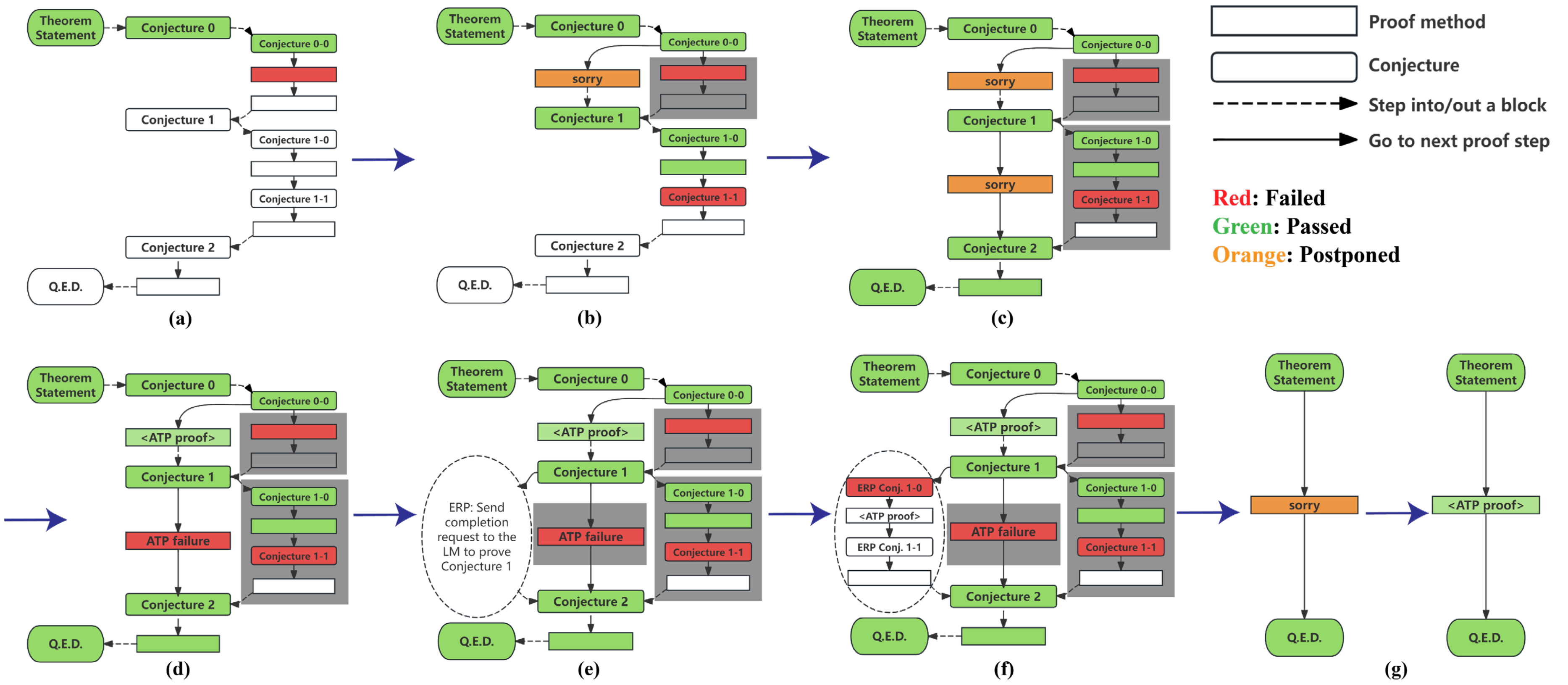
ProofAug: Efficient Neural Theorem Proving via Fine-grained Proof Structure Analysis

Haoxiong Liu [1]   Jiacheng Sun [2]   Zhenguo Li [2]   Andrew C Yao [1 3]



Formal theorem proving

AI achieves silver-medal standard solving International Mathematical Olympiad problems

25 JULY 2024

AlphaProof and AlphaGeometry teams

Let $\mathbb{Q}$ be the set of rational numbers. A function $f : \mathbb{Q} \to \mathbb{Q}$ is called \emph{aquaesulian} if the following property holds: for every $x, y \in \mathbb{Q}$,

$$f(x + f(y)) = f(x) + y \quad \text{or} \quad f(f(x) + y) = x + f(y).$$

Show that there exists an integer c such that for any aquaesulian function f there are at most c different rational numbers of the form $f(r) + f(-r)$ for some rational number r, and find the smallest possible value of c.

Solution: c=2

Formalized to Lean 4

```
theorem imo_2024_p6
    (IsAquaesulian : (ℚ → ℚ) → Prop)
    (IsAquaesulian_def : ∀ f, IsAquaesulian f ↔
      ∀ x y, f (x + f y) = f x + y ∨ f (f x + y) = x + f y) :
    IsLeast {(c : ℤ) | ∀ f, IsAquaesulian f → {(f r + f (-r)) | (r : ℚ)}.Finite ∧
      {(f r + f (-r)) | (r : ℚ)}.ncard ≤ c} 2 := by
```

Proof search with prover and compiler

The best practice is still unclear

```
theorem imo_2024_p6
    (IsAquaesulian : (ℚ → ℚ) → Prop)
    (IsAquaesulian_def : ∀ f, IsAquaesulian f ↔
      ∀ x y, f (x + f y) = f x + y ∨ f (f x + y) = x + f y) :
    IsLeast {(c : ℤ) | ∀ f, IsAquaesulian f → {(f r + f (-r)) | (r : ℚ)}.Finite ∧
      {(f r + f (-r)) | (r : ℚ)}.ncard ≤ c} 2 := by
  exists@?_
  · useλu b=>if j:u 0=0then by_contra λc=>?_ else ?_
    · suffices:({J|∃k,u k+u (-k)= J}) ⊆{0}
      · simp_all[this.antisymm]
      rintro - ⟨a, rfl⟩
      contrapose! c
      simp_all
      suffices:{U|∃examples6, (u) ‹ℚ› +u ( -‹_›)= U} ⊆{0,(u (a : Rat)+ (u<|@@↑((
      (-a ))))) } ..
      · use ( Set.toFinite ( _) ).subset ↑@@this , (Set.ncard_le_ncard$ (((this )) ) ).
      trans (Set.ncard_pair$ Ne.symm (↑ ( (c)) ) ).le
      rintro-⟨hz, rfl⟩
      induction b @hz a
      · have:=b (-a)$ hz+u a
        have:=b hz hz
        simp_all[add_comm]
        have:=b (-hz) (hz+u ↑(hz))
        simp_all[ add_assoc, C]
        induction this
        · simp_all
          have:=b hz (hz+(u a+u (-a)))
          have:=b (hz+(u a+u (-a)))$ hz+(u a+u (-a))
          use .inr$ by_contra$ by hint
        have:=b hz$ hz+(u hz+u (-hz))
        cases b (hz+(u hz+u (-hz)))$ hz+(u hz+u (-hz))with|_=>hint
      have:=b (-hz) (u hz+a)
      have:=b$ -a
      specialize this (u hz+a)
      simp_all[ ←add_assoc]
⋮
```

Verified in Lean compiler

Procedural (tactic-style) vs. Declarative

<table>
<tr><td>Natural Language</td><td>Two non-zero real numbers, a and b, satisfy $ab = a - b$. Which of the following is a possible value of $\frac{a}{b} + \frac{b}{a} - ab$? (A) -2 (B) $\frac{-1}{2}$ (C) $\frac{1}{3}$ (D) $\frac{1}{2}$ (E) 2</td></tr>
<tr><td>Metamath</td><td>${
amc12-2000-p11.0 $e |- (ph -> A e. RR) $.
amc12-2000-p11.1 $e |- (ph -> B e. RR) $.
amc12-2000-p11.2 $e |- (ph -> A =/= 0) $.
amc12-2000-p11.3 $e |- (ph -> B =/= 0) $.
amc12-2000-p11.4 $e |- (ph -> (A x. B) =
  (A - B)) $.
amc12-2000-p11 $p |- (ph -> (((A / B) +
  (B / A)) - (A x. B)) = 2)
$=
(cdiv co caddc cmul cmin c2 cexp eqcomd ... $.
$}</td></tr>
<tr><td>Lean</td><td>theorem amc12_2000_p11
  (a b : ℝ)
  (h₀ : a ≠ 0 ∧ b ≠ 0)
  (h₁ : a * b = a - b) :
  a / b + b / a - a * b = 2 :=
begin
  field_simp [h₀.1, h₀.2],
  simp only [h₁, mul_comm, mul_sub],
  ring,
end</td></tr>
<tr><td>Isabelle</td><td>theorem amc12_2000_p11:
  fixes a b::real
  assumes "a \<noteq> 0" "b \<noteq> 0"
    and "a * b = a - b"
    shows "a / b + b / a - a * b = 2"
  using assms
  by (smt (verit, ccfv_threshold)
    diff_divide_distrib
    div_self divide_divide_times_eq
    eq_divide_imp nonzero_mult_div_cancel_left)
end</td></tr>
</table>

Procedural proofs (Adapted from Polu et al. 2020)

VS.

```
lemma prime_mod_4_cases:
  fixes p :: nat
  assumes "prime p"
  shows   "p = 2 ∨ [p = 1] (mod 4) ∨ [p = 3] (mod 4)"
proof (cases "p = 2")
  case False
  with prime_gt_1_nat[of p] assms have "p > 2" by auto
  have "¬4 dvd p"
    using assms product_dvd_irreducibleD[of p 2 2]
    by (auto simp: prime_elem_iff_irreducible simp flip: prime_elem_nat_iff)
  hence "p mod 4 ≠ 0"
    by (auto simp: mod_eq_0_iff_dvd)
  moreover have "p mod 4 ≠ 2"
  proof
    assume "p mod 4 = 2"
    hence "p mod 4 mod 2 = 0"
      by (simp add: cong_def)
    thus False using ‹prime p› ‹p > 2› prime_odd_nat[of p]
      by (auto simp: mod_mod_cancel)
  qed
  moreover have "p mod 4 ∈ {0,1,2,3}"
    by auto
  ultimately show ?thesis by (auto simp: cong_def)
qed auto
```

A typical Isabelle/Isar proof adapted from AoFP

```
theorem aime_1983_p2 (x p : ℝ) (f : ℝ → ℝ)
    (h₀ : 0 < p ∧ p < 15) (h₁ : p ≤ x ∧ x ≤ 15)
    (h₂ : f x = abs (x-p) + abs (x-15) + abs (x-p-15)) :
    15 ≤ f x := by
  have h3 : abs (x - p) = x - p := by
    rw [abs_of_nonneg]
    linarith
  have h4 : abs (x - 15) = 15 - x := by
    rw [abs_of_nonpos]
    linarith
    all_goals linarith
  have h5 : abs (x - p - 15) = p + 15 - x := by
    rw [abs_of_nonpos]
    linarith
    all_goals linarith
  rw [h₂, h3, h4, h5]
  linarith
```

A Lean 4 proof by Kimina-Prover-Preview-Distill-7B

Lean

```
theorem amc12_2000_p11
  (a b : ℝ)
  (h₀ : a ≠ 0 ∧ b ≠ 0)
  (h₁ : a * b = a - b) :
  a / b + b / a - a * b = 2 :=
begin
  field_simp [h₀.1, h₀.2],
  simp only [h₁, mul_comm, mul_sub],
  ring,
end
```

Proof-step generation methods

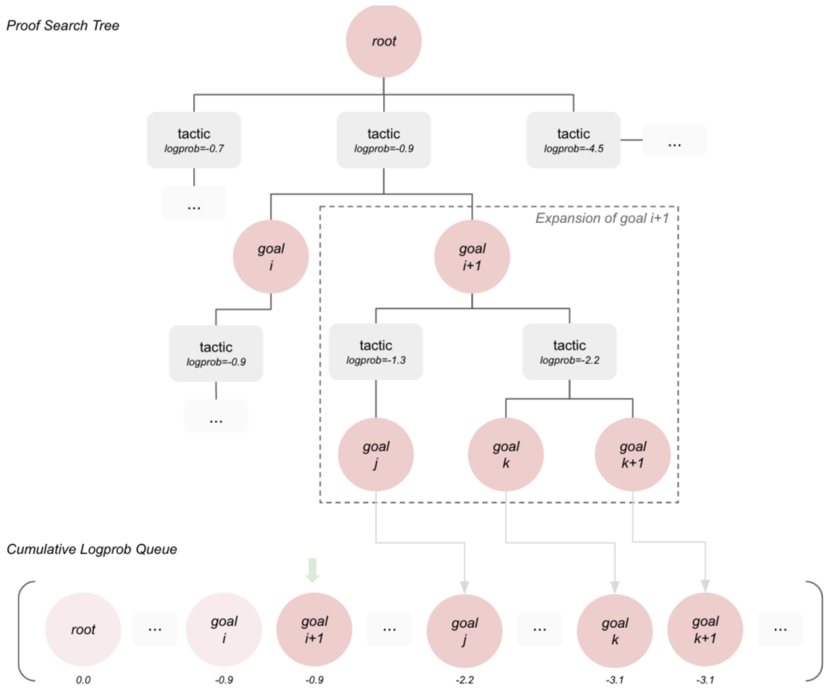


Figure 1: Proof search consists in maintaining a proof tree where multiple tactics are explored for each goal, starting from the root goal. Goals are expanded by cumulative (tactic) logprob priority.

GPT-f (Polu et al. 2020)

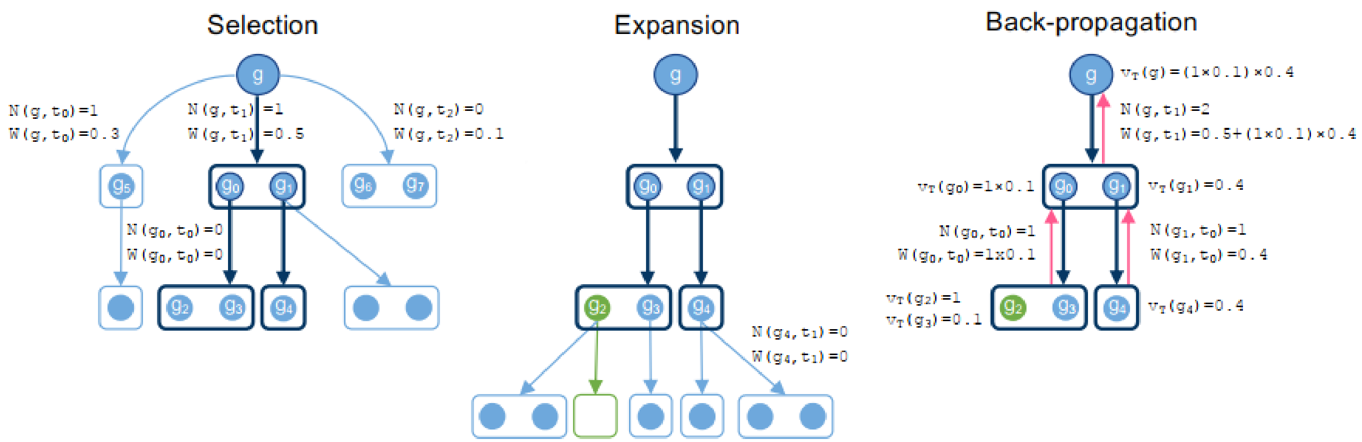


Figure 5: **HyperTree Proof Search**. We aim at finding a proof of the root theorem g with HTPS. Proving either $\{g_5\}$, $\{g_0, g_1\}$, or $\{g_6, g_7\}$ would lead to a proof of g by tactic t_0, t_1, or t_2. The figure represents the three steps of HTPS that are repeated until a proof is found. Guided by the search policy, we select a hypertree whose leaves are unexpanded nodes. The selected nodes are then expanded, adding new tactics and nodes to the hypergraph. Finally, during back-propagation we evaluate the node values of the hypertree, starting from the leaves back to the root, and update the visit counts and total action values.

HTPS (Lample et al. 2022)

Recent work: InternLM2.5-StepProver, Hunyuan-Prover, BFS-Prover, ……

Pitfall: Heavy communication between the prover and the verification environment.
InternLM2.5-StepProver search budget: 256 (#passes) x 32 (#expansion width) x 600 (#max expansions per pass)

Whole-proof generation

- Isabelle/Isar: declarative style proof

```
theorem mathd_algebra_405:
  fixes x :: nat
  assumes h0 : "0 < x"
    and h1 : "x ^ 2 + 4 * x + 4 < 20"
  shows "x = 1 ∨ x = 2"
```

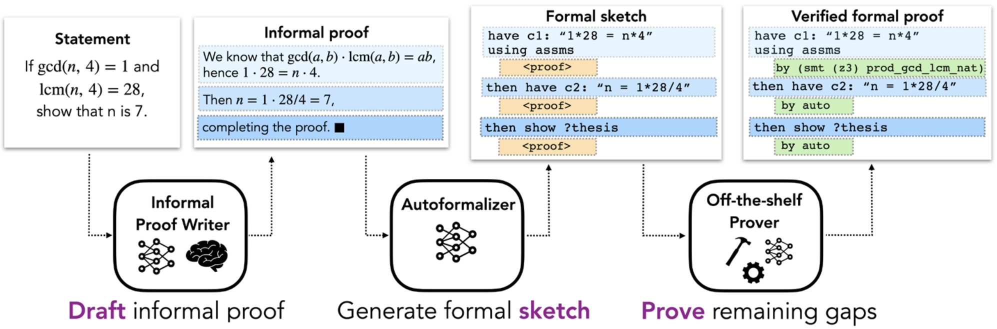
LLM

```
proof (rule ccontr)
  assume "¬ (x = 1 ∨ x = 2)"
  then have "x > 3" using h0 by auto
  then have "x^2 + 4*x + 4 ≥ 3^2 + 4*3 + 4"
   by (metis add_le_cancel_right add_le_mono
        nat_mult_le_cancel_disj power2_nat_le_eq_le)
  then show False using h1 by auto
qed
```

Verified by

- Humans/LLMs are better at writing conjectures than proof methods
  - [green box] : conjecture, [red box] : proof method

- Draft, Sketch, and Prove (Jiang et al. 2023)
  - LLMs compose intermediate conjectures
    - Using few-shot examples of proof sketches
  - Off-the-shelf ATPs fill the gaps



Pitfalls of DSP prompting style

- (Hard Conjectures): the conjectures could be too hard for ATPs to solve.

```
theorem numbertheory_sqmod3in01d:
  fixes a :: int
  shows "a^2 mod 3 = 0 ∨ a^2 mod 3 = 1"
proof -
(* Let $a$ be an integer, then $a \pmod 3 \in {0, 1, 2}$. *)
  have c0: "a mod 3 ∈ {0,1,2}" by fastforce
(* Using that $a^2 \pmod 3 = (a \pmod 3)^2 \pmod 3$ *)
  have "a^2 mod 3 = (a mod 3)^2 mod 3"  by (simp add: power_mod)
(* we have $a^2 \pmod 3 \in {0, 1}$. *)
  then show ?thesis using c0 sledgehammer
qed
```

```
Sledgehammering...
No proof found
```

Easy to human ≠ Easy to ATPs

- (Complicated Draft): the autoformalization process is not robust, and there is a mismatch betweeen informal and formal proof.

```
theorem
  "gcd 180 168 = (12::nat)"
proof -
(* If a number divides into both $180$ and $168$ *)
  have "gcd 180 168 dvd 180" by eval
  moreover have "gcd 180 168 dvd 168" by eval
(* it must also divide into their difference. *)
  finally have "gcd 180 168 dvd 12" sorry
qed
```

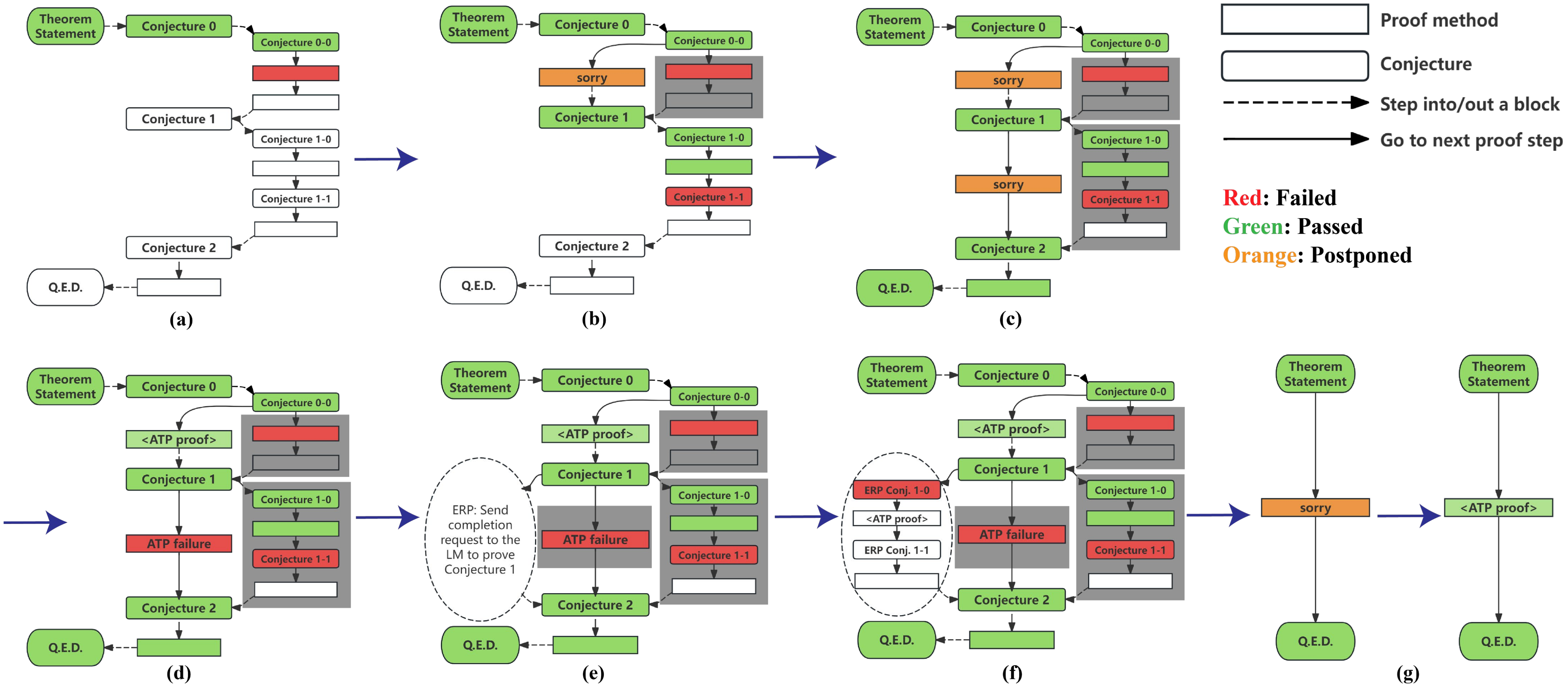
```
theorem
  "gcd 180 168 = (12::nat)"
  by eval
```

A seemingly honest translation can be a disaster!

Our solution to Solve Pitfalls of DSP

- Pitfalls of DSP prompting style
  - (Hard Conjectures): the conjectures could be too hard for ATPs to solve.
  - (Complicated Draft): the autoformalization process is not robust, and there is a mismatch betweeen informal and formal proof.
- Our Solution
  1. Let the model generate the whole proof rather than a proof sketch
     - DSP prompting suppresses the low-level details in the proof sketch
  2. We find **compatible semi-proofs** from the ‘proof proposal’ generated by the model
     - Semi-proofs: valid proofs that can contain ‘sorry’s, which indicate skip of the local proof
     - Compatible: every ‘sorry’ corresponds to some tactics in the original proof proposal
  3. Use ATPs to fill in the gaps in the semi-proofs
     - ATP = sledgehammer / heuristic methods as in DSP

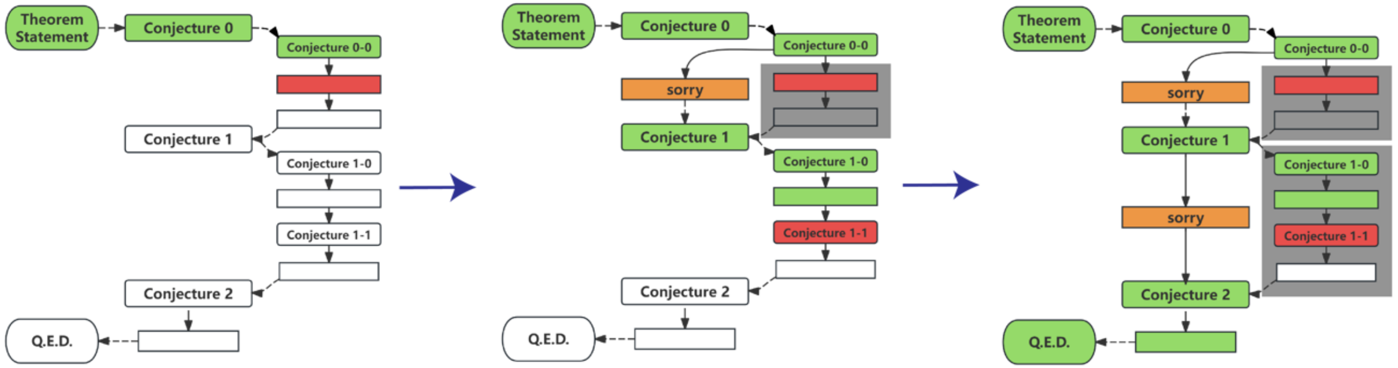
Illustration of Our Solution



Solution Part 1: Find the MCSP

```
theorem algebra_sqineq_at2malt1_init_proof:
  fixes a::real
  shows "a * (2 - a) \<le> 1"
proof -
  have "(a - 1)\<^sup>2 \<ge> 0" for a::real
  proof -
    have "0 \<le> (a - 1) * (a - 1)"
      using zero_le_square by auto
    then show "(a - 1)\<^sup>2 \<ge> 0"
      by (simp add: power2_eq_square)
  qed
  then have "a * (2 - a) \<le> 1" for a::real
  proof -
    have "a * (2 - a) = 2 * a - a\<^sup>2" by (simp add: power2_eq_square)
    also have "... = (a - 1)\<^sup>2 + 1 - a\<^sup>2" by (simp add: algebra_simps)
    also have "... \<le> 1"
      using \<open>0 \<le> (a - 1)\<^sup>2\<close> by linarith
    finally show ?thesis .
  qed
  then show ?thesis .
qed
```

```
theorem algebra_sqineq_at2malt1_MCSP:
  fixes a::real
  shows "a * (2 - a) \<le> 1"
proof -
  have "(a - 1)\<^sup>2 \<ge> 0" for a::real
  proof -
    have "0 \<le> (a - 1) * (a - 1)"
      using zero_le_square by auto
    then show "(a - 1)\<^sup>2 \<ge> 0"
      by (simp add: power2_eq_square)
  qed
  then have "a * (2 - a) \<le> 1" for a::real
  proof -
    have "a * (2 - a) = 2 * a - a\<^sup>2" sorry
    also have "... = (a - 1)\<^sup>2 + 1 - a\<^sup>2" sorry
    also have "... \<le> 1"
      using \<open>0 \<le> (a - 1)\<^sup>2\<close> sorry
    finally show ?thesis .
  qed
  then show ?thesis .
qed
```

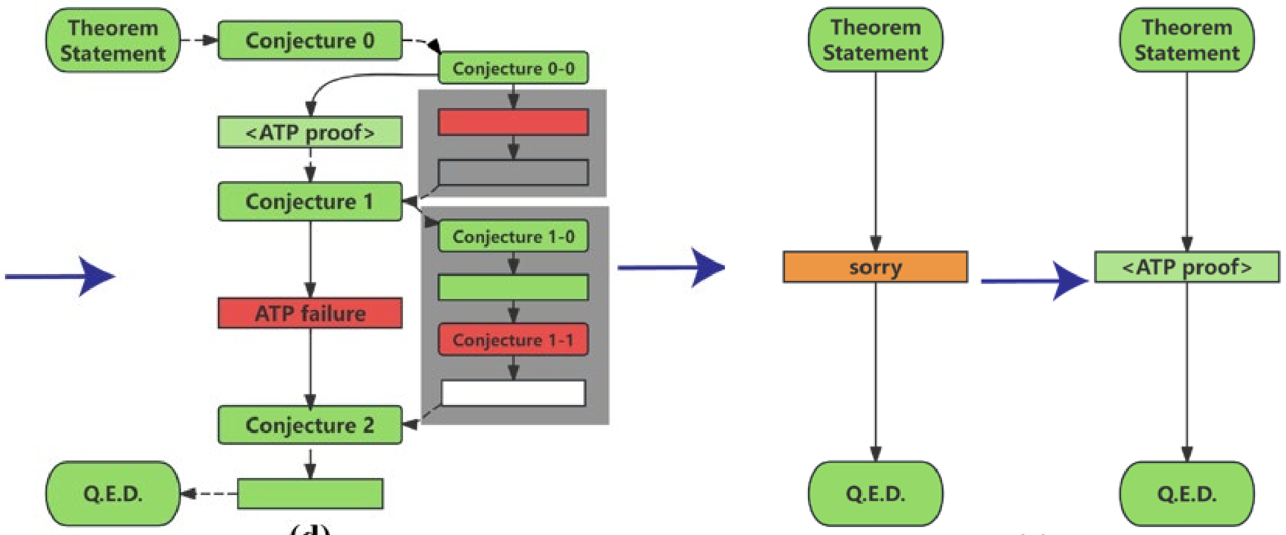


(a) (b) (c)

Algorithm 1 Find the Maximal Compatible Semi-Proof

Input: initial proof y_f^0, ITP $(\mathcal{A}, \mathcal{S}, T, F)$
$\mathbf{a} \leftarrow \text{Parse}(y_f^0)$
$\mathbf{s} \leftarrow [\text{Null}] \times \text{len}(\mathbf{a})$ ▷ States before each step
$i, s_{this} \leftarrow 1, s_0$
while $i \leq \text{len}(\mathbf{a})$ **do**
  $\mathbf{s}[i] \leftarrow s_{this}$
  $s_{next} \leftarrow T(s_{this}, \mathbf{a}[i])$
  if $s_{next}.error$ **then**
    if $s_{this}.mode = \text{proof(prove)}$ **then**
      $\mathbf{a}[i] \leftarrow \text{sorry}$
    else ▷ Error in other modes, skip the block
      $block \leftarrow \text{InnermostBlock}(i, \mathbf{a})$
      if $block$ **is** Null **then**
        return Null ▷ $\mathbf{a}[i]$ not in any block, terminate
      end if
      $\mathbf{a}[block.start..block.end - 1] \leftarrow \text{Null}$
      $\mathbf{a}[block.end] \leftarrow \text{sorry}$
      $i, s_{this} \leftarrow block.end, \mathbf{s}[block.start]$
    end if
  else
    $i, s_{this} \leftarrow i + 1, s_{next}$
  end if
end while
if $s_{this}.finish$ **then**
  return $\text{Concat}(\mathbf{a})$
end if

Solution Part 2: Proof Augmentation

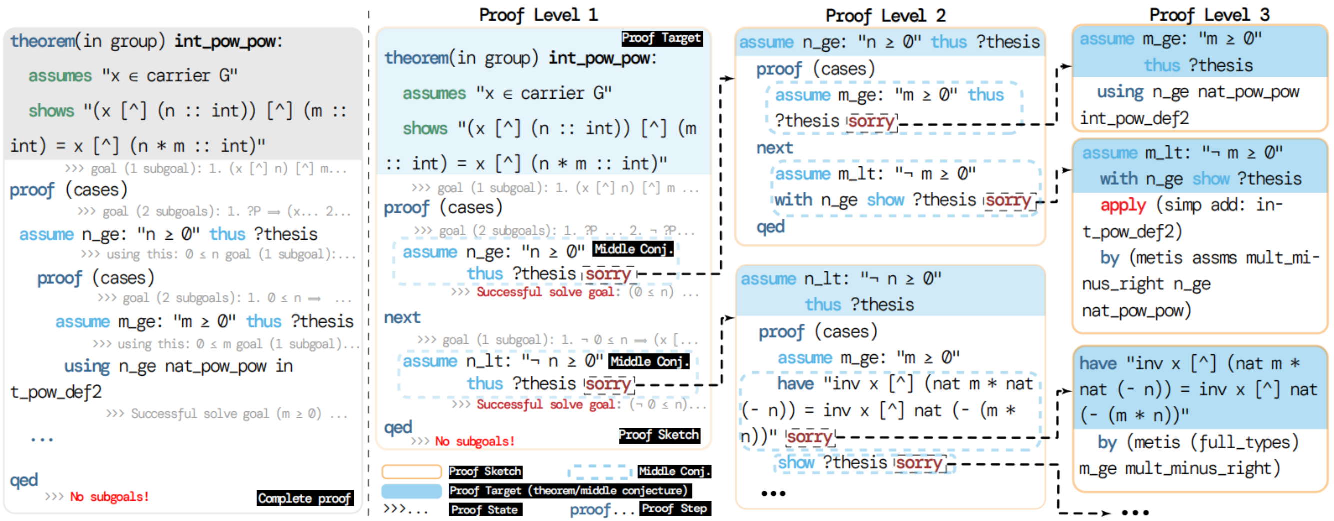


(d) (g)

Algorithm 2 Proof Augmentation (ProofAug)

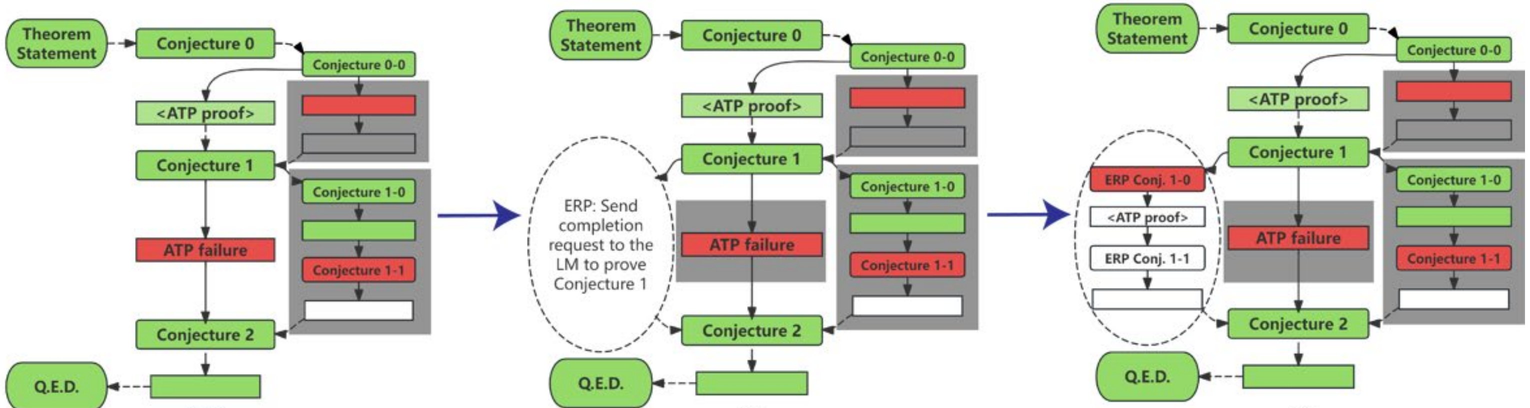
Input: theorem statement x_f, informal statement & problem $x_i \| y_i$, prompter $p(\cdot, \cdot)$, LM $\pi(\cdot|\cdot)$, ITP $(\mathcal{A}, \mathcal{S}, T)$
Sample $y_f^0 \sim \pi(\cdot | p(x_i \| y_i, x_f))$
$\mathbf{a} \leftarrow \text{Parse}(\text{FindMCSP}(y_f^0))$ ▷ Apply Algorithm 1
$\mathbf{s} \leftarrow [\text{Null}] \times \text{len}(\mathbf{a})$
$i, s_{this} \leftarrow 1, s_0$
while $i \leq \text{len}(\mathbf{a})$ **do**
  $s_{next} \leftarrow T(s_{this}, \mathbf{a}[i])$
  if $\mathbf{a}[i] \neq$ sorry **then**
    $error \leftarrow False$
  else
    $error \leftarrow T(s_{this}, \texttt{<ATP>}).error$ ▷ Try ATPs
  end if
  if $error$ **then** ▷ Resort to the last level
    $block \leftarrow \text{InnermostBlock}(i, \mathbf{a})$
    if $block$ **is** Null **then return** Null
    $\mathbf{a}[block.start..block.end - 1] \leftarrow \text{Null}$
    $\mathbf{a}[block.end] \leftarrow$ sorry
    $i, s_{this} \leftarrow block.end, \mathbf{s}[block.start]$
  else
    $i, s_{this} \leftarrow i + 1, s_{next}$
  end if
end while
return Concat($\mathbf{a}$) ▷ The final proof

Solution Part 3: Efficient Recursive Proving (Optional)



(a) Step-by-step Proof (b) Recursive Proof

POETRY (Wang et al. 2024)



(d) (e) (f)

Efficient Recursive Proving (ERP) Module

Algorithm 2 Proof Augmentation (ProofAug)

Input: theorem statement x_f, informal statement & problem $x_i \| y_i$, prompter $p(\cdot,\cdot)$, LM $\pi(\cdot|\cdot)$, ITP $(\mathcal{A}, \mathcal{S}, T)$

Sample $y_f^0 \sim \pi(\cdot|p(x_i\|y_i, x_f))$

$\mathbf{a} \leftarrow$ Parse(FindMCSP(y_f^0)) ▷ Apply Algorithm 1

$\mathbf{s} \leftarrow$ [Null] $\times$ len($\mathbf{a}$)

$i, s_{this} \leftarrow 1, s_0$

while $i \leq$ len($\mathbf{a}$) **do**

  $s_{next} \leftarrow T(s_{this}, \mathbf{a}[i])$

  if $\mathbf{a}[i] \neq$ sorry **then**

    $error \leftarrow False$

  else

    $error \leftarrow T(s_{this},$ <ATP>$).error$ ▷ Try ATPs

  end if

  if $error$ **and** $useERP$ **then** ▷ ERP module

    $y_f^p \leftarrow \mathbf{a}[1..i-1] \| s_{this}.state$

    $y_f^c \sim \pi(\cdot|p(x_i\|y_i, x_f\|y_f^p))$

    if $T(s_{this}, y_f^c) = s_{next}$ **then**

      $\mathbf{a}[i], error \leftarrow y_f^c, False$

    else

      $y_f^a \leftarrow$ FailedTactics2ATP(y_f^c)

      if $T(s_{this}, y_f^a) = s_{next}$ **then**

        $\mathbf{a}[i], error \leftarrow y_f^a, False$

      end if

    end if

  end if

  if $error$ **then** ▷ Resort to the last level

    $block \leftarrow$ InnermostBlock($i, \mathbf{a}$)

    if $block$ **is** Null **then return** Null

    $\mathbf{a}[block.start..block.end-1] \leftarrow$ Null

    $\mathbf{a}[block.end] \leftarrow$ sorry

    $i, s_{this} \leftarrow block.end, \mathbf{s}[block.start]$

  else

    $i, s_{this} \leftarrow i+1, s_{next}$

  end if

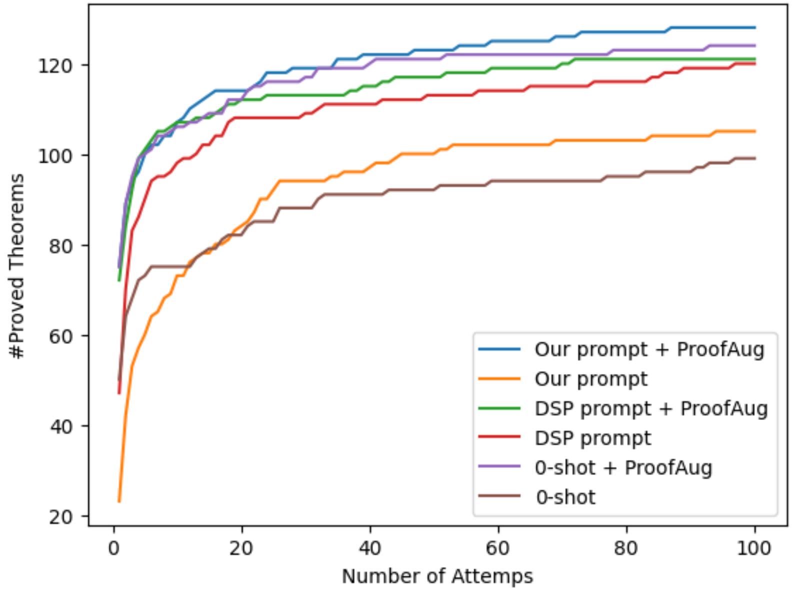
end while

return Concat($\mathbf{a}$) ▷ The final proof

Results

Table 2: **Comparison of methods using Isabelle as the proof assistant on MiniF2F-test.** For BFS methods, the sample budget $N \times S \times T$ corresponds to N attempts of S expansion with T iterations. As to tree-search methods, it becomes $N \times T$, with the same meanings for the symbols. A † indicates this result is obtained by using a mixed strategy.

Method	Model	Sample Budget	miniF2F-test
Methods using Isabelle			
DSP (Jiang et al., 2023)	CodeX	100	39.3%
Subgoal-XL(Zhao et al., 2024)	Fine-tuned Llama-8B	64	39.3%
		16384†	56.1%
LEGO-Prover(Wang et al., 2023)	mixed GPTs	100	50.0%
Lyra(Zheng et al., 2024)	GPT-4	100	47.1%
		200	51.2%
POETRY(Wang et al., 2024)	Fine-tuned ProofGPT (1.3B)	$1 \times 32 \times 128$	42.2%
Our Experiments (using Isabelle)			
DSP baseline	deepseek-math-7b-base	1	28.7%
		10	40.6%
		100	49.2%
ProofAug	deepseek-math-7b-base	1	36.5%(+7.8%)
		10	44.7%(+4.1%)
		100	52.5%(+3.3%)
ProofAug(0-shot)	deepseek-math-7b-base	500	54.5%
ProofAug(0-shot) + ERP	deepseek-math-7b-base	500	56.1%
Cumulative	deepseek-math-7b-base	1400†	61.9%
Cumulative + Dataset Curation	deepseek-math-7b-base	2100†	**66.0%**
Methods using Lean			
HTPS(Lample et al., 2022)	Evariste (600M)	64×5000	41.0%
RMaxTS(Xin et al., 2024b)	DeepSeek-Prover-V1.5-RL (7B)	$32 \times 6400^\dagger$	63.5%
BFS + CG(Wu et al., 2024)	InternLM2.5-StepProver (7B)	$256 \times 32 \times 600$	65.9%



Curation of miniF2F(Isabelle)

- Typos

```diff
 5   theory mathd_numbertheory_764                                   | 6   theory mathd_numbertheory_764
@@ -8,7 +9,7 @@ theory mathd_numbertheory_764
 8   begin                                                           | 9   begin
 9                                                                   | 10
 10  definition inv_mod::"nat \<Rightarrow> nat \                    | 11  definition inv_mod::"nat \<Rightarrow> nat \
     <Rightarrow> nat" where                                         |     <Rightarrow> nat" where
 11 -  "inv_mod d p = (SOME x. [x*p = 1] (mod p))"                   | 12 +  "inv_mod d p = (SOME x. [d*x = 1] (mod p))"
 12                                                                  | 13
 13  theorem mathd_numbertheory_764:                                 | 14  theorem mathd_numbertheory_764:
 14    fixes p :: nat                                                | 15    fixes p :: nat
```

- Minus for Nat.

```diff
5 isabelle/test/mathd_algebra_392.thy
@@ -7,8 +7,9 @@ theory mathd_algebra_392
 7   begin                                       | 7   begin
 8                                               | 8
 9   theorem mathd_algebra_392:                  | 9   theorem mathd_algebra_392:
 10 -  fixes n :: nat                            | 10 +  fixes n :: int
 11 -  assumes "even n"                          | 11 +  assumes "n > 0"
                                                 | 12 +    and "even n"
 12      and "(n - 2)^2 + n^2 + (n + 2)^2        | 13      and "(n - 2)^2 + n^2 + (n + 2)^2
     = (12296::int)"                             |     = (12296::int)"
 13    shows "((n - 2) * n * (n + 2)) / 8        | 14    shows "((n - 2) * n * (n + 2)) / 8
     = (32736::int)"                             |     = (32736::int)"
 14    sorry                                     | 15    sorry
```

- ~15 corrected compared with the DSP version, 4 in the PR to upstream

Lean 4 Implementation

- Lean 4 proofs are naturally less declarative compared to Isabelle
    - Nevertheless, Kimina-Prover-Preview takes a rather declarative way
    - We build a pre-parser inferring the block structures by indents

```
theorem aime_1983_p2 (x p : ℝ) (f : ℝ → ℝ)
    (h₀ : 0 < p ∧ p < 15) (h₁ : p ≤ x ∧ x ≤ 15)
    (h₂ : f x = abs (x-p) + abs (x-15) + abs (x-p-15)) :
    15 ≤ f x := by
  have h3 : abs (x - p) = x - p := by
    rw [abs_of_nonneg]
    linarith
  have h4 : abs (x - 15) = 15 - x := by
    rw [abs_of_nonpos]
    linarith
    all_goals linarith
  have h5 : abs (x - p - 15) = p + 15 - x := by
    rw [abs_of_nonpos]
    linarith
    all_goals linarith
  rw [h₂, h3, h4, h5]
  linarith
```

- No default hammer tools come with Lean 4
    - We choose Aesop, Omega, and a combination of some useful tactics for illustration

- Result
    - Pass@1 acc for Kimina-Prover-Preview-Distill-1.5B: 44.3% -> 50.4%
    - We are doing more extensive results

Takeaways

- Let the LLM generate the full proof, instead of a sketch first
  - This aligns with the pre-training data
  - ProofAug helps correct the mistakes in details!

- If ATPs cannot help find a proof from semi-proofs found by ProofAug …
  - Use the recursive proving module

Thank you!