Self-Supervised Theorem Discovery in a Formal Axiomatic System

Kazuki Ota [1 2] **Takayuki Osa** [2] **Tatsuya Harada** [1 2]

Abstract

Recent artificial intelligence (AI) systems have shown remarkable progress in mathematical reasoning. Many existing approaches, including large language models (LLMs), draw on human prior knowledge in the form of mathematical text, code, or theorem libraries. Although these approaches are highly effective in practice, it remains an open question whether an agent can autonomously discover useful theorems without such human priors. We study this question in a formal axiomatic system by developing an agent that starts from axioms and inference rules alone and gradually grows a library of useful theorems. Concretely, we propose a self-supervised theorem-discovery algorithm that alternates between proof search and useful-theorem extraction, building a theorem library whose entries are reused as lemmas for subsequent proof search. Experiments show that the agent discovers tens of thousands of theorems and finds proofs for human-written benchmark problems, suggesting that its discoveries include theorems meaningful from a human mathematical perspective. Furthermore, the discovered theorems improve LLM proof performance when provided as prompt lemmas, indicating that they can serve as external knowledge for LLM reasoning. Our results provide evidence that useful theorems can emerge from proof search without relying on human-provided theorem libraries. More broadly, they suggest a path toward self-evolving AI systems for mathematics whose discoveries remain formally verifiable.

1. Introduction

Artificial intelligence (AI) systems are rapidly improving their mathematical reasoning abilities. Large language models (LLMs) such as GPT and Gemini have shown steady gains on mathematical benchmarks (OpenAI, 2024b; 2025; Gemini Team, 2025). A recent Gemini model was also reported to achieve gold-medal standard performance at the International Mathematical Olympiad 2025 (Luong & Lockhart, 2025), highlighting advances in LLM-based mathematical reasoning.

Although LLMs can perform mathematical reasoning in natural language, their outputs can be difficult to verify because they may contain plausible but incorrect statements caused by hallucination (Farquhar et al., 2024; Kalai et al., 2025). This has motivated work on theorem proving with formal languages and proof assistants as a path toward rigorous mathematical reasoning. Recent work includes autoformalization and proof sketching methods that connect natural-language mathematics to formal proofs (Wu et al., 2022; Jiang et al., 2023), methods for generating proofs in proof assistants such as Lean (Han et al., 2022; Polu et al., 2023; Yang et al., 2023; Xin et al., 2024; Zimmer et al., 2025; Hubert et al., 2026), and methods that combine language-model generation with symbolic reasoning or learned libraries (Trinh et al., 2024; Wang et al., 2024; Dong & Ma, 2025). This line of work highlights the value of combining formally verifiable reasoning substrates with the search capabilities of machine learning models.

At the same time, many current methods start from mathematical knowledge constructed by humans. For example, mathematical language models may use natural-language mathematical knowledge through pretraining on mathematical text and code (Azerbayev et al., 2024), and methods that connect natural language to formal proofs often start from human-written problem statements or informal proofs (Wu et al., 2022; Jiang et al., 2023). In addition, methods for proof generation and retrieval-augmented theorem proving often use existing formal libraries and the theorems and proofs contained in them (Han et al., 2022; Polu et al., 2023; Yang et al., 2023). Even methods based on large-scale synthetic data often construct training data from natural-language mathematical problems (Xin et al., 2024). Such knowledge is highly useful in practice, but it makes it difficult to isolate the question of how much mathematical structure an agent can discover from formal rules alone.

A related line reduces reliance on human-provided proofs or

[1]The University of Tokyo, Japan [2]RIKEN AIP, Japan. Correspondence to: Kazuki Ota <ota@mi.t.u-tokyo.ac.jp>.

The 3^{rd} *AI for Math Workshop* at the 43^{rd} International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.

existing data by learning from axioms and formal rules (Wu et al., 2021; Laurent & Platzer, 2022; Poesia et al., 2024). These studies show that agents can self-improve in axiomatized domains and discover theorems or proofs. Kasriel et al. (2025) is especially close, as it discovers theorems from axioms, reuses them for further search, and evaluates external usefulness by asking an LLM judge whether they appear useful. However, it remains open whether such theorems can improve an external reasoning agent's proof performance when supplied as lemmas. We address this question and show that the theorems discovered by our approach improve both the agent's own proof search and LLM proof search.

In this study, we ask whether an agent can autonomously discover useful theorems without relying on human prior knowledge. To isolate this question, we require the agent to start only from the primitive rules of a formal axiomatic system, without using an existing theorem library, a proof corpus, natural-language priors, or an externally provided training problem set. Human-written problems are used only for evaluation, not for training the agent or constructing the theorem library.

We instantiate this formal-system setting in a Hilbert axiom system for propositional logic. This setting is deliberately minimal: it has only a few primitive rules, but proof search is still nontrivial and every discovered theorem remains formally checkable. Specifically, we formulate Hilbert-system proof search as a stack-machine decision process, where the state is the current proof stack and the actions push Hilbert axioms onto the stack or apply Modus Ponens. Because formulas obtained during search can be regarded as theorems, proof search itself generates theorem candidates.

On top of this stack-machine formulation, we propose a self-supervised theorem-discovery algorithm that grows a theorem library from the agent's own proof search. The agent reuses theorems reached during search as future proof goals and learns a goal-conditioned policy from the action sequences that reach them. The supervision comes from the agent's own successful proof prefixes: whenever search reaches a theorem, the corresponding action sequence becomes a training signal for proving that theorem again. It then extracts useful theorems from the discovered set and adds them as theorem actions that can be used as lemmas in later generations of proof search. In this way, the agent learns not only to discover theorems, but also to use them to prove new theorems.

Experiments show that the proposed method discovers theorems that are useful both for the agent's own proof search and for proof search by external LLMs. Specifically, the agent discovers tens of thousands of theorems and finds proofs for human-written benchmark problems. Moreover, the extracted theorems improve LLM proof performance when provided as prompt lemmas. These results show that useful theorems can emerge from proof search based only on axioms and inference rules, and that they can serve not only as internal lemmas for the agent but also as knowledge that assists external reasoning systems.

Our contributions can be summarized as follows:

1. We formulate propositional theorem proving as a stack-machine decision process based directly on the axioms and the inference rule of a Hilbert axiom system.
2. We propose a self-supervised theorem-discovery algorithm that grows a theorem library from the primitive rules of a Hilbert axiom system and then reuses extracted theorems as lemmas for further proof search.
3. Our experiments show that our agent discovers tens of thousands of theorems and finds proofs for human-written benchmark problems.
4. Finally, we show that the extracted theorems also improve LLM proof performance when provided as prompt lemmas, serving as external knowledge for LLM reasoning.

2. Preliminaries

This section defines the logical setting used throughout the paper. It provides the basis for the stack-machine decision process introduced in Section 3.

Notation for Propositional Formulas. To keep syntax simple, this paper builds formulas only from implication $\to$ and falsity $\bot$. Standard presentations often use primitive connectives such as negation $\neg$, conjunction $\wedge$, disjunction $\vee$, implication $\to$, and biconditional $\leftrightarrow$. For expressive power in classical propositional logic, these connectives need not all be primitive, since implication and falsity form a basis for the usual connectives (Post, 1921). Concretely, define $\neg A := A \to \bot$, $A \vee B := (A \to \bot) \to B$, and $A \wedge B := (A \to (B \to \bot)) \to \bot$. Define $A \leftrightarrow B$ as $(A \to B) \wedge (B \to A)$, expanded using conjunction. Thus, unless otherwise stated, every formula uses only $\to$ and $\bot$, and other connectives are syntactic sugar expanded by the translations above.

Rules of the Hilbert System. A Hilbert system is an axiomatic proof system in which theorems are derived from axioms using inference rules (Hilbert & Ackermann, 1950; Mendelson, 2015). The system used in this paper, shown in Figure 1, consists of three axioms, `Ax1`, `Ax2`, `Ax3`, and one inference rule, Modus Ponens. The symbols A, B, C in the axioms stand for arbitrary formulas. A proof repeatedly introduces axioms and applies Modus Ponens, which derives Y from previously obtained formulas X and $X \to Y$. When the final formula matches the target, it has been de-

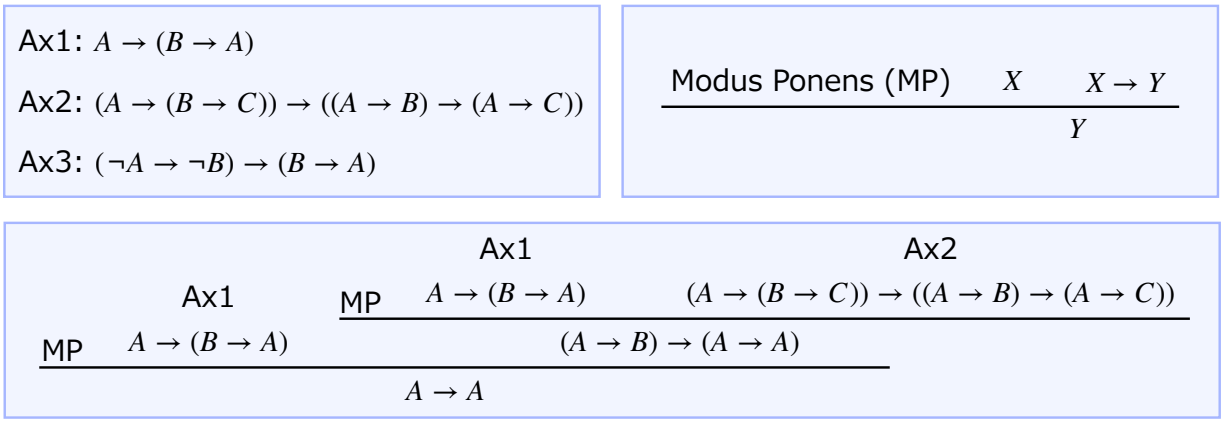

Figure 1. Hilbert system and proof tree for $A \to A$. Top: the three axioms and Modus Ponens. The axioms respectively mean weakening, distribution of implication, and contraposition. Bottom: a proof tree deriving $A \to A$ from two `Ax1` instances, one `Ax2` instance, and two Modus Ponens steps.

rived as a theorem. The bottom panel of Figure 1 shows a proof tree for the theorem $A \to A$. It uses two instances of `Ax1`, one instance of `Ax2`, and two applications of Modus Ponens. In this way, a Hilbert system proves theorems by combining axioms and inference rules.

Soundness and Completeness. This Hilbert system is known to be sound and complete for classical propositional logic (Church, 1996). Soundness means that every formula derived in this system is valid under the standard semantics of classical propositional logic. Completeness gives the converse guarantee, namely every formula valid in classical propositional logic can be derived in this Hilbert system. These properties justify viewing the Hilbert system as one of the standard axiomatic systems for classical propositional logic.

Difficulty of Finding Proofs. While soundness and completeness guarantee correctness and existence of proofs, they do not imply that proofs are easy to find. Indeed, deciding propositional validity is a coNP-complete problem (Cook, 1971). Moreover, from the perspective of proof complexity, valid formulas need not have short proofs in a given proof system, and proof length can vary substantially across proof systems (Cook & Reckhow, 1979). Therefore, efficiently finding proofs in the Hilbert system is a nontrivial problem.

3. Stack-Machine Formulation of Theorem Proving

We formulate proof construction in the Hilbert system as an action sequence executed by a stack machine. A Hilbert proof is usually represented as a proof tree built from axioms and Modus Ponens. For a learning algorithm, however, it is more natural to represent proof construction as a sequential decision process. We therefore flatten proof trees into sequences of stack operations.

Formally, for a target formula g, we define the stack-machine decision process as a deterministic, goal-conditioned decision process given by the tuple $(\mathcal{S}, \mathcal{A}, s_0, P)$, where $\mathcal{S}$ is the set of finite stacks of formulas, $\mathcal{A} = \{\texttt{Ax1}, \texttt{Ax2}, \texttt{Ax3}, \texttt{MP}\}$ is the action space, and the initial state s_0 is the empty stack. The transition rule P is deterministic. Each axiom action pushes the corresponding Hilbert axiom schema onto the stack. For the `MP` action, we use Hindley–Milner unification over formula metavariables (Hindley, 1969; Milner, 1978; Damas & Milner, 1982). If the top formula can be unified with an implication $X \to Y$ and the formula immediately below it can be unified with X, then the `MP` action is legal and, when applied, pops both formulas and pushes the resulting consequent Y. The process succeeds for goal g when the stack consists of the single formula g. We do not specify a scalar reward function in this formulation, because the learning signals in our method are constructed from self-discovered theorems rather than externally defined rewards.

Figure 2 illustrates this view by showing the proof tree for $A \to A$ as a stack-machine execution. In this example, two `Ax1` actions, one `Ax2` action, and two Modus Ponens actions produce the theorem $A \to A$ on the stack. Following this stack-machine decision process, Hilbert-style theorem proving can be treated as a sequential decision problem in which the agent chooses which axiom to introduce and when to apply Modus Ponens.

4. Self-Supervised Theorem Discovery

In this section, we describe how the proposed algorithm operates on the stack-machine decision process formulated in Section 3 and how it reuses discovered theorems in subsequent proof search. The central idea is to construct learning

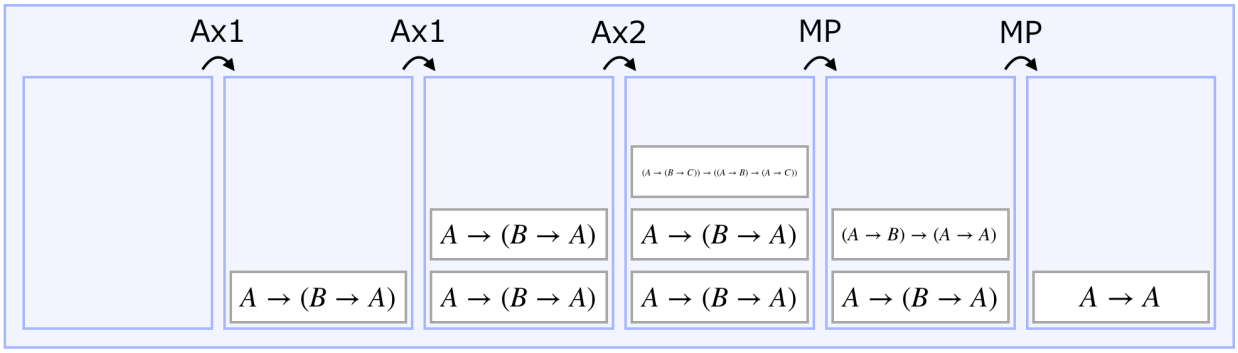

Figure 2. Stack-machine execution for the proof tree of $A \to A$. The proof tree in Figure 1 is converted into a linear action sequence: axiom instances at the leaves are executed as push actions `Ax1`, `Ax2`, and `Ax3`, and internal Modus Ponens nodes are executed as `MP` actions. Each `MP` action pops an implication and its antecedent from the stack and pushes the corresponding consequent. This example shows that the five-action sequence `Ax1`, `Ax1`, `Ax2`, `MP`, `MP` is a proof of $A \to A$.

signals from the agent's own search trajectories, rather than relying on external proof data or human-provided lemmas. Specifically, the policy is learned from self-discovered theorems through goal-conditioned supervised learning. In addition, the algorithm extracts theorems that are difficult to reprove and highly general, and adds them as new actions so that they can be reused as lemmas in subsequent search. We describe these components in the following subsections.

4.1. Learning a Goal-Conditioned Policy from Self-Discovered Theorems

The proposed method learns a goal-conditioned policy $\pi_\theta(A|S, g)$, where S is the current proof stack, g is the target theorem, and A is a stack-machine action. Here, A either introduces `Ax1`, `Ax2`, or `Ax3`, or applies `MP`. The policy therefore predicts which action should be taken next to prove the goal formula g from the current stack S.

During search, any state whose stack contains a single formula yields a reached theorem. If the state S_t contains only one formula f_t at time t, we call S_t a single-formula stack and treat f_t as a theorem reached in that episode, regardless of whether it matches the original goal formula g. Thus, even a failed attempt can provide a learning signal when a single-formula stack appears.

Figure 3 illustrates this construction. The marked states S_1, S_3, and S_7 contain f_1, f_3, and f_7, and the corresponding action prefixes are proofs of those formulas. These reached theorems are then used in two ways, as future goals and as supervised training data.

We maintain a goal buffer $\mathcal{G}$ containing theorems discovered during search. Training begins by bootstrapping $\mathcal{G}$ with random rollouts. Whenever search produces new reached theorems, they are added to $\mathcal{G}$, and later iterations sample a goal formula g uniformly from $\mathcal{G}$ before rolling out the current policy $\pi_\theta(\cdot|S, g)$.

Algorithm 1 Policy Learning in a Fixed Action Space

Input: action space $\mathcal{A}$.
Initialize a new policy $\pi_\theta(A|S, g)$ for $\mathcal{A}$.
Bootstrap the goal buffer $\mathcal{G}$ with random rollouts.
for each iteration **do**
  Initialize current training set $\mathcal{D}_{\text{train}} \leftarrow \emptyset$.
  for each episode **do**
    Sample a goal formula g uniformly from $\mathcal{G}$.
    Roll out $\pi_\theta(\cdot|S, g)$ for T steps.
    for each reached theorem f_t at time t **do**
      Add f_t to $\mathcal{G}$.
      Add proof-prefix examples for f_t to $\mathcal{D}_{\text{train}}$.
    end for
  end for
  Update π_θ by cross-entropy on $\mathcal{D}_{\text{train}}$.
end for
Return: goal buffer $\mathcal{G}$.

Reached theorems also define supervised targets for the current policy update. When f_t is reached at time t, the prefix up to t is treated as a proof of f_t, and for each $0 \le t' < t$ we add $(S_{t'}, A_{t'}, f_t)$ to $\mathcal{D}_{\text{train}}$. The policy $\pi_\theta(A|S, g)$ is then updated by cross-entropy on $\mathcal{D}_{\text{train}}$, increasing the likelihood of $\pi_\theta(A_{t'}|S_{t'}, f_t)$.

Algorithm 1 summarizes this procedure for a fixed action space. It makes explicit that each reached theorem is inserted into two objects with different roles. The training set $\mathcal{D}_{\text{train}}$ stores proof-labeled examples for the current policy update and is reset at each iteration. In contrast, the goal buffer $\mathcal{G}$ is maintained across iterations, so the agent gradually expands the set of goals available for later search. This separation keeps policy updates local to the current iteration while preserving discovered theorems as persistent future goals.

Our method shares with Hindsight Experience Replay

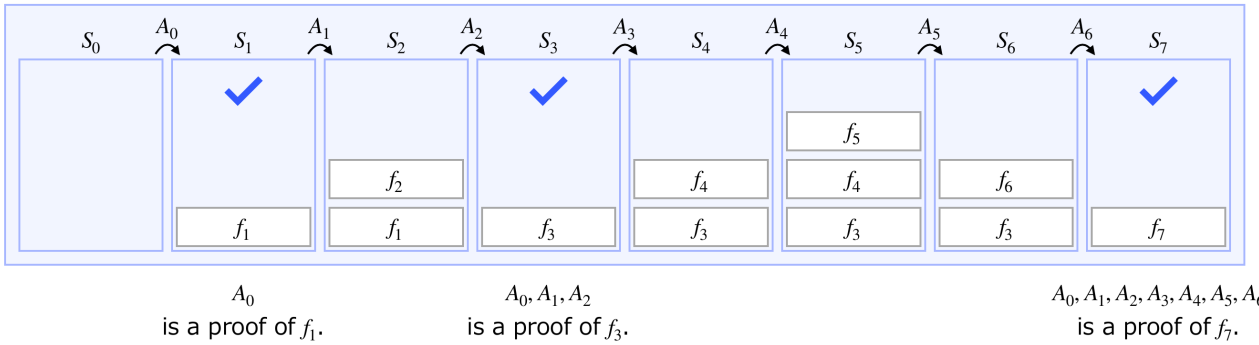

Figure 3. An example proof-search episode used to generate goals and training data. Marked states are single-formula stacks, and the corresponding formulas f_1, f_3, and f_7 are added to the goal buffer $\mathcal{G}$. For each reached theorem, we use all preceding state-action prefixes as training examples: $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \left(\cup_{t\in\{1,3,7\}} \{(S_{t'}, A_{t'}, f_t) \mid 0 \le t' < t\}\right)$.

(HER) and goal-conditioned supervised learning (GCSL) the idea of reusing what was actually reached in past trajectories as new goals (Andrychowicz et al., 2017; Ghosh et al., 2021). The difference is that the relabeled goals are not environment states, but proved theorems obtained as single-formula stacks in the Hilbert system. Thus, formulas encountered during exploration can be reused as valid theorem-proving targets in subsequent search.

4.2. Extracting Useful Theorems

We extract useful theorems from the discovered theorem set using two criteria, generality and reprovability. The goal buffer $\mathcal{G}$ constructed in the previous section serves as the candidate set. Generality removes overly specialized theorems, while reprovability prioritizes theorems that have already been discovered but cannot be reliably reproved by the current policy. Adding such theorems as lemmas can expand the set of theorems reached in subsequent proof search.

The first criterion uses the generality order induced by substituting formulas for variables to remove overly specialized theorems. For example, the theorem $A \to A$ is more general than the theorem $(A \to B) \to (A \to B)$, because substituting the formula $A \to B$ for A in the former yields the latter, whereas no variable substitution in the latter yields the former. We keep only the theorems that are not specializations of other candidates. Formally, we define a partial order so that more general theorems are smaller, and retain the minimal elements under this order. This filtering removes redundant specialized theorems while retaining more general candidates.

The second criterion ranks the remaining candidates by estimating how reliably the current policy can reprove them. For each candidate theorem $g \in \mathcal{G}$, we record the number n_g of times g was sampled as a proof goal and the number m_g of those episodes in which g was proved. A theorem with low reprovability has already been discovered, but the current policy cannot reliably prove it again. Making such a theorem available as a lemma can compress a difficult partial proof into a single lemma call. To estimate reprovability, we use the Bayesian posterior mean $\hat{p}_g = (m_g + 1)/(n_g + 2)$, which corresponds to a Beta$(1, 1)$ prior, rather than the raw empirical success rate. This smoothing prevents the ranking from over-prioritizing theorems that fail after only a few samples and helps identify theorems with persistently low reprovability.

Finally, generality filtering and reprovability ranking are combined to select useful theorems from $\mathcal{G}$. The method first keeps only the theorems that are not specializations of other candidates, thereby removing redundant specializations. It then sorts the remaining candidates so that lower values of the reprovability posterior mean $\hat{p}_g$ receive higher priority. Finally, a fixed number of the highest-priority theorems are selected from this ranking and regarded as useful theorems.

4.3. Reusing Useful Theorems

The proposed method reuses extracted theorems by adding them to the action space of the stack-machine decision process. The original action space consists of four actions: pushing an instance of `Ax1`, `Ax2`, or `Ax3` onto the stack, and applying the inference rule `MP`. For each extracted theorem, we add one library action, denoted by `Thm1`, `Thm2`, . . . , `ThmN`. When one of these theorem actions is selected, the corresponding theorem is pushed onto the stack instead of an axiom. The resulting stack-machine decision process allows the agent to reuse theorems discovered in earlier generations as lemmas in later proof search.

Algorithm 2 summarizes how the action space is expanded across generations. The algorithm alternates between policy learning under a fixed action space and action-space expansion using extracted theorems. In each generation, we train a new policy network for the current action space, because adding library actions changes the action space of the stack-machine decision process. During policy learning, reached

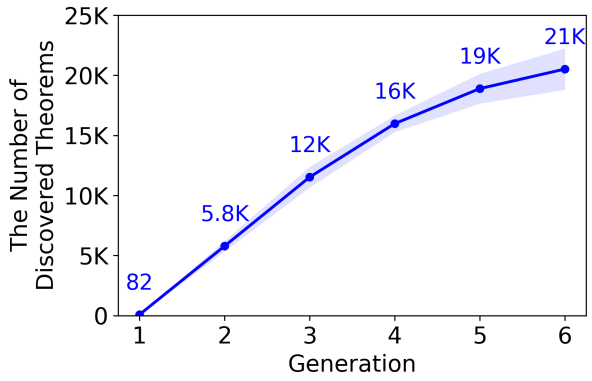

Figure 4. Number of discovered theorems in the Hilbert axiom system. The count increases over generations, indicating that the agent can grow a theorem library within the Hilbert axiom system.

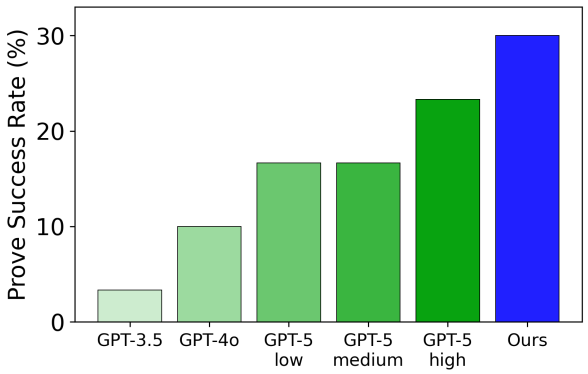

Figure 5. Proof success rates on human-written benchmark problems. Our agent finds Hilbert-system proofs for 30% of them, while GPT baselines indicate the difficulty of the task.

theorems accumulate in the goal buffer $\mathcal{G}$ through multiple iterations of proof search and supervised updates. At the end of the generation, useful theorems are extracted from $\mathcal{G}$, and the corresponding library actions are added to the action space $\mathcal{A}$. This process turns theorems discovered in one generation into reusable action primitives for later generations. By repeating this process, the agent gradually expands its action space and grows a library of useful theorems.

Algorithm 2 Multi-Generation Library Expansion

Initialize action space $\mathcal{A} \leftarrow \{\texttt{Ax1}, \texttt{Ax2}, \texttt{Ax3}, \texttt{MP}\}$.
for each generation **do**
  Give $\mathcal{A}$ to Algorithm 1 and obtain theorem buffer $\mathcal{G}$.
  Extract useful theorems $\{\texttt{Thm1}, \ldots, \texttt{ThmN}\}$ from $\mathcal{G}$.
  Expand $\mathcal{A} \leftarrow \mathcal{A} \cup \{\texttt{Thm1}, \ldots, \texttt{ThmN}\}$.
end for

5. Experiments

5.1. Experimental Setup

We run theorem discovery and library construction in the Hilbert axiom system shown in Figure 1. The initial action space consists of `Ax1`, `Ax2`, `Ax3`, and `MP`. Thus, before any library theorem is extracted, the agent can only push instances of the three axioms onto the stack or apply Modus Ponens. We repeat proof search and library extraction over six generations. The numbers of extracted theorems added to the library are $20, 10, 5, 2, 1, 0$ from Generation 1 to Generation 6, which yields a final library of 38 theorem actions.

For benchmark evaluation, we use the 30 human-written propositional-logic problems derived from Kleene's textbook (Kleene, 1952) and curated by Poesia et al. (2024). This benchmark provides a human-written test set that is independent of our theorem-discovery process. Each LLM baseline is evaluated once per problem with fixed decoding settings, so the reported success rates reflect single-shot proof generation rather than best-of-k sampling. A problem is counted as solved only when the generated action sequence is accepted by a stack-machine proof checker implementing `Ax1`–`Ax3`, theorem actions, and Modus Ponens with the same unification rule as in Section 3. In the lemma-augmented setting, each theorem action pushes the corresponding theorem from the extracted library onto the stack. Since each extracted theorem is stored with its primitive axiom/`MP` proof trajectory, lemma-augmented proofs can in principle be expanded into axiom-only Hilbert proofs.

5.2. Quantitative Evaluation of Theorem Discovery

We first evaluate whether the proposed method can grow the goal buffer $\mathcal{G}$ starting only from the axioms and inference rule of the Hilbert axiom system defined in Section 2.

Figure 4 shows that the number of discovered theorems increases over generations. This result demonstrates that our approach accumulates found theorems and reuses them to derive further theorems within the formal axiomatic system.

We next evaluate coverage on the human-written benchmark problems. Discovering many theorems alone does not guarantee that they are meaningful from a human perspective. This benchmark consists of classical-logic problems. As shown in Figure 5, the discovered theorem set covers 30% of these benchmark problems. This result suggests that the agent does not merely generate many theorems, but also discovers theorems that are meaningful from a human perspective.

We further evaluate the difficulty of solving these benchmark problems in the Hilbert system using LLM baselines. Figure 5 shows the fraction of benchmark problems for which each GPT baseline (OpenAI, 2023; 2024a; 2025)

Table 1. Qualitative examples in which GPT-5-high failed without prompt lemmas but succeeded when supplied with extracted theorem actions. The last column lists the theorem actions used in each successful proof.

Example	Goal formula	Used theorem actions
1	$A \to (\neg A \to B)$	`Thm1`, `Thm2`
2	$(A \to B) \to ((B \to C) \to (A \to C))$	`Thm3`, `Thm5`
3	$\neg((A \to B) \to \neg(B \to A)) \to (B \to A)$	`Thm1`, `Thm2`, `Thm3`, `Thm4`

Table 2. Extracted theorem actions appearing in the successful qualitative proofs in Table 1. Each action label denotes a discovered theorem supplied to GPT-5-high as a prompt lemma.

Action	Theorem formula
`Thm1`	$\neg A \to (A \to B)$
`Thm2`	$(A \to (B \to C)) \to (D \to ((A \to B) \to (A \to C)))$
`Thm3`	$(A \to B) \to ((C \to A) \to (C \to B))$
`Thm4`	$(\neg A \to \neg(B \to (C \to B))) \to A$
`Thm5`	$((A \to (B \to C)) \to (((A \to B) \to (A \to C)) \to D)) \to ((A \to (B \to C)) \to D)$

finds a Hilbert-system proof path. For GPT-5, low, medium, and high denote the reasoning effort used at inference time. Even GPT-5-high, the strongest GPT baseline in our evaluation, reaches only about 23% accuracy, which indicates that constructing Hilbert-system proofs for these benchmark theorems is nontrivial. Here, we do not aim to claim that the proposed method generally outperforms state-of-the-art LLMs. Rather, these results show that the benchmark problems are difficult to solve in the Hilbert system and indicate that our approach can discover nontrivial proofs.

5.3. Quantitative Evaluation of Extracted Theorems

We evaluate whether the extracted theorems also function as external knowledge for LLM proof search. Specifically, we provide the extracted theorems as prompt lemmas and compare the proof success rate of GPT baselines on human-written benchmark problems. This evaluation tests whether theorems obtained by our proof-search agent can transfer to LLMs, which rely on a different reasoning architecture.

Figure 6 compares the proof success rate with and without the extracted theorems provided as prompt lemmas. For GPT-4o and the GPT-5 variants, proof success rates improve when the extracted theorems are provided as prompt lemmas. This result shows that the extracted theorems can help LLMs find proofs in the Hilbert system. Overall, these results show that theorems extracted through our approach function as transferable knowledge for LLM-based theorem proving.

5.4. Qualitative Analysis of Extracted Theorems

Finally, we examine concrete examples of how extracted theorems assist LLM proof construction. Table 1 summarizes three goals for which GPT-5-high failed to find a proof without prompt lemmas but succeeded when the extracted

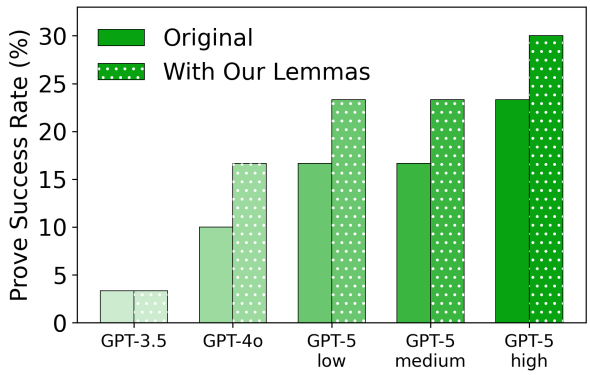

Figure 6. Effect of extracted theorems as prompt lemmas for LLM proof search. Providing extracted theorems improves proof success rates for GPT-4o and GPT-5 variants.

theorem library was supplied. Table 2 lists the extracted theorem actions that appear in these successful proofs.

The first example is especially informative because the two lemmas used in the proof have different roles. With the extracted theorems provided as prompt lemmas, GPT-5-high constructed the following Hilbert-system proof action sequence:

$$\texttt{Ax1}, \texttt{Thm1}, \texttt{Thm2}, \texttt{MP}, \texttt{Ax2}, \texttt{MP}, \texttt{MP}.$$

In this proof, `Thm1` corresponds to the principle of explosion, deriving an arbitrary B from $\neg A$ and A. Thus, the usefulness of `Thm1` is easy to interpret from a human perspective. In contrast, `Thm2` does not appear among the human-written benchmark problems and was discovered by our approach. Although its usefulness is not obvious at first glance, `Thm2` lifts an implication under an additional con-

text and contributes to proving the first goal in Table 1 when combined with `Thm1`. This contrast shows that usefulness is not always apparent from human inspection alone.

These examples show that extracted theorems do not merely improve aggregate evaluation metrics, but also play concrete roles in individual proof constructions. In particular, not only human-interpretable theorems such as `Thm1` and `Thm3`, but also theorems whose usefulness is not obvious at first glance, such as `Thm2`, `Thm4`, and `Thm5`, function as effective supporting knowledge when an LLM constructs Hilbert-system proofs. More broadly, this suggests that theorem discovery without human prior knowledge may allow AI systems to surface useful lemmas that humans did not explicitly provide.

6. Related Work

Learning-guided formal theorem proving. Prior work on learning-guided formal theorem proving has studied reinforcement-learning proof search, tactic prediction, proof generation, and LLM-based proving in Lean (Kaliszyk et al., 2018; Han et al., 2022; Polu et al., 2023; Yang et al., 2023; Xin et al., 2024; Zimmer et al., 2025; Hubert et al., 2026). These methods build strong provers using existing formal libraries or proof data. In contrast, we ask how far theorem discovery and reuse can go using only the three axioms and one inference rule of a Hilbert axiom system.

Self-play and intrinsic theorem discovery. Another line reduces dependence on human proof examples by learning proof-search or conjecturing through self-play or intrinsic motivation (Wu et al., 2021; Lample et al., 2022; Poesia et al., 2024; Laurent & Platzer, 2022; Dong & Ma, 2025; Zhao et al., 2025). Kasriel et al. (2025) is especially close because it discovers useful theorems from axioms, reuses them for further search, and evaluates external usefulness by asking an LLM judge whether they appear useful. Complementing this direction, we ask whether self-supervised theorem-discovery methods can find theorems that not only appear useful, but also actually improve LLM-based proof search when supplied as lemmas.

Goal-conditioned and hindsight learning. Our learning rule relates to Hindsight Experience Replay and goal-conditioned supervised learning, which reuse reached states as goals (Andrychowicz et al., 2017; Ghosh et al., 2021). In our setting, the reached objects are theorems, not environment states, and some are reused as both training goals and action primitives in subsequent stack-machine decision processes. Thus, a proof prefix reaching a formula becomes hindsight supervision for proving it. This connects goal relabeling with theorem-proving library learning.

Lemma discovery and library learning. Lemma discovery and library learning are important directions for accelerating proof search (Yang et al., 2023; Wang et al., 2024). In contrast to methods that rely on existing formal libraries, human-provided lemmas, or LLM-based proving pipelines, our agent discovers useful theorems through proof search using only the axioms and inference rule.

Curry–Howard motivation. Our stack-machine view is also motivated by the Curry–Howard correspondence, which relates propositions to types and proofs to programs (Howard, 1980; Sørensen & Urzyczyn, 2006). There, implication resembles a function type and `MP` resembles function application. This is not central to our claims, but explains why Hilbert proofs can be flattened into action sequences that push axioms or lemmas and compose them on a stack. Our `MP` implementation follows this intuition by unifying an implication antecedent with another stack formula, as in Hindley–Milner type inference (Hindley, 1969; Milner, 1978; Damas & Milner, 1982).

7. Conclusions

We studied whether useful theorems can be discovered within a formal axiomatic system using only its primitive rules. We formulated Hilbert-system propositional theorem proving as a stack-machine decision process and proposed a learning algorithm that grows a theorem library through proof search. The algorithm treats reached theorems as future proof goals and extracts useful theorems for reuse as lemmas. Experiments show that the agent discovers many theorems over multiple generations in the Hilbert axiom system and finds proofs for a meaningful fraction of the human-written benchmark problems. We also demonstrate that the extracted theorems improve LLM proof performance when provided as prompt lemmas.

A limitation of this study is that our experiments are conducted in a single axiom system. However, the proposed framework is not specific to this axiom system and should be applicable to multiple axiom systems; evaluating this broader applicability is a natural next step. Furthermore, it also remains an important direction for future work to test whether the same approach can scale to richer formal systems with broader applications, such as first-order logic, set theory, and more general mathematical domains.

While we have focused on Hilbert-style propositional theorem proving, our experiments provide evidence that useful theorems can emerge from proof search without relying on human-provided theorem libraries. More broadly, these findings suggest a path toward self-evolving AI systems for mathematics: systems that autonomously expand their formal knowledge through proof search while keeping each discovery formally verifiable.

Acknowledgements

This work was partially supported by JST Moonshot R&D Grant Number JPMJPS2011, CREST Grant Number JPMJCR2015 and Basic Research Grant (Super AI) of Institute for AI and Beyond of the University of Tokyo. Kazuki Ota was supported by JST SPRING, Grant Number JPMJSP2108. Takayuki Osa was supported by JSPS KAKENHI Grant Number JP25K03176.

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, pp. 5048–5058, 2017.

Azerbayev, Z., Schoelkopf, H., Paster, K., Dos Santos, M., McAleer, S., Jiang, A. Q., Deng, J., Biderman, S., and Welleck, S. Llemma: An open language model for mathematics. In *International Conference on Learning Representations*, 2024.

Church, A. *Introduction to Mathematical Logic*. Princeton University Press, Princeton, NJ, 1996. Reprint of the revised 1956 edition. See Propositions 235 and 239, pp. 127–128.

Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158. Association for Computing Machinery, 1971.

Cook, S. A. and Reckhow, R. A. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

Damas, L. and Milner, R. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 207–212. Association for Computing Machinery, 1982.

Dong, K. and Ma, T. STP: Self-play LLM theorem provers with iterative conjecturing and proving. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 14114–14136. PMLR, 2025.

Farquhar, S., Kossen, J., Kuhn, L., and Gal, Y. Detecting hallucinations in large language models using semantic entropy. *Nature*, 630:625–630, 2024. doi: 10.1038/s41586-024-07421-0.

Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. arXiv preprint arXiv:2507.06261, 2025.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C., Eysenbach, B., and Levine, S. Learning to reach goals via iterated supervised learning. In *International Conference on Learning Representations*, 2021.

Han, J. M., Rute, J., Wu, Y., Ayers, E. W., and Polu, S. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*, 2022.

Hilbert, D. and Ackermann, W. *Principles of Mathematical Logic*. Chelsea Publishing Company, New York, 1950. Translated by Lewis M. Hammond, George G. Leckie, and F. Steinhardt.

Hindley, R. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

Howard, W. A. The formulae-as-types notion of construction. In Seldin, J. P. and Hindley, J. R. (eds.), *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 479–490. Academic Press, New York, 1980.

Hubert, T., Mehta, R., Sartran, L., Horvath, M. Z., Zuzic, G., Wieser, E., Huang, A., Schrittwieser, J., Schroecker, Y., Masoom, H., Bertolli, O., Zahavy, T., Mandhane, A., Yung, J., Beloshapka, I., Ibarz, B., Veeriah, V., Yu, L., Nash, O., Lezeau, P., Mercuri, S., Sonne, C., Mehta, B., Davies, A., Zheng, D., Pedregosa, F., Li, Y., von Glehn, I., Rowland, M., Albanie, S., Velingker, A., Schmitt, S., Lockhart, E., Hughes, E., Michalewski, H., Sonnerat, N., Hassabis, D., Kohli, P., and Silver, D. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026. doi: 10.1038/s41586-025-09833-y.

Jiang, A. Q., Welleck, S., Zhou, J. P., Lacroix, T., Liu, J., Li, W., Jamnik, M., Lample, G., and Wu, Y. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *International Conference on Learning Representations*, 2023.

Kalai, A. T., Nachum, O., Vempala, S. S., and Zhang, E. Why language models hallucinate. arXiv preprint arXiv:2509.04664, 2025.

Kaliszyk, C., Urban, J., Michalewski, H., and Olšák, M. Reinforcement learning of theorem proving. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Kasriel, T., Lu, T., Ding, Q., He, J., and Song, D. Usefulness-driven learning of formal mathematics. In *The 5th Workshop on Mathematical Reasoning and AI (MATH-AI) at NeurIPS*, 2025.

Kleene, S. C. *Introduction to Metamathematics*. Van Nostrand, 1952.

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*, volume 35, 2022.

Laurent, J. and Platzer, A. Learning to find proofs and theorems by learning to refine search strategies: The case of loop invariant synthesis. In *Advances in Neural Information Processing Systems*, volume 35, pp. 4843–4856, 2022.

Luong, T. and Lockhart, E. Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad. Google DeepMind blog, 2025. URL: https://deepmind.google/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/. Accessed: May 25, 2026.

Mendelson, E. *Introduction to Mathematical Logic*. CRC Press, Boca Raton, 6 edition, 2015. ISBN 9781482237726.

Milner, R. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.

OpenAI. Introducing APIs for GPT-3.5 Turbo and Whisper, 2023. OpenAI blog. URL: https://openai.com/index/introducing-chatgpt-and-whisper-apis/. Accessed: May 25, 2026.

OpenAI. GPT-4o system card, 2024a. URL: https://openai.com/index/gpt-4o-system-card/. Accessed: May 25, 2026.

OpenAI. OpenAI o1 system card, 2024b. URL: https://openai.com/index/openai-o1-system-card/. Accessed: May 25, 2026.

OpenAI. GPT-5 system card, 2025. URL: https://openai.com/index/gpt-5-system-card/. Accessed: May 25, 2026.

Poesia, G., Broman, D., Haber, N., and Goodman, N. D. Learning formal mathematics from intrinsic motivation. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Polu, S., Han, J. M., Zheng, K., Baksys, M., Babuschkin, I., and Sutskever, I. Formal mathematics statement curriculum learning. In *International Conference on Learning Representations*, 2023.

Post, E. L. Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43(3): 163–185, 1921.

Sørensen, M. H. and Urzyczyn, P. *Lectures on the Curry–Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006. ISBN 9780444520777.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024. doi: 10.1038/s41586-023-06747-5.

Wang, H., Xin, H., Zheng, C., Liu, Z., Cao, Q., Huang, Y., Xiong, J., Shi, H., Xie, E., Yin, J., Li, Z., and Liang, X. LEGO-Prover: Neural theorem proving with growing libraries. In *International Conference on Learning Representations*, 2024.

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*, volume 34, 2021.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

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. arXiv preprint arXiv:2405.14333, 2024.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. LeanDojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Zhao, A., Wu, Y., Yue, Y., Wu, T., Xu, Q., Yue, Y., Lin, M., Wang, S., Wu, Q., Zheng, Z., and Huang, G. Absolute zero: Reinforced self-play reasoning with zero data. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Zimmer, M., Ji, X., Tutunov, R., Bordg, A., Wang, J., and Bou Ammar, H. Bourbaki: Self-generated and goal-conditioned MDPs for theorem proving. arXiv preprint arXiv:2507.02726, 2025.

A. Experimental Details

In this section, we explain implementation details for theorem-discovery training and library construction. Each generation uses 25 training iterations. In each iteration, the agent generates 8192 episodes, and each episode has a maximum length of 7 steps. Therefore, each generation uses $25 \times 8192 \times 7$ simulator evaluations. Episodes are processed in batches of 2048 episodes.

The policy is parameterized by a Transformer network. The model uses embedding dimension 128, four Transformer blocks, four attention heads, feed-forward dimension 512, dropout rate 0.1, and learned positional encodings. The policy is trained with cross-entropy loss under a legal-action mask. We use Adam with learning rate 0.001, batch size 512, and one training epoch per iteration.

The policy observation is a fixed-length sequence of 1024 integer tokens. It consists of the sampled goal formula followed by the current proof stack, separated by a special field-separator token, and padded to a fixed length. Each formula is tokenized at the symbol level in reverse Polish notation, with tokens for implication, falsity, variables, and formula separators. Thus, a training example for $\pi_\theta(A|S, g)$ presents the goal formula g and the current stack S to the Transformer.

Across generations, the action space consists of the three axiom actions, the `MP` action, and the library theorem actions available at that generation. The legal-action mask is computed from the current stack, the remaining horizon, and the current action space. It is used to mask illegal actions during sampling and in the cross-entropy loss. When new theorems are extracted and added to the library, the action space changes accordingly. We therefore initialize and train a new policy network for the updated action space at each generation.

B. Prompts for LLM Evaluation

We show the prompt template used for LLM evaluation. In the prompt, the axioms corresponding to `Ax1`, `Ax2`, and `Ax3` in the main text are denoted by 'A01', 'A02', and 'A03', respectively, and Modus Ponens is denoted by 'MP'. When discovered theorems are provided as lemmas, they are listed as 'L01', 'L02', and so on. For the baseline without lemmas, this lemma list is empty, and the LLM is asked to construct a proof using only 'A01', 'A02', 'A03', and 'MP'. The final answer is requested as an RPN sequence in which actions are separated by spaces.

```
# Rules
Prove the given proposition in the Hilbert system (with axioms A01, A02, A03 and the
inference rule Modus Ponens).

## Axioms:
A01: A -> (B -> A)
A02: (A -> (B -> C)) -> ((A -> B) -> (A -> C))
A03: (!A -> !B) -> (B -> A)

## Inference Rule (MP):
In the stack, the top element is regarded as X->Y, and the one below it as X. Unify the
antecedents of X and X->Y as in Hindley-Milner type inference, and as a result derive the
consequent Y, which is then pushed onto the stack. For example, if the top of the stack is
 (A->(B->C))->((A->B)->(A->C)) and the next one is A->(B->A), then A->(B->C) and A->(B->A)
 are unified with A=C, yielding the result (A->B)->(A->A).

## Lemmas:
You can use the following lemmas if you need.
L01: (A -> B) -> ((C -> A) -> (C -> B))
L02: (A -> (B -> C)) -> (B -> (A -> C))

# Output Format
The proof must ultimately be represented as a combinator expression in Reverse Polish
Notation (RPN). In RPN, A01, A02, A03 each push an instance of the corresponding axiom
onto the stack, L01, L02, ... do the same for lemmas, and MP represents Modus Ponens
according to the above rule. You may include explanations, but enclose the final proof
expression between <final_answer> and </final_answer>.

# Example Problem
Target proposition: A -> A
Reference proof steps:
```

```
1. A -> (B -> A) (Axiom A01)
2. A -> (B -> A) (Axiom A01)
3. (A -> (B -> C)) -> ((A -> B) -> (A -> C)) (Axiom A02)
4. (A -> B) -> (A -> A) (Apply MP to 3 and 2)
5. A -> A (Apply MP to 4 and 1)
In this case, the combinator expression in RPN is A01 A01 A02 MP MP. Final output example
is the following:
<final_answer>A01 A01 A02 MP MP</final_answer>

# Problem
Prove the following proposition:
(A -> B) -> ((B -> C) -> (A -> C))
```