CoqPilot

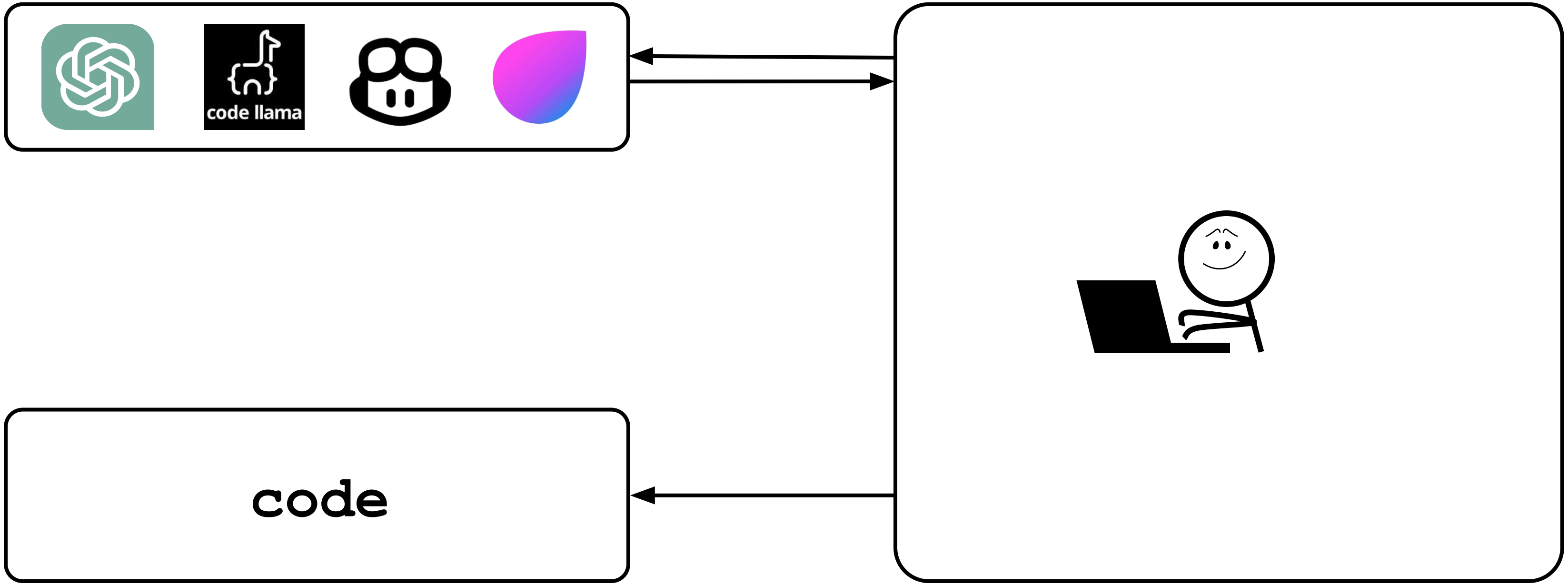
a plugin for LLM-based generation of proofs

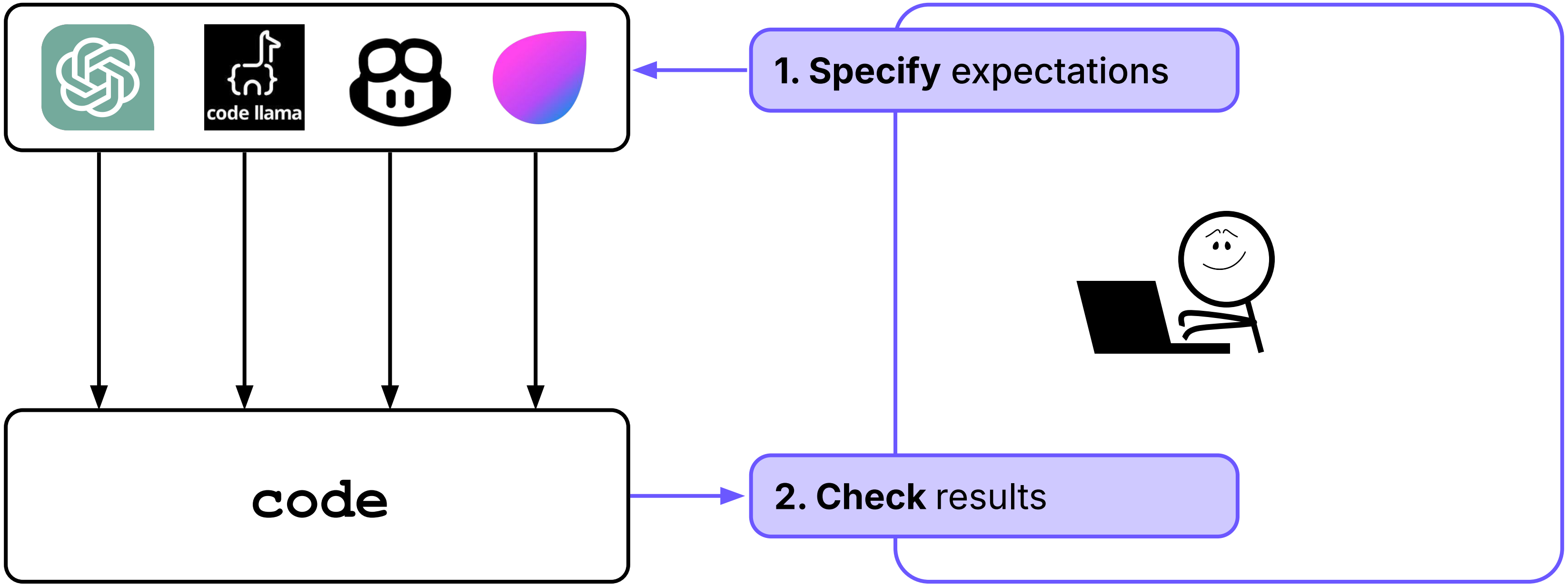
Andrei Kozyrev
Gleb Solovev
Nikita Khramov
Anton Podkopaev

Programming Languages and Program Analysis Lab (PLAN), JetBrains Research

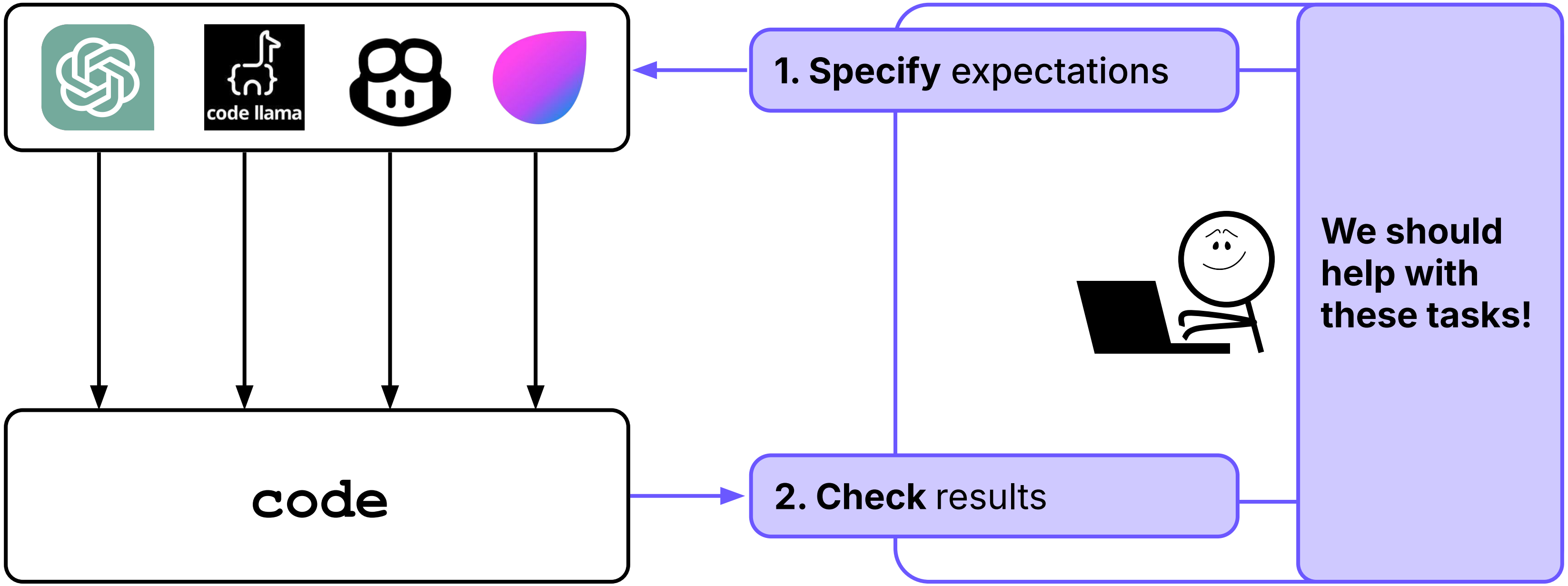
September, 2024

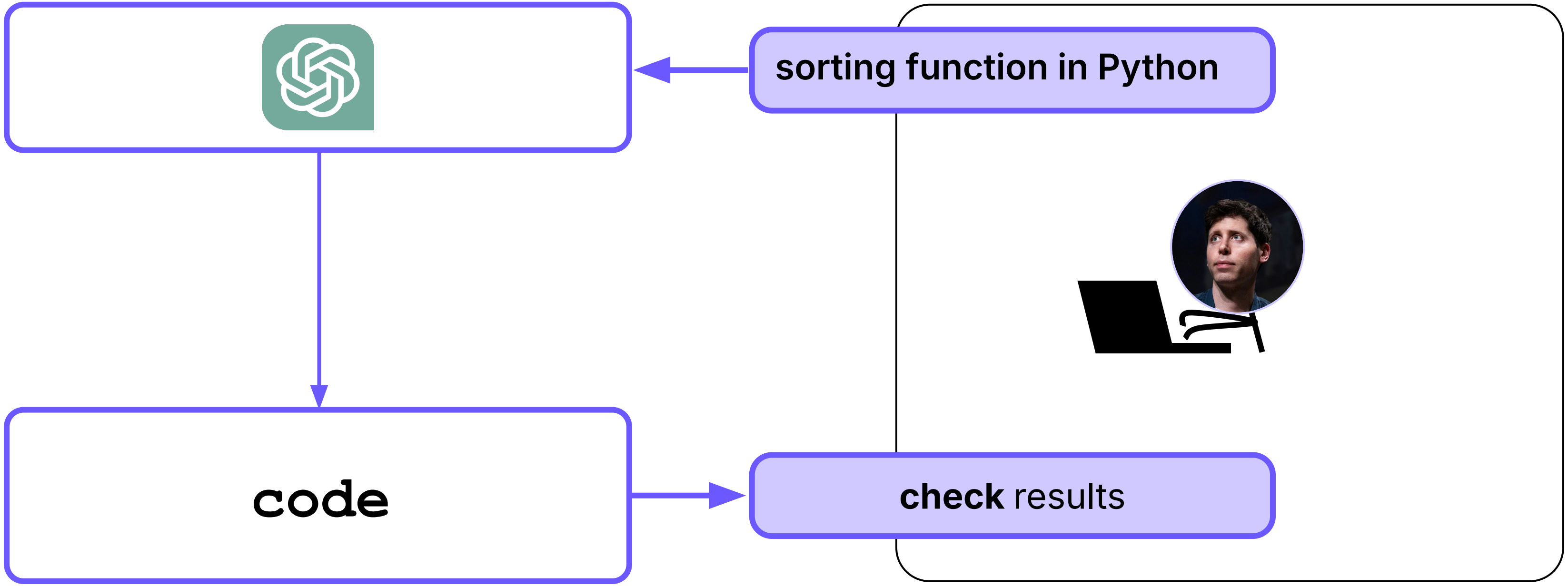
LLMs are used more and more for code generation




code llama
code
1. Specify expectations
2. Check results

Developer — LLM supervisor




sorting function in Python
code
check results

Generate a list sorting function in Python without using any library functions

```python
def custom_sort(arr):
  n = len(arr)
  for i in range(n):
    # Last i elements are already sorted
    for j in range(0, n - i - 1):
      # Traverse the array from 0 to n-i-1
      # Swap if the element found is greater
      # than the next element
      if arr[j] > arr[j + 1]:
        arr[j], arr[j + 1] = arr[j + 1], arr[j]
  return arr
```

Do Users Write More Insecure Code with AI Assistants?

Neil Perry *
Stanford University

Megha Srivastava *
Stanford University

Deepak Kumar
Stanford University

Dan Boneh
Stanford University

Do Users Write More Insecure Code with AI Assistants?

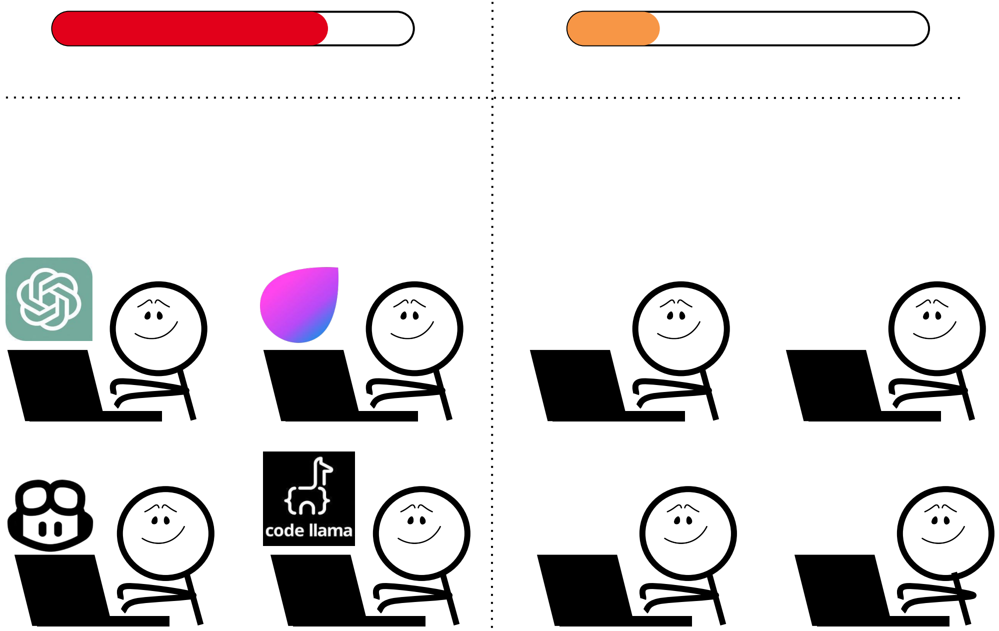
Neil Perry *
Stanford University

Megha Srivastava *
Stanford University

Deepak Kumar
Stanford University

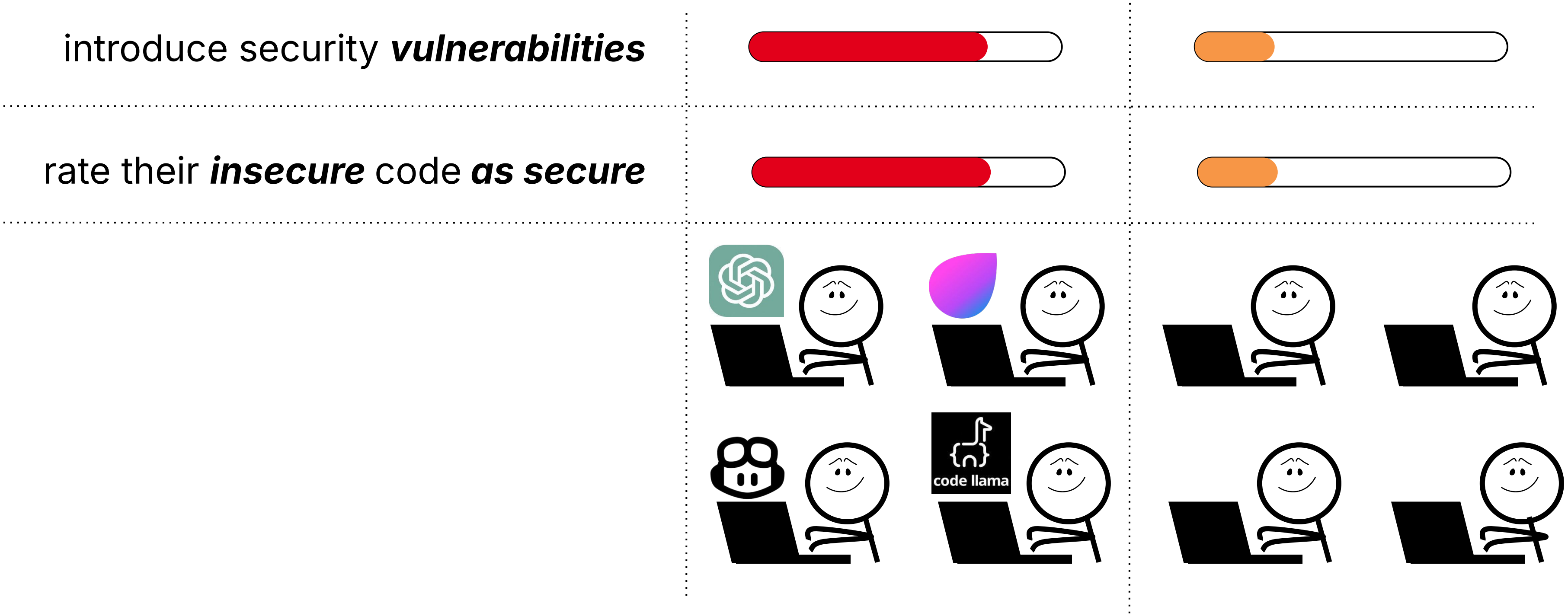
Dan Boneh
Stanford University

introduce security ***vulnerabilities***



Do Users Write More Insecure Code with AI Assistants?

Neil Perry * Stanford University
Megha Srivastava * Stanford University
Deepak Kumar Stanford University
Dan Boneh Stanford University



Generate a list sorting function in Python without using any library functions


```
def custom_sort(arr):

    # Last i elements are already sorted
    for j in range(0, n - i - 1):
      # Traverse the array from 0 to n-i-1
      # Swap if the element found is greater
      # than the next element
      if arr[j] > arr[j + 1]:
        arr[j], arr[j + 1] = arr[j + 1], arr[j]
  return arr
```

Not only **familiar** code is needed

Generate a list sorting function in Python without using any library functions


```
def custom_sort(arr):
        # Last i elements are already sorted
        for j in range(0, n - i - 1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

Not only **familiar** code is needed

Testing may **not** be **sufficient** (eg, concurrency)

Generate a list sorting function in Python without using any library functions


```python
def custom_sort(arr):
        # Traverse the array from 0 to n-i-1
        # Swap if the element found is greater
        # than the next element
        if arr[j] > arr[j + 1]:
            arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

Not only **familiar** code is needed

Testing may **not** be **sufficient** (eg, concurrency)

Plain **English** is **hard** to debug and **imprecise**

Is there a better way?

The Coq programming language

```
Definition sort (l : list nat) : {l' : list nat | Permutation l l' & is_sorted l'}.
```

The Coq programming language

Definition sort $\underbrace{\text{(l : list nat)}}_{\text{Argument}}$: {l' : list nat | Permutation l l' & is_sorted l'}.

The Coq programming language

```
Definition sort (l : list nat) :  {l' : list nat | Permutation l l' & is_sorted l'}.
```

Argument

Returning type

The Coq programming language

```
Definition sort (l : list nat) :  {l' : list nat | Permutation l l' & is_sorted l'}.
```

Argument

Returning type

+ **Any implementation** is a **correct** sorting function

The Coq programming language

```
Definition sort (l : list nat) :  {l' : list nat | Permutation l l' & is_sorted l'}.
```

Argument

Returning type

- **Any implementation** is a **correct** sorting function
- **Specification** is a type–automatic **checking**

s.v M

src > s.v > sort

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v

Not in proof mode

main* Environment: default.nix 0 0 0 Ready ✓ coqpilot "s.v" 152L 2817C written Ln 94, Col 59 Spaces: 2 UTF-8 LF Coq

s.v M

src > s.v > sort

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

Main buffer

ProofView: s.v

Not in proof mode

main* Environment: default.nix 0 0 0 Ready coqpilot "s.v" 152L 2817C written Ln 94, Col 59 Spaces: 2 UTF-8 LF Coq

s.v M

src > s.v > sort

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v

Not in proof mode

Interactively typechecked part

main* Environment: default.nix 0 0 0 Ready coqpilot "s.v" 152L 2817C written Ln 94, Col 59 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

MAIN 1 SHELVED 0 GIVEN UP 0

```
l: list nat

(1/1)
{l' : list nat | Permutation l l' & is_sorted l'}
```

Interactively typechecked part

main* Environment: default.nix ⊗ 0 ⚠ 0 0 Ready ✓ coqpilot "s.v" 152L 2817C written Ln 95, Col 7 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

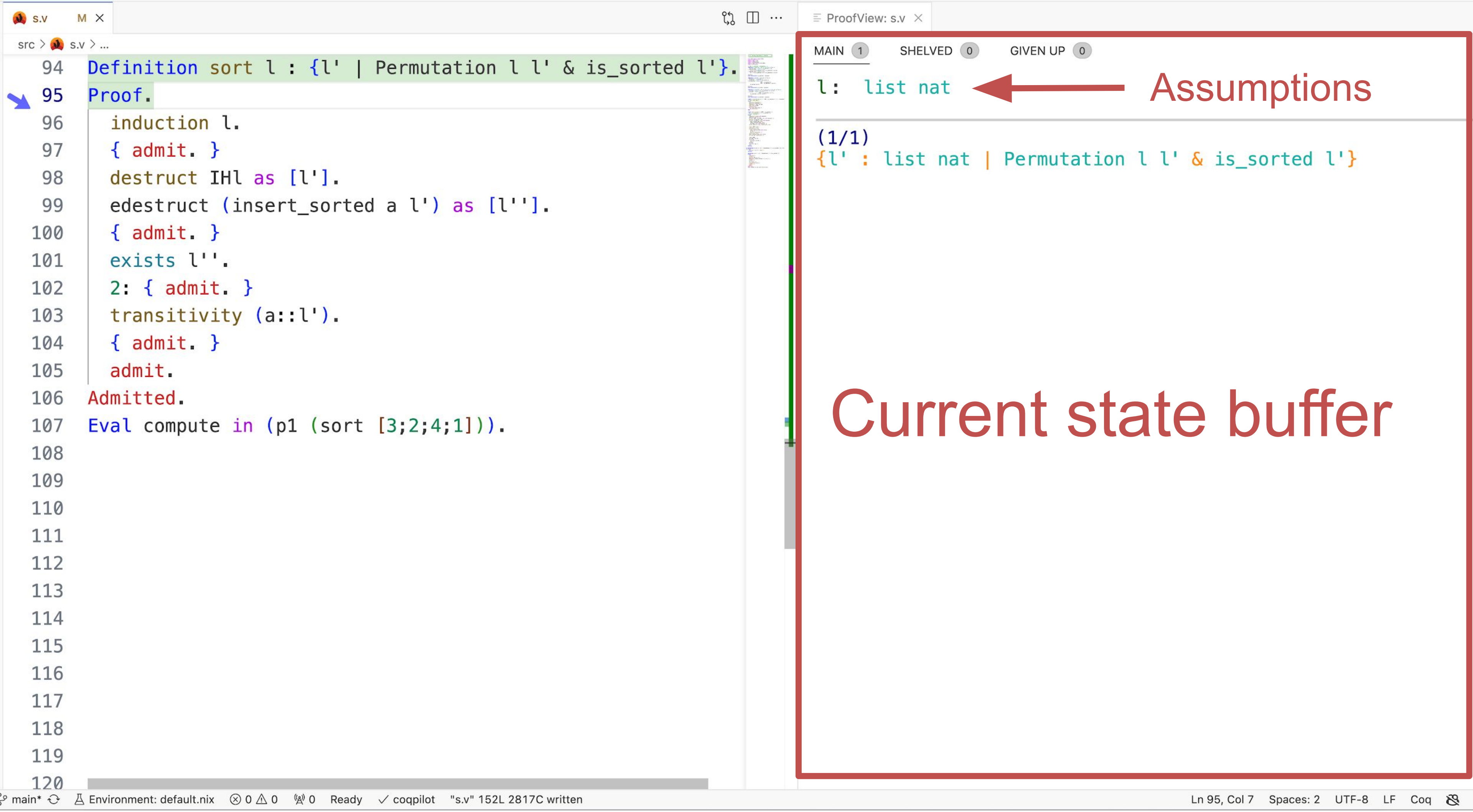
ProofView: s.v ×

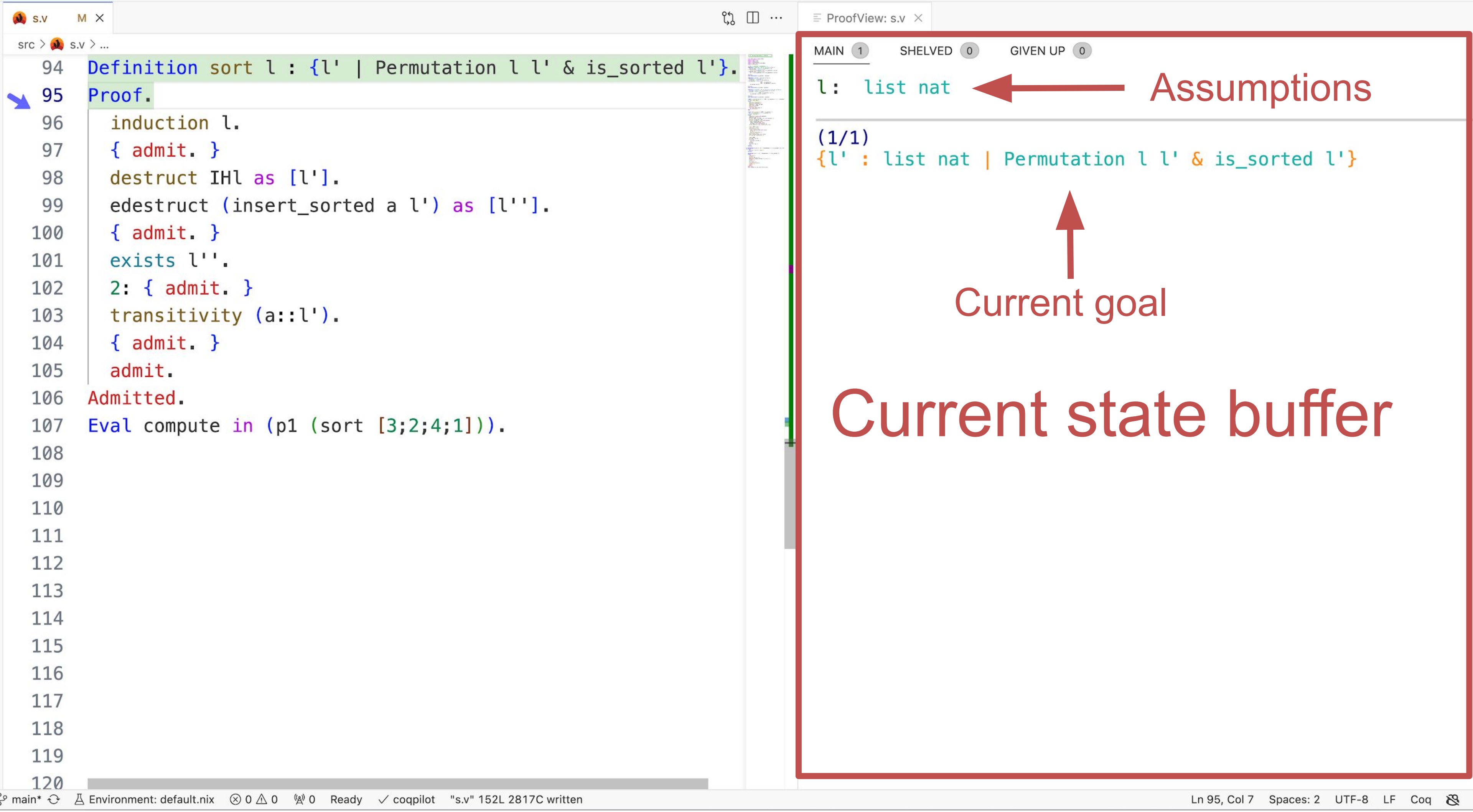
MAIN 1 SHELVED 0 GIVEN UP 0

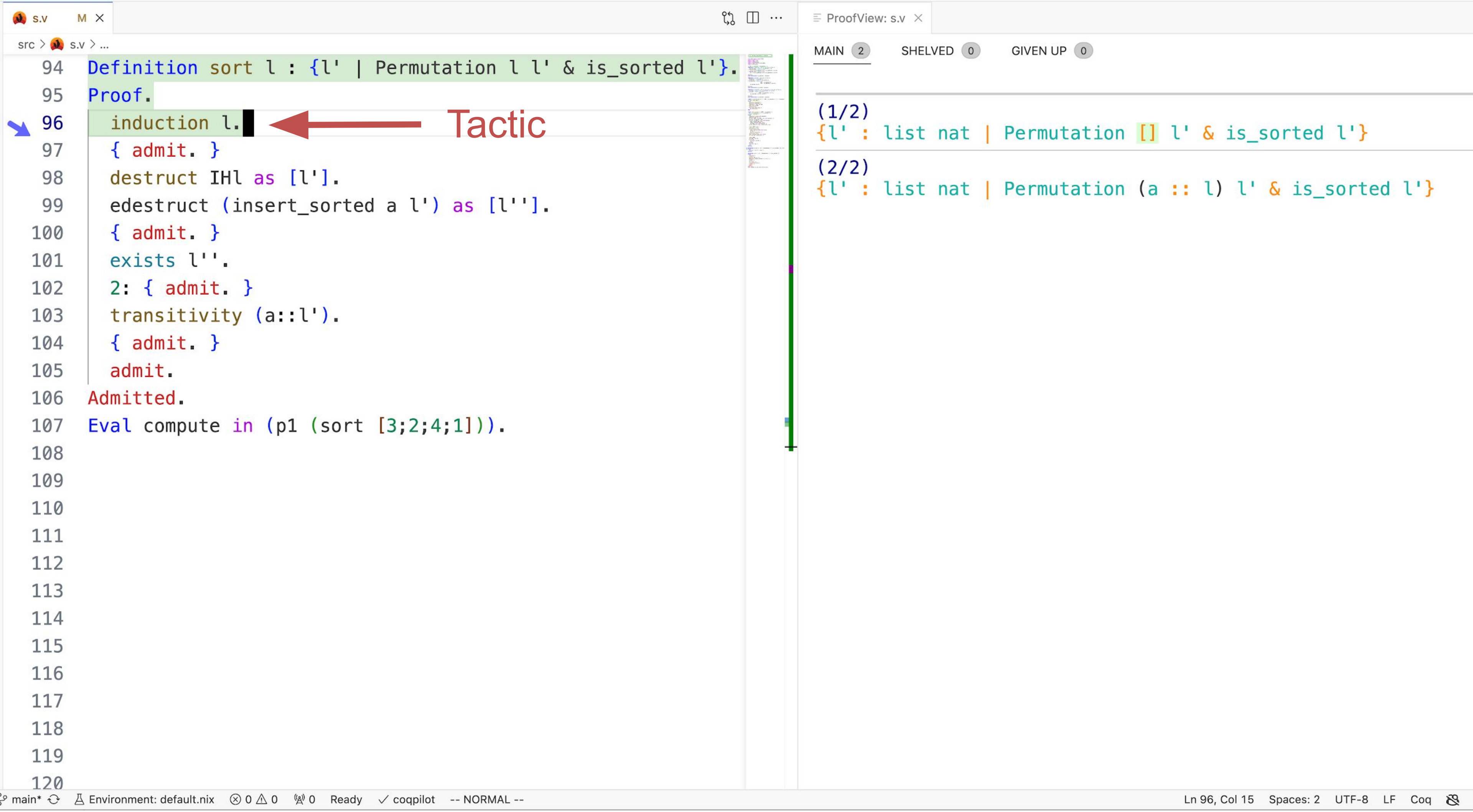
```
l: list nat

(1/1)
{l' : list nat | Permutation l l' & is_sorted l'}
```

Current state buffer

main* Environment: default.nix ⓧ 0 ⚠ 0 0 Ready ✓ coqpilot "s.v" 152L 2817C written Ln 95, Col 7 Spaces: 2 UTF-8 LF Coq


s.v M
src > s.v > ...
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
induction l.
{ admit. }
destruct IHl as [l'].
edestruct (insert_sorted a l') as [l''].
{ admit. }
exists l''.
2: { admit. }
transitivity (a::l').
{ admit. }
admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
ProofView: s.v
MAIN 1
SHELVED 0
GIVEN UP 0
l: list nat
Assumptions
(1/1)
{l' : list nat | Permutation l l' & is_sorted l'}
Current state buffer
main* Environment: default.nix 0 0 0 Ready coqpilot "s.v" 152L 2817C written
Ln 95, Col 7 Spaces: 2 UTF-8 LF Coq


s.v
ProofView: s.v
src > s.v > ...
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
induction l.
{ admit. }
destruct IHl as [l'].
edestruct (insert_sorted a l') as [l''].
{ admit. }
exists l''.
2: { admit. }
transitivity (a::l').
{ admit. }
admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
MAIN 1
SHELVED 0
GIVEN UP 0
l: list nat
Assumptions
(1/1)
{l' : list nat | Permutation l l' & is_sorted l'}
Current goal
Current state buffer
main*
Environment: default.nix
Ready
coqpilot
"s.v" 152L 2817C written
Ln 95, Col 7
Spaces: 2
UTF-8
LF
Coq


s.v
ProofView: s.v
src > s.v > ...
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
induction l.
{ admit. }
destruct IHl as [l'].
edestruct (insert_sorted a l') as [l''].
{ admit. }
exists l''.
2: { admit. }
transitivity (a::l').
{ admit. }
admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
Tactic
MAIN 2
SHELVED 0
GIVEN UP 0
(1/2)
{l' : list nat | Permutation [] l' & is_sorted l'}
(2/2)
{l' : list nat | Permutation (a :: l) l' & is_sorted l'}
main*
Environment: default.nix
Ready
coqpilot
-- NORMAL --
Ln 96, Col 15
Spaces: 2
UTF-8
LF
Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

MAIN 1 SHELVED 0 GIVEN UP 0

(1/1)

{l' : list nat | Permutation [] l' & is_sorted l'}

main* Environment: default.nix ⓧ 0 ⚠ 0 0 Ready ✓ coqpilot "s.v" 152L 2817C written Ln 97, Col 4 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

MAIN 0 SHELVED 0 GIVEN UP 1

There are unfocused goals

main* Environment: default.nix ⊗ 0 ⚠ 0 0 Ready ✓ coqpilot "s.v" 152L 2817C written Ln 97, Col 11 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

MAIN 1 SHELVED 0 GIVEN UP 1

Warning: You have given up goals

```
a : nat
l : list nat
IHl : {l' : list nat | Permutation l l' & is_sorted l'}
```

```
(1/1)
{l' : list nat | Permutation (a :: l) l' & is_sorted l'}
```

main* Environment: default.nix ⓧ 0 ⚠ 0 (·) 0 Ready ✓ coqpilot "s.v" 152L 2817C written Ln 97, Col 13 Spaces: 2 UTF-8 LF Coq

s.v M

src > s.v > ...

```coq
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v

MAIN 1 SHELVED 0 GIVEN UP 1

Warning: You have given up goals

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
________________________________
(1/1)
{l'0 : list nat | Permutation (a :: l) l'0 & is_sorted l'0}
```

main* Environment: default.nix 0 0 0 Ready coqpilot -- NORMAL -- Ln 98, Col 24 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

MAIN 2 SHELVED 0 GIVEN UP 1

Warning: You have given up goals

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'

(1/2)
is_sorted l'

(2/2)
{l'0 : list nat | Permutation (a :: l) l'0 & is_sorted l'0}
```

main* Environment: default.nix ⊗ 0 ⚠ 0 0 Ready ✓ coqpilot -- NORMAL -- Ln 99, Col 43 Spaces: 2 UTF-8 LF Coq

s.v M

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v

MAIN 1 SHELVED 0 GIVEN UP 2

Warning: You have given up goals

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
l'' : list nat
i0 : is_inserted a l' l''
i1 : is_sorted l''
----------------------------------------
(1/1)
{l'0 : list nat | Permutation (a :: l) l'0 & is_sorted l'0}
```

main* Environment: default.nix 0 0 0 Ready ✓ coqpilot -- NORMAL -- Ln 100, Col 13 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

MAIN 2 SHELVED 0 GIVEN UP 2

Warning: You have given up goals

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
l'' : list nat
i0 : is_inserted a l' l''
i1 : is_sorted l''
```

(1/2)

```
Permutation (a :: l) l''
```

(2/2)

```
is_sorted l''
```

main* Environment: default.nix ⊗ 0 ⚠ 0 0 Ready ✓ coqpilot -- NORMAL -- Ln 101, Col 14 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

MAIN 1 SHELVED 0 GIVEN UP 3

Warning: You have given up goals

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
l'' : list nat
i0 : is_inserted a l' l''
i1 : is_sorted l''
______________________________________
(1/1)
Permutation (a :: l) l''
```

main* Environment: default.nix ⊗ 0 ⚠ 0 0 Ready ✓ coqpilot -- NORMAL -- Ln 102, Col 16 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

MAIN 2 SHELVED 0 GIVEN UP 3

Warning: You have given up goals

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
l'' : list nat
i0 : is_inserted a l' l''
i1 : is_sorted l''
```

```
(1/2)
Permutation (a :: l) (a :: l')
```

```
(2/2)
Permutation (a :: l') l''
```

main* Environment: default.nix ⊗ 0 ⚠ 0 0 Ready ✓ coqpilot -- NORMAL -- Ln 103, Col 24 Spaces: 2 UTF-8 LF Coq

s.v M

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v

MAIN 1 SHELVED 0 GIVEN UP 4

Warning: You have given up goals

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
l'' : list nat
i0 : is_inserted a l' l''
i1 : is_sorted l''
________________________
(1/1)
Permutation (a :: l') l''
```

main* Environment: default.nix 0 0 0 Ready coqpilot -- NORMAL -- Ln 104, Col 13 Spaces: 2 UTF-8 LF Coq

s.v M

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v

MAIN 0 SHELVED 0 GIVEN UP 5

There are some goals you gave up. You need to go back and solve them, or use Admitted.

main* Environment: default.nix 0 0 0 Ready coqpilot -- NORMAL -- Ln 105, Col 9 Spaces: 2 UTF-8 LF Coq

s.v M ×

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

ProofView: s.v ×

Not in proof mode

main* Environment: default.nix ⊗ 0 ⚠ 0 0 Ready ✓ coqpilot "s.v" 152L 2817C written Ln 106, Col 10 Spaces: 2 UTF-8 LF Coq

s.v M

ProofView: s.v

src > s.v > ...

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
```

Not in proof mode

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Notices

```
= let (x, _, _) := sort [3; 2; 4; 1] in x
: list nat
```

main* Environment: default.nix 0 0 0 Ready coqpilot -- NORMAL -- Ln 107, Col 39 Spaces: 2 UTF-8 LF Coq

How do we fill those admits?

Coq is very suitable for code generation

Coq is very suitable for code generation

- **Type-checking** validates **deep** program's **properties**

Coq is very suitable for code generation

- **Type-checking** validates **deep** program's **properties**
- **Tactics** do **not** enforce **strict** structure

Coq is very suitable for code generation

- **Type-checking** validates **deep** program's **properties**
- **Tactics** do **not** enforce **strict** structure
- Easy to **guide generation** with providing **structure and insights**

Coq is very suitable for code generation

- **Type-checking** validates **deep** program's **properties**
- **Tactics** do **not** enforce **strict** structure
- Easy to **guide generation** with providing **structure and insights**
- **Holes** are independent and might be **filled in parallel**

CoqPilot

Coq Goals | s.v M

src > s.v

```
Lemma is_inserted_perm a l l' (INS : is_inserted a l
(* Hint: perm_swap *)
Proof.
  generalize dependent l'.
  generalize dependent a.
  induction l; ins; inv INS.
  apply IHl in INS0.
  etransitivity.
  { by apply perm_swap. }
    by constructor.
Qed.

Lemma insert_sorted a l (SORT : is_sorted l) :
  {l' | is_inserted a l l' & is_sorted l'}.
(* Hint: le_gt_dec *)
Proof.
  induction l; eauto with myconstr.
  edestruct IHl as [l'].
  { clear -SORT. inv SORT. auto with myconstr. }
  destruct (le_gt_dec a a0).
  { exists (a::a0::l); auto with myconstr.
    apply sorted_cons; auto.
    eapply smallest_head; eauto.
    inv SORT. auto with myconstr. }
  exists (a0::l'); auto. constructor; auto.

  clear -SORT i i0 g.
  induction i; auto.
```

Coq Goals

Proof

Main 1 Shelved 0 Given up 1

Goal 1

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
(1 / 1)
is_sorted l'
```

Messages

main* 0↓ 3↑ ⊗ 0 ⚠ 0 ⓘ 1 0 opam(coqpilot) Ln 99, Col 5 Spaces: 2 UTF-8 LF Coq Go Live Record Story (beta) Prettier

Coq Goals | s.v M

src > s.v

```
Lemma is_inserted_perm a l l' (INS : is_inserted a l
(* Hint: perm_swap *)
Proof.
  generalize dependent l'.
  generalize dependent a.
  induction l; ins; inv INS.
  apply IHl in INS0.
  etransitivity.
  { by apply perm_swap. }
    by constructor.
Qed.

Lemma insert_sorted a l (SORT : is_sorted l) :
  {l' | is_inserted a l l' & is_sorted l'}.
(* Hint: le_gt_dec *)
Proof.
  induction l; eauto with myconstr.
  edestruct IHl as [l'].
  { clear -SORT. inv SORT. auto with myconstr. }
  destruct (le_gt_dec a a0).
  { exists (a::a0::l); auto with myconstr.
    apply sorted_cons; auto.
    eapply smallest_head; eauto.
    inv SORT. auto with myconstr. }
  exists (a0::l'); auto. constructor; auto.

  clear -SORT i i0 g.
  induction i; auto.
```

Coq Goals

Proof

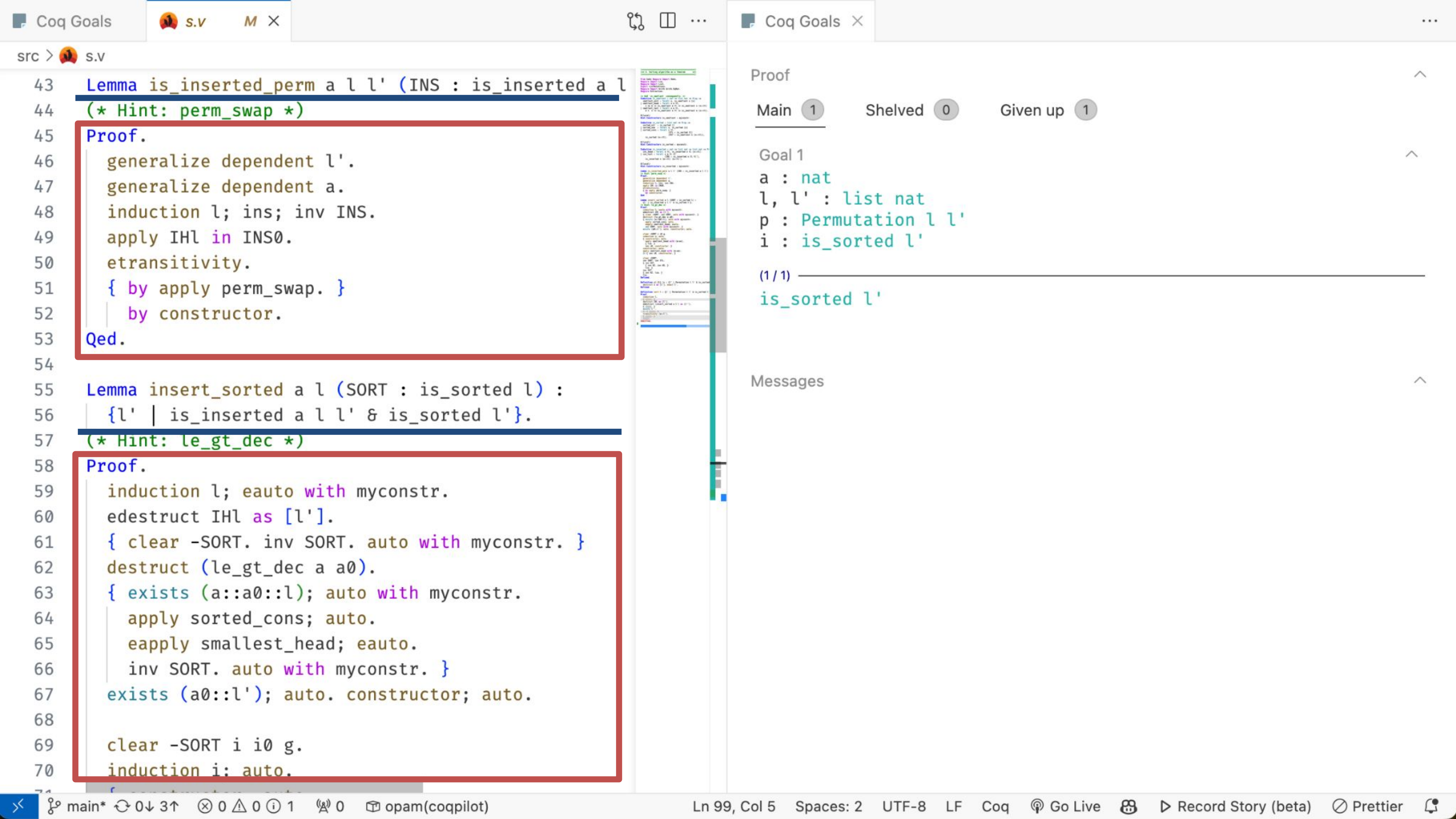
Main 1 | Shelved 0 | Given up 1

Goal 1

```
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
(1 / 1)
is_sorted l'
```

Messages

main* 0↓ 3↑ ⊗ 0 ⚠ 0 ⓘ 1 0 opam(coqpilot) Ln 99, Col 5 Spaces: 2 UTF-8 LF Coq Go Live Record Story (beta) Prettier


Coq Goals
s.v M
Coq Goals
src > s.v
Lemma is_inserted_perm a l l' (INS : is_inserted a l
(* Hint: perm_swap *)
Proof.
generalize dependent l'.
generalize dependent a.
induction l; ins; inv INS.
apply IHl in INS0.
etransitivity.
{ by apply perm_swap. }
by constructor.
Qed.
Lemma insert_sorted a l (SORT : is_sorted l) :
{l' | is_inserted a l l' & is_sorted l'}.
(* Hint: le_gt_dec *)
Proof.
induction l; eauto with myconstr.
edestruct IHl as [l'].
{ clear -SORT. inv SORT. auto with myconstr. }
destruct (le_gt_dec a a0).
{ exists (a::a0::l); auto with myconstr.
apply sorted_cons; auto.
eapply smallest_head; eauto.
inv SORT. auto with myconstr. }
exists (a0::l'); auto. constructor; auto.
clear -SORT i i0 g.
induction i; auto.
Proof
Main 1
Shelved 0
Given up 1
Goal 1
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
(1 / 1)
is_sorted l'
Messages
main* 0↓ 3↑
0 0 1
0
opam(coqpilot)
Ln 99, Col 5
Spaces: 2
UTF-8
LF
Coq
Go Live
Record Story (beta)
Prettier

s.v M

src > s.v

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.

Eval compute in (p1 (sort [3;2;4;1])).
```

Coq Goals

Proof

Not in proof mode

Messages

```
= let (x, _, _) := sort [3; 2; 4; 1] in x
: list nat
```

main* 0↓ 3↑ Launchpad 0 0 1 0 opam(coqpilot) vscoq-language-server 2.1.0, coq 8.19.0 Go Live Record Story (beta)

s.v M

src > s.v

```
Definition sort l : {l' | Permutation l l' & is_sorted l'}.
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.

Eval compute in (p1 (sort [3;2;4;1])).
```

Coq Goals

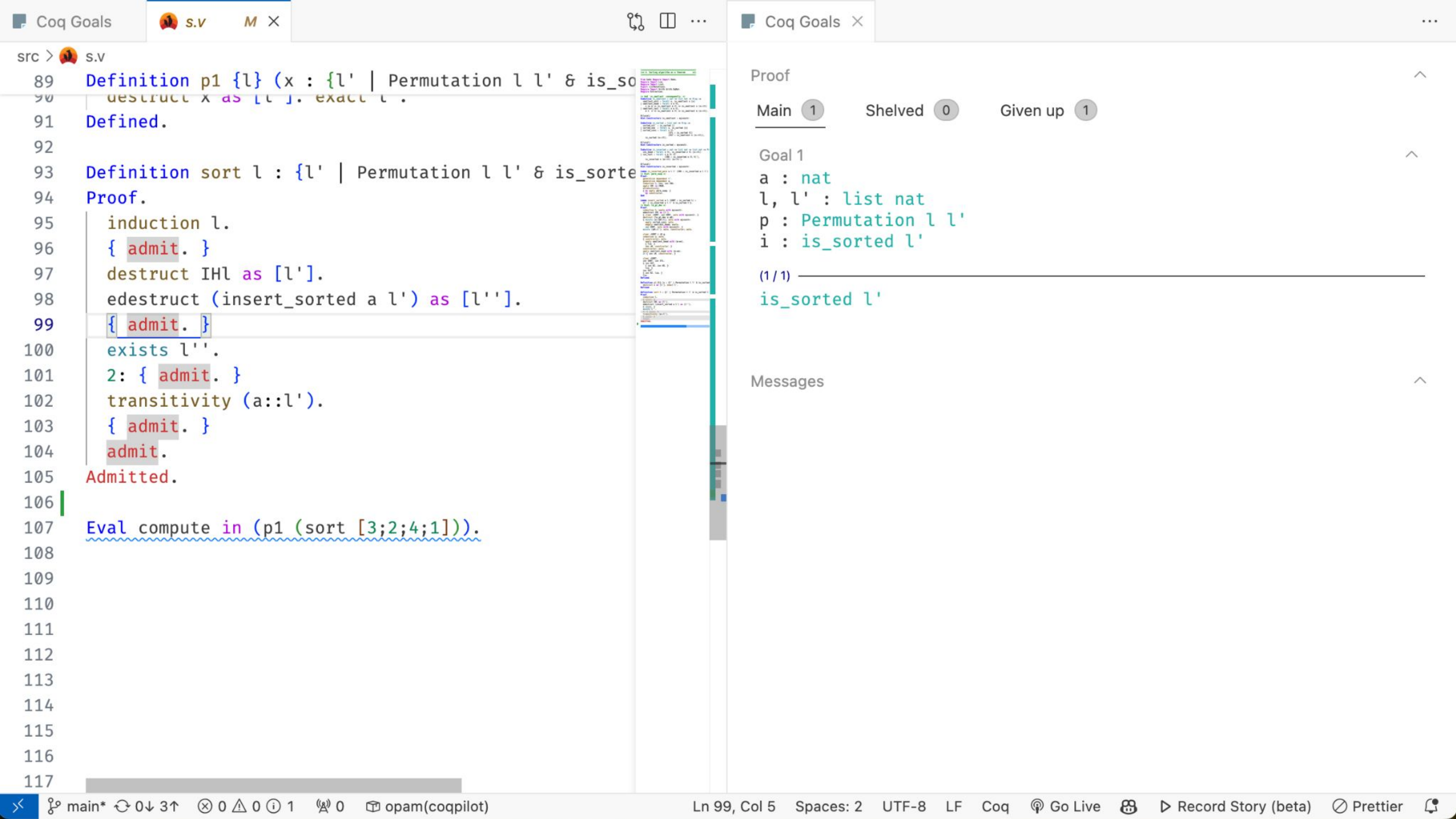
Proof

```
Not in proof mode
```

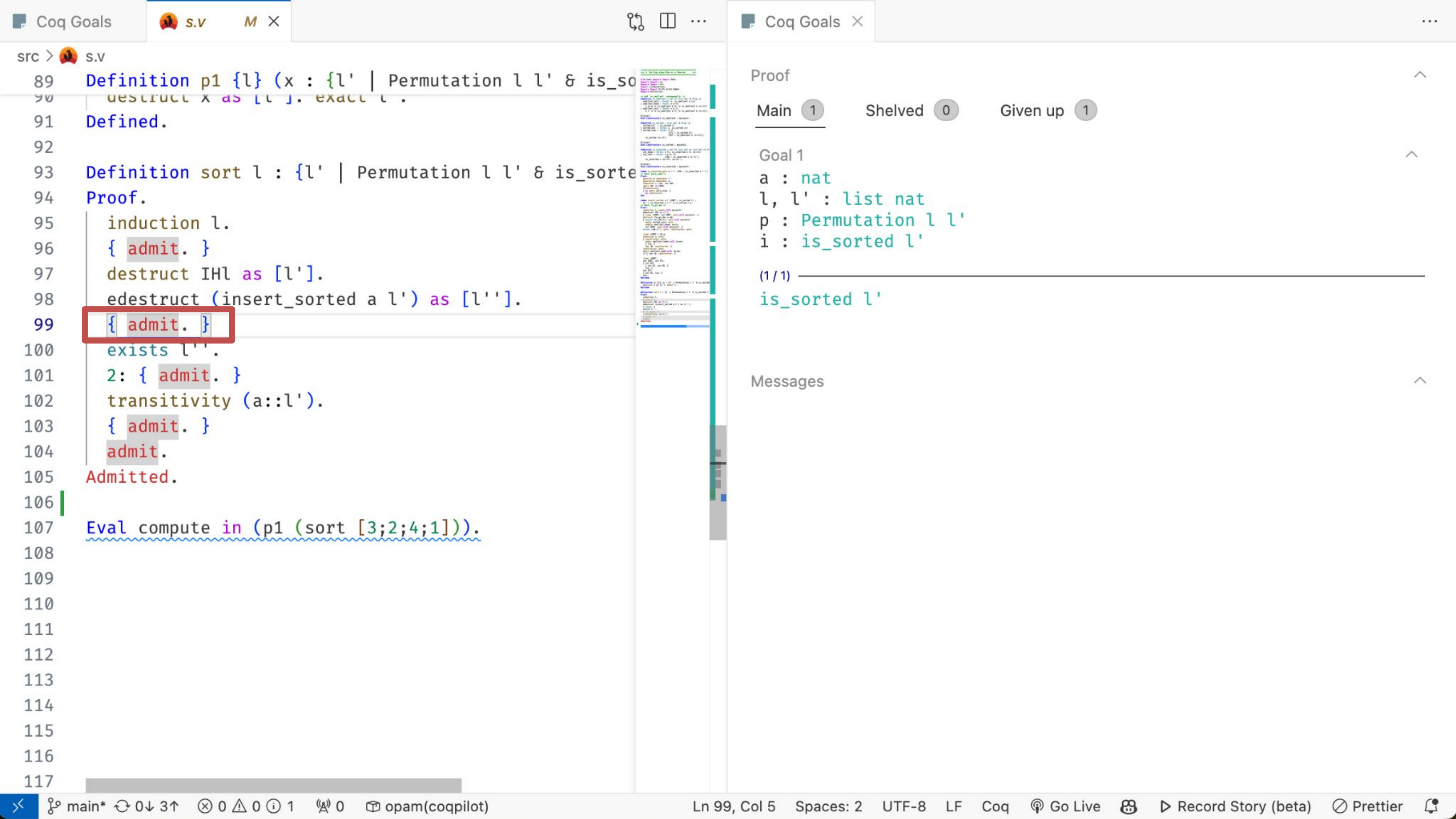
Messages

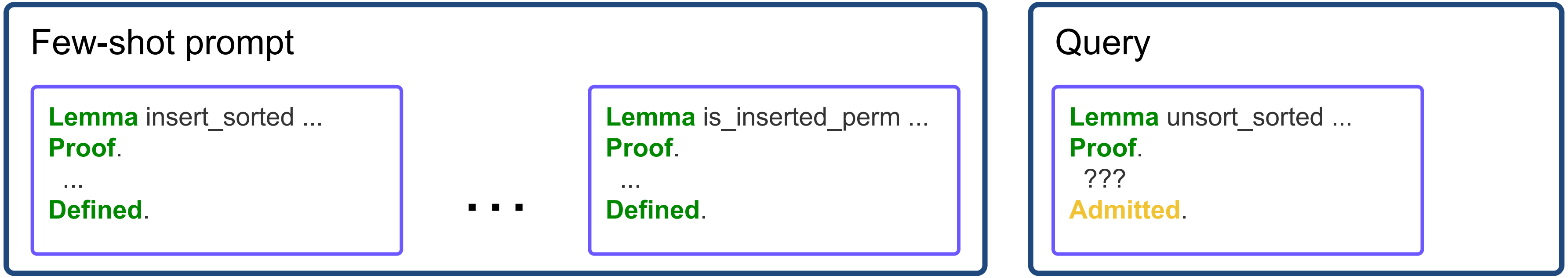
```
= let (x, _, _) := sort [3; 2; 4; 1] in x
: list nat
```

main* 0↓ 3↑ Launchpad 0 0 1 0 opam(coqpilot) vscoq-language-server 2.1.0, coq 8.19.0 Go Live Record Story (beta)

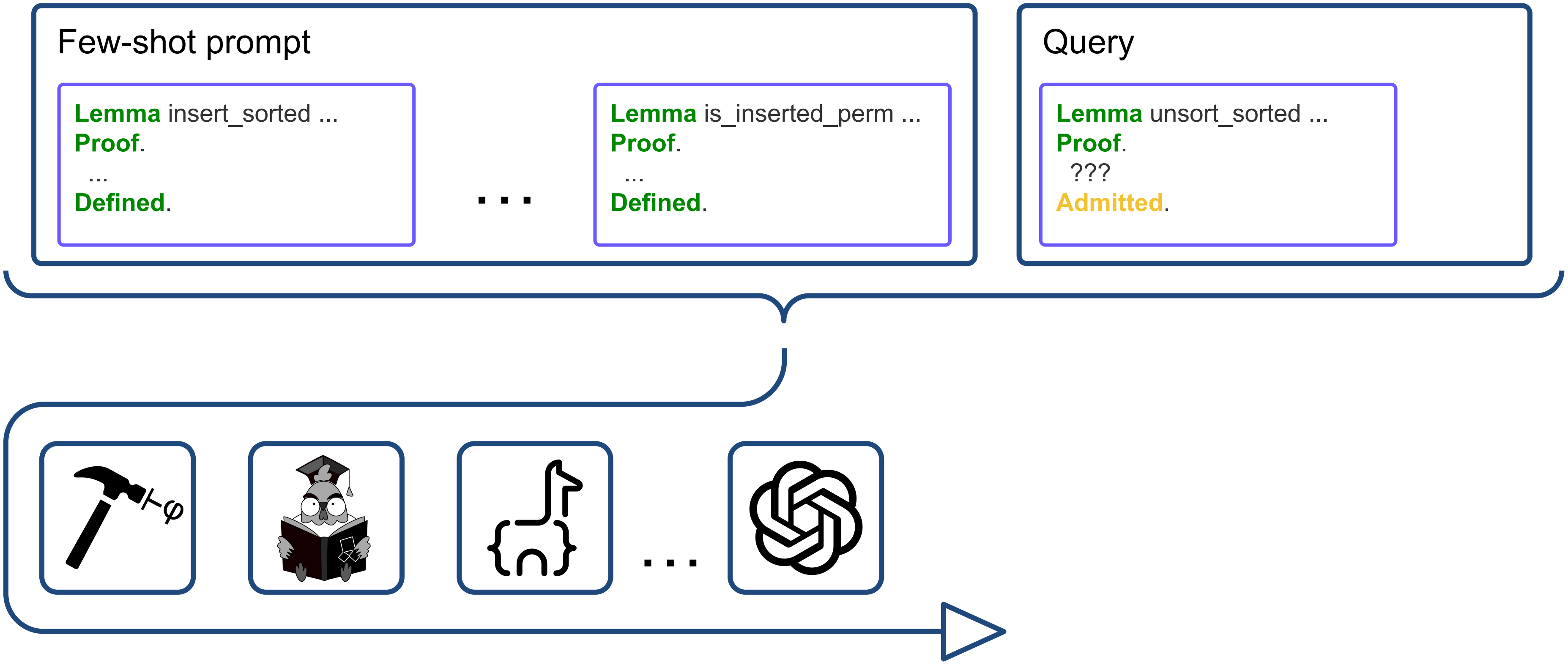


Coq Goals
s.v M
src > s.v
Definition p1 {l} (x : {l' | Permutation l l' & is_so
destruct x as [l']. exact l'.
Defined.
Definition sort l : {l' | Permutation l l' & is_sorte
Proof.
induction l.
{ admit. }
destruct IHl as [l'].
edestruct (insert_sorted a l') as [l''].
{ admit. }
exists l''.
2: { admit. }
transitivity (a::l').
{ admit. }
admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
Coq Goals
Proof
Main 1
Shelved 0
Given up 1
Goal 1
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
(1 / 1)
is_sorted l'
Messages
main* 0↓ 3↑
0 0 1
0
opam(coqpilot)
Ln 99, Col 5
Spaces: 2
UTF-8
LF
Coq
Go Live
Record Story (beta)
Prettier

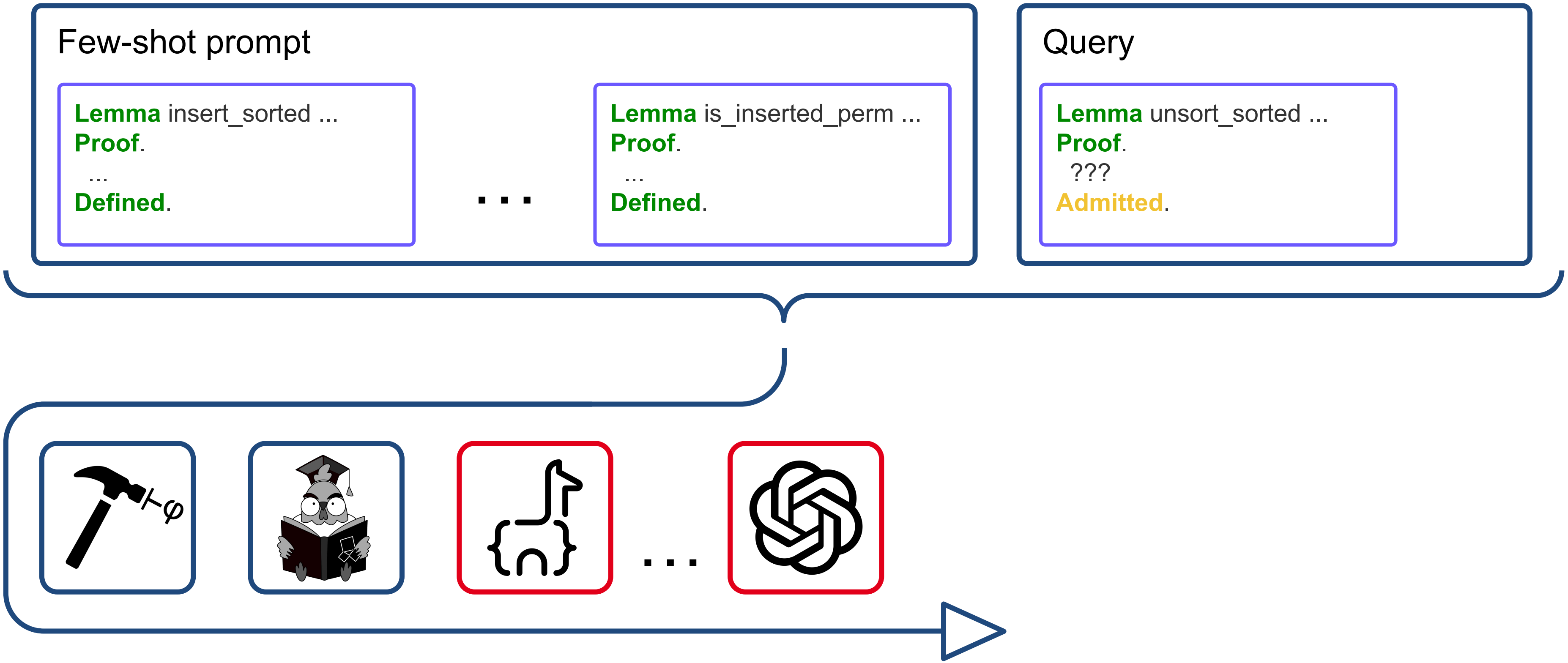

Coq Goals
s.v M
src > s.v
Definition p1 {l} (x : {l' | Permutation l l' & is_so
Defined.
Definition sort l : {l' | Permutation l l' & is_sorte
Proof.
induction l.
{ admit. }
destruct IHl as [l'].
edestruct (insert_sorted a l') as [l''].
{ admit. }
exists l''.
2: { admit. }
transitivity (a::l').
{ admit. }
admit.
Admitted.
Eval compute in (p1 (sort [3;2;4;1])).
Coq Goals
Proof
Main 1
Shelved 0
Given up 1
Goal 1
a : nat
l, l' : list nat
p : Permutation l l'
i : is_sorted l'
(1 / 1)
is_sorted l'
Messages
main* 0↓ 3↑
opam(coqpilot)
Ln 99, Col 5
Spaces: 2
UTF-8
LF
Coq
Go Live
Record Story (beta)
Prettier



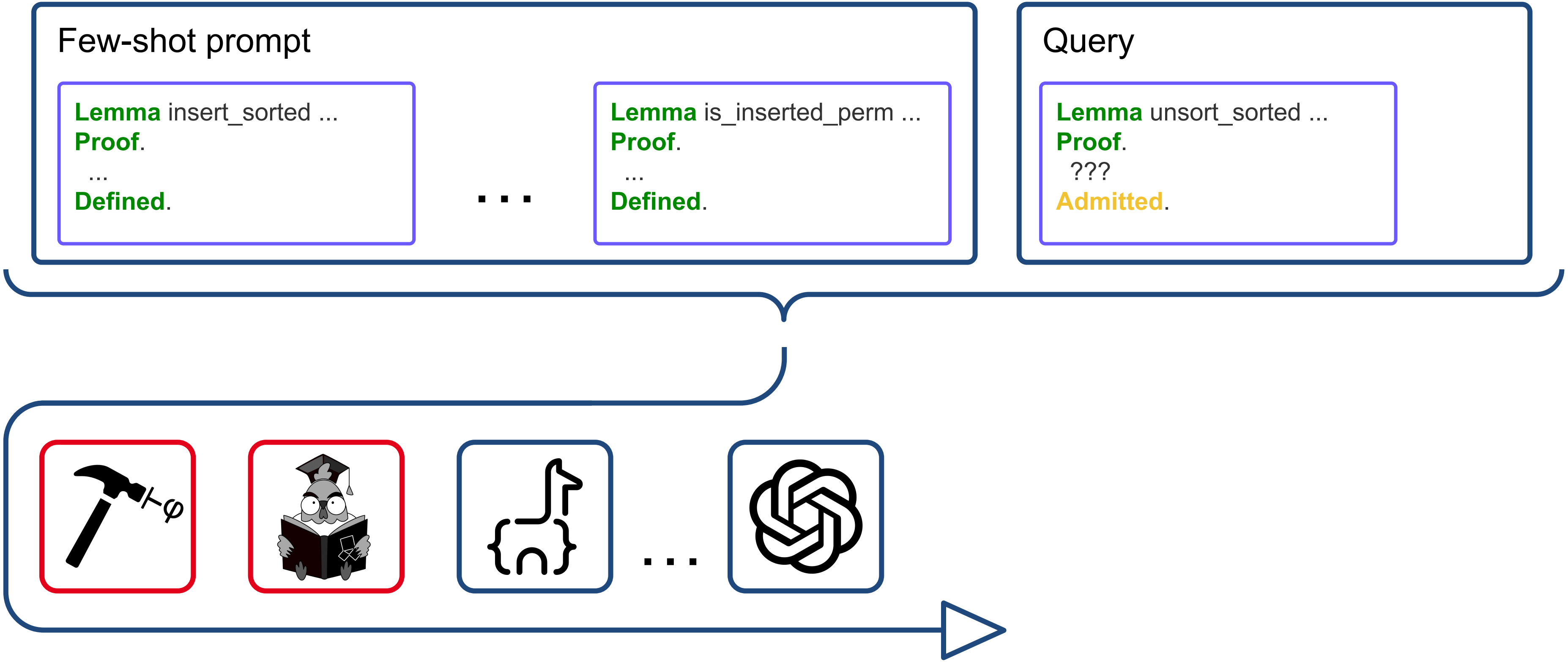
Few-shot prompt
Lemma insert_sorted ...
Proof.
...
Defined.
. . .
Lemma is_inserted_perm ...
Proof.
...
Defined.
Query
Lemma unsort_sorted ...
Proof.
???
Admitted.

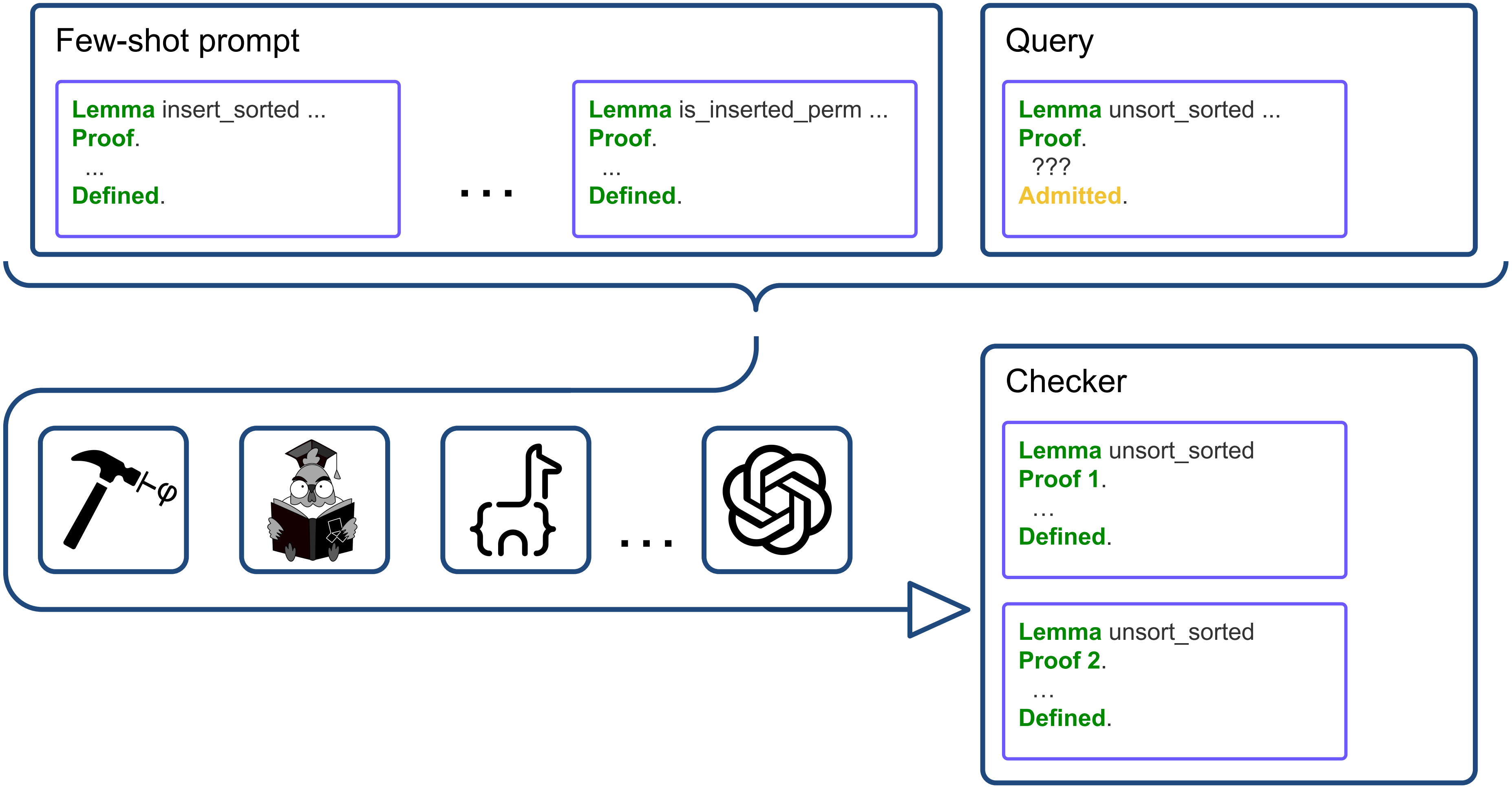


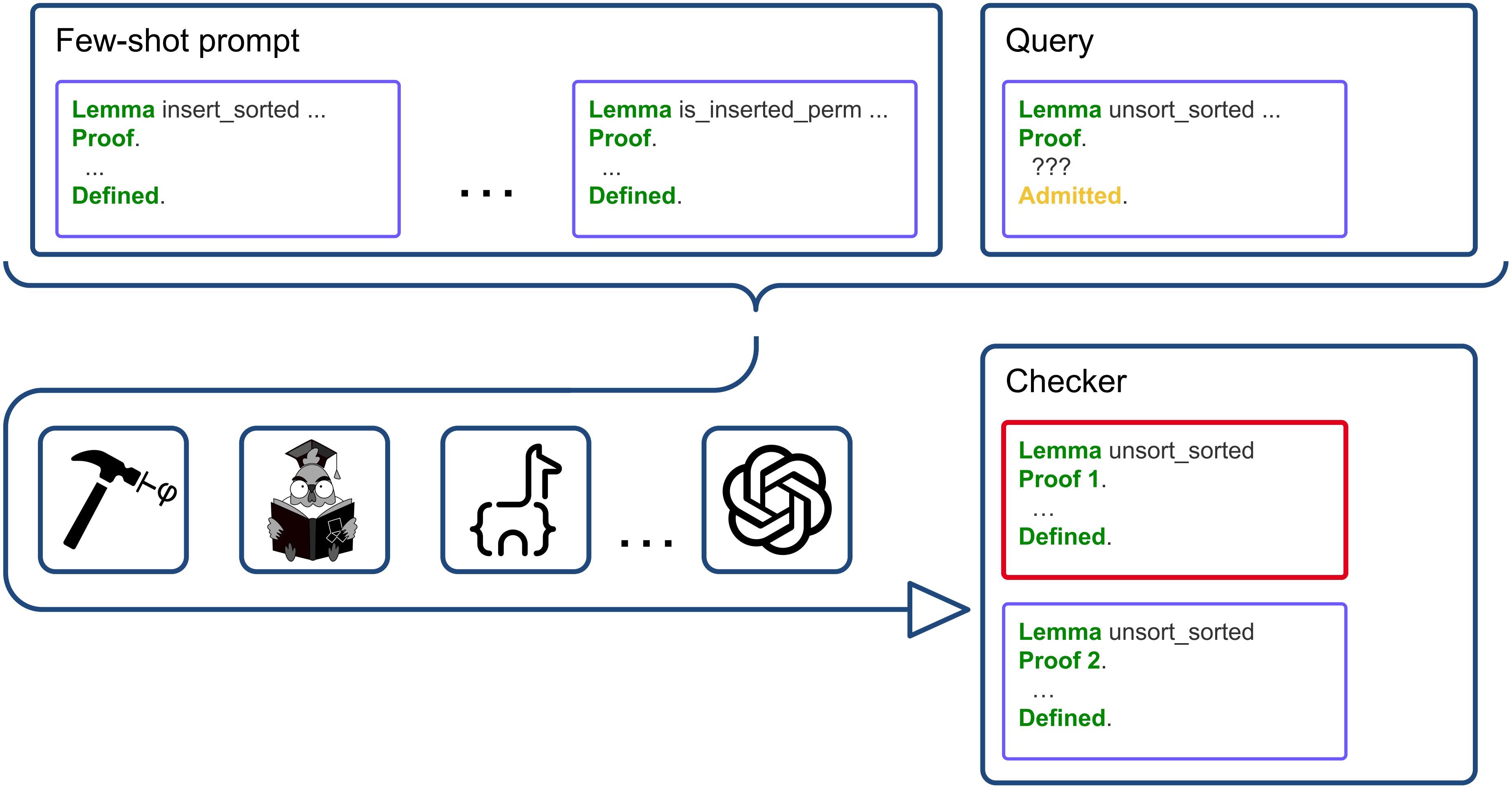
Few-shot prompt
Lemma insert_sorted ...
Proof.
...
Defined.
...
Lemma is_inserted_perm ...
Proof.
...
Defined.
Query
Lemma unsort_sorted ...
Proof.
???
Admitted.
⊢φ
...



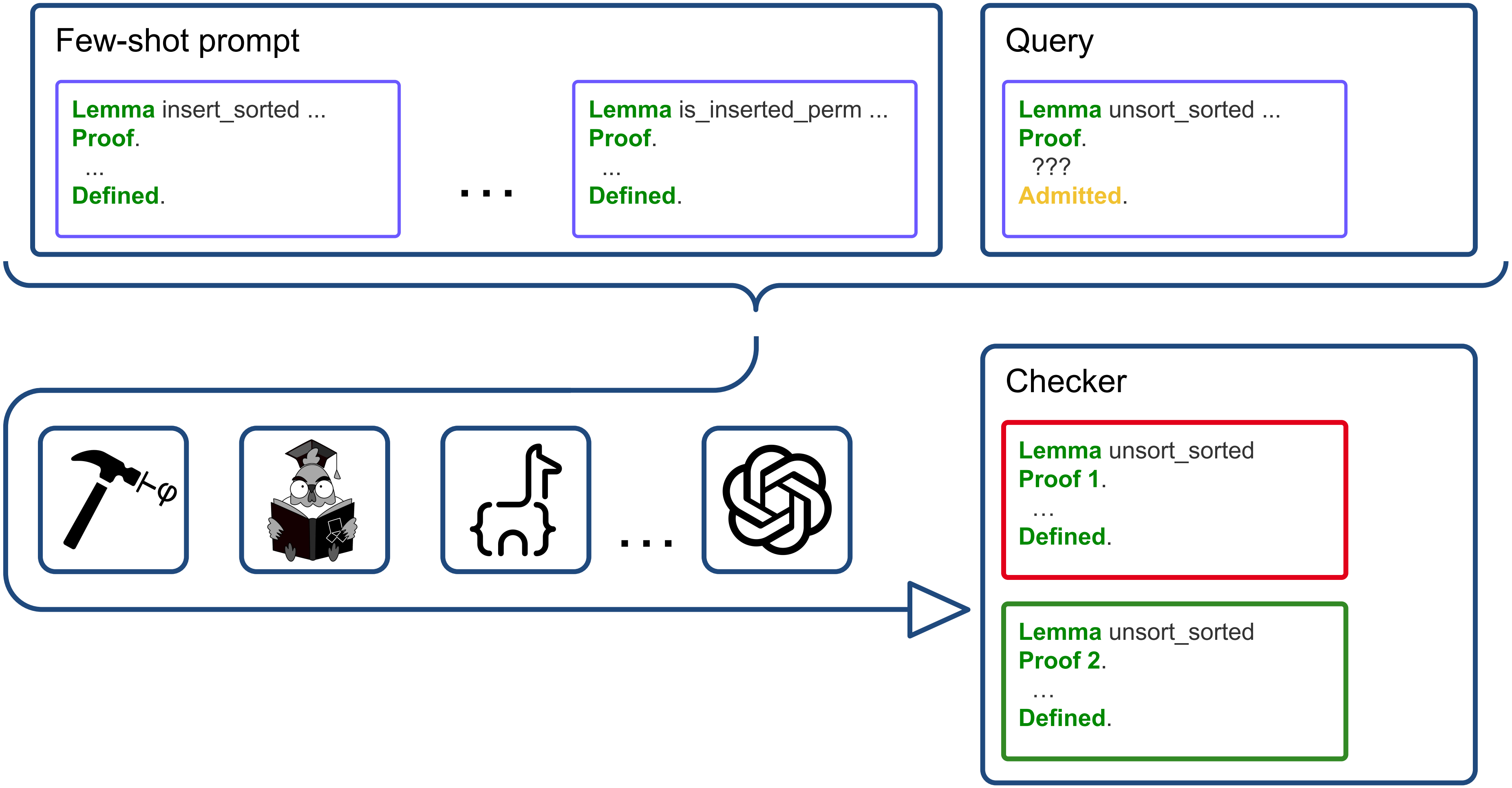
Few-shot prompt
Lemma insert_sorted ...
Proof.
...
Defined.
...
Lemma is_inserted_perm ...
Proof.
...
Defined.
Query
Lemma unsort_sorted ...
Proof.
???
Admitted.
⊢φ
...


Few-shot prompt
Lemma insert_sorted ...
Proof.
...
Defined.
. . .
Lemma is_inserted_perm ...
Proof.
...
Defined.
Query
Lemma unsort_sorted ...
Proof.
???
Admitted.
. . .


Few-shot prompt
Lemma insert_sorted ...
Proof.
...
Defined.
Lemma is_inserted_perm ...
Proof.
...
Defined.
Query
Lemma unsort_sorted ...
Proof.
???
Admitted.
Checker
Lemma unsort_sorted
Proof 1.
...
Defined.
Lemma unsort_sorted
Proof 2.
...
Defined.



Few-shot prompt
Lemma insert_sorted ...
Proof.
...
Defined.
Lemma is_inserted_perm ...
Proof.
...
Defined.
Query
Lemma unsort_sorted ...
Proof.
???
Admitted.
Checker
Lemma unsort_sorted
Proof 1.
...
Defined.
Lemma unsort_sorted
Proof 2.
...
Defined.


Few-shot prompt
Lemma insert_sorted ...
Proof.
...
Defined.
. . .
Lemma is_inserted_perm ...
Proof.
...
Defined.
Query
Lemma unsort_sorted ...
Proof.
???
Admitted.
. . .
Checker
Lemma unsort_sorted
Proof 1.
...
Defined.
Lemma unsort_sorted
Proof 2.
...
Defined.

s.v — coqpilotTests

s.v M × | Settings | settings.json | Coq Goals ×

src > s.v

```
Definition sort l : {l' | Permutation l l' & is_sorte
Proof.
  induction l.
  { admit. }
  destruct IHl as [l'].
  edestruct (insert_sorted a l') as [l''].
  { admit. }
  exists l''.
  2: { admit. }
  transitivity (a::l').
  { admit. }
  admit.
Admitted.

Eval compute in (p1 (sort [3;2;4;1])).
```

Proof

Main 1 Shelved 0 Given up 0

Goal 1

l : list nat

(1 / 1)

{l' : list nat | Permutation l l' & is_sorted l'}

Messages

main* 0↓ 3↑ ⊗ 0 ⚠ 0 ⓘ 1 0 vscoq-language-server 2.1.0, coq 8.19.0 Ln 106, Col 1 Spaces: 2 UTF-8 LF Coq Go Live Record Story (beta) Prettier

Research Questions

Research Questions

- **RQ1**: How well general purpose LLMs can write Coq proofs?

Research Questions

- **RQ1:** How well general purpose LLMs can write Coq proofs?
- **RQ2:** To which extent does CoqPilot improve the LLM approach to Coq generation?

Research **Questions**

- **RQ1**: How well general purpose LLMs can write Coq proofs?
- **RQ2**: To which extent does CoqPilot improve the LLM approach to Coq generation?
- **RQ3**: What is the additional value CoqPilot contributes to other Coq automation tools such as CoqHammer and Tactician?

Informal Evaluation

Informal Evaluation

Reference proof length	⩽ 4	5 – 8	9 – 20	Total
Group size	131	98	71	300
`firstorder auto with *`	11%	2%	1%	6%
OpenAI GPT-3.5	29%	17%	6%	20%
OpenAI GPT-4o	50%	26%	15%	34%
LLaMA-2 13B Chat	2%	0%	0%	0.5%
Anthropic Claude	21%	7%	7%	13%
All models together	57%	32%	18%	39%
Tactician	45%	23%	10%	29%
CoqHammer	23%	4%	0%	11%
All methods together	71%	45%	23%	51%

IMM Project: https://github.com/weakmemory/imm
CoqPilot ASE'24 Tool Demo paper on: https://podkopaev.net/

Informal Evaluation

Reference proof length	$\leqslant 4$	$5-8$	$9-20$	Total
Group size	131	98	71	300
`firstorder auto with *`	11%	2%	1%	6%
OpenAI GPT-3.5	29%	17%	6%	20%
OpenAI GPT-4o	50%	26%	15%	34%
LLaMA-2 13B Chat	2%	0%	0%	0.5%
Anthropic Claude	21%	7%	7%	13%
All models together	57%	32%	18%	39%
Tactician	45%	23%	10%	29%
CoqHammer	23%	4%	0%	11%
All methods together	71%	45%	23%	51%

IMM Project: https://github.com/weakmemory/imm
CoqPilot ASE'24 Tool Demo paper on: https://podkopaev.net/

Informal Evaluation

Reference proof length	⩽ 4	5 – 8	9 – 20	Total
Group size	131	98	71	300
`firstorder auto with *`	11%	2%	1%	6%
OpenAI GPT-3.5	29%	17%	6%	20%
OpenAI GPT-4o	50%	26%	15%	34%
LLaMA-2 13B Chat	2%	0%	0%	0.5%
Anthropic Claude	21%	7%	7%	13%
All models together	57%	32%	18%	39%
Tactician	45%	23%	10%	29%
CoqHammer	23%	4%	0%	11%
All methods together	71%	45%	23%	51%

IMM Project: https://github.com/weakmemory/imm
CoqPilot ASE'24 Tool Demo paper on: https://podkopaev.net/

Informal Evaluation

Reference proof length	$\leqslant 4$	$5-8$	$9-20$	Total
Group size	131	98	71	300
`firstorder auto with *`	11%	2%	1%	6%
OpenAI GPT-3.5	29%	17%	6%	20%
OpenAI GPT-4o	50%	26%	15%	34%
LLaMA-2 13B Chat	2%	0%	0%	0.5%
Anthropic Claude	21%	7%	7%	13%
All models together	57%	32%	18%	39%
Tactician	45%	23%	10%	29%
CoqHammer	23%	4%	0%	11%
All methods together	71%	45%	23%	51%

IMM Project: https://github.com/weakmemory/imm
CoqPilot ASE'24 Tool Demo paper on: https://podkopaev.net/

Improvement directions

Improvement directions

Integrate more Coq generation tools, such as Copra, Graph2Tac, TacTok etc.

Improvement directions

Improve premise selection and add new selection techniques

Integrate more Coq generation tools, such as Copra, Graph2Tac, TacTok etc.

Improvement directions

Improve premise selection and add new selection techniques

Currently used heuristics

Integrate more Coq generation tools, such as Copra, Graph2Tac, TacTok etc.

Improvement directions

Currently used heuristics

Improve premise selection and add new selection techniques

Distance from the target theorem

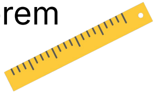
Integrate more Coq generation tools, such as Copra, Graph2Tac, TacTok etc.

Improvement directions

Improve premise selection and add new selection techniques

Currently used heuristics

Distance from the target theorem



Similarity of the goal

Integrate more Coq generation tools, such as Copra, Graph2Tac, TacTok etc.

Improvement directions

Improve premise selection and add new selection techniques

Explore and improve locally available models in order to make inference cheaper and preserve privacy

Integrate more Coq generation tools, such as Copra, Graph2Tac, TacTok etc.

Improvement directions

Improve premise selection and add new selection techniques

Integrate more Coq generation tools, such as Copra, Graph2Tac, TacTok etc.

Please talk to us if you have ideas!

CoqPilot: a plugin for LLM-based generation of proofs

JetBrains-Research/**coqpilot**

extension: **coqpilot**

Feedback
&
Collaboration

{andrei.kozyrev, gleb.solovev, nikita.khramov, anton.podkopaev}@jetbrains.com

Programming Languages and Program Analysis Lab (PLAN), JetBrains Research