UNIFIED OPERATIONAL FORMALISM FOR LLM-BASED THEOREM-PROVING SYSTEMS

Avaljot Singh*, **Shaurya Gomber***, **Yasmin Sarita, Jose Meseguer, Gagandeep Singh**
University of Illinois Urbana Champaign
{avaljot2,sgomber2,ysarita2,meseguer,ggnds}@illinois.edu

ABSTRACT

LLM-based theorem provers differ widely in how they organise interaction between language models and interactive theorem provers, ranging from whole-proof generation to tactic-level and multi-stage pipelines. We propose a unified operational framework that makes this interaction structure explicit by modelling proof search as transitions over a joint formal and informal state and by introducing *orchestration* as a first-class abstraction that controls how calls to tools such as language models, retrieval components, and provers are scheduled and coordinated. Within this framework, existing systems such as Baldur, COPRA, DSP, POETRY, etc, can be expressed uniformly as different orchestration strategies, enabling principled comparison of interaction patterns, rapid prototyping of new strategies, and backend-agnostic evaluation and reuse across provers, libraries, and models.

1 INTRODUCTION

In recent years, a rapidly growing body of work has proposed LLM-based theorem-proving systems that combine large language models (LLMs) with interactive theorem provers (ITPs) such as Lean (Moura & Ullrich, 2021) and Coq (The Coq Development Team, 2017). Representative examples include systems such as Baldur (First et al., 2023), COPRA (Thakur et al., 2024), DSP (Jiang et al., 2023), and POETRY (Wang et al., 2024). These systems differ substantially in how they organise the interaction between an auxiliary component (typically an LLM, often combined with retrieval and other heuristics) and a formal proof engine. Some approaches like Baldur aim to synthesize an entire proof in a single step and subsequently repair it when verification fails. Others like COPRA proceed incrementally by repeatedly predicting and applying individual tactics. In contrast, some approaches use multi-stage or recursive pipelines (e.g., DSP, POETRY) that first produce informal proof sketches and only later attempt full formalization. Despite their apparent algorithmic diversity, these systems repeatedly alternate between informal reasoning and formal verification while maintaining an evolving internal state. However, this common structure is not made explicit in existing work. Instead, each system is presented as a standalone algorithm, and the interaction pattern between the LLM and the ITP is hard-coded into the system design. So, it remains difficult to pinpoint which design elements are responsible for the improved results over baselines.

We present MAESTRO (Figure 1), a unifying operational formalism for LLM-based theorem-proving systems that makes this implicit structure explicit. Our formalism models the internal algorithm state as a pair consisting of a formal component for interaction with an ITP and an informal component that stores auxiliary information produced and consumed by components such as LLMs and retrieval systems. Proof search is described as a sequence of transitions over this shared state, where each transition advances either the formal or the informal component or both. A central concept in our formulation is *orchestration*: the policy that determines which component of the state is advanced at each step and how information flows between the informal and formal parts. Under this view, systems such as Baldur, COPRA, DSP, POETRY, etc, are instances of the same operational model with different orchestrators. This formalism yields several important benefits.

Unifying view and theoretical comparability. First, it exposes a common design space for describing diverse existing theorem-proving systems that are currently presented as fundamentally

*Equal contribution.

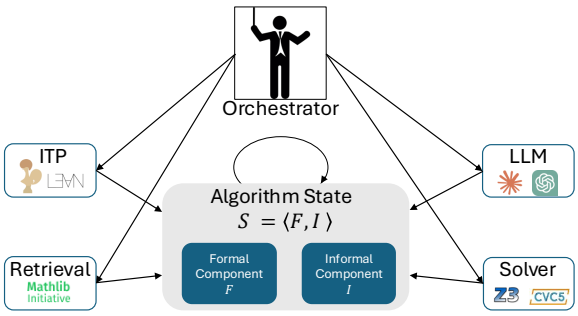

Figure 1: Conceptual overview of MAESTRO.

different algorithms. By expressing these systems in terms of a shared notion of state and an orchestrator, it is possible to compare and analyse their interaction strategies at a conceptual level, independently of implementation details, engineering choices, or presentation style.

Rapid prototyping of new strategies. It enables rapid prototyping of new proof strategies by allowing alternative interaction patterns to be explored through changes to the orchestrator alone, without modifying prover-specific interfaces, prompt pipelines, or retrieval infrastructure.

Backend-agnostic transfer of orchestration strategies. Third, our formalism separates orchestration from the ITP, libraries, and LLMs on which a system is instantiated. This is crucial in practice, because many existing approaches implicitly rely on prover-specific ecosystems, such as large curated libraries (e.g., `mathlib` in Lean) and LLMs fine-tuned on data from that environment. So, it is unclear whether their reported effectiveness reflects a generally useful interaction or merely the availability of rich supporting infrastructure. By making orchestration independent, the same orchestrator can be evaluated on different ITPs for which no large training corpora exist, like NuITP (Durán et al., 2024). This enables testing if an orchestrator generalizes across proof environments.

Towards adaptive and autonomous orchestration. Making orchestration a first-class component highlights a key limitation of most existing systems: the interaction between the tools is fixed in advance and applied uniformly to all problems. In practice, different stages of a proof may benefit from different modes of interaction, for example, whole-proof synthesis in easy phases and tactic-level exploration in harder ones. Our formulation naturally motivates a mixed or completely *autonomous* orchestrator that dynamically selects interaction modes based on the evolving algorithm state. Further, by implementing existing strategies in a unified framework, it would enable a systematic collection of diverse execution traces describing how different orchestrators guide proof search. These can serve as training data for an autonomous orchestrator. While this is beyond the scope of this work, the proposed formalism provides a foundation for such autonomous approaches.

In summary, this paper makes the following contributions:

- We introduce MAESTRO, a unifying operational model for theorem-proving systems that separates the algorithm state from the orchestrator that governs the interactions of tools.
- We show that a range of recent systems, including Baldur, COPRA, DSP, and POETRY, can be expressed within this model by specifying only their orchestration strategies.

2 DIFFERENT PROVING STYLES

Without providing an exhaustive survey, we review some representative theorem-proving systems that have emerged in recent work. While not mutually exclusive, these exhibit substantially different interaction patterns between different tools such as LLMs, retrieval systems, and ITPs.

Whole Proof Generation. A first class of systems follows a coarse-grained LLM-interaction based on whole-proof generation. Given a theorem statement, these systems prompt the model to produce a complete proof in a single step, which is then submitted to an ITP for verification. If the prover rejects the script, some systems (e.g., Baldur First et al. (2023)) invoke the language model again to repair or regenerate the proof and repeat this process for a bounded number of iterations. Several theorem-proving systems fall into this category, differing mainly in the choice of theorem prover, language model, prompting strategy, and auxiliary tooling (First et al. (2023); Zhang et al. (2025); Dong & Ma (2025); Lin et al. (2025c); Wang et al. (2025); Xin et al. (2024)).

Next Tactic Prediction. A second class of systems adopts a fine-grained, tactic-level interaction. Starting from an initial theorem statement, these systems repeatedly query a model for the next tactic to apply. Each proposed tactic is immediately executed in the ITP, yielding a new proof state. This process continues until all goals are discharged. When progress stalls, some systems may backtrack (e.g., COPRA Thakur et al. (2024)) to earlier proof states and explore alternative tactic sequences. Representative examples of this class include systems such as COPRA and related tactic-prediction approaches (Thakur et al. (2024); Yang et al. (2023); Lin et al. (2025b); Gauthier et al. (2021)).

Hybrid Approaches. Besides the two extremes, a third class of systems typically decomposes proof construction into ad hoc steps. An example of this is DSP (Jiang et al., 2023), which uses a multi-stage proof synthesis pipeline. It first uses an LLM to generate an informal proof draft, which is subsequently refined into a higher-level proof sketch, again using the LLM. Then, a fully formal proof is verified by the ITP. Related ideas appear in Wang et al. (2024); Lin et al. (2025a); Dong et al. (2024).

Despite their apparent fundamental differences, these systems share a common operational structure: each maintains an evolving algorithm state and advances it by selectively invoking external tools. The primary distinction among existing approaches therefore lies not in the representation of this state, but in the policy governing how the state transitions using different tool interactions. For instance, whole-proof generation, tactic-level, and hybrid systems differ mainly in how they schedule and interleave these interactions. We refer to this as the system's *orchestration*. Viewed this way, these systems are different orchestrator instantiations in a unified operational framework.

3 MAESTRO: AN OPERATIONAL FRAMEWORK

We now introduce MAESTRO, a unified operational framework for theorem-proving systems (Figure 1). The framework comprises of 3 conceptual components: (i) the algorithm state, (ii) the tools that alter the algorithm state, and finally, (iii) the orchestrator that acts as a control plane to orchestrate different tools to alter the algorithm state iteratively until the given theorem is proved.

3.1 ORCHESTRATING THE ALGORITHM STATE

The algorithm state is defined as $S = \langle F, I \rangle$, where F is the *formal component* and I is the *informal component*. At any step, the formal component F represents the current formal proof object and its associated proof obligations. Concretely, it contains a proof tree or proof term under construction, together with a set of open goals and auxiliary metadata maintained in the formal syntax of the ITP. Some nodes of this proof structure may be *open* (unproven), while others are already *closed* (proved) by previously applied proof steps. The informal component I contains all auxiliary information that is not part of the formal proof object but is used to guide the proof. This includes, for example, natural-language explanations, informal proof sketches, partial scripts, natural-language conjectures, retrieved lemmas from external libraries, prompt templates, interaction histories with one or more language models, and any additional search or bookkeeping data maintained by the system.

Under this formulation, theorem proving is a search over a state space whose nodes are algorithm states of the form $\langle F, I \rangle$. The search starts from the state $\langle T, \phi \rangle$, where the formal component contains the original theorem T as an open goal, and the informal component is empty (ϕ). The goal is any state $\langle P(T), \_ \rangle$, whose formal component contains a complete proof $P(T)$, while the informal component is unconstrained. An orchestrator controls the search process by mapping the current state to a choice of tool invocation. Intuitively, the orchestrator decides which tool to invoke at a given point, such as a particular LLM, a retrieval engine, or an ITP, and how the resulting information should be incorporated into the evolving state. So, it governs when to attempt formal proof steps, when to gather auxiliary information, when to revise previous decisions, when to backtrack, etc.

3.2 STATE TRANSITIONS USING TOOL CALLS

In MAESTRO, the orchestrator has a set of *tools* at its disposal to modify the algorithm state iteratively. Typical tools include theorem provers, LLMs, retrieval engines over libraries of proven theorems (e.g., `mathlib`), and human-in-the-loop (Song et al., 2025). Each tool call induces a transition on the algorithm states: $\langle F, I \rangle \xrightarrow{t} \langle F', I' \rangle$, progressing the search space traversal. The

Baldur	$\langle T, \phi\rangle \xrightarrow{\text{LLM}} \langle T, C_1\rangle \xrightarrow{\text{ITP}} \langle F_1, \phi\rangle \xrightarrow{\text{LLM}} \langle F_1, C_2\rangle \xrightarrow{\text{ITP}} \langle F_2, \phi\rangle$
COPRA	$\langle T, \phi\rangle \xrightarrow{\text{Retrieve}} \langle T, I_1\rangle \xrightarrow{\text{LLM}} \langle T, t_1\rangle \xrightarrow{\text{ITP}} \langle F_1, \phi\rangle \longrightarrow \cdots$
DSP	$\langle T, \phi\rangle \xrightarrow{\text{LLM}} \langle T, D\rangle \xrightarrow{\text{LLM}} \langle T, S\rangle \xrightarrow{\text{ITP}} \langle F_1, \phi\rangle$
Poetry	$\langle T, \phi\rangle \xrightarrow{\text{LLM}} \langle T, S_1\rangle \xrightarrow{\text{ITP}} \langle F_1, \phi\rangle \xrightarrow{\text{Recursive}} \cdots \xrightarrow{\text{ITP}} \langle F_i, \phi\rangle \cdots$

Table 1: Instantiations of MAESTRO

final goal of the orchestrator is to find a sequence of tool calls t_i that leads it to the goal state, i.e., $\langle T, \phi\rangle \xrightarrow{t_1} \langle F_1, I_1\rangle \xrightarrow{t_2} \cdots \langle P(T), \_\rangle$. Some tools affect the informal component (e.g., generating a proof sketch, retrieving related lemmas, refining a prompt), while others affect the formal component (e.g., applying a tactic, assuming a lemma, etc). However, a tool may, in general, update both components. In particular, backtracking, state repair, or state rewriting operations are naturally modeled as tools that replace the current formal or informal component by a previous version.

4 INSTANTIATING THE FRAMEWORK

We view most existing systems as instances of MAESTRO. Although they appear diverse, MAESTRO reveals that they primarily differ in orchestration policies. This perspective clarifies variations in search strategies and tool use, enabling simpler comparison and a unified framework. We illustrate representative theorem-proving systems by describing their orchestrators in Table 1.

Baldur. It begins by prompting an LLM to generate a complete proof candidate, C_1, within the informal state. It then submits this candidate to the ITP for formal verification, leading to the formal state F_1. If verification fails, the orchestration prompts another informal transition to repair or regenerate the candidate, C_2, before submitting it again to the ITP, resulting in the formal state F_2.

COPRA. Starting from the current state $\langle F_i, I_i\rangle$, the orchestration retrieves related lemmas to update the informal state, I_{i+1}. Next, an LLM is prompted to propose a tactic t_i, which is applied by the ITP, leading to a new state $\langle F_{i+1}, I_{i+1}\rangle$. This cycle continues until all goals are discharged.

DSP. DSP follows a 3-step orchestration. First, an LLM generates a proof draft, D. Next, the LLM refines this draft into a formal proof sketch, S. Finally, the orchestration invokes the ITP to verify the sketch, transitioning to the formal state F_1.

POETRY. POETRY employs a recursive sketch-refinement approach. At each level, the orchestrator advances the informal component to produce a proof sketch, S_{i+1}, outlining the high-level structure of the formal state F_i. Intermediate subproofs are abstracted with placeholders, deferring their verification. The ITP is then invoked to verify the sketch, changing the new formal component to F_{i+1}. If the verification fails, the system backtracks and repeats the process.

5 CONCLUSION AND FUTURE WORK

We presented MAESTRO, a unifying operational formalism for LLM-based theorem-proving systems that separates the evolving algorithmic state from the set of available tools and the orchestrator that governs their interaction. By making orchestrator a first-class component, our framework provides a common basis for describing and comparing existing systems, clarifies the role played by interaction strategies independently of specific models or prover infrastructures, and enables systematic exploration of alternative proof-search behaviours. As future work, we plan to realise this formalism as a practical framework in which existing systems and representative orchestration strategies can be instantiated and re-implemented over a shared set of state representations and tool interfaces. We will design and evaluate new orchestrators, including mixed strategies that combine coarse-grained synthesis and fine-grained tactic exploration. A key goal is to study transfer across backends by instantiating the same orchestration policies with less commonly used and structurally different theorem provers, such as NuITP. Finally, we will collect execution traces from multiple orchestrators and tool configurations and use them to train and fine-tune an autonomous orchestrator that learns to select and adapt interaction strategies based on the evolving proof state. We believe that this framework will enable more robust, modular, and general theorem-proving systems.

REFERENCES

Kefan Dong and Tengyu Ma. STP: Self-play LLM theorem provers with iterative conjecturing and proving. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=zWArMedNuW.

Kefan Dong, Arvind V. Mahankali, and Tengyu Ma. Formal theorem proving by rewarding LLMs to decompose proofs hierarchically. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=D83tiHiNfF.

Francisco Durán, Santiago Escobar, Jose Meseguer, and Julia Sapiña. Nuitp: An inductive theorem prover for equational program verification. In *Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming*, PPDP '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400709692. doi: 10.1145/3678232.3678236. URL https://doi.org/10.1145/3678232.3678236.

Emily First, Markus N. Rabe, Talia Ringer, and Yuriy 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*, ESEC/FSE 2023, pp. 1229–1241, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703270. doi: 10.1145/3611643.3616243. URL https://doi.org/10.1145/3611643.3616243.

Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Tactictoe: Learning to prove with tactics. *J. Autom. Reason.*, 65(2):257–286, February 2021. ISSN 0168-7433. doi: 10.1007/s10817-020-09580-x. URL https://doi.org/10.1007/s10817-020-09580-x.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=SMa9EAovKMC.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Reviving DSP for advanced theorem proving in the era of reasoning models. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=SOWZ59UyNc.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. Lean-STar: Learning to interleave thinking and proving. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=SOWZ59UyNc.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia LI, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source automated theorem proving. In *Second Conference on Language Modeling*, 2025c. URL https://openreview.net/forum?id=x2y9i2HDjD.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In André Platzer and Geoff Sutcliffe (eds.), *Automated Deduction – CADE 28*, pp. 625–635, Cham, 2021. Springer International Publishing. ISBN 978-3-030-79876-5.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean copilot: Large language models as copilots for theorem proving in lean, 2025. URL https://arxiv.org/abs/2404.12534.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. An in-context learning agent for formal theorem-proving. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=V7HRrxXUhN.

The Coq Development Team. Coq, 2017. URL https://coq.inria.fr. v.8.7.

Haiming Wang, Huajian Xin, Zhengying Liu, Wenda Li, Yinya Huang, Jianqiao Lu, Zhicheng Yang, Jing Tang, Jian Yin, Zhenguo Li, and Xiaodan Liang. Proving theorems recursively. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning, 2025. URL https://arxiv.org/abs/2504.11354.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data, 2024. URL https://arxiv.org/abs/2405.14333.

Kaiyu Yang, Aidan M Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=g7OX2sOJtn.

Jingyuan Zhang, Qi Wang, Xingguang Ji, Yahui Liu, Yang Yue, Fuzheng Zhang, Di Zhang, Guorui Zhou, and Kun Gai. Leanabell-prover: Posttraining scaling in formal reasoning, 2025. URL https://arxiv.org/abs/2504.06122.