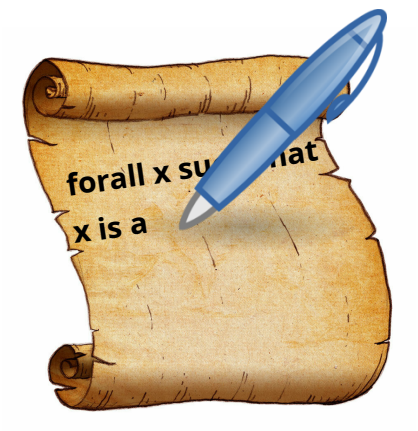
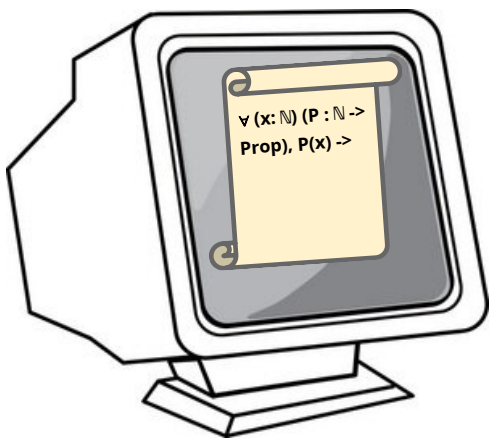
Synthesizing Proofs of Software Correctness with AI

Alex Sanchez-Stern


forall x su
hat
x is a


∀ (x: ℕ) (P : ℕ ->
Prop), P(x) ->

Obstacles

- Computer-aided proofs can make writing proofs much easier in some cases

- But they're still really hard!

```
Theorem of_intu_of_int_3:
  forall x,
  of_intu x = sub (of_int (Int.and x ox7FFF_FFFF)) (of_int (Int.and x ox8000_0000)).
Proof.
  intros.
  set (hi := Int.and x ox8000_0000).
  set (lo := Int.and x ox7FFF_FFFF).
  assert (R: forall n, integer_representable 53 1024 (Int.signed n)).
  { intros. pose proof (Int.signed_range n).
    apply integer_representable_n. smart_omega. }
  unfold sub, of_int. rewrite BofZ_minus by auto. unfold of_intu. f_equal.
  assert (E: Int.add hi lo = x).
  { unfold hi, lo. rewrite Int.add_is_or.
  - rewrite <- Int.and_or_distrib. apply Int.and_mone.
  - rewrite Int.and_assoc. rewrite (Int.and_commut ox8000_0000). rewrite Int.and_assoc.
    change (Int.and ox7FFF_FFFF ox8000_0000) with Int.zero. rewrite ! Int.and_zero; auto.
  }
  assert (RNG: 0 <= Int.unsigned lo < two_p 31).
  { unfold lo. change ox7FFF_FFFF with (Int.repr (two_p 31 - 1)). rewrite <- Int.zero_ext_and by lia.
    apply Int.zero_ext_range. compute_this Int.zwordsize. lia. }
  assert (B: forall i, 0 <= i < Int.zwordsize -> Int.testbit ox8000_0000 i = if zeq i 31 then true else false).
  { intros; unfold Int.testbit. change (Int.unsigned ox8000_0000) with (2^31).
    destruct (zeq i 31). subst i; auto. apply Z.pow2_bits_false; auto. }
  assert (EITHER: hi = Int.zero \/ hi = ox8000_0000).
  { unfold hi; destruct (Int.testbit x 31) eqn:B31; [right|left];
    Int.bit_solve; rewrite B by auto.
  - destruct (zeq i 31). subst i; rewrite B31; auto. apply andb_false_r.
  - destruct (zeq i 31). subst i; rewrite B31; auto. apply andb_false_r.
  }
  assert (SU: - Int.signed hi = Int.unsigned hi).
  { destruct EITHER as [EQ|EQ]; rewrite EQ; reflexivity. }
  unfold Z.sub; rewrite SU, <- E.
  unfold Int.add; rewrite Int.unsigned_repr, Int.signed_eq_unsigned. lia.
  - assert (Int.max_signed = two_p 31 - 1) by reflexivity. lia.
  - assert (Int.unsigned hi = 0 \/ Int.unsigned hi = two_p 31)
    by (destruct EITHER as [EQ|EQ]; rewrite EQ; [left|right]; reflexivity).
    assert (Int.max_unsigned = two_p 31 + two_p 31 - 1) by reflexivity.
    lia.
Qed.
```

AI and Computer Proofs: A Match Made in Heaven

- When AI writes paper proofs, there are trust issues

- But when it writes computer proofs, we can mechanistically check them



- Untrusted proof search + trusted verifier = ❤️

These Techniques ≠ AI Mathematicians

The systems in this talk don't generate insights or conjectures

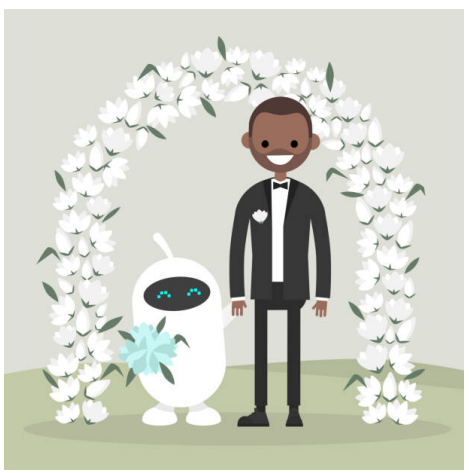
They just empower mathematicians

There are many ways to marry AI and proofs

In this talk we'll go over just a few of them

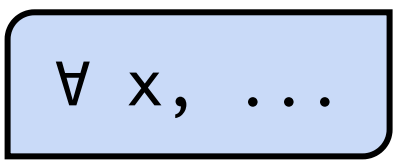
- In particular, not many LLMs!
- And not even all AI!

But first...



What does it mean to prove software correct?

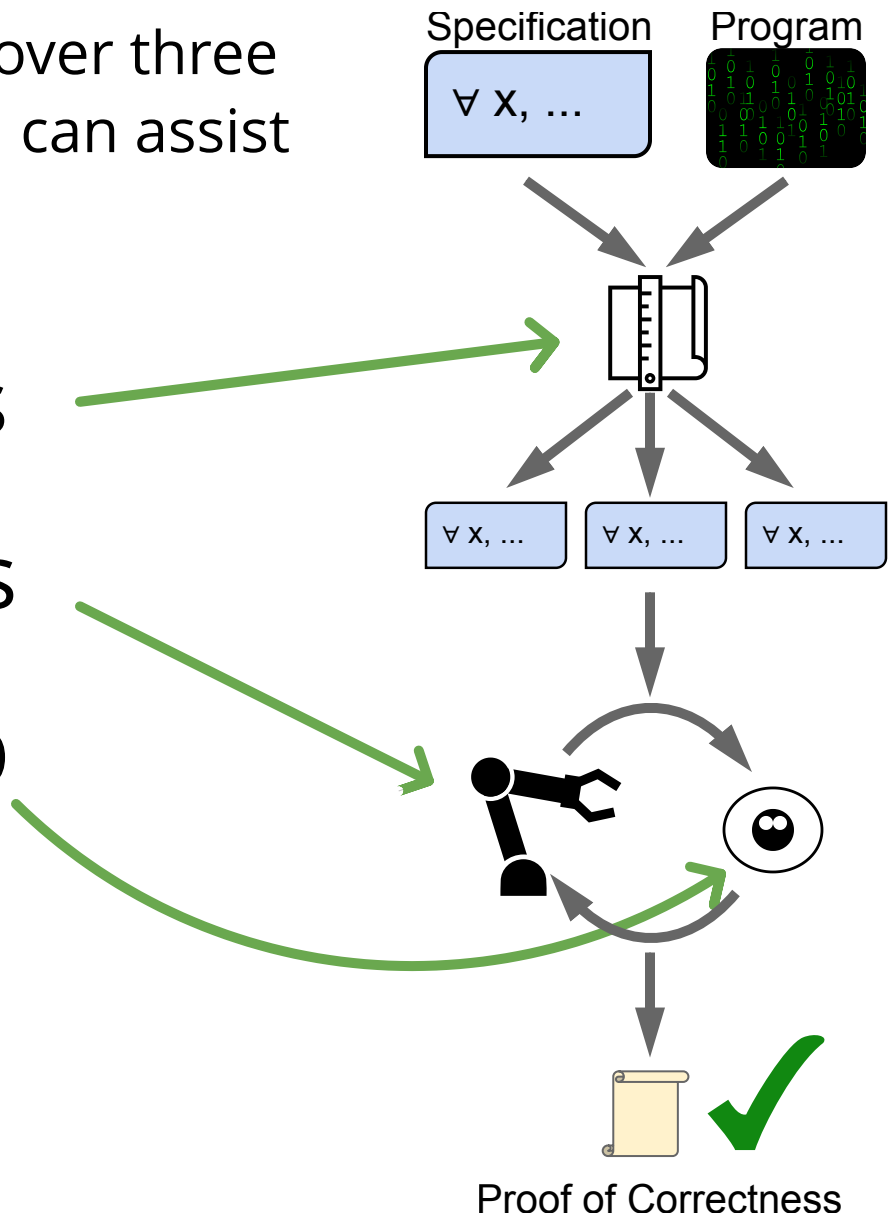
Specifications!



(mathematical statements describing the behavior of the program on all valid inputs)

For the rest of this talk we'll cover three stages of proof writing that AI can assist

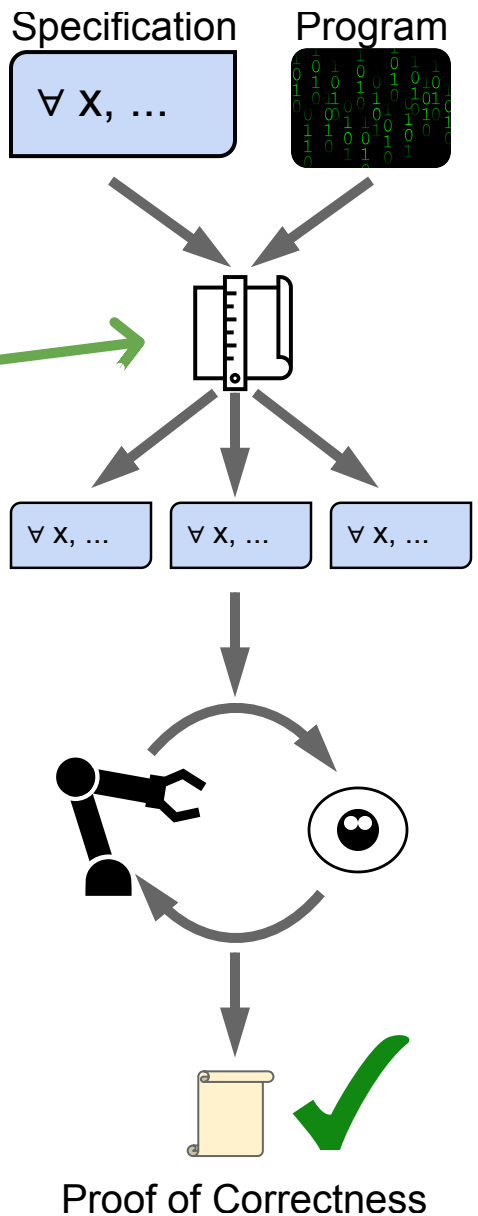
1. Structuring Proofs
2. Synthesizing Steps
3. Searching for QED



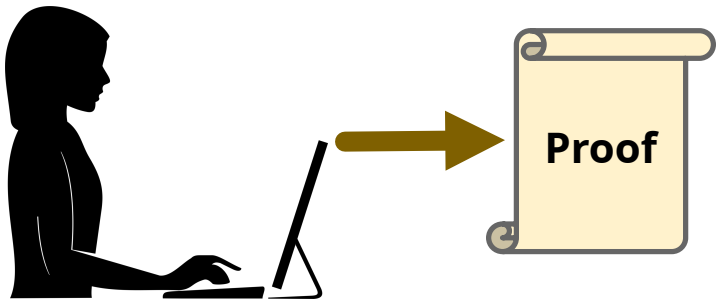
1. Structuring Proofs

2. Synthesizing Steps

3. Searching for QED



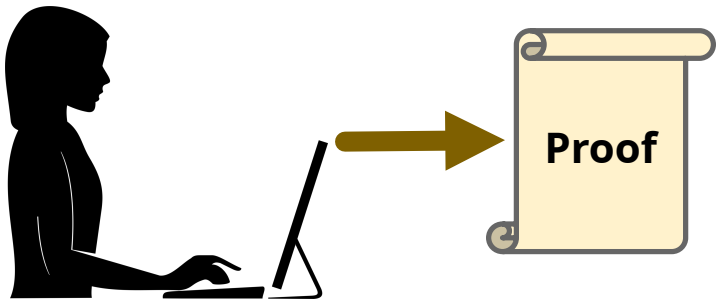
Specification




```
forall (L: List T),
  sorted(isort(L)) AND
    length(isort(L)) = length(L) AND
      elements(isort(L)) = elements(L)
```

Like other proofs, computer proofs need **decomposition**

Specification

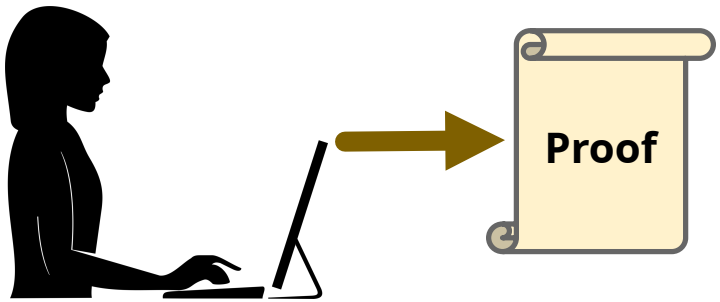



```
forall (L: List T),
  sorted(isort(L)) AND
    length(isort(L)) = length(L) AND
      elements(isort(L)) = elements(L)
```

Like other proofs, computer proofs need **decomposition**

```
forall (L: List T) (x: T),
  length([x] + L) = length(L) + 1
```

Specification




```
forall (L: List T),
  sorted(isort(L)) AND
    length(isort(L)) = length(L) AND
      elements(isort(L)) = elements(L)
```

Like other proofs, computer proofs need **decomposition**

Helper Lemmas

```
forall (L: List T) (x: T),
  length([x] + L) = length(L) + 1
```

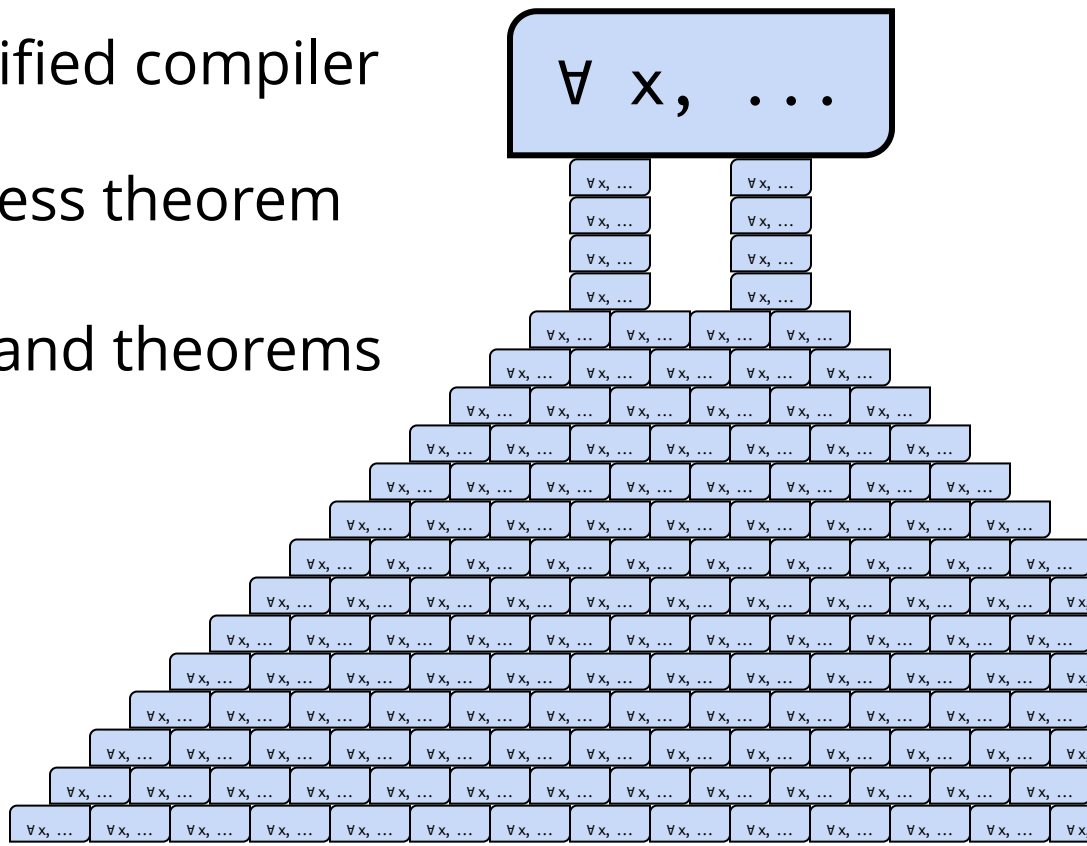
```
forall (L: List T) (x: T),
  sorted(L) ->
    sorted(sinsert(x, L))
```

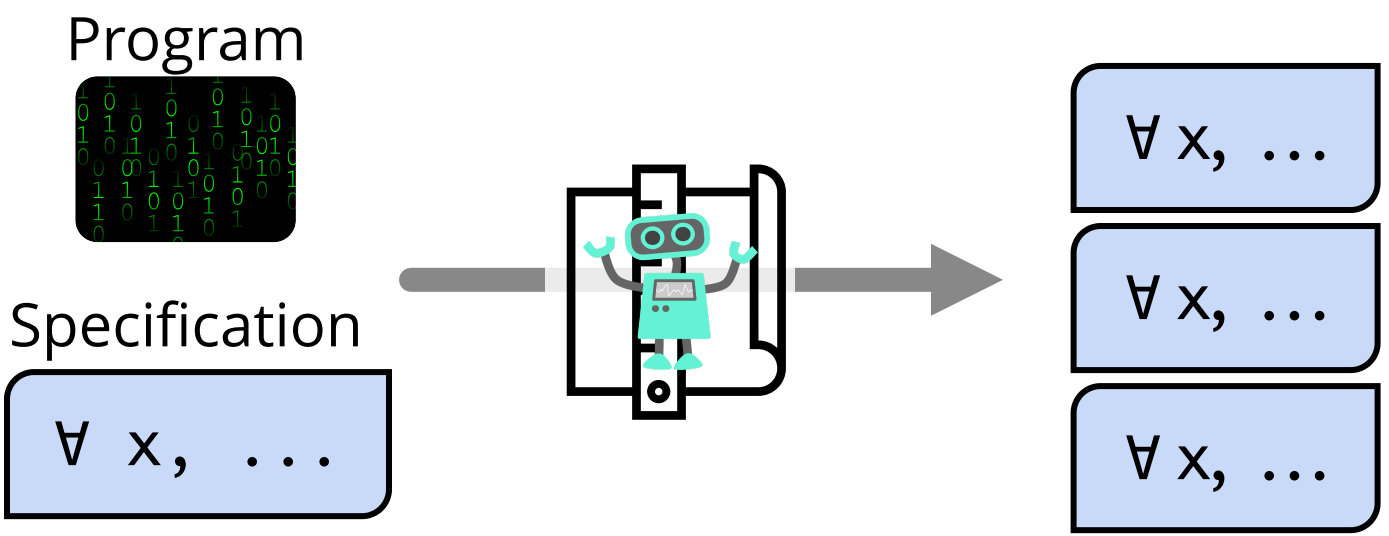
Helper Lemmas Are Everywhere

In the CompCert verified compiler

~1 top-level correctness theorem

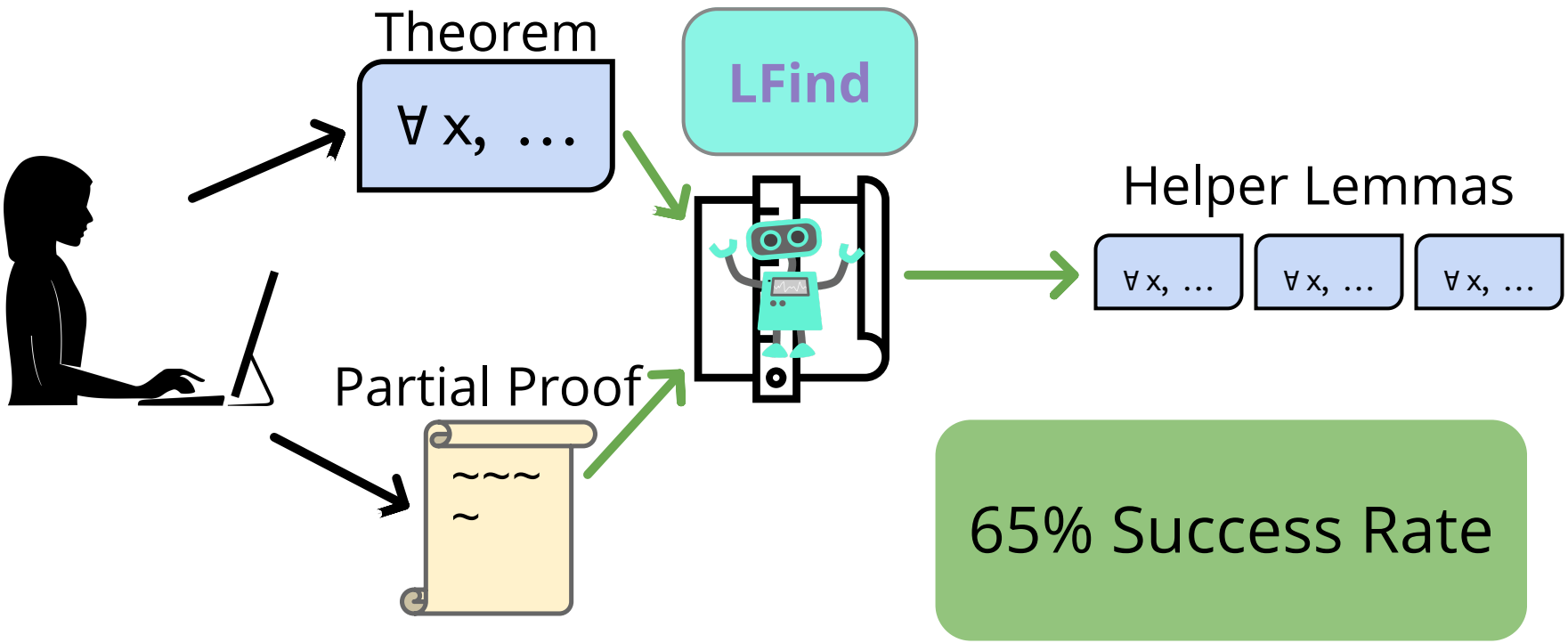
6257 other lemmas and theorems



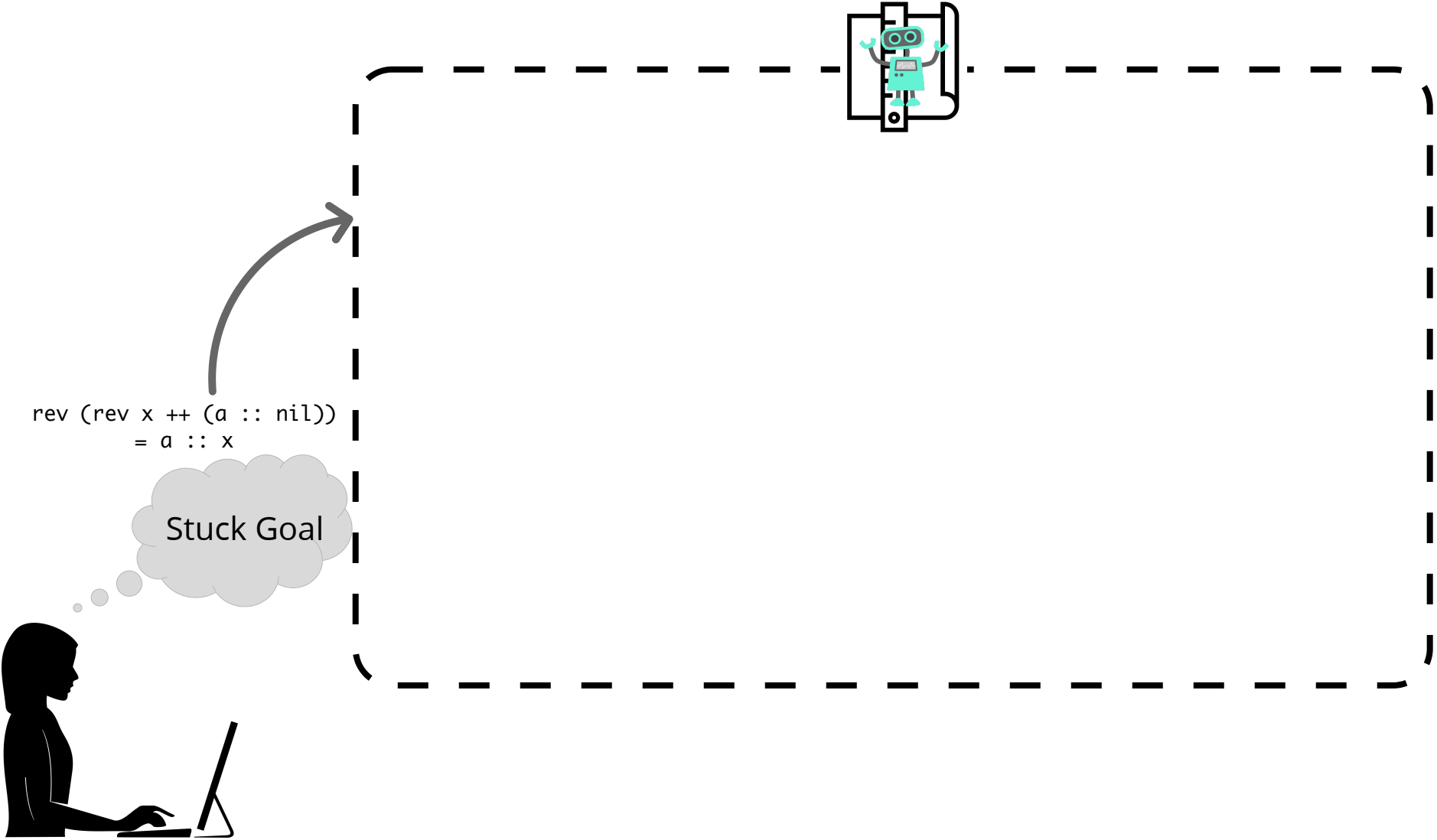


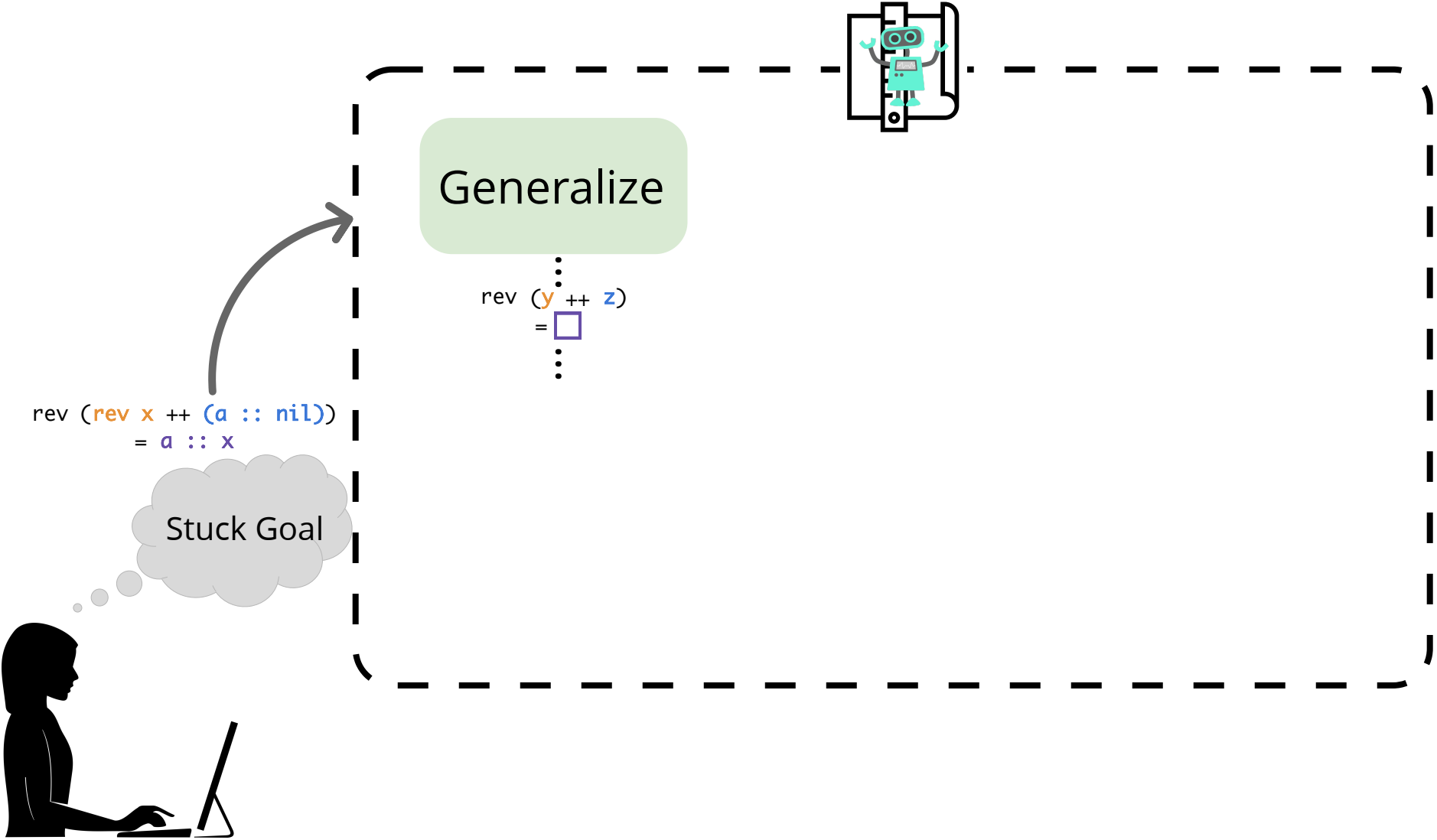
Lemma Synthesis

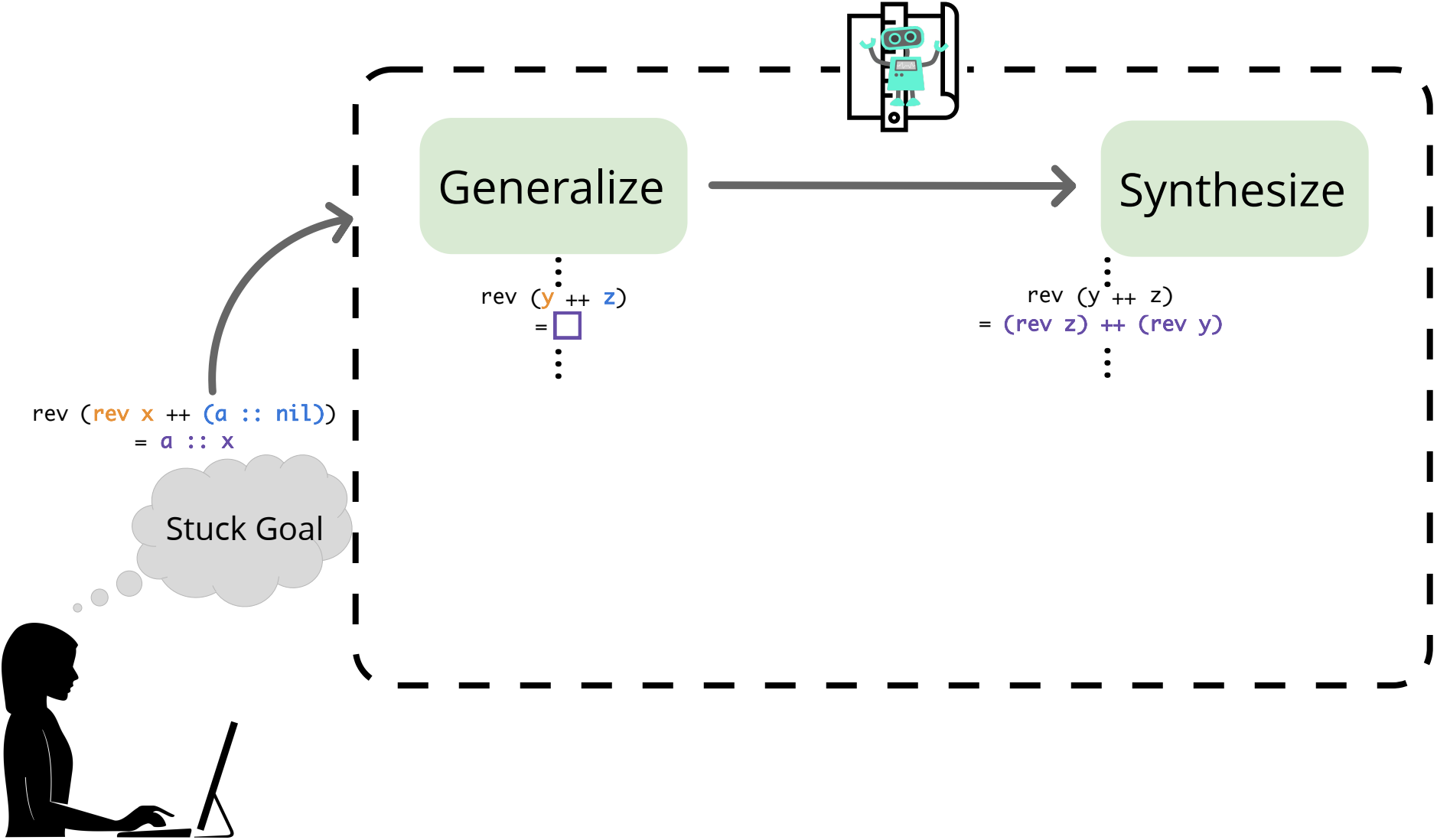
Lemma Synthesis - LFind

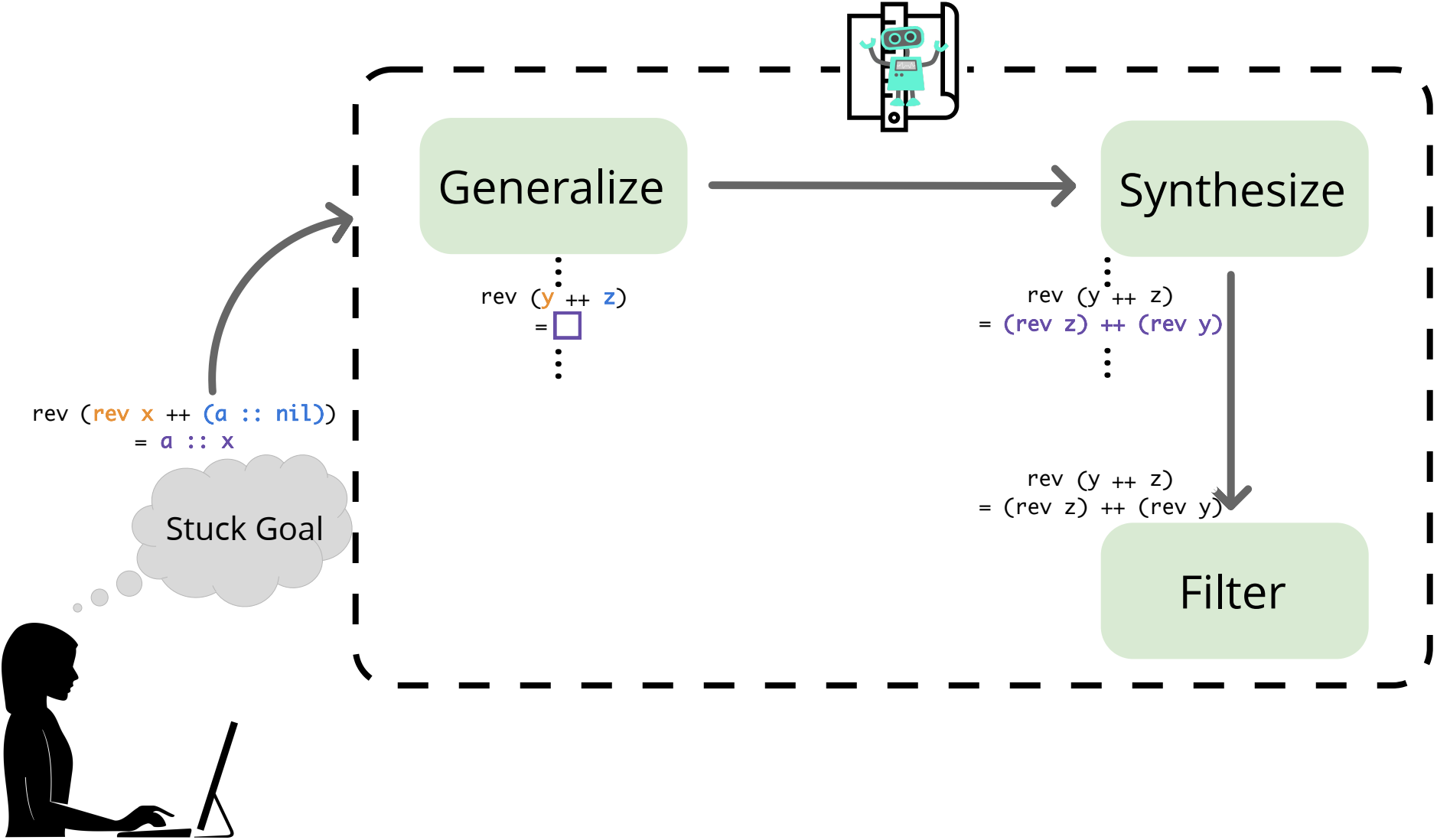


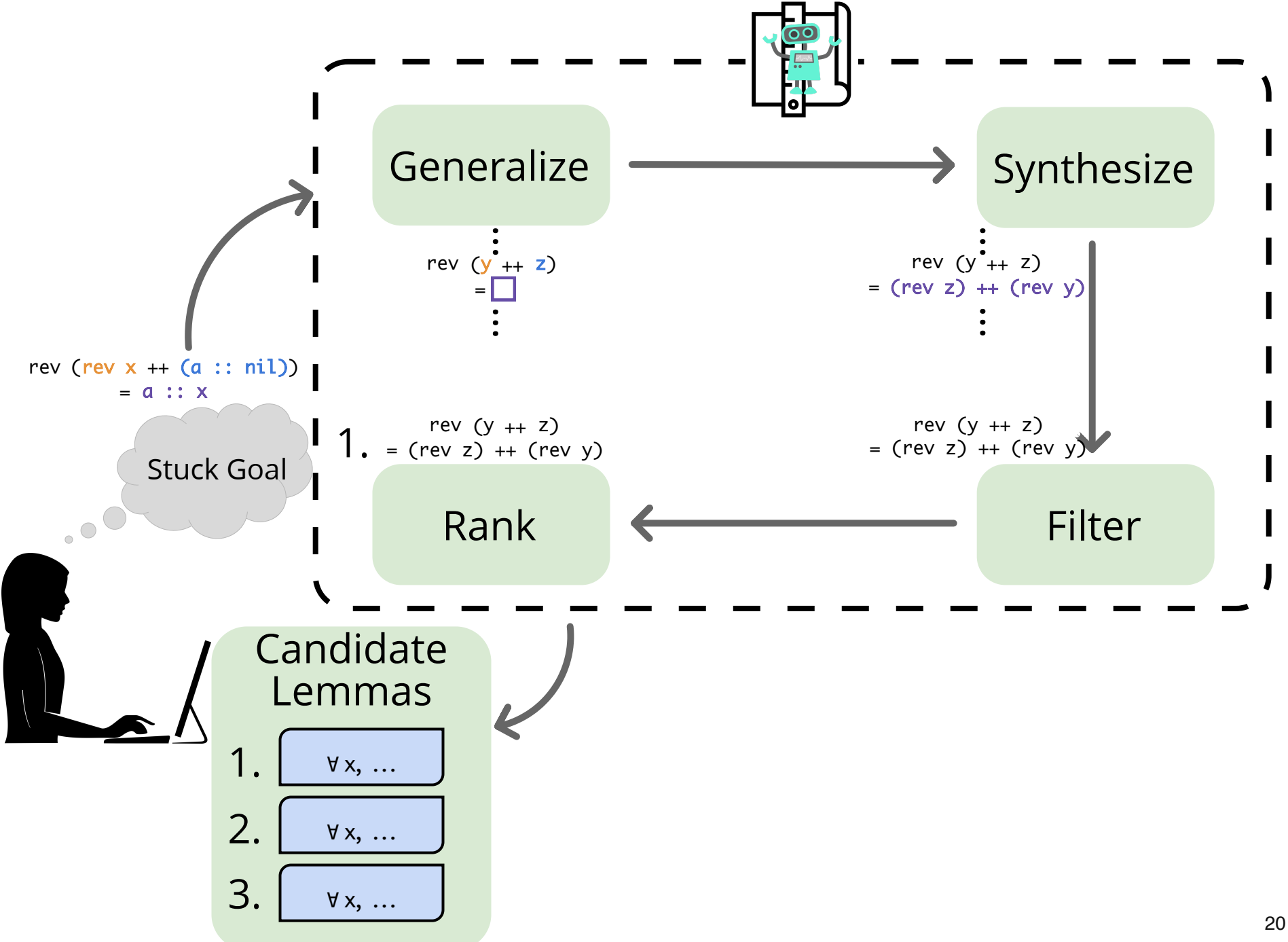
[1] Sivaraman, Aishwarya, Alex, Sanchez-Stern, Bretton, Chen, Sorin, Lerner, and Millstein Todd. "Data-Driven Lemma Synthesis for Interactive Proofs." . In *Object-Oriented Programming, Systems, Languages & Applications*. ACM SIGPLAN, 2022


rev (rev x ++ (a :: nil))
= a :: x
Stuck Goal


Generalize
rev (y ++ z)
= □
rev (rev x ++ (a :: nil))
= a :: x
Stuck Goal


Generalize
Synthesize
rev (y ++ z)
= □
rev (y ++ z)
= (rev z) ++ (rev y)
rev (rev x ++ (a :: nil))
= a :: x
Stuck Goal

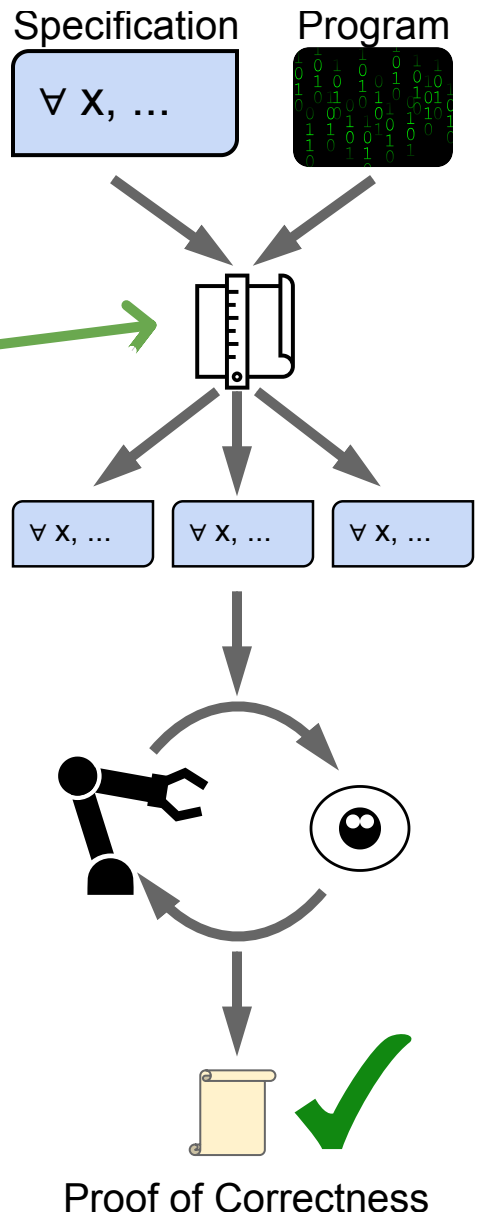

Generalize
Synthesize
rev (y ++ z)
= □
rev (y ++ z)
= (rev z) ++ (rev y)
rev (y ++ z)
= (rev z) ++ (rev y)
Filter
rev (rev x ++ (a :: nil))
= a :: x
Stuck Goal


Generalize
Synthesize
rev (y ++ z)
=
rev (y ++ z)
= (rev z) ++ (rev y)
rev (rev x ++ (a :: nil))
= a :: x
Stuck Goal
1. rev (y ++ z)
= (rev z) ++ (rev y)
rev (y ++ z)
= (rev z) ++ (rev y)
Rank
Filter
Candidate Lemmas
1. ∀ x, …
2. ∀ x, …
3. ∀ x, …

1. Structuring Proofs

2. Synthesizing Steps

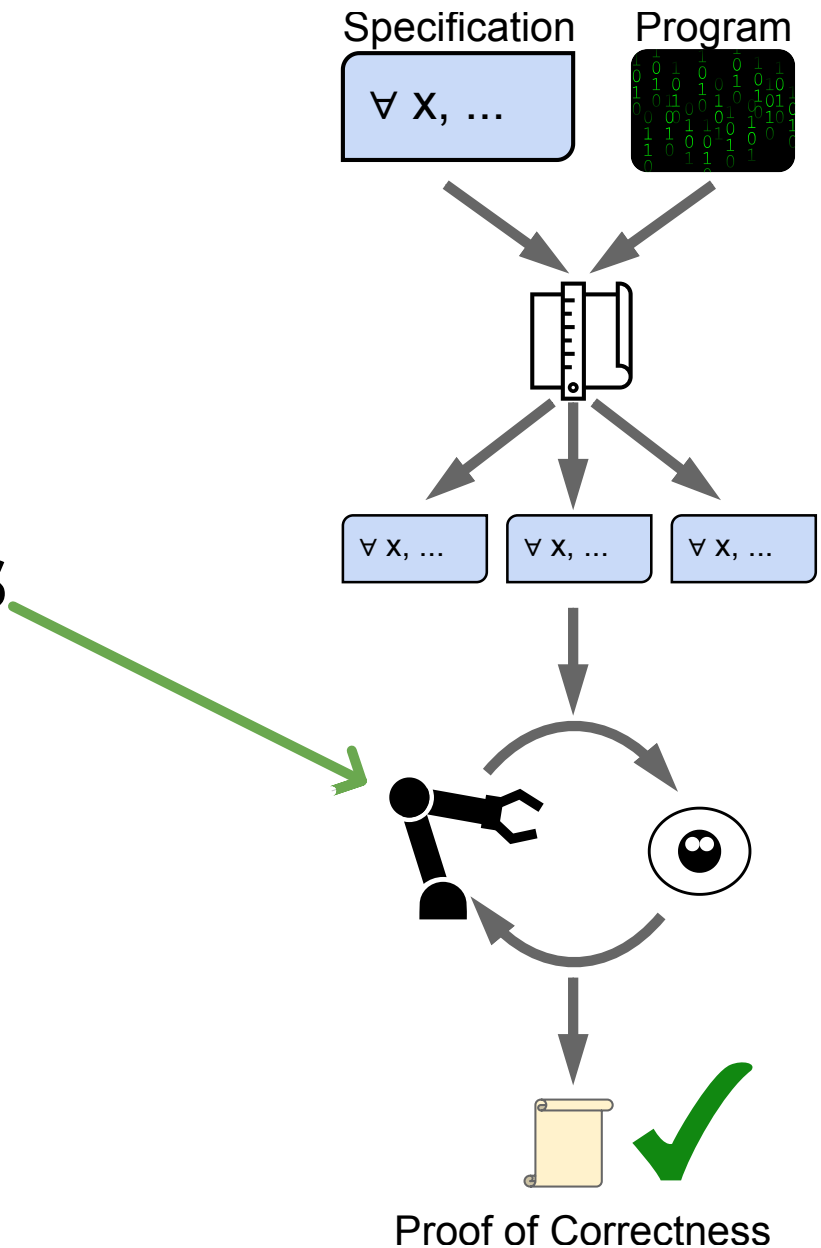
3. Searching for QED

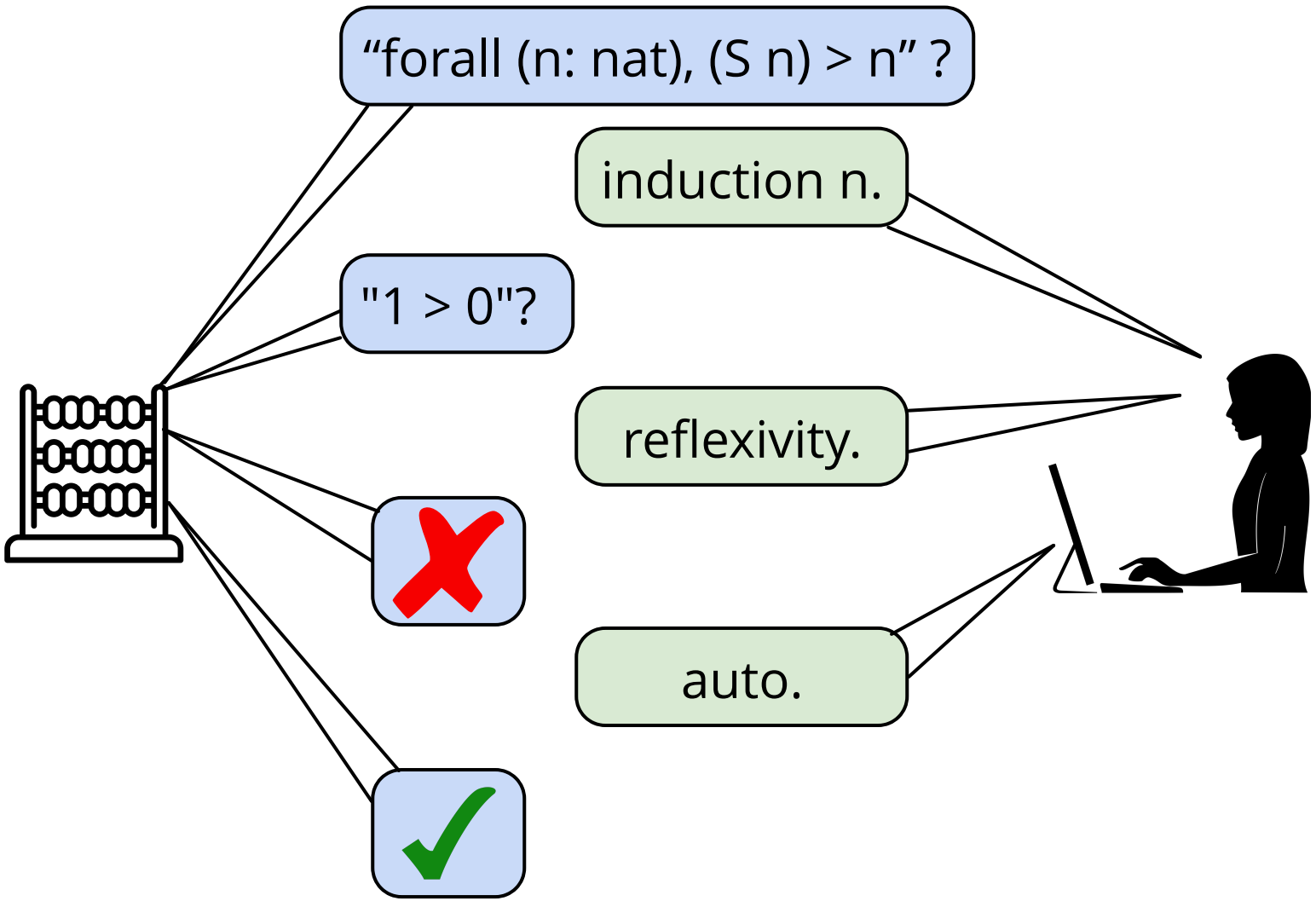


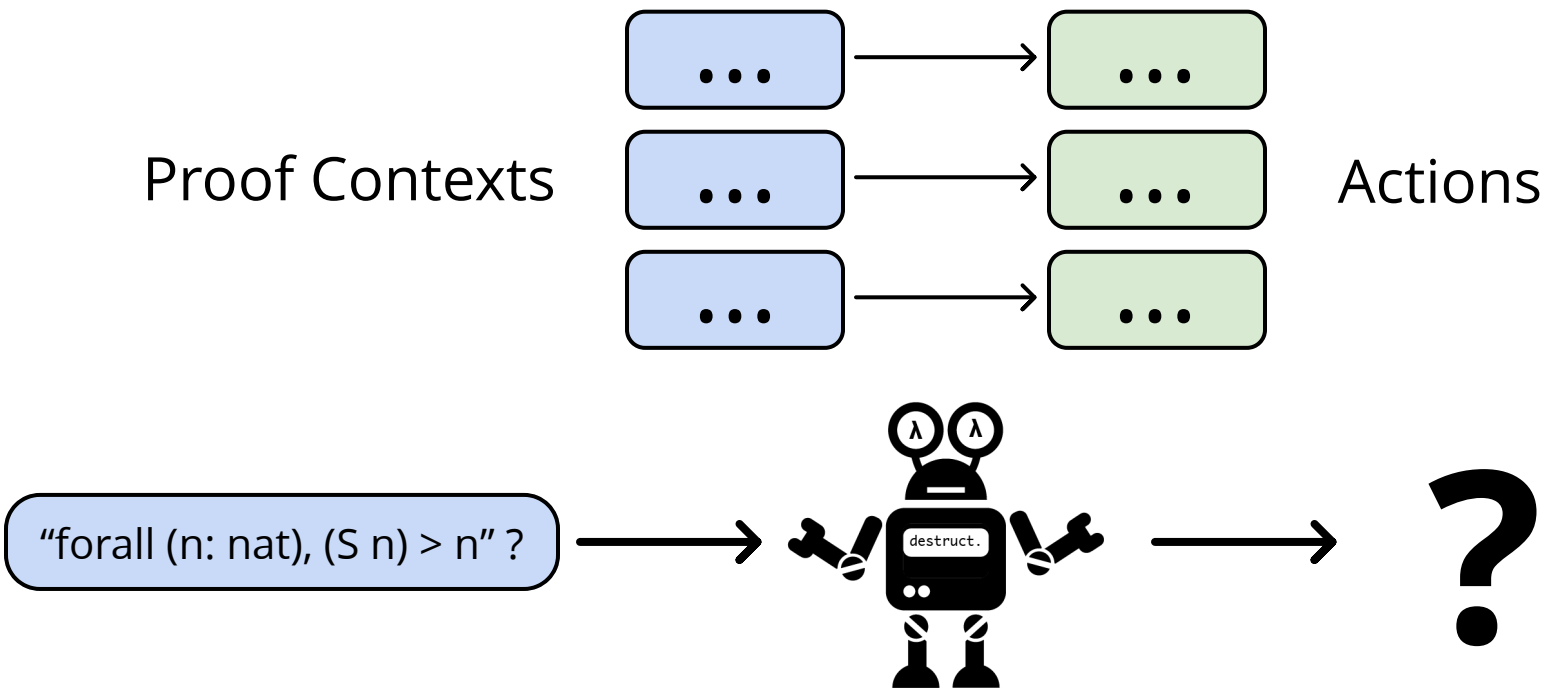
1. Structuring Proofs

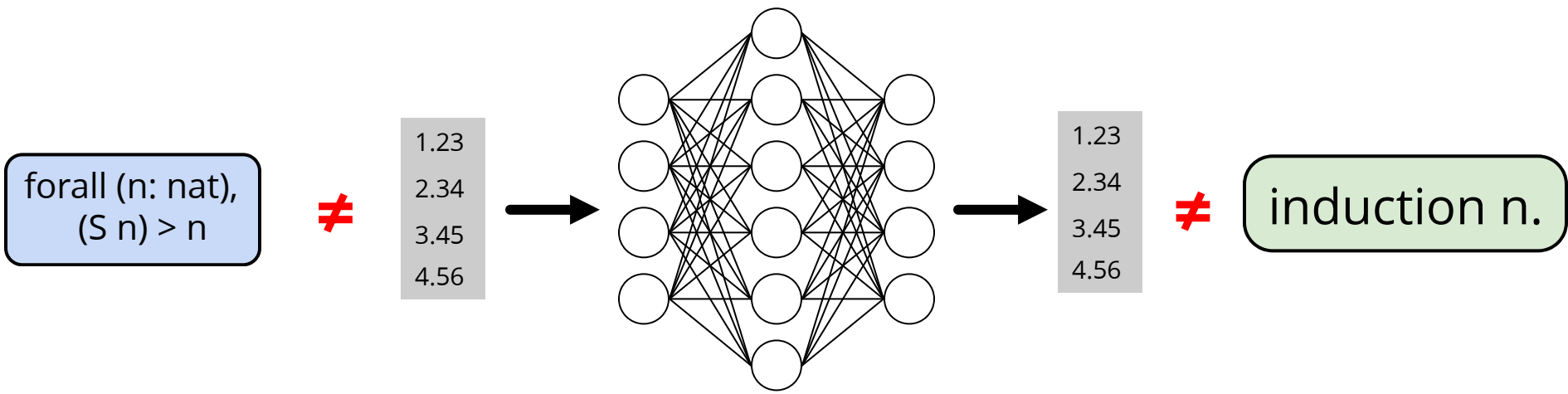
2. Synthesizing Steps

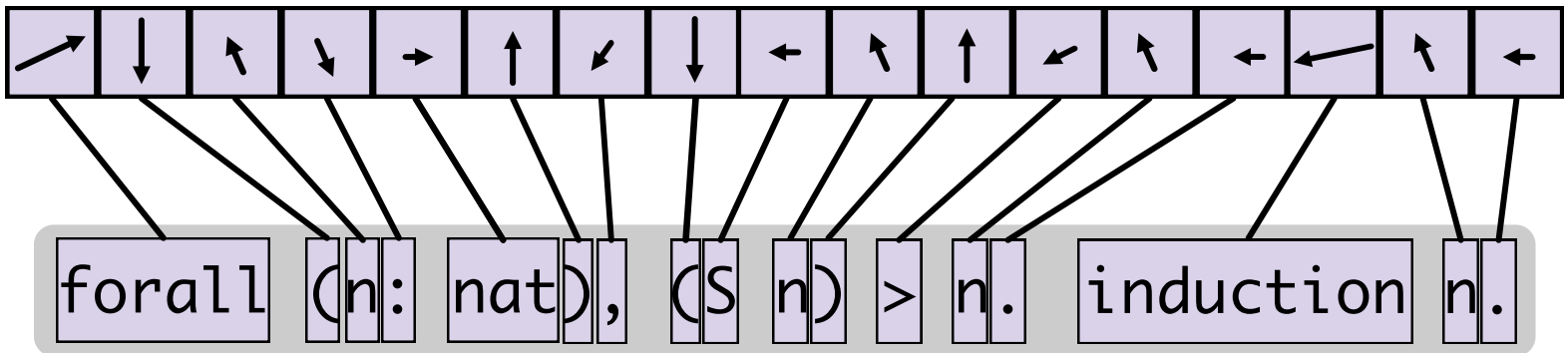
3. Searching for QED



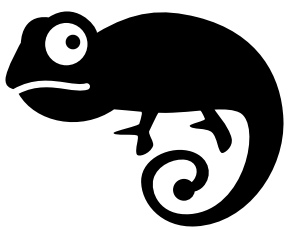

“forall (n: nat), (S n) > n” ?
induction n.
"1 > 0"?
reflexivity.
auto.


Proof Contexts
Actions
“forall (n: nat), (S n) > n” ?
destruct.
?


forall (n: nat),
(S n) > n
≠
1.23
2.34
3.45
4.56
1.23
2.34
3.45
4.56
≠
induction n.

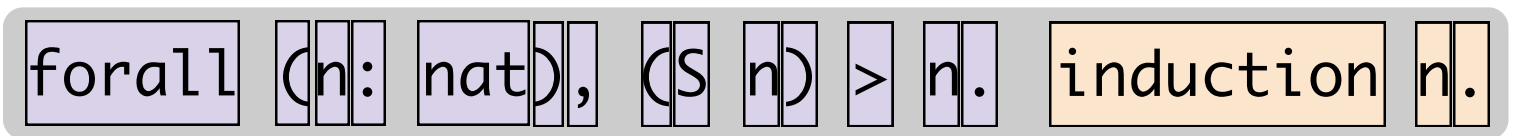


Rango



Kyle Thompson, Nuno Saavedra, Pedro Carrott, Kevin Fisher, Alex Sanchez-Stern, Yuriy Brun, Joao F Ferreira, Sorin Lerner, Emily First. To appear in ICSE 2025

Input (Proof Context) Output (Proof Step)

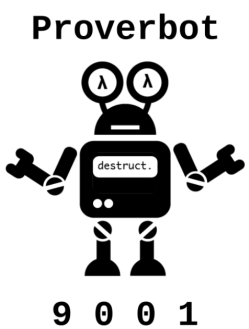


Input (Proof Context)

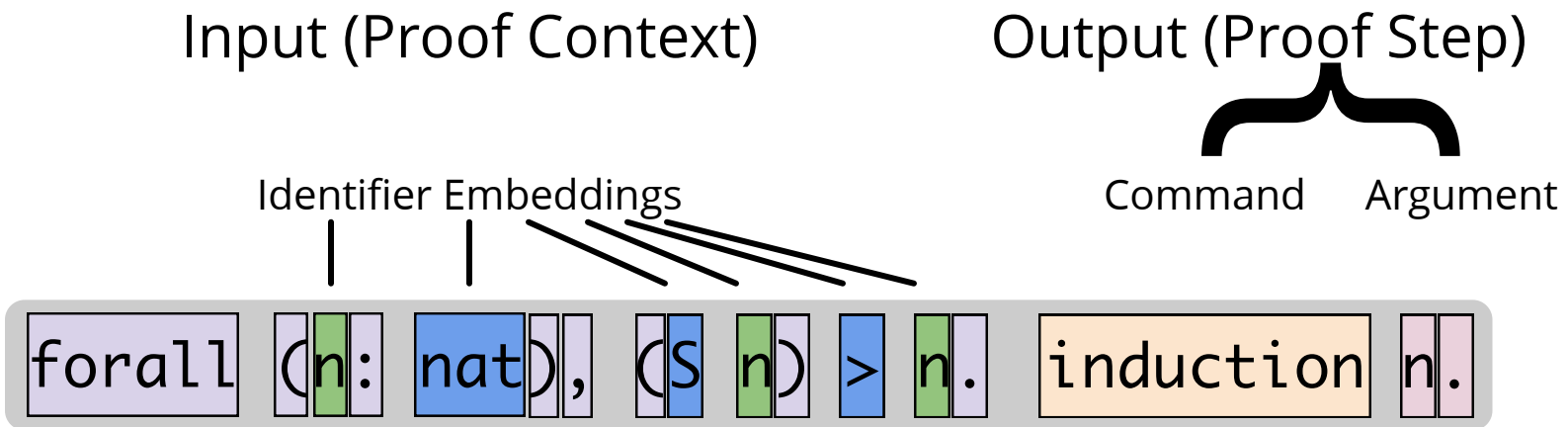
Output (Proof Step)

Command Argument

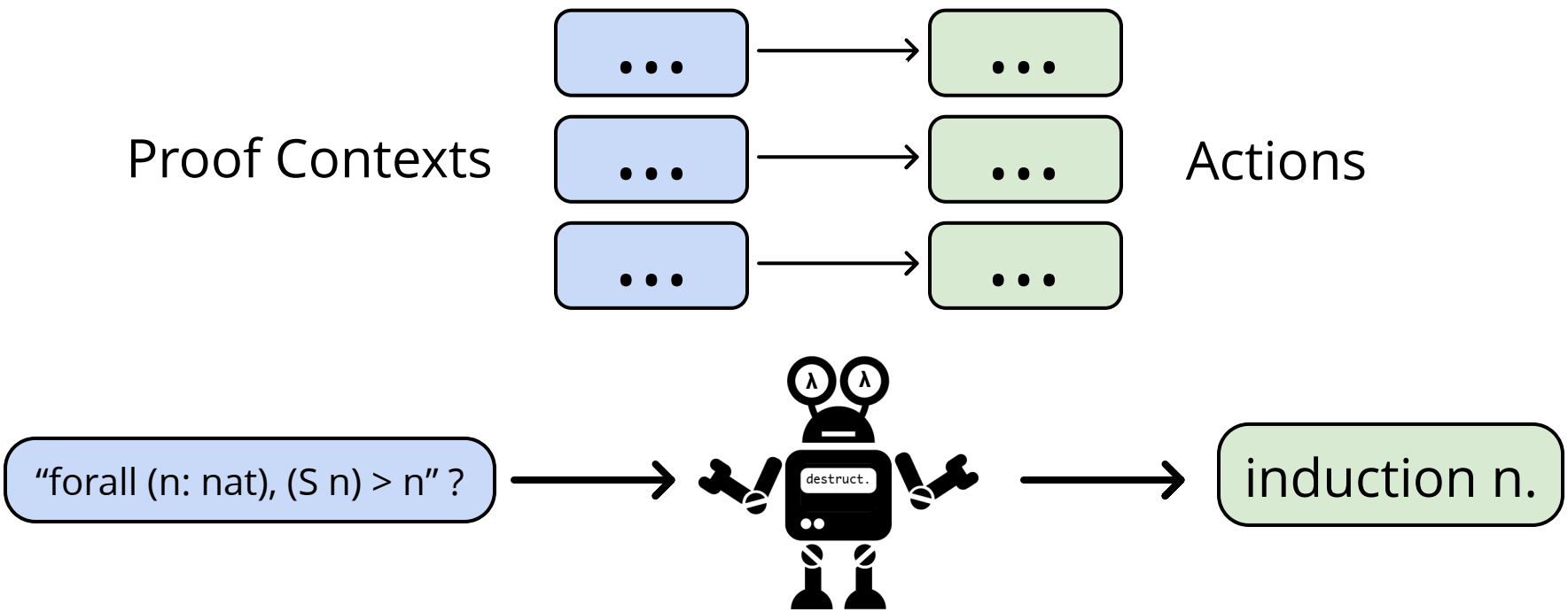
```
forall (n: nat), (S n) > n. induction n.
```



[1] Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. "Generating Correctness Proofs with Neural Networks." . In *Machine Learning in Programming Languages*. ACM SIGPLAN, 2020.



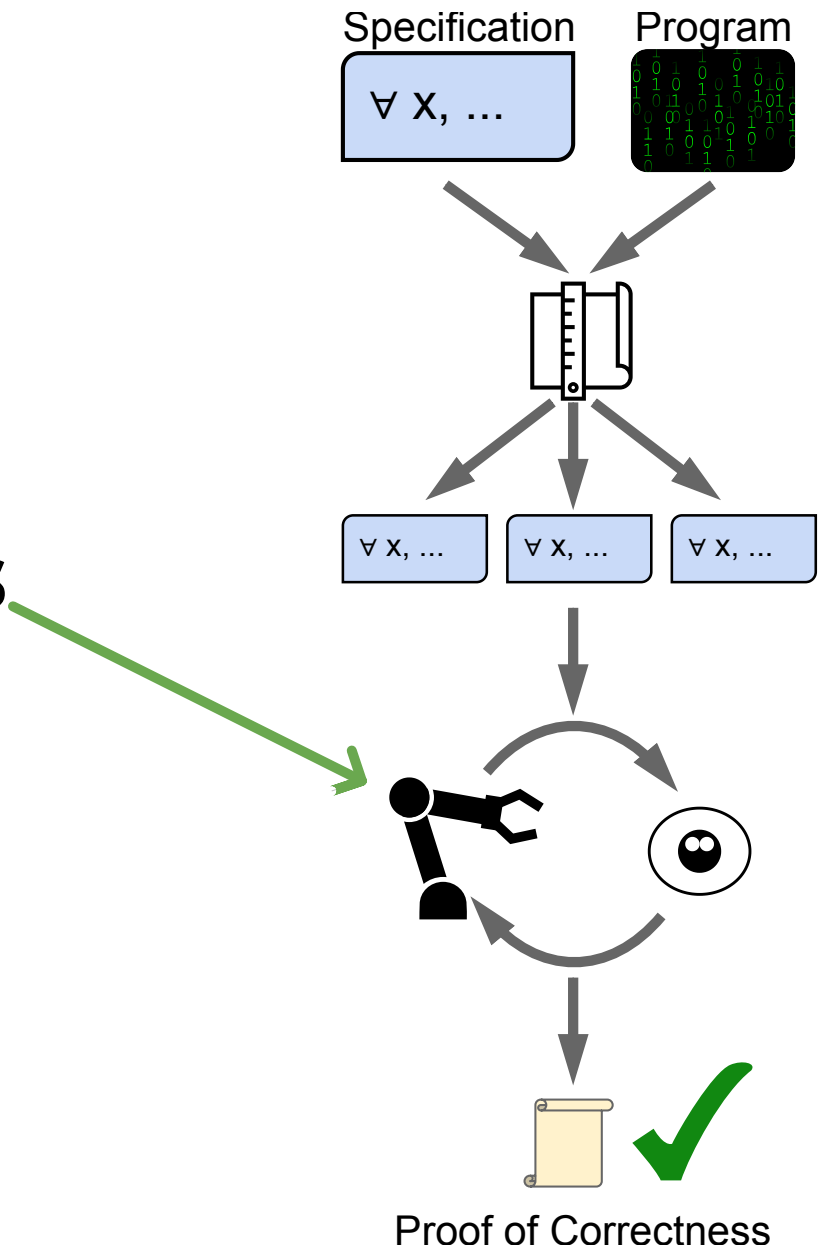
[1] Alex Sanchez-Stern, Emily First, Timothy Zhou, Zhanna Kaufman, Yuriy Brun, and Talia Ringer. "Passport: Improving Automated Formal Verification Using Identifiers." (2022).


Proof Contexts
...
...
...
...
...
...
Actions
“forall (n: nat), (S n) > n” ?
destruct.
induction n.

1. Structuring Proofs

2. Synthesizing Steps

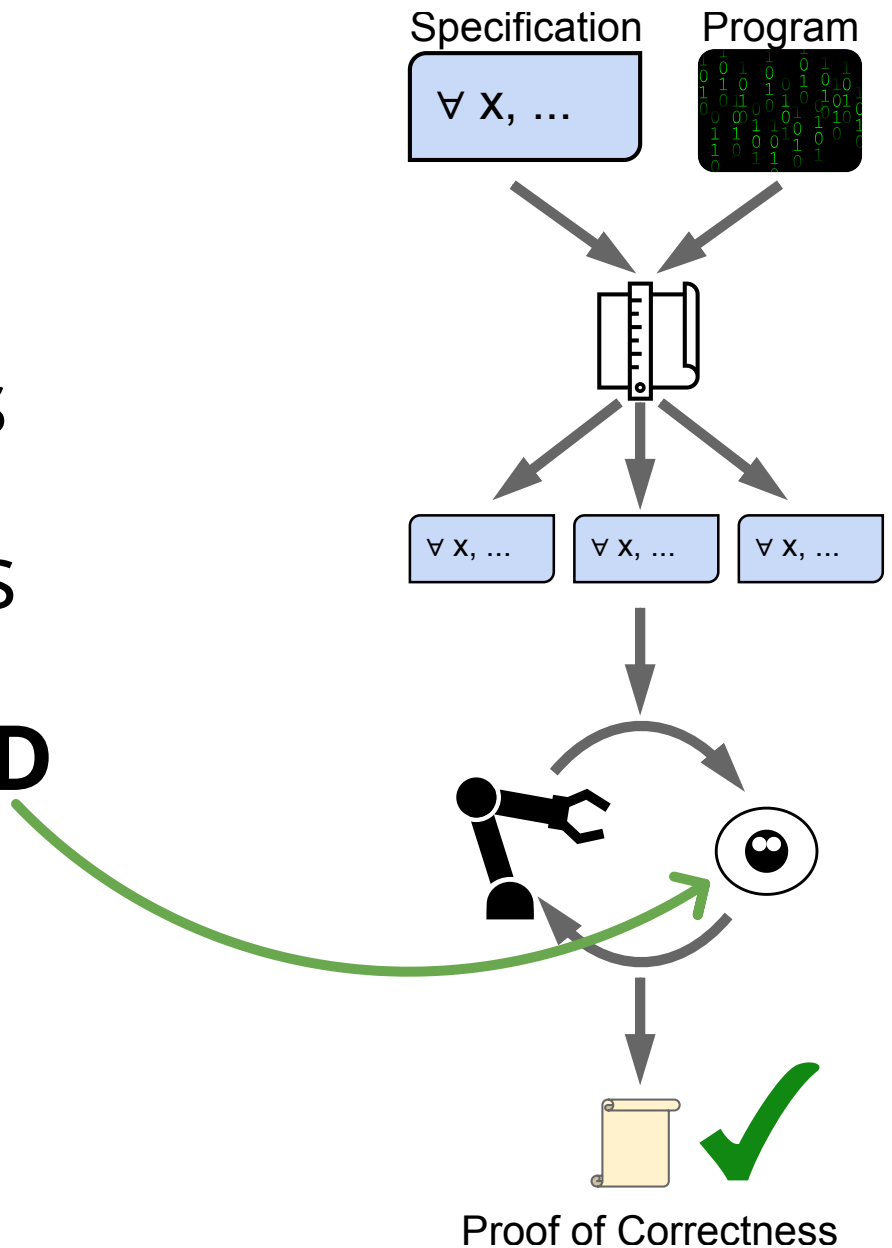
3. Searching for QED



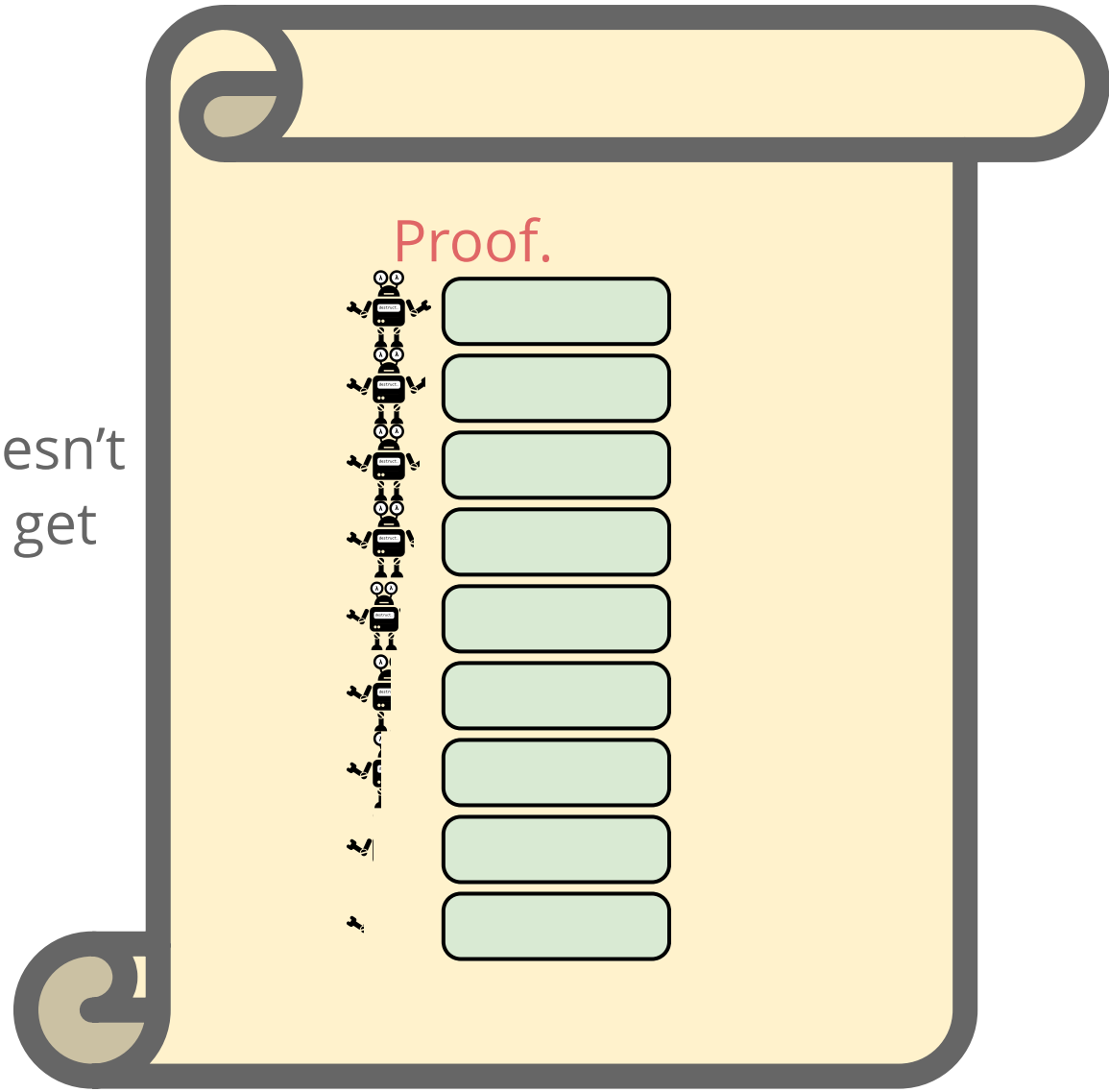
1. Structuring Proofs

2. Synthesizing Steps

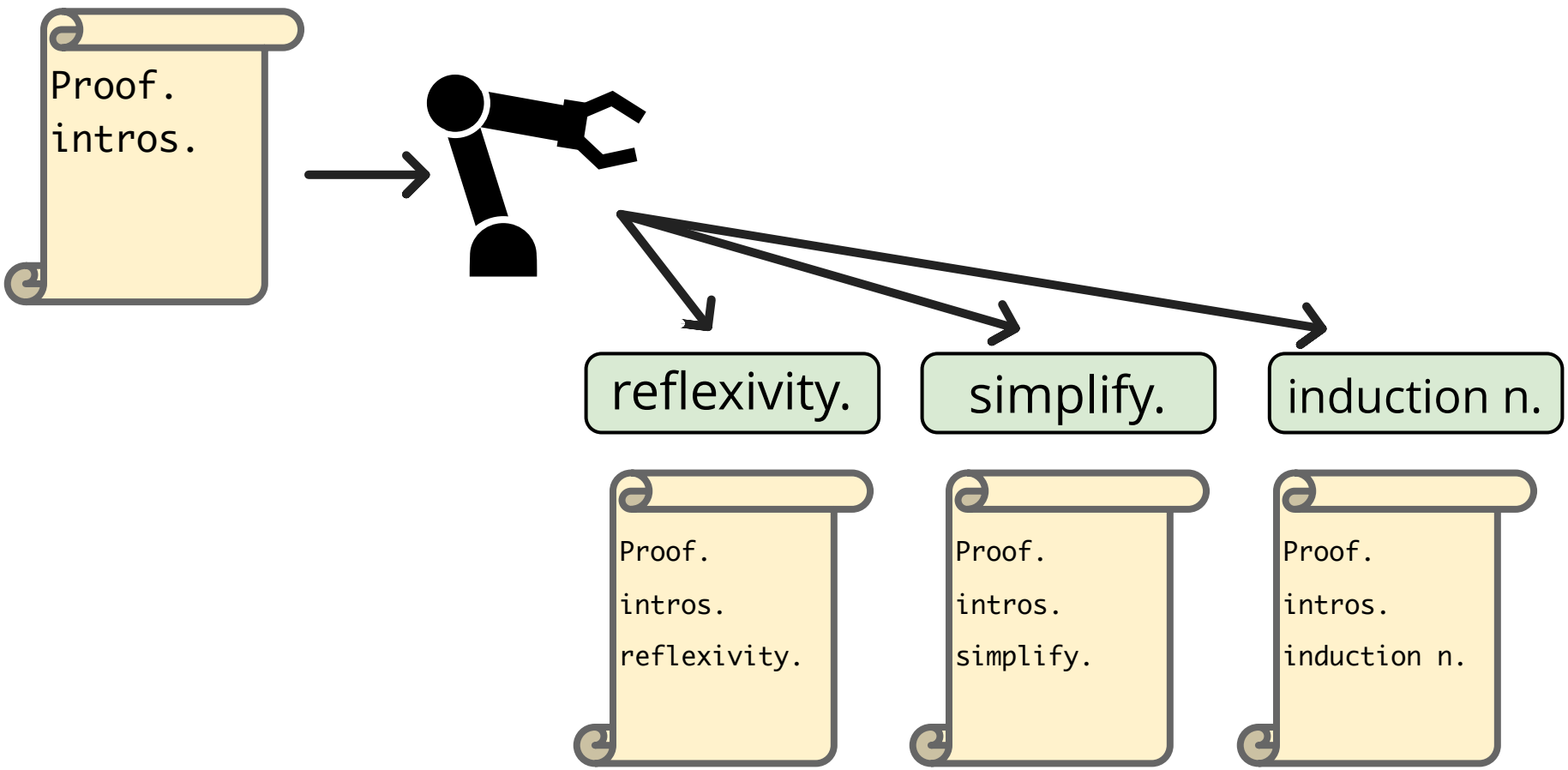
3. Searching for QED

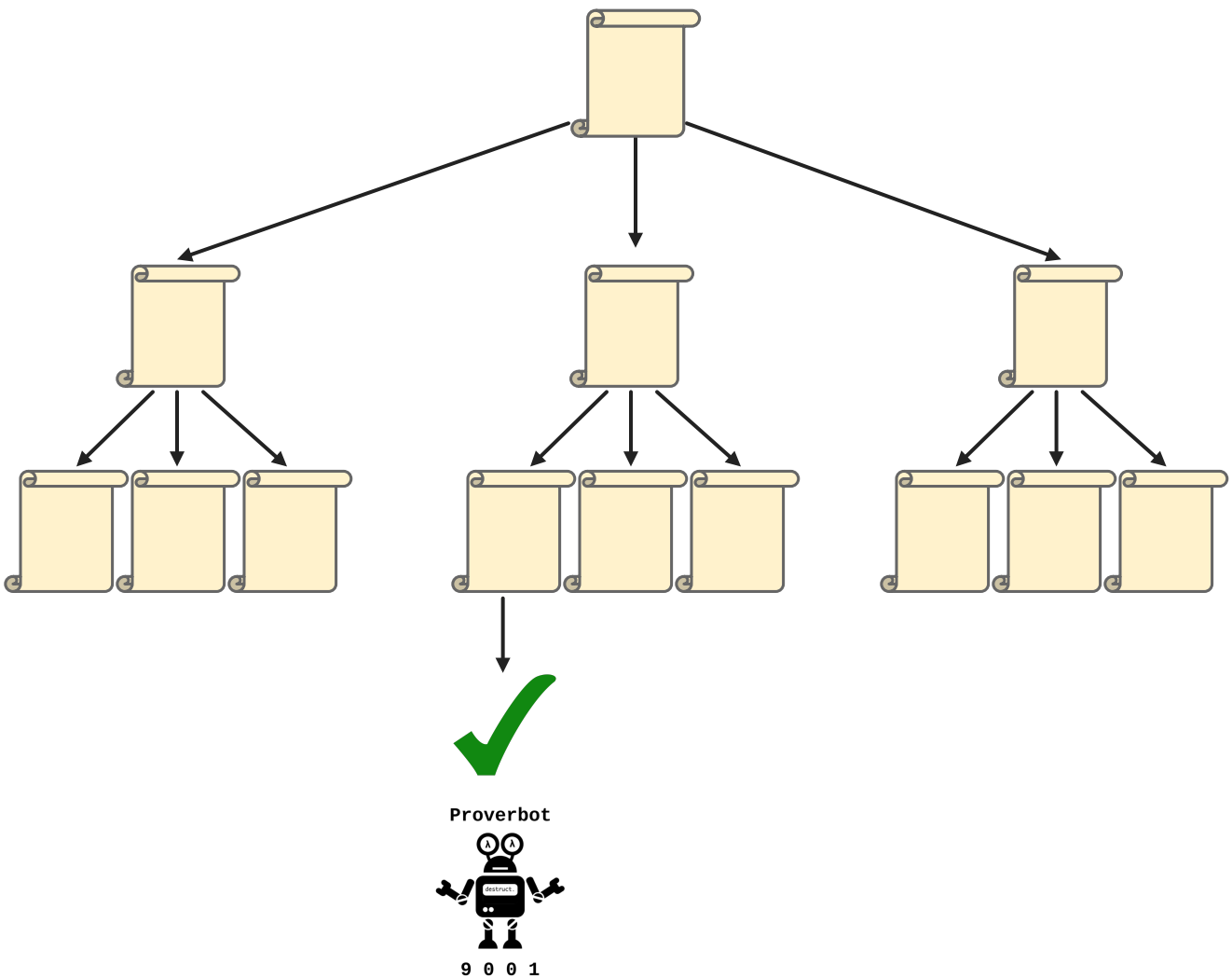


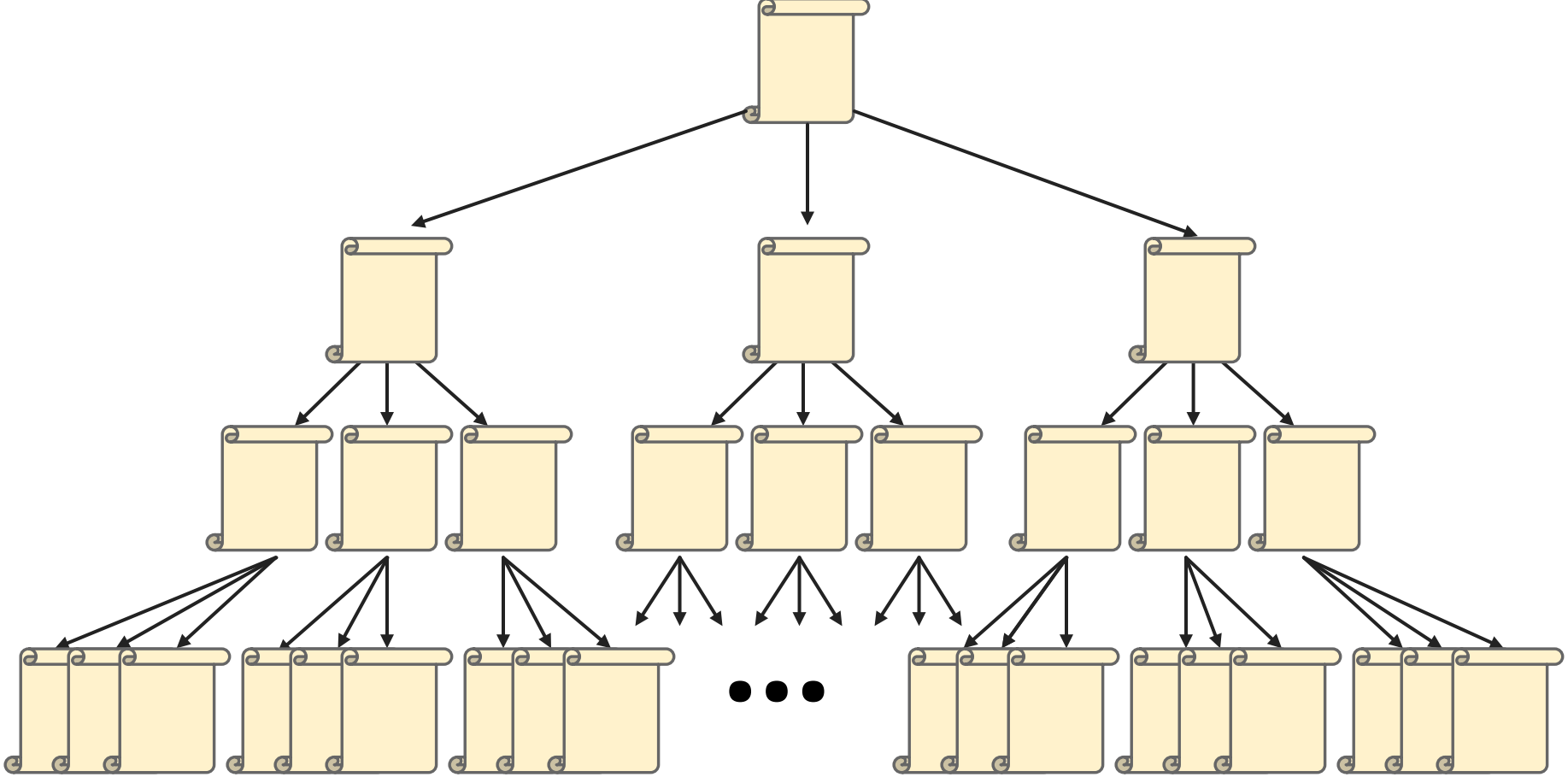
90% prediction accuracy doesn't mean much if you need to get 10 proof steps right.



The Solution: Search!



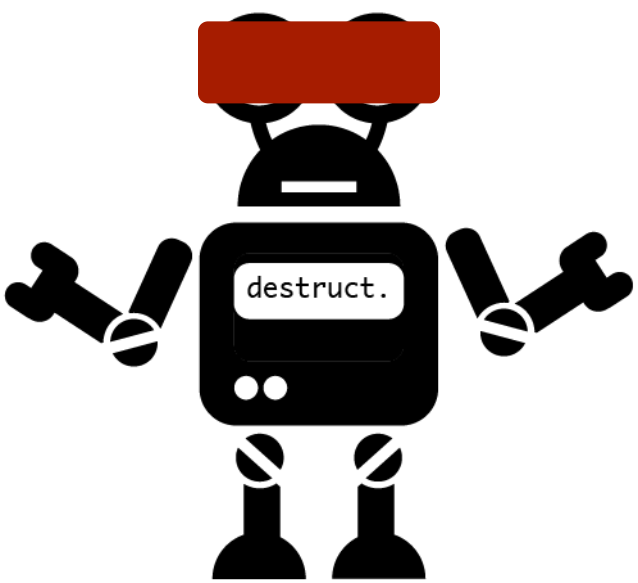

Proverbot
destruct
9 0 0 1



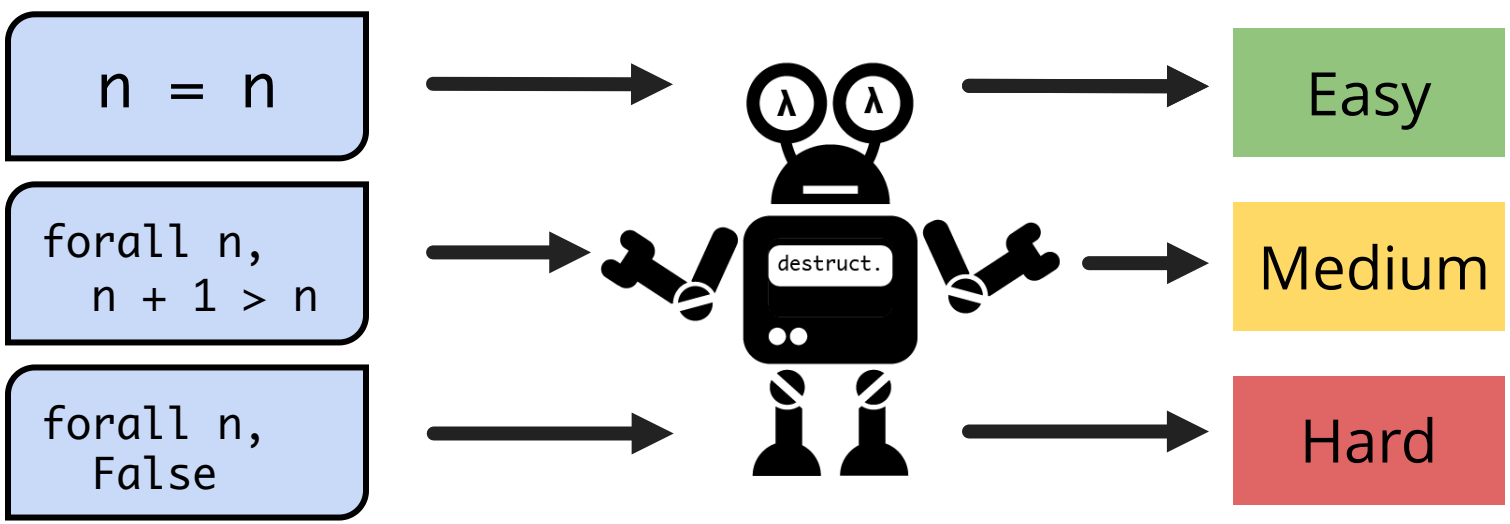
Unbounded Search Space

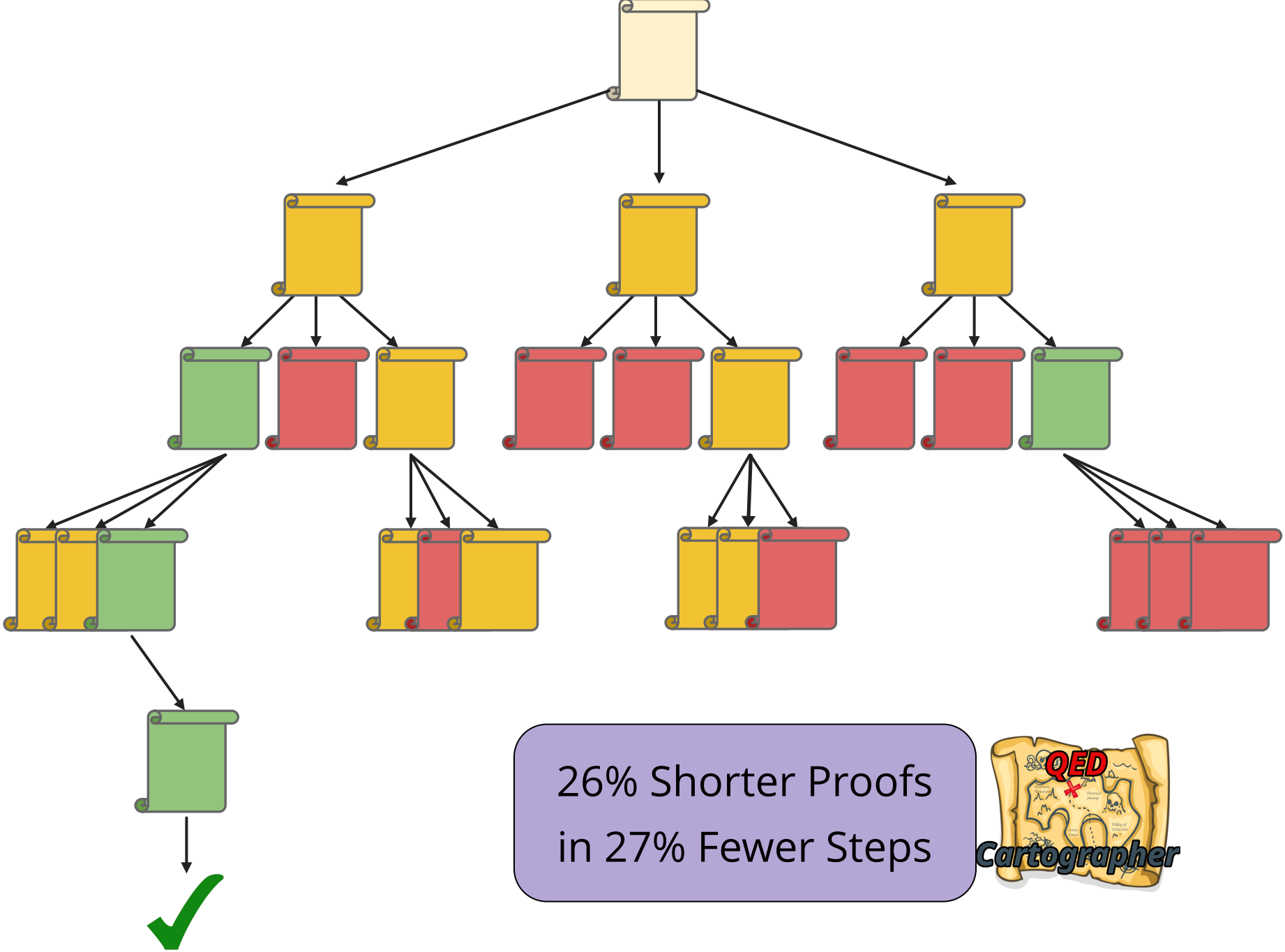
There are some techniques that can help us prune the search tree

It's hard to explore when you don't know where you are!



We can search more efficiently if we can evaluate proof states

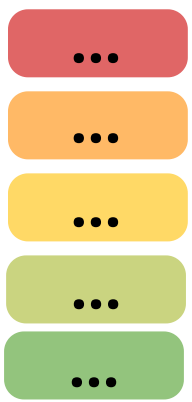




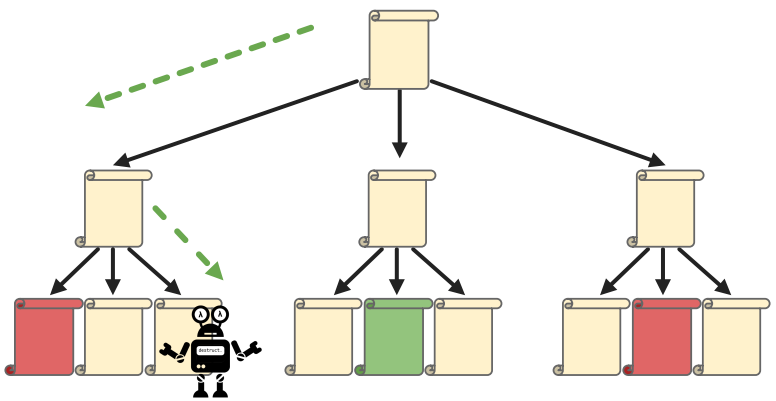
26% Shorter Proofs
in 27% Fewer Steps

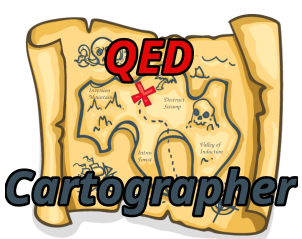
Off-Policy vs On-Policy

Proof.



Qed.



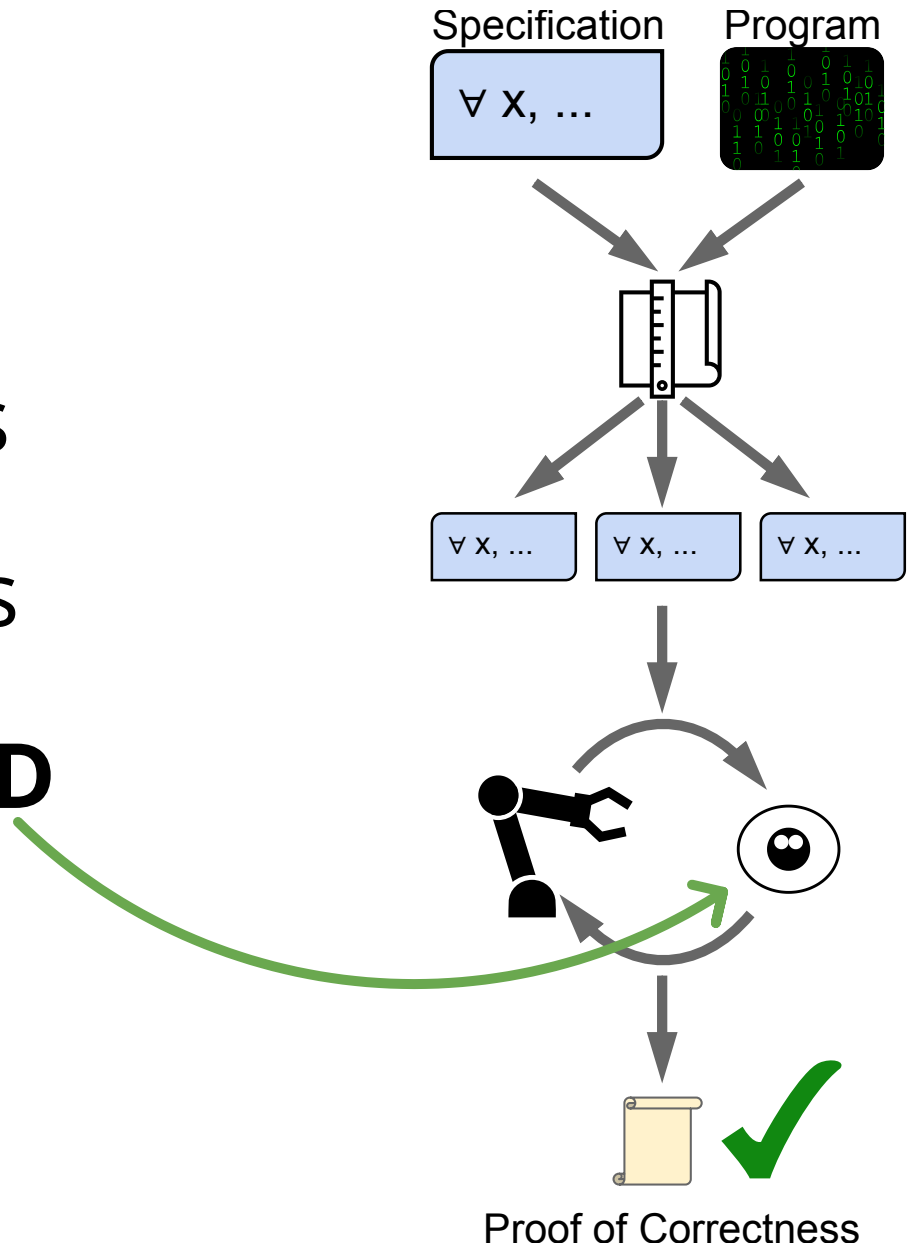


Alex Sanchez-Stern, et al. "QEDCartographer: Automating formal verification using reward-free reinforcement learning." *To appear at ICSE 2025*

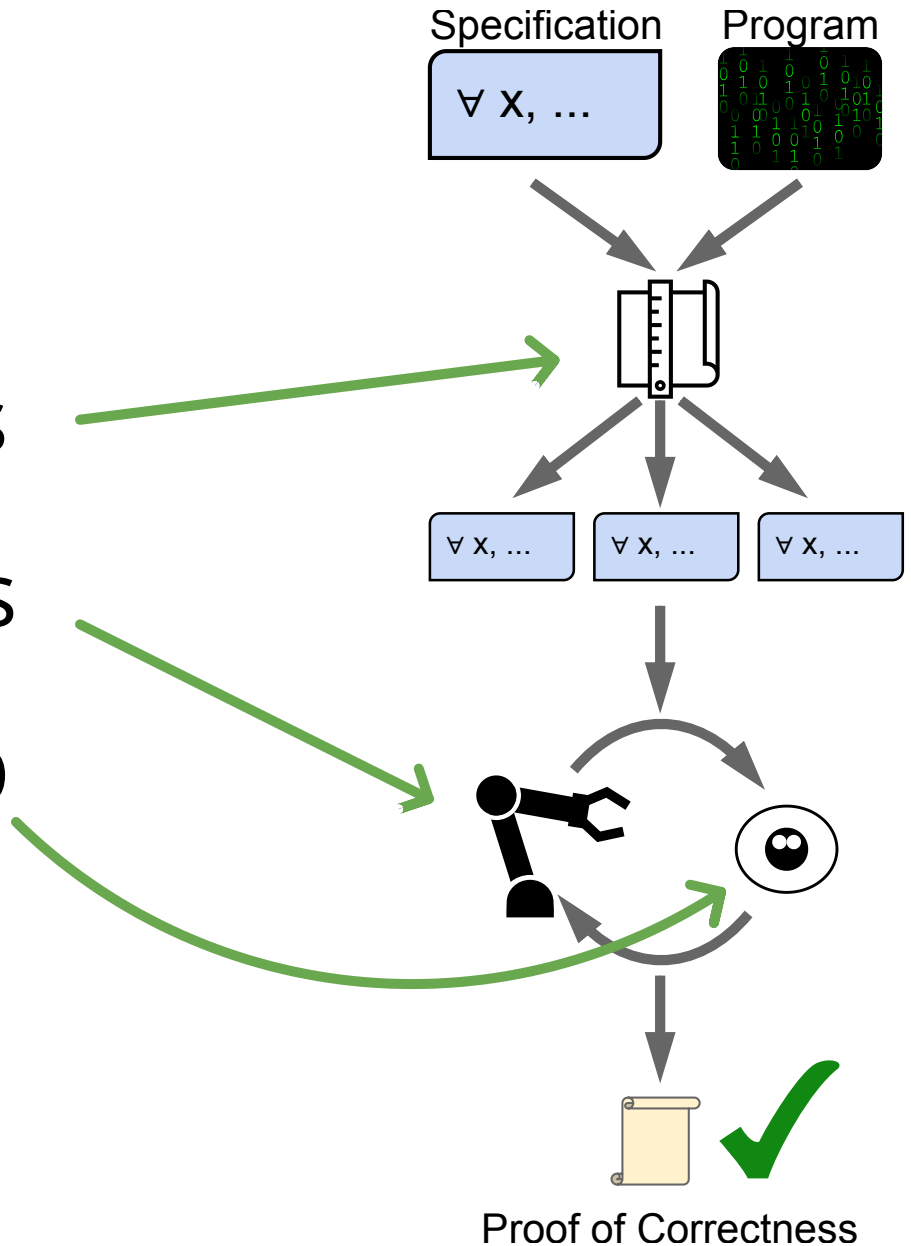
1. Structuring Proofs

2. Synthesizing Steps

3. Searching for QED



1. Structuring Proofs
2. Synthesizing Steps
3. Searching for QED



Collaborators

UCSDCSE
Computer Science and Engineering

LASER

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

UCLA

Emily First

Zhanna Kaufman

Talia Ringer

Aishwarya Sivaraman

Ana Brendel

Yousef Alhessi

Abhishek Varghese

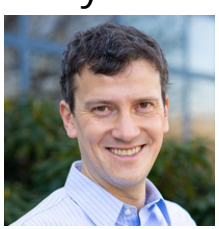
Timothy Zhou

Bretton Chen

Todd Millstein

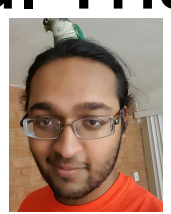
Sorin Lerner

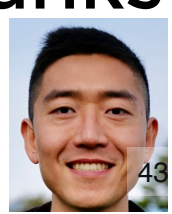
Yuriy Brun



Special Thanks

d_{model}





Machine-Checkable Proofs + Search + AI = ❤️