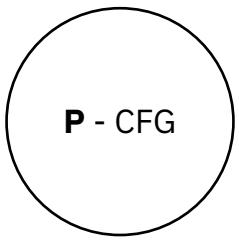
Navigating Proof Search in Theorem Provers

A Brief Tour of ATP in Type Theory

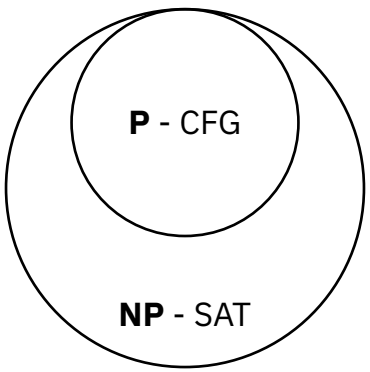
Presented by Yanning Chen @ ProSE Seminar

ATP, as a Complexity Problem



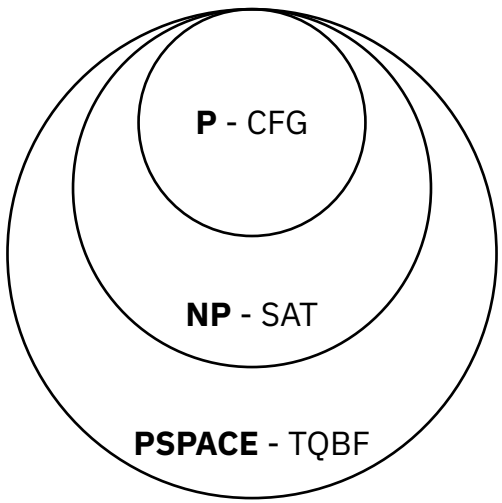
[1] https://www.youtube.com/watch?v=VAHyU6gHDdk

ATP, as a Complexity Problem



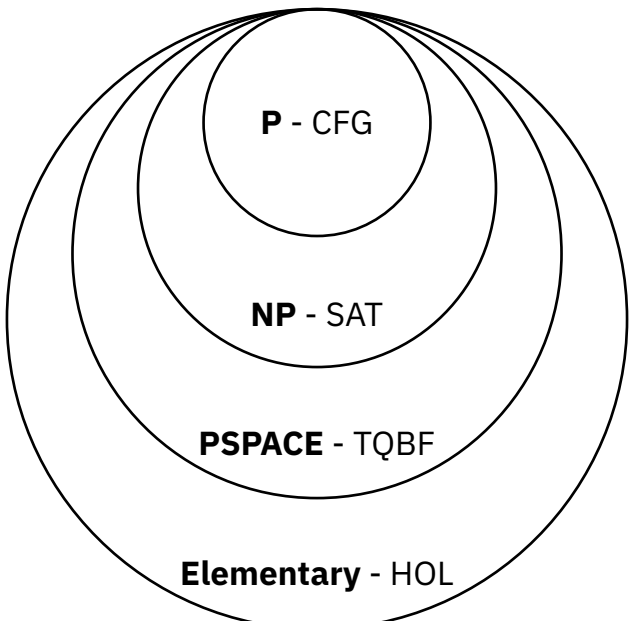
[1] https://www.youtube.com/watch?v=VAHyU6gHDdk

ATP, as a Complexity Problem



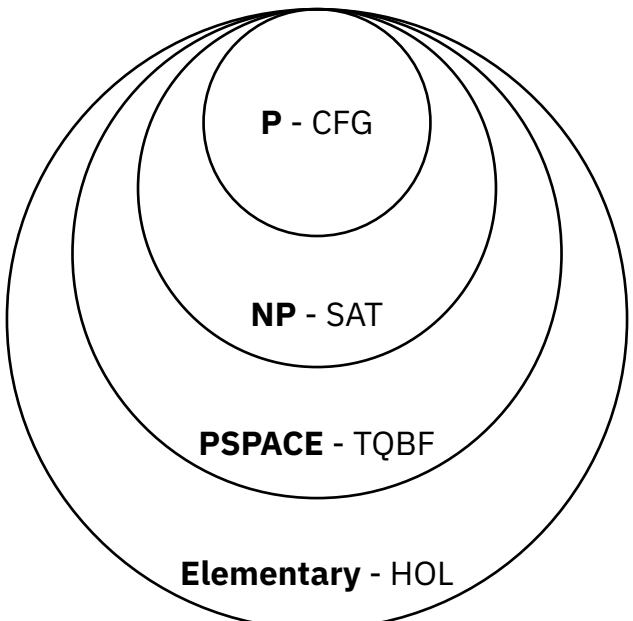
[1] https://www.youtube.com/watch?v=VAHyU6gHDdk

ATP, as a Complexity Problem



[1] https://www.youtube.com/watch?v=VAHyU6gHDdk

ATP, as a Complexity Problem



1

RE - General ATP

[1] https://www.youtube.com/watch?v=VAHyU6gHDdk

ATP, as a PL Problem

The good old C.H. Correspondence:

Logic Side	**Programming Side**
proposition	type
proof	term
proposition is true	type has an inhabitant
proposition is false	type does not have an inhabitant
proving a proposition	**finding an inhabitant**

ATP, as a PL Problem

The good old C.H. Correspondence:

Logic Side	**Programming Side**
proposition	type
proof	term
proposition is true	type has an inhabitant
proposition is false	type does not have an inhabitant
proving a proposition	**finding an inhabitant**

ATP is the **type-inhabitation problem**:

Given a proposition P, find a term of type P.

How humans prove (in Lean4)

```
inductive Or (a b : Prop) : Prop
| inl : a → a ∨ b
| inr : b → a ∨ b
```

How humans prove (in Lean4)

```
Or.inl : ∀ {a b : Prop}, a → a ∨ b
Or.inr : ∀ {a b : Prop}, b → a ∨ b
Or.elim : ∀ {a b c : Prop}, a ∨ b → (a → c) → (b → c) → c
```

How humans prove (in Lean4)

```
Or.inl : ∀ {a b : Prop}, a → a ∨ b
Or.inr : ∀ {a b : Prop}, b → a ∨ b
Or.elim : ∀ {a b c : Prop}, a ∨ b → (a → c) → (b → c) → c
```

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  ?_
```

How humans prove (in Lean4)

```
Or.inl : ∀ {a b : Prop}, a → a ∨ b
Or.inr : ∀ {a b : Prop}, b → a ∨ b
Or.elim : ∀ {a b c : Prop}, a ∨ b → (a → c) → (b → c) → c
```

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => ?_
```

How humans prove (in Lean4)

```
Or.inl : ∀ {a b : Prop}, a → a ∨ b
Or.inr : ∀ {a b : Prop}, b → a ∨ b
Or.elim : ∀ {a b c : Prop}, a ∨ b → (a → c) → (b → c) → c
```

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim hor ?_ ?_
```

How humans prove (in Lean4)

```
Or.inl : ∀ {a b : Prop}, a → a ∨ b
Or.inr : ∀ {a b : Prop}, b → a ∨ b
Or.elim : ∀ {a b c : Prop}, a ∨ b → (a → c) → (b → c) → c
```

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim hor
    (λ hp => ?_)
    (λ hq => ?_)
```

How humans prove (in Lean4)

```
Or.inl : ∀ {a b : Prop}, a → a ∨ b
Or.inr : ∀ {a b : Prop}, b → a ∨ b
Or.elim : ∀ {a b c : Prop}, a ∨ b → (a → c) → (b → c) → c
```

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim hor
    (λ hp => Or.inr ?_)
    (λ hq => Or.inl ?_)
```

How humans prove (in Lean4)

```
Or.inl : ∀ {a b : Prop}, a → a ∨ b
Or.inr : ∀ {a b : Prop}, b → a ∨ b
Or.elim : ∀ {a b c : Prop}, a ∨ b → (a → c) → (b → c) → c
```

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim hor
    (λ hp => Or.inr hp)
    (λ hq => Or.inl hq)
```

What are these “holes”?

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim h
    (λ hp => Or.inr ?1)
    (λ hq => Or.inl ?2)
```

What are these “holes”?

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim h
    (λ hp => Or.inr ?1)
    (λ hq => Or.inl ?2)
```

- **Metavariable**: A meta-level placeholder for an object-level term. It might have an assigned value (`?1 = hp`), partially assigned (`?x [a ↦ 1]`), or completely unknown.
- **Goal**: Any metavariable that’s not completely assigned.

What are these “holes”?

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim h
    (λ hp => Or.inr ?1)
    (λ hq => Or.inl ?2)
```

$$?1 : \forall (P\ Q : \text{Prop}), h_{or} : P \vee Q, h_p : P \vdash P$$

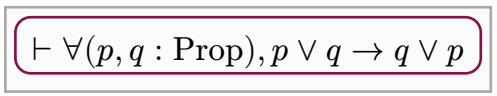
$$?2 : \forall (P\ Q : \text{Prop}), h_{or} : P \vee Q, h_q : Q \vdash Q$$

Local Context: Available premises and the target type for this hole.

How humans prove, alternatively

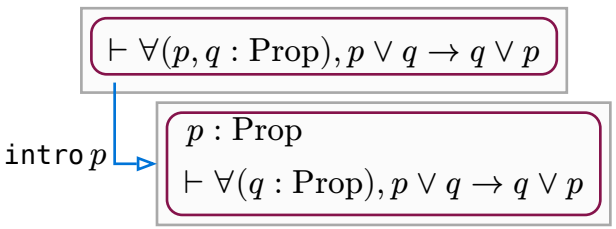
```
example : ∀ (P Q: Prop), P ∨ Q → Q ∨ P := by
  intro p q h
  cases h
  . case inl hp =>
    right
    exact hp
  . case inr hq =>
    left
    exact hq
```

How humans prove, alternatively



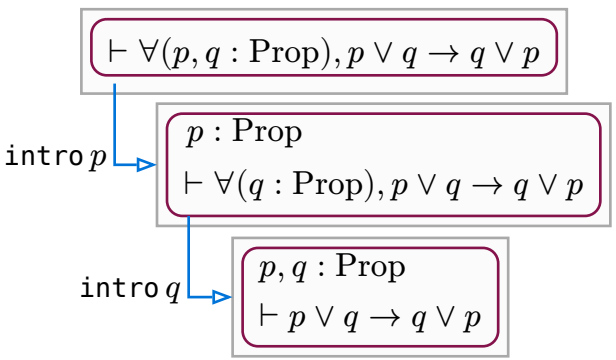

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  ?_
```

How humans prove, alternatively



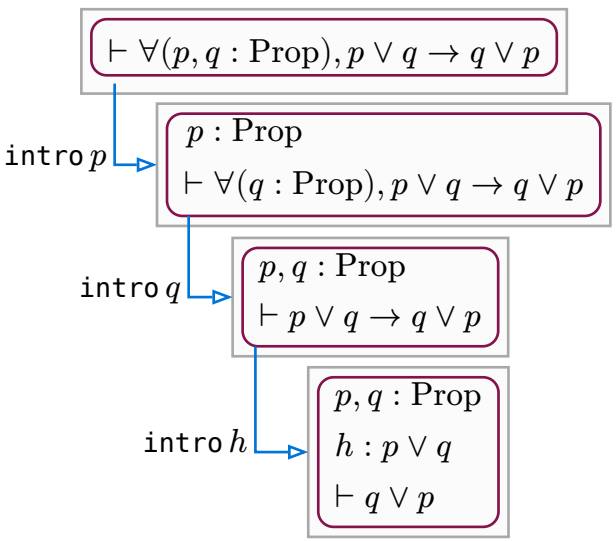

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P => ?_
```

How humans prove, alternatively



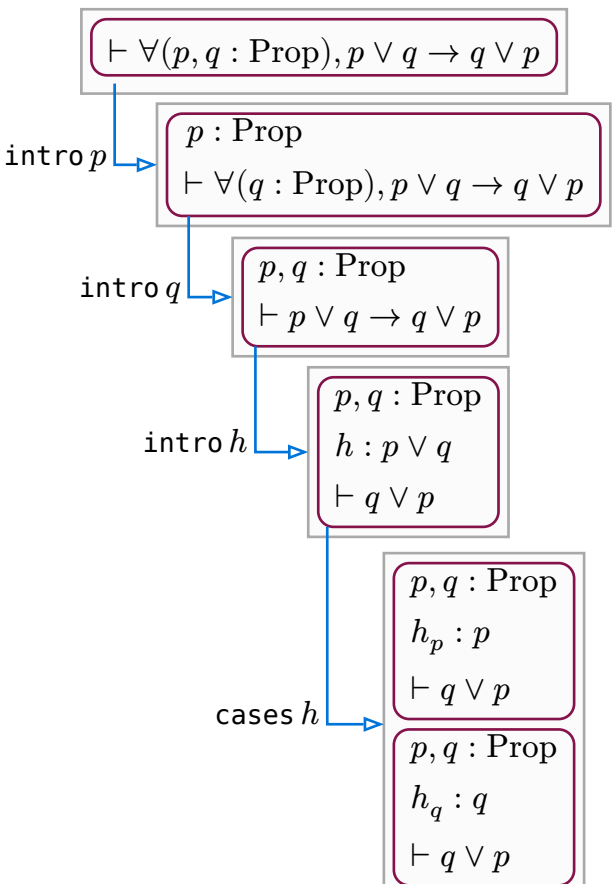

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q => ?_
```

How humans prove, alternatively



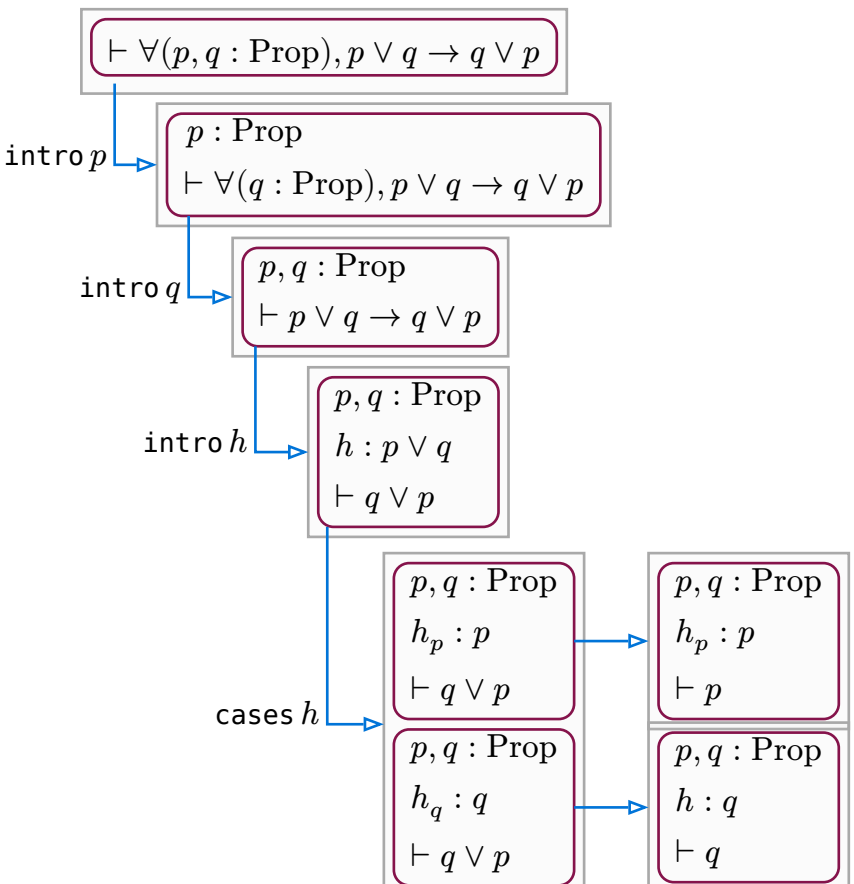

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => ?_
```

How humans prove, alternatively



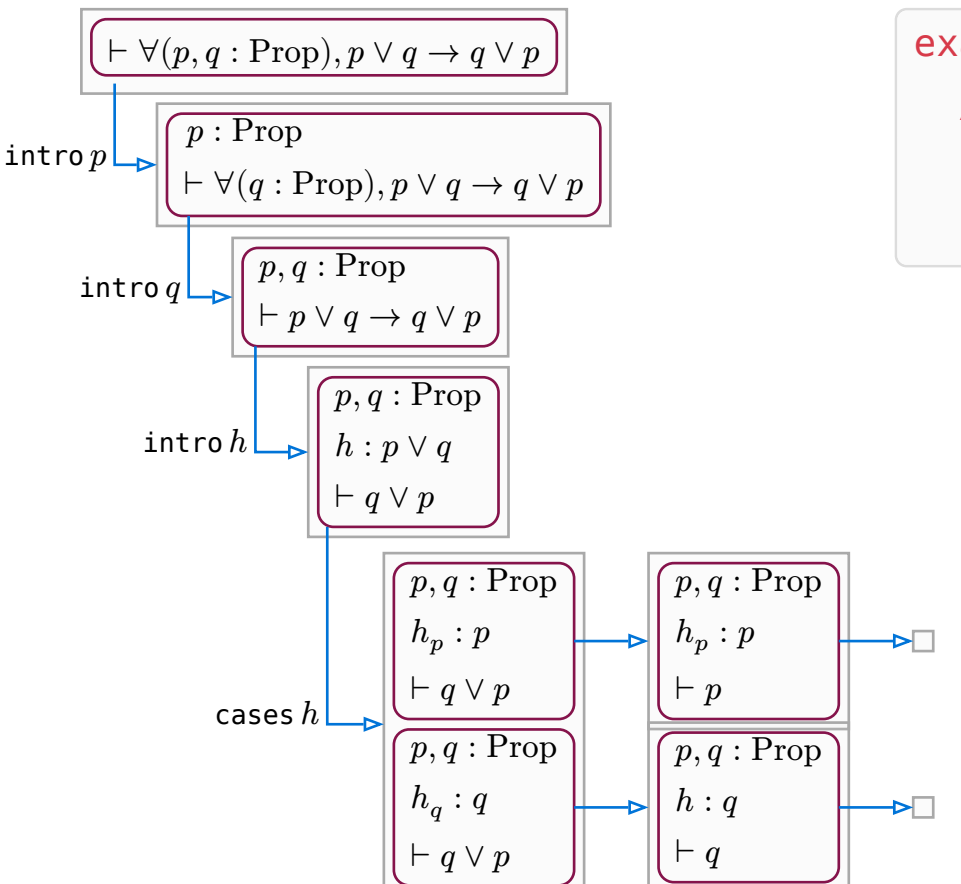

```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim h
    (λ hp => ?_)
    (λ hq => ?_)
```

How humans prove, alternatively




```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim hor
    (λ hp => Or.inr ?_)
    (λ hq => Or.inl ?_)
```

How humans prove, alternatively




```
example: ∀ (P Q: Prop), P ∨ Q → Q ∨ P :=
  λ P Q hor => Or.elim hor
    (λ hp => Or.inr hp)
    (λ hq => Or.inl hq)
```

Two Paradigms: term vs tactic

- **Term based**: giving a direct witness of type inhabitation
- **Tactic based**: refining the term by filling “holes” with meta-level instructions

How humans prove, alternatively

```
⟨x: T, p: P x⟩: ∃ x: T, P x
```

How humans prove, alternatively

```
⟨x: T, p: P x⟩: ∃ x: T, P x
```

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 := ...
```

How humans prove, alternatively

```
⟨x: T, p: P x⟩: ∃ x: T, P x
```

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 := by
  refine ⟨3, ?_⟩
  rfl
```

How humans prove, alternatively

```
⟨x: T, p: P x⟩: ∃ x: T, P x
```

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 := by
  refine ⟨3 -- magical num?, ?_⟩
  rfl
```

How humans prove, alternatively

```
⟨x: T, p: P x⟩: ∃ x: T, P x
```

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 := by
  refine ⟨?_, ?_⟩
  pick_goal 2
  rfl
```

How humans prove, alternatively

```
⟨x: T, p: P x⟩: ∃ x: T, P x
```

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 := by
  refine ⟨?_, ?_⟩
  pick_goal 2
  rfl -- closes 1st goal automatically via unification
```

Intuition:

1. The difficulty of constructing a proof sometimes depends on the order one chooses to fill the holes.
2. Solving a goal might help solve another goal.

How humans prove, alternatively

```
⟨x: T, p: P x⟩: ∃ x: T, P x
```

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 := by
  refine ⟨?_, ?_⟩
  pick_goal 2
  rfl -- closes 1st goal automatically via unification
```

Two Representations: A view shift from *Search View* to *Presentation View*.

Three Views of Proofs[1]

- **Presentation View**: Polished proof for presentation (natural language) or verification (machine readable). Contains "*magical*" values.

[1] Aniva et al. (TACAS 2025). Pantograph: A Machine-to-Machine Interaction Interface for Advanced Theorem proving, ... in Lean 4.

[2] R. L. Morris. Motivated Proofs: What They Are, Why They Matter and How to Write Them

Three Views of Proofs[1]

- **Presentation View**: Polished proof for presentation (natural language) or verification (machine readable). Contains "*magical*" values.

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

[1] Aniva et al. (TACAS 2025). Pantograph: A Machine-to-Machine Interaction Interface for Advanced Theorem proving, ... in Lean 4.

[2] R. L. Morris. Motivated Proofs: What They Are, Why They Matter and How to Write Them

Three Views of Proofs[1]

- **Presentation View**: Polished proof for presentation (natural language) or verification (machine readable). Contains "*magical*" values.

  E.g. "Let $\varepsilon > 0$. Define $\delta := \min\left(2\varepsilon^2, \frac{\varepsilon}{5}, \frac{1}{3}\right)$, ..."
- **Search (proof) View**: The trajectory a prover (machine/human) follows, i.e. *Motivated Proof*[2].

  "Let $\varepsilon > 0$. Let $\delta :=? 1$. Using lemma ..., we know $\delta \leq \frac{\varepsilon}{5}$, ... Therefore we can set $\delta := \min(\ldots)$."

---

[1] Aniva et al. (TACAS 2025). Pantograph: A Machine-to-Machine Interaction Interface for Advanced Theorem proving, ... in Lean 4.

[2] R. L. Morris. Motivated Proofs: What They Are, Why They Matter and How to Write Them

Three Views of Proofs[1]

- **Presentation View**: Polished proof for presentation (natural language) or verification (machine readable). Contains "*magical*" values.
- **Search (proof) View**: The trajectory a prover (machine/human) follows, i.e. *Motivated Proof*[2].
- **Kernel View**: The proof term itself. Might contain metavars (holes) if unfinished.

---

[1] Aniva et al. (TACAS 2025). Pantograph: A Machine-to-Machine Interaction Interface for Advanced Theorem proving, ... in Lean 4.

[2] R. L. Morris. Motivated Proofs: What They Are, Why They Matter and How to Write Them

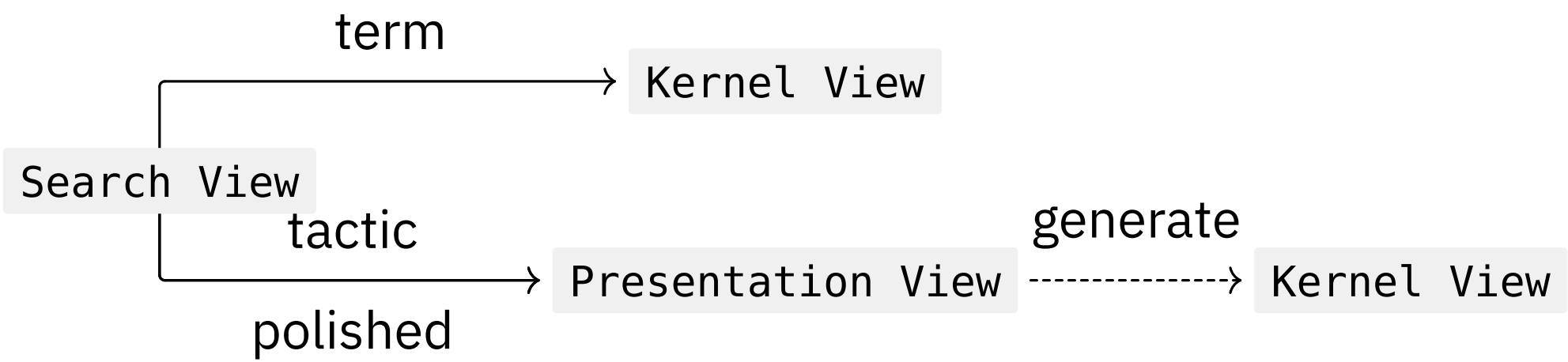
How humans prove

1. Prove a theory (either by constructing terms by hand or by using tactics) in the **Search View**.
   - Term based: a tree of *terms (with holes) attempted*
   - Tactic based: a tree of *tactic sequence applied*

How humans prove

1. Prove a theory (either by constructing terms by hand or by using tactics) in the **Search View**.
   - Term based: a tree of *terms (with holes) attempted*
   - Tactic based: a tree of *tactic sequence applied*
2. Close the proof
   - If term based: **Kernel View** proof
   - If tactic based:
     1. Polish the proof
     2. Get a **Presentation View** proof (→ **Kernel View**)

How humans prove



How machines prove: a historical survey of ATP

1. Brute-force term search (90s)
2. Hammer & SMT bridge (2000-)
3. Data-driven (201?-)

The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
?_
```

The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
λ p q h => ?_
```

The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
λ p q h => Or.inl ?_
```
⛔

The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
λ p q h => Or.inr ?_
```
⛔

The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
λ p q h => Or.elim ?_ ?_
```

The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
λ p q z => Or.elim (λ h => ?_) (λ h => ?_)
```

The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
λ p q z => Or.elim (λ h => Or.inr h) (λ h => Or.inl h)
```

The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
λ p q z => Or.elim (λ h => Or.inr h) (λ h => Or.inl h)
```

Mimics how humans prove by hand, sans the intuition and experience.

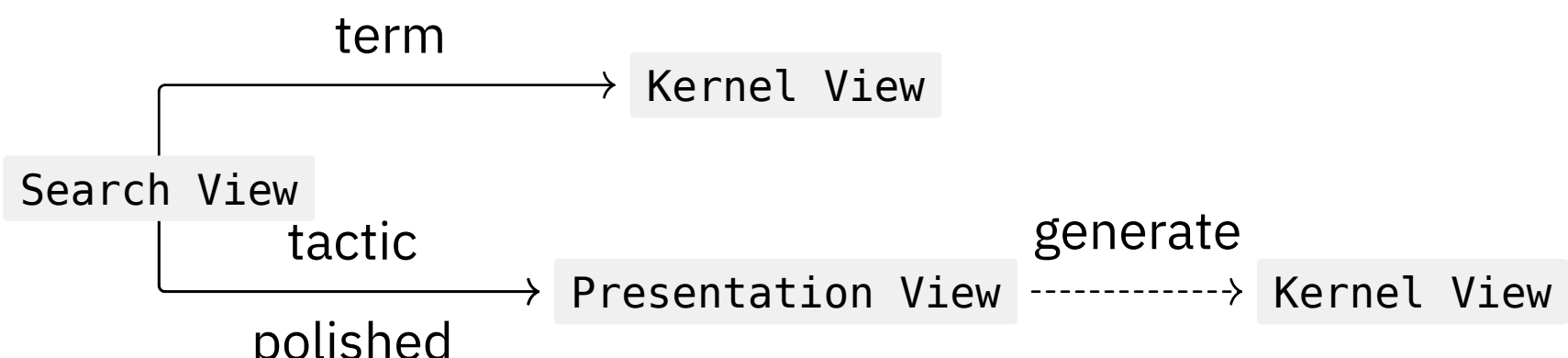
The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

```
λ p q z => Or.elim (λ h => Or.inr h) (λ h => Or.inl h)
```

Mimics how humans prove by hand, sans the intuition and experience.



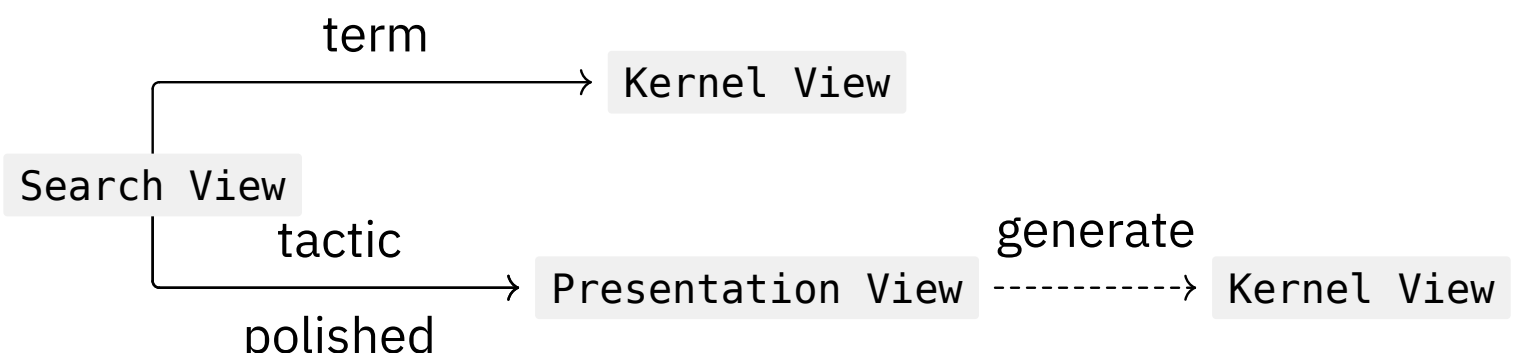
The brute-force way

The most intuitive and straightforward way.

Theorem: $\forall (p\ q : \mathrm{Prop}), P \vee Q \to Q \vee P$

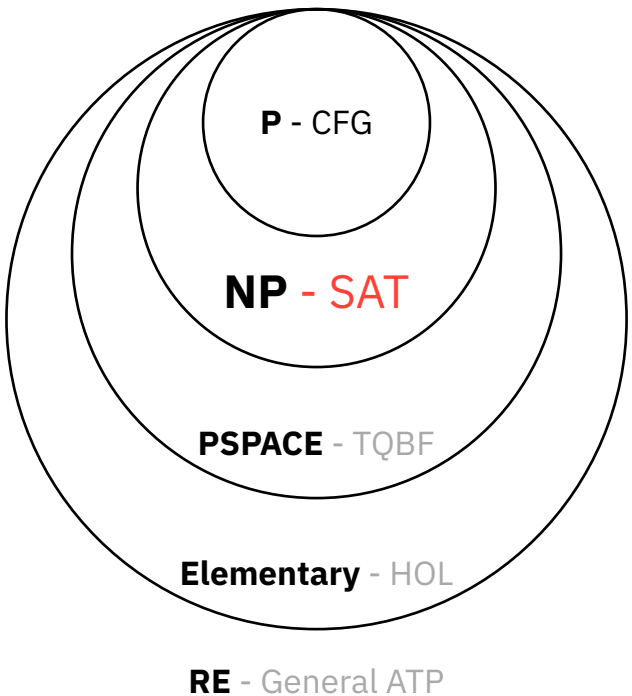
```
λ p q z => Or.elim (λ h => Or.inr h) (λ h => Or.inl h)
```

Mimics how humans prove by hand, sans the intuition and experience.



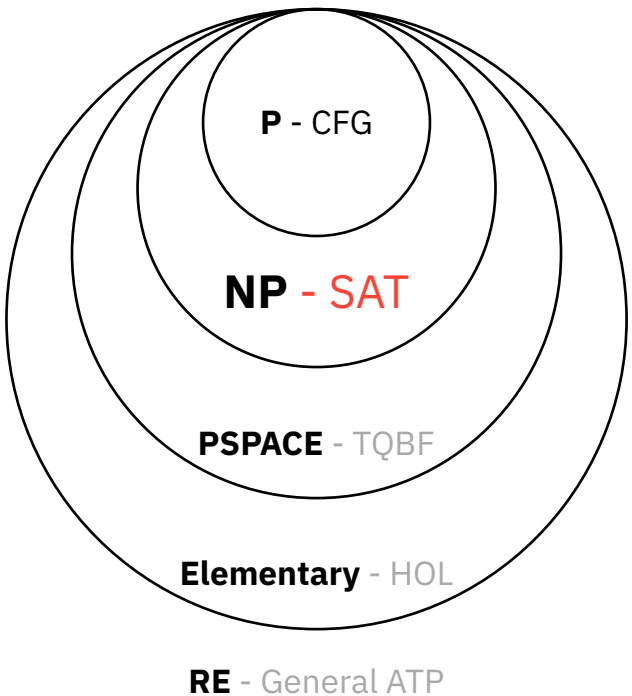
Problem: huge search space

The hammer way



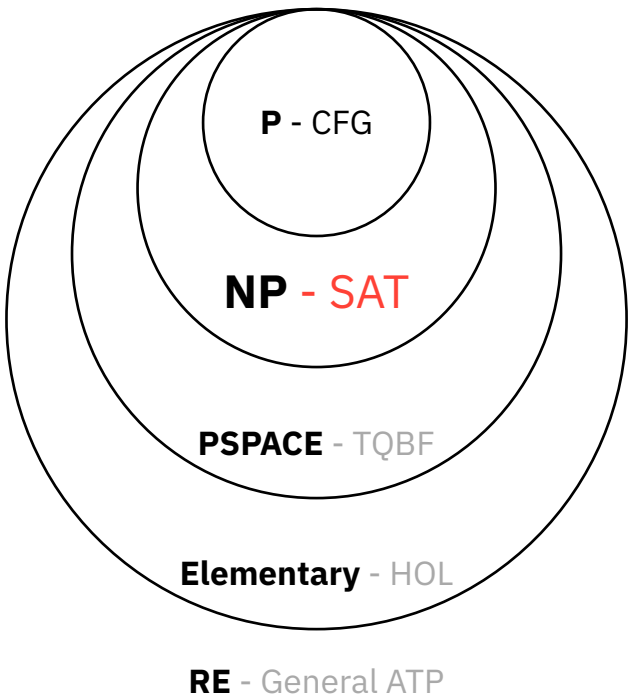
1. Cut out a decidable fragment (e.g. QF_uf)

The hammer way



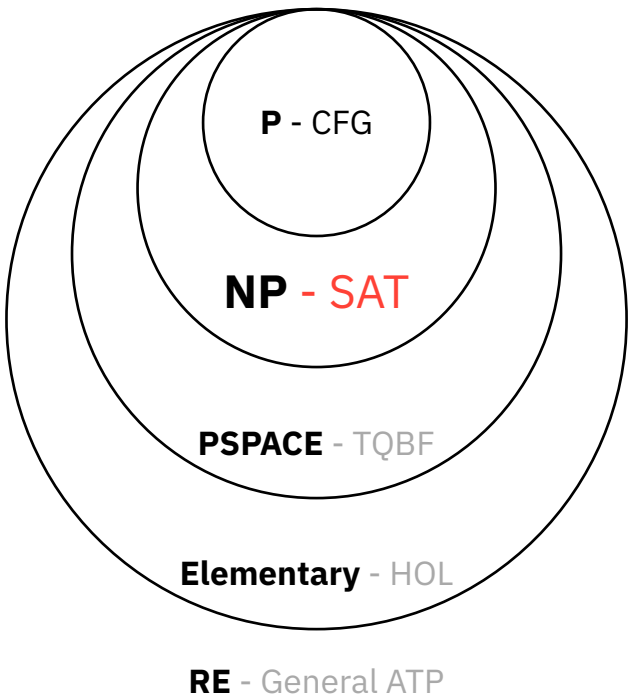
1. Cut out a decidable fragment (e.g. QF_uf)
2. Find out related theorems

The hammer way



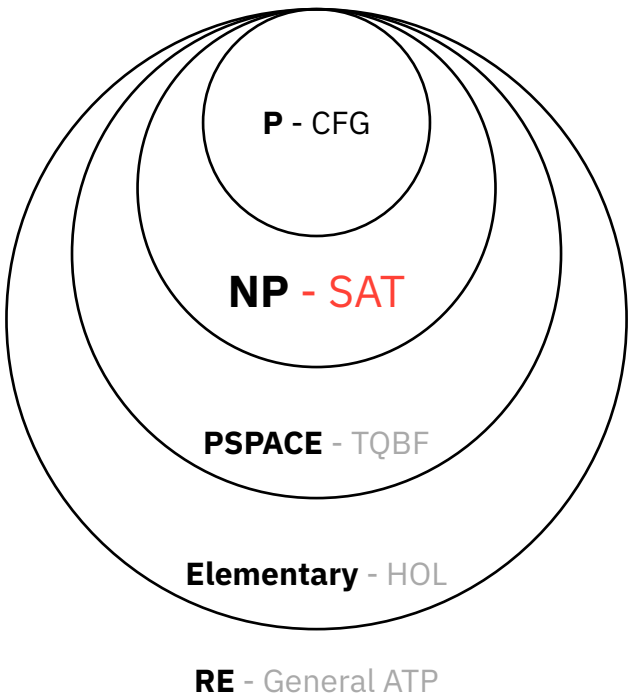
1. Cut out a decidable fragment (e.g. QF_uf)
2. Find out related theorems
3. Send them to a solver

The hammer way

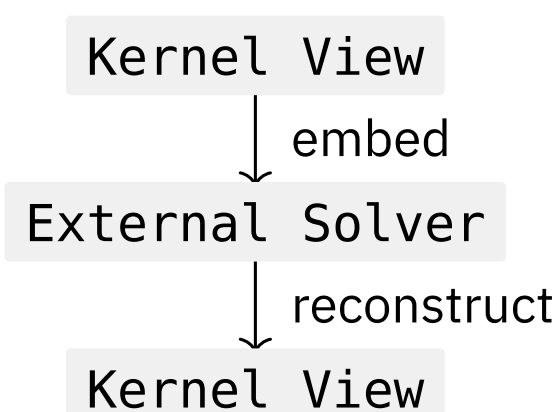


1. Cut out a decidable fragment (e.g. QF_uf)
2. Find out related theorems
3. Send them to a solver
4. Reconstruct the proof

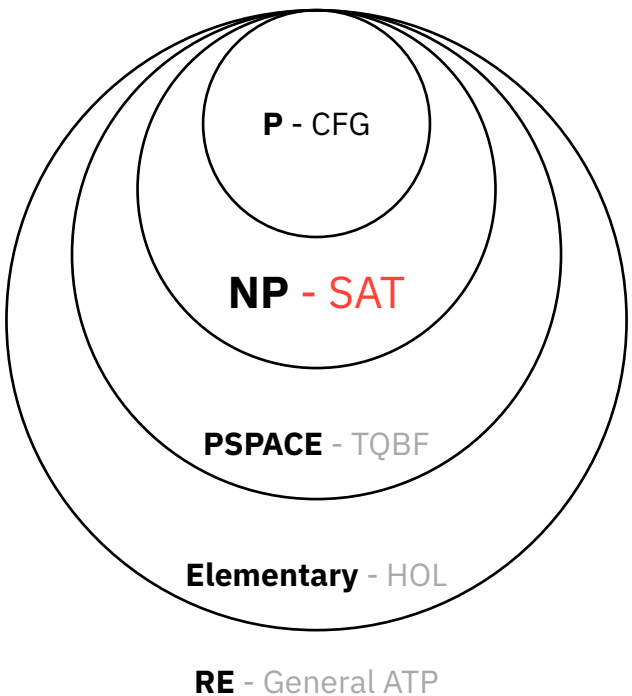
The hammer way



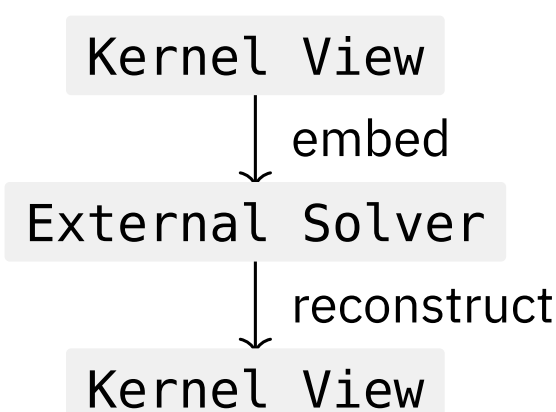
1. Cut out a decidable fragment (e.g. QF_uf)
2. Find out related theorems
3. Send them to a solver
4. Reconstruct the proof



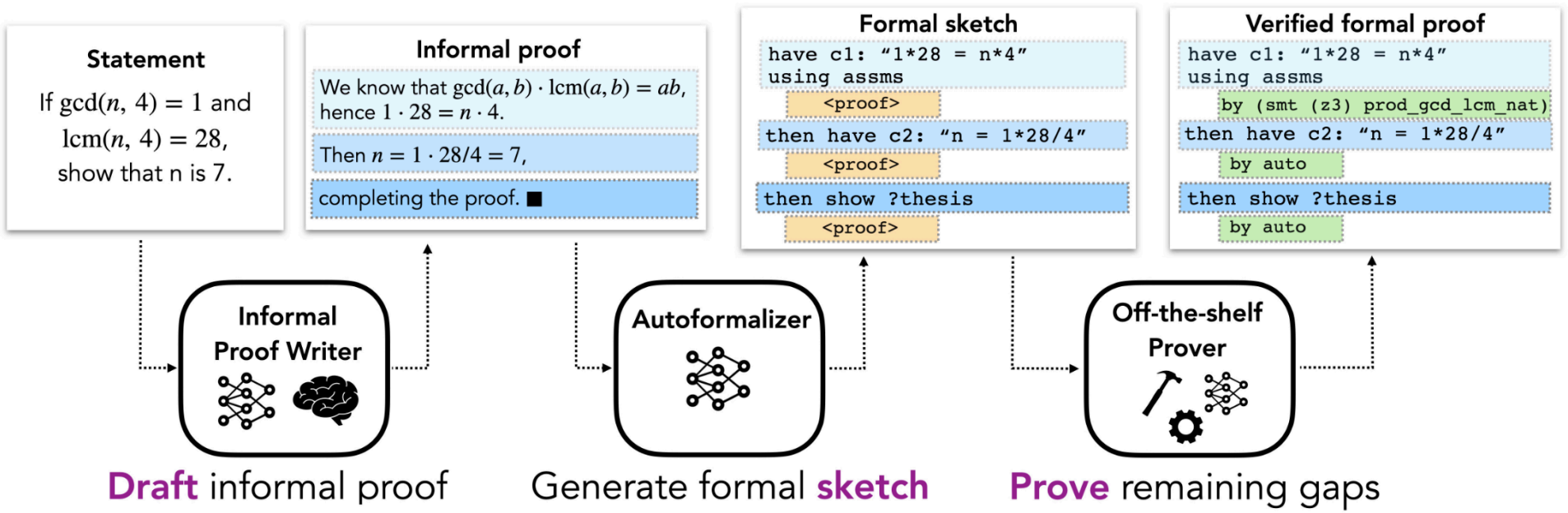
The hammer way



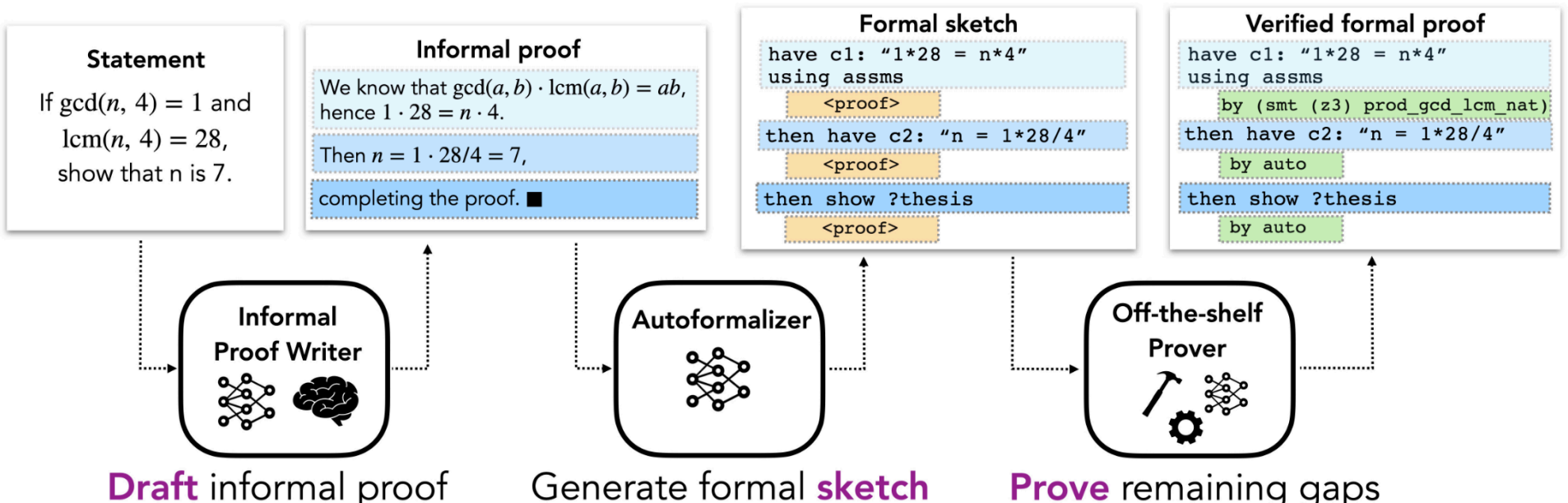
1. Cut out a decidable fragment (e.g. QF_uf)
2. **Find out related theorems**
3. Send them to a solver
4. Reconstruct the proof



The data-driven way (~~The magic blackbox~~ 😨)



The data-driven way (~~The magic blackbox~~ 😨)



sketch ⟶ `Presentation` draft ⟶ `Search` prove ⟶ `Presentation` generate ⇢ `Kernel`

Challenges

- **Premise Selection**: Which constructor/lemma to use?
  A huge headache: induction!

[1]Limperg and Halkjær. (CPP 2023). Aesop: White-Box Best-First Proof Search for Lean.

Challenges

- **Premise Selection**: Which constructor/lemma to use?
  A huge headache: induction!
- **Search Space Pruning**: Do not revisit the same state.

[1]Limperg and Halkjær. (CPP 2023). Aesop: White-Box Best-First Proof Search for Lean.

Challenges

- **Premise Selection**: Which constructor/lemma to use?
  A huge headache: induction!
- **Search Space Pruning**: Do not revisit the same state.
- **Metavar Coupling[1] (Selection)**: Which goal to solve first?

[1]Limperg and Halkjær. (CPP 2023). Aesop: White-Box Best-First Proof Search for Lean.

Metavariable Coupling

Recall

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 := by
  refine ⟨?_, ?_⟩
  pick_goal 2
  rfl -- closes 1st goal automatically via unification
```

Metavariable Coupling

Recall

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 :=
  ⟨?1, ?2 -- fill with rfl⟩
```

Metavariable Coupling

Recall

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 :=
  ⟨?1 -- 3 by n = 3 ∼ 3 = 3, rfl⟩
```

Metavariable Coupling

Recall

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 :=
  ⟨ ?1 -- 3 by n = 3 ∼ 3 = 3, rfl⟩
```

There's a dependency between `?1` and `?2` [1].
So, the algorithm must be able to ...

[1] Note that while the term "metavar coupling" was first introduced by aesop, the concept itself is not new. It's a natural consequence of dependent unification.

Metavariable Coupling

Recall

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 :=
  ⟨?1 -- 3 by n = 3 ∼ 3 = 3, rfl⟩
```

There's a dependency between `?1` and `?2` [1].
So, the algorithm must be able to ...
1. keep track of the dependencies and update them during unification.

[1] Note that while the term "metavar coupling" was first introduced by aesop, the concept itself is not new. It's a natural consequence of dependent unification.

Metavariable Coupling

Recall

```
theorem exists_nat_eq_3: ∃ n: Nat, n = 3 :=
  ⟨?1 -- 3 by n = 3 ∼ 3 = 3, rfl⟩
```

There's a dependency between `?1` and `?2` [1].
So, the algorithm must be able to ...

1. keep track of the dependencies and update them during unification.
2. choose wisely which metavariable to solve first.

[1] Note that while the term "metavar coupling" was first introduced by aesop, the concept itself is not new. It's a natural consequence of dependent unification.

Case Study

- **Canonical**: Brute-force search, done smartly
- **Hammer**: When type theory based reasoning meets solvers
- **DSP** (**LLM**): The **panacea**, but at what cost?

Case Study: Canonical

Conventional term-based ID-DFS, with a special type theory.

Case Study: Canonical

Conventional term-based ID-DFS, with a special type theory.

Core idea: maintain β-normal η-long form, even through substitution, for free.

$(\lambda x.C\ x)\ 1$	🚫 β-reducible
$f : \tau_1 \to \tau_2$	🚫 η-expandable
$\lambda x.fx : \tau_1 \to \tau_2$	✅ β-normal η-long
$(fg)h$	🚫 η-expandable: $(\lambda x.fgx)h$, i.e. no partial app

Case Study: Canonical

Maintain β-normal η-long form, even through substitution, for free.

Why? It solves the state pruning problem.

All $\alpha\beta\eta$-equiv terms are syntactically equal! (βnηl + ln)
A hashset of seen terms is enough. No normalization.

Case Study: Canonical

Maintain β-normal η-long form, even through substitution, for free.

Why? It (partly) simplifies the premise selection problem.

E.g. $h\ ?1$ with local context $C_1 : a \to T, C_2 : a \to b \to T, D_1 : a \to U \vdash T$

The only two viable candidates are

- C_1: $h\ (C_1\ ?2)$, and
- C_2: $h\ (C_1\ ?2\ ?3)$

Case Study: Canonical

Maintain β-normal η-long form, even through substitution, for free.

Why? It (partly) solves the premise selection problem.

E.g. $\mathrm{id}\ \ ?1\ \ ?2$ with local context $C_1 : a \to T, C_2 : a \to b \to T, D_1 : a \to U \vdash ?\, t$

We have three candidates.

Case Study: Canonical

Maintain β-normal η-long form, even through substitution, for free.

Why? It (partly) solves the premise selection problem.

E.g. $\mathrm{id}\ \ ?1\ ?2$ with local context $C_1 : a \to T, C_2 : a \to b \to T, D_1 : a \to U \vdash ?\, t$

If, we are not enforcing βnηl form, it can be a lambda! (We always have $n + 1$ choices)

Worse, $\mathrm{id}\ \ (\Pi_{x:a}\ ?1)\ ?2\ ?3$!

Case Study: Canonical

Maintain β-normal η-long form, even through substitution, for free.

Why? It (partly) solves the premise selection problem.

E.g. $\mathrm{id}\ ?1\ ?2$ with local context $C_1 : a \to T, C_2 : a \to b \to T, D_1 : a \to U \vdash ?\, t$

But now that we are enforcing βnηl, it's impossible for a metavar to be a lambda:

$?1 : \tau_1 \to \tau_2$ => $\lambda x : \tau_1 . (?1 : \tau_2)$

Case Study: Canonical

Maintain β-normal η-long form, even through substitution, for free.

Why? It enforces static arity, enabling usage of efficient data structures.

Always full app: `{ head: Term, args: Array Term }`
Partial app: `{ lhs: Term, rhs: { lhs: Term, rhs: ... }}`

Case Study: Canonical, and its Achilles' Heel

Induction is a big problem.

Case Study: Canonical, and its Achilles' Heel

Induction is a big problem.

```
Nat.rec : {motive : Nat → Type} →
  motive Nat.zero → ((n : Nat) → motive n → motive n.succ) →
  (t : Nat) → motive t
```

`motive` is universally quantified. It's a candidate for every hole.

Case Study: Hammer

Outsourcing the proof search to an external solver (e.g. Z3, Vampire).

Case Study: Hammer

Outsourcing the proof search to an external solver (e.g. Z3, Vampire).

1. Export subgoals to external solvers, often making dependent types opaque

Case Study: Hammer

Outsourcing the proof search to an external solver (e.g. Z3, Vampire).

1. Export subgoals to external solvers, often making dependent types opaque
2. Filter out a list of likely related theorems and send them to the solver

Case Study: Hammer

Outsourcing the proof search to an external solver (e.g. Z3, Vampire).

1. Export subgoals to external solvers, often making dependent types opaque
2. Filter out a list of likely related theorems and send them to the solver
3. Replaying external proofs back to the type theory via reconstruction

Case Study: Hammer

Outsourcing the proof search to an external solver (e.g. Z3, Vampire).

	Hammer	**ATP**
Automation	push-button	varies
Expressiveness	limited to decidable fragment	full support for DT and HoL
Trust	longer trust chain	kernel-checked directly
Performance	highly optimized	slow in large libs or proofs
Visibility	hidden in the solver	full trace available

Case Study: Hammer

Outsourcing the proof search to an external solver (e.g. Z3, Vampire).

	Hammer	**ATP**
Automation	push-button	varies
Expressiveness	limited to decidable fragment	full support for DT and HoL
Trust	longer trust chain	kernel-checked directly
Performance	highly optimized	slow in large libs or proofs
Visibility	hidden in the solver	full trace available

Trade-off between expressiveness and automation.

Case Study: Hammer

Outsourcing the proof search to an external solver (e.g. Z3, Vampire).

Challenges:

1. Premise Selection: depending on other solutions (adhoc heuristics, reinforcement learning, etc.)

Case Study: Hammer

Outsourcing the proof search to an external solver (e.g. Z3, Vampire).

Challenges:

1. Premise Selection: depending on other solutions (adhoc heuristics, reinforcement learning, etc.)
2. Induction: no support (recursors are dependently typed)

Case Study: DSP (LLM)

The good:

1. **It works magically!** ($\sim 50\%$ success rate on some datasets)

Case Study: DSP (LLM)

The good:

1. **It works magically!** ($\sim 50\%$ success rate on some datasets)
2. Somewhat explainable: the informal proof sketch is human-readable.

Case Study: DSP (LLM)

The ugly:

- Limited to tactic based approach due to the nature of DSP, thus

Case Study: DSP (LLM)

The ugly:

- Limited to tactic based approach due to the nature of DSP, thus
- Need to tackle with tactic selection, a slightly different problem with premise selection, and

Case Study: DSP (LLM)

The ugly:

- Limited to tactic based approach due to the nature of DSP, thus
- Need to tackle with tactic selection, a slightly different problem with premise selection, and
- Need to learn how to parametrize the tactic.

Case Study: DSP (LLM)

The bad:

1. Neural networks are not explainable.

Case Study: DSP (LLM)

The bad:

1. Neural networks are not explainable.
2. Insufficient dataset: most datasets are synthetic, and the real-world datasets are not large enough.

Case Study: DSP (LLM)

The bad:

1. Neural networks are not explainable.
2. Insufficient dataset: most datasets are synthetic, and the real-world datasets are not large enough.
3. Unreliable benchmarking: data leak problem
   - Avoidable: training set pollution
   - Unavoidable: prior knowledge from natural language pre-training

Best of all three worlds

1. Symbolic approach with a solver heart
   Regain the power of domain specific tactics.

Best of all three worlds

1. Symbolic approach with a solver heart
   Regain the power of domain specific tactics.
2. Symbolic approach with a neuro heart
   (Domain-specific) learning based premise/metavar selection.

Best of all three worlds

1. Symbolic approach with a solver heart
   Regain the power of domain specific tactics.
2. Symbolic approach with a neuro heart
   (Domain-specific) learning based premise/metavar selection.
3. Neuro approach with a symbolic heart
   Explainable & reliable logic core.

Open problems

1. Premise Selection

Open problems

1. Premise Selection
2. Lemma Discovery: reuse proofs, generalizing theorems
   Might help with the induction headache.

Acknowledgements

Leni Aniva, for her brilliant idea of the three views of proof, insights on neuro-symbolic methods, and also her help on the diagrams.

Tesla Zhang, for his clear explanation of the Canonical theorem prover.

Alexander Chichigin, for his attentive review and constructive criticism.