Peano Player: Interactive Theorem Proving as a Constrained MDP

Matthew Retchin [1] **Kianté Brantley** [1] **Nada Amin** [1]

Abstract

Reinforcement learning with verifiable rewards is frequently used for neural theorem proving and typically trains on a single binary signal indicating whether a complete generated proof verifies. However, proof assistants are interactive systems: after each tactic, they return an updated proof state and report whether the tactic failed. Prior methods that use tactic-failure signals treat failure as a minor penalty and allow generation to continue after a tactic fails. We introduce Peano Player, a tactic-level theorem prover formulated as a constrained Markov decision process, where the agent observes the verifier-produced proof state, receives reward for closing proofs, and incurs a terminal cost of equal magnitude for failing a tactic. We evaluate in a controlled equational-theory environment over the Peano numbers that generates unlimited provable theorems at controlled difficulty. An ablation over proof-state observation and tactic-failure cost shows that each improves solve rate independently, and their combination more than quadruples overall solve rate relative to the strongest baseline, with a tenfold difference at the greatest difficulty level. These results suggest that theorem-proving agents would benefit from learning from errors during interaction—not only from whether a completed proof verifies—because any tactic-level proof assistant exposes the same per-step signals.

1. Introduction

Theorem provers provide two signals at each step that whole-proof RLVR methods discard. After each tactic, the prover produces an updated proof state, which is the new goal that the next tactic must address. When a tactic is inapplicable, the prover reports failure.

[1]Harvard University. Correspondence to: Matthew Retchin <mretchin@g.harvard.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

Much recent RL work on theorem proving uses whole-proof generation trained via RLVR with a single binary verification reward, turning theorem proving into a bandit problem with outcome supervision (Xin et al., 2025; Ren et al., 2025; Wang et al., 2025). In these works, the agent does not observe verifier-produced intermediate proof states during generation and receives no online signal about individual tactic failures.

Kim & Yun (2026) augment bandit-style RLVR with tactic-level process supervision. They execute the generated proof against the verifier, identify the first failing tactic, and propagate error scores backward. This is an improvement over binary outcome supervision, but the model still generates a full proof before receiving any verifier feedback. Their error-propagation scheme penalizes tactics generated *after* the first error, but in an interactive setting the verifier terminates on failure, so there are no post-failure tactics to penalize. Their default penalty magnitudes also penalize tactic failure substantially less than proof closure, which motivates evaluating a stronger failure penalty in our setting.

We present Peano Player, a constrained Markov decision process (CMDP) agent for tactic-level theorem proving. We frame tactic validity as a constraint instead of an objective because applying tactics correctly is a necessary, though not sufficient, condition for proof closure.

A failure to close a proof can be diagnosed in two ways. First, tactic failure immediately implies proof failure. Second, a proof attempt can exhaust the episode horizon without any individual tactic failing; every applied tactic can be valid, yet the agent may never close a proof before exhausting the step limit. The CMDP distinguishes these outcomes through step-level cost that penalizes tactic failure immediately and sparse binary reward that tracks overall proof closure. This reward/cost separation is well established in safe RL for robotics (Altman, 1999; Tessler et al., 2019). For example, avoiding collisions is naturally treated as a constraint on motion planning rather than as part of the objective of reaching the goal.

We train and evaluate Peano Player in an environment with four deterministic rewrite tactics. The environment's rewrite system is confluent and terminating, enabling theorem generation by sampling normal forms and applying expansions that preserve reducibility to the same normal form. This

avoids proof search during theorem generation: every sampled theorem is provable by construction, and a reference proof can be obtained by normalization. Our environment isolates whether agents learn theorem proving more effectively when they receive the proof states and tactic-failure signals that the verifier already provides.

We make three contributions. First, we formulate tactic-level theorem proving as a CMDP that separates overall proof closure from tactic failure. Second, we construct a controlled equational-theoretic environment with deterministic rewrite transitions and proofs by construction, enabling isolated evaluation of verifier interaction signals. Third, we evaluate our method against baselines that isolate the effects of proof-state observation from tactic-failure feedback and show a compounding improvement when combined, with their combination more than quadrupling solve rate over the strongest baseline.

2. Related Work

Proof-step vs. whole-proof generation. Two paradigms dominate for generating proofs with language models: proof-step and whole-proof generation. Proof-step methods condition on an intermediate proof state and generate the next proof step or tactic (Polu & Sutskever, 2020; Lample et al., 2022; Wu et al., 2021; Gloeckle et al., 2024; Lin et al., 2025a). Whole-proof methods generate a complete proof candidate and then check it with the verifier (First et al., 2023; Lin et al., 2025b; Ren et al., 2025; Xin et al., 2024). Recent RL-based theorem-proving systems train from verifier outcomes, proof-assistant feedback, or related verifiable signals (Xin et al., 2025; Shao et al., 2024). Kim & Yun (2026) augment whole-proof generation with per-tactic process supervision via first-error propagation. They apply the method to STP-Lean (Dong & Ma, 2025) and DeepSeek-Prover-V1.5 (Xin et al., 2025), evaluating on Lean benchmarks such as MiniF2F (Zheng et al., 2022) and ProofNet (Azerbayev et al., 2023). Our experiment instead uses a procedural theorem generator to isolate observation structure from tactic-failure feedback under controlled dynamics.

Environmental feedback in theorem proving. Several proof-step systems combine learned tactic generation with explicit proof search (Lample et al., 2022; Wu et al., 2021; Xin et al., 2025; Wu et al., 2025). We do not formulate theorem proving as search over partial proof trees because search alone does not address the problem of proposing valid tactics in a discrete and effectively unbounded action space (Yang et al., 2025). This paper instead isolates how an agent should consume the verifier's per-step feedback when generating a single tactic.

An et al. (2024) train on failed proof paths via supervised fine-tuning. Ji et al. (2025) integrate multi-turn verifier feedback into RL training for Lean, incorporating error messages into the reasoning chain. Our approach separates tactic failure as a CMDP cost rather than embedding it in the language model's context, and we consider context-based feedback to be complementary.

Constrained RL. CMDPs and constrained policy optimization have a long history in robotics and optimal control (Altman, 1999; Tessler et al., 2019). Our contribution is not a new constrained-RL algorithm. Rather, we observe that tactic-level theorem proving exposes the reward/cost interface required by CMDPs. To our knowledge, prior theorem-proving RL work has not formulated tactic-level proving as a CMDP with proof closure as reward and tactic failure as a separate cost.

Controlled axiomatic domains. Poesia & Goodman (2023) introduce Peano, a theorem-proving environment with finite action spaces over algebraic axioms, and use RL to learn from scratch on Khan Academy algebra problems. Poesia et al. (2024) extend this to Minimo, an agent that jointly learns conjecturing and proving in propositional logic, arithmetic, and group theory. Our domain follows the same principle of using a small formal theory with controlled theorem generation so that RL design-space axes can be studied in isolation. In our environment, confluence and termination in the rewriting system enable unlimited theorem generation at controlled difficulty with proofs guaranteed by construction, without requiring an expensive proof search to build the dataset.

3. Background

3.1. Constrained MDPs

A constrained Markov decision process is a tuple $(S, A, P, R, C, \gamma, \alpha)$ (Altman, 1999), where S is the state space, A is the action space, $P(\cdot \mid s, a)$ is the transition kernel, $R : S \times A \to \mathbb{R}$ is the reward function, $C : S \times A \to \mathbb{R}$ is the cost function, $\gamma \in [0, 1)$ is the return discount factor, and α is the constraint threshold. The agent maximizes the expected discounted return $J_R^\pi = \mathbb{E}_\pi[\sum_t \gamma^t R(s_t, a_t)]$, subject to the constraint $J_C^\pi \leq \alpha$, where $J_C^\pi = \mathbb{E}_\pi[\sum_t \gamma^t C(s_t, a_t)]$ is the expected discounted cost. We employ a Lagrangian relaxation to convert this to an unconstrained objective with penalized reward $\hat{r}(s, a) = R(s, a) - \lambda\, C(s, a)$, where $\lambda \geq 0$ is the Lagrange multiplier. RCPO adapts λ on a slow timescale (Tessler et al., 2019), but we fix $\lambda = 1$ and do not tune it in this work.

3.2. Sparse Proof Reward and Dense Tactic Feedback

Current RLVR methods typically train from a sparse binary reward indicating whether a generated proof verifies. Kim & Yun (2026) introduce dense tactic-level feedback by executing the generated proof, identifying the first failing tactic, and assigning tactic-level scores by first-error propagation. We explore a similar tactic-level penalty, but we instantiate it in a multi-step environment. Proof closure contributes reward, while tactic failure contributes cost. The dense signal in our MDP is the verifier's immediate failure signal observed during interaction.

3.3. Bandit and Multi-Step MDPs

In a multi-step MDP, the agent selects one action, observes the resulting state, and selects the next action conditioned on that state. In a bandit formulation, the agent generates an entire action sequence from the initial state and receives feedback only after the completed sequence is executed. Many whole-proof theorem-proving RL methods treat tactic generation as a bandit problem, where the agent produces all tactics at once and the verifier scores the result. An alternative is to treat it as a multi-step MDP over verifier states. This paper tests whether the multi-step formulation improves learning.

3.4. GRPO

Group Relative Policy Optimization is a bandit-style policy-gradient method (Shao et al., 2024). Given a theorem, the policy samples G complete proofs. Each proof is verified and receives a binary reward. Advantages are normalized across the group, and this advantage is applied uniformly to all tokens of that proof regardless of which tactic each token belongs to. No value function is learned, and credit assignment relies entirely on group normalization.

Kim & Yun (2026) extend GRPO with post-hoc per-tactic feedback. The generated proof is executed tactic-by-tactic against the verifier. If the proof verifies, each tactic receives 1. If the proof fails, tactics before the first error receive d_1, and the first failing tactic and all subsequent tactics receive d_2. This adds within-proof credit assignment. However, the method remains bandit-style: the agent generates the full proof before receiving any tactic-level feedback.

4. Method

4.1. CMDP Instantiation

We map tactic-level theorem proving onto the CMDP framework (Section 3.1). Any tactic-level proof assistant exposes the relevant signals—per-step states and failure reports. Whole-proof methods discard state observations during generation, and outcome-only methods also discard tactic-level failure feedback. The CMDP formulation preserves both. Section 4.2 describes our specific environment.

Component	Instantiation
State S	Current proof state (goal string lhs = rhs), produced by the verifier after each tactic. In this environment, the goal string is a Markov state: given the current equation and a rewrite tactic, the next equation, proof closure, or tactic failure is determined without reference to the initial theorem or prior actions.
Actions A	One tactic per step (four rewrite rules; see Section 4.2).
Transition P	Deterministic. A valid tactic rewrites the goal, and an invalid tactic terminates the episode.
Reward R	1 on proof closure (both sides syntactically identical), 0 otherwise.
Cost C	1 on tactic failure, 0 otherwise.
Discount γ	0.99; earlier closure receives higher return.

Table 1. CMDP instantiation for tactic-level theorem proving.

An episode ends in one of three ways: (1) the proof closes ($R = 1$); (2) a tactic fails ($C = 1$) and the episode terminates immediately; (3) the agent exhausts the step limit without closing the proof or failing a tactic ($R = 0$, $C = 0$).

4.2. Environment

The environment operates over terms built from `zero`, `succ`, `pred`, and `add`. The action space consists of four oriented rewrite rules (Table 2).

Rule	Rewrite
`pred_succ`	$\texttt{pred}(\texttt{succ}(n)) \to n$
`pred_zero`	$\texttt{pred}(\texttt{zero}) \to \texttt{zero}$
`add_zero`	$\texttt{add}(n, \texttt{zero}) \to n$
`add_succ`	$\texttt{add}(n, \texttt{succ}(m)) \to \texttt{succ}(\texttt{add}(n, m))$

Table 2. The four rewrite rules constituting the agent's action space.

Each rule replaces a matching subexpression on one side of the goal. The rules are terminating and non-overlapping, yielding a confluent rewrite system in which every term has a unique normal form.

Theorem generation with controlled difficulty. We call d the theorem difficulty. To generate a difficulty-d theorem, we (1) sample a normal-form expression e; (2) apply d expansion operations, each replacing a subterm with a more complex expression that reduces back to the original under

the rewrite system; and (3) optionally expand both sides independently from the same normal form. The first four expansion operations are direct inverses of the four rewrite rules. Two additional compound operations introduce subterms requiring two or three rewrite steps to normalize, so proof length can exceed d. The resulting equation is provable by construction because greedy normalization of both sides reaches the same normal form. The full procedure appears in Algorithm 1 (Appendix A).

Median proof length increases monotonically with d (Table 3). Proof by construction avoids search during theorem generation, producing unlimited provable theorems at controlled difficulty without depending on a prover.

d	Median steps	Mean steps	Max steps
1	1	1.4	3
2	2	2.2	5
3	3	3.2	6
4	4	4.5	10
5	5	5.5	9
6	6	6.8	11
7	7	7.9	14
8	8	9.2	17

Table 3. Proof length (rewrite steps) as a function of difficulty parameter d, measured over 500 generated theorems per level. Median proof length increases monotonically with d, and the gap between median and max arises from compound expansions that introduce multiple redexes per step.

Example episode. Let the goal be $\texttt{add}(\texttt{zero}, \texttt{succ}(\texttt{zero})) = \texttt{succ}(\texttt{zero})$. If an agent applies `add_succ`, the tactic is valid ($C = 0$) and the proof is not yet closed ($R = 0$). The environment therefore produces $\texttt{succ}(\texttt{add}(\texttt{zero}, \texttt{zero})) = \texttt{succ}(\texttt{zero})$. If the agent then applies `add_zero`, which is again valid ($C = 0$), the environment will produce $\texttt{succ}(\texttt{zero}) = \texttt{succ}(\texttt{zero})$. The sides match, so the proof closes ($R = 1$). Over two steps, no tactic fails (total cost 0), and the discounted return is $\gamma^1 = 0.99$.

The environment is a Python simulation of Lean 4's `rw` tactic, using leftmost-outermost matching and replacement. To confirm that this emulation is faithful for our rule set, we type-check the theory definition and 200 generated theorem–proof pairs with Lean's compiler.

4.3. Optimization

We optimize a fixed-Lagrangian relaxation with PPO, using the scalarized per-step signal $\hat{r}(s, a) = R(s, a) - \lambda C(s, a)$. The reward-only MDP ablation sets $\lambda = 0$, so learning is driven only by proof-closure reward. The full CMDP condition sets $\lambda = 1$, adding an immediate penalty for tactic failure. With fixed λ, the algorithm is equivalent to standard PPO applied to the per-step reward $R(s, a) - \lambda C(s, a)$. The CMDP notation makes the intended constraint explicit and supports the ablation ($\lambda = 0$ versus $\lambda = 1$).

5. Experimental Setup

Our experiments span two axes: observation structure (bandit vs. multi-step) and feedback structure (sparse proof-closure reward vs. proof-closure reward and step-level tactic-failure cost). Our method is the full CMDP agent: PPO with multi-step observation and tactic-failure cost ($\lambda = 1$). We compare against two baselines and one ablation. The baselines are standard GRPO (Shao et al., 2024) and GRPO with Kim & Yun's first-error feedback, both of which generate whole proofs before receiving feedback. The ablation removes tactic-failure cost ($\lambda = 0$) while retaining multi-step observation. This 2×2 structure lets each axis be evaluated while holding the other fixed.

For the Kim & Yun baseline, we set the tactic-level penalties to $d_1 = 0$, $d_2 = -1$ to match our PPO cost structure ($c = 0$ for valid tactics, $c = 1$ for failures). In preliminary experiments, Kim & Yun's published values ($d_1 = -0.05$, $d_2 = -0.1$) provided too weak a signal in our environment. This aligns with our observation that prior work underweights tactic failure relative to proof closure. Setting $d_2 = -1$ ensures that the failure-aware conditions use comparable penalty magnitudes rather than confounding the comparison.

Across experiments, we hold constant the architecture, optimizer, training budget, theorem difficulty distribution, and maximum episode length. Full hyperparameters appear in Appendix B.

5.1. Implementation

The two observational regimes (bandit vs. multi-step MDPs) require different architectures. The PPO agent observes a single goal string at each step and selects one of four tactics, so it uses an encoder-style Transformer with last-token pooling feeding into a policy head and a value head. The bandit agent must generate a variable-length tactic sequence from the initial goal alone, so it uses a causally masked autoregressive Transformer conditioned on the concatenation of the goal and all previously generated tactics. Both architectures use byte-level tokenization and have approximately 4.08M parameters each. Full hyperparameters appear in Appendix B.

6. Results

We evaluate all four conditions on held-out theorems, sampling the theorem difficulty d uniformly from $\{1, \ldots, 8\}$

Condition	Solve (%) ↑	Failure (%) ↓
Peano Player	**92.9 ± 0.7**	**1.46 ± 0.15**
Multi-Step (reward only)	69.4 ± 1.3	7.18 ± 0.35
Bandit + cost	24.2 ± 1.7	21.70 ± 0.69
Bandit	21.0 ± 0.8	23.68 ± 0.55

Table 4. Held-out evaluation at convergence, averaged over 5 seeds. Solve rate is aggregated across difficulties. Failure rate is the fraction of executed tactic steps that fail.

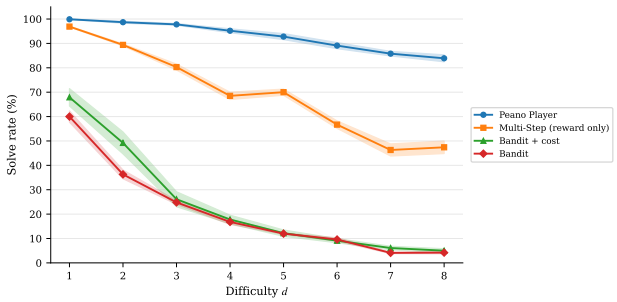

Figure 1. Held-out solve rate by theorem difficulty. Curves show mean ± standard error over 5 seeds, with 200 generated theorems per difficulty and seed. Peano Player combines multi-step proof-state observation with tactic-failure cost.

as defined in Section 4.2. For evaluation, all methods use the same procedurally generated theorem stream at each difficulty, with 200 theorems per difficulty and seed. Table 4 reports aggregate metrics. Figure 1 shows solve rate as a function of difficulty.

The full CMDP agent substantially outperforms every baseline. Overall, it solves more than four times as many held-out theorems as the strongest bandit baseline, and at the greatest difficulty level ($d = 8$), the gap is roughly tenfold. We decompose this improvement along the two experimental axes: multi-step proof-state observation and tactic-failure cost.

Multi-step outperforms bandit. Comparing rows while holding feedback structure fixed, multi-step observation roughly triples solve rate over the corresponding bandit condition (Table 4). The gap widens with difficulty: at $d = 8$, the multi-step conditions solve 44–85% of theorems depending on feedback structure, while both bandit conditions fall below 10% (Figure 1).

Reward and cost together outperform reward alone. Adding tactic-failure cost to proof-closure reward improves performance under both observation structures, but the gain is much larger under multi-step observation (Table 4). The two bandit conditions converge above $d = 5$: first-error feedback helps at low difficulty but provides diminishing returns as proofs grow longer (Figure 1). Under multi-step

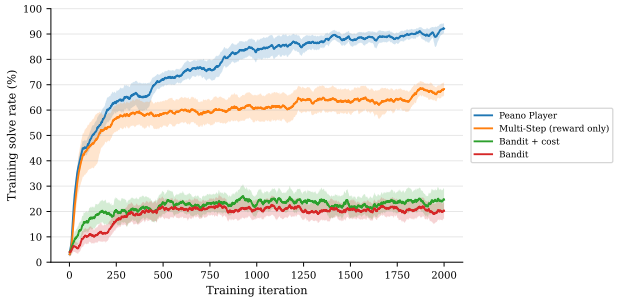

Figure 2. On-policy training solve rate over training. Each point is computed from the batch of theorems sampled for that training iteration. Curves show mean ± standard error over 5 seeds, with exponential moving-average smoothing using $\beta = 0.95$ for visualization.

observation, the cost signal maintains a roughly 40-point advantage over sparse reward even at $d = 8$.

Full CMDP performs best. Multi-step observation without cost reaches 70%, while reward and cost without multi-step observation reaches 27%. Combining both gives the full CMDP agent, which exceeds 90% overall solve rate, more than quadrupling the strongest bandit baseline. At $d = 8$, the full CMDP condition is roughly ten times higher than either bandit condition. The cost signal identifies which tactic failed, but exploiting this information requires observing proof states across the trajectory.

Convergence rate. The convergence curves of the solve rate (Figure 2) show that the multi-step conditions converge faster than their bandit counterparts under both feedback structures.

Tactic failure rate. At convergence, the cost-aware conditions produce fewer tactic failures than their sparse-reward counterparts (Table 4), with far fewer failures for Peano Player relative to the ablation and baselines (Figure 3). The cost signal drives the agent to select applicable tactics, producing more informative trajectories: episodes terminate in proof closure or horizon exhaustion rather than premature failure from an invalid tactic.

7. Discussion

In our environment, the goal string after each tactic satisfies the Markov property: the transition, reward, and cost depend only on the current equation and the tactic applied, not on the initial theorem or action history. A bandit, by contrast, conditions on the initial theorem and a growing action history. The bandit has enough information in principle to reconstruct the current proof state, but it must learn this reconstruction from delayed, higher-variance feedback. The multi-step MDP receives the verifier's transition result

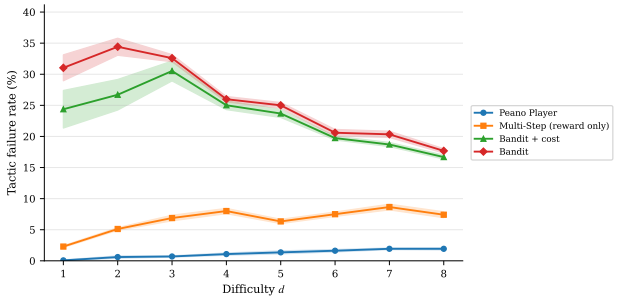

Figure 3. Tactic failure rate by theorem difficulty. Curves show mean ± standard error over 5 seeds. Failure rate is measured as the fraction of executed tactic steps that fail; generated but unexecuted post-failure tactics in the bandit setting are not counted.

explicitly. The improvement from multi-step observation should therefore be interpreted as a learning-effectiveness gap rather than a difference in what is recoverable in principle.

The human-readable goal string is not in general a sufficient state representation for theorem proving. Proof assistants maintain a full tactic state—goals, local context, metavariables, universe constraints, and elaboration state—whose transition under a tactic is exactly what the verifier computes. Treating theorem proving as a multi-step MDP means exposing this verifier state, or an adequate representation of it, to the policy instead of requiring the model to infer it from a whole-proof prefix.

When controlling for the same multi-step observation process, the gap between our reward-only ($\lambda = 0$) ablation and the CMDP ($\lambda = 1$) shows that tactic-failure cost contributes learning signal that terminal proof reward cannot provide alone. A failed tactic and a valid-but-incomplete trajectory both receive zero terminal reward, but they are different errors. The former is an invalid action, while the latter is insufficient progress. The CMDP separates these cases through cost and reward.

In the four-tactic Peano environment, invalid actions can be filtered before execution. This filtering does not scale to tactic generation in proof assistants such as Lean, where validity is semantic and state-dependent: tactics may include terms, theorem names, local hypotheses, generated expressions, user-defined tactics, and metaprogrammed procedures. The decoder can enforce syntax, but whether a syntactically valid tactic succeeds in the current proof state is determined only by executing it. Peano Player models tactic failure as verifier feedback rather than assuming a valid-action mask so that the formulation transfers without modification.

Our environment's equational theory is confluent and deterministic by design. This guarantees proofs by construction to enable tractable theorem generation without search, but it also limits the scope of the theorems that agents must prove. In particular, agents never need to decide whether a theorem is provable; every theorem presented to the agent is provable by construction. Because tactic execution is deterministic and the action space is small, the domain reduces search-level strategy relative to full Lean theorem proving. It does not remove tactic-level interaction: agents must still choose applicable tactics from verifier states, avoid failures, and close proofs within the episode horizon. Our setting therefore isolates proof-state observation and tactic-failure feedback from the broader difficulties of large-scale proof search.

7.1. Limitations and Future Work

Our environment's domain is synthetic, with a four-tactic action space and reduced search-level strategy compared to full-scale proof assistants. The simplicity of the rewrite theory is what makes unlimited theorem generation with known proofs possible. All experiments use a single difficulty distribution (uniform $d \in \{1, \ldots, 8\}$). Performance under different distributions or at higher difficulty levels could show different trends.

We fix $\lambda = 1$ and do not perform constrained policy optimization. Constrained policy optimization methods such as RCPO (Tessler et al., 2019) would instead adapt the multiplier during training to find the reward–cost balance induced by the constrained objective rather than fixing this balance manually. The direction of our current results suggests $\lambda > 1$ may be beneficial.

The CMDP formulation transfers to any tactic-level proof assistant that reports proof state and tactic failure after each step. The nontrivial gap is not the feedback interface but the initial-goal distribution for training and the action space used by the policy. In this work's environment, the confluent rewrite system makes theorem generation cheap; for general-purpose theorem proving, provability of arbitrary propositions is not cheaply decidable. Theorem statements from Lean's Mathlib library would provide known-provable initial goals, and truncated Mathlib proofs would provide curricular intermediate states. Refutation tasks could be constructed from known theorems: if Mathlib proves P, one could task the agent with refuting $\neg P$.

Beyond the goal distribution, scaling to Lean would require replacing finite atomic tactic selection with tactics that take generated arguments. In particular, an action would no longer be a single class label selecting one of four rewrite tactics, but would instead be a tactic string specifying a tactic together with its arguments. Despite this change in action representation, the CMDP cost could still be assigned at the tactic level. If Lean were to execute the generated tactic successfully, the agent would receive the next tactic state, but if parsing, elaboration, type checking, or tactic application

were to fail, the generated tactic would receive tactic-failure cost. For an autoregressive policy, the log-probability of the generated tactic would be the sum of the log-probabilities of its tokens, conditioned on the proof state and preceding tactic tokens. Therefore, a tactic-level cost could be applied to the full generated tactic without identifying which token caused the failure. Fine-grained credit assignment to the tactic name, theorem arguments, local hypotheses, or generated terms would require additional feedback from error spans.

These approaches would retain the same per-step CMDP formulation while replacing the synthetic goal distribution with goals drawn from widely used standard libraries.

8. Conclusion

Proof assistants already report the result of each tactic: an updated proof state when the tactic succeeds, and a failure signal when it does not. The updated proof state supplies the next decision point, making tactic-level proving naturally interactive rather than a one-shot proof-generation problem. Much recent RL work on theorem proving discards this interaction by training from whole-proof verification alone; prior methods that use tactic-failure information still treat it as a weak post-hoc signal. In a controlled setting, this study shows that both parts of the tactic-level interface matter. Proof-state observation and tactic-failure cost each improve solve rate independently, and their combination more than quadruples solve rate over the strongest baseline. The feedback interface that makes this possible—per-step proof states and tactic-failure reports—is native to any tactic-level proof assistant. Thus, extending to full-scale proof assistants will require multi-token tactics that take arguments and an alternative source of training goals, while the per-step feedback protocol and CMDP formulation will remain unchanged.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning for theorem proving. More capable theorem-proving systems could eventually support improved formal verification workflows, including verification of safety-critical software and hardware, but this paper does not evaluate such applications directly.

Code Availability

Code is available at https://github.com/mhr/peano-player.

Acknowledgments

Matthew Retchin and Nada Amin were partially supported by an ARPA-H award (Federal Award number: 1AYSAX0000025-01; Sub-Award number: 000542772-SC003) and a Coefficient Giving grant. Kianté Brantley acknowledges that this work has been made possible in part by a gift from the Chan Zuckerberg Initiative Foundation to establish the Kempner Institute for the Study of Natural and Artificial Intelligence.

References

Altman, E. *Constrained Markov Decision Processes*. Chapman & Hall/CRC, 1999.

An, C., Chen, Z., Ye, Q., First, E., Peng, L., Zhang, J., Wang, Z., Lerner, S., and Shang, J. Learn from failure: Fine-tuning LLMs with trial-and-error data for intuitionistic propositional logic proving. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 776–790. Association for Computational Linguistics, 2024.

Azerbayev, Z., Piotrowski, B., Schoelkopf, H., Ayers, E. W., Radev, D., and Avigad, J. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. *arXiv preprint arXiv:2302.12433*, 2023. URL https://arxiv.org/abs/2302.12433.

Dong, K. and Ma, T. 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.

First, E., Rabe, M. N., Ringer, T., and Brun, Y. 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.

Gloeckle, F., Limperg, J., Synnaeve, G., and Hayat, A. ABEL: Sample efficient online reinforcement learning for neural theorem proving. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*, 2024. URL https://openreview.net/forum?id=kk3mSjVCUO.

Ji, X., Liu, Y., Wang, Q., Zhang, J., Yue, Y., Shi, R., Sun, C., Zhang, F., Zhou, G., and Gai, K. Leanabell-Prover-V2: Verifier-integrated reasoning for formal theorem proving via reinforcement learning. *arXiv preprint arXiv:2507.08649*, 2025.

Kim, M. and Yun, S.-Y. Process-verified reinforcement learning for theorem proving via Lean. In *International Conference on Learning Representations (ICLR)*, 2026.

Lample, G., Lachaux, M.-A., Lavril, T., Martinet, X., Hayat, A., Ebner, G., Rodriguez, A., and Lacroix, T. HyperTree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 26337–26349, 2022.

Lin, H., Sun, Z., Welleck, S., and Yang, Y. Lean-STaR: Learning to interleave thinking and proving, 2025a. URL https://arxiv.org/abs/2407.10040.

Lin, Y., Tang, S., Lyu, B., Wu, J., Lin, H., Yang, K., Li, J., Xia, M., Chen, D., Arora, S., and Jin, C. Goedel-Prover: A frontier model for open-source automated theorem proving. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=x2y9i2HDjD.

Poesia, G. and Goodman, N. D. Peano: Learning formal mathematical reasoning. *Philosophical Transactions of the Royal Society A*, 381(2251):20220044, 2023.

Poesia, G., Broman, D., Haber, N., and Goodman, N. D. Learning formal mathematics from intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Polu, S. and Sutskever, I. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Ren, Z. Z., Shao, Z., Song, J., Xin, H., Wang, H., Zhao, W., Zhang, L., Fu, Z., Zhu, Q., Yang, D., Wu, Z. F., Gou, Z., Ma, S., Tang, H., Liu, Y., Gao, W., Guo, D., and Ruan, C. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Tessler, C., Mankowitz, D. J., and Mannor, S. Reward constrained policy optimization. In *International Conference on Learning Representations (ICLR)*, 2019.

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

Wu, M., Norrish, M., Walder, C., and Dezfouli, A. TacticZero: Learning to prove theorems from scratch with deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

Wu, Z., Huang, S., Zhou, Z., Ying, H., Yuan, Z., Zhang, W., Lin, D., and Chen, K. InternLM2.5-StepProver: Advancing automated theorem proving via critic-guided search, 2025. URL https://arxiv.org/abs/2410.15700.

Xin, H., Guo, D., Shao, Z., Ren, Z., Zhu, Q., Liu, B., Ruan, C., Li, W., and Liang, X. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data, 2024. URL https://arxiv.org/abs/2405.14333.

Xin, H., Ren, Z. Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., Gao, W., Zhu, Q., Yang, D., Gou, Z., Wu, Z. F., Luo, F., and Ruan, C. DeepSeek-Prover-V1.5: Harnessing proof assistant feedback for reinforcement learning and Monte-Carlo tree search. In *International Conference on Learning Representations (ICLR)*, 2025.

Yang, K., Poesia, G., He, J., Li, W., Lauter, K. E., Chaudhuri, S., and Song, D. Position: Formal mathematical reasoning—a new frontier in AI. In *Forty-second International Conference on Machine Learning Position Paper Track*, 2025. URL https://openreview.net/forum?id=HuvAM5x2xG.

Zheng, K., Han, J. M., and Polu, S. MiniF2F: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9ZPegFuFTFv.

A. Theorem Generation Algorithm

Algorithm 1 gives the theorem generation procedure. The generator first samples a normal-form expression e. It then applies expansion operations to obtain one or two more complex expressions that reduce back to the same normal form under the rewrite system. The resulting equation is provable by construction, because normalizing both sides yields the same expression.

The nominal difficulty d controls the number of expansion steps applied to the left-hand side. With probability $p_{\text{both}} = 0.35$, the right-hand side is also expanded from the same normal form using fewer expansion steps. The expansion procedure uses a size threshold $S_{\max} = 60$ to prevent very large terms. These constants are generation hyperparameters chosen to increase dataset variety and were not tuned.

After generating a theorem, we extract a concrete rewrite trace by repeatedly applying applicable rewrite rules until both sides are syntactically equal. This trace is used for Lean validation and dataset statistics, and theorem validity follows from the shared-normal-form construction.

Algorithm 1 Theorem generation by expansion from normal forms

Require: Difficulty d
1: Sample a normal-form expression e
2: Apply d expansion operations to e to obtain lhs
3: Set rhs $\leftarrow e$
4: With probability p_{both}, apply $\lfloor d/2 \rfloor$ expansion operations to e to obtain rhs
5: Normalize both sides to extract a rewrite trace
6: **return** theorem lhs $=$ rhs and proof trace

Table 5 lists the expansion cases used by the generator. Each case replaces a subterm with a more complex expression that reduces back to the original subterm under the corresponding rewrite tactics. The first four rows are direct inverses of the four rewrite rules: each introduces exactly one redex that is normalized by one rewrite step. The last two rows are the two variants of the compound operation ADDSUCCCOMPOUND. These are not direct inverses of any single rewrite rule; they introduce subterms requiring two or three tactic steps to normalize. Therefore, proof length can exceed the nominal difficulty d.

Expansion case	Replacement	Normalizing tactic sequence
PREDSUCC	$s \mapsto \texttt{pred}(\texttt{succ}(s))$	`rw [pred_succ]`
PREDZERO	$\texttt{zero} \mapsto \texttt{pred}(\texttt{zero})$	`rw [pred_zero]`
ADDZERO	$s \mapsto \texttt{add}(s, \texttt{zero})$	`rw [add_zero]`
ADDSUCC	$\texttt{succ}(\texttt{add}(n, m)) \mapsto \texttt{add}(n, \texttt{succ}(m))$	`rw [add_succ]`
ADDSUCCCOMPOUND-1	$\texttt{succ}(s) \mapsto \texttt{add}(s, \texttt{succ}(\texttt{zero}))$	`rw [add_succ]; rw [add_zero]`
ADDSUCCCOMPOUND-2	$\texttt{succ}(s) \mapsto \texttt{add}(s, \texttt{succ}(\texttt{succ}(\texttt{zero})))$	`rw [add_succ]; rw [add_succ]; rw [add_zero]`

Table 5. Expansion cases used to generate reducible expressions from normal forms. The first four are direct inverses of the rewrite rules; each introduces one redex normalized by one step. The two compound cases are not direct inverses of any single rule and require multiple rewrite steps to normalize.

In the implementation, expansion positions are sampled by shuffling all subterm positions and selecting the first position with an applicable expansion. Applicable expansions are sampled with fixed weights. We discard degenerate samples where expansion leaves the two sides syntactically identical, as well as samples whose extracted proof exceeds the episode limit. Exact weights appear in the source code.

B. Additional Experimental Details

B.1. Architecture

Both the multi-step and bandit agents use byte-level tokenization: each character of the goal string (or tactic history, for the bandit) is mapped to its byte value and embedded into a learned d_{model}-dimensional vector.

The multi-step agent uses an encoder-style Transformer (5 layers, 4 heads, $d_{\text{model}} = 256$, feedforward dimension 1024) that encodes the current goal string and applies last-token pooling to produce a single representation. This representation feeds into a linear policy head with 4 outputs (one per tactic) and a linear value head with 1 output.

The bandit agent uses a decoder-style Transformer with the same width, depth, and head count and a causal attention mask, differing only in a longer maximum sequence length (it conditions on the goal bytes plus the generated tactics) and the absence of a value head. It autoregressively generates tactic tokens conditioned on the concatenation of the goal bytes, a separator token, and all previously generated tactic tokens.

The multi-step agent has 4,076,549 parameters and the bandit agent 4,082,436; the bandit is marginally larger because its longer positional table outweighs the omitted value head.

B.2. Hyperparameters

PPO (multi-step conditions). The discount factor is $\gamma = 0.99$. GAE uses $\lambda_{\text{GAE}} = 0.95$. The clip ratio is $\epsilon = 0.2$, with 4 update epochs per iteration and a KL target of 0.02. Each iteration collects 64 episodes. The maximum episode length is 20 steps, which exceeds the longest observed proof in the training distribution (19 steps at $d = 8$). Extending to higher difficulty would require scaling the horizon with d. The CMDP Lagrange multiplier is $\lambda = 1$ for the cost condition and $\lambda = 0$ for the sparse reward condition. The optimizer is Adam with learning rate 3×10^{-4} and no weight decay.

GRPO (bandit conditions). The group size is $G = 4$ rollouts per theorem, with 16 theorems per iteration for a total of 64 rollouts, matching the PPO episode budget. The clip ratio is $\epsilon = 0.2$, taken against the rollout policy, with an entropy bonus of coefficient 0.01, 4 update epochs per iteration, and early stopping when the approximate KL to the rollout policy exceeds 0.02, as in the PPO conditions. No frozen reference policy or separate KL penalty term is used. The cost condition uses tactic-level penalties $d_1 = 0$ and $d_2 = -1$. The optimizer is Adam with learning rate 3×10^{-4} and no weight decay.

Common settings. The training budget is 2000 iterations for all conditions. Theorem difficulty is sampled uniformly from $\{1, 2, \ldots, 8\}$. Each condition is trained with 5 random seeds, for 20 runs total.

B.3. Normal-Form Sampling

The number of variables per theorem is sampled from $\{0, 1, 2, 3\}$ with arbitrary weights biased toward 1–2 variables. Exact values appear in the source code.

B.4. Lean Validation

The generated Lean 4 file defines an inductive type, `PNat`, with `pred` and `add` declared `opaque`, so Lean's kernel cannot reduce them, and the only available rewrite lemmas are the four axioms. Type-checking the file with Lean's compiler validates both the theory definition and 200 generated theorem–proof pairs, confirming that the Python simulation faithfully reproduces Lean's `rw` semantics for this rule set.