

Proof Discovery as Monte Carlo Tree Search

A Walkthrough of Pantograph's Features

Leni Aniva

2025-04-25

Stanford University Centaur Lab / Project Trillium

Outline

Background	2	Proof in the Presentation View	15
Definitions	3	Proof in the Search View	16
Difficulty of MATP	4	Proof Verification	17
Expressions in Lean	5	Branch Unification	18
Proof Discovery in Lean	6	Frontend Features	19
A Non-branching Proof	7	Data Extraction	20
A Branching Proof	8	Incremental Tactic Execution	21
Reinforcement Learning	9	Drafting	22
Features	10	Experiments	23
Why Pantograph?	11	Draft-Sketch-Prove	24
The Three Views of a Proof	12	DSP Experiment Result	25
Detour: Motivated Proofs	13	Appendix	26
Handling of Metavariable Coupling	14	Demo	27

Outline

Background	2	Proof in the Presentation View	15
Definitions	3	Proof in the Search View	16
Difficulty of MATP	4	Proof Verification	17
Expressions in Lean	5	Branch Unification	18
Proof Discovery in Lean	6	Frontend Features	19
A Non-branching Proof	7	Data Extraction	20
A Branching Proof	8	Incremental Tactic Execution	21
Reinforcement Learning	9	Drafting	22
Features	10	Experiments	23
Why Pantograph?	11	Draft-Sketch-Prove	24
The Three Views of a Proof	12	DSP Experiment Result	25
Detour: Motivated Proofs	13	Appendix	26
Handling of Metavariable Coupling	14	Demo	27

Definitions

- **Machine-Assisted Theorem Proving:** Use machine learning and formal methods to automatically prove theorems.

^[1]L. de Moura and S. Ullrich, “The Lean 4 Theorem Prover and Programming Language,” in *Automated Deduction – CADE 28*, A. Platzer and G. Sutcliffe, Eds., Cham: Springer International Publishing, 2021, pp. 625–635.

Definitions

- **Machine-Assisted Theorem Proving:** Use machine learning and formal methods to automatically prove theorems.
- **Autoformalization:** Using machine learning to convert natural language informal mathematics into formal mathematics.

^[1]L. de Moura and S. Ullrich, “The Lean 4 Theorem Prover and Programming Language,” in *Automated Deduction – CADE 28*, A. Platzer and G. Sutcliffe, Eds., Cham: Springer International Publishing, 2021, pp. 625–635.

Definitions

- **Machine-Assisted Theorem Proving:** Use machine learning and formal methods to automatically prove theorems.
- **Autoformalization:** Using machine learning to convert natural language informal mathematics into formal mathematics.
- **Lean 4:** A proof assistant using the Calculus of Inductive Constructions type theory^[1].

^[1]L. de Moura and S. Ullrich, “The Lean 4 Theorem Prover and Programming Language,” in *Automated Deduction – CADE 28*, A. Platzer and G. Sutcliffe, Eds., Cham: Springer International Publishing, 2021, pp. 625–635.

Definitions

- **Machine-Assisted Theorem Proving:** Use machine learning and formal methods to automatically prove theorems.
- **Autoformalization:** Using machine learning to convert natural language informal mathematics into formal mathematics.
- **Lean 4:** A proof assistant using the Calculus of Inductive Constructions type theory^[1].
- **Pantograph:** A tool for training neural and hybrid theorem proving agents.

^[1]L. de Moura and S. Ullrich, “The Lean 4 Theorem Prover and Programming Language,” in *Automated Deduction – CADE 28*, A. Platzer and G. Sutcliffe, Eds., Cham: Springer International Publishing, 2021, pp. 625–635.

Difficulty of MATP

Difficulty of MATP

- The problem is general: SMT, Program Synthesis, Chess/Go, can all be represented as formal theorem proving problems.

Difficulty of MATP

- The problem is general: SMT, Program Synthesis, Chess/Go, can all be represented as formal theorem proving problems.
- The search space is unbounded and infinite. There is no algorithm that can solve it in the general case.

Difficulty of MATP

- The problem is general: SMT, Program Synthesis, Chess/Go, can all be represented as formal theorem proving problems.
- The search space is unbounded and infinite. There is no algorithm that can solve it in the general case.
- The unit of action (tactic) is unbounded. Tactics can be arbitrarily long.

Difficulty of MATP

- The problem is general: SMT, Program Synthesis, Chess/Go, can all be represented as formal theorem proving problems.
- The search space is unbounded and infinite. There is no algorithm that can solve it in the general case.
- The unit of action (tactic) is unbounded. Tactics can be arbitrarily long.
- Lack of data makes training large ML models difficult.

Difficulty of MATP

- The problem is general: SMT, Program Synthesis, Chess/Go, can all be represented as formal theorem proving problems.
- The search space is unbounded and infinite. There is no algorithm that can solve it in the general case.
- The unit of action (tactic) is unbounded. Tactics can be arbitrarily long.
- Lack of data makes training large ML models difficult.
- There is no unified agreed upon benchmark.

Difficulty of MATP

- The problem is general: SMT, Program Synthesis, Chess/Go, can all be represented as formal theorem proving problems.
- The search space is unbounded and infinite. There is no algorithm that can solve it in the general case.
- The unit of action (tactic) is unbounded. Tactics can be arbitrarily long.
- Lack of data makes training large ML models difficult.
- There is no unified agreed upon benchmark.
- There is no easy way for adversarial training of prover and problem generator.

Expressions in Lean

Everything (theorem, statement, proof, function) in Lean 4 is an expression.

Expressions in Lean

Everything (theorem, statement, proof, function) in Lean 4 is an expression.

A proof of tautology $\forall (p : \text{Prop}), p \rightarrow p$.

```
example :  $\forall (p : \text{Prop}) (h : p), p := \lambda p h => h$ 
```

lean

Expressions in Lean

Everything (theorem, statement, proof, function) in Lean 4 is an expression.

A proof of tautology $\forall (p : \text{Prop}), p \rightarrow p$.

```
example :  $\forall (p : \text{Prop}) (h : p), p := \lambda p h => h$ 
```

lean

A proof of the commutativity of logical or $\forall (p q : \text{Prop}), p \vee q \rightarrow q \vee p$:

```
example (p q : Prop) : p v q  $\rightarrow$  q v p :=
```

lean

```
   $\lambda$  hor => match hor with
```

```
    | .inl hp => .inr hp
```

```
    | .inr hq => .inl hq
```

Proof Discovery in Lean

It is hard to produce the entire proof expression in one shot.

Proof Discovery in Lean

It is hard to produce the entire proof expression in one shot.

- **Metavariable**: A variable with a given type which may or may not have an assigned value. Denoted by a question mark e.g. $?x$.

Proof Discovery in Lean

It is hard to produce the entire proof expression in one shot.

- **Metavariable**: A variable with a given type which may or may not have an assigned value. Denoted by a question mark e.g. $?x$.
- **Goal**: An unassigned metavariable.

Proof Discovery in Lean

It is hard to produce the entire proof expression in one shot.

- **Metavariable**: A variable with a given type which may or may not have an assigned value. Denoted by a question mark e.g. $?x$.
- **Goal**: An unassigned metavariable.
- **Tactic**: A function which assigns a value to the current goal(s), producing more goals in the process.

Proof Discovery in Lean

It is hard to produce the entire proof expression in one shot.

- **Metavariable**: A variable with a given type which may or may not have an assigned value. Denoted by a question mark e.g. $?x$.
- **Goal**: An unassigned metavariable.
- **Tactic**: A function which assigns a value to the current goal(s), producing more goals in the process.

After executing the `intro` tactic:

$$?1 : \vdash \forall (p q : \text{Prop}), p \vee q \rightarrow q \vee p$$

Proof Discovery in Lean

It is hard to produce the entire proof expression in one shot.

- **Metavariable:** A variable with a given type which may or may not have an assigned value. Denoted by a question mark e.g. $?x$.
- **Goal:** An unassigned metavariable.
- **Tactic:** A function which assigns a value to the current goal(s), producing more goals in the process.

After executing the `intro` tactic:

$$?1 : \vdash \forall (p q : \text{Prop}), p \vee q \rightarrow q \vee p$$

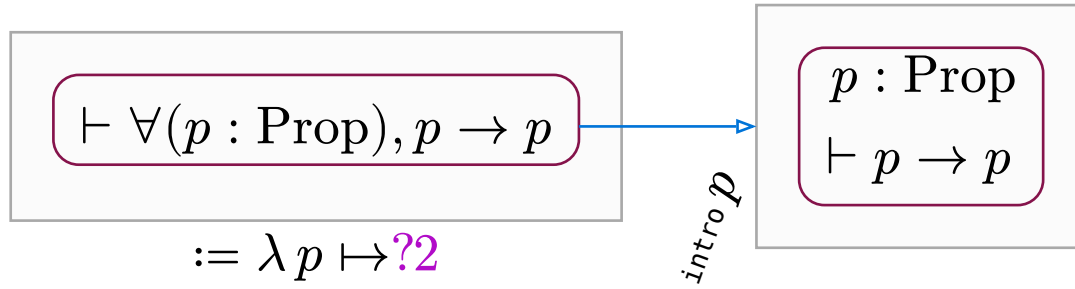
$$:= \lambda (p q : \text{Prop}), ?2[p, q]$$

$$?2 : (p q : \text{Prop}) \vdash p \vee q \rightarrow q \vee p$$

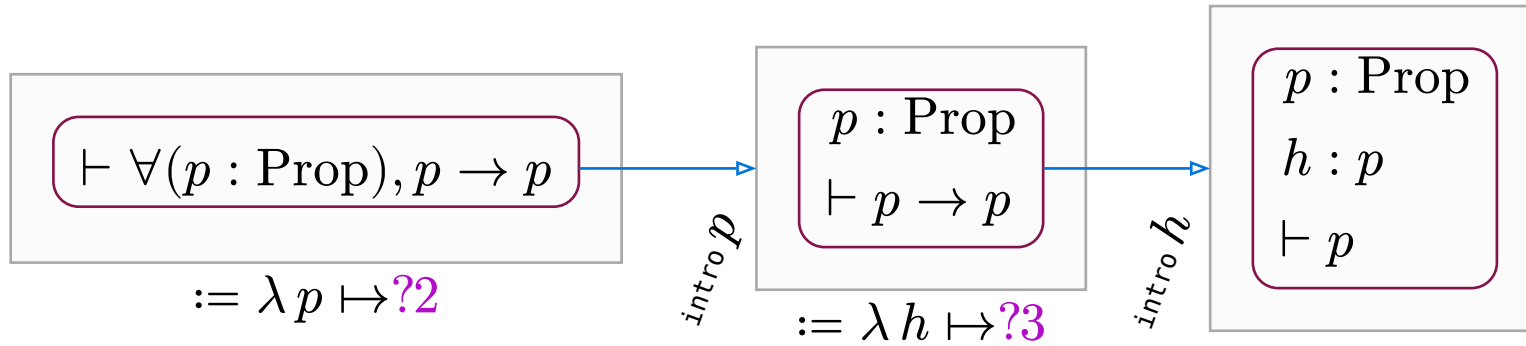
A Non-branching Proof

$$\vdash \forall (p : \text{Prop}), p \rightarrow p$$

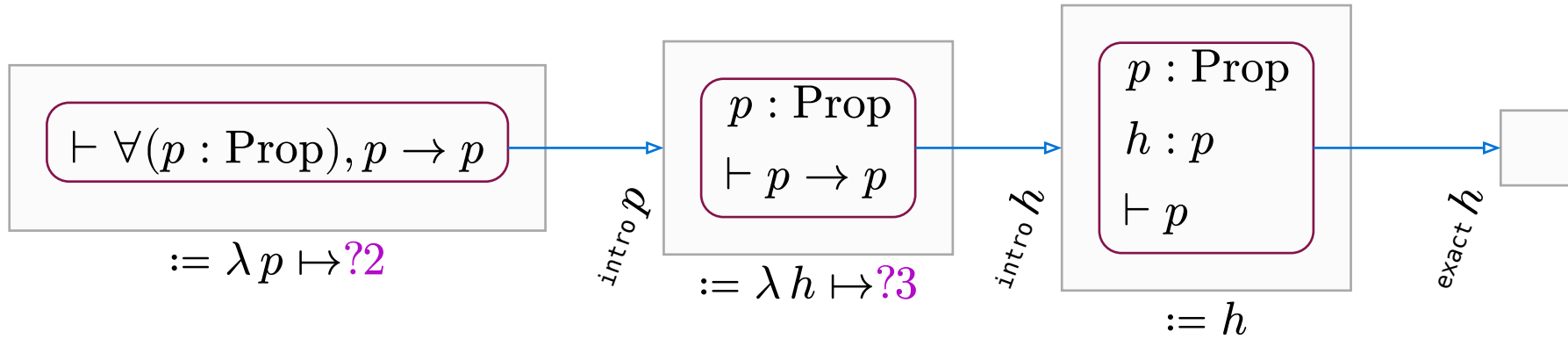
A Non-branching Proof



A Non-branching Proof



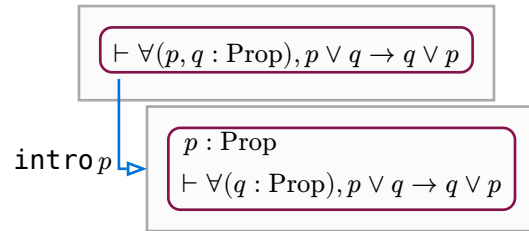
A Non-branching Proof



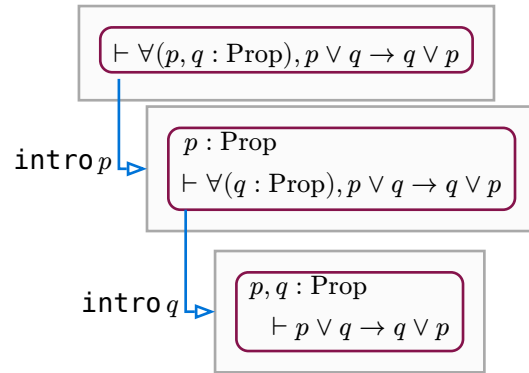
A Branching Proof

$$\vdash \forall (p, q : \text{Prop}), p \vee q \rightarrow q \vee p$$

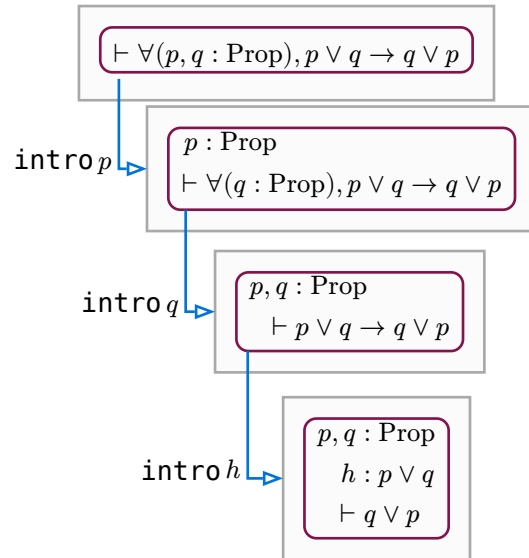
A Branching Proof



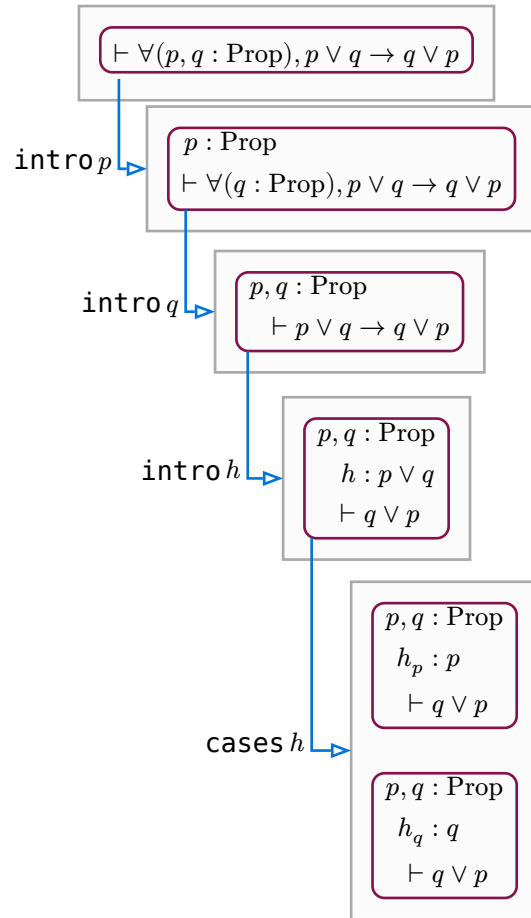
A Branching Proof



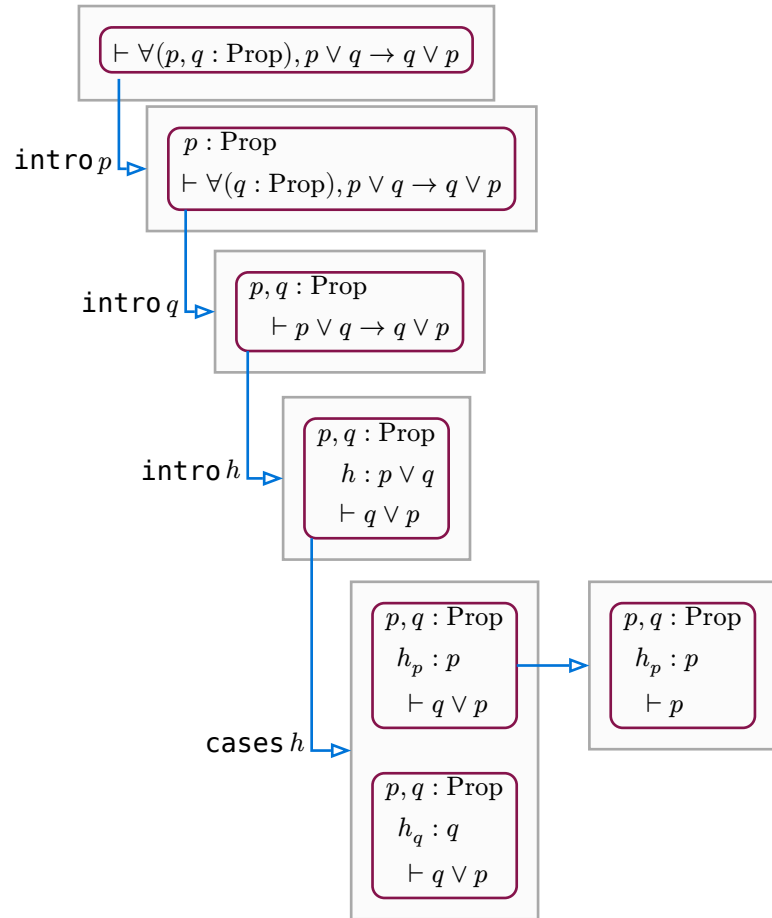
A Branching Proof



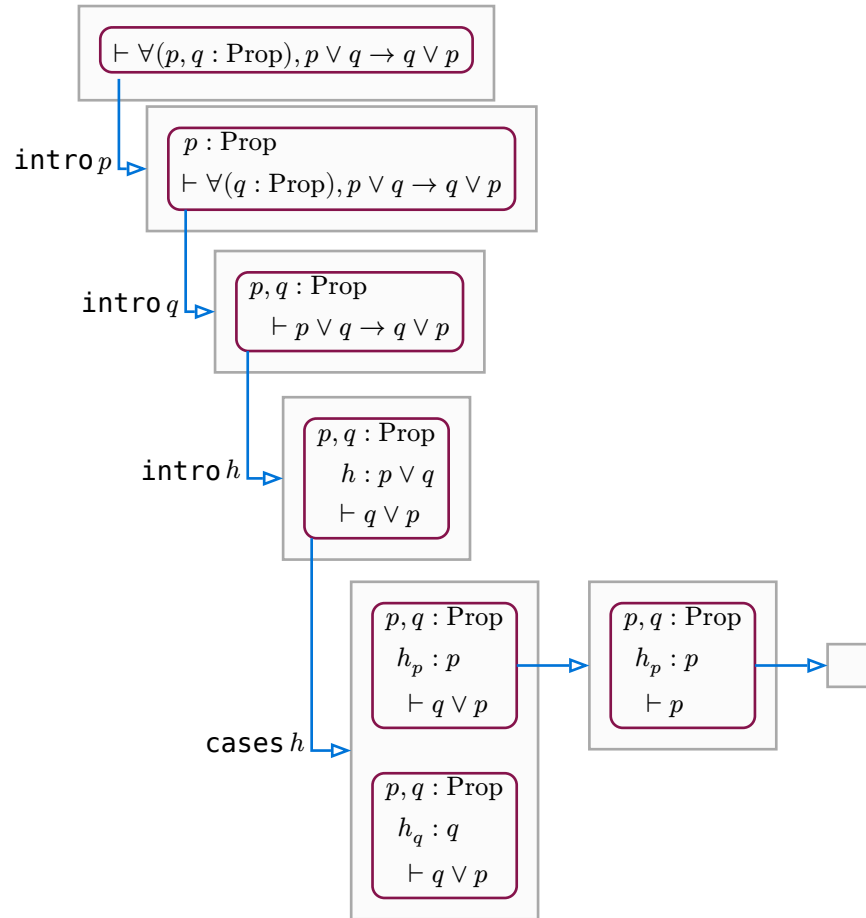
A Branching Proof



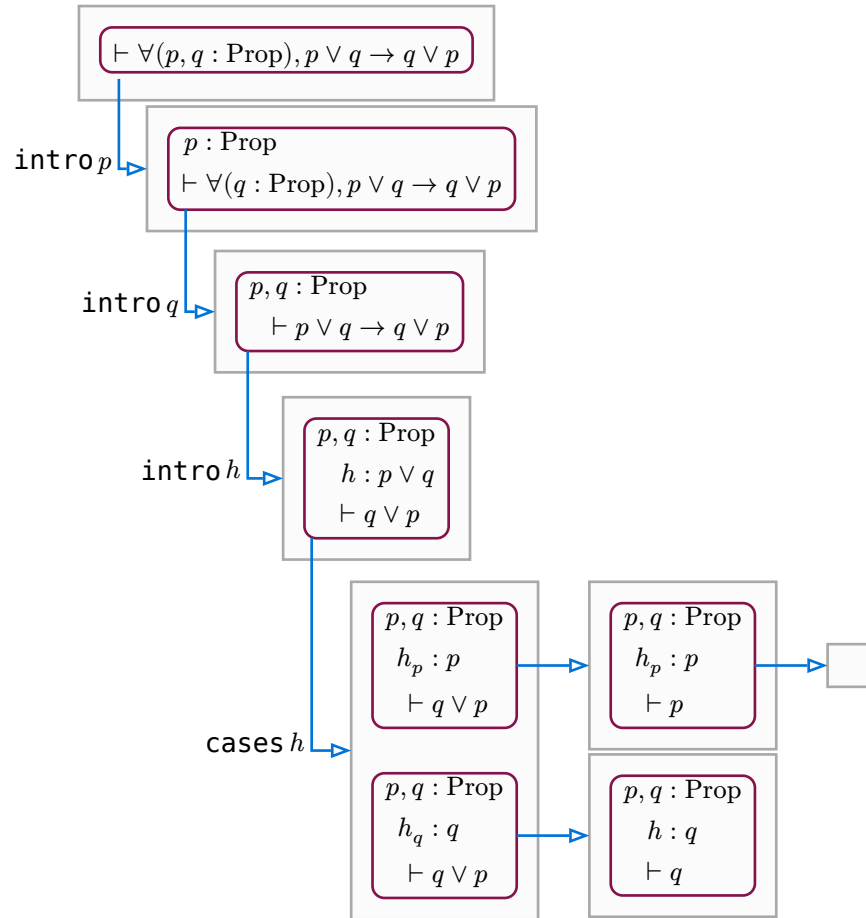
A Branching Proof



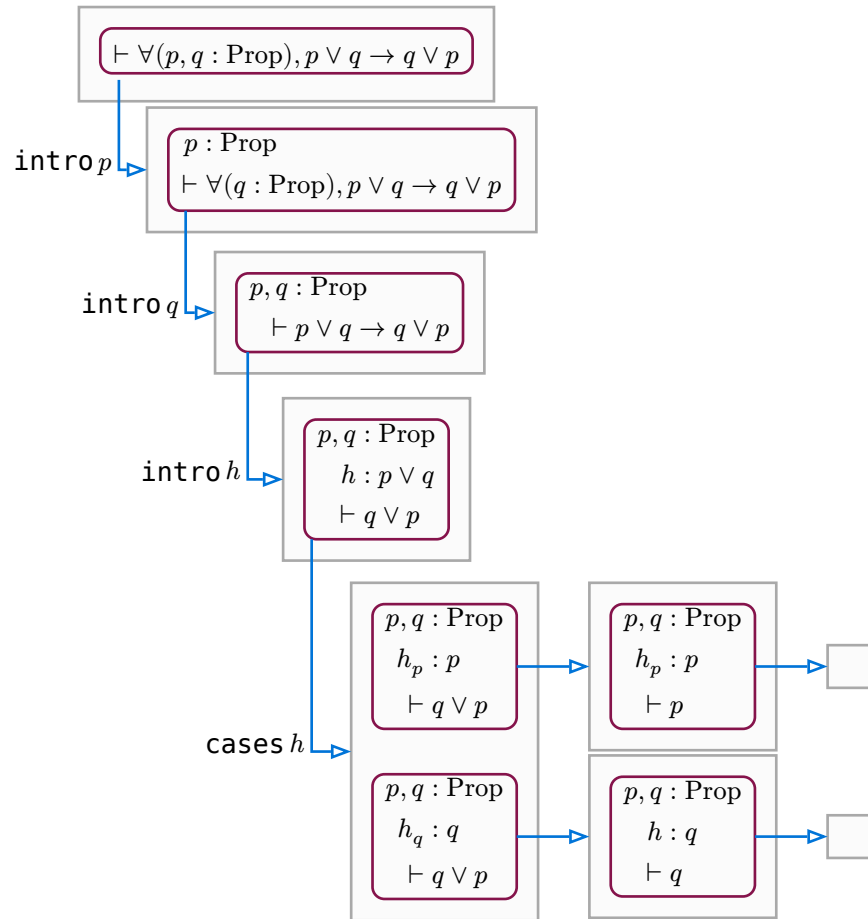
A Branching Proof



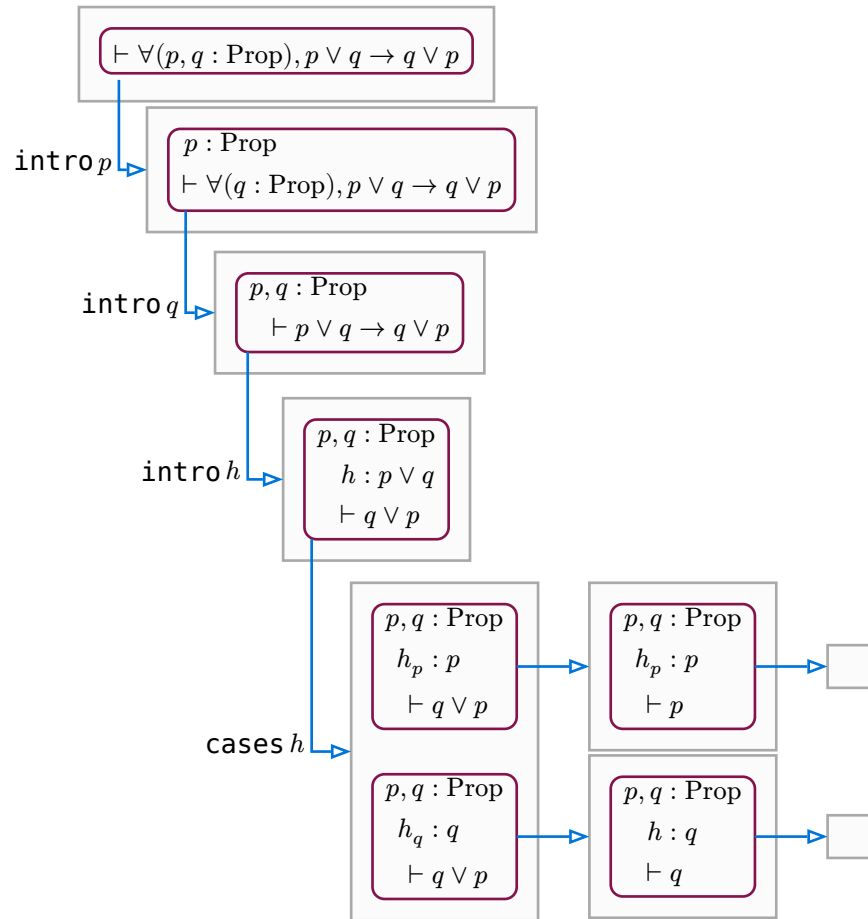
A Branching Proof



A Branching Proof



A Branching Proof



```
example :  $\forall (p q : \text{Prop}), p \vee q \rightarrow q \vee p := \text{by}$  lean  
  intro p q h  
  cases h  
  . apply Or.inr  
  assumption  
  . apply Or.inl  
  assumption
```

Reinforcement Learning



Figure 1: The action space is the set of tactics τ and the policy function π determines which goal to explore. This was popularized in Hypertree ^[1].

^[1]G. Lample *et al.*, “HyperTree Proof Search for Neural Theorem Proving.” 2022.

Reinforcement Learning

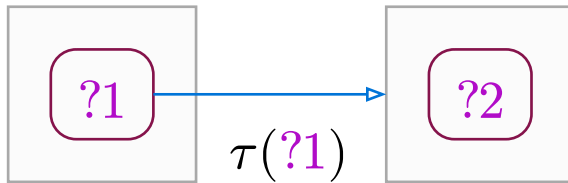


Figure 1: The action space is the set of tactics τ and the policy function π determines which goal to explore. This was popularized in Hypertree^[1].

^[1]G. Lample *et al.*, “HyperTree Proof Search for Neural Theorem Proving.” 2022.

Reinforcement Learning

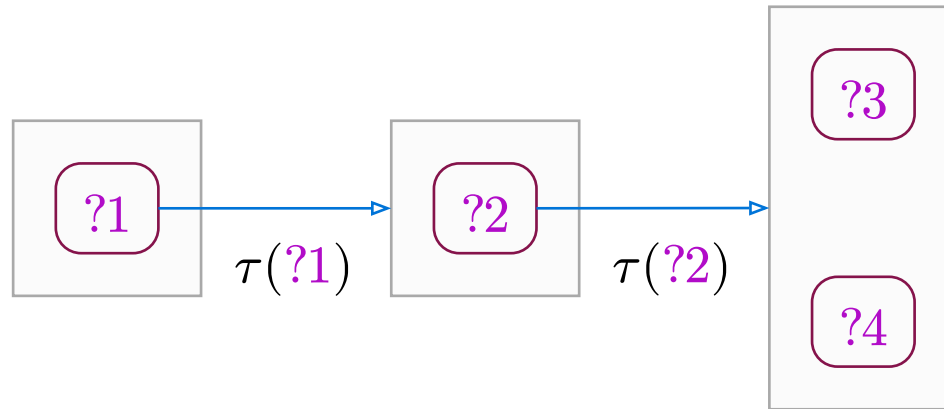


Figure 1: The action space is the set of tactics τ and the policy function π determines which goal to explore. This was popularized in Hypertree^[1].

^[1]G. Lample *et al.*, “HyperTree Proof Search for Neural Theorem Proving.” 2022.

Reinforcement Learning

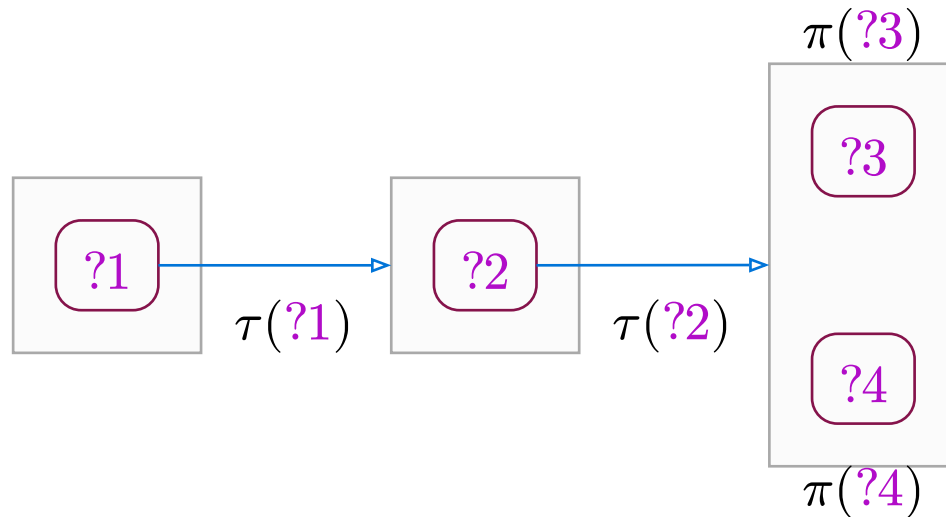


Figure 1: The action space is the set of tactics τ and the policy function π determines which goal to explore. This was popularized in Hypertree^[1].

^[1]G. Lample *et al.*, "HyperTree Proof Search for Neural Theorem Proving." 2022.

Reinforcement Learning

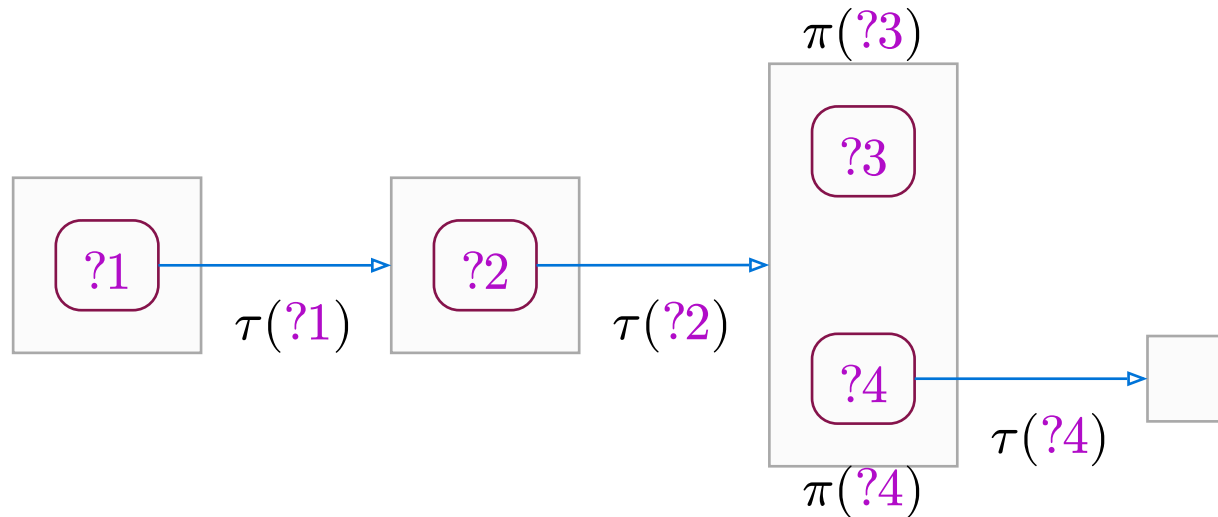


Figure 1: The action space is the set of tactics τ and the policy function π determines which goal to explore. This was popularized in Hypertree^[1].

^[1]G. Lample *et al.*, “HyperTree Proof Search for Neural Theorem Proving.” 2022.

Reinforcement Learning

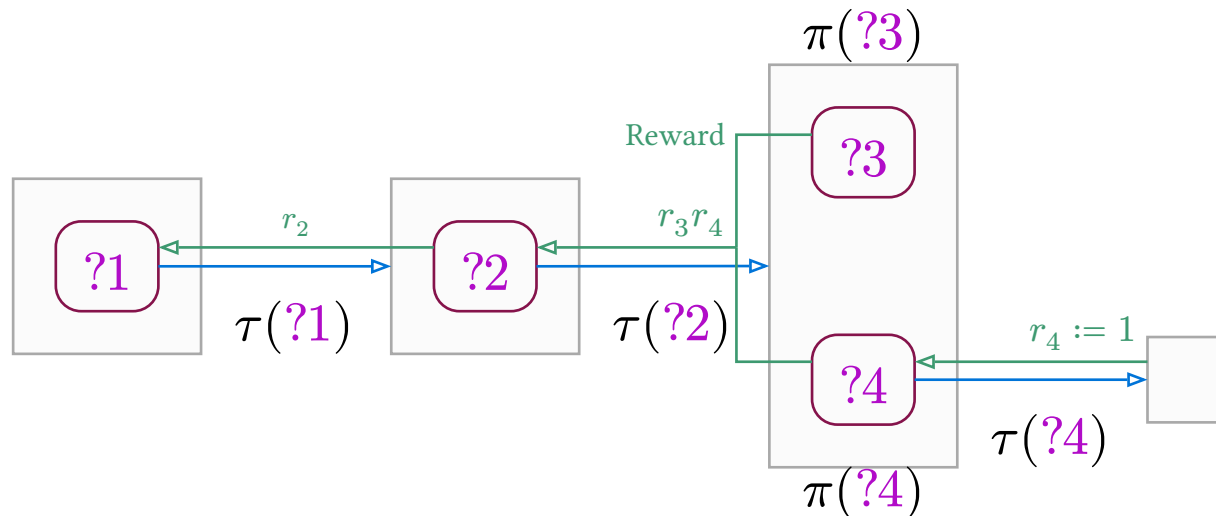


Figure 1: The action space is the set of tactics τ and the policy function π determines which goal to explore. This was popularized in Hypertree^[1].

^[1]G. Lample *et al.*, “HyperTree Proof Search for Neural Theorem Proving.” 2022.

Outline

Background	2	Proof in the Presentation View	15
Definitions	3	Proof in the Search View	16
Difficulty of MATP	4	Proof Verification	17
Expressions in Lean	5	Branch Unification	18
Proof Discovery in Lean	6	Frontend Features	19
A Non-branching Proof	7	Data Extraction	20
A Branching Proof	8	Incremental Tactic Execution	21
Reinforcement Learning	9	Drafting	22
Features	10	Experiments	23
Why Pantograph?	11	Draft-Sketch-Prove	24
The Three Views of a Proof	12	DSP Experiment Result	25
Detour: Motivated Proofs	13	Appendix	26
Handling of Metavariable Coupling	14	Demo	27

Why Pantograph?

Why Pantograph?

- Direct interaction with Lean LSP in an IDE requires a non-linear workflow e.g. moving the cursor, rewriting existing proofs. This is a problem of Lean Repl and Lean Dojo.

Why Pantograph?

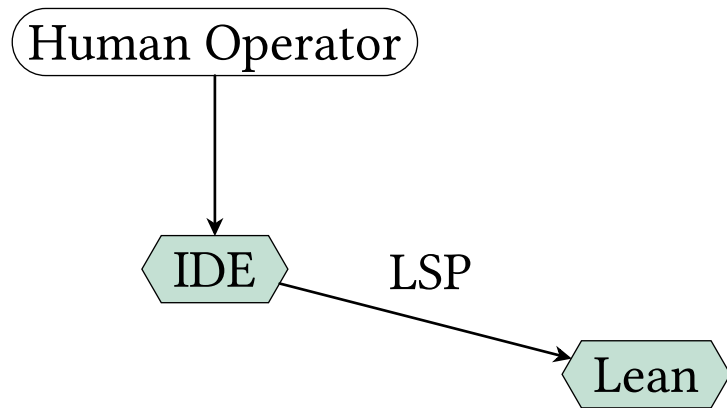
- Direct interaction with Lean LSP in an IDE requires a non-linear workflow e.g. moving the cursor, rewriting existing proofs. This is a problem of Lean Repl and Lean Dojo.
- Motivated proofs is possible

Why Pantograph?

- Direct interaction with Lean LSP in an IDE requires a non-linear workflow e.g. moving the cursor, rewriting existing proofs. This is a problem of Lean Repl and Lean Dojo.
- Motivated proofs is possible
- Tactics can have incremental feedback (`calc`, `conv`)

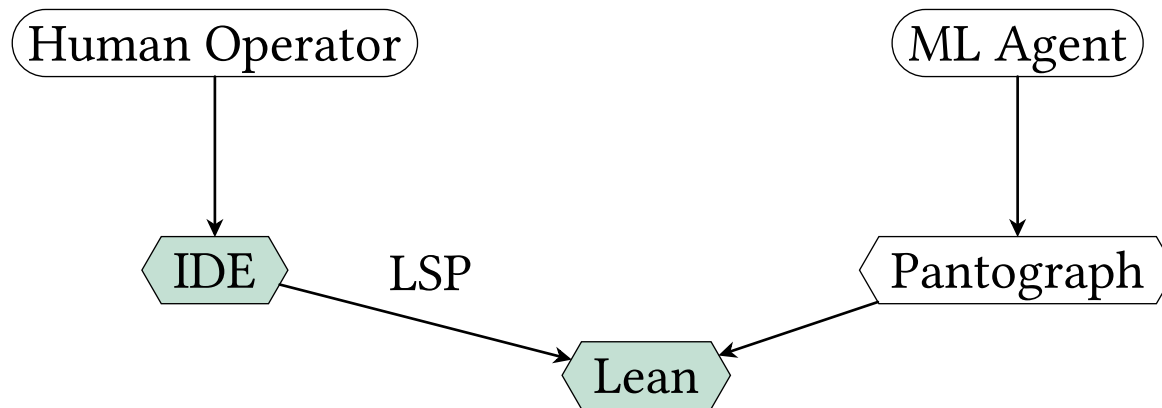
Why Pantograph?

- Direct interaction with Lean LSP in an IDE requires a non-linear workflow e.g. moving the cursor, rewriting existing proofs. This is a problem of Lean Repl and Lean Dojo.
- Motivated proofs is possible
- Tactics can have incremental feedback (`calc`, `conv`)



Why Pantograph?

- Direct interaction with Lean LSP in an IDE requires a non-linear workflow e.g. moving the cursor, rewriting existing proofs. This is a problem of Lean Repl and Lean Dojo.
- Motivated proofs is possible
- Tactics can have incremental feedback (`calc`, `conv`)



The Three Views of a Proof

The Three Views of a Proof

- **Presentation View:** A proof written down for presentation and verification. No metavariable coupling exists. It may contain “magical” values that are not apparent before reading the whole proof.

The Three Views of a Proof

- **Presentation View:** A proof written down for presentation and verification. No metavariable coupling exists. It may contain “magical” values that are not apparent before reading the whole proof.
- **Search View:** The trajectory a proof search agent traverses while finding the proof. It may contain metavariable coupling, backtracking, and resumption.

The Three Views of a Proof

- **Presentation View:** A proof written down for presentation and verification. No metavariable coupling exists. It may contain “magical” values that are not apparent before reading the whole proof.
- **Search View:** The trajectory a proof search agent traverses while finding the proof. It may contain metavariable coupling, backtracking, and resumption.
- **Kernel View:** The proof viewed as a set of metavariables and their assignments.

Detour: Motivated Proofs

Concept proposed by Pólya ^[1].

“Let $\varepsilon > 0$. Define $\delta := \min\left(2\varepsilon^2, \frac{\varepsilon}{5}, \frac{1}{3}\right), \dots$ ”

^[1]R. L. Morris, “Motivated Proofs: What They Are, Why They Matter and How to Write Them,” *The Review of Symbolic Logic*, vol. 13, no. 1, pp. 23–46, Nov. 2019, doi: [10.1017/s1755020319000583](https://doi.org/10.1017/s1755020319000583).

Detour: Motivated Proofs

Concept proposed by Pólya^[1].

“Let $\varepsilon > 0$. Define $\delta := \min\left(2\varepsilon^2, \frac{\varepsilon}{5}, \frac{1}{3}\right), \dots$ ”

Reader: How did you come up with this number?

^[1]R. L. Morris, “Motivated Proofs: What They Are, Why They Matter and How to Write Them,” *The Review of Symbolic Logic*, vol. 13, no. 1, pp. 23–46, Nov. 2019, doi: [10.1017/s1755020319000583](https://doi.org/10.1017/s1755020319000583).

Detour: Motivated Proofs

Concept proposed by Pólya^[1].

“Let $\varepsilon > 0$. Define $\delta := \min(2\varepsilon^2, \frac{\varepsilon}{5}, \frac{1}{3}), \dots$ ”

Reader: How did you come up with this number?

Motivated form:

“Let $\varepsilon > 0$. Let $\delta := \square$. Using lemma ..., we know $\delta \leq \frac{\varepsilon}{5}, \dots$ Therefore we can set $\delta := \min(\dots)$ ”

^[1]R. L. Morris, “Motivated Proofs: What They Are, Why They Matter and How to Write Them,” *The Review of Symbolic Logic*, vol. 13, no. 1, pp. 23–46, Nov. 2019, doi: [10.1017/s1755020319000583](https://doi.org/10.1017/s1755020319000583).

Handling of Metavariable Coupling

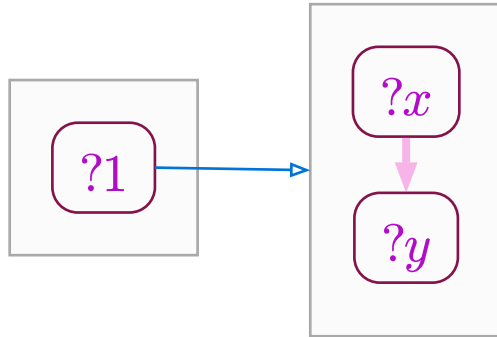
Aesop ^[1] handles coupling by copying.



^[1]J. Limperg and A. H. From, “Aesop: White-Box Best-First Proof Search for Lean,” in *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, in CPP 2023. Boston, MA, USA: Association for Computing Machinery, 2023, pp. 253–266. doi: [10.1145/3573105.3575671](https://doi.org/10.1145/3573105.3575671).

Handling of Metavariable Coupling

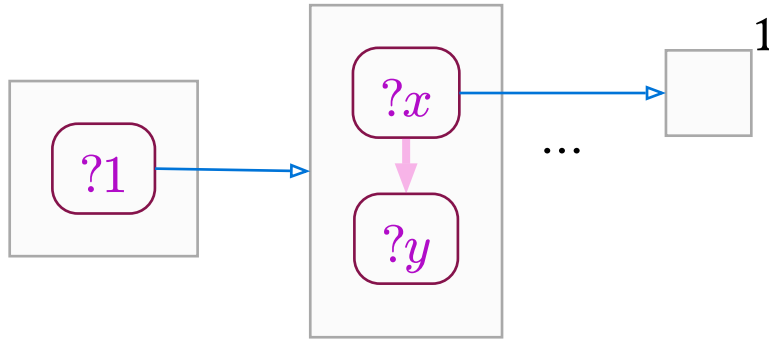
Aesop^[1] handles coupling by copying.



^[1]J. Limperg and A. H. From, “Aesop: White-Box Best-First Proof Search for Lean,” in *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, in CPP 2023. Boston, MA, USA: Association for Computing Machinery, 2023, pp. 253–266. doi: [10.1145/3573105.3575671](https://doi.org/10.1145/3573105.3575671).

Handling of Metavariable Coupling

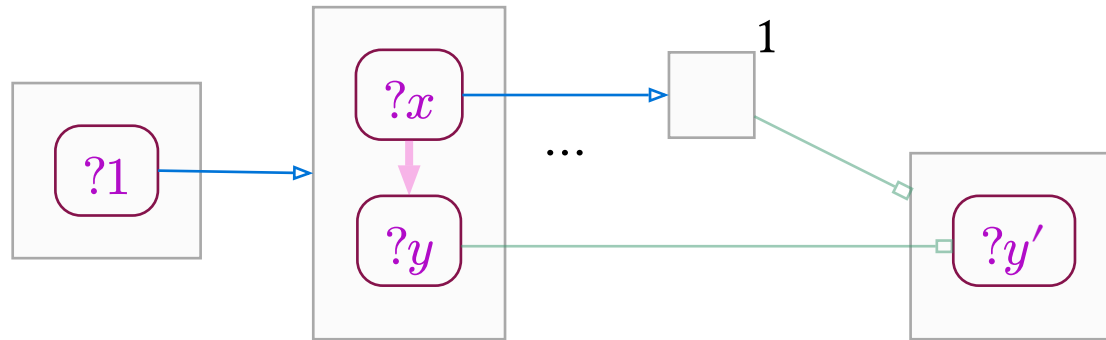
Aesop^[1] handles coupling by copying.



^[1]J. Limperg and A. H. From, “Aesop: White-Box Best-First Proof Search for Lean,” in *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, in CPP 2023. Boston, MA, USA: Association for Computing Machinery, 2023, pp. 253–266. doi: [10.1145/3573105.3575671](https://doi.org/10.1145/3573105.3575671).

Handling of Metavariable Coupling

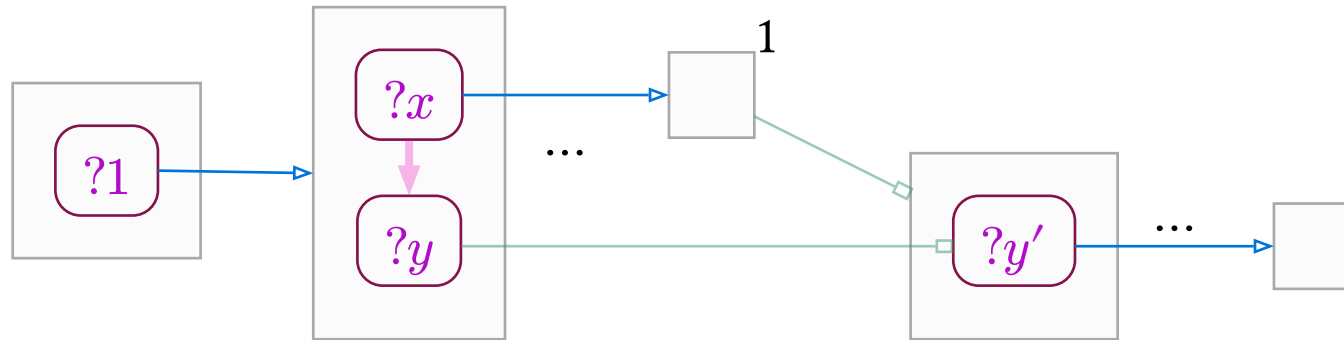
Aesop^[1] handles coupling by copying.



^[1]J. Limperg and A. H. From, “Aesop: White-Box Best-First Proof Search for Lean,” in *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, in CPP 2023. Boston, MA, USA: Association for Computing Machinery, 2023, pp. 253–266. doi: [10.1145/3573105.3575671](https://doi.org/10.1145/3573105.3575671).

Handling of Metavariable Coupling

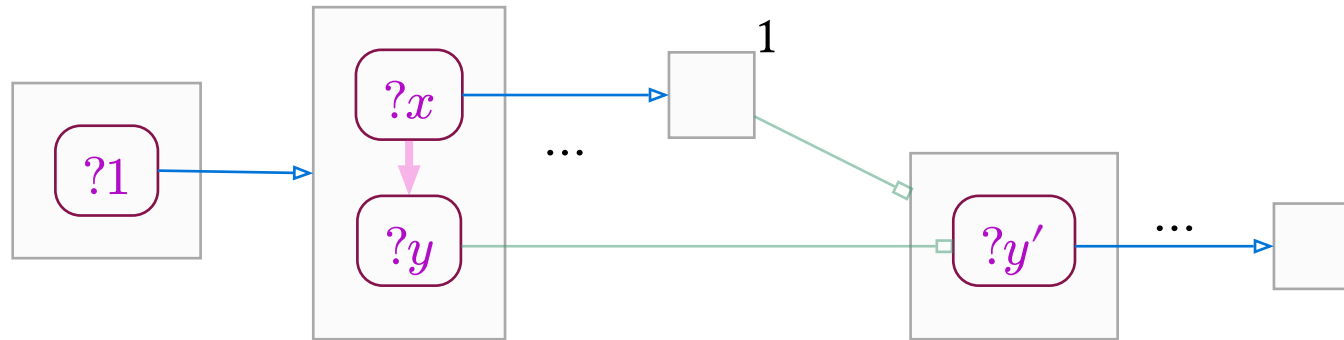
Aesop^[1] handles coupling by copying.



^[1]J. Limperg and A. H. From, “Aesop: White-Box Best-First Proof Search for Lean,” in *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, in CPP 2023. Boston, MA, USA: Association for Computing Machinery, 2023, pp. 253–266. doi: [10.1145/3573105.3575671](https://doi.org/10.1145/3573105.3575671).

Handling of Metavariable Coupling

Aesop^[1] handles coupling by copying.



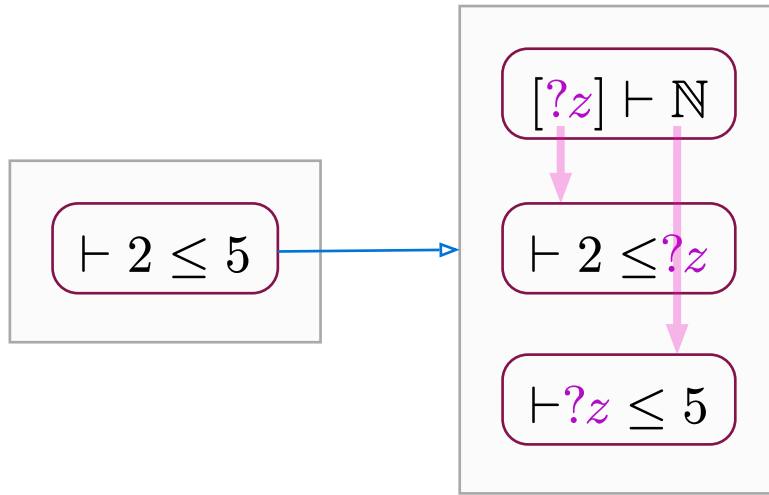
What if $?x : \mathbb{N}$ and $?y : \text{perfect}(?x)$? This will try random natural numbers until one is perfect.

^[1]J. Limperg and A. H. From, “Aesop: White-Box Best-First Proof Search for Lean,” in *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, in CPP 2023. Boston, MA, USA: Association for Computing Machinery, 2023, pp. 253–266. doi: [10.1145/3573105.3575671](https://doi.org/10.1145/3573105.3575671).

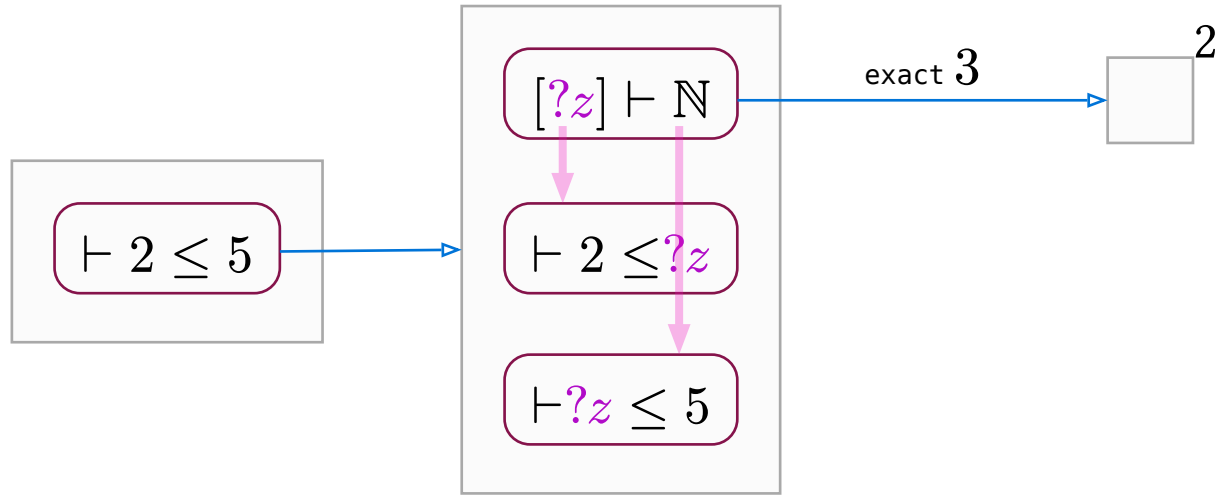
Proof in the Presentation View

$$\vdash 2 \leq 5$$

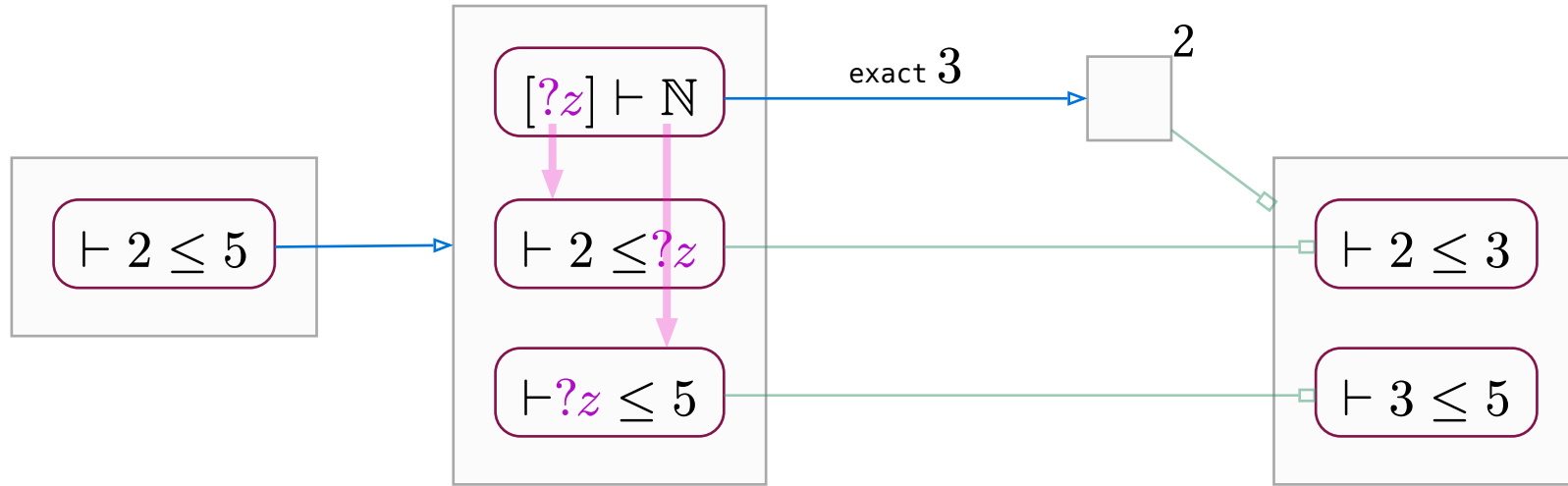
Proof in the Presentation View



Proof in the Presentation View



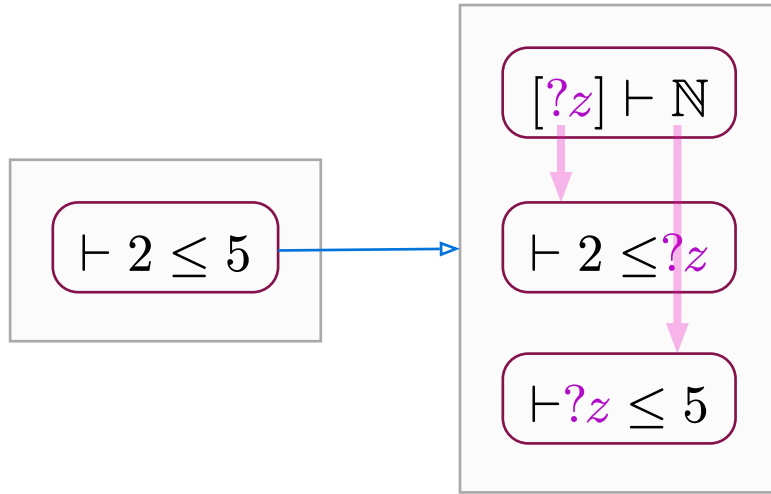
Proof in the Presentation View



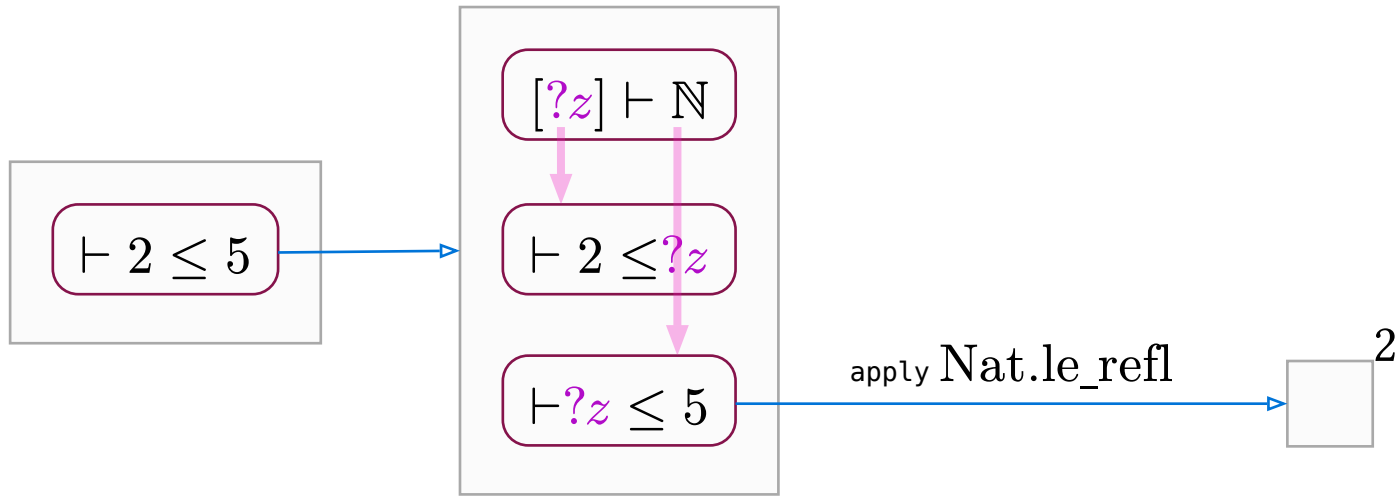
Proof in the Search View

$$\vdash 2 \leq 5$$

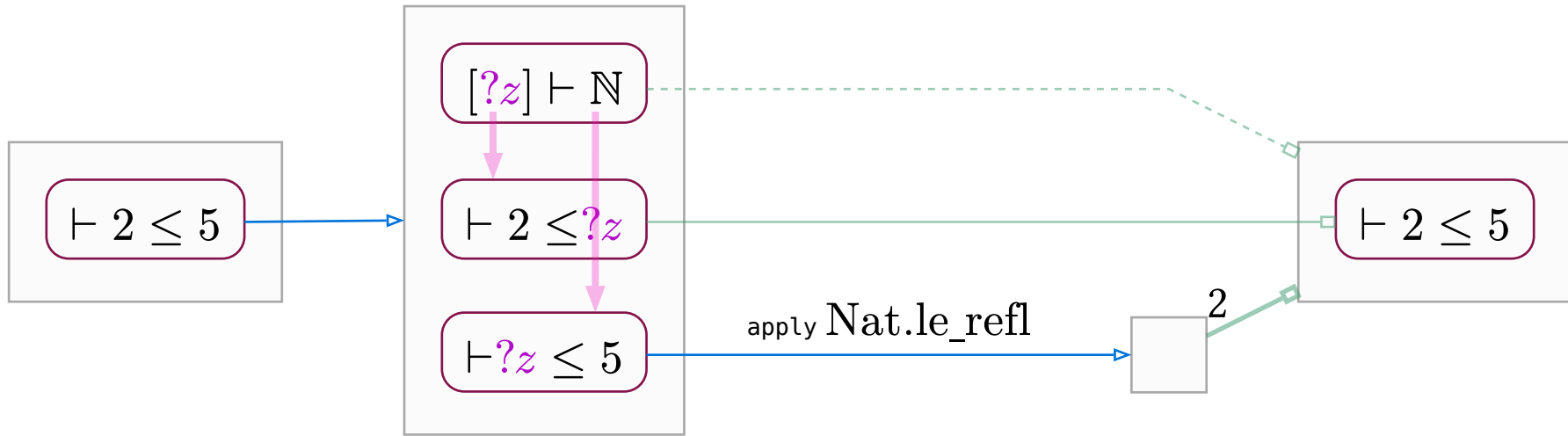
Proof in the Search View



Proof in the Search View



Proof in the Search View



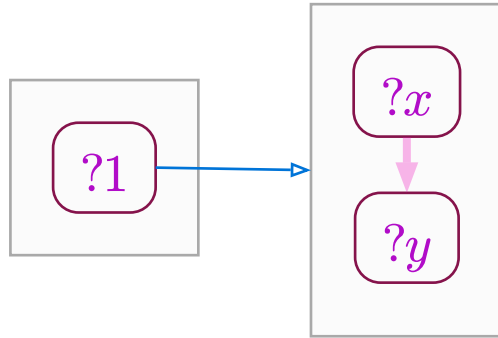
Proof Verification

Invoke the Lean kernel at the end of the proof to check its correctness



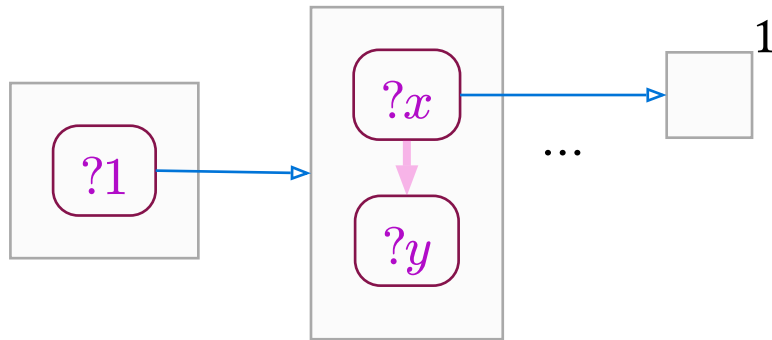
Proof Verification

Invoke the Lean kernel at the end of the proof to check its correctness



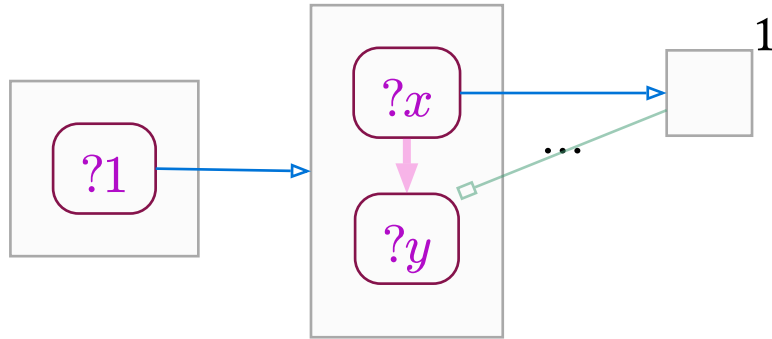
Proof Verification

Invoke the Lean kernel at the end of the proof to check its correctness



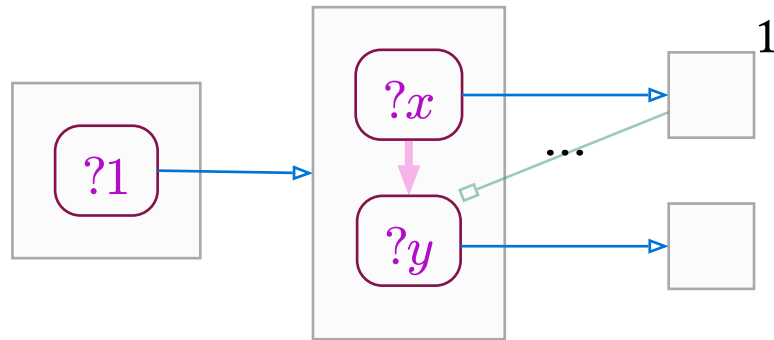
Proof Verification

Invoke the Lean kernel at the end of the proof to check its correctness



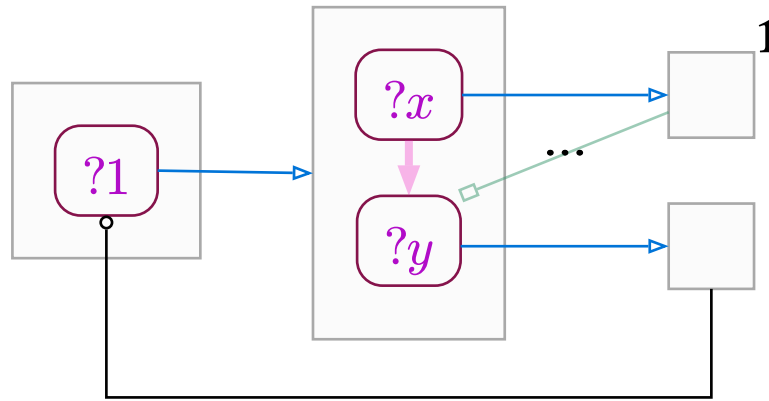
Proof Verification

Invoke the Lean kernel at the end of the proof to check its correctness



Proof Verification

Invoke the Lean kernel at the end of the proof to check its correctness



Kernel Check!

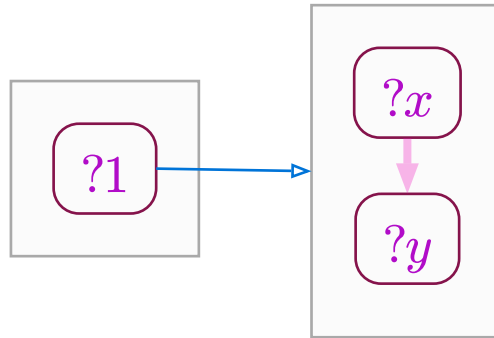
Branch Unification

Planned: Proof branches running on different machines can be unified.



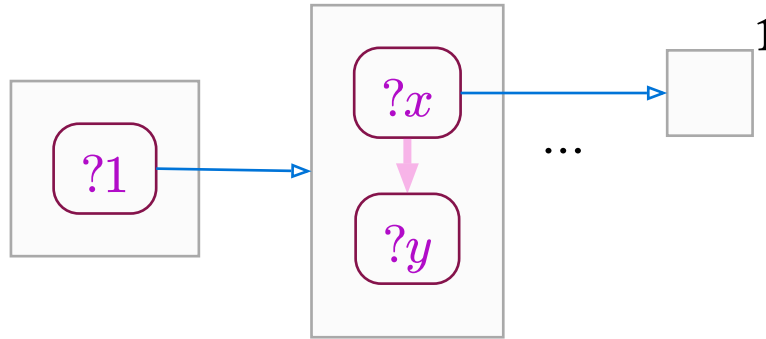
Branch Unification

Planned: Proof branches running on different machines can be unified.



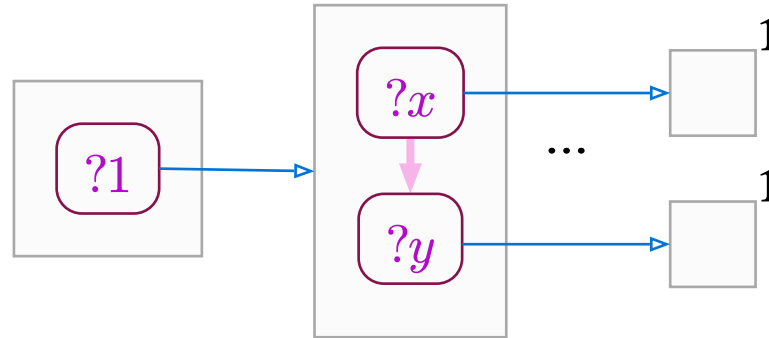
Branch Unification

Planned: Proof branches running on different machines can be unified.



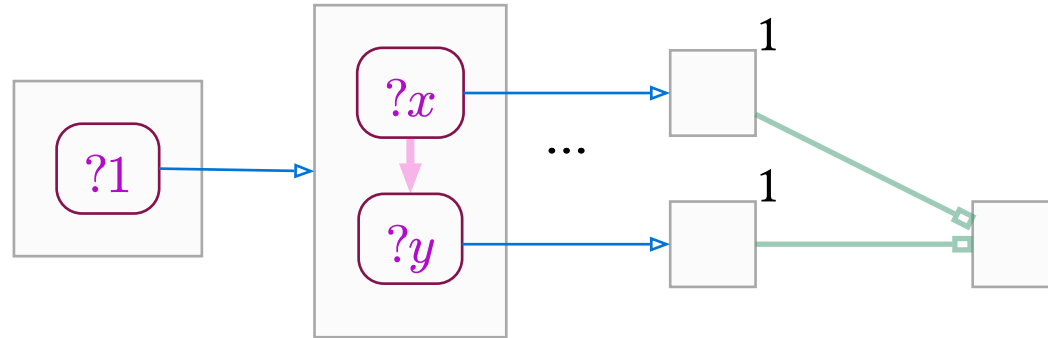
Branch Unification

Planned: Proof branches running on different machines can be unified.



Branch Unification

Planned: Proof branches running on different machines can be unified.



Frontend Features

Frontend Features

- **Data Extraction:** Pantograph can extract the before/after states of tactic invocations.

Frontend Features

- **Data Extraction:** Pantograph can extract the before/after states of tactic invocations.
- **Incremental Tactic Invocation:** Some tactics such as `conv` or `calc` provide incremental states. Pantograph can execute these tactics.

Frontend Features

- **Data Extraction:** Pantograph can extract the before/after states of tactic invocations.
- **Incremental Tactic Invocation:** Some tactics such as `conv` or `calc` provide incremental states. Pantograph can execute these tactics.
- **Drafting:** Extract `sorry`s and turn them into goals.

Data Extraction

```
example :  $\forall$  (p q: Prop), p v q  
→ q v p := by
```

lean

```
intro p q h
```

```
cases h
```

```
. apply Or.inr
```

```
assumption
```

```
. apply Or.inl
```

```
assumption
```

Before

```
p q : Prop
```

lean

```
h : p v q
```

```
⊢ q v p
```

After

```
p q : Prop
```

lean

```
h : p
```

```
⊢ q v p
```

```
p q : Prop
```

```
h : q
```

```
⊢ q v p
```

Incremental Tactic Execution

```
example (a b c : Nat) : a * (b  
* c) = a * (c * b) := by
```

lean

```
conv =>
```

lhs

```
congr
```

```
rfl
```

```
rw [Nat.mul_comm]
```

```
| a * (b * c)
```

lean

Drafting

```
theorem add_comm :  $\forall$  n m : Nat,  
n + m = m + n := by
```

lean

```
  intros n m
```

```
  induction n with
```

```
  | zero => sorry
```

```
  | succ n ih =>
```

```
    have h_succ: m + n.succ = (m +  
n).succ := sorry
```

```
  sorry
```

```
m : Nat
```

lean

```
⊢ 0 + m = m + 0
```

```
n m : Nat
```

lean

```
ih : m + n = n + m
```

```
⊢ m + n.succ = (m + n).succ
```

```
n m : Nat
```

lean

```
ih : m + n = n + m
```

```
h_succ : m + n.succ = (m +  
n).succ
```

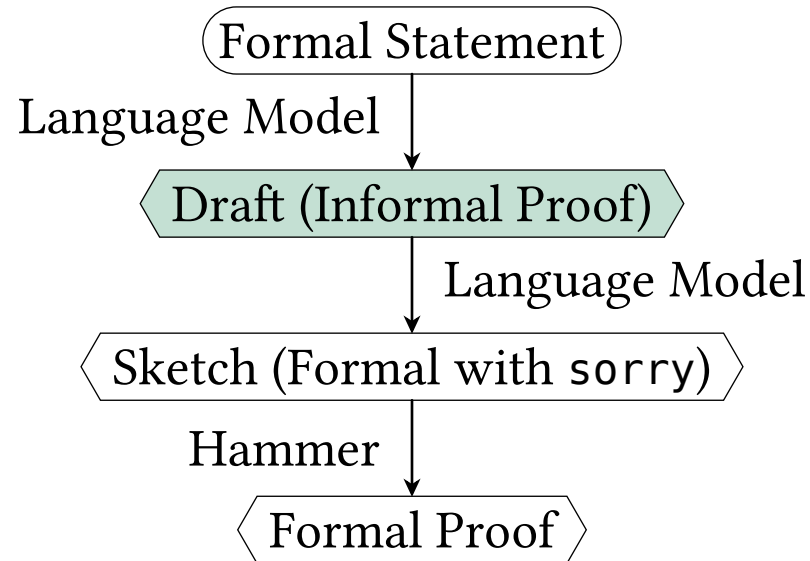
```
⊢ n + 1 + m = m + (n + 1)
```

Outline

Background	2	Proof in the Presentation View	15
Definitions	3	Proof in the Search View	16
Difficulty of MATP	4	Proof Verification	17
Expressions in Lean	5	Branch Unification	18
Proof Discovery in Lean	6	Frontend Features	19
A Non-branching Proof	7	Data Extraction	20
A Branching Proof	8	Incremental Tactic Execution	21
Reinforcement Learning	9	Drafting	22
Features	10	Experiments	23
Why Pantograph?	11	Draft-Sketch-Prove	24
The Three Views of a Proof	12	DSP Experiment Result	25
Detour: Motivated Proofs	13	Appendix	26
Handling of Metavariable Coupling	14	Demo	27

Draft-Sketch-Prove

Language-model based agents can generate a proof outline/sketch. This was introduced in Draft-Sketch-Prove ^[1].



^[1]A. e. a. Jiang, “Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs,” Oct. 2022, doi: [10.48550/arXiv.2210.12283](https://doi.org/10.48550/arXiv.2210.12283).

DSP Experiment Result

Results of running GPT-4o and GPT-o1 on the MiniF2F dataset, using Aesop as a hammer:

Benchmark Set	Validation			Test		
Sketches (Model)	1 (4o)	1 (o1)	3 (4o)	1 (4o)	1 (o1)	3 (4o)
Success Rate (%)	12.7	10.9	23.6	14.7	16.0	28.4
Hammer Invocations	4.17	5.38	7.93	4.46	5.72	7.34
Runtime (s)	17.25	73.23	23.98	28.39	88.38	36.41

Outline

Background	2	Proof in the Presentation View	15
Definitions	3	Proof in the Search View	16
Difficulty of MATP	4	Proof Verification	17
Expressions in Lean	5	Branch Unification	18
Proof Discovery in Lean	6	Frontend Features	19
A Non-branching Proof	7	Data Extraction	20
A Branching Proof	8	Incremental Tactic Execution	21
Reinforcement Learning	9	Drafting	22
Features	10	Experiments	23
Why Pantograph?	11	Draft-Sketch-Prove	24
The Three Views of a Proof	12	DSP Experiment Result	25
Detour: Motivated Proofs	13	Appendix	26
Handling of Metavariable Coupling	14	Demo	27

Demo

<https://github.com/stanford-centaur/PyPantograph/tree/doc/demo>

1. Clone Mathlib and tag to v4.18.0
2. Execute `lake exe cache get` in Mathlib directory
3. Execute `poetry build; poetry install` in PyPantograph directory
4. Execute `poetry run jupyter notebook` in PyPantograph directory
5. Navigate to `doc/Demo.ipynb`.