A Minimalist Proof Language for Neural Theorem Proving over Isabelle/HOL

ANONYMOUS AUTHOR(S)

Neural Theorem Proving (NTP) employs Large Language Models (LLMs) to automate formal proofs in proof assistants. While LLMs have achieved relatively remarkable success in informal reasoning tasks using natural languages, the transition to mechanized formal theorem proving presents persistent challenges. Mechanized proof languages often contain many syntactic constructs and diverse, specialized proof tactics, which facilitate expert use but have no direct counterpart in informal mathematical proofs. These prover-specific idioms represent an additional burden for LLM-based NTPs that might be otherwise successful in generating informal proofs. Seeking to bridge this gap between formal proof construction and informal reasoning, in order to better facilitate NTP, this work approaches these challenges from a language design perspective. We look at common reasoning patterns in informal proofs and in existing mechanized proofs, and design *Minilang* — a minimalist proof language that captures these reasoning patterns. In contrast to proof languages (informal and formal) that often feature a large collection of operations with unclear semantic boundaries, Minilang is deliberately kept minimalist — its core design comprises only 10 proof operations, each with clear semantic distinctions. We further develop a rule-based translator from Isabelle's proof language (Isar) to Minilang, translating ~340K existing Isabelle proofs with an ~85% success rate. Using this translated corpus, we finetune two LLMs to compare machine learning performance on Minilang versus the original Isar language. Experiments show Minilang benefits the two LLMs by improving the pass@1 success rate on the PISA benchmark by up to 20/29 percentage points in comparison to the Isar-based LLMs w/wo Sledgehammer. The pass@1 rate reaches 69.1%, exceeding the prior work Baldur's pass@64 (65.7%); the pass@8 rate reaches 79.2%, exceeding the state-of-the-art on PISA (71.0%) achieved by Magnushammer.

ACM Reference Format:
Anonymous Author(s). 2018. A Minimalist Proof Language for Neural Theorem Proving over Isabelle/HOL. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Formal verification of software systems fundamentally relies on theorem proving to certify safety-critical properties. While interactive theorem proving (ITP) requires substantial manual effort and automated theorem proving (ATP) excels only within restricted domains, the rise of Large Language Models (LLMs) has introduced a promising alternative: Neural Theorem Proving (NTP). By using LLMs to interface with interactive theorem provers, NTP enables automated proof construction for complex properties [1]. Recent NTP systems have demonstrated remarkable success in mathematical competitions [2–4], achieving gold medal performance at the International Mathematical Olympiad [5], indicating promising reasoning and theorem-proving capabilities.

A fundamental challenge in NTP is the gap between natural-language-based informal reasoning and formal-language-based theorem proving. Prior works focus on training language models to master the existing languages of proof assistants [4, 6–8]. In this work, we take a complementary

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

approach by exploring the potential of proof language design focused specifically on facilitating NTP. Our key hypothesis is that the performance of language model-based neural theorem proving can be improved by reducing the prover-specific idioms and the complexity of the proof language.

We therefore propose *Minilang*, a minimalist proof language that *restricts its constructs to only those operations with natural counterparts in informal mathematical reasoning*. An example is illustrated in Fig. 1. For the proof obligations raised from those operations, Minilang delegates them to ATPs and instead focuses on high-level proof structures, as language models excel at proof outlining rather than fine-grained deduction. Its semantics is formalized using a simple tree-based state machine, which not only provides the foundation for establishing its soundness but also demonstrates the language's conceptual simplicity at the semantic level, maintaining consistency with its minimalist design philosophy.

We also present *Sledgehammer**, the ATP used by Minilang, which is an improved version of Sledgehammer. Our improvements lie in two aspects. First, Sledgehammer* incorporates a preprocessing step that simplifies and decomposes proof goals into subgoals more amenable to automated proving. Second, we train our proof generation models to produce relevant lemma hints to guide Sledgehammer*'s premise selection, allowing the models' global proof strategy to directly complement the hammer's local focus on individual subgoals.

Beyond this language design, we further contribute a rule-based translator to convert existing Isabelle corpora into Minilang for model training. This is challenging due to Isabelle's complex syntax and numerous corner cases. We propose three strategies to overcome this complexity: (i) *Elaboration*: Make implicit information explicit by exposing hidden details in a clear, structured manner; (ii) *Normalization*: Consolidate diverse approaches for achieving the same logical purpose into uniform representations; (iii) *Elimination of tactics*: Replace tactics with Sledgehammer*, except those that correspond well to informal proofs. Applying this translator at scale, we translate 85.25% of ~340K existing Isar proofs into Minilang. This translation distills Isar's versatile syntax and idiomatic reasoning into unified minimalist proof structures, eliminating unnecessary variations and theorem-proving idioms for more efficient LLM learning.

To quantify the gains from this representation distillation, we fine-tune two pretrained LLMs — Llemma [9] and Deepseek Prover Base v1.5 [10] — on both the translated Minilang corpus and the original Isar corpus for comparison. The results show that Minilang models achieve about a 29 percentage point improvement over Isar models.

Since Minilang incorporates ATP to discharge proof obligations while Isar does not, we conduct an ablation study to isolate the impact of language redesign from the ATP enhancement. As our baseline, we adopt the currently strongest known method, Thor's approach [11]. It augments Isar scripts by replacing tactics with Sledgehammer calls wherever applicable. Even against this stronger baseline, Minilang maintains a 20 percentage point advantage, demonstrating the effectiveness of its minimalist language design.

To measure the improvement provided by Sledgehammer* and our integration of relevant lemma hints, we conduct an ablation study that compares Sledgehammer* against the original Sledgehammer on both Minilang and Isar. The result shows at least 6 percentage points of improvement across all our metrics.

Contributions. To conclude this introduction, we summarize our contributions:

- We present Minilang, a minimalist proof language over Isabelle, designed to restrict language constructs to operations that align with informal proofs, to better support language model-based neural theorem proving.
- We develop a rule-based translator from Isar to Minilang that successfully converts 85.28% of 340K existing Isar proofs.

- We present two fine-tuned whole-proof generation models over Minilang, respectively based on Llemma and Deepseek-Prover Base v1.5.
- We present Sledgehammer*, an improved Sledgehammer that additionally takes relevant remise hints as input. We further provide two fine-tuned language models for generating these hints, constituting two automatic hammers.
- We develop an open-source socket-based Isabelle Read-Eval-Print-Loop infrastructure that scales to large compute clusters. We have deployed it on a cluster of 24 nodes × 128 CPUs.

Outline. The remainder of this paper is organized as follows. We begin by presenting our motivation in §2. In §3, we introduce Minilang, a minimalist proof language, along with Sledgehammer*, the enhanced ATP system it employs. We then propose an automated translation from Isar to Minilang in §4, which provides a substantial proof corpus for NTP training. Our experimental results are presented in §5, followed by a discussion of related work in §6. Finally, we conclude with the limitations and future work in §7.

2 Motivation

Early NTP works adopt LLM-guided proof search, where language models recommend the next tactic based on the current proof goal to heuristically guide the search process [15]. As LLMs continue to advance in reasoning capabilities, recent works have shifted toward *declarative theorem proving* [3, 4, 8, 16]. In this paradigm, language models construct proofs in a manner closer to informal pen-and-paper proofs — by declaring a series of subgoals to plan and propose proof strategies from a high-level perspective — thereby enabling models to more effectively leverage proof techniques learned from informal reasoning tasks.

In some purely declarative NTP works [11, 16], language models are used solely to draft high-level proof outlines, delegating all proof obligations to ATPs. The rationale is that language models are better suited for planning high-level proof strategies rather than fine-grained reasoning, while proof obligations of subgoals are better handled by ATPs with their deterministic search algorithms.

Ideally, the training and generation of declarative NTP models should be based on a simple formal language that can succinctly express proof outlines while maintaining alignment with informal pen-and-paper proofs. However, such an ideal language does not yet exist, despite the presence of all required declarative constructs in existing proof languages.

The problems are at least threefold: an overabundance of expert features, extensive syntactic redundancy, and substantial requirements for proof automation in declarative proving.

These expert features include specialized mechanisms and tactics for fine-grained control over the reasoning process. Such mechanisms include Isabelle's meta-logic encoding (e.g., generalized elimination), Rocq's `Ltac` tactic programming language, and Lean's `conv` mode for targeted rewriting. Such tactics include Isabelle's `blast lim:` *n* for tableaux reasoning with specific search depth, Rocq's `rewrite ... at ...` for specifying rewrite locations, and Lean's `aesop` with customizable rule sets. These features involve numerous configurations and technical details that impose substantial learning burdens on language models, particularly in NTP scenarios where training data is scarce.

Syntactic redundancy is particularly pronounced in Isabelle, the proof assistant underlying our work. Isabelle's proof language *Isar* admits multiple, often interchangeable proof idioms — such as fact chaining via connectives versus labels, and structured proofs versus apply-style scripts — yielding numerous ways to formalize identical proof procedures. The syntactic redundancy addressed in this work includes at least: 4 mechanisms for opening proof contexts (`subgoal`, `proof`, `{}`, `goal_cases`), 5 ways to apply tactics (`apply`, `by`, `proof`, `qed`, tactic combinators), 11 constructs for passing facts (`using`, `use-in`, `from`, `with`, `then`, `thus`, `hence`, `also`, `moreover`, `ultimately`, `finally`), 6 syntactic sugar for referring to facts and terms (`this`, `that`,

```
Theorem sqrt2_not_rational:
  ∀ pq : nat, q <> 0 → p*p = 2*(q*q) → ⊥.
intros p q; generalize p; clear p;
elim q using (well_founded_ind lt_wf).
clear q; intros q Hrec p Hneq;
pose proof Hneq as Hlt_0_q;
apply Nat.neq_0_lt_0 in Hlt_0_q;
intros Heq.
apply (Hrec (3 * q - 2 * p)
          (comparison4 _ _ Hlt_0_q Heq)
          (3 * p - 4 * q)).
apply sym_not_equal; apply lt_neq;
apply Nat.add_lt_mono_l with (2 * p);
rewrite ← plus_n_0; rewrite Nat.add_comm;
rewrite Nat.sub_add; auto with *.
apply new_equality; auto.
Qed.
```

(a) Rocq, from [12]

```
theorem sqrt_two_irrational {a b : ℕ}
  (co : gcd a b = 1) : a^2 ≠ 2 * b^2 :=
by rintro h : a^2 = 2 * b^2
  have : 2 ∣ a^2 := by simp [h]
  have : 2 ∣ a := dvd_of_dvd_pow prime_two this
  apply Exists.elim this rintro c aeq
  have : 2 * (2 * c^2) = 2 * b^2 := by
    simp [Eq.symm h, aeq]
    simp [pow_succ' _, mul_comm, ...]
  have : 2 * c^2 = b^2 := by
    apply mul_left_cancel₀ _ this decide
  have : 2 ∣ b^2 := by simp [Eq.symm this]
  have : 2 ∣ b := by
    exact dvd_of_dvd_pow prime_two this
  have : 2 ∣ gcd a b := by
    apply dvd_gcd . assumption . assumption
  have _ : 2 ∣ (1 : ℕ) := by simp [co] at *
  contradiction
```

(b) Lean, from [13].

```
theorem sqrt2_not_rational: "sqrt 2 ∉ ℚ"
proof
  let ?x = "sqrt 2"
  assume "?x ∈ ℚ"
  then obtain m n :: nat where
    "|?x| = m / n" and "coprime m n"
    by (rule Rats_abs_nat_div_natE)
  hence "m^2 = ?x^2 * n^2"
    by (auto simp add: power2_eq_square)
  hence eq: "m^2 = 2 * n^2"
    using of_nat_eq_iff power2_eq_square by force
  hence "2 dvd m^2" by simp
  hence "2 dvd m" by simp
  have "2 dvd n" proof -
    from "2 dvd m" obtain k where "m = 2 * k" ..
    with eq have "2 * n^2 = 2^2 * k^2" by simp
    hence "2 dvd n^2" by simp
    thus "2 dvd n" by simp
  qed
  with "2 dvd m" have "2 dvd gcd m n"
    by (rule gcd_greatest)
  with lowest_terms have "2 dvd 1" by simp
  thus False using odd_one by blast
qed
```

(c) Isabelle/Isar, from [14]

```
theorem sqrt2_not_rational: "√2 ∉ ℚ"
  RULE proof_by_contradiction
  CONSIDER "∃mn. |√2| = m/n ∧ coprime m n" END
  HAVE "m^2 = (√2)^2 * n^2" END
  HAVE eq: "m^2 = 2 * n^2" END
  HAVE "2 dvd m^2" END WITH eq
  HAVE "2 dvd m" END
  HAVE "2 dvd n"
    CONSIDER k where "m = 2 * k" END
    HAVE "2 * n^2 = 2^2 * k^2" END WITH eq
    HAVE "2 dvd n^2" END
  END
  HAVE "2 dvd gcd m n" END
  HAVE "2 dvd 1" END
END
```

(d) Minilang

Fig. 1. Minilang and mainstream proof languages on the same proof goal. While Lean and Isar incorporate natural-language-like structures, (e.g., build-in keywords such ashave and hence), their proofs are interspersed with tactics and other expert-oriented constructs that complicate learning. In contrast, Minilang aims to highlight the key proof outlines with close alignment to informal proofs.

Table 1. Core constructs of Minilang

Operation	Description
Declarative constructs	
HAVE	decomposing a proof goal into step-by-step subgoals.
CONSIDER$_\vee$	analyzing a proof goal by cases, e.g., consider the cases where x is positive, zero, or negative.
CONSIDER$_\exists$	binding variables to the witnesses of existential statements, e.g., consider a number p such that p is a prime greater than 2025.
Proof operations commonly found in informal proofs	
RULE	proving a goal by a specific mode of argument, e.g., arguing by contradiction for a given goal, and deriving $A \longrightarrow B$ and $B \longrightarrow A$ to show $A \longleftrightarrow B$.
CHOOSE	proving an existential statement by providing a witness.
SIMPLIFY	equivalently rewriting the proof goal into a simpler form.
CASE_SPLIT	applying structural case analysis to the goal.
INDUCT	applying induction to the goal.
END WITH *ps*	indicates that the target proof goal straightforwardly follows from the given premises *ps*.
NEXT	concludes the current goal's proof and moves to the next sibling goal when sibling goals exist. In terms of formalized semantics, it is an alias of **END**.
Necessary technical command	
INTRO	for management of variable and hypothesis context

`assms`, `prems`, `?thesis`, `?case`), and many semantically similar operations with subtle differences (`unfolding` vs. `unfold_tac` vs. `subst`, `cases` vs. `case_tac`, `induction` vs. `induct` vs. `induct_tac`).

While redundant syntax may be harmless to humans, it becomes problematic for NTP given corpus scarcity: it either dilutes limited training data or forces models to consume substantial data to learn that different forms are semantically equivalent.

Finally, if NTP models are to focus on proposing high-level proof outlines, the underlying proof assistant must provide powerful ATPs to discharge the proof obligations of the subgoals raised in the proof outlines. The capabilities of ATPs and NTP models are complementary: when ATP support is insufficient, NTP models must generate more fine-grained subgoals to make them amenable to automated proving. In the worst case, some declarative NTP systems must handle proof obligation discharge themselves, even resorting to using additional NTP models in place of ATPs.

These three challenges motivate us to propose Minilang, a minimalist proof language, and Sledgehammer*, an enhanced version of the well-known ATP, Sledgehammer. By *minimalist*, we mean Minilang's design aims to minimize its language constructs, retaining only declarative structures and operations that align well with informal proofs.

3 Minilang: A Minimalist Proof Language Mirroring Informal Proofs

Minilang's constructs, as listed in Table 1, consist of (1) declarative constructs for decomposing hard proof goals into simpler subgoals, (2) operations commonly found in informal proofs, and (3) one necessary technical command. These constructs, though minimalist, are sufficient for drafting high-level proof outlines, fulfilling the goals of the purely declarative paradigm. Minilang's high-level design does not incorporate a tactic system, instead delegating all proof obligations to ATPs.

As a pragmatic decision to maximize the success of our Isar translation procedure, we additionally extend Minilang with a small number of escape hatches back to Isar, which are generated infrequently in the training corpus but allow us to avoid individual local translation failures invalidating multiple dependent definitions (§4.2.3).

This section is organized as follows: §3.1 presents Minilang's syntax; §3.2 builds the proof model on which the semantics is formalized; §3.3 elaborates on the semantics; §3.4 details Sledgehammer*, the ATP used in MiniLang; §3.5 discusses MiniLang's soundness and relative completeness.

3.1 Syntax

A Minilang proof script comprises a sequence of commands whose syntax is defined as follows. The question mark (?) means that the clause can be omitted if the *facts* is empty.

Proof Script ::= Command$^+$
Command ::= **HAVE** *props* | **CONSIDER** *props* | **INTRO**
| **RULE** *fact* | **CHOOSE** *term* | **SIMPLIFY** *facts*$^?$
| **INDUCT** *the same argument syntax as Isabelle's* `induct` *tactic*
| **CASE_SPLIT** *the same argument syntax as Isabelle's* `cases` *tactic*
| **END** (**WITH** *facts*)$^?$ (**WITHOUT** *facts*)$^?$
| **NEXT** (**WITH** *facts*)$^?$ (**WITHOUT** *facts*)$^?$

3.2 Proof Model: State Machine over Trees

The semantics of Minilang commands are defined as transitions over a state machine (see Fig. 2). An example is illustrated in Fig. 4. Specifically, a state is either the special proof completion state or a labelled tree that hierarchically organizes subgoals into contexts,

State ::= Tree Tree ::= Leaf | (*label:* Context, *children:* Tree$^+$)
Context ::= (a set of variables, a set of named hypotheses)
Leaf ::= Context $\vdash$ Goal Goal ::= Term

where leaves represent unproven subgoals, and internal nodes group sibling subgoals that inherit a common context from decomposing a larger goal (e.g., lines 3, 9, 10).

For example, the resulting tree after executing the line 10 of Fig. 3 (**CONSIDER** $\exists k.\ m = 2 * k$) is given in the dashed box of Fig. 4. This tree contains three subgoals arranged left to right. The leftmost subgoal (opened by line 10) requires proving the existence of k. The middle subgoal, opened by **HAVE** E at line 9, is from the parent of the first subgoal. Its context has variable k fixed with condition $m = 2k$, allowing the proof for the middle subgoal to use this condition once the first subgoal establishes k's existence. The rightmost subgoal (False) is the top goal of the entire proof. It can similarly utilize the conclusion (2 dvd m) from the middle subgoal. Each leaf subgoal's context includes all labeled contexts from its ancestors in the tree, so all subgoals can access the previously established lemmas A, B, C, D.

Given the top-level proof goal G, the *initial state* is the tree with a single root node $((\varnothing, \varnothing) \vdash G)$ that represents the top goal G itself. In the `sqrt2_not_rational` example, this initial state is,

$$\Big((\varnothing, \varnothing) \vdash \sqrt{2} \notin \mathbb{Q}\Big) \qquad \text{(Initial State)}$$

Given a proof script as a sequence of commands, Minilang's proof system verifies this proof script by executing each of its commands successively to transition the state machine. This execution yields three results:

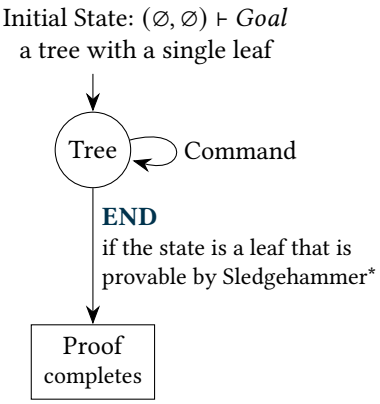


Fig. 2. Minilang's state machine.

```
1  theorem sqrt2_not_rational: "√2 ∉ ℚ"
2    RULE proof_by_contradiction
3    CONSIDER "∃mn. |√2| = m/n ∧ coprime m n"
4        END
5    HAVE B: "m^2 = (√2)^2 · n^2" END
6    HAVE eq: "m^2 = 2 · n^2" END
7    HAVE C: "2 dvd m^2" END WITH eq
8    HAVE D: "2 dvd m" END
9    HAVE E: "2 dvd n"
10     CONSIDER "∃k. m = 2k"
11       END
12     HAVE "2 * n^2 = 2^2 · k^2" END WITH eq
13     HAVE "2 dvd n^2" END
14   END
15   HAVE I: "2 dvd (gcd m n)" END
16   HAVE J: "2 dvd 1" END
17 END
```

Fig. 3. An example written in Minilang.

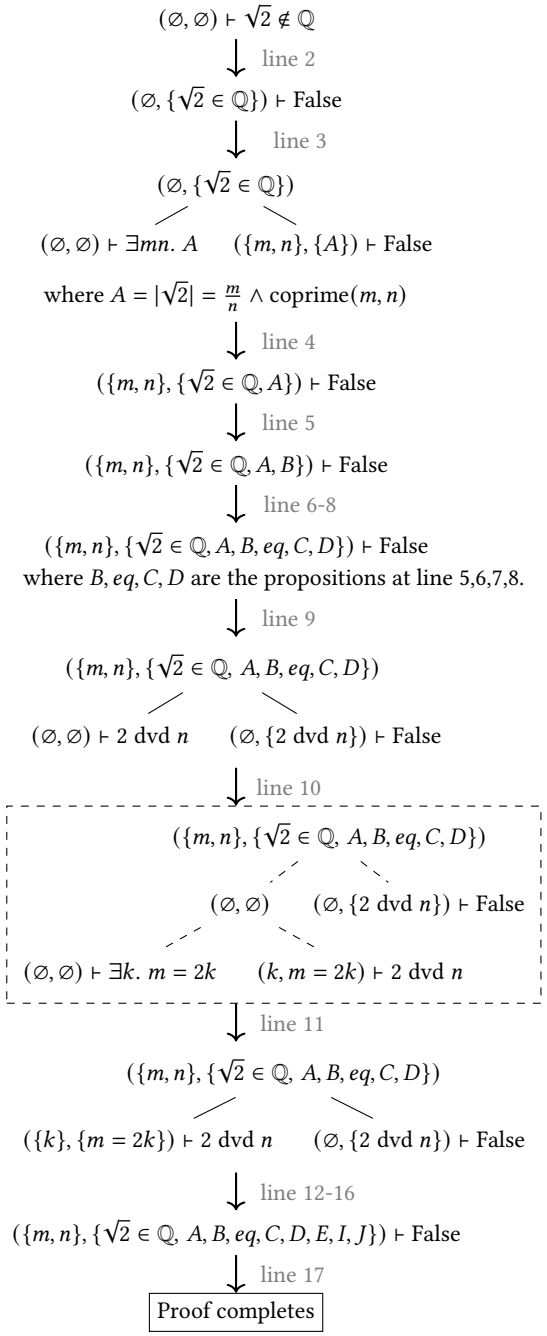


Fig. 4. The transition of the tree state in the example of Fig. 3. Each node represents the state after the execution of the labeled line.

(1) The execution gets stuck at one of the commands because no corresponding transition rule can be found, indicating proof failure.
(2) All commands are successfully executed through to the last one, and the machine reaches a special terminal state (the square node in Fig. 2) that represents proof completion.
(3) All commands are successfully executed through to the last one, but the machine does not reach the proof-complete state, indicating that the proof script is incomplete and the proof remains unfinished.

3.3 Semantics

The semantics of Minilang commands are formalized as transition schemas presented in Fig. 5. Each transition rewrites only the leftmost leaf or the leftmost non-leaf node of the state tree; consequently, it suffices to describe the change at that leftmost position. We use a schematic diagram $X^{\mathcal{R}}_{\mathcal{S}}$ to represent an arbitrary tree whose leftmost leaf is X. The schematic variable $\mathcal{R}$ represents X's parent node and all its upward context, while the schematic variable $\mathcal{S}$ represents potentially multiple sibling subtrees of X. As a special case, a schematic diagram $X^{\mathcal{R}}_{\mathcal{S}}$ can be instantiated to a tree with a single root node X, where $\mathcal{R}$ and S are considered instantiated to empty.

Since all transitions act on the leftmost subgoal, we refer to that leftmost goal as the current goal in what follows. We now describe each command's semantics.

HAVE $G_1, \cdots, G_n$ is the key to decomposing a proof goal G_0 into subgoals $G_1, \cdots, G_n$. It decomposes the proof problem about the goal G_0 into: 1) proving the subgoals $G_1, \cdots, G_n$ successively; and then 2) using their proved conclusions as lemmas to prove the original goal G_0.

CONSIDER combines two functions: (a) case-analysis (e.g., consider the cases where x is positive, zero, or negative) and (b) existential-witness extraction (e.g., let p be a prime greater than 2025). This design is logically reasonable because both of the functions perform elimination of disjunctive connectives ($\exists$ and $\vee$). However, for clarity, we elaborate on these two functions separately.

Illustrated in Fig. 6a, **CONSIDER** $P_1 \mid \cdots \mid P_n$ splits the proof of the current goal G into n cases. This operation produces $n+1$ subgoals, where the first goal $P_1 \vee \cdots \vee P_n$ verifies that the case split is exhaustive, namely, $P_1, \cdots, P_n$ are all the possible situations and no other case is missing from consideration. Once the exhaustiveness is proven, the proof proceeds with n separate branches, each proving the goal under the assumption of case P_i.

Illusrated in Fig. 6b, **CONSIDER** $\exists \bar{x}.\ P(\bar{x})$ introduce a sequence of fresh variables $\bar{x}$ and bind them to certain terms satisfying the condition $P(\bar{x})$, once the existence of such terms is shown. Regarding the notation, $\bar{x}$ represents a sequence of variables. This command generates 2 subgoals: The first subgoal asserts the existence of such terms satisfying P; the second subgoal augments the original goal G by binding $\bar{x}$ to the terms satisfying P.

HAVE and CONSIDER introduce lemmas within the current goal's context. However, we have not yet provided a command for introducing variables and hypotheses into the context. Without it, every context would remain trivially empty. This is the motivation for the **INTRO** command. The command rewrites the current goal by moving into the context all universally quantified variables ($\bar{x}$ in Fig. 5) and all hypotheses ($\{H_i\}_{1 \le i \le n}$ in Fig. 5) occurring in the goal proposition, so that these variables and hypotheses are in scope for subsequent lemmas.

While **INTRO** is admittedly technical, it is necessary for explicit context management. Minilang offers an option to automatically invoke INTRO before HAVE and CONSIDER, allowing proofs to more closely resemble informal mathematics at the cost of less fine-grained control. In our experiments, we disable this option and require models to use INTRO explicitly.

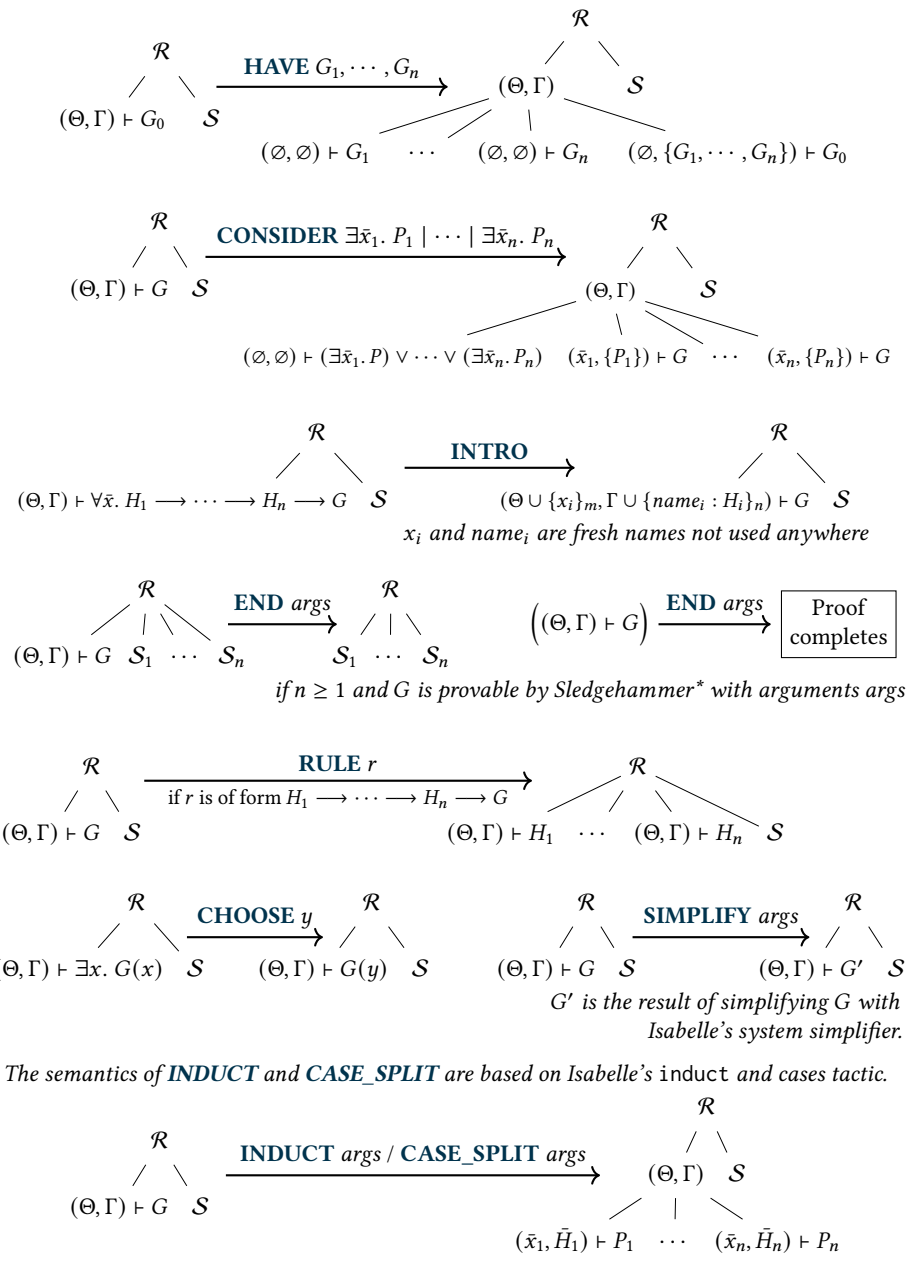
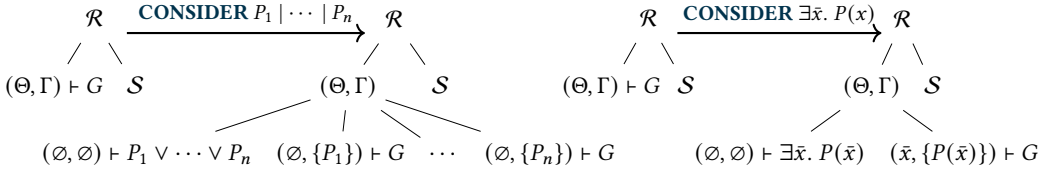


Fig. 5. The semantics of Minilang commands. Notation $\bar{x}/\bar{H}$ denote a sequence of variables/hypothesis



(a) One function: Case-analysis (b) The other function: Extential-Witness Extraction

Fig. 6. The semantics of **CONSIDER** combines two functions.

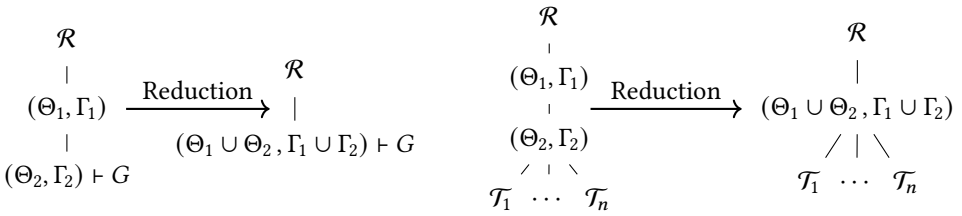


Fig. 7. Minilang's semantics additionally incorporates two automatic reduction rules. $\mathcal{T}_1, \cdots, \mathcal{T}_n$ are schematic variables denoting arbitrary subtrees.

END and **NEXT** are syntactic aliases of one another that are semantically equivalent. They correspond to the common idiom in informal proofs — "the goal is easy to show" or "the goal follows straightforwardly from the premises". Operationally, these commands conclude the proof of a subgoal by invoking Sledgehammer* to discharge its proof obligations. If Sledgehammer* fails to prove the goal or times out, the proof execution terminates with an error.

The motivation of introducing the **NEXT** alias is solely for readability: the word "next" naturally reads as "finish this subgoal and move to its sibling".

The first **END** rule in Fig. 5 removes the leftmost subgoal when it has siblings, but no rule handles the case where it is an only child. This does not compromise completeness: two automatic reduction rules (Fig. 7) maintain the invariant that every non-leaf node has at least two children. These rules collapse any single-child node into its parent by merging their contexts and are applied preemptively before all other rules.

The second **END** rule concludes the entire proof, transitioning to the terminal "proof complete" state. The rule applies only when the proof state is a single-node tree.

The foregoing commands comprise all declarative structures in Minilang. We now present commands that align with common proof operations in informal mathematics.

The **RULE** command corresponds to adopting a specific route of argument in an informal proof (e.g., proof by contradiction). Such a route of argument is encoded as an inference rule of shape, $H_1 \longrightarrow \cdots \longrightarrow H_n \longrightarrow G$, which indicates that the goal G can be reduced to the subgoals $\{H_i\}_{1 \le i \le n}$. For example, the rule `proof_by_contradiction` in Fig. 3 has form, $(P \longrightarrow \text{False}) \longrightarrow \neg P$.

The command **CHOOSE** supplies a witness for an existential goal. When the goal is to prove $\exists x.\ G(x)$, CHOOSE y provides a concrete term y and reduces the goal to proving $G(y)$. The command **SIMPLIFY** *args* corresponds to the simplification operation in informal proofs. It is implemented by invoking Isabelle's system simplifier on the proof goal, with argument *args*. The command **CASE_SPLIT** and **INDUCT** perform structural case analysis and induction. They are implemented by Isabelle's `cases` and `induct` tactics.

3.4 Sledgehammer*: An Improved Sledgehammer

All proof obligations in Minilang are delegated to an improved Sledgehammer which we call Sledgehammer*. The improvement (of Sledgehammer* itself and the way we incorporate it) includes three aspects.

3.4.1 Premise selection aided by proof generation models. The **END** and **NEXT** commands incorporate a specific syntax to annotate the relevant lemmas that Sledgehammer* could consider using as premises when carrying out the proof search, and the lemmas that should be avoided.

END / **NEXT** **WITH** *relevant-lemmas* **WITHOUT** *avoided-lemmas*

Our translation pipeline employs a specific pass (§4.2.4) to extract the relevant and avoided lemmas from ATP proofs and annotate them in the training corpus. This allows proof generation models to learn the relevant and avoided lemmas through the annotations in the corpus, as well as how these lemmas relate to both the local proof step and the overall proof strategy. Consequently, the models can generate relevant/avoided lemma annotations from a global proof strategy perspective.

On the implementation side, we reuse Sledgehammer's existing interface for relevant/avoided lemma hints. Sledgehammer's original premise selection is based on heuristics [17] and classical machine learning methods like k-NN [18]. We continue to use this premise selection mechanism in addition to our explicit hints. We include a preprocessing pass that filters out undefined references produced by language-model hallucinations. This prevents syntax errors and lets proofs proceed even when some annotations are invalid.

3.4.2 Integration with simplification system. The next improvement of Sledgehammer*is its integration with Isabelle's simplification system, a term rewriting engine enhanced with programmable simplification procedures. This system is notably powerful because any decision problem in Isabelle/HOL can reduce to rewriting propositions to True, allowing arbitrary decision procedures to be embedded as simplification procedures. These embedded procedures are often complementary to SMT-based Sledgehammer, and can discharge many goals that Sledgehammer alone fails to prove.

Specifically, the improvement comprises two parts: First, tactic `auto` is applied before invoking Sledgehammer. It rewrites the goal using the system simplifier and may split it into multiple, simpler subgoals — cases where Sledgehammer often fails on the original goal but can succeed on the decomposed subgoals one by one. However, `auto` can be time-consuming on complex goals, so we impose a timeout and, if it expires, fall back to the safer tactic `clarsimp`. Second, Sledgehammer* runs the simplification-based brute-force tactic `fastforce` in parallel with Sledgehammer as an extra ATP backend.

Moreover, the relevant/avoided lemma hints are also used to augment the simplifiers invoked by Sledgehammer*. A challenge is that these lemmas should not be fed naively as undifferentiated inference rules, because the simplifiers distinguish rule roles (e.g., rewriting, splitting, and introduction/elimination in tableaux-style reasoning). We develop a heuristic to guess the intended role of a given rule and configure the simplifiers accordingly.

This heuristic combines name-based and expression-based analysis. Rule names in Isabelle follow certain naming conventions — for example, rewriting rules typically end in `_simp`, and splitting rules typically end in `.splits`. If a rule's name matches one of these conventions, the heuristic classifies it accordingly. Second, many role-specific rules exhibit characteristic shapes; for example, elimination rules always have the form $(term_1 \longrightarrow ?C) \longrightarrow \cdots \longrightarrow (term_n \longrightarrow ?C) \longrightarrow ?C$ where $?C$ is a schematic variable. The heuristic pattern-matches such shapes in a rule's statement to determine its role. Finally, if none of these signals determines a role, the heuristic defaults to classifying the rule as a rewriting rule, which is the most common case.

3.4.3 Caching proofs. Another minor improvement is that Sledgehammer* records all obtained proofs in a persistent cache keyed by the goal's expression. Because Sledgehammer is time-consuming and somewhat nondeterministic, caching results substantially accelerates proof replay.

3.5 Soundness and Relative Completeness

Theorem 1 (Soundness). *If Isabelle's simplifiers, tactic* `cases`*, tactic* `induct`*, and Sledgehammer* are sound, then* Minilang *is sound: The execution of a proof script reaches the terminal "proof completes" state only if the target proof goal is semantically valid.*

The formalization of Minilang's semantics is tied to Sledgehammer*. If we generalize the semantics to be parametric in the choice of ATP (rather than fixed to Sledgehammer*), we can show that there exists an ATP with respect to which Minilang is at least as proof-theoretically complete as a Natural Deduction system.

Theorem 2 (Relative Completeness). *Given a Natural Deduction system ND, there exists an ATP such that any proposition provable by ND is provable by Minilang with respect to the ATP.*

Proof. The idea is to use HAVE statements to replay the proof tree of this provable proposition. Write all the formulas in the tree's nodes into a sequence of HAVE statements, from the leaves to the root, by a post-order traversal. Note that the proof for every statement requires only one *ND* rule application. We show that an ATP exists to prove all the HAVE statements: Because the number of *ND* rules and possible choices of their operands is finite, an ATP can enumerate all possible applications to find the correct one. □

4 Translation from Isar to Minilang

Training NTP models requires substantial proof data, which does not exist for our newly designed language. We address this through automated translation from Isar to Minilang, successfully converting 85.28% of 340K existing Isar proofs obtained from Isabelle's Archive of Formal Proofs (AFP) [19] and Isabelle/HOL system library. This process of eliminating non-essential components and concepts from Isar to ultimately obtain Minilang also reflects the differences between Minilang's minimalist design and Isar's design.

4.1 Background

Isar is a powerful and intricate language. Its diverse syntax and rich features pose substantial challenges for translating into Minilang.

4.1.1 Declarative Proof Structure. Isar provides a suite of declarative statements for various functions, such as `have` for introducing subgoals, `obtain` for extracting witnesses from existential statements, and `assume` for assumption management. All such statements must appear inside a declarative proof block.

A declarative proof block begins with **proof** and ends with **qed**. When multiple goals are present, command **next** can be inserted within a proof-qed block to close the current block and open a fresh one at the same level. However, a complication is that, **next**-separated blocks do not correspond one-to-one, in sequence, to the proof goals. Instead, Isar uses statement **show** *goal* **by** *tactic* to explicitly target a particular *goal* and discharge it with a *tactic*. Moreover, a block may contain multiple **show** statements proving multiple goals in any order, and even a single **show** can target several goals. Fig. 8 illustrates an example.

This intricate proof context in Isar complicates translation to Minilang, because in Minilang proof blocks have a simple one-to-one, sequential correspondence with proof goals; the translation must therefore normalize Isar proofs to Minilang's block organization.

assume we face multiple proof goals $G_1, \cdots, G_4$
proof (*some tactic*)
  have *some lemma* A **by** ...
  then show G_3 **by** *a tactic that may use* A
  moreover with A **have** *some lemma* B **by** ...
  finally show G_1 **by** *a tactic that may use* A, B, G_3
next
  obtain x **where** x *satisfies some condition* $C(x)$
  hence G_2 **and** G_4 **by** *a tactic that may use* $C(x)$
qed

Fig. 8. An example of complications in Isar's proof structure. **hence** ≡ **then show** is an abbreviation.

proof (*some tactic*)
  have A **by** ...
  show G_3 **using** A **by** *tactic*
  have B **using** A **by** ...
  show G_1 **using** G_3, B **by** *tactic*
next
  obtain x **where** $C(x)$
  show G_2 **and** G_4 **using** $C(x)$ **by** *tactic*
qed

Fig. 9. Normalization of the connectives in Fig. 8.

4.1.2 Connectives. As illustrated in Fig. 8, Isar uses a variety of connectives such as **then**, **moreover**, **finally**, and **hence** to chain proof steps within a proof block. Semantically, they pass previously derived lemmas or conclusions as arguments to subsequent proof methods. Hence, all such connectives can be uniformly expressed as **using** <lemmas> **by** <method>. For instance, Fig. 9 shows the example from Fig. 8 rewritten in this normalized form.

4.1.3 Redundant Syntax. As mentioned in §2, Isar incorporates substantial syntactic redundancy — numerous ways to perform the same logical operation. Concrete examples include that Isar supports both forward, declarative proof construction (§4.1.1) and backward, tactic-based reasoning. Although the **proof**–**qed** structure already supports opening proof contexts in the declarative style, Isar supplies the **subgoal** command to provide essentially equivalent support in the tactic style (Fig. 10). Moreover, even within the declarative style, Isar offers redundant mechanisms for opening proof contexts. An instance is the block syntax (Fig. 11), whose effect is equivalent to a standard **proof**–**qed** block. In §4.2.2, we devote significant effort to normalizing these variants.

4.2 The Translation Process

The translation process includes 20+ passes, each of which represents one step from an intermediate language to another closer to Minilang. These passes fall into four stages:

(1) Parsing the given Isar proof script into an Abstract Syntax Tree (AST).
(2) Elaborating Isar's syntactic sugars & normalizing redundant proof idioms. This phase corresponds to the first two translation strategies mentioned in §1 (page 2).
(3) Translating into Minilang (mapping the Isar AST into Minilang AST).
(4) Refinement: eliminating tactics successively with Sledgehammer*. This corresponds to the last strategy in §1 (page 2).

We discuss each stage in the subsections that follow.

4.2.1 Parsing. Isar is implemented in Isabelle/ML, and so is our translator. This allows us to reuse Isar's parser to extract ASTs, ensuring correct handling of Isar's various syntactic corner cases. Concretely, we reuse Isar's lexer in full and modify the final stage of the parser to emit our custom AST. This custom AST comprises 42 node kinds, covering a broad range of Isar constructs.

4.2.2 Elaboration & Normalization. The stage (i) elaborates syntax sugars, making implicit information explicit, and (ii) normalizes equivalent or near-equivalent ways of proof formalization into a

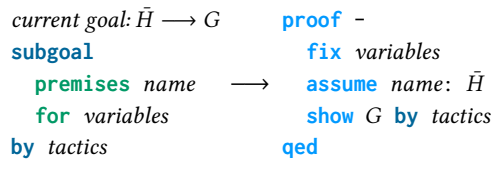


Fig. 10. Transforming **subgoal** into **proof-qed**.

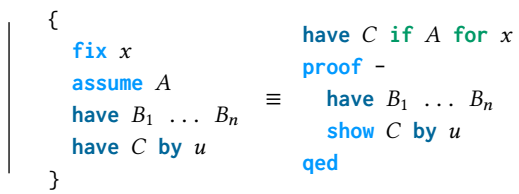


Fig. 11. Rewriting bracket block into **proof-qed**.

uniform representation. This reduces the AST from 42 node kinds to 15, simplifying the subsequent translation to Minilang.

Many similar ways of proof formalization are not strictly equivalent: they coincide in most situations but diverge in certain corner cases, exhibiting subtle semantic differences. Our normalization attempts to account for these cases, but cannot cover them exhaustively. Consequently, normalization may change the semantics of a proof script. We therefore re-run Isabelle's proof checker to ensure that the modified proofs remain valid, and discard any that fail. This normalization step contributes a major source of our translation failures.

Major steps in the elaboration & normalization process are presented as follows, starting with localized transformations:

- Normalize all connectives into **using** with the lemmas' explicit names. As mentioned in §4.1.2, this is implemented by rewriting rules such as

  have A ... **then have** B **apply** m ≡ **have** A ... **have** B **using** A **apply** m
  with A **have** B **apply** m ≡ **have** B **using** this A **apply** m

- Assign anonymous lemmas with generated names. This prepares the next pass.
- Resolve pronouns `this`, `that`, `prems`, and `assms`. Depending on context, these pronouns refer to the recent lemmas or assumptions. Our translator replays the Isar proof to obtain each pronoun's binding context, uses Isabelle/ML internals to resolve the reference, and replaces it with the generated name from the previous pass, making all references explicit.
- Unfolding the macro variables `?thesis` and `?case` that refer to proof goals in local contexts. We use the same approach as above, invoking Isabelle's internal interfaces to resolve these references and replace them with explicit terms.
- Isar allows lemmas to be referenced either by name or by their full expressions. This pass replaces expression-based references with name-based ones.
- Add type annotations to variables and numbers. Although Isabelle's type inference works well in most cases, it sometimes fails to infer types for certain variables or numbers, causing proofs to fail. In this pass, the translator explicitly annotates all variables and numerals with their types and retains these annotations in the final training corpus, so the model learns to emit the same type markings at generation time and thus avoids such failures.

Next are transformations on proof structure:

- Reorganize Isar's **proof-qed** structure to align with Minilang's one-block-per-subgoal organization. This reorganization process is illustrated in Fig. 12. As noted in §4.1.1, a single Isar proof-qed block may contain several **next**-delimited sub-blocks; each sub-block may include multiple **show** statements; and each **show** may address multiple goals. To reorganize this nested structure, the translator first splits each multi-goal **show** into single-goal **show** statements. The split also applies to the statement's associated tactic sequence: our translator tracks the effect of each tactic application to determine which goal it addresses,

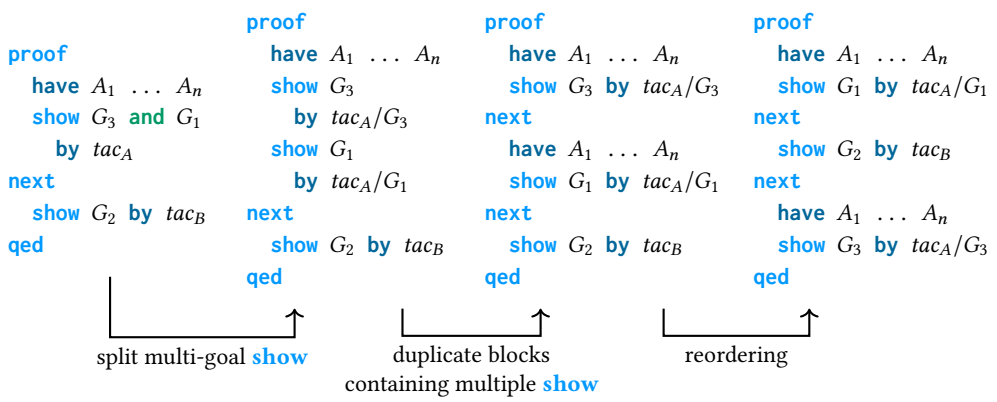


Fig. 12. Restructuring Isar's **proof**-**qed** blocks to align with Minilang's one-block-per-subgoal organization. Notation t/G_i denotes the tactic t's segment for goal G_i

and duplicates any tactic that affects multiple goals. Next, any proof block that contains multiple **show** statements is duplicated (as illustrated in Fig. 12) so that each resulting block contains exactly one **show**. In addition, a dependency analysis is performed to remove unused lemmas from the duplicated blocks. Finally, the blocks are reordered to follow the sequence of the proof goals.

- Normalize each **subgoal** statement into **proof**-**qed** block using the transformation in Fig. 10.
- Rewrite each bracket structure into a **proof**-**qed** block (Fig. 11).
- Normalize tactic `goal_cases` into the standard **proof** - **case** - **qed** structure.
- In Isar, variable declarations by **fix** and assumption declarations by **assume** may appear anywhere within a **proof**-**qed** block, interleaved with other proof operations. This pass normalizes that freedom by moving all such declarations to the beginning of their containing block. Additionally, not all variables and assumptions in Isar are declared explicitly via **fix**/**assume**; some are introduced implicitly within **show** statements. This pass lifts such implicit introductions to explicit declarations at the block header. Consequently, each **proof**–**qed** block is normalized into a form amenable to direct translation to Minilang's **INTRO** command, via the mapping in Fig. 13 as discussed later in §4.2.3.

Finally, we consider elaboration and normalization of tactic sequences:

- Tactic application in Isar may appear in different syntax (**proof**, **qed**, **apply**, and **by**). This pass rewrites these occurrences so that every tactic application is expressed uniformly as **apply**, allowing subsequent passes to consider **apply** only. This rewrite is implemented by rules like, **proof** (*tactic*) ≡ **apply** (*tactic*) **proof** – , and **by** (*tactic*) ≡ **apply** (*tactic*) **done**.
- Isar tactics can be combined by combinators (`,` `+` `?` `|` `[n]`). This step eliminates combinators and normalizes composite tactics into sequences of atomic tactics. This is implemented by tracing the execution of composite tactics and expanding each composite into the exact sequence of atomic invocations that actually run.
- Isar tactics may operate on multiple goals (e.g., `auto` affects all open goals). By contrast, Minilang requires operations for different proof goals to be delimited by **NEXT**; a single operation is disallowed to range over multiple goals. As a result, the translation must identify, for each goal, the sequence of tactics that act on it. This is implemented by tracing

$$\left[\!\!\left[\begin{array}{l}\textbf{fix}\ vars\\ \textbf{assume}\ assms\\ operations\\ \textbf{show}\ goal\\ tactic\ sequence\end{array}\right]\!\!\right] \triangleq \left(\begin{array}{l}\textbf{INTRO}\\ [\![operations]\!]\\ [\![tactic\ sequence]\!]\end{array}\right) \qquad \left[\!\!\left[\begin{array}{l}\textbf{proof}\ -\\ \quad operations_1\\ \textbf{next}\\ \quad \cdots\\ \textbf{next}\\ \quad operations_n\\ \textbf{qed}\end{array}\right]\!\!\right] \triangleq \left(\begin{array}{l}[\![operations_1]\!]\\ \textbf{NEXT}\\ \cdots\\ \textbf{NEXT}\\ [\![operations_n]\!]\end{array}\right)$$

$$[\![\textbf{apply}\ (tactic)]\!] \triangleq \textbf{APPLY}\ (tactic) \qquad [\![\textbf{done}]\!] \triangleq \textbf{END}$$

$$[\![\textbf{have}\ propositions]\!] \triangleq \textbf{HAVE}\ propositions \qquad [\![\textbf{consider}\ cases]\!] \triangleq \textbf{CONSIDER}\ cases$$

$$[\![\textbf{obtain}\ vars\ \textbf{where}\ conditions]\!] \triangleq \textbf{CONSIDER}\ \exists vars.\ conditions$$

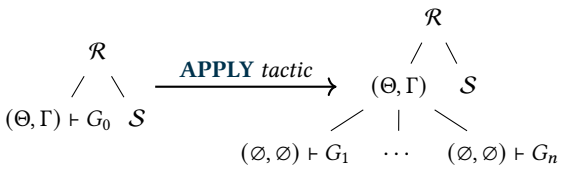
$$[\![\textbf{ForGoals}(seq_1, seq_2, \cdots, seq_n)]\!] \triangleq [\![seq_1]\!]\ \textbf{NEXT}\ [\![seq_2]\!]\ \textbf{NEXT}\ \cdots\ \textbf{NEXT}\ [\![seq_n]\!]$$

Fig. 13. Selected translation rules from Isar to Minilang. $[\![\cdot]\!]$: Isar → Minilang is the translation mapping.

every tactic invocation and recording its target goals; if an invocation spans several goals, we duplicate it once per goal and confine each copy to that goal.
The resulting per-goal sequences are represented as an AST node **ForGoals**$(seq_1, \cdots, seq_n)$, where each seq_i can only be a sequence of **apply** and/or **ForGoals**. This ForGoals AST node is subsequently translated into Minilang's **NEXT** in §4.2.3 by the rule in Fig. 13.

4.2.3 Translating into Minilang. By this stage, most redundant Isar constructs have been eliminated, and the proof scripts have been normalized. From their normalized forms, a small set of mapping rules suffices to translate the scripts into Minilang. A subset of these rules is shown in Fig. 13.

Note that tactic applications have not yet been eliminated at this stage. To express them in Minilang, we extend Minilang with an **APPLY** command to denote a tactic application. Its semantics is defined as follows, which mirrors Isar's **apply** but acts only on the first subgoal.



if applying *tactic* to G_0 yields subgoals $G_1, \cdots, G_n$

Additionally, to translate as much Isar as possible into Minilang, we extend Minilang with two auxiliary commands: one configures Isabelle's attribute system (e.g., to register local simplification rules), and the other activates Isabelle's bundle and module mechanisms. These commands account for ~1.01% of the translated code, and we defer their details to our supplementary materials.

4.2.4 Refinement. Tactic applications still remain after the translation step in 4.2.3. Since Minilang follows a purely declarative paradigm (§2) that discourages the use of tactics, the final stage of the translation process aims to replace these remaining tactics with Sledgehammer* wherever possible.

By this stage, Minilang scripts express all tactic applications as sequences of **APPLY** commands that terminate with **END** or **NEXT**. Attempting to eliminate these tactics, we repeatedly apply the following substitutions, which progressively absorbs each **APPLY** command immediately preceding an **END** or **NEXT** into that terminal statement.

$$\textbf{APPLY}\ tactic\ \textbf{END}\ \textbf{WITH}\ w\ \textbf{WITHOUT}\ wo \quad \mapsto \quad \textbf{END}\ \textbf{WITH}\ w\ lem^+\ \textbf{WITHOUT}\ wo\ lem^-$$
$$\textbf{APPLY}\ tactic\ \textbf{NEXT}\ w\ \textbf{WITHOUT}\ wo \quad \mapsto \quad \textbf{NEXT}\ \textbf{WITH}\ w\ lem^+\ \textbf{WITHOUT}\ wo\ lem^-$$
if, Sledgehammer can still prove the goal after removing the tactic*

Recall from §3.4 that a key innovation of this work is that we extract both relevant lemmas and lemmas to avoid as annotations into the training corpus, enabling proof generation models to learn how to suggest these hints to assist Sledgehammer*'s premise selection. In the substitution rule above, lem^+ and lem^- represent respectively the relevant and avoided lemmas extracted from the arguments of *tactic*. An example is `auto add:` lem^+ `del:` lem^-. These lemmas are merged into the **WITH** and **WITHOUT** clauses as hints, where *w* and *wo* represent the lemma hints already present before the substitution.

The extraction of the lem^+ and lem^- is performed by a heuristic we developed. It recognizes the argument syntax of 44 common Isabelle tactics to identify relevant/avoided lemmas, and defaults to treating all lemma arguments as relevant for unrecognized tactics.

The substitutions continue until either no **APPLY** remains or further substitution would cause **END** or **NEXT** to fail to prove the goal using the Sledgehammer*. In the latter case, we are forced to retain the remaining **APPLY** commands in the training corpus, which represents a compromise to the purely declarative ideal advocated in §3. Fortunately, with the aid of relevant lemma annotations, >95% of tactics are successfully eliminated, leaving APPLY commands accounting for only 6.7% of all translated Minilang commands.

As an exception, when tactics correspond to common proof operations that Minilang directly supports, the tactics are substituted into the corresponding commands instead of **END** or **NEXT**,

APPLY (`auto` *args*) ↦ **SIMPLIFY** *args* **APPLY** (`simp` *args*) ↦ **SIMPLIFY** *args*
APPLY (`cases` *args*) ↦ **CASE_SPLIT** *args* **APPLY** (`induct` *args*) ↦ **INDUCT** *args*

5 Evaluation

The translation method in §4 provides a way to construct Minilang corpora for training language models. This allows us to evaluate Minilang's effectiveness for NTP through fine-tuning LLMs. In this section, we conduct an ablation study by fine-tuning 2 pretrained LLMs under 3 ATP configurations (Sledgehammer*, original Sledgehammer, and no ATP) for both Minilang and Isar, yielding 12 models in total to compare their pass rates on the same benchmark (PISA [20]).

Based on the evaluation results, we answer the following research questions:

RQ 1: Can we effectively improve declarative NTP by redesigning the underlying proof language?
RQ 2: Is improvement confined to syntax error reduction, or are reasoning errors also reduced?
RQ 3: For a redesigned proof language, how can we obtain effective training corpora for it?

In what follows, §5.1 analyzes the translation results and the corpus features; §5.2 details the ablation study design and fine-tuning configurations; §5.3 analyzes the experiment results.

5.1 Analysis of the Translation Results

As discussed in Section 4.2.2, Isar's language features encompass numerous corner cases that our translator cannot exhaustively cover. Additionally, Isar is extensible, and many works in AFP define custom commands that fall outside our translator's scope. These factors limit our translation success rate to ~85.28%. Nevertheless, the resulting 290K proofs are still a substantial corpus sufficient for supervised fine-tuning.

To examine the quality of the translation, we analyze the occurrence frequency of every command in the translation result. The result is

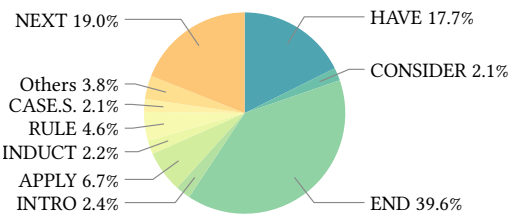


Fig. 14. Distribution of Minilang commands in the translation result.

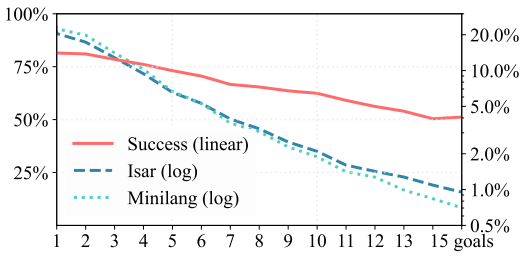


Fig. 15. Translation success rate (solid line) and distributions of Isar (dashed) and Minilang proofs (dotted) over the number of subgoal declarations.

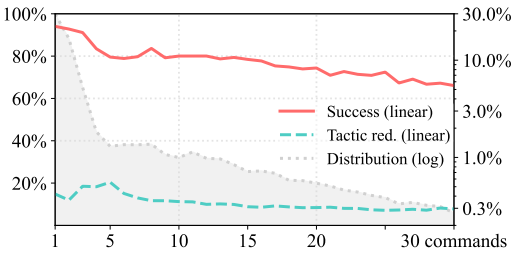


Fig. 16. Translation success rate (solid line), the ratio of tactic counts after/before translation (dashed line), and the distribution of Isar proofs (blue shadow) over the number of Isar commands.

shown in Fig. 14, where **END** and **NEXT** are the most frequent commands. This is because (1) every goal and subgoal must be closed by END or NEXT, and (2) 42.65% of Isar proofs can be proven directly by Sledgehammer* with a single END statement, requiring no other proof steps.

Subgoal declarations (**HAVE** and **CONSIDER**) constitute the next largest portion, confirming their central role in declarative proofs. To evaluate how well our translation preserves declarative structures, we first examine the distribution of subgoal declaration counts before and after the translation. Specifically, Isar's count is measured by the number of **have**, **hence**, **consider**, and **obtain** commands; Minilang's is by **HAVE** and **CONSIDER**. As shown by the dashed line (original Isar) and dotted line (translated MiniLang) in Figure 15, the distributions remain nearly identical, indicating that the declarative structures are largely well preserved.

Using the number of subgoal declarations as a proxy for proof complexity, we further assess how the translation success rate varies with this complexity. The result is illustrated by the solid line in Fig. 15. For short-to-moderate declarative proofs (≤ 5 subgoals), the success rate exceeds 70%. As the number of subgoal declarations increases, the success rate declines gradually; however, even for complex proofs with 15 subgoals — which account for less than 1% of all declarative proofs — the success rate remains above 50%. This demonstrates that our translator is reasonably successful in preserving declarative proof structures, given the syntactic complexity of Isar.

As a complementary measure, we assess the translation success using the number of Isar commands as an alternative complexity metric. As illustrated in Fig. 16, for moderate-sized Isar proofs (up to 15 commands), the success rate remains above 78%. Even for longer proofs with 30 commands, the success rate maintains 66%. These results further confirm that our translator is capable of handling proofs of substantial length.

At last, we evaluate the effectiveness of the refinement process (§4.2.4) in eliminating tactics. The dashed line in Figure 16 plots the ratio of atomic tactics in the translated corpus to those in the original Isar. As it reveals, the refinement process reduces tactics by more than 80% across proofs of all sizes. This confirms the efficacy of our refinement strategy for tactic elimination.

5.2 Model Training and Evaluation Setup

To minimize confounding factors in this evaluation, we carefully set up our experiments: (1) we use Supervised Fine-Tuning (SFT) — a standard and basic training approach — to train NTP models on both Isar and MiniLang; (2) we adopt whole-proof generation, where language models directly produce complete proofs without additional search frameworks or inference-time scaffolding

beyond temperature sampling; (3) we follow the prompt setup from Baldur, a prior NTP work on Isabelle to eliminate variations from prompt design. The concrete setup is detailed as follows.

5.2.1 Base Models. To assess Minilang across multiple models, we fine-tune two 7B base models: Llemma [9] and DeepSeek-Prover-Base v1.5 (DPSK-PB) [10]. Both models have been pre-trained on datasets from major proof assistants including Coq, Lean, and Isabelle.

5.2.2 Data Source. Our fine-tuning datasets are sourced from Isabelle AFP version 2025-02-12 [19] and Isabelle 2024 HOL libraries [21]. We apply several preprocessing steps to these sources: we remove unreachable code, exclude proofs of benchmark targets (such as those in PISA [20]), and filter out proofs that exceed the context window limits of our base LLMs. After preprocessing, these sources yield a total of ~332K proofs.

These preprocessed datasets are then converted into six versions, one corresponding to each experimental condition in our ablation study, as detailed in the following subsection.

5.2.3 Abolation Setup & Dataset Construction. Our work introduces two primary improvements: the redesign of the proof language and the enhancement of Sledgehammer. To comprehensively evaluate their respective impacts on NTP performance, we design an ablation study covering all combinations of two language choices (Isar vs. Minilang) and three ATP configurations, Sledgehammer* (SH*), Sledgehammer (SH), and no ATP. This constitutes $2 \times 3 = 6$ conditions, necessitating six versions of the training corpus. These conditions and their corpus are detailed as follows:

- **Isar + no ATP**: This evaluates the original Isar, using the corpus obtained in §5.2.2.
- **Isar + SH**: This evaluates *Thor-style Isar* language [11], which represents the best-known language for declarative NTP on Isabelle. Thor-style Isar minimizes tactic usage by delegating proof obligations to Sledgehammer. The corpus is constructed by exhaustively replacing tactics with Sledgehammer whenever Sledgehammer can discharge the proof obligations addressed by the original tactics.
- **Isar + SH***: Same as Isar + SH, but use Sledgehammer* instead. The corpus contains the relevant lemma hints extracted using the same heuristic described in §4.2.4.
- **Minilang + no ATP**: The intermediate Minilang code obtained directly from the translation step (§4.2.3) without applying the refinement step (§4.2.4). This intermediate representation preserves all tactics as **APPLY** commands and deviates from Minilang's design principle of purely declarative theorem proving.
- **Minilang + SH**: This evaluates a weakened version of Minilang where the ATP backend is downgraded from Sledgehammer* to the original Sledgehammer. Its corpus is constructed by removing the lemma hints from the Minilang + SH* corpus.
- **Minilang + SH***: This evaluates the full-featured MiniLang. Its corpus is constructed by translating the Isar + no ATP corpus using the pipeline established in §4.

These ablation conditions differ only in their training corpus, proof language, and ATP backend; all other experimental settings remain identical across the conditions.

5.2.4 Benchmark. We use PISA [20] as our evaluation benchmark. It originally comprises 3K goals randomly sampled from Isabelle AFP version 2022-12-06. Due to the ongoing development of Isabelle and AFP, some goals have been moved or removed from the newer versions. We manually updated the dataset to AFP version 2025-02-12, removing available goals, resulting in 2,962 goals.

5.2.5 Data Contamination. As PISA is sampled from the open-access AFP, it faces the same data contamination risks inherent to all benchmarks derived from public datasets. Llemma explicitly reports that they removed any proofs whose names appear in PISA from their pretraining data [9].

Table 2. PISA evaluation of models over MiniLang and Isar. SH = Sledgehammer, SH* = Sledgehammer*.

Base Model	Language	ATP	pass@1	pass@4	pass@8
DPSK-PB	Minilang	SH*	**69.1%**	**76.0%**	**79.2%**
DPSK-PB	Minilang	SH	59.9%	68.1%	72.4%
DPSK-PB	Thor-style Isar	SH*	63.9%	69.6%	74.3%
DPSK-PB	Thor-style Isar	SH	45.7%	54.9%	59.7%
DPSK-PB	Minilang	none	35.5%	40.6%	44.9%
DPSK-PB	Isar	none	40.2%	45.8%	50.5%
Llemma	Minilang	SH*	**68.0%**	**74.9%**	**78.9%**
Llemma	Minilang	SH	58.7%	67.9%	72.2%
Llemma	Thor-style Isar	SH*	63.3%	66.9%	72.1%
Llemma	Thor-style Isar	SH	46.1%	51.9%	57.5%
Llemma	Minilang	none	35.2%	39.8%	44.6%
Llemma	Isar	none	38.6%	43.9%	48.6%

However, DPSK-PB makes no such claim in their paper [10]; we cannot rule out the possibility of data leakage between PISA and DPSK-PB's pretraining corpus.

Nevertheless, any such contamination would more likely inflate Isar's performance relative to Minilang, since Minilang proofs do not appear in either base model's pretraining data. Thus, if Minilang still demonstrates superior performance, data contamination concerns would minimally undermine the conclusion of the paper.

5.2.6 Prompt Setup. Following Baldur's approach, we use a simple prompt setup with two parts: context and goal. Context includes declarations, lemmas, and proofs immediately preceding the goal in the same file. The goal contains the name and statement. Given the 4K token context window for both Llemma and DPSK-PB, we reserve 2K tokens each for prompt and models' response, truncating distant content when necessary.

5.2.7 Supervised Fine-Tuning (SFT). We use LLaMA-Factory [22], a widely used LLM training framework to train Llemma and DPSK-PB with supervised fine-tuning. Both models are fine-tuned for 2 epochs with a batch size of 256. The learning rate is set to 2×10^{-5}, and linearly scaled to 0 during training. The training is run on 8 Nvidia H200 GPUs, and each takes ~12 hours to finish.

5.2.8 Proof Check. We evaluate model performance using the standard pass@k metric. For each benchmark entry, we sample k proof attempts from the model and verify each sample using Isabelle's proof checker. An entry is considered proved if at least one of the k samples passes verification. The pass@k is then computed as the proportion of proved entries in the test set.

Sledgehammer and Sledgehammer* include a self-learning system that maintains local databases of premise-goal connections. Performance on a goal improves when the hammers have previously encountered similar problems. For fair comparison, we reset the local database before each model evaluation. Nonetheless, this also means our results underestimate real-world performance, where Sledgehammer* would retain contextual knowledge and perform better.

5.2.9 Infrastructure. The proof check process is powered by a socket-based parallel Read-Eval-Print-Loop infrastructure that we developed. It is implemented in the Isabelle/ML language [23] by interfacing with Isabelle's internals.

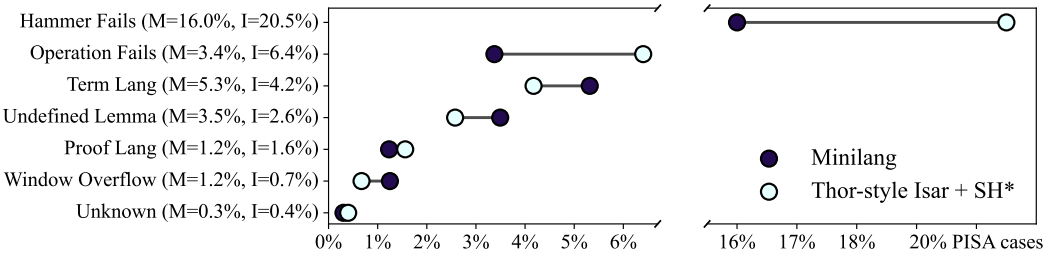


Fig. 17. Comparison of failure causes between fine-tuned models over Minilang (M) and Isar+SH* (I).

5.3 Results

Table 2 presents the evaluation results on PISA. Based on these results, we answer the three research questions posed at the beginning.

5.3.1 RQ 1. Effectiveness of Redesigning Proof Language. To answer RQ 1, we compare our redesigned language, Minilang + SH*, against Thor-style Isar + SH, the best-known language for declarative NTP on Isabelle [11]. The results demonstrate substantial improvements of ~20 percentage points across both base models and evaluation metrics.

Our ablation study further reveals the sources of this improvement. When Minilang and Isar use the same ATP backend (either SH or SH*), Minilang consistently outperforms Isar by at least 5 percentage points across both base models and all evaluation metrics. This advantage grows to over 12 percentage points under SH. Conversely, holding the language fixed and upgrading from SH to SH* yields at least 6 percentage points improvement across all models and metrics. These results demonstrate that both Minilang's language design and the ATP improvement contribute significantly to NTP performance. These results directly answer RQ 1:

RA 1: Redesigning the proof language can effectively improve declarative NTP.

When SH and SH* are disabled, proofs rely completely on tactic applications, turning the scenario into tactic-based theorem proving, as opposed to declarative proving. In this case, Minilang no longer maintains its advantage and falls behind Isar by ~5 percentage points. This is reasonable given that Minilang is specifically designed for declarative proofs.

In particular, Minilang restricts every tactic application to act only on the leading subgoal. While this brings clearer proof structure and better alignment between tactics and subgoals, it also prevents models from using terminating tactics (e.g., `fastforce`, `auto`) to conclude all subgoals at once. Instead, models must precisely understand each tactic's effect and track proof state changes — a task that is prohibitively difficult without interactive feedback from the proof assistant, even for human users. Indeed, 37.4% of proof failures result from premature termination: models incorrectly assume all subgoals are resolved when unproven subgoals remain. Thus, we attribute Minilang's performance degradation to its explicit exposure of the inherent difficulties of tactic-based proving for language models.

5.3.2 RQ 2. Sources of Performance Improvement. Holding the ATP fixed at SH*, Figure 17 compares the failure causes between Minilang and Isar across both base models in the pass@1 evaluation.

Failures are categorized as follows: (1) *Hammer Fails* indicates that SH* failed to discharge a proof obligation; (2) *Operation Fails* represents failures in executing proof operations, such as tactics and other specific mechanisms like calculational reasoning; (3) *Term Lang* denotes syntax errors in term expressions; (4) *Proof Lang* captures ill-formed proof scripts; (5) *Window Overflow* arises when

the allocated 2K context window is insufficient for the model to complete the proof; (6) *Unknown* occurs when the evaluation infrastructure fails to capture specific error information.

The Proof Lang category shows that Minilang does help models generate syntactically correct proofs, but this improvement accounts for no more than 10% of the total performance gain.

The primary improvements come from the reduced failures in proof operations and the increased hammer success rate. These gains cannot be attributed to the ATP enhancement, because both Minilang and Isar use the same SH* backend in this comparison. Since our controlled comparison isolates language choice as the only variable, we believe these gains stem from Minilang's design helping models generate more logically valid proof steps (reducing operation failures) whose resulting proof obligations are more tractable for SH* (increasing its success rate).

RA 2: The improvement brought by redesigning the proof language can extend beyond syntax errors to enhance models' capability of generating logically correct proofs

We also observe an increase in term language errors for Minilang, despite no modifications to this aspect of the language. This likely reflects the sequential nature of our error analysis, which reports only the first error encountered — as Minilang resolves certain types of errors, other unrelated issues may become more visible in the error categorization.

5.3.3 RQ 3: Training Corpora for a New Proof Language. The Minilang training corpus is obtained by translating Isabelle's existing proofs. The translation process is incomplete, losing ~15% of proofs, with higher loss rates observed for longer declarative proofs (Fig. 14). Despite this data reduction, Minilang's experimental results still demonstrate clear advantages due to language improvements. This validates the effectiveness of our translation-based approach.

RA 3: Rule-based translation is an effective approach to obtain training corpora for a redesigned proof language.

6 Related Works

To the best of our knowledge, this work represents the first attempt to improve NTP through comprehensive proof language redesign. We discuss related work across several areas.

6.1 Analysis of LLMs' challenges in learning proof languages

PALM [6] conducts a formative study analyzing the direct causes of failures in a GPT-based NTP system for Coq's proof language Gallina. One of their key findings is that LLMs often produce proofs with correct high-level structure but struggle with low-level details, justifying the declarative NTP paradigm underlying our work. Additionally, they demonstrate that proof language features — such as explicit structuring mechanisms like bullets — significantly impact LLM performance, supporting our approach of improving NTP through language redesign.

6.2 Proof Representations for Neural Theorem Proving

Many works [4, 24–27] design specialized proof representations for training language models on theorem-proving tasks.

One line of work uses representations that combine multiple types of information while retaining the original proof language syntax. For instance, Kimina [4] integrates informal proofs with formal proof sketches; TacticToe [25] combines proof states with tactic application histories; HOLStep [24] and IsarStep [27] provide datasets and benchmarks where models work with proof states alongside proof step sequences. LLMSTEP [26] uses representations combining proof states and file context.

Another line of work transforms proof texts into alternative data structures. Passport [28] represents Coq proofs as trees to capture their hierarchical structure; Paliwal et al.[29] use graph representations to model dependencies between proof steps; CoqGym[30], MLFMF [31], and HOList [32] adopt S-expression formats.

Finally, Thor [11] pioneered the use of Sledgehammer to simplify proof languages for declarative NTP. It exhaustively replaces tactics with Sledgehammer to obtain proofs closer to declarative outlines, thereby shifting the burden of tactic reasoning from language models to automated theorem provers. However, this elimination is incomplete in several aspects. First, Thor does not normalize Isar's redundant proof idioms and retains Isar's reasoning mechanisms that are unnecessary for declarative NTP — such as connectives, generic elimination, and calculational reasoning. Second, it does not leverage relevant lemma hints to improve Sledgehammer's premise selection, limiting the effectiveness of proof automation. Third, Thor does not properly handle cases where a single tactic addresses multiple subgoals. Our work extends Thor's vision of a purely declarative language for NTP by addressing these limitations through comprehensive language redesign and enhanced Sledgehammer*.

6.3 Prior Neural Theorem Provers on Isabelle

Leading NTP works on Isabelle and PISA include Thor [11], Baldur [7], and Magnushammer [33]. Thor has been discussed in §6.2. It employs Google's proprietary models and devices, achieving a success rate of 57.0% on PISA under a computational budget of at most 300 model queries. Baldur fine-tunes the LLM Minerva [7] to generate whole Isabelle/HOL proofs. It also incorporates a repair model that leverages Isabelle's error messages to fix broken proofs. Baldur reaches 65.7% with pass@64 on PISA. Magnushammer [33] adopts contrastive learning to target premise selection, the same task as Sledgehammer. Though this approach does not involve proof generation by LLMs, its combination with Thor reaches the previous state-of-the-art, a success rate of 71% on PISA.

All aforementioned works use closed-source models, preventing direct performance comparisons under identical configurations. Nonetheless, our ablation study replicates Thor's approach within our experimental framework. Under identical experimental conditions, Minilang achieves ~20 percentage points improvement over Thor-style Isar (Table 2). Furthermore, under a smaller computational budget than Magnushammer (500s timeout, 16 CPU cores, and up to 8 proof attempts per goal), we achieve a pass@8 success rate of 79.2% on PISA. Our end-to-end results represent new state-of-the-art performance on the PISA benchmark.

7 Limitations and Future Work

The main trade-off due to Minilang's restriction on language elements is that some goals may require more proof steps, although they are still provable because Minilang is no less proof-theoretically complete than Natural Deduction (Theorem 2). Though this trade-off might be unacceptable to human users who prefer efficiency, it aligns well with LLMs' computational strengths: generating lengthy proofs by repeatedly applying learned operations.

While this work implements Minilang in Isabelle/HOL, it is possible to port at least the core of Minilang to other proof assistants like Lean. This portability stems from Minilang's focus on declarative proof structures, where the notions are general across specific logics and software systems. As an instance, all Minilang commands in Table 1 have correspondences in Lean. Relying on powerful ATPs, we believe most disparities between proof assistants can be harmonized. Hopefully, Minilang can serve as a bridge, preventing the long-standing fragmentation in the field of proof assistants from spreading into the field of NTP.

8 Data Availability Statement

The REPL infrastructure, MiniLang interpreter, translator, training data, machine learning framework, Sledgehammer*, and parameters of the finetuned models will be open-sourced and are present in the submitted supplementary materials.

Our artifact is 7GB, exceeding HotCRP's upload limit. We upload it to an anonymous Google Drive instead. The link is as follows. We will attend the Artifact Evaluation.

https://drive.google.com/file/d/1CyMravL9d4-BntoRqBykfgAZBGznxiT_/view?usp=sharing

References

[1] Z. Li, J. Sun, L. Murphy, Q. Su, Z. Li, X. Zhang, K. Yang, and X. Si, "A survey on deep learning for theorem proving," in *The 1st Conference on Language Modeling*, 2024. [Online]. Available: https://openreview.net/forum?id=zlw6AHwukB

[2] Google DeepMind, "AI achieves silver-medal standard solving International Mathematical Olympiad problems," https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/, 2024.

[3] L. Chen, J. Gu, L. Huang, W. Huang, Z. Jiang, A. Jie, X. Jin, X. Jin, C. Li, K. Ma, C. Ren, J. Shen, W. Shi, T. Sun, H. Sun, J. Wang, S. Wang, Z. Wang, C. Wei, S. Wei, Y. Wu, Y. Wu, Y. Xia, H. Xin, F. Yang, H. Ying, H. Yuan, Z. Yuan, T. Zhan, C. Zhang, Y. Zhang, G. Zhang, T. Zhao, J. Zhao, Y. Zhou, and T. H. Zhu, "Seed-prover: Deep and broad reasoning for automated theorem proving," 2025. [Online]. Available: https://arxiv.org/abs/2507.23726

[4] H. Wang, M. Unsal, X. Lin, M. Baksys, J. Liu, M. D. Santos, F. Sung, M. Vinyes, Z. Ying, Z. Zhu, J. Lu, H. de Saxcé, B. Bailey, C. Song, C. Xiao, D. Zhang, E. Zhang, F. Pu, H. Zhu, J. Liu, J. Bayer, J. Michel, L. Yu, L. Dreyfus-Schmidt, L. Tunstall, L. Pagani, M. Machado, P. Bourigault, R. Wang, S. Polu, T. Barroyer, W.-D. Li, Y. Niu, Y. Fleureau, Y. Hu, Z. Yu, Z. Wang, Z. Yang, Z. Liu, and J. Li, "Kimina-prover preview: Towards large formal reasoning models with reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2504.11354

[5] T. Achim, A. Best, K. Der, M. Fédérico, S. Gukov, D. Halpern-Leister, K. Henningsgard, Y. Kudryashov, A. Meiburg, M. Michelsen, R. Patterson, E. Rodriguez, L. Scharff, V. Shanker, V. Sicca, H. Sowrirajan, A. Swope, M. Tamas, V. Tenev, J. Thomm, H. Williams, and L. Wu, "Aristotle: Imo-level automated theorem proving," 2025. [Online]. Available: https://arxiv.org/abs/2510.01346

[6] M. Lu, B. Delaware, and T. Zhang, "Proof automation with large language models," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1509–1520.

[7] E. First, M. N. Rabe, T. Ringer, and Y. Brun, "Baldur: Whole-proof generation and repair with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 1229–1241.

[8] Z. Z. Ren, Z. Shao, J. Song, H. Xin, H. Wang, W. Zhao, L. Zhang, Z. Fu, Q. Zhu, D. Yang, Z. F. Wu, Z. Gou, S. Ma, H. Tang, Y. Liu, W. Gao, D. Guo, and C. Ruan, "DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition," 2025. [Online]. Available: https://arxiv.org/abs/2504.21801

[9] Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. M. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck, "Llemma: An open language model for mathematics," in *12th International Conference on Learning Representations (ICLR)*, Vienna, Austria, May 2024.

[10] H. Xin, Z. Z. Ren, J. Song, Z. Shao, W. Zhao, H. Wang, B. Liu, L. Zhang, X. Lu, Q. Du, W. Gao, H. Zhang, Q. Zhu, D. Yang, Z. Gou, Z. F. Wu, F. Luo, and C. Ruan, "DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search," in *ICLR 2025, Singapore, April 24-28*, 2025.

[11] A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzygózdz, P. Milos, Y. Wu, and M. Jamnik, "Thor: Wielding Hammers to Integrate Language Models and Automated Theorem Provers," in *NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[12] K. Palmskog, "qarith-stern-brocot," https://github.com/rocq-community/qarith-stern-brocot/blob/9400dafdc7e35b53d23ab336e3a1b548c3b09133/theories/sqrt2.v, 2022, [Online; accessed 07-May-2025].

[13] Lean community, "Logic and Proof," https://leanprover-community.github.io/logic_and_proof/introduction.html, 2025, [Online; accessed 07-May-2025].

[14] Wikipedia, "Isabelle (proof assistant)," http://en.wikipedia.org/w/index.php?title=Isabelle%20(proof%20assistant), 2025, [Online; accessed 07-May-2025].

[15] R. Xin, C. Xi, J. Yang, F. Chen, H. Wu, X. Xiao, Y. Sun, S. Zheng, and K. Shen, "Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving," 2025. [Online]. Available: https://arxiv.org/abs/2502.03438

[16] A. Q. Jiang, S. Welleck, J. P. Zhou, T. Lacroix, J. Liu, W. Li, M. Jamnik, G. Lample, and Y. Wu, "Draft, sketch, and prove: Guiding formal theorem provers with informal proofs," in *The Eleventh International Conference on Learning*

Representations, ICLR 2023, Kigali, Rwanda, 2023.

[17] J. Meng and L. C. Paulson, "Translating higher-order clauses to first-order clauses," *Journal of Automated Reasoning*, vol. 40, no. 1, pp. 35–60, Jan 2008.

[18] D. Kühlwein, J. C. Blanchette, C. Kaliszyk, and J. Urban, "MaSh: Machine Learning for Sledgehammer," in *Interactive Theorem Proving*, S. Blazy, C. Paulin-Mohring, and D. Pichardie, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 35–50.

[19] AFP contributors, "Archive of Formal Proofs," https://www.isa-afp.org/, 2025.

[20] A. Q. Jiang, W. Li, J. M. Han, and Y. Wu, "LISA: Language models of ISAbelle proofs," in *6th Conference on Artificial Intelligence and Theorem Proving (AITP)*, 2021.

[21] Isabelle contributors, "Hol libraries," the `src/HOL` folder in the redistribution pack of Isabelle 2024, https://isabelle.in.tum.de/website-Isabelle2024, 2024, [Online; accessed 07-May-2025].

[22] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, and Z. Luo, "LlamaFactory: Unified efficient fine-tuning of 100+ language models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Y. Cao, Y. Feng, and D. Xiong, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 400–410. [Online]. Available: https://aclanthology.org/2024.acl-demos.38/

[23] M. Wenzel, *The Isabelle/Isar Implementation*, chapter 0 covers Isabelle/ML. With contributions by Stefan Berghofer, Florian Haftmann, and Larry Paulson. [Online]. Available: https://isabelle.in.tum.de/doc/implementation.pdf

[24] C. Kaliszyk, F. Chollet, and C. Szegedy, "HolStep: A Machine Learning Dataset for Higher-order Logic Theorem Proving," in *ICLR 2017, Toulon, France, April 24-26, 2017*, 2017.

[25] T. Gauthier, C. Kaliszyk, J. Urban, R. Kumar, and M. Norrish, "Tactictoe: Learning to prove with tactics," *Journal of Automated Reasoning*, vol. 65, no. 2, pp. 257–286, Feb 2021.

[26] S. Welleck and R. Saha, "Llmstep: Llm proofstep suggestions in lean," 2023. [Online]. Available: https://arxiv.org/abs/2310.18457

[27] W. Li, L. Yu, Y. Wu, and L. C. Paulson, "IsarStep: a Benchmark for High-level Mathematical Reasoning," in *ICLR 2021, Virtual Event, Austria, May 3-7*, 2021.

[28] A. Sanchez-Stern, E. First, T. Zhou, Z. Kaufman, Y. Brun, and T. Ringer, "Passport: Improving Automated Formal Verification Using Identifiers," *ACM Trans. Program. Lang. Syst.*, vol. 45, no. 2, Jun. 2023.

[29] A. Paliwal, S. Loos, M. Rabe, K. Bansal, and C. Szegedy, "Graph Representations for Higher-Order Logic and Theorem Proving," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, pp. 2967–2974, Apr. 2020.

[30] K. Yang and J. Deng, "Learning to Prove Theorems via Interacting with Proof Assistants," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97. PMLR, 09–15 Jun 2019, pp. 6984–6994.

[31] A. Bauer, M. Petkovic, and L. Todorovski, "MLFMF: Data Sets for Machine Learning for Mathematical Formalization," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023.

[32] K. Bansal, S. Loos, M. Rabe, C. Szegedy, and S. Wilcox, "HOList: An Environment for Machine Learning of Higher Order Logic Theorem Proving," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 454–463. [Online]. Available: https://proceedings.mlr.press/v97/bansal19a.html

[33] M. Mikula, S. Tworkowski, S. Antoniak, B. Piotrowski, A. Q. Jiang, J. P. Zhou, C. Szegedy, L. Kucinski, P. Milos, and Y. Wu, "Magnushammer: A Transformer-Based Approach to Premise Selection," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.