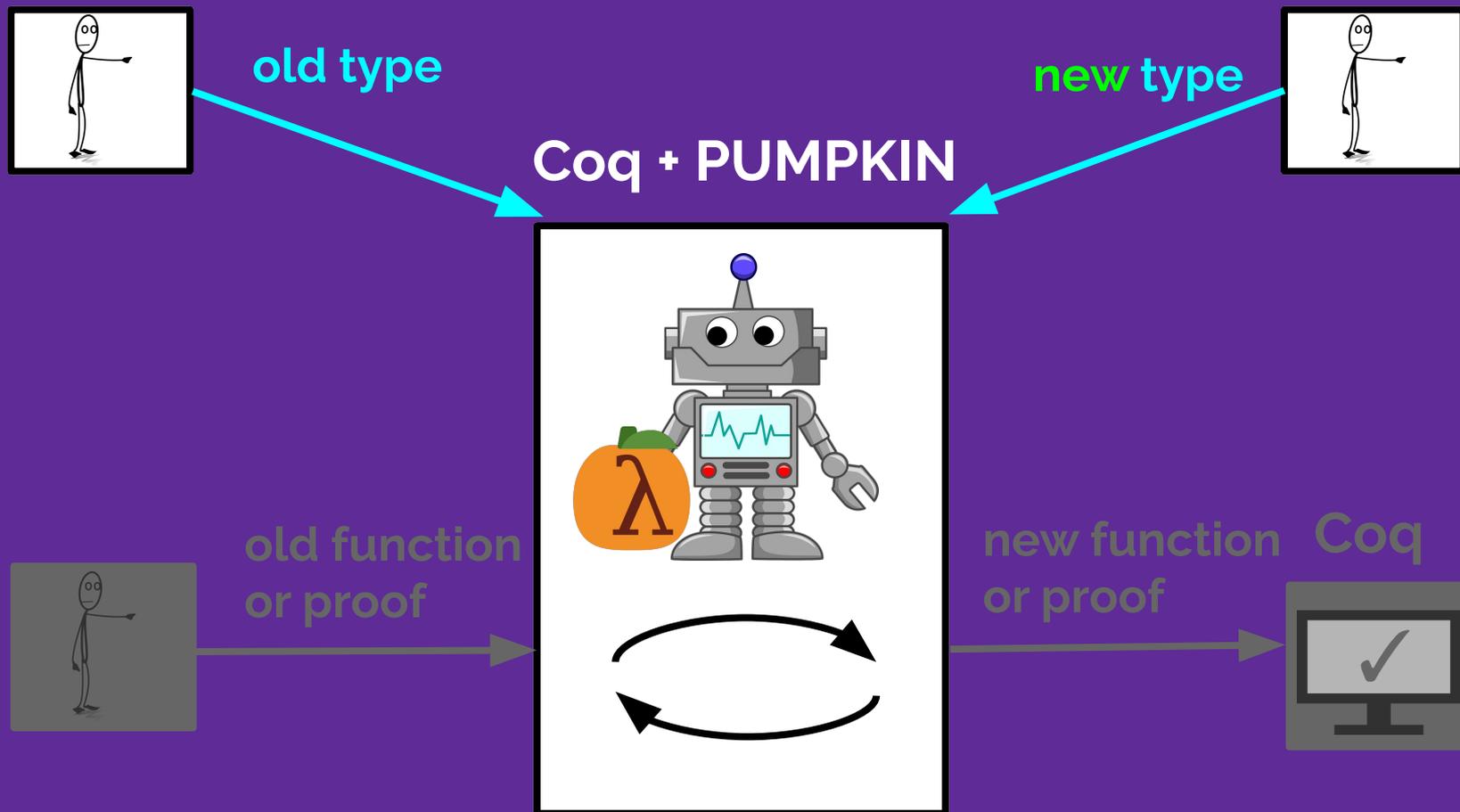
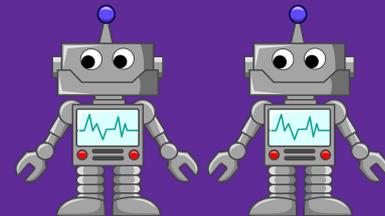


Equivalences



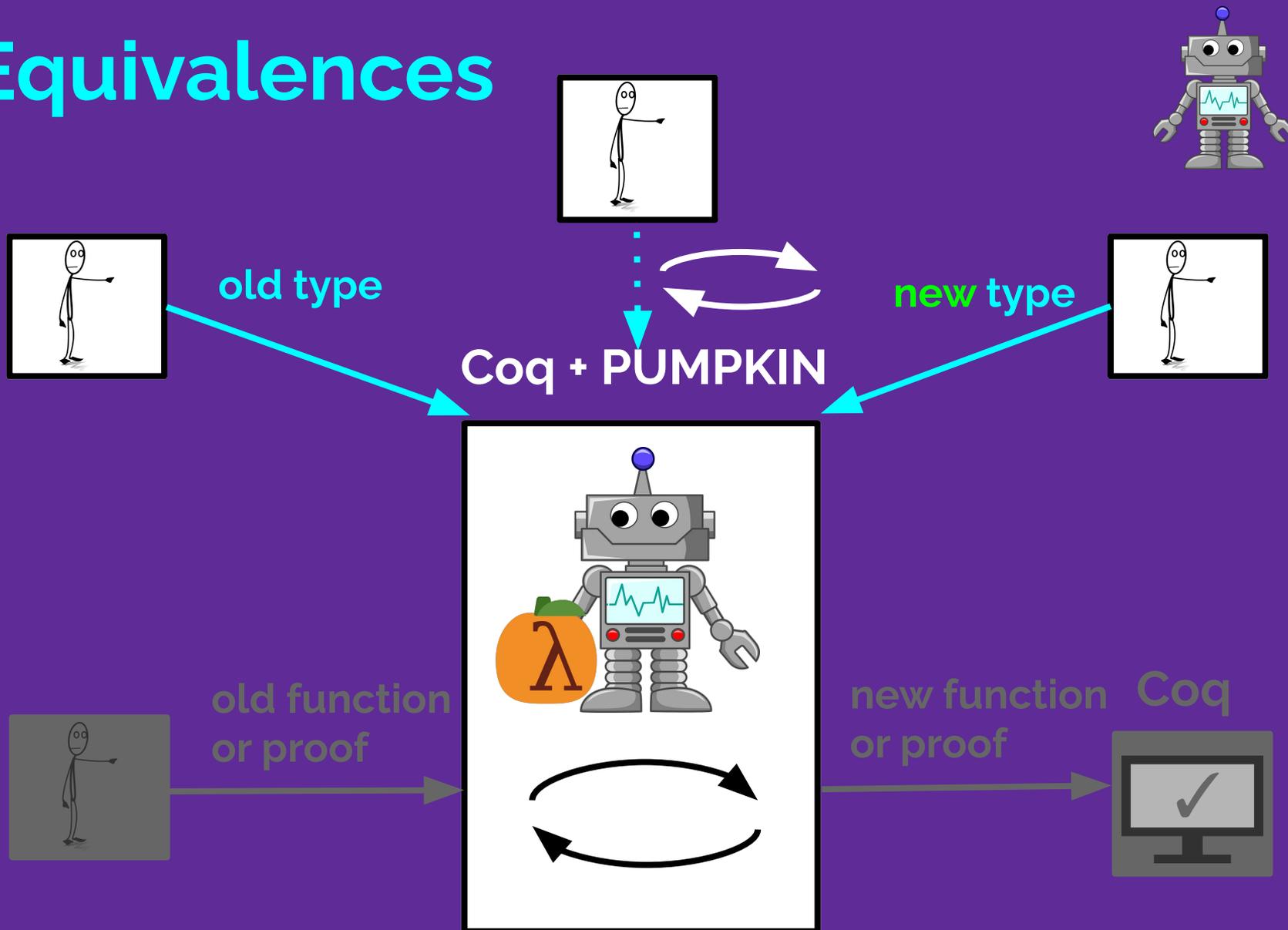
Traditional Automation (Part 2 of 5)

Equivalences



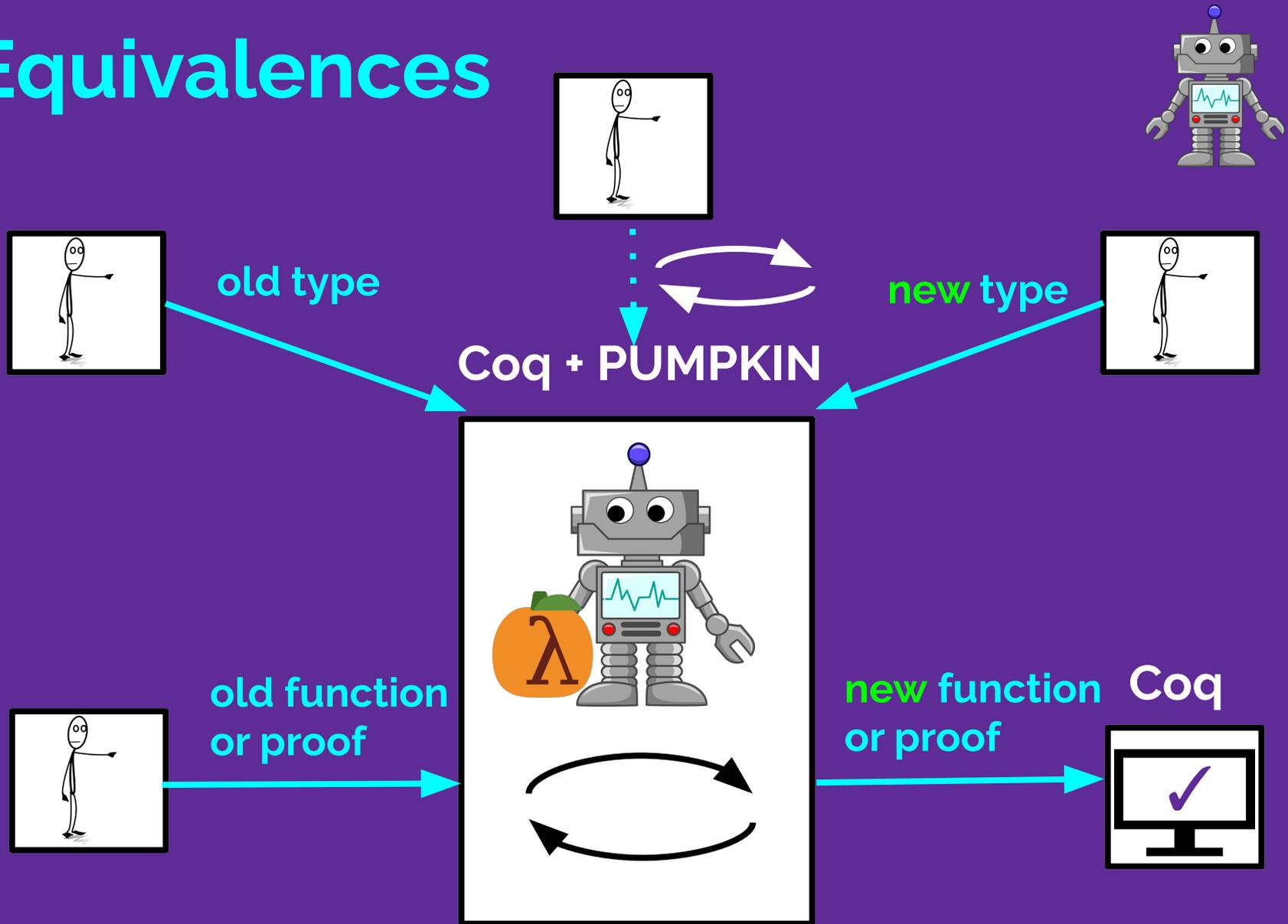
Traditional Automation (Part 2 of 5)

Equivalences



Traditional Automation (Part 2 of 5)

Equivalences



Traditional Automation (Part 2 of 5)

Proof Repair – Dependable

**PUMPKIN Pi is
flexible & useful
for real scenarios.**

Traditional Automation (Part 2 of 5)

Proof Repair – Dependable

Equivalences

are even more **expressive**
than they may sound.

Traditional Automation (Part 2 of 5)

Proof Repair – Dependable

Adding New Information

Traditional Automation (Part 2 of 5)

Traditional proof repair:

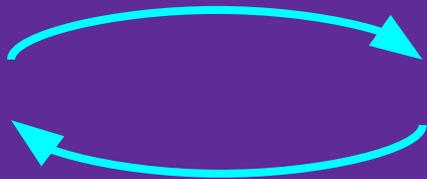
- + predictable
- + dependable
- + understandable
- limited in scope
- takes expertise to extend

Traditional proof repair:

- + predictable
- + dependable
- + understandable* (for type nerds)
- limited in scope
- takes expertise to extend

Traditional Automation (Part 2 of 5)

Proof Repair – Understandable



Transport: Rewriting across Equivalences

The Univalent Foundations Program. 2013. **Homotopy Type Theory: Univalent Foundations of Mathematics.**
Institute for Advanced Study.

Traditional Automation (Part 2 of 5)

Proof Repair – Understandable

Transport as a Proof Term Transformation

Traditional Automation (Part 2 of 5)

Proof Repair – Understandable

For type nerds:

**Deconstruct Equivalence
(Lambek's Theorem)**

Traditional Automation (Part 2 of 5)

Traditional proof repair:

- + predictable
- + dependable
- + understandable* (for type nerds)
- limited in scope
- takes expertise to extend

Proof Repair – Limited Scope

Proof Repair across Quotient Type Equivalences

Internal and External Views

COSMO VIOLA, University of Illinois Urbana-Champaign, USA

MAX FAN, University of Illinois Urbana-Champaign, USA

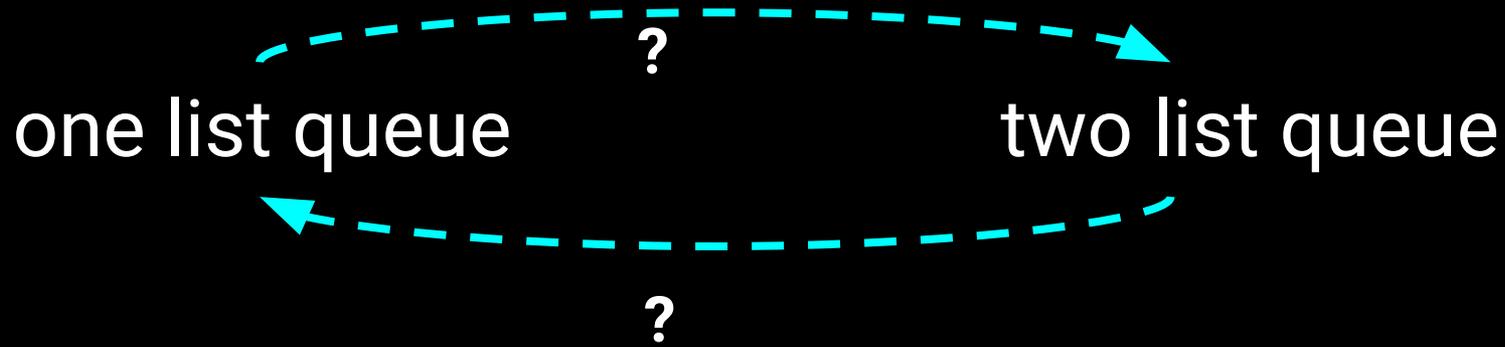
TALIA RINGER, University of Illinois Urbana-Champaign, USA

Proofs in proof assistants like Coq can be brittle, breaking easily in response to changes in the terms and types those proofs depend on. To address this, recent work introduced an algorithm and tool in Coq to automatically repair broken proofs in response to changes that correspond to type equivalences. However, many changes remained out of the scope of this algorithm and tool—especially changes in underlying *behavior*. We extend this proof repair algorithm so that it can express certain changes in behavior that were previously out of scope. We focus in particular on equivalences between *quotient types*—types equipped with a relation that describes what it means for any two elements of that type to be equal. Quotient type equivalences can be used to express interesting changes in representations of mathematical structures, as well as changes in the underlying implementations of data structures—two use cases highlighted by our case studies.

We extend this algorithm to support quotient type equivalences in two different ways: (1) internally to cubical type theory (applied to Cubical Agda), and (2) externally to CIC_ω (applied to Coq). While our approach in Coq comes equipped with prototype automation, it suffers notably from Coq's lack of quotient types—something we circumvent using Coq's setoid machinery and an extension to the proof repair algorithm to support the corresponding new proof obligations. In contrast, while our approach in Cubical Agda is completely manual, it takes advantage of cubical type theory's internal quotient types, which makes the algorithm straightforward. Furthermore, it includes the first internal proof of correctness of repaired proofs, something not possible in general. **Under Submission** We compare these two approaches, and demonstrate these tradeoffs on proof repair case studies for previously unsupported changes.

Traditional Automation (Part 2 of 5)

Quotient Type Equivalences



Traditional Automation (Part 2 of 5)

Traditional proof repair:

- + predictable
- + dependable
- + understandable* (for type nerds)
- limited in scope
- takes expertise to extend

Proof Repair – Hard to Extend

One PhD student,
one undergraduate,
one advisor,
2.5 years.

Is this sustainable?

Traditional Automation (Part 2 of 5)

Proof Repair – Hard to Extend

One PhD student,
one undergraduate,
one advisor,
2.5 years.

Is this sustainable?

Traditional Automation (Part 2 of 5)

1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. Opportunities

Language models:

- *unpredictable*
- *not* dependable
- *not* understandable
- + *not very* limited in scope
- + takes *little* expertise to extend

Big Interest

Proofster: Automated Formal Verification

Arpan Agrawal
University of Illinois
Urbana-Champaign, IL, USA
arpan2@illinois.edu

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Zhanna Kaufman
University of Massachusetts
Amherst, MA, USA
zhanna.kaufma@cs.umass.edu

Tom Reichel
University of Illinois
Urbana-Champaign, IL, USA
reichel3@illinois.edu

Shizhuo Zhang
University of Illinois
Urbana-Champaign, IL, USA
shizhuo2@illinois.edu

Timothy Zhou
University of Illinois
Urbana-Champaign, IL, USA
tz2@illinois.edu

Alex Sanchez-Stern
University of Massachusetts
Amherst, MA, USA
sanchezsstern@cs.umass.edu

Talia Ringer
University of Illinois
Urbana-Champaign, IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

effective but extremely
ware quality. Verifying
n requires significantly
the first place, despite
Coq, aiding the process.
the synthesis of formal
exists for practitioners.
d tool aimed at assisting
cess via proof synthesis.
fying a property of a
ally synthesize a formal
When it is possible to

Meanwhile, it took 11 person-years to write the proofs required to verify the seL4 microkernel [17], which represents a tiny fraction of the functionality of a full kernel.
Recent work has aimed to simplify the process of writing proofs [2], [6], [7], [9], [10], [14], [11], [23], [24], [30]. Some formal verification can even be fully automated via proof synthesis. For example, CoqHammer [4] uses a set of precomputed mathematical facts to attempt to “hammer” out a proof. Meanwhile, ASTactic [30], Proverbot9001 [23], TacTok [17], Diva [6], and Passport [24] learn a predictive model to guide

Passport: Improving Automated Formal Verification Using Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

Formally verifying system properties is one of the most effective ways of improving system quality, but its high manual effort requirements often render it prohibitively expensive. Tools that automate formal verification, by learning from proof corpora to suggest proofs, have just begun to show their promise. These tools are effective because of the richness of the data the proof corpora contain. This richness comes from the stylistic conventions followed by communities of proof developers, together with the powerful logical systems beneath proof assistants. However, this richness remains underexploited, with most work thus far focusing on architecture rather than on how to make the most of the proof data.

In this paper, we develop Passport, a fully-automated proof-synthesis tool that systematically explores how to most effectively exploit one aspect of that proof data: identifiers. Passport enriches a predictive Coq model used by proof-synthesis tools with three new encoding mechanisms for identifiers: category vocabulary

**TOPLAS Vol. 45, Issue 2:
No. 12, pp 1-30, 2023**

ormation
erall, our
eading to

model to guide
om scratch.
sts for practi-
or example, of
the above-mentioned search-based tools, all but one have neither

- 1 Tom Reichel ✉
- 2 University of Illinois Urbana-Champaign, USA
- 3 R. Wesley Henderson ✉
- 4 Radiance Technologies, Inc., Huntsville, AL, USA
- 5 Andrew Touchet ✉
- 6 Radiance Technologies, Inc., Huntsville, AL, USA
- 7 Andrew Gardner* ✉
- 8 Radiance Technologies, Inc., Huntsville, AL, USA
- 9 Talia Ringer* ✉
- 10 University of Illinois Urbana-Champaign, USA

Abstract

We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide r
Our hope is to make it eas
for proofs will move to tarj

ITP 2023

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Markus N. Rabe
Google, Inc.
CA, USA
mrabe@google.com

Talia Ringer
University of Illinois Urbana-Champaign
IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT

Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle-HOL, e.g. by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation model with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that (1) Whole proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the ensuing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle-HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 33.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

As a result, recent research has focused on automated proof synthesis, which can lead to fully automating formal verification. There are two promising approaches for automating proof synthesis. The first is to use *hammers*, such as Sledgehammer [54] for the Isabelle proof assistant. Hammers iteratively apply known mathematical facts using heuristics. The second is to use search-based *neural theorem provers*, such as DeepFOLK [4], GPT-F [60], TacticZero [91], Lisa [54], EvolveIt [42], Diva [60], TacTok [72], and ASTactic [16]. Given a partial proof and the current *proof state* (which consists of the current goal to prove and the list of known assumptions), these tools use neural networks to predict the next *individual proof step*. They use the *proof assistant* to evaluate the proposed next proof steps, which returns a new set of proof states. Neural theorem provers rely on diverse neural architectures, such as Wavenet [4, 84], graph neural networks [62], short long-term memory models [20], and language models with the transformer architecture [27, 66].

In this paper, we propose Baldur, a different, simpler approach to proof synthesis. We show that using large language models (LLMs), fine-tuned on proofs, can produce entire proofs for theorems. LLMs are scaled-up transformer models trained on a large amount of text data, including natural language and code, that have proven to be remarkably effective across a wide variety of applications, including question answering, and text and code generation [7, 14]. Here, we show their remarkable effectiveness for whole proof generation.

The main contributions of our work are:

- We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammers or conventional, computationally expensive search.
- We repair tasks and demonstrate that repaired proofs with LLMs further improve proving power when the LLM proof assistant’s error messages originate on a large benchmark that

ESEC/FSE 2023 Distinguished Paper

DAVID E. LONG, UNIVERSITY OF CALIFORNIA, SAN DIEGO, SAN DIEGO, CA, USA
EMILY FIRST, UNIVERSITY OF MASSACHUSETTS, AMHERST, MA, USA
TALIA RINGER, UNIVERSITY OF ILLINOIS, URBANA-CHAMPAIGN, IL, USA
YURIY BRUN, UNIVERSITY OF MASSACHUSETTS, AMHERST, MA, USA

ESEC/FSE 2023 Distinguished Paper

Supervised Models: Building a Large Proof Repair Dataset

- 1 Tom Reichel ✉
- 2 University of Illinois Urbana-Champaign, USA
- 3 R. Wesley Henderson ✉
- 4 Radiance Technologies, Inc., Huntsville, AL, USA
- 5 Andrew Touchet ✉
- 6 Radiance Technologies, Inc., Huntsville, AL, USA
- 7 Andrew Gardner* ✉
- 8 Radiance Technologies, Inc., Huntsville, AL, USA
- 9 Talia Ringer* ✉
- 10 University of Illinois Urbana-Champaign, USA

Abstract

We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide r
Our hope is to make it eas
for proofs will move to tarj

ITP 2023

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

LM-Based Automation (Part 3 of 5)

04.10370v2 [cs.PL] 2 Aug 2022

First Project: Passport

04.10370v2 [cs.PL] 2 Aug 2022

Passport: Improving Automated Formal Verification Using Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

Formally verifying system properties is one of the most effective ways of improving system quality, but its high manual effort requirements often render it prohibitively expensive. Tools that automate formal verification, by learning from proof corpora to suggest proofs, have just begun to show their promise. These tools are effective because of the richness of the data the proof corpora contain. This richness comes from the stylistic conventions followed by communities of proof developers, together with the powerful logical systems beneath proof assistants. However, this richness remains underexploited, with most work thus far focusing on architecture rather than on how to make the most of the proof data.

In this paper, we develop Passport, a fully-automated proof-synthesis tool that systematically explores how to most effectively exploit one aspect of that proof data: identifiers. Passport enriches a predictive Coq model used by proof-synthesis tools with three new encoding mechanisms for identifiers: category vocabulary

TOPLAS Vol. 45, Issue 2: No. 12, pp 1-30, 2023

Proofster: Automated Formal Verification

Arpan Agrawal
University of Illinois
Urbana-Champaign, IL, USA
arpan2@illinois.edu

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Zhanna Kaufman
University of Massachusetts
Amherst, MA, USA
zhannakaufma@cs.umass.edu

Tom Reichel
University of Illinois
Urbana-Champaign, IL, USA
reichel3@illinois.edu

Shizhuo Zhang
University of Illinois
Urbana-Champaign, IL, USA
szhang2@illinois.edu

Timothy Zhou
University of Illinois
Urbana-Champaign, IL, USA
tzhou2@illinois.edu

Alex Sanchez-Stern
University of Massachusetts
Amherst, MA, USA
sanchezsstern@cs.umass.edu

Talia Ringer
University of Illinois
Urbana-Champaign, IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

active but extremely
are quality. Verifying
requires significantly
be first place, despite
og, aiding the process.
he synthesis of formal
ists for practitioners.
tool aimed at assisting
ess via proof synthesis
ing a property of a
ly synthesizes a formal
When it is unable to
of a search tree
alder of proof
nt. Proofster
e video demonstrating
QA166RfWl.

Meanwhile, it took 11 person-years to write the proofs required to verify the seL4 microkernel [17], which represents a tiny fraction of the functionality of a full kernel.
Recent work has aimed to simplify the process of writing proofs [2], [6], [7], [9], [10], [14], [11], [23], [24], [30]. Some formal verification can even be fully automated via proof synthesis. For example, CoqHammer [4] uses a set of precomputed mathematical facts to attempt to “hammer” out a proof. Meanwhile, AStactic [30], Proverbot0001 [23], TacTok [7], Diva [6], and Passport [24] learn a predictive model from a corpus of existing proofs and use that model to guide their search for proofs. These tools support lists for practitioners to use these Coq proof-synthesis tools. For example, of the above-mentioned search-based tools, all but one have neither

- 1 Tom Reichel ✉
- 2 University of Illinois Urbana-Champaign, USA
- 3 R. Wesley Henderson ✉
- 4 Radiance Technologies, Inc., Huntsville, AL, USA
- 5 Andrew Touchet ✉
- 6 Radiance Technologies, Inc., Huntsville, AL, USA
- 7 Andrew Gardner* ✉
- 8 Radiance Technologies, Inc., Huntsville, AL, USA
- 9 Talia Ringer* ✉
- 10 University of Illinois Urbana-Champaign, USA

13 Abstract

14 We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide recommendations for how the proof assistant community can address them. Our hope is to make it easier to build datasets and infrastructure so that machine-learning tools for proofs will move to target the tasks that machine-learning tools can address most effectively across proof assistants.

2023

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Markus N. Rabe
Google, Inc.
CA, USA
mrabe@google.com

Talia Ringer
University of Illinois Urbana-Champaign
IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT
Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle-HOL, e.g. by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation method with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that (1) Whole-proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the ensuing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle-HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 34.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

As a result, recent research has focused on automated proof synthesis, which can lead to fully automating formal verification. There are two promising approaches for automating proof synthesis. The first is to use hammers, such as Sledgehammer [54] for the Isabelle proof assistant. Hammers iteratively apply known mathematical facts using heuristics. The second is to use search-based neural theorem provers, such as DeepFOL [4], GPT4 [50], TacticZero [91], Lisa [54], Evariste [62], Diva [20], TacTok [22], and AStactic [30]. Given a partial proof and the current proof state (which consists of the current goal to prove and the list of known assumptions), these tools use neural networks to predict the next individual proof step. They use the proof assistant to evaluate the proposed next proof steps, which returns a new set of proof states. Neural theorem provers rely on diverse neural architectures, such as Wavenet [4, 84], graph neural networks [62], short long-term memory models [20], and language models with the transformer architecture [27, 66].

In this paper, we propose Baldur, a different, simpler approach to proof synthesis. We show that using large language models (LLMs), fine-tuned on proofs, can produce entire proofs for theorems. LLMs are scaled-up transformer models trained on a large amount of text data, including natural language and code, that have proven to be remarkably effective across a wide variety of applications, including question answering, and text and code generation [7, 14]. Here, we show their remarkable effectiveness for whole proof generation.

The main contributions of our work are:

- We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammers or computationally expensive search.
- We demonstrate that Baldur can generate proofs with LLMs for theorems that existing tools cannot prove when the LLM is used as a proof assistant’s error messages.
- We demonstrate empirically on a large benchmark that Baldur, when combined with prior techniques, significantly improves the state-of-the-art for theorem proving in Isabelle-HOL. We compare Baldur to Thor, the state-of-the-art tool, but we evaluate our implementation using 62 versions of the benchmark. By contrast, existing tools that use LLMs for theorem

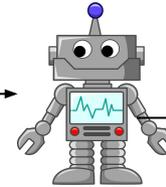
ESEC/FSE 2023 Distinguished Paper

LM-Based Automation (Part 3 of 5)

First Project: Passport

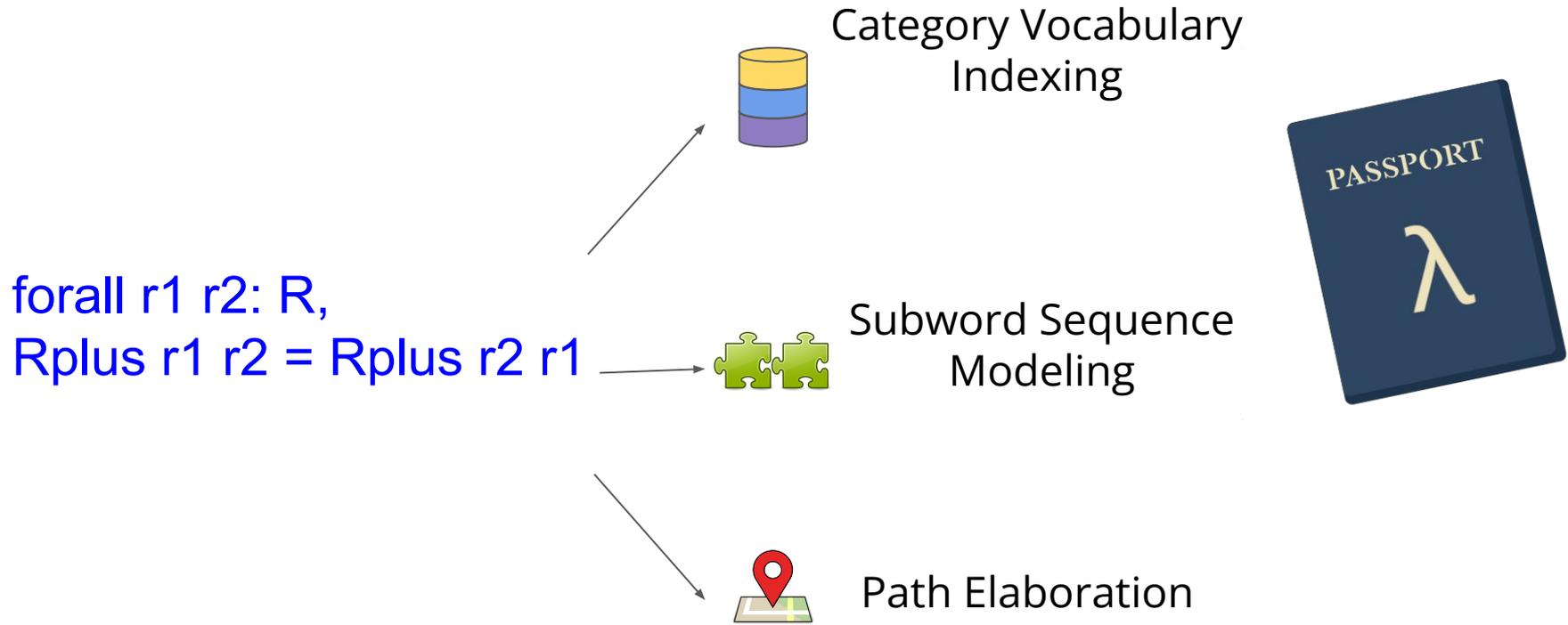
Addition of real numbers is commutative

forall $r1\ r2: R,$
 $Rplus\ r1\ r2 = Rplus\ r2\ r1$

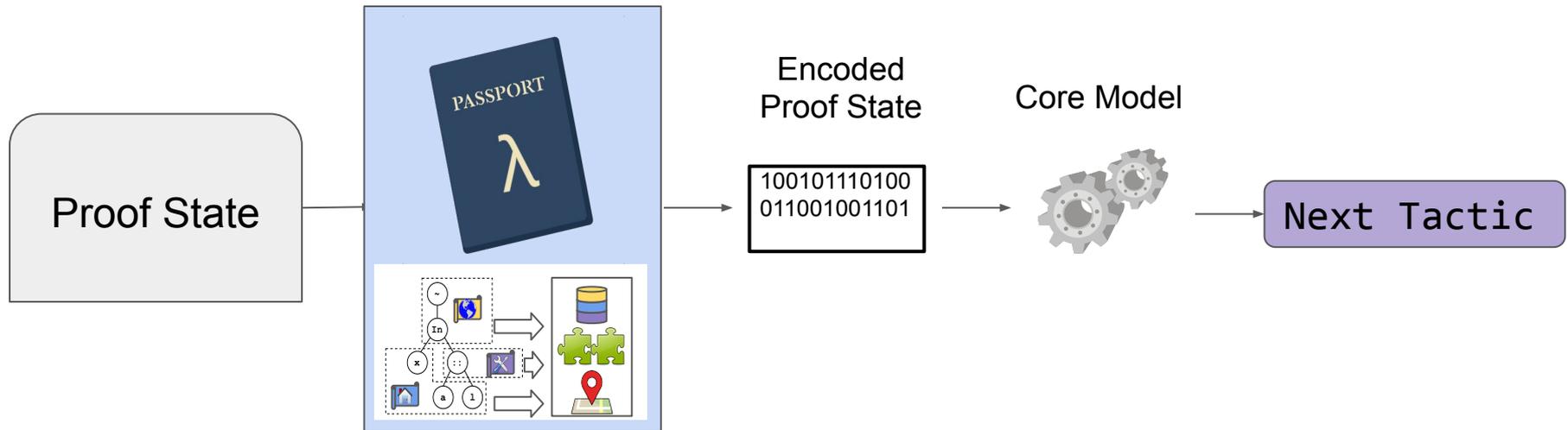


Next Tactic

First Project: Passport



First Project: Passport



First Project: Passport

Language models:

- *unpredictable*
- *not* dependable
- *not* understandable
- + *not very* limited in scope
- + takes *little* expertise to extend

First Project: Passport – Big Scope

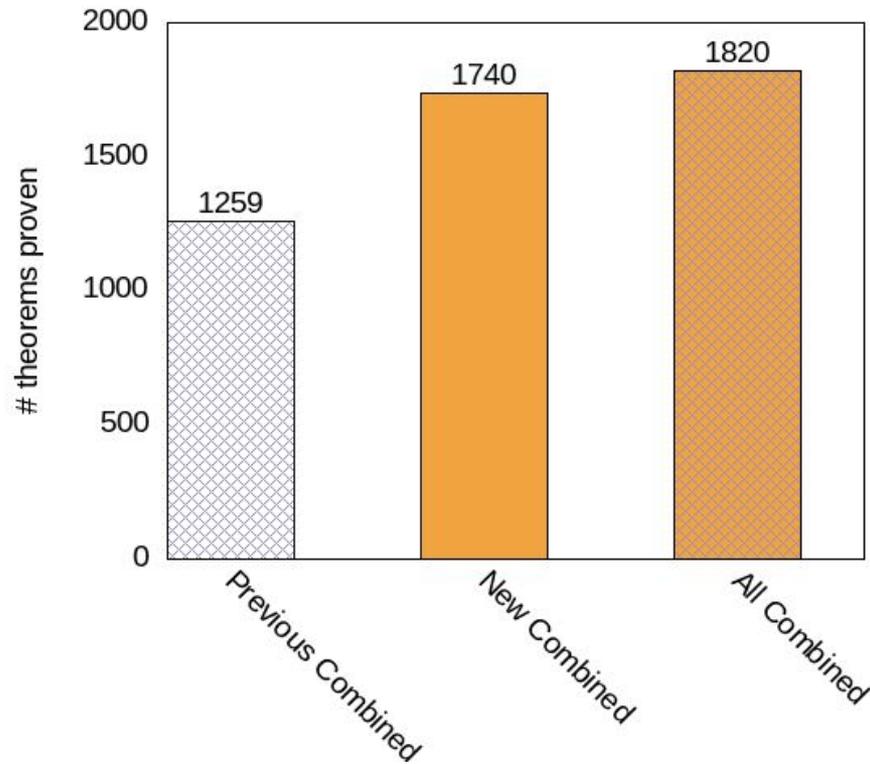
- Yang and Deng 2019
- **Mathematical formalizations, proven correct programs, and Coq automation libraries**
- 123 open-source Coq projects
- Trained on **97 projects (57,719 theorems)**
- Tested on **26 projects (10,782 theorems)**

CoqGym



First Project: Passport – Big Scope

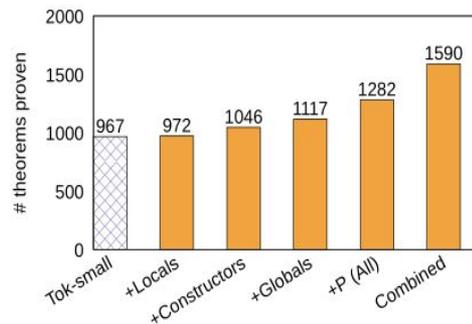
We can prove **45% more** theorems than before!



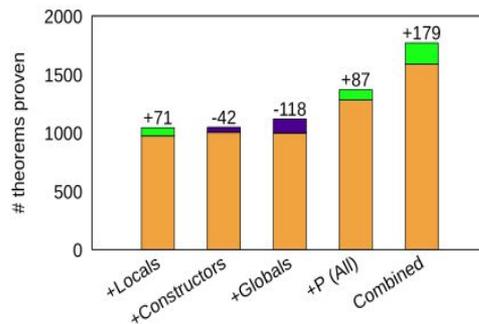
First Project: Passport – Big Scope

Diversity brings even higher returns!

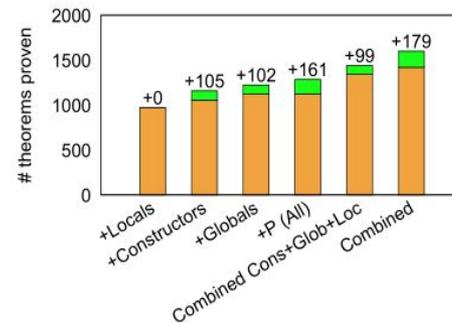
64% more theorems than the baseline!



(a) The impact of category vocabulary indexing on three identifier categories (without subwords or paths): local variables, type constructors, and global definitions.



(b) The impact of subword encoding on each of the categories of identifiers (with category vocabulary indexing but without paths).



(c) The impact of fully-qualified path encoding of type constructors and global definitions (with category vocabulary indexing but without subwords).

First Project: Passport – Easy to Extend

- Some **easy Python scripts** on top of someone else's **existing project**
- **Parallelized work** for different extensions between me and five other authors
- **Undergraduate** implemented most challenging extension in an order of **weeks**
- Scripts were **simple and fun** enough that I got excited when writing one in between drafting thesis chapters, ran into a couch, and broke my big toe

First Project: Passport

Language models:

- *unpredictable*
- *not* dependable
- *not* understandable
- + *not very* limited in scope
- + takes *little* expertise to extend

First Project: Passport – Confusion

- Somehow, the ***name*** of the user running the training script impacted the **file order**, which impacted the **results** of training a model on **identical data** in an **identical way**
- We found a **nondeterminism bug in Pytorch**
- **Some combinations** of extensions worked **mysteriously poorly**, even though all together they helped
- Apparently **this is just life** with even small **LMs**? Is this life now? **Help?**

More in the Paper!

04.10370v2 [cs.PL] 2 Aug 2022

Passport: Improving Automated Formal Verification Using Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
 EMILY FIRST*, University of Massachusetts Amherst, USA
 TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
 ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
 YURIY BRUN, University of Massachusetts Amherst, USA
 TALIA RINGER, University of Illinois Urbana-Champaign, USA

Formally verifying system properties is one of the most effective ways of improving system quality, but its high manual effort requirements often render it prohibitively expensive. Tools that automate formal verification, by learning from proof corpora to suggest proofs, have just begun to show their promise. These tools are effective because of the richness of the data the proof corpora contain. This richness comes from the stylistic conventions followed by communities of proof developers, together with the powerful logical systems beneath proof assistants. However, this richness remains underexploited, with most work thus far focusing on architecture rather than on how to make the most of the proof data.

In this paper, we develop Passport, a fully-automated proof-synthesis tool that systematically explores how to most effectively exploit one aspect of that proof data: identifiers. Passport enriches a predictive Coq model used by proof-synthesis tools with three new encoding mechanisms for identifiers: category vocabulary

**TOPLAS Vol. 45, Issue 2:
No. 12, pp 1-30, 2023**

Information, email, or reading to

Proofster: Automated Formal Verification

Arpan Agrawal
University of Illinois Urbana-Champaign, IL, USA
arpan2@illinois.edu

Emily First
University of Massachusetts Amherst, MA, USA
efirst@cs.umass.edu

Zhanna Kaufman
University of Massachusetts Amherst, MA, USA
zhanakaufma@cs.umass.edu

Tom Reichel
University of Illinois Urbana-Champaign, IL, USA
reichel3@illinois.edu

Shizhuo Zhang
University of Illinois Urbana-Champaign, IL, USA
szhang2@illinois.edu

Timothy Zhou
University of Illinois Urbana-Champaign, IL, USA
tzhou2@illinois.edu

Alex Sanchez-Stern
University of Massachusetts Amherst, MA, USA
sanchezstern@cs.umass.edu

Talia Ringer
University of Illinois Urbana-Champaign, IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts Amherst, MA, USA
brun@cs.umass.edu

active but extremely are quality. Verifying requires significantly be first place, despite, og, aiding the process. he synthesis of formal sists for practitioners. tool aimed at assisting s via proof synthesis ing a property of a ly synthesizes a formal When it is unable to ndance search tree nterpreter. Proof stering. QAI66RfWf.

Meanwhile, it took 11 person-years to write the proofs required to verify the seL4 microkernel [17], which represents a tiny fraction of the functionality of a full kernel.

Recent work has aimed to simplify the process of writing proofs [2], [6], [7], [9], [10], [14], [11], [23], [24], [30]. Some formal verification can even be fully automated via proof synthesis. For example, CoqHammer [4] uses a set of precomputed mathematical facts to attempt to “hammer” out a proof. Meanwhile, AStactic [30], Proverbot9001 [23], TacTok [7], Diva [6], and Passport [24] learn a predictive model from a corpus of existing proofs and use that model to guide their search for new proofs. These tools support lists for practitioners to use these Coq proof-synthesis tools. For example, of the above-mentioned search-based tools, all but one have neither

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First
University of Massachusetts Amherst, MA, USA
efirst@cs.umass.edu

Markus N. Rabe
Google, Inc.
CA, USA
mrabe@google.com

Talia Ringer
University of Illinois Urbana-Champaign IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT

Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle-HOL, e.g., by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation model with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that (1) Whole-proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the ensuing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle-HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 34.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

In this paper, we propose Baldur, a different, simpler approach to proof synthesis. We show that using large language models (LLMs), fine-tuned on proofs, can produce entire proofs for theorems. LLMs are scaled-up transformer models trained on a large amount of text data, including natural language and code, that have proven to be remarkably effective across a wide variety of applications, including question answering, and text and code generation [7], [14]. Here, we show their remarkable effectiveness for whole proof generation.

ESEC/FSE 2023 Distinguished Paper

1. INTRODUCTION

Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle-HOL, e.g., by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation model with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that (1) Whole-proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the ensuing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle-HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 34.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

The main contributions of our work are:

- We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammers or computationally expensive search.
- We demonstrate that LLMs can generate proofs with LLMs for theorems that were previously considered too difficult for the proof assistant’s error messages.
- We demonstrate empirically on a large benchmark that Baldur, when combined with prior techniques, significantly improves the state-of-the-art for theorem proving.

CC BY, 4/2023. This paper is licensed under a Creative Commons Attribution 4.0 International License. We have made our implementation available at <https://github.com/EmilyFirst/baldur>. We evaluate our implementation using two versions of Mistral v0.1 [18], one with 8 billion parameters and another with 62 billion parameters. By contrast, existing tools that use LLMs for theorem

Preprint 2023

Supervised Models: Building a Large Proof Repair Dataset

1 Tom Reichel ✉
2 University of Illinois Urbana-Champaign, USA

3 R. Wesley Henderson ✉
4 Radiance Technologies, Inc., Huntsville, AL, USA

5 Andrew Touchet ✉
6 Radiance Technologies, Inc., Huntsville, AL, USA

7 Andrew Gardner* ✉
8 Radiance Technologies, Inc., Huntsville, AL, USA

9 Talia Ringer* ✉
10 University of Illinois Urbana-Champaign, USA

11

12 **Abstract**

13 We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide recommendations for how the proof assistant community can address them. Our hope is to make it easier to mine datasets and make them so that machine-learning tools for proofs will move to target the tasks that machine-learning tools can address more effectively across proof assistants.

14

15 2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

LM-Based Automation (Part 3 of 5)

Since Then

Passport: Improving Automated Formal Verification Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

Formally verifying system properties is one of the most effective ways to improve its high manual effort requirements often render it prohibitively expensive. To verify, by learning from proof corpora to suggest proofs, have just begun to suggest tools are effective because of the richness of the data the proof corpora contain. The stylistic conventions followed by communities of proof developers, together with systems beneath proof assistants. However, this richness remains underexploited focusing on architecture rather than on how to make the most of the proof data.

In this paper, we develop Passport, a fully-automated proof-synthesis tool that how to most effectively exploit one aspect of that proof data: identifiers. Passport enriches a predictive Coq model used by proof-synthesis tools with three new encoding mechanisms for identifiers: category vocabulary indexing, subword sequence modeling, and path elaboration. We compare Passport to three existing base tools: *Hammer*, *AStactic*, and *Tok*. In head-to-head comparisons, Passport automatically proves 38% more theorems than the best of the three tools. Combining Passport with the other two enhanced tools automatically proves 38% more theorems than the three base tools together, without Passport's enhancements. Finally, together, these base tools and Passport tools enhanced with identifier information prove 45% more theorems than the three base tools with Passport's enhancements. In general, our findings suggest that a good identifier encoding may significantly improve the quality of the proof, leading to higher-quality software.

**TOPLAS Vol. 45, Issue 2:
No. 12, pp 1-50, 2023**

Proofster: Automated Formal Verification

Arpan Agrawal
University of Illinois
Urbana-Champaign, IL, USA
arpan2@illinois.edu

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Zhanna Kaufman
University of Massachusetts
Amherst, MA, USA
zhanna.kaufma@cs.umass.edu

Tom Reichel
University of Illinois
Urbana-Champaign, IL, USA
reichel3@illinois.edu

Shizhuo Zhang
University of Illinois
Urbana-Champaign, IL, USA
shizhuo2@illinois.edu

Timothy Zhou
University of Illinois
Urbana-Champaign, IL, USA
tz2@illinois.edu

Alex Sanchez-Stern
University of Massachusetts
Amherst, MA, USA
sanchezsstern@cs.umass.edu

Talia Ringer
University of Illinois
Urbana-Champaign, IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

Abstract—Formal verification is an effective but extremely work-intensive method of improving software quality. Verifying the correctness of software systems often requires significantly more effort than implementing them in the first place, despite the existence of proof assistants, such as Coq, aiding the process. Recent work has aimed to fully automate the synthesis of formal verification proofs, but little tool support exists for practitioners. This paper presents Proofster, a web-based tool aimed at assisting developers with the formal verification process via proof synthesis. Proofster inputs a Coq theorem specifying a property of a software system and attempts to automatically synthesize a formal proof of the correctness of that theorem. When it is unable to produce a proof, Proofster offers its synthesis explored, a hint to enable Proofster online at <https://proofster.org>. Proofster is available at <https://github.com/proofster/proofster>.

Meanwhile, it took 11 person-years to write the proofs required to verify the *seL4* microkernel [17], which represents a tiny fraction of the functionality of a full kernel. Recent work has aimed to simplify the process of writing proofs [2], [6], [7], [9], [10], [14], [11], [23], [24], [30]. Some formal verification can even be fully automated via proof synthesis. For example, CoqHammer [4] uses a set of precomputed mathematical facts to attempt to “hammer” out a proof. Meanwhile, AStactic [30], Proverbot0001 [23], TacTok [17], Diva [6], and Passport [24] learn a predictive model to guide

the above-mentioned search-based tools, all but one have neither

ICSE Demo 2023

Building a Large Proof Repair Dataset

- 1 Tom Reichel ✉
- 2 University of Illinois Urbana-Champaign, USA
- 3 R. Wesley Henderson ✉
- 4 Radiance Technologies, Inc., Huntsville, AL, USA
- 5 Andrew Touchet ✉
- 6 Radiance Technologies, Inc., Huntsville, AL, USA
- 7 Andrew Gardner* ✉
- 8 Radiance Technologies, Inc., Huntsville, AL, USA
- 9 Talia Ringer* ✉
- 10 University of Illinois Urbana-Champaign, USA

Abstract

We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide a community can address them. Our hope is to make it easier for machine-learning tools to learn from this dataset. We hope that machine-learning tools will move to target this dataset across proof assistants.

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Talia Ringer
University of Illinois Urbana-Champaign
IL, USA
tringer@illinois.edu

Markus N. Rabe
Google, Inc.
CA, USA
mrabe@google.com

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT

Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle-HOL, e.g., by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation model with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that (1) Whole proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the ensuing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle-HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 3.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

As a result, recent research has focused on automated proof synthesis, which can lead to fully automating formal verification.

There are two promising approaches for automating proof synthesis. The first is to use *hammers*, such as Sledgehammer [64] for the Isabelle proof assistant. Hammers iteratively apply known mathematical facts using heuristics. The second is to use search-based *neural theorem provers*, such as DeepProof [4], GPT4 [66], TacticZero [91], Lisa [54], EvolveIt [42], Diva [60], TacTok [72], and AStactic [36]. Given a partial proof and the current *proof state* (which consists of the current goal to prove and the list of known assumptions), these tools use neural networks to predict the next *individual proof step*. They use the *proof assistant* to evaluate the proposed next proof steps, which returns a new set of proof states. Neural theorem provers rely on diverse neural architectures, such as Wavernet [4, 84], graph neural networks [62], short long-term memory models [20], and language models with the transformer architecture [27, 66].

In this paper, we propose Baldur, a different, simpler approach to proof synthesis. We show that using large language models (LLMs), fine-tuned on proofs, can produce entire proofs for theorems. LLMs are scaled-up transformer models trained on a large amount of text data, including natural language and code, that have proven to be remarkably effective across a wide variety of applications, including question answering, and text and code generation [7, 14]. Here, we show their remarkable effectiveness for whole proof generation.

The main contributions of our work are:
• We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammers or conventional algorithmically expensive search.
• We repair and demonstrate that generated proofs with LLMs further improve proving power when the LLM proof assistant’s error messages originate on a large benchmark that

ESEC/FSE 2023 Distinguished Paper

EMILY FIRST, TALIA RINGER, YURIY BRUN, ALEX SANCHEZ-STERN, MARKUS N. RABE, AND ZHANNA KAUFMAN. 2023. BALDUR: WHOLE-PROOF GENERATION AND REPAIR WITH LARGE LANGUAGE MODELS. *ACM SIGPLAN NOTICES*, Vol. 54, No. 4, Article 14. 14 pages.

Supervised Models:

- 1 Tom Reichel ✉
- 2 University of Illinois Urbana-Champaign, USA
- 3 R. Wesley Henderson ✉
- 4 Radiance Technologies, Inc., Huntsville, AL, USA
- 5 Andrew Touchet ✉
- 6 Radiance Technologies, Inc., Huntsville, AL, USA
- 7 Andrew Gardner* ✉
- 8 Radiance Technologies, Inc., Huntsville, AL, USA
- 9 Talia Ringer* ✉
- 10 University of Illinois Urbana-Champaign, USA

Abstract

We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide a community can address them. Our hope is to make it easier for machine-learning tools to learn from this dataset. We hope that machine-learning tools will move to target this dataset across proof assistants.

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

LM-Based Automation (Part 3 of 5)

04.10370v2 [cs.PL] 2 Aug 2022

Second Project: Proofster

Proofster: Automated Formal Verification

Arpan Agrawal
University of Illinois
Urbana-Champaign, IL, USA
arpan2@illinois.edu

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Zhanna Kaufman
University of Massachusetts
Amherst, MA, USA
zhanna Kaufman@cs.umass.edu

Tom Reichel
University of Illinois
Urbana-Champaign, IL, USA
reichel3@illinois.edu

Shizhuo Zhang
University of Illinois
Urbana-Champaign, IL, USA
shizhuo2@illinois.edu

Timothy Zhou
University of Illinois
Urbana-Champaign, IL, USA
ttz2@illinois.edu

Alex Sanchez-Stern
University of Massachusetts
Amherst, MA, USA
sanchezsstern@cs.umass.edu

Talia Ringer
University of Illinois
Urbana-Champaign, IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

Abstract—Formal verification is an effective but extremely work-intensive method of improving software quality. Verifying the correctness of software systems often requires significantly more effort than implementing them in the first place, despite the existence of proof assistants, such as Coq, aiding the process. Recent work has aimed to fully automate the synthesis of formal verification proofs, but little tool support exists for practitioners. This paper presents Proofster, a web-based tool aimed at assisting developers with the formal verification process via proof synthesis. Proofster inputs a Coq theorem specifying a property of a software system and attempts to automatically synthesize a formal proof of the correctness of that theorem. When it is unable to produce a proof, Proofster provides hints to the user. Proofster's synthesis explored, we hint to enable Proofster online at <https://proofster.org>. Proofster is available at <https://github.com/proofster/proofster>.

Meanwhile, it took 11 person-years to write the proofs required to verify the seL4 microkernel [17], which represents a tiny fraction of the functionality of a full kernel. Recent work has aimed to simplify the process of writing proofs [2], [6], [7], [9], [10], [14], [11], [23], [24], [30]. Some formal verification can even be fully automated via proof synthesis. For example, CoqHammer [4] uses a set of precomputed mathematical facts to attempt to “hammer” out a proof. Meanwhile, ASTactic [30], Proverbot0001 [23], TacTok [17], Diva [6], and Passport [24] learn a predictive model to guide the proof synthesis process. In this paper, we present Proofster, a web-based tool that generates whole-proof formal proofs using LLMs, without using hammer-like search-based techniques. We show that Proofster is as effective as search-based techniques without requiring costly search. We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle/HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 3.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

ICSE Demo 2023

- 1 Tom Reichel ✉
- 2 University of Illinois Urbana-Champaign, USA
- 3 R. Wesley Henderson ✉
- 4 Radiance Technologies, Inc., Huntsville, AL, USA
- 5 Andrew Touchet ✉
- 6 Radiance Technologies, Inc., Huntsville, AL, USA
- 7 Andrew Gardner* ✉
- 8 Radiance Technologies, Inc., Huntsville, AL, USA
- 9 Talia Ringer* ✉
- 10 University of Illinois Urbana-Champaign, USA

Abstract

We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide a community can address them. Our hope is to make it easier for machine-learning tools to learn from this dataset. We hope that machine-learning tools will move to target this dataset across proof assistants.

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

Passport: Improving Automated Formal Verifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

Formally verifying system properties is one of the most effective ways of improving its high manual effort requirements often render it prohibitively expensive. Verification, by learning from proof corpora to suggest proofs, have just begun to tools are effective because of the richness of the data the proof corpora contain. The stylistic conventions followed by communities of proof developers, together with systems beneath proof assistants. However, this richness remains underexploited focusing on architecture rather than on how to make the most of the proof data.

In this paper, we develop Passport, a fully-automated proof-synthesis tool that how to most effectively exploit one aspect of that proof data: identifiers. Passport model used by proof-synthesis tools with three new encoding mechanisms for identifiers: category vocabulary indexing, subword sequence modeling, and path elaboration. We compare Passport to three existing base tools: Isabelle/HOL, Coq, and Tact. In head-to-head comparisons, Passport automatically proves 38% more theorems than the three base tools together, without Passport’s enhancements. Finally, together, these base tools and Passport tools enhanced with identifier information prove 45% more theorems than the three base tools with Passport’s enhancements. In general, our findings suggest that a good identifier encoding can significantly improve the quality of proof synthesis leading to higher-quality software.

TOPLAS Vol. 45, Issue 2: No. 12, pp 1-50, 2023

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Talia Ringer
University of Illinois Urbana-Champaign
IL, USA
tringer@illinois.edu

Markus N. Rabe
Google, Inc.
CA, USA
mrabe@google.com

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT

Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle/HOL, e.g., by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation model with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that (1) Whole proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the existing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle/HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 3.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

As a result, recent research has focused on automated proof synthesis, which can lead to fully automating formal verification.

There are two promising approaches for automating proof synthesis. The first is to use hammers, such as Sledgehammer [64] for the Isabelle proof assistant. Hammers iteratively apply known mathematical facts using heuristics. The second is to use search-based neural theorem provers, such as DeepEuler [4], GPT4 [66], TacticZero [91], Lisa [54], EvolveIt [42], Diva [60], TacTok [22], and ASTactic [36]. Given a partial proof and the current proof state (which consists of the current goal to prove and the list of known assumptions), these tools use neural networks to predict the next individual proof step. They use the proof assistant to evaluate the proposed next proof steps, which returns a new set of proof states. Neural theorem provers rely on diverse neural architectures, such as Wavernet [48], graph neural networks [62], short long-term memory models [20], and language models with the transformer architecture [27, 66].

In this paper, we propose Baldur, a different, simpler approach to proof synthesis. We show that using large language models (LLMs), fine-tuned on proofs, can produce entire proofs for theorems. LLMs are scaled-up transformer models trained on a large amount of text data, including natural language and code, that have proven to be remarkably effective across a wide variety of applications, including question answering, and text and code generation [7, 14]. Here, we show their remarkable effectiveness for whole proof generation.

The main contributions of our work are:

- We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammer-like search-based techniques.
- We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammer-like search-based techniques. We show that Proofster is as effective as search-based techniques without requiring costly search. We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle/HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 3.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

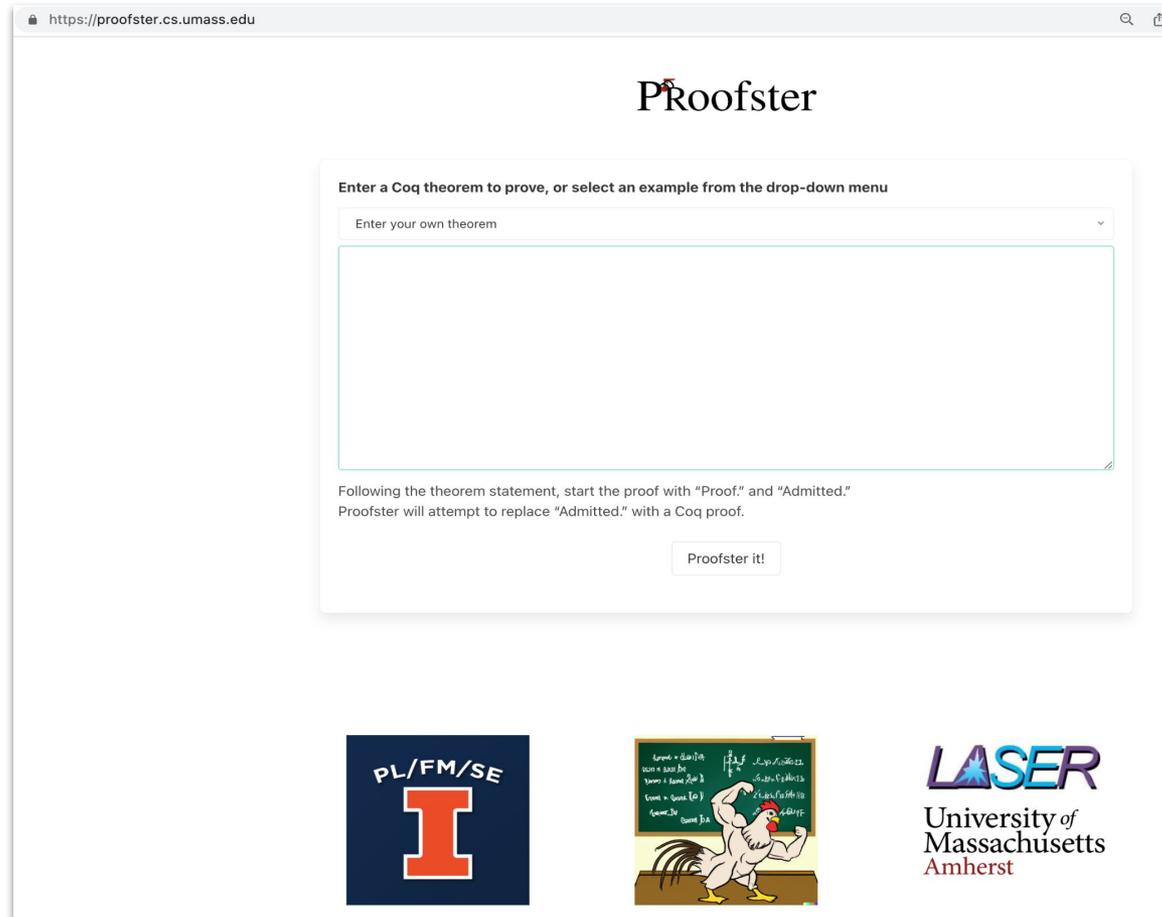
ESEC/FSE 2023 Distinguished Paper

EMILY FIRST, TALIA RINGER, YURIY BRUN, MARKUS N. RABE, AND ZHANNA KAUFMAN. THE PROVERBOT DATASET: A LARGE-BENCHMARK OF MACHINE-GENERATED PROOFS WITH LLMs FOR FURTHER EVALUATING THE PROVER POWER OF THE LLM PROOF ASSISTANT’S ERROR MESSAGES. SPECIALLY ON A LARGE BENCHMARK THAT

Supervised Models: Dataset

LM-Based Automation (Part 3 of 5)

Second Project: Proofster



<https://proofster.cs.umass.edu/>

LM-Based Automation (Part 3 of 5)

Second Project: Proofster

```
Inductive ev: nat → Prop :=
| ev_0 : ev 0
| ev_SS (n: nat) (H: ev n) : ev (S (S n)).
```

```
Theorem ev_inversion: forall (n: nat),
  ev n →
  (n = 0) ∨ (exists n', n = S (S n') ∧ ev n'). =
```

```
Proof. =
intros. =
```

```
n : nat  H : ev n
-----
n = 0 ∨ (exists n' : nat, n = S (S n') ∧ ev n')
```

```
elim H. =
left. =
eauto. =
```

```
n : nat  H : ev n
-----
forall n : nat,
ev n →
n = 0 ∨ (exists n' : nat, n = S (S n') ∧ ev n') →
S (S n) = 0 ∨
(exists n' : nat, S (S n) = S (S n') ∧ ev n')
```

```
intros. =
destruct H1. =
```

```
n : nat  H : ev n  n0 : nat  H0 : ev n0  H1 : n0 = 0
-----
S (S n0) = 0 ∨
(exists n' : nat, S (S n0) = S (S n') ∧ ev n')
```

```
S (S n0) = 0 ∨
(exists n' : nat, S (S n0) = S (S n') ∧ ev n')
```

```
eauto. =
eauto.
Qed.
```

Third Project: PRISM

Proofster: Automated Formal Verification

Arpan Agrawal
University of Illinois
Urbana-Champaign, IL, USA
arpan2@illinois.edu

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Zhanna Kaufman
University of Massachusetts
Amherst, MA, USA
zhanna.kaufman@cs.umass.edu

Tom Reichel
University of Illinois
Urbana-Champaign, IL, USA
reichel3@illinois.edu

Shizhuo Zhang
University of Illinois
Urbana-Champaign, IL, USA
shizhuo2@illinois.edu

Timothy Zhou
University of Illinois
Urbana-Champaign, IL, USA
tz2@illinois.edu

Alex Sanchez-Stern
University of Massachusetts
Amherst, MA, USA
sanchezsstern@cs.umass.edu

Talia Ringer
University of Illinois
Urbana-Champaign, IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

Abstract—Formal verification is an effective but extremely work-intensive method of improving software quality. Verifying the correctness of software systems often requires significantly more effort than implementing them in the first place, despite the existence of proof assistants, such as Coq, aiding the process. Recent work has aimed to fully automate the synthesis of formal verification proofs, but little tool support exists for practitioners. This paper presents Proofster, a web-based tool aimed at assisting developers with the formal verification process via proof synthesis. Proofster inputs a Coq theorem specifying a property of a software system and attempts to automatically synthesize a formal proof of the correctness of that property. When it is unable to produce a proof, Proofster outputs the **proof synthesis** and **proof synthesis** its synthesis explored, which guides the developer to a hint to enable Proofster's synthesis. This paper is available at <https://proofster.cs.umass.edu> and a video demo at <https://youtu.be/xQA166lRfw>.

Meanwhile, it took 11 person-years to write the proofs required to verify the `seL4` microkernel [17], which represents a tiny fraction of the functionality of a full kernel. Recent work has aimed to simplify the process of writing proofs [2], [6], [7], [9], [10], [14], [11], [23], [24], [30]. Some formal verification can even be fully automated via proof synthesis. For example, CoqHammer [4] uses a set of precomputed mathematical facts to attempt to “hammer” out a proof. Meanwhile, ASTactic [30], Proverbot9001 [23], TacTic [7], Dava [6], and Passport [24] learn a **proof synthesis** model

Passport: Improving Automated Formal Verification Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

Formally verifying system properties is one of the most effective ways of improving its high manual effort requirements often render it prohibitively expensive. Tool verification, by learning from proof corpora to suggest proofs, have just begun to suggest tools are effective because of the richness of the data the proof corpora contain. The stylistic conventions followed by communities of proof developers, together with systems beneath proof assistants. However, this richness remains underexploited focusing on architecture rather than on how to make the most of the proof data.

In this paper, we develop Passport, a fully-automated proof-synthesis tool that how to most effectively exploit one aspect of that proof data: identifiers. Passport enriches a predictive Coq model used by proof-synthesis tools with three new encoding mechanisms for identifiers: category vocabulary indexing, subword sequence modeling, and path elaboration. We compare Passport to three existing base tools

TOPLAS Vol. 45, Issue 2: No. 2, pp 1-50, 2023

enhance tools automatically proves 38% more theorems than the three base tools together, without Passport’s enhancements. Finally, together, these base tools and Passport tools enhanced with identifier information prove 45% more theorems than the three base tools with Passport’s enhancements. In general, our findings suggest that exploring identifiers can pay a significant role in improving proof synthesis leading to higher-quality software.

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Talia Ringer
University of Illinois Urbana-Champaign
IL, USA
tringer@illinois.edu

Markus N. Rabe
Google, Inc.
CA, USA
mrabe@google.com

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT

Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle-HOL, e.g., by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation model with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that (1) Whole proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the ensuing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle-HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 3.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

As a result, recent research has focused on automated proof synthesis, which can lead to fully automating formal verification.

There are two promising approaches for automating proof synthesis. The first is to use *hammers*, such as Sledgehammer [64] for the Isabelle proof assistant. Hammers iteratively apply known mathematical facts using heuristics. The second is to use search-based *neural theorem provers*, such as DeepProof [14], GPT4 [60], TacticZero [91], Lisa [54], EvolveIt [42], Dava [60], TacTic [22], and ASTactic [36]. Given a partial proof and the current *proof state* (which consists of the current goal to prove and the list of known assumptions), these tools use neural networks to predict the next *individual proof step*. They use the *proof assistant* to evaluate the proposed next proof steps, which returns a new set of proof states. Neural theorem provers rely on diverse neural architectures, such as Wavenet [4, 84], graph neural networks [62], short long-term memory models [20], and language models with the transformer architecture [27, 66].

In this paper, we propose Baldur, a different, simpler approach to proof synthesis. We show that using large language models (LLMs), fine-tuned on proofs, can produce entire proofs for theorems. LLMs are scaled-up transformer models trained on a large amount of text data, including natural language and code, that have proven to be remarkably effective across a wide variety of applications, including question answering, and text and code generation [7, 14]. Here, we show their remarkable effectiveness for whole proof generation.

The main contributions of our work are:

- We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammers or conventional algorithmically expensive search.
- We repair and demonstrate that repair generated proofs with LLMs further improve their proving power when the LLM proof assistant’s error messages, especially on a large benchmark that

ESEC/FSE 2023 Distinguished Paper

1 Proof Repair Infrastructure for Supervised Models: Building a Large Proof Repair Dataset

2 Tom Reichel ✉
3 University of Illinois Urbana-Champaign, USA
4 R. Wesley Henderson ✉
5 Radiance Technologies, Inc., Huntsville, AL, USA
6 Andrew Touchet ✉
7 Radiance Technologies, Inc., Huntsville, AL, USA
8 Andrew Gardner* ✉
9 Radiance Technologies, Inc., Huntsville, AL, USA
10 Talia Ringer* ✉
11 University of Illinois Urbana-Champaign, USA

12 Abstract

13 We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide a community can address them.
14 Our hope is to make it easier for machine-learning tools that machine-learning tools iteratively across proof assistants.

ITP 2023

15 2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

04.10370v2 [cs.PL] 2 Aug 2022

LM-Based Automation (Part 3 of 5)

Third Project: PRISM

- **Dataset** for **proof repair** models for Coq
- **Actual proof repairs** by proof engineers
- **Collaboration with Radiance**
- **Massive infrastructure undertaking**
 - Building many different projects
 - ... with many different Coq versions
 - ... for many different commits
 - ... and aligning data across commit pairs
- **WIP Training Repair Models**

Fourth Project: Baldur

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Markus N. Rabe
Google, Inc.
CA, USA
mrabe@google.com

Talia Ringer
University of Illinois Urbana-Champaign
IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT

Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle-HOL, e.g., by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation model with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that (1) Whole proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the existing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle-HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool, Thor, by automatically generating proofs for an additional 3.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

As a result, recent research has focused on automated proof synthesis, which can lead to fully automating formal verification.

There are two promising approaches for automating proof synthesis. The first is to use *hammers*, such as Sledgehammer [64] for the Isabelle proof assistant. Hammers iteratively apply known mathematical facts using heuristics. The second is to use search-based *neural theorem provers*, such as DeepHKM [4], GPT4 [66], TacticZero [91], Lisa [54], EvolveIt [42], Diva [20], TacTok [22], and AS2Tactic [96]. Given a partial proof and the current *proof state* (which consists of the current goal to prove and the list of known assumptions), these tools use neural networks to predict the next *individual proof step*. They use the *proof assistant* to evaluate the proposed next proof steps, which returns a new set of proof states. Neural theorem provers rely on diverse neural architectures, such as Wavenet [4, 84], graph neural networks [62], short long-term memory models [20], and language models with the transformer architecture [27, 66].

In this paper, we propose Baldur, a different, simpler approach to proof synthesis. We show that using large language models (LLMs), fine-tuned on proofs, can produce entire proofs for theorems. LLMs are scaled-up transformer models trained on a large amount of text data, including natural language and code, that have proven to be remarkably effective across a wide variety of applications, including question answering, and text and code generation [7, 14]. Here, we show their remarkable effectiveness for whole proof generation.

- The main contributions of our work are:
 - We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammers or conventional, typically expensive search.
 - We repair and demonstrate that generated proofs with LLMs further improve proving power when the LLM proof assistant's error messages originate on a large benchmark that

ESEC/FSE 2023 Distinguished Paper

PROVER IS 10 TIMES PROBABLY FIVE TIMES AS FAST AS THE SMC PROVER IS MORE THAN THREE TIMES AS LONG AS THE COMPILER CODE ITSELF [67].

Proofster: Automated Formal Verification

Arpan Agrawal
University of Illinois
Urbana-Champaign, IL, USA
arpan2@illinois.edu

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Zhanna Kaufman
University of Massachusetts
Amherst, MA, USA
zhanna.kaufma@cs.umass.edu

Tom Reichel
University of Illinois
Urbana-Champaign, IL, USA
reichel3@illinois.edu

Shizhuo Zhang
University of Illinois
Urbana-Champaign, IL, USA
shizhuo2@illinois.edu

Timothy Zhou
University of Illinois
Urbana-Champaign, IL, USA
tz2@illinois.edu

Alex Sanchez-Stern
University of Massachusetts
Amherst, MA, USA
sanchezsstern@cs.umass.edu

Talia Ringer
University of Illinois
Urbana-Champaign, IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

Abstract—Formal verification is an effective but extremely work-intensive method of improving software quality. Verifying the correctness of software systems often requires significantly more effort than implementing them in the first place, despite the existence of proof assistants, such as Coq, aiding the process. Recent work has aimed to fully automate the synthesis of formal verification proofs, but little tool support exists for practitioners. This paper presents Proofster, a web-based tool aimed at assisting developers with the formal verification process via proof synthesis. Proofster inputs a Coq theorem specifying a property of a software system and attempts to automatically synthesize a formal proof of the correctness of that property. When it is unable to produce a proof, Proofster outputs the *proof state* to the user, which its synthesis explored, which guides the developer to a hint to enable Proofster to synthesize the proof. Proofster is available at <https://proofster.cs.umass.edu> and a video demo at <https://youtu.be/xQA166lRfwI>.

Meanwhile, it took 11 person-years to write the proofs required to verify the `seL4` microkernel [17], which represents a tiny fraction of the functionality of a full kernel. Recent work has aimed to simplify the process of writing proofs [2], [6], [7], [9], [10], [14], [11], [23], [24], [30]. Some formal verification can even be fully automated via proof synthesis. For example, CoqHammer [4] uses a set of precomputed mathematical facts to attempt to “hammer” out a proof. Meanwhile, AS2Tactic [30], Proverbot9001 [23], TacTok [7], Diva [6], and Passport [24] learn a predictive model

Passport: Improving Automated Formal Verification Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

Formally verifying system properties is one of the most effective ways of improving its high manual effort requirements often render it prohibitively expensive. To verify, by learning from proof corpora to suggest proofs, have just begun to suggest tools are effective because of the richness of the data the proof corpora contain. The stylistic conventions followed by communities of proof developers, together with systems beneath proof assistants. However, this richness remains underexploited focusing on architecture rather than on how to make the most of the proof data.

In this paper, we develop Passport, a fully-automated proof-synthesis tool that how to most effectively exploit one aspect of that proof data: identifiers. Passport enriches a predictive Coq model used by proof-synthesis tools with three new encoding mechanisms for identifiers: category vocabulary indexing, subword sequence modeling, and path elaboration. We compare Passport to three existing base tools

TOPLAS Vol. 45, Issue 2: No. 2, pp 1-30, 2023

1 Proof Repair Infrastructure for Supervised Models: Building a Large Proof Repair Dataset

1 Tom Reichel
2 University of Illinois Urbana-Champaign, USA

3 R. Wesley Henderson
4 Radiance Technologies, Inc., Huntsville, AL, USA

5 Andrew Touchet
6 Radiance Technologies, Inc., Huntsville, AL, USA

7 Andrew Gardner*
8 Radiance Technologies, Inc., Huntsville, AL, USA

9 Talia Ringer*
10 University of Illinois Urbana-Champaign, USA

11 Abstract

12 We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide recommendations for how the proof assistant community can address them. Our hope is to make it easier to build sets of aligned definitions and proofs so that machine-learning tools for proofs will move to target the tasks that machine-learning tools can address more reliably across proof assistants.

13 2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

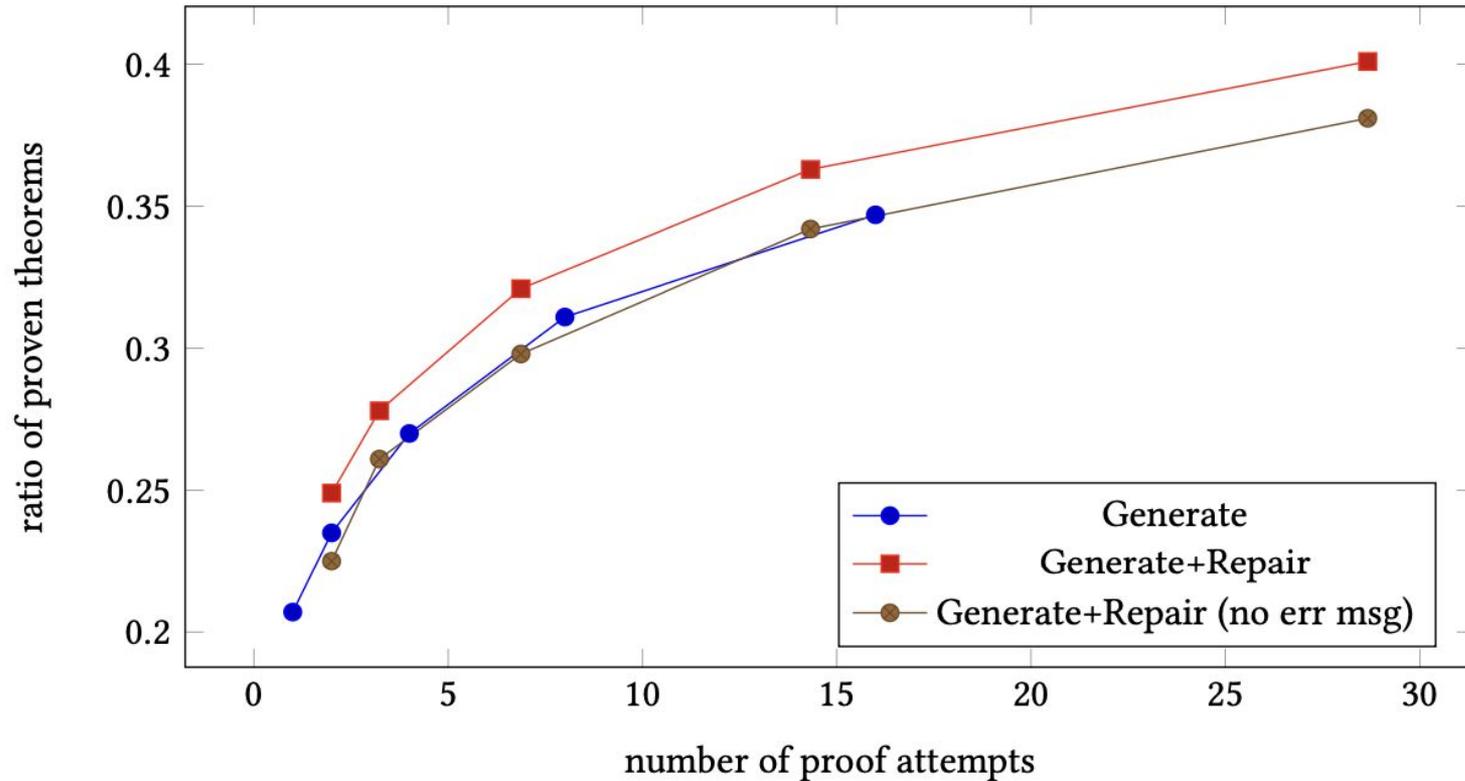
LM-Based Automation (Part 3 of 5)

04.10370v2 [cs.PL] 2 Aug 2022

Fourth Project: Baldur

- Using an **LLM**, one could, conceivably, synthesize **entire proofs at once**.
- Collaborating with Google, we fine-tuned the Minerva model to synthesize proofs in Isabelle/HOL
- Evaluated on PISA dataset (theorems in Isabelle/HOL)

Fourth Project: Baldur



Fourth Project: Baldur

- Baldur (without repair) can **synthesize whole proofs** for **47.9%** of the theorems, whereas search-based approaches prove **39.0%**.
- Baldur can **repair its own erroneous proof attempts** using the error message from the proof assistant, proving another **1.5%**.
- **Diversity continues to help.** Together with Thor, a tool that combines a model, search, and a hammer, Baldur can prove **65.7%**.

More in the Papers

Proofster: Automated Formal Verification

Arpan Agrawal
University of Illinois
Urbana-Champaign, IL, USA
arpan2@illinois.edu

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Zhanna Kaufman
University of Massachusetts
Amherst, MA, USA
zhanna Kaufman@cs.umass.edu

Tom Reichel
University of Illinois
Urbana-Champaign, IL, USA
reichel3@illinois.edu

Shizhuo Zhang
University of Illinois
Urbana-Champaign, IL, USA
shizhuo2@illinois.edu

Timothy Zhou
University of Illinois
Urbana-Champaign, IL, USA
tz2@illinois.edu

Alex Sanchez-Stern
University of Massachusetts
Amherst, MA, USA
sanchezsstern@cs.umass.edu

Talia Ringer
University of Illinois
Urbana-Champaign, IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

effective but extremely
ware quality. Verifying
n requires significantly
the first place, despite
Coq, aiding the process.
the synthesis of formal
exists for practitioners.
d tool aimed at assisting
cess via proof synthesis.
fying a property of a
ally synthesize a formal
When it is possible to

Meanwhile, it took 11 person-years to write the proofs required to verify the seL4 microkernel [17], which represents a tiny fraction of the functionality of a full kernel.

Recent work has aimed to simplify the process of writing proofs [2], [6], [7], [9], [10], [14], [11], [23], [24], [30]. Some formal verification can even be fully automated via proof synthesis. For example, CoqHammer [4] uses a set of precomputed mathematical facts to attempt to “hammer” out a proof. Meanwhile, ASTactic [30], Proverbot9001 [23], TacTok [17], Diva [6], and Passport [24] learn a predictive model to guide

model to guide
on scratch.
ists for practi-
or example, of
the above-mentioned search-based tools, all but one have neither

- E Demo 2023**
- 1 Tom Reichel ✉
University of Illinois Urbana-Champaign, USA
 - 2 R. Wesley Henderson ✉
Radiance Technologies, Inc., Huntsville, AL, USA
 - 3 Andrew Touchet ✉
Radiance Technologies, Inc., Huntsville, AL, USA
 - 4 Andrew Gardner* ✉
Radiance Technologies, Inc., Huntsville, AL, USA
 - 5 Talia Ringer* ✉
University of Illinois Urbana-Champaign, USA

Abstract

We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide r

ITP 2023

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

Baldur: Whole-Proof Generation and Repair with Large Language Models

Emily First
University of Massachusetts
Amherst, MA, USA
efirst@cs.umass.edu

Markus N. Rabe
Google, Inc.
CA, USA
mrabe@google.com

Talia Ringer
University of Illinois Urbana-Champaign
IL, USA
tringer@illinois.edu

Yuriy Brun
University of Massachusetts
Amherst, MA, USA
brun@cs.umass.edu

ABSTRACT

Formally verifying software properties is a highly desirable but labor-intensive task. Recent work has developed methods to automate formal verification using proof assistants, such as Coq and Isabelle-HOL, e.g. by training a model to predict one proof step at a time, and using that model to search through the space of possible proofs. This paper introduces a new method to automate formal verification: We use large language models trained on natural language text and code and fine-tuned on proofs, to generate whole proofs for theorems at once, rather than one step at a time. We combine this proof generation model with a fine-tuned repair model to repair generated proofs, further increasing proving power. As its main contributions, this paper demonstrates for the first time that: (1) Whole proof generation using transformers is possible and is as effective as search-based techniques without requiring costly search. (2) Giving the learned model additional context, such as a prior failed proof attempt and the ensuing error message, results in proof repair and further improves automated proof generation. (3) We establish a new state of the art for fully automated proof synthesis. We reify our method in a prototype, Baldur, and evaluate it on a benchmark of 6,336 Isabelle-HOL theorems and their proofs. In addition to empirically showing the effectiveness of whole-proof generation, repair, and added context, we show that Baldur improves on the state-of-the-art tool Thor, by automatically generating proofs for an additional 33.7% of the theorems. Together, Baldur and Thor can prove 65.7% of the theorems fully automatically. This paper paves the way for new research into using large language models for automating formal verification.

As a result, recent research has focused on automated proof synthesis, which can lead to fully automating formal verification.

There are two promising approaches for automating proof synthesis. The first is to use *hammers*, such as Sledgehammer [54] for the Isabelle proof assistant. Hammers iteratively apply known mathematical facts using heuristics. The second is to use search-based *neural theorem provers*, such as DeepFOLK [4], GPT-F [56], TacticZero [91], Lisa [54], EvolveIt [42], Diva [50], TacTok [22], and ASTactic [16]. Given a partial proof and the current *proof state* (which consists of the current goal to prove and the list of known assumptions), these tools use neural networks to predict the next *individual proof step*. They use the *proof assistant* to evaluate the proposed next proof steps, which returns a new set of proof states. Neural theorem provers rely on diverse neural architectures, such as Wavenets [4, 84], graph neural networks [62], short long-term memory models [20], and language models with the transformer architecture [27, 66].

In this paper, we propose Baldur, a different, simpler approach to proof synthesis. We show that using large language models (LLMs), fine-tuned on proofs, can produce entire proofs for theorems. LLMs are scaled-up transformer models trained on a large amount of text data, including natural language and code, that have proven to be remarkably effective across a wide variety of applications, including question answering, and text and code generation [7, 14]. Here, we show their remarkable effectiveness for whole proof generation.

- The main contributions of our work are:
- We develop Baldur, a novel method that generates whole formal proofs using LLMs, without using hammers or conventional, typically expensive search.
 - We repair and demonstrate that generated proofs with LLMs further improve their proving power when the LLM proof assistant’s error messages are explicitly on a large benchmark that

ESEC/FSE 2023 Distinguished Paper

EMILY FIRST, MARKUS N. RABE, AND TALIA RINGER. Baldur: Whole-Proof Generation and Repair with Large Language Models. In Proceedings of the ACM Conference on Computer Supported Software Reuse (CSCSR), 2023.

Building a Large Proof Repair Dataset

- Supervised Models:**
- 1 Tom Reichel ✉
University of Illinois Urbana-Champaign, USA
 - 2 R. Wesley Henderson ✉
Radiance Technologies, Inc., Huntsville, AL, USA
 - 3 Andrew Touchet ✉
Radiance Technologies, Inc., Huntsville, AL, USA
 - 4 Andrew Gardner* ✉
Radiance Technologies, Inc., Huntsville, AL, USA
 - 5 Talia Ringer* ✉
University of Illinois Urbana-Champaign, USA

Abstract

We introduce a new, large proof-repair dataset and benchmark suite for the Coq proof assistant. The dataset is made up of Git commits from dozens of open-source projects with old and new versions of definitions and proofs aligned across commits. Building this dataset was a significant undertaking, highlighting a number of challenges and gaps in existing infrastructure. We discuss these challenges and gaps, and we provide r

ITP 2023

2012 ACM Subject Classification Computing methodologies → Machine learning; Software and its

Passport: Improving Automated Formal Verification Using Identifiers

ALEX SANCHEZ-STERN*, University of Massachusetts Amherst, USA
EMILY FIRST*, University of Massachusetts Amherst, USA
TIMOTHY ZHOU, University of Illinois Urbana-Champaign, USA
ZHANNA KAUFMAN, University of Massachusetts Amherst, USA
YURIY BRUN, University of Massachusetts Amherst, USA
TALIA RINGER, University of Illinois Urbana-Champaign, USA

Formally verifying system properties is one of the most effective ways of improving system quality, but its high manual effort requirements often render it prohibitively expensive. Tools that automate formal verification, by learning from proof corpora to suggest proofs, have just begun to show their promise. These tools are effective because of the richness of the data the proof corpora contain. This richness comes from the stylistic conventions followed by communities of proof developers, together with the powerful logical systems beneath proof assistants. However, this richness remains underexploited, with most work thus far focusing on architecture rather than on how to make the most of the proof data.

In this paper, we develop Passport, a fully-automated proof-synthesis tool that systematically explores how to most effectively exploit one aspect of that proof data: identifiers. Passport enriches a predictive Coq model used by proof-synthesis tools with three new encoding mechanisms for identifiers: category vocabulary

**TOPLAS Vol. 45, Issue 2:
No. 12, pp 1-30, 2023**

Information
email, our
eading to

04.10370v2 [cs.PL] 2 Aug 2022

LM-Based Automation (Part 3 of 5)

Language models:

- *unpredictable*
- *not* dependable
- *not* understandable
- + *not very* limited in scope
- + takes *little* expertise to extend

Checking the Proof

Small Logical Kernel

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof **LMs**

LM-Based Automation (Part 3 of 5)



With **de Bruijn**, as long as you don't touch the **kernel**, your automation is **safe**.



With **de Bruijn**, as long as you don't touch the **kernel**, your automation is **safe**.^{*}
But *boy* does this make the **development process** suck.

Help at Every Stage

Spoiler!



With **de Bruijn**, as long as you don't touch the **kernel**, your automation is **safe**.^{*}
(If your specification is OK, your kernel has no bugs, and you don't introduce axioms)

LM-Based Automation (Part 3 of 5)

1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. **Best of Both Worlds**
5. **Opportunities**

Already Neurosymbolic

Checking the Proof

Small Logical Kernel

Search Procedures

Domain-Specific Heuristics

Proof Transformations

Producing the Proof **LMs**

Best of Both Worlds (Part 4 of 5)

But we want **even more** of the benefits of both kinds of automation.

Best of Both Worlds (Part 4 of 5)

Observation: We can do fairly well sometimes without **search**. Maybe we can use search at a **higher level** than before and get further returns?

Best of Both Worlds (Part 4 of 5)

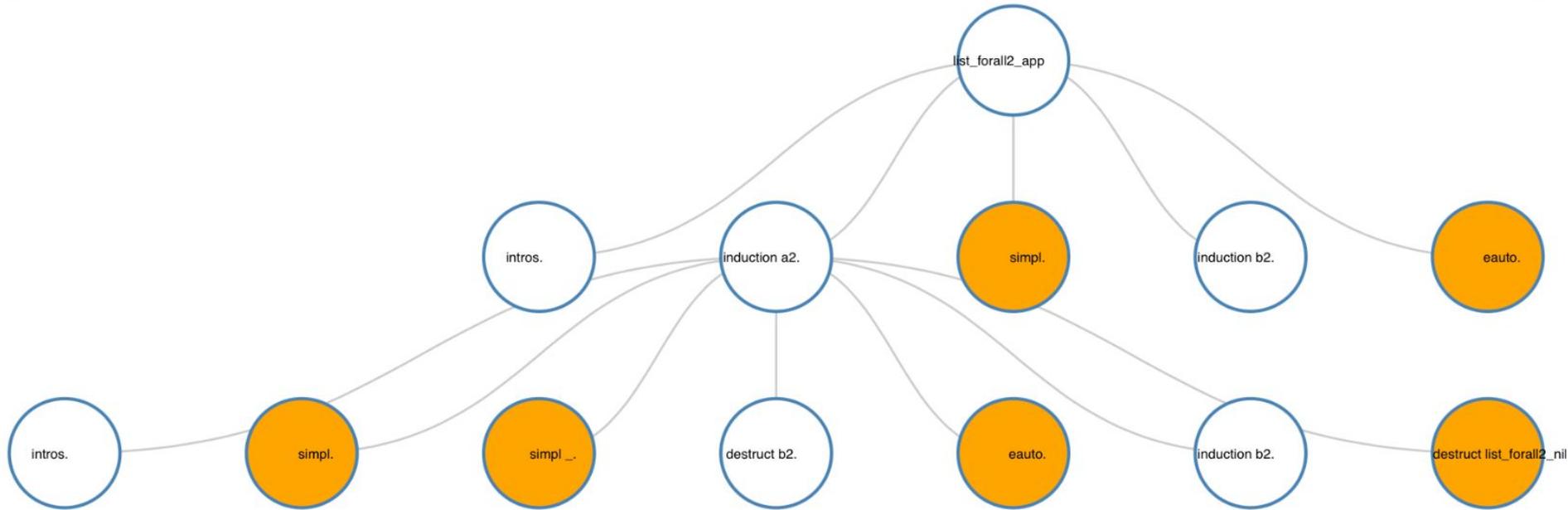
One idea: Move the search process *up* in abstraction.

Best of Both Worlds (Part 4 of 5)

One idea: Move the search process *up* in abstraction.

Best of Both Worlds (Part 4 of 5)

Proof Search



Best of Both Worlds (Part 4 of 5)

Conversational Action Search

Getting More out of Large Language Models for Proofs

Shizhuo Dylan Zhang¹, Emily First², and Talia Ringer¹

¹ University of Illinois Urbana-Champaign, USA

² University of Massachusetts Amherst, USA

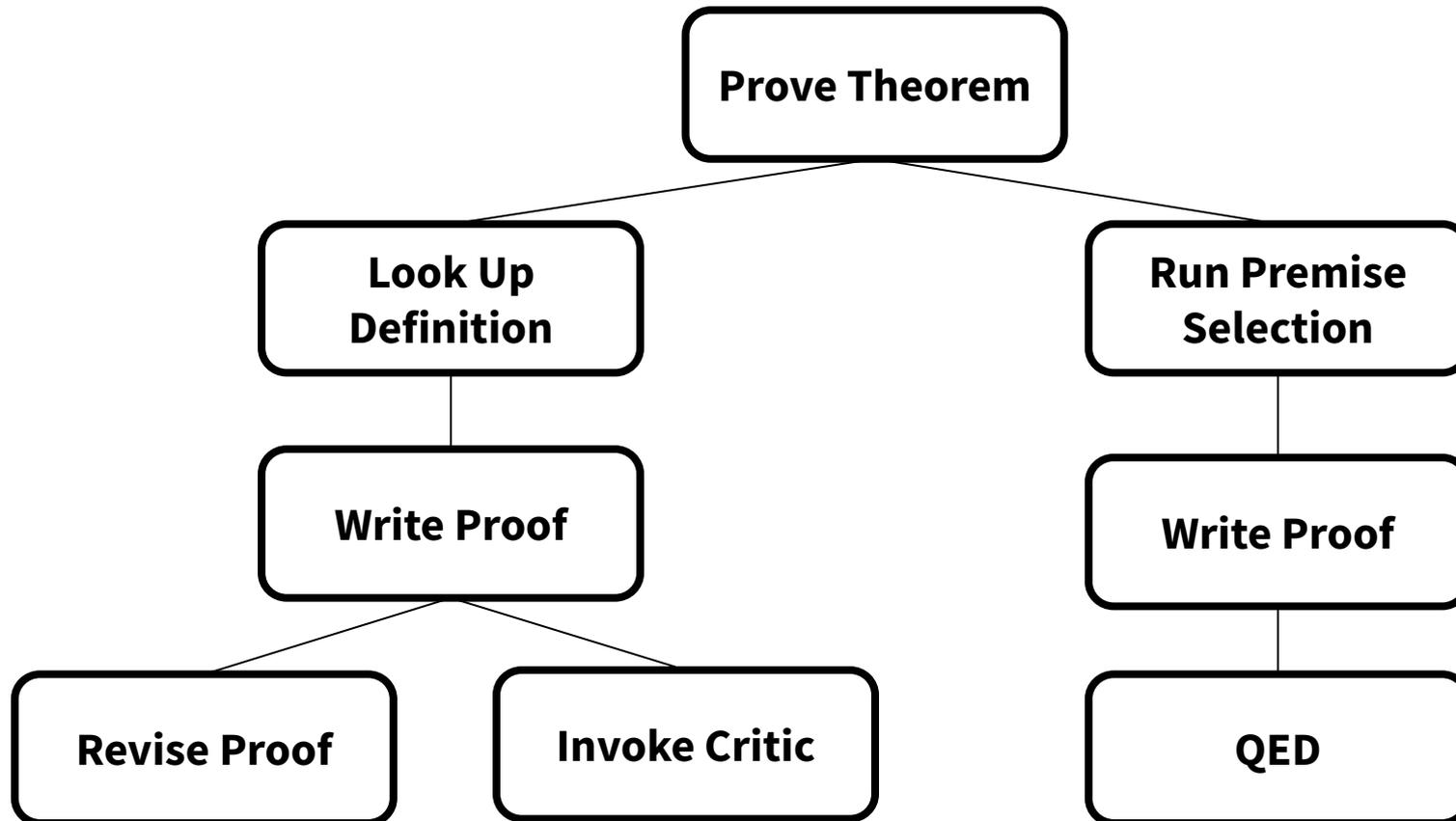
Abstract

Large language models have the potential to simplify formal theorem proving and make it more accessible. But how to get the most out of these models is still an open question. To answer this question, we take a step back and explore the failure cases of these models using common prompting-based techniques. Our talk will discuss these failure cases and what they can teach us about how to use these models.

AITP 2023

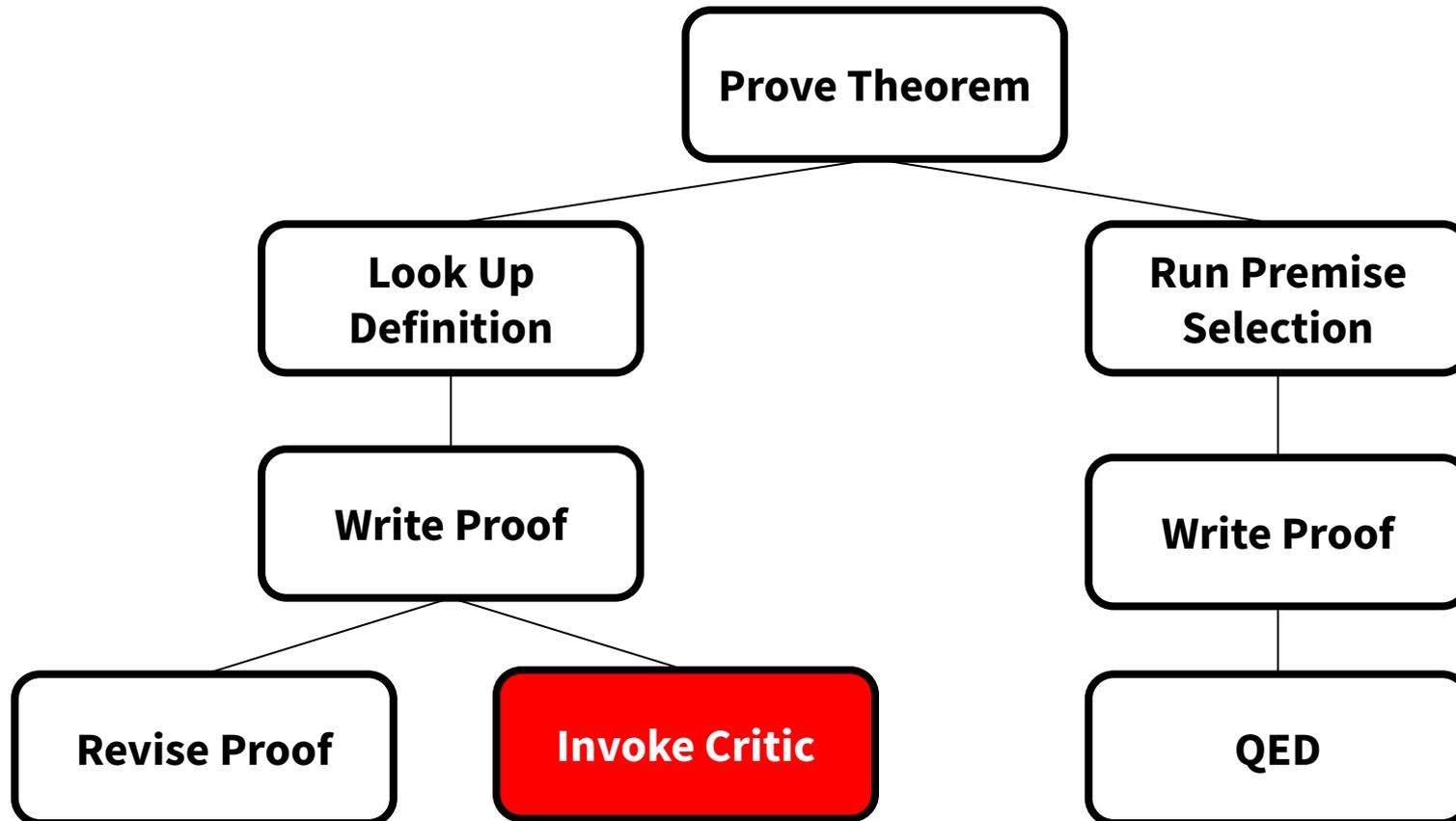
Best of Both Worlds (Part 4 of 5)

Conversational Action Search



Best of Both Worlds (Part 4 of 5)

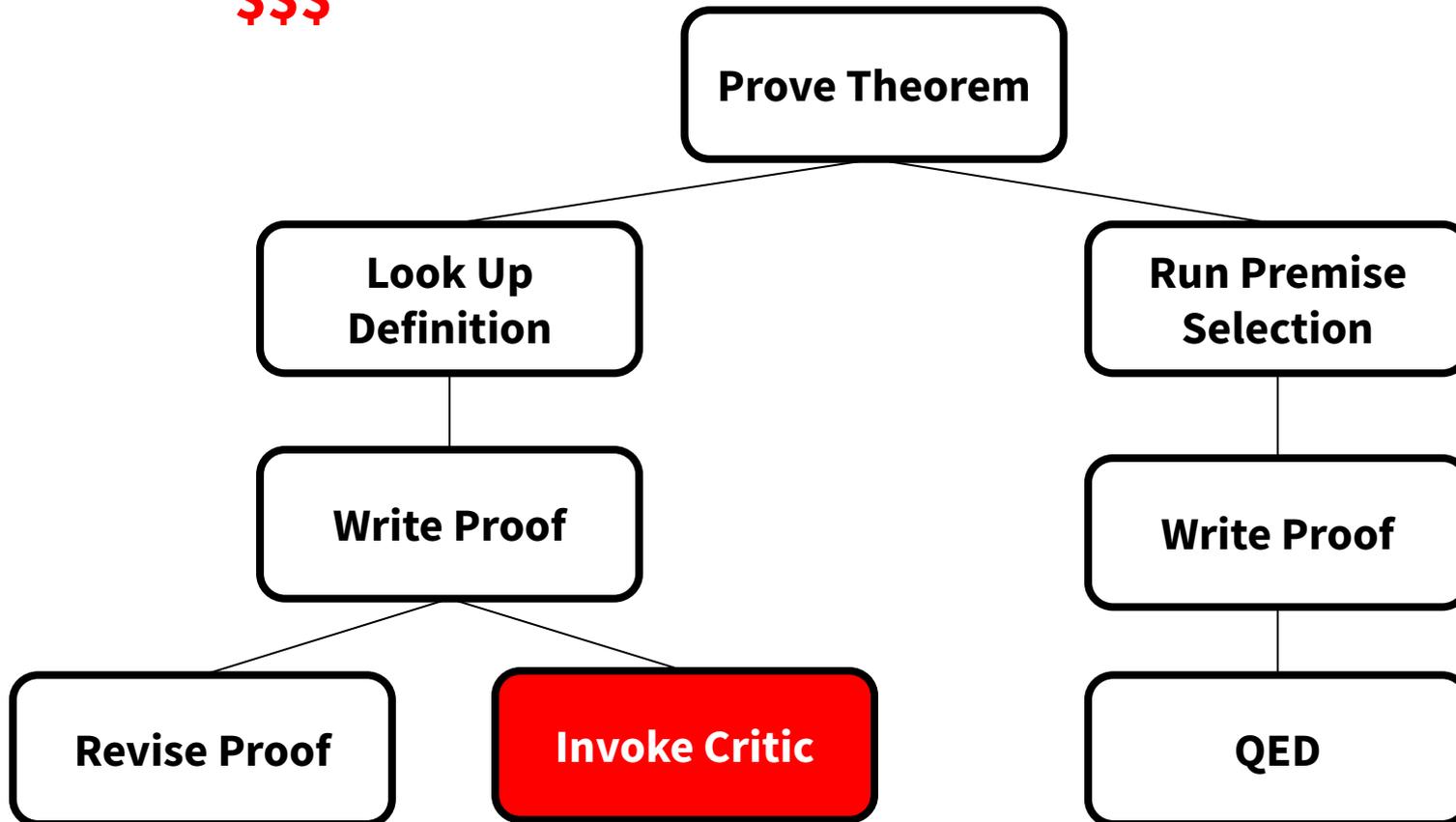
Conversational Action Search



Best of Both Worlds (Part 4 of 5)

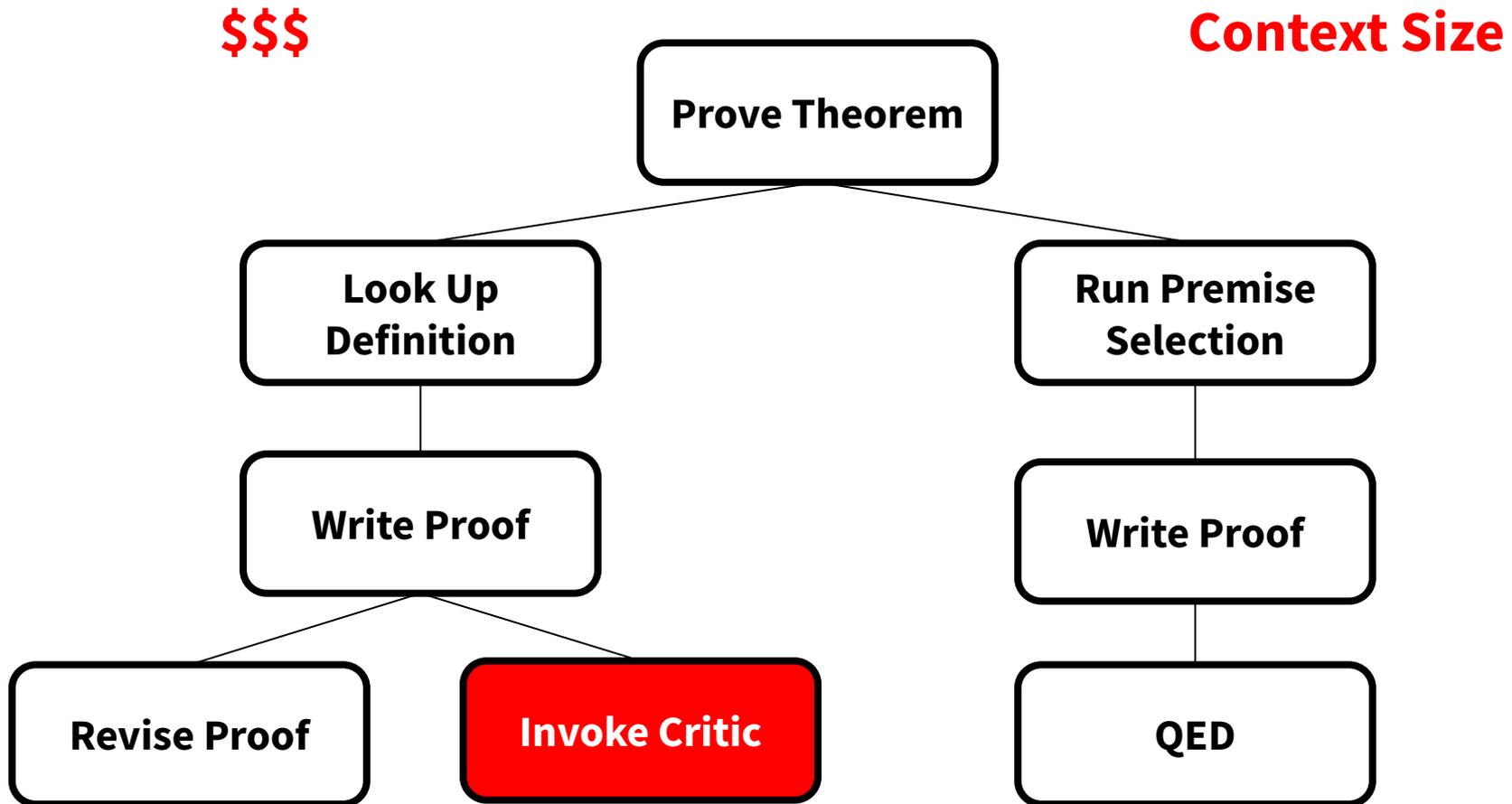
Conversational Action Search

\$\$\$



Best of Both Worlds (Part 4 of 5)

Conversational Action Search



Best of Both Worlds (Part 4 of 5)

Conversational Action Search

Promising Results

Best of Both Worlds (Part 4 of 5)

Observation: Diversity in models helps, and diversity in techniques appears to help, too. Let's keep taking advantage of that.

Best of Both Worlds (Part 4 of 5)

Soon: Best of both worlds
for **proof repair**, too.

Best of Both Worlds (Part 4 of 5)

1. Proof Assistants
2. Traditional Automation
3. LM-Based Automation
4. Best of Both Worlds
5. Opportunities

So far I've assumed the **specification** already exists.

Opportunities (Part 5 of 5)

What if LMs can help people *specify* software too? This is risky, but promising.

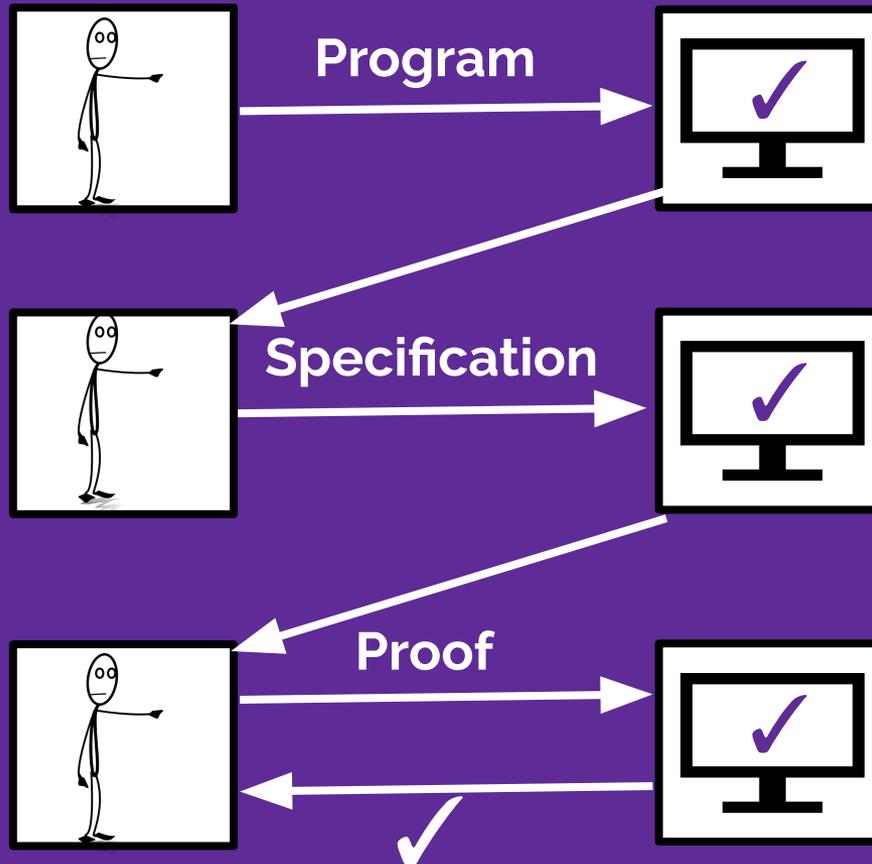
Opportunities (Part 5 of 5)

What if LMs can help people *specify* software too? This is *risky*, but *promising*.

Opportunities (Part 5 of 5)

Proof Engineer

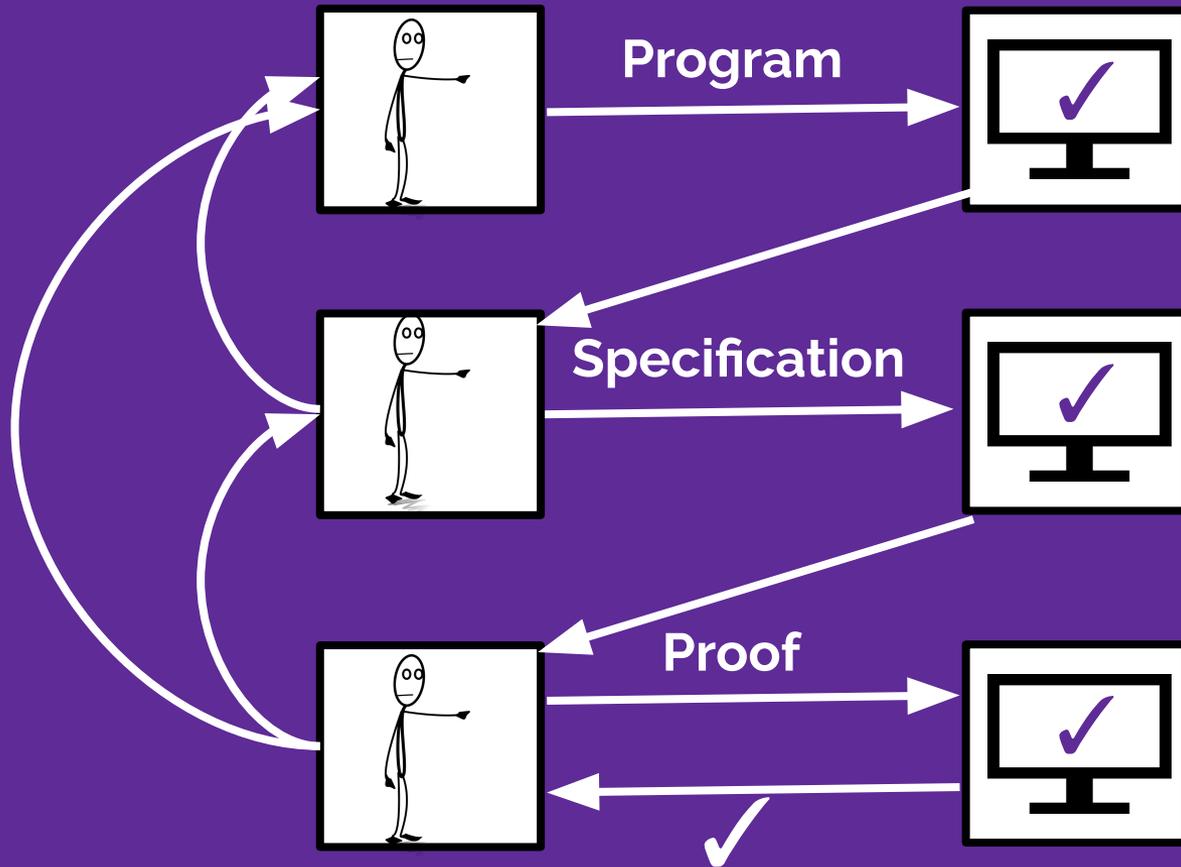
Proof Assistant



Opportunities (Part 5 of 5)

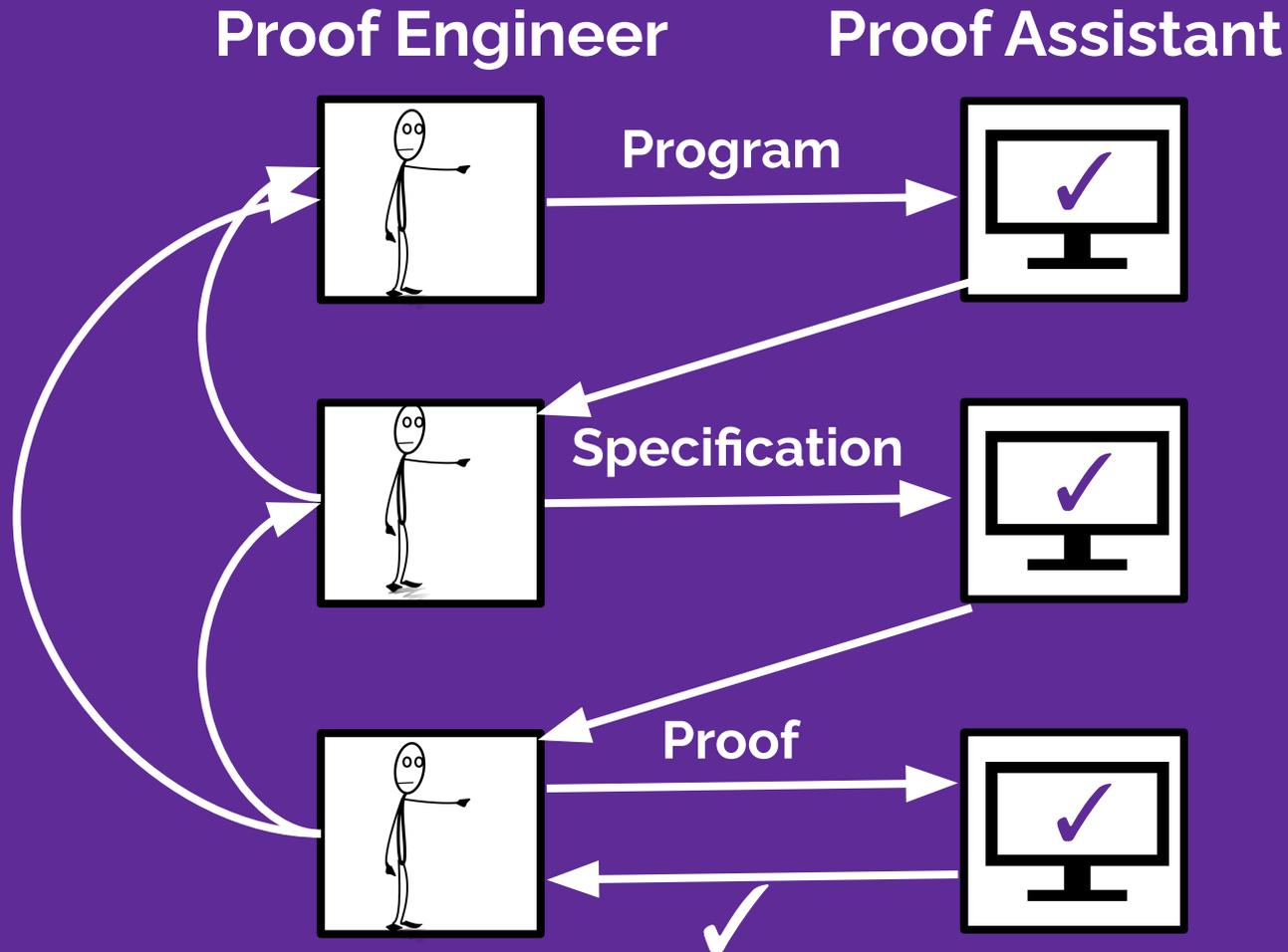
Proof Engineer

Proof Assistant



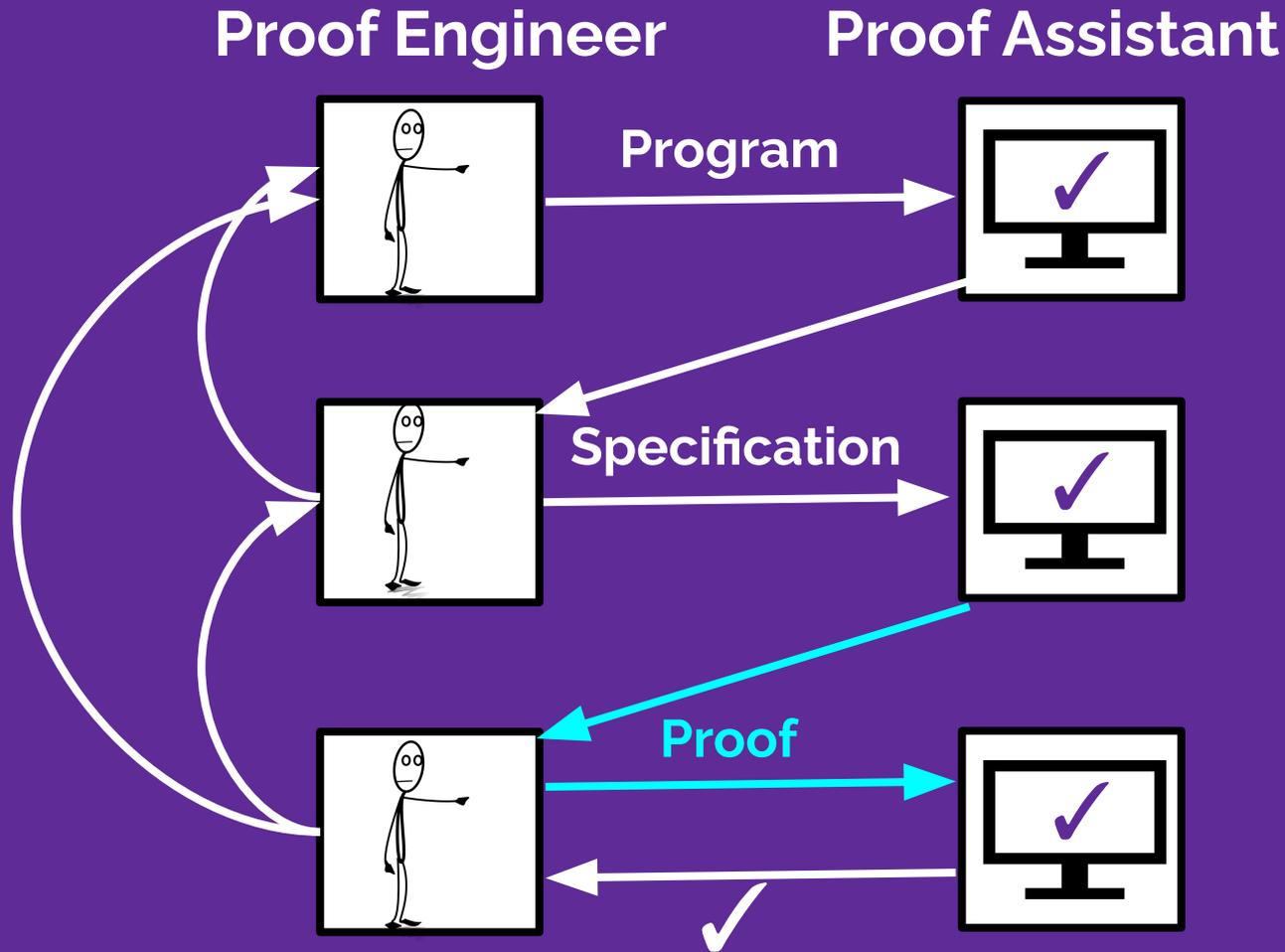
Opportunities (Part 5 of 5)

Help at Every Stage



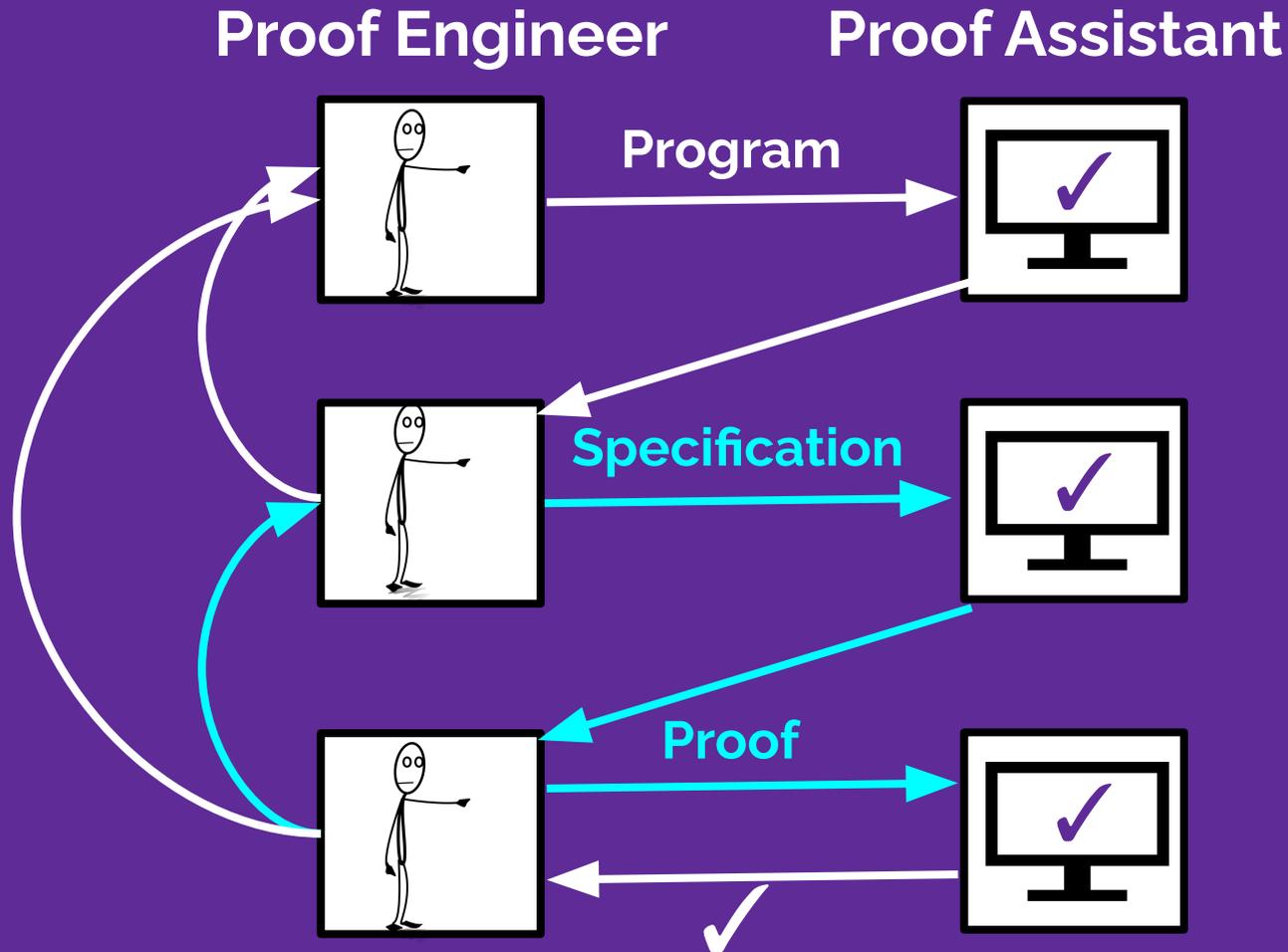
Opportunities (Part 5 of 5)

Help at Every Stage



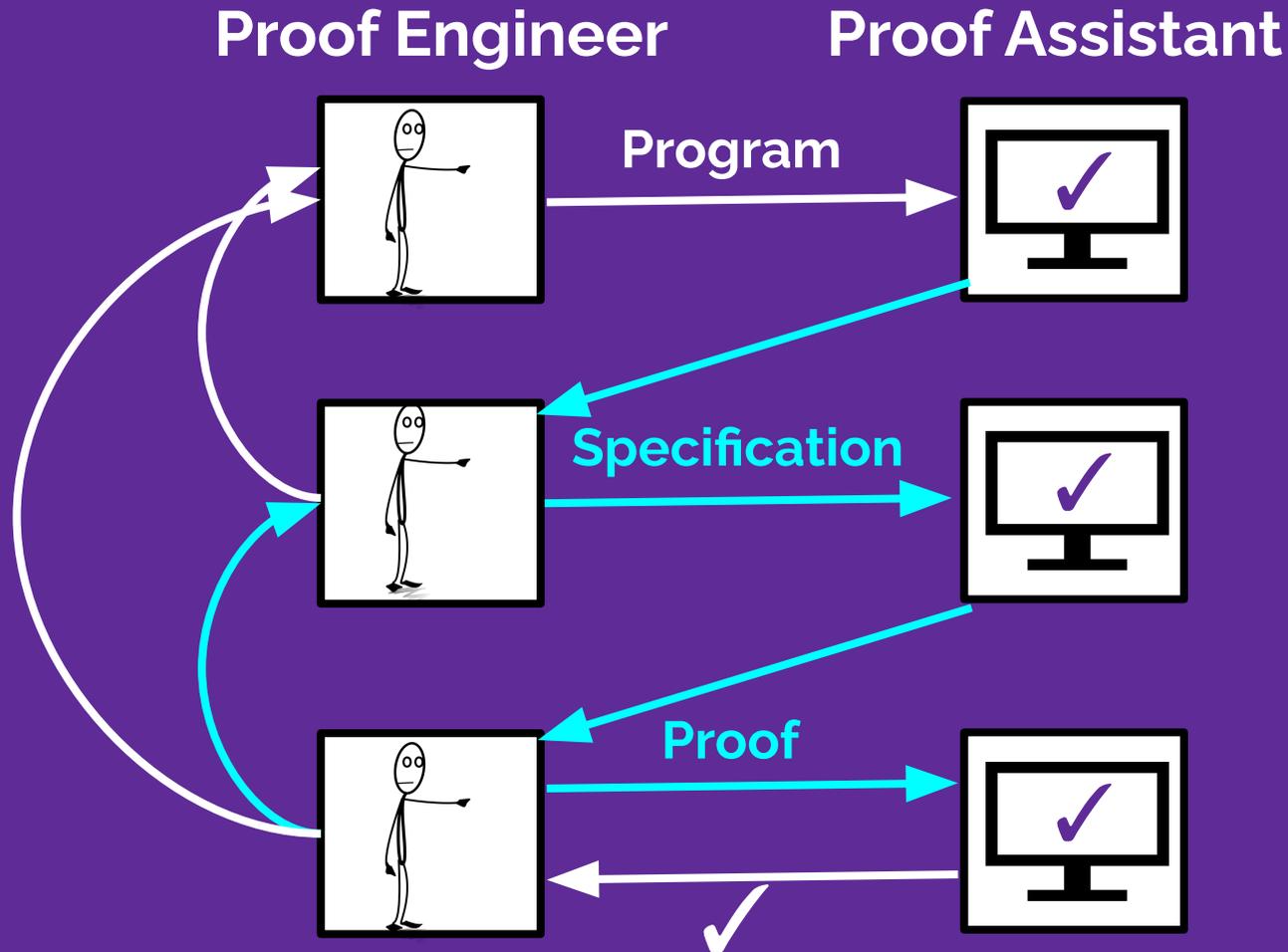
Opportunities (Part 5 of 5)

Help at Every Stage



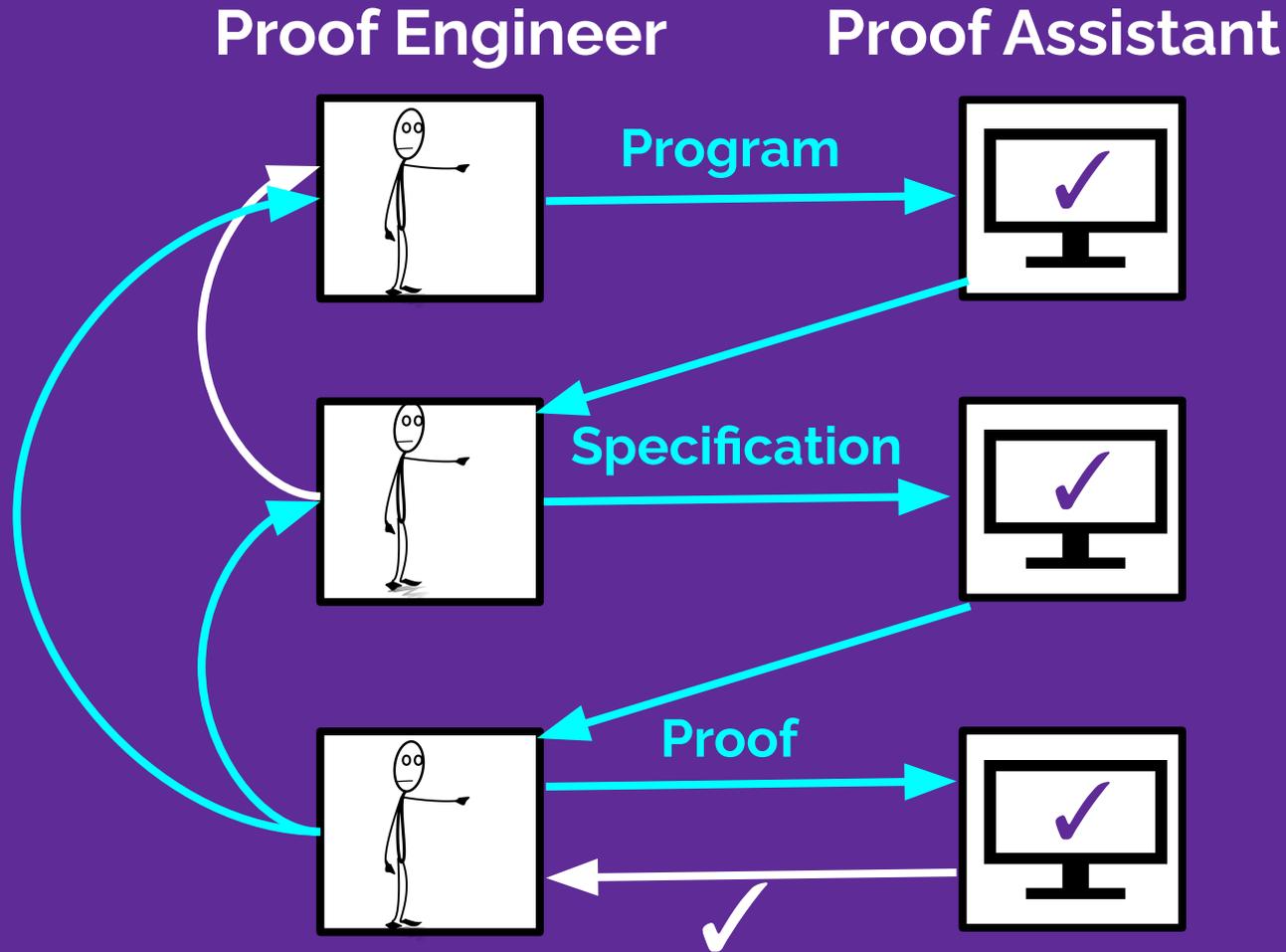
Opportunities (Part 5 of 5)

Help at Every Stage



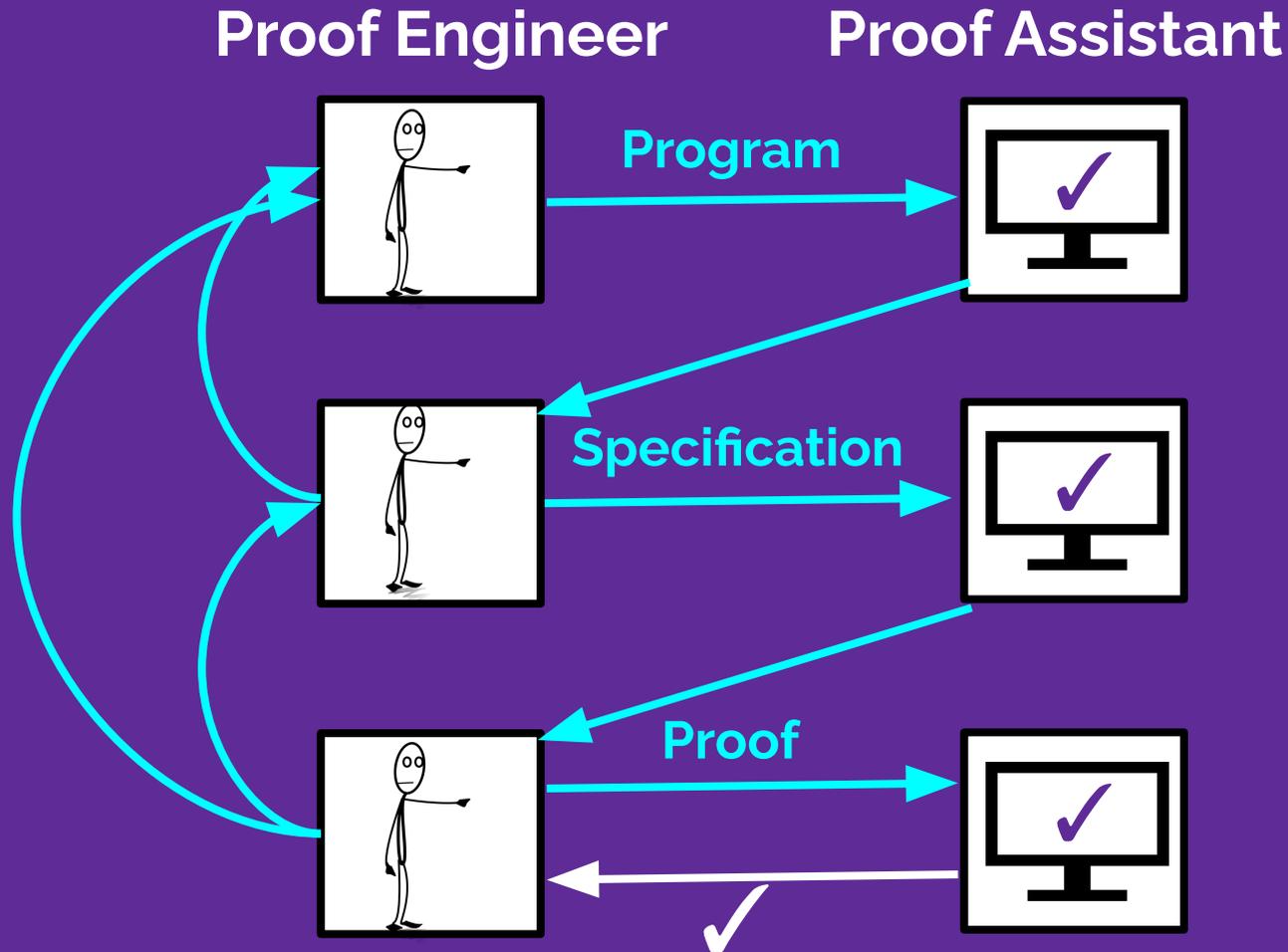
Opportunities (Part 5 of 5)

Help at Every Stage



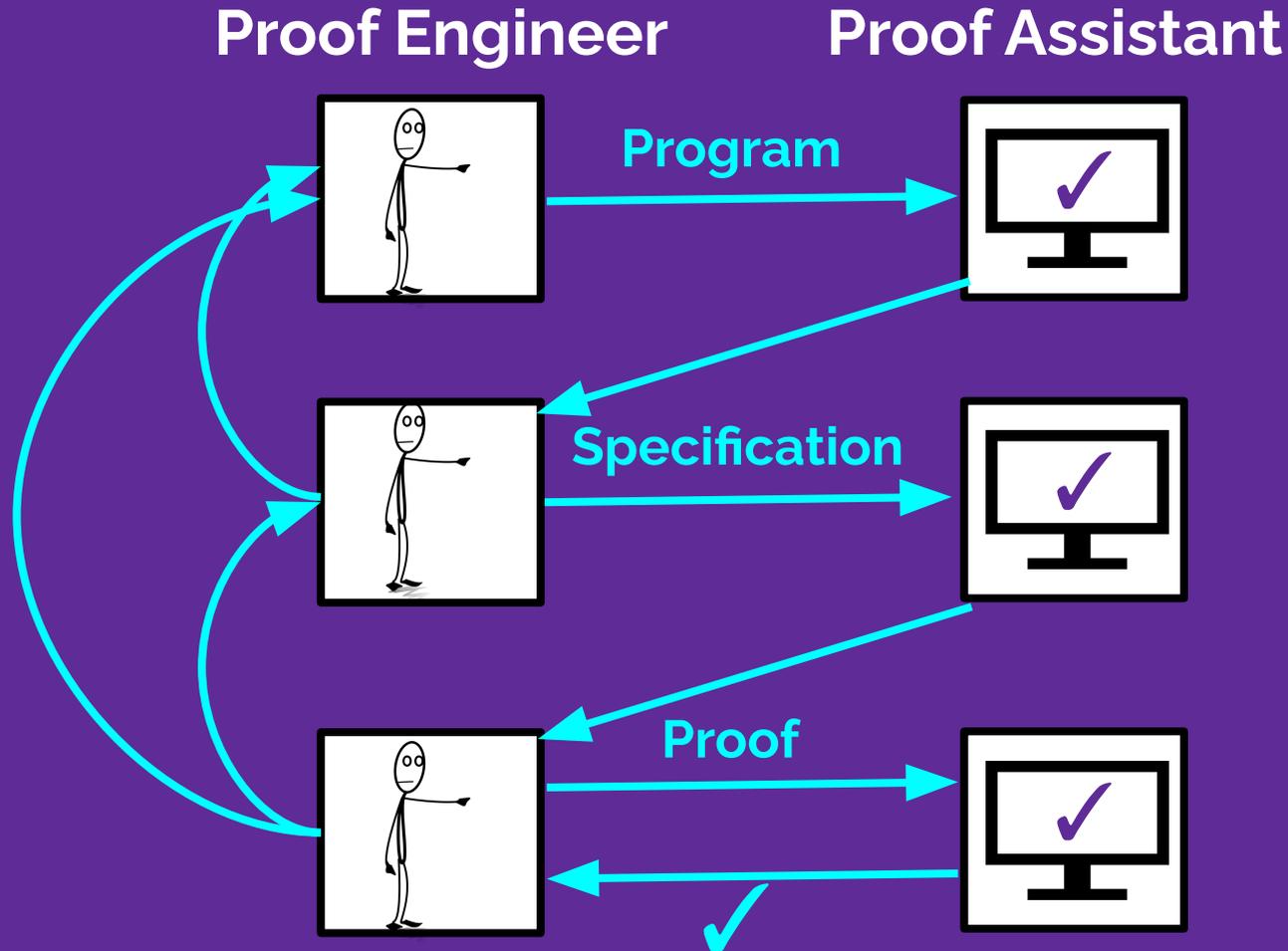
Opportunities (Part 5 of 5)

Help at Every Stage



Opportunities (Part 5 of 5)

Help at Every Stage



Opportunities (Part 5 of 5)

Help at Every Stage



With **de Bruijn**, as long as you don't touch the **kernel**, your automation is **safe**.*

(If your specification is OK, your kernel has no bugs, and you don't introduce axioms)

Opportunities (Part 5 of 5)

Help at Every Stage



With **de Bruijn**, as long as you don't touch the **kernel**, your automation is **safe**.^{*}
(If your specification is OK, your kernel has no bugs, and you don't introduce axioms)

Opportunities (Part 5 of 5)

Key Challenge:
**There is no oracle for a
specification!**

Opportunities (Part 5 of 5)

Key Challenge:

What tools can best help users make sense of generated **specifications?**

What information presented in what ways best helps users ensure that they **match their intentions?**

Opportunities (Part 5 of 5)

Key Challenge:

What tools can best help users make sense of generated **specifications**?

What information presented in what ways best helps users ensure that they **match their intentions**?