Neural Networks for Mathematical Reasoning – Evaluations, Capabilities, and Techniques

by

Yuhuai Tony Wu

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy

Department of Computer Science
University of Toronto

Neural Networks for Mathematical Reasoning
– Evaluations, Capabilities, and Techniques

Yuhuai Tony Wu
Doctor of Philosophy

Department of Computer Science
University of Toronto
2024

Abstract

This thesis investigates the potential of neural networks as a powerful approach to reasoning. It argues that neural networks can effectively leverage statistical structures to learn useful heuristics and model arbitrary action distributions, making them well-suited for reasoning tasks. To test this claim, the thesis focuses on mathematical reasoning as a well-defined domain with broad applications. The author sets up various benchmarks to define the problem and measure progress in this emerging field. Through extensive experimentation, the thesis demonstrates that neural networks can perform non-trivial mathematical reasoning tasks in both abduction and induction. Additionally, the author shows that various techniques, such as improved inductive bias design, high-level proof sketching, and self-supervised learning, can enhance neural networks' mathematical reasoning capabilities. Overall, this thesis provides compelling evidence that neural networks are a promising tool for mathematical reasoning, with the potential to surpass existing methods in the field.

Acknowledgements

Time flies! As I look back on my PhD journey, I am grateful to have received the support of so many individuals.

I would like to express my deepest gratitude to my PhD advisor, Roger Grosse. His support and guidance have been invaluable to me throughout my PhD journey, which was not an easy one. For the first four years of my PhD, I explored various ideas without settling on a specific direction. Despite this, Roger gave me tremendous freedom to pursue my research interests and never pushed me to work on something that I was not interested in. Instead, he provided invaluable support and encouragement as I explored different directions, regardless of whether or not they aligned with his own interests. I also learned so much from him, including his rigorous attitude towards science, his deep insights into machine learning, and exceptional writing and presentation skills. I am truly fortunate to have had him as my advisor, and I will always cherish his guidance and support.

I would also like to thank my PhD co-advisor Jimmy Ba, whose brilliant ideas and insightful feedback have been invaluable to my research. Brainstorming with him has been a delightful experience, and his guidance has helped shape my research direction.

I am deeply indebted to Christian Szegedy, my mentor in the field of mathematical reasoning. His guidance and support have been instrumental in my research journey. He led me into the field, and together we experienced many moments of surprises and fun.

I also feel fortunate for having met James McClelland at Deepmind in the summer of 2018. At that time, I was uncertain about the focus of my Ph.D. thesis, but Jay's talk on mathematical reasoning completely convinced me of focusing on this topic. Since then, Jay has been an invaluable mentor, imparting knowledge on neural networks, human intelligence, and life itself. Later he also helped to recruit me to Stanford for a PostDoc study with him and Percy Liang. I am very grateful for everything he did for me, and having a chance to know him and learn from him.

Another person I would like to express my huge gratitude to is Yoshua Bengio. Back in 2014, I was a pure mathematics student with no knowledge of computer science or programming. I sent an email to Yoshua expressing my interest in an internship, with little hope of receiving a reply. Much to my surprise, Yoshua not only responded but also offered me an internship position for the summer of 2015. This opportunity became the foundation of my machine learning career. Since then, Yoshua has remained a steadfast source of support and encouragement in my research endeavor.

I want to express my sincere gratitude to my parents, Shilai Wu and Chao Li, who have been my pillars of strength throughout my life. Their love, support, and unwavering belief in me have been a constant source of inspiration. They encouraged me to pursue my dreams and never gave up on me, no matter how tough things got. Their love and guidance have been instrumental in my success.

I am also incredibly grateful for my wife, Ziwen Tan, who has been my rock through thick and thin. Her support, encouragement, and companionship have been invaluable to me. Her belief in me never wavered, and her love and support have been my anchor throughout my PhD journey.

I am thankful for the endless fun and adventure that my friends Shun Liao and Yiqing Yang brought to my life, whether it was ice-fishing in Alaska or skiing in Banff.

Finally, I would like to acknowledge the many other individuals who have contributed to my research journey, including my friends, colleagues, and mentors: Saizheng Zhang, Albert Jiang, Behnam Neyshabur, Eric Zelikman, Markus Rabe, Cem Anil, Szymon Tworkowski, Felix Li, Wenda Li, Jesse Han, Jin Zhou, Jiaming Song, Ethan Dyer, Stanislas Polu, Honghua Dong, Mengye Ren,

Renjie Liao, Yura Burda, Ruslan Salakhutdinov, Geoffrey Hinton, Sheila Mcilraith, Percy Liang, Pieter Abbeel, David Duvenaud, Oriol Vinyal, Radford Neal, and many more. Your support and encouragement have been instrumental in my success.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Reasoning and AI

Over the past decade, the field of artificial intelligence has made significant advancements in recognizing perceptual patterns, such as image recognition and object detection, largely due to breakthroughs in neural network research. However, one defining property of advanced intelligence – reasoning [104] – goes beyond mere perceptual understanding and has not yet been fully demonstrated in artificial intelligent systems.

In classical AI, reasoning is often achieved by *search* [104], and a large body of algorithms has been developed over the years [104, Section 3.5]. However, many algorithms predominantly depend on hand-crafted heuristics, which introduce a series of challenges. First, the creation of these heuristics demands extensive human intervention. It necessitates an in-depth understanding of the task domain, and even with ingenuity and a bit of luck, identifying effective problem-solving strategies can be arduous. Once identified, these strategies must be translated into formal heuristics, a process that can be time-consuming, particularly for intricate tasks with myriad edge cases. Second, such heuristics are often tailored to specific domains, relying heavily on extensive domain knowledge. This specialization limits their applicability to a narrow set of problems. Finally, regardless of their innovative designs, human-engineered heuristics rarely, if ever, represent optimal strategies. Improving upon these heuristics also lacks a systematic method, typically requiring the insights of domain experts.

Recent progress demonstrated that a more efficient search can be achieved by combining search with learning [10]. One notable example example of such a blend is the invention of AlphaGo [112] and subsequently AlphaZero [113] for solving a range of classical board games (Go, Chess, Shogi). The algorithm makes use of powerful neural networks to exploit the statistical structures of winning experience to distill better and better value functions. It resolves some of the previously mentioned issues by first not relying on any human labor when constructing heuristics, secondly not relying on domain knowledge and the methodologies can be applied to all three board games, and lastly, the algorithm exhibits a self-improvement cycle that allows the heuristic to be improved as one learns.

1.1.1 Beyond Search: Unboundedness and Generativity

Let's consider a few more scenarios that employ reasoning.

Playing simple games Playing Go, Chess, Shogi, Sudoku, mazes. These classical games are those domains where the combination search and learning methods have already demonstrated great successes. One common attribute is that the action space of the player, which determines the branching factor of the search, is finite, well-defined, and fixed. For example, in the game of Go, the action space is constrained by all grid points on the board.

Running a Company Imagine one is a CEO facing a sudden drop in company sales. The task of the CEO is to determine the root cause of this downturn. After a thorough analysis of company data and market conditions, one identifies certain patterns and anomalies which might have contributed to the decline. Using the principle of abduction based on these findings, one hypothesizes potential reasons for the sales drop, such as a faulty product, ineffective marketing strategy, or external market shifts. With a hypothesis in hand, one seeks further data to validate or refute it, leading to subsequent refined hypotheses until the core issue is identified and addressed.

Mathematical theorem proving In mathematical theorem proving, one is given a set of axioms and a targeted mathematical proposition, and the goal is to demonstrate a sequence of logical deductions that prove the targeted proposition (or to show a counterexample that disproves it). Oftentimes, one makes use of *abduction* to hypothesize what needs to be true to prove the target proposition and find ways to support the hypothesis[1]. One can also obtain more facts and knowledge by playing with mathematical examples and making useful observations.

Scientific discovery Scientific discovery delves into unraveling the intrinsic laws of nature. Faced with observations from the natural realm, the challenge is to discern a rigorous scientific theory that elucidates the foundational cause. This endeavor bears resemblance to the roles of CEO piecing together theories for company downturn and mathematicians drafting theorems to connect established facts with pending proofs. Grounded in observed phenomena, scientists postulate hypotheses. Importantly, the formulated hypothesis often seeks to generalize existing phenomena, invoking the use of *induction*. This mirrors the mathematician's practice of extrapolating specific mathematical observations to broader theorems (e.g., progressing from dimensions 1 and 2 to n). However, in science, these laws come from nature, rather than a predefined set of axioms.

Intuitively, the latter three reasoning applications are also a search process – searching for the cause that leads to the downturn, for the sequence of deduction that proves the theorem, and for the true nature law that explains the underlying phenomenon. However, they are very different from the kind of search employed in the classical games. The main difference lies in the action spaces of the latter three: they are *unbounded* and *generative*.

The action space is *unbounded*, meaning that one can potentially search over infinitely many possible actions at each step of the search process. Let's consider the problem of mathematical theorem proving, where the action at each step is a sequence of tokens (a token is a subword) forming an intermediate proposition. The intermediate proposition can be arbitrarily long, hence the space of possible actions is infinitely large. Perhaps one could set a limit of 10 tokens for each proposition. But even with 10 tokens the branching factor can be not much different from infinite:

[1] We will demonstrate more of this in Chapter 5.

say the vocabulary size is 20000, then the branching factor, i.e. the possible number of messages, is $20000^{10} \sim 10^{43}$.

The action space is also *generative*, meaning that one needs to generate actions instead of enumerating over a predefined set. For example, in the case of mathematical theorem proving, one needs to come up with original conjecture from a vast space of possibilities. Because of the unbounded action space, the existing successful methods (including AlphaZero) that expand the search node over all possible actions cannot be applied. Methods that allow modeling the distribution of actions and that are capable of sampling actions are hence essential for addressing these challenges.

Additionally, for reasoning problems such as mathematical theorem proving and scientific discovery, the foundational meaning of the action space can evolve. When one proposes new definitions or proves new lemmas, one can then discuss newly defined concepts or apply newly proved lemmas as new actions. Consequently, the action space is dynamic and evolving, rather than static and fixed.

Therefore, reasoning should not be understood just as the traditional search – it is a kind of search process, but the approach we are looking for is much more open-ended and powerful. Recent progress in neural network models shows that they may be a good candidate to resolve the action generation problem: they are good at capturing interesting distributions and capable of generating faithful samples from those distributions. In particular, the large-scale unsupervised pre-training language models revolutionized the field of natural language processing [100, 26, 144, 77, 101]. The GPT-3 [15] pushes the model scale to a new level with 175B parameters. Besides improving performances on common benchmarks, the GPT-3 model also demonstrates many impressive demonstrations that require generation. For example, it is able to generate realistic news articles [15], novel python code given the docstring[2], layouts in JSX code given a sentence of natural language description[3]. Encouraged by such promising evidence, we mainly focus on neural networks and especially language models in our exploration of reasoning mechanisms.

To study the reasoning problem concretely, we focus on a specific domain in this thesis: mathematical theorem proving. Advanced mathematical reasoning is unique in human intelligence, and it is also a fundamental building block for many intellectual pursuits and scientific developments. We believe that investigating this reasoning domain has the potential to shed light on a path towards general reasoning mechanisms.

1.2 Mathematical Reasoning Problem

Mathematics is a fascinating subject, a pinnacle of human reasoning achievements, filled with all kinds of complexities and challenges. The mathematical reasoning problem refers to the vision of building a versatile mathematician artificial intelligence that is able to prove a large number of (if not all) useful and challenging mathematical theorems that are interesting to humans. One concrete example could be proving the Riemann Hypothesis that no human has yet been able to prove.

In this thesis, we mainly focus on what is often referred to as *formal mathematical theorem proving*. The formal theorem proving interface is different from what normal mathematicians work with: instead of writing an *informal mathematical proof* on sheets of paper or typed in LaTeX, one writes a *formal* proof with an interactive theorem prover (ITP, also known as a proof assistant),

[2] https://www.youtube.com/watch?v=fZSFNUT6iY8.
[3] https://twitter.com/sharifshameem/status/1282676454690451457.

a computer software tool that assists the developments of the proof. The form and level of detail involved may differ, but the logical content is similar between informal and formal proofs. The formalized proof builds on a foundation of mathematics used by the ITP, such as the simply-typed lambda calculus used by Isabelle, and each step of the proof is formally checked by the ITP. Because of this, formal mathematical proving provides a simulated environment, which makes it a convenient starting point for developing machine learning models. They not only verify the proof automatically (i.e., automatic evaluation of machine learning models) but also allow the models to interact with it and discover new proofs.

In the following section, we formulate the formal reasoning problem as a Markov Decision Process: a starting point to tackle the mathematical reasoning problem. We are aware of the limitations of the formulation and will expand our discussion on our vision of the general mathematical reasoning problem in Chapter 9.

1.2.1 Formal Mathematical Reasoning as a Decision Making Process

We model formal theorem proving as a Markov Decision Process $(\mathcal{S}, \mathcal{A}, \gamma, P, r)$. Given an interactive theorem prover (such as Isabelle),

- **State**: A state $s \in \mathcal{S}$ in the MDP is the *proof state* maintained by the ITP. The proof state includes three types of statements: 1. the remaining propositions that need to be proved, known as *goals* in ITPs, 2. the premises of the theorem, and 3. new facts that have been proven along the way.

- **Action**: An action $a \in \mathcal{A}$ in in the MDP as a sequence of tokens forming a *proof step*. Using a sequence of tokens to represent the action allows one to issue any kinds of proof steps: the theorem (name) that one invokes to apply to the current goals, ITP *tactics*[4], or intermediate propositions that one conjectures to be true and useful for proving the final goal. There are past works that use a more structured action space, which however constrain the kinds of actions the model can issue. We hence do not favor those approaches.

  Echoing the discussion from the last section, the action space of this problem is *unbounded*. Hence, most of the existing works use sequence generative models to *generate* such actions (e.g., sequence-to-sequence models [117]). However, sequence generation is often quite slow, as the generation of each new token requires a forward pass of the model. This also becomes another bottleneck one may face when interacting with the environment.

- **Transition function**: The deterministic state transition function $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is carried out by the ITP. The ITP first receives the proof step (action a), and if the action is invalid, then the state stays the same (or issued a warning message that "the action is invalid"). Otherwise, it applies the proof step to the existing proof state. As the proof proceeds, the goals will be transformed from more obscure ones to simpler ones, and at the same time, one collects more and more proven facts.

- **Reward**: Once there are no further remaining goals to be proved, then a theorem is proven. The episode terminates and a reward of one is given. Otherwise, the reward is zero at each

[4] See more discussion of this in Chapter 2.

step. Note that we present the cleanest reward function, but the downside is that it is a sparse reward. We leave a more sophisticated reward design to future investigations.

Note that such an MDP is defined with one target theorem in mind. For training and evaluation of machine learning models, we often assume there are a suite of training theorems and a suite of test theorems. The model performance is reported based on the percentage of the proved theorems, instead of the success of a single theorem[5]. From a learning perspective, the nature of such training and test split also poses brand new challenges: one may often need to generalize to theorems unlike those seen at training time. This breaks the common I.I.D. assumption in the existing machine learning paradigm. We will investigate the implications of this challenge in more detail in Chapter 3.

In addition, even though we formulate the formal reasoning problem under the fundamentals of Markov Decision Process, this does not constrain one to train machine learning models via major reinforcement learning algorithms (such as policy gradient methods to maximize the expected sum of rewards). In fact, as one can immediately recognize, the complexity of the action space and the sparse reward problem make it almost hopeless to train models using RL methods from scratch[6]. Most of the existing work, and those that will be discussed in this thesis, perform imitation learning on expert data (either from human-written formal libraries or synthetic generated proofs). One of the major future directions is designing better exploration methods to improve machine learning models, going beyond the environment of imitation learning.

One element we intentionally omitted for simplicity in this framework is the notion of a *global state* of the ITP. Besides the three elements that constitute the proof state, ITP also maintains a global library that keeps track of all the theorems that have been proved and can be invoked without further justification. The library contains a large number of theorems (e.g., 100k). Due to the limitations of existing machine learning models, we could not afford to present the entire library as an input to the machine learning model during learning and evaluation. Hence we assume the library is an external object that is pre-determined before proving a theorem. To avoid circular reasoning (invoking a theorem to prove itself), during theorem proving evaluation, we set the theorem library as the one that was used when humans wrote the theorem. Clearly, it is a major limitation of the current framework. The model is not only unaware of which theorem it can use during the proving process, but it also cannot flexibly add new theorems into the library during the proving process. We leave this as another major future direction in Chapter 9.

Lastly, even though formal reasoning is the focus of this thesis, we strongly believe that constraining the scope to formal reasoning is insufficient to achieve the end goal of building a versatile mathematician, largely because the existing human formalized library only contains a small fragment of all mathematics. We hence discuss the challenges ahead and the potential use of informal mathematics in Chapter 9.

1.3 Why Mathematical Reasoning

In this section, we discuss why mathematical reasoning is one of the most suitable fields (if not the most) to study the reasoning problem for the current machine learning community.

[5]It can be that sometimes we care the success of a particular proposition more than others (e.g. Riemann Hypothesis), but we find the proof success rate as a proxy metric suitable for developing machine learning models.

[6]There are some past works [4] showing successes with training machine learning models using RL from scratch, but these methods assume a very constrained action space, which allows them to explore efficiently.

1.3.1 Reasoning-complete

First and foremost, mathematical reasoning represents a significant challenge for both human and machine intelligence. In fact, it is often considered the pinnacle of human reasoning.

It shares many common attributes with other reasoning applications, such as scientific reasoning, logical reasoning, and philosophical reasoning. For instance, both mathematical and scientific reasoning require the ability to formulate and test hypotheses, analyze data, and draw conclusions based on evidence. Similarly, both logical and philosophical reasoning involve the application of formal rules and principles to make valid inferences and deductions.

Given these similarities, it is reasonable to expect that techniques and approaches that prove effective for mathematical reasoning may also have broader applications across other reasoning domains. By developing AI systems that can perform well on mathematical reasoning tasks, we can potentially enhance their overall reasoning capabilities and enable them to tackle a wider range of complex problems.

1.3.2 Principled and automatic metric for evaluation and groundings

The formal mathematical reasoning problem involves a proof assistant that automatically checks a given proof for correctness, that is, a principled and automatic metric. Compared to other main reasoning applications, having such a metric is a unique attribute of this problem, hence allowing one to not only evaluate but also ground symbols in the theorem proving environment.

Evaluation An essential attribute of the theorem proving problem distinguishes itself from the other reasoning applications, making it a better choice for a subject of study: it has an automated way of evaluating success. Namely, when an agent outputs a proof, a theorem proving assistant can check its correctness automatically. However, most of the reasoning domains such as natural language and program synthesis do not have such a principled and automatic metric. To evaluate a natural language agent, one can only resort to biased metrics (e.g., Bleu scores for machine translation [92]). Or, one needs to rely on humans to evaluate the quality of an agent, an approach that would require millions of human laborers. As for program synthesis, automatically evaluating the correctness of a program is exactly the job of an automated theorem prover, hence improving automated theorem proving directly contributes to research in program synthesis. As a fully automated verification technology is still in its infancy, one can only resort to unsatisfied solutions, such as writing unit tests and with randomly generated test inputs [68].

Grounding Moreover, the evaluation signal also allows the theorem proving agent to have a meaningful grounding of symbols in how a theorem is proved, beyond statistical correlations. Lack of groundings has been a common critique [9] to existing learning models for natural language understanding. For example, to train a language model, the objective weights every symbol equally and lacks a notion of what is most important to predict and what is less important. Hence the model is trained to only capture the statistical correlations between symbols. As GPT-3 authors noticed, the model trained on large text corpus seems to have special difficulty with "common sense physics", due to the lack of experience in the actual physical world. However, the simulated environment in theorem proving can alleviate this problem. Through interacting with the theorem proving

environment, the agent can learn to discover causal structures among the symbolic statements and do credit assignments accordingly.

1.3.3 Applications

There have been a few grand challenges we have attempted to solve over the years as a measure of progress in artificial intelligence: playing board games (Go, Chess, Shogi) [113], playing real-time computer strategy games, such as DotaAI [88], AlphaStar [124]. However, the impacts of these successes are limited because these gaming domains do not directly contribute to the development of any essential and practical applications. Unlike playing games, mathematical theorem proving has been a driving force for all science, ranging from Riemannian manifold theory that lays the foundation of General Relativity to the fundamental problem in artificial intelligence: P vs. NP. Solving these extremely difficult problems may be a long-term project, but a reasonably strong theorem proving agent can already catalyze developments in various fields, including formalizing mathematics [81], software verification [24], and hardware design [65]. Over the last decade, non-trivial mathematical theorems [44, 45, 47], security protocols [43, 96], and industry-scale software systems [67, 72] have been mechanically checked, highlighting numerous issues and bugs. Despite those successes, mechanization still requires substantial human effort: in the seL4 project [67], 20 person-years were devoted to verifying an OS kernel written in 8700 lines of C. Such a scale of human labor strongly motivates research for fully automated theorem proving.

1.4 Thesis Overview

We summarize the main claims of the thesis as follows.

Claim of the thesis Neural networks present a promising approach for reasoning, harnessing statistical structures to formulate effective heuristics, and modeling the distribution of reasoning actions while generating rich steps from it. We selected the domain of mathematical reasoning as our testbed due to its well-defined metrics and immense potential across a wide range of applications. In order to precisely define the problem and evaluate our research progress, we have established several benchmarks, with both synthetic and real-world examples. Through our work, we demonstrate the non-trivial mathematical reasoning abilities of neural networks in abduction and induction. Furthermore, we enhance these abilities through various techniques, such as improved inductive bias design and self-supervised learning.

Thesis structure To support the claims of the thesis, we divide the thesis into three parts: evaluations, capabilities, and techniques.

Evaluation: This part focuses on developing theorem proving benchmarks for the machine learning community. Through building these benchmarks, we concretely formulate the mathematical reasoning problems and identify major challenges and limitations. A good benchmark is also crucial for measuring and accelerating research progress, and hence conducive to the research community as a whole. In particular, we build two kinds of benchmarks – a synthetic benchmark and two realistic benchmarks. In Chapter 3, we introduce a synthetic inequality theorem proving benchmark, with two motivations: 1. to create a lightweight and swift benchmark, serving as the "MNIST" for

theorem proving, 2. to established well-controlled training and test distribution to investigate the out-of-distribution challenge in theorem proving. In Chapter 4, we further developed two realistic benchmarks with two commonly used ITPs: Isabelle and Lean, allowing machine learning models to make a real impact in realistic theorem proving practices.

Capabilities: This part of the thesis focuses on neural networks' capabilities in mathematical reasoning. We show examples of neural networks' successes with a focus on conjecturing and abstraction – two very essential and sophisticated mechanisms employed by human mathematicians. In Chapter 5, we assess neural networks' abilities to perform abduction. More concretely, we set up a task with Isabelle, where we ask the neural network model to conjecture the missing step in a logical deduction. Neural networks are demonstrated to achieve non-trivial reasoning abilities in this task. In Chapter 6, we take a first step towards using neural networks for recognizing modular structures and generalizing to more abstract patterns – a process of induction. In particular, we show neural network agent is able to extract commonly used theorems from proofs. Those newly extracted theorems help to refactor the entire theorem library and compress the existing proofs. Furthermore, by frequently using these newly extracted useful theorems to prove unseen theorems, the neural network models also achieve better theorem proving performance.

Techniques: This part of the thesis focuses on developing better techniques for neural networks in mathematical reasoning. We mainly focuse on three problems: 1. neural architecture inductive biases for reasoning, 2. high-level flexible reasoning, and 3. data scarcity problem in formal libraries. In Chapter 7, we first take a look at how to design better neural network inductive biases for reasoning. We find pretraining on synthetic tasks motivated by Pierce's reasoning primitives (deduction, induction, and abduction) greatly improves the performances across mathematical reasoning benchmarks. Lastly, in Chapter 8, motivated by the recent successes brought by self-supervised methods, we develop a set of self-supervision tasks to improve the representation learning of neural network models and hence alleviating the data scarcity problem. The self-supervised tasks are constructed by mining kernel-level theorem constructions of the ITP. We demonstrate the self-supervision learning greatly improves the proof success rates of learning models in Lean – where the formal data are the most scarce.

Chapter 2

Background

In this section, I will provide a background for the thesis by discussing two key areas of research: automated reasoning and neural networks. Automated reasoning involves the use of computer programs to automatically prove mathematical theorems and solve other complex logical problems. Automated reasoning has a wide range of applications in fields such as mathematics, computer science, and engineering. On the other hand, neural networks are a type of machine learning algorithm inspired by the structure and function of the human brain. Neural networks have been increasingly used in various applications, such as image and speech recognition, natural language processing, and decision-making. By exploring the intersection of these two fields, this thesis aims to investigate the potential of neural networks to enhance the performance and scalability of automated reasoning systems.

2.1 Automated reasoning

In this section, we would like to discuss automated reasoning, a subfield of AI concerned with developing algorithms and systems that can reason and make inferences automatically. The goal of automated reasoning is to create machines that can perform reasoning tasks typically associated with human intelligence, such as logical deduction, problem-solving, and decision-making. The field has its roots in mathematical logic and has been heavily influenced by developments in computer science and cognitive psychology.

The field of automated reasoning involves the use of computational tools and techniques to automate the process of reasoning and deduction. This field has a long history dating back to the development of the first automated theorem prover in the 1950s. Since then, a wide range of automated reasoning tools and techniques have been developed, including automated theorem provers (ATPs) and interactive theorem provers.

2.1.1 Automated theorem prover (ATP)

Automated theorem provers (ATPs) are computer programs designed to prove mathematical theorems automatically. The goal of an ATP is to find a proof of a given theorem that is both correct and concise. ATPs are widely used in various fields, including mathematics, computer science, and engineering, to verify the correctness of complex systems and algorithms.

ATPs typically work by translating the statement of the theorem to be proved into a logical language, such as first-order logic, and then using a combination of logical inference rules and search techniques to find a proof of the theorem. The process of proving a theorem involves constructing a sequence of logical deductions from a set of axioms and inference rules until the theorem is derived. The proofs generated by an ATP are typically in the form of a sequence of logical steps, with each step being justified by an inference rule or a previously derived theorem.

There are many different types of ATPs, each with its own strengths and weaknesses. Some ATPs are designed to work with specific domains, such as geometry or algebra, while others are general-purpose ATPs that can handle a wide range of mathematical problems. Two of the most widely used approaches are Boolean Satisfiability (SAT) solvers and Satisfiability Modulo Theories (SMT) solvers.

SAT solvers are algorithms that determine if a given Boolean formula can be satisfied by finding an assignment of truth values to the variables that make the formula true. SAT solvers are widely used in various fields, including hardware and software verification, planning, and scheduling. The basic approach of SAT solvers involves the use of a search algorithm to explore the space of possible variable assignments until a satisfying assignment is found or all possible assignments have been exhausted.

SMT solvers, on the other hand, extend the SAT solving approach to handle formulas that involve theories beyond propositional logic. These theories typically include arithmetic, bit-vectors, arrays, and other complex data structures. SMT solvers are widely used in formal verification of hardware and software systems, automated testing, and program synthesis. The basic approach of SMT solvers involves the use of a combination of SAT-solving and theory-specific reasoning to check the satisfiability of the input formula.

Both SAT and SMT solvers have been used in various applications, and have proven to be effective in solving a wide range of problems. However, they have their own strengths and weaknesses. For instance, SAT solvers tend to be faster and more efficient at handling large Boolean formulas, while SMT solvers are better suited for handling formulas that involve theories beyond propositional logic.

However, both SAT and SMT solvers are faced with significant challenges in scaling to larger and more complex problems. As the size and complexity of the problem increases, the search space for possible proofs grows exponentially. This makes it difficult to find a proof for large and complex theorems, and even the most powerful ATPs struggle to handle problems beyond a certain size and complexity.

2.1.2 Interactive theorem prover (ITP)

Interactive theorem provers (ITPs) are another type of software tool used for automated reasoning. Unlike ATPs, which are fully automated, ITPs require human interaction to guide the proof process. In an ITP, the user inputs a formal specification or statement of a theorem, and then manually constructs a proof by providing a series of logical steps or lemmas that are verified by the software.

The core of an ITP is a logical engine or kernel that provides a set of primitive inference rules for manipulating mathematical expressions and statements. Users interact with the ITP through a user interface or programming language, and specify the proof by providing a sequence of statements and commands that are checked and verified by the kernel.

To assist users in constructing the proof, ITPs often include a range of automated proof search

and synthesis tools. For example, an ITP may include a library of pre-proven lemmas, or automated tactics that can be used to discharge certain kinds of subgoals. ITPs may also include decision procedures or solvers that can automatically reason about certain mathematical concepts, such as linear arithmetic or boolean algebra.

Declarative vs. Tactic In interactive theorem proving (ITP), there are two main approaches to constructing proofs: declarative proof and tactic proof.

Declarative proof involves specifying the entire proof in advance using a high-level language or syntax, and then letting the ITP automatically verify the correctness of the proof. The user provides the ITP with a complete formal statement of the theorem or proposition, along with a proof script that specifies the sequence of logical steps required to establish the truth of the statement. The ITP then checks the proof script against the logical engine or kernel to verify its correctness, and outputs the result.

Tactic proof, on the other hand, involves incrementally constructing the proof using a series of smaller proof steps or tactics. The user provides the ITP with a high-level statement of the theorem or proposition, and then uses the ITP to generate a sequence of subgoals or smaller proof obligations that must be satisfied to establish the truth of the statement. The user then applies a series of proof tactics or commands to each subgoal to attempt to derive a proof. The ITP then checks each tactic or command for correctness and generates new subgoals until all subgoals have been satisfied.

Notable examples

Isabelle The Isabelle theorem prover is a popular interactive theorem proving (ITP) system used for formalizing and verifying mathematical proofs. It was developed in the 1980s by a team of researchers at the University of Cambridge led by Lawrence C. Paulson, and has since been maintained and extended by a large community of developers and users. Isabelle is based on the Higher Order Logic (HOL) logical framework, which allows users to reason about complex mathematical structures such as sets, functions, and types. Isabelle provides a rich library of pre-defined logical concepts and inference rules, as well as a powerful tactic language for constructing proofs interactively. Isabelle also supports a range of automated reasoning tools, including decision procedures and simplifiers, which can be used to discharge simpler subgoals automatically. Isabelle is widely used in academia and industry for formal verification of software and hardware systems, as well as for research in areas such as mathematics, computer science, and artificial intelligence. Its popularity and flexibility make it a powerful tool for conducting formal verification in a wide range of applications.

Lean Lean is an interactive theorem prover and functional programming language [83]. It has an extremely active community and is host to some of the most sophisticated formalized mathematics in the world, including scheme theory [17], forcing [50], perfectoid spaces [16], and condensed mathematics [109]. Lean's foundational logic is based on a dependent type theory called the calculus of inductive constructions [95]. This design means that terms, types and proofs are all represented with a single datatype called an *expression*. A *proof term* is a Lean expression whose type is a proposition, *i.e.* a theorem. This proof term serves as a checkable artifact for verifying the proposition. Lean uses a small, trusted kernel to verify proof terms. The primary repository of formalized mathematics in Lean is `mathlib` [21]. At the time of writing, 140 contributors have contributed almost 500,000 lines

of code; `mathlib` contains over 46,000 formalized lemmas backed by over 21,000 definitions,covering topics such as algebraic geometry, computability, measure theory, category theory. The range of theories and the monolithic, unified organization of `mathlib` makes it an excellent foundation for a neural theorem proving dataset.

2.1.3 ITP vs ATP

One of the main advantages of ITPs over ATPs is their ability to handle more complex and abstract mathematical concepts. ITPs typically provide a more flexible and expressive language for specifying mathematical statements, and can support a wider range of proof techniques than ATPs. Additionally, ITPs can be used to prove more general statements, such as type systems and program correctness, that are difficult or impossible to express in a formal language suitable for ATPs.

Another advantage of ITPs is their ability to provide high levels of assurance and confidence in the correctness of the proofs produced. Because ITPs require human guidance and intervention, errors and mistakes are less likely to go unnoticed. Furthermore, ITPs often include features such as proof assistants and proof checkers that can help users detect errors and inconsistencies in their proofs.

However, the reliance on human intervention and guidance also makes ITPs more time-consuming and labor-intensive than ATPs. Constructing a proof in an ITP can require a significant amount of effort and expertise, and may be impractical for larger and more complex problems.

2.2 Neural Networks

2.2.1 Overview

Neural networks have revolutionized the field of artificial intelligence over the past decades. They have been applied to a wide range of tasks, from perception problems such as image classification and object recognition, to strategic games such as chess and Go, and to natural language processing and machine translation. Recent advances in neural network architectures and training algorithms have led to remarkable improvements in performance across a variety of applications. For example, AlphaGo [114], a neural network-based program developed by DeepMind, became the first computer program to beat a human world champion at the ancient Chinese game of Go in 2016. More recently, we also witnessed emergent abilities from large language models [15], such as the ability to generate coherent and contextually appropriate text, to perform language translation with near-human accuracy, and to even answer questions and perform basic reasoning tasks. These breakthroughs have demonstrated that neural networks are not only capable of perception tasks, but can also be extremely useful in complex reasoning and planning tasks.

Architecture Neural networks are a type of machine learning algorithm inspired by the structure and function of the human brain. They consist of interconnected nodes, called neurons, arranged in layers. Each neuron receives inputs from other neurons, applies a mathematical function to those inputs, and then sends an output to other neurons in the next layer. The strength of the connections between neurons is determined by the weights assigned to each connection, which are adjusted during training to optimize the performance of the network on a given task. The architecture of a neural

network refers to the specific arrangement of these layers and neurons. Common architectures include feedforward neural networks, where information flows in one direction through a series of layers; convolutional neural networks, which are specialized for processing images; and recurrent neural networks, which are designed to handle sequential data such as text or speech. The choice of architecture depends on the specific task at hand and the nature of the input data. Different architectures can have varying levels of complexity, which can impact their ability to accurately model and classify data. The design of neural network architecture is an active area of research and innovation, with new architectures being developed and tested all the time.

Learning Neural networks learn by adjusting the strengths of the connections between neurons, known as weights, to better predict the desired output for a given input. This process is known as training, and it involves presenting the network with a set of input/output pairs, known as a training set. The network then makes a prediction for each input and compares it to the desired output, calculating a loss function that measures the difference between the predicted and actual values. The goal of training is to minimize this loss function by adjusting the weights, typically using a method called backpropagation. During backpropagation, the error signal is propagated backwards through the network, and the weights are updated in proportion to their contribution to the error. This process is repeated over many iterations, or epochs, until the network's predictions on a separate validation set start to plateau or improve only minimally. At this point, the network has learned a mapping from inputs to outputs that generalizes well to new, unseen data. The choice of loss function, optimization method, and hyperparameters such as the learning rate and batch size can all affect the performance of the network and the speed at which it converges during training.

2.2.2 Architectures

Transformers Transformers are a type of neural network architecture that was first introduced in 2017 by [123]. They are designed to process sequential data, such as text or time series, and have been particularly successful in natural language processing (NLP) tasks such as language translation, question answering, and text summarization. In more recent years, the architecture was also adopted for computer vision tasks [29].

The key innovation of the transformer architecture is the use of self-attention mechanisms to compute contextualized representations of the input sequence. Self-attention allows the network to attend to different parts of the input sequence at different times, enabling it to better capture long-range dependencies and relationships between the elements of the sequence. In a transformer network, the input sequence is first embedded into a high-dimensional vector space, and then processed through a series of identical layers, each of which applies a multi-head self-attention mechanism followed by a position-wise feedforward network. Layer normalization and residual connections are used to help stabilize training and improve performance.

Transformers have several advantages over earlier sequence modeling approaches such as recurrent neural networks (RNNs). Unlike RNNs, which are difficult to parallelize and can suffer from vanishing or exploding gradients, transformers can be easily parallelized and do not suffer from these issues.

GNNs Graph neural networks (GNNs) are a type of neural network architecture that is designed to process and model graph-structured data, where the data is represented as a network of interconnected nodes and edges. Graphs are a natural way to represent many real-world phenomena, such as social networks, protein structures, and computer networks.

The key idea behind GNNs is to learn a representation for each node in the graph by aggregating information from its neighboring nodes and edges. This is done through a series of message passing steps, where each node updates its representation based on the representations of its neighbors. This allows the network to capture the local structure and connectivity of the graph. As a result, they have shown to be effective in a wide range of graph-related tasks, such as node classification, link prediction, and graph classification.

2.3 Machine learning for mathematical reasoning

While automated theorem proving has long been a major component of classical AI, among the first applications of machine learning to interactive and automated theorem proving was lemma selection [62, 57]. This includes lemma selection in hammers [13, 58] which use naive Bayes and other ML algorithms to filter relevant lemmas to send to an SMT solver.

More recently, progress has been made in directly applying machine learning to prove theorems [102, 34, 111, 71]. The logic and mathematics in these works is usually more geared to logical syllogisms or industrial problems (*e.g.* logical search in a relational database or SAT solving).

There are also machine learning based provers for tactic-based ITPs, including TacticToe [40] for HOL4; HOList/DeepHOL [5, 4, 90] for HOL Light; and CoqGym/ASTactic [143], ProverBot9001 [105] and Tactician [11] for Coq. These works, similar to ours, use learned models which suggest tactics for a given tactic state, and when combined with a search algorithm, are able to build complete proofs.

However, these past works cannot freely synthesize complete tactics, including tactic combinators and tactics with expression parameters. The models limit the tactic grammar to only produce tactics with simple parameters (*e.g.* theorem names) or select tactics exactly as they appear in the dataset (possibly with small modifications to variable names). Such approaches unnecessarily constrain the model, especially in situations where the user needs to supply an existential witness (in the form of an expression) to the tactic.

Metamath is another interactive theorem prover which does not use tactics, but instead relies on a low-level proving framework. Neural provers for Metamath, such as Holophrasm [133], MetaGen [125], and GPT-f [97] all use sequence or language models to generate tactics. GPT-f, presented in [97], is a seminar work that employs a Transformer architecture and sequence-to-sequence framework to generate proof steps. This approach offers increased flexibility and diversity in generating proof steps, since the sequence-to-sequence framework allows GPT-f to generate the complete tactic step, unrestricted by a fixed set of predefined actions.

Besides theorem proving, a number of recent papers have shown that language models, especially Transformers, are capable of performing tasks akin to mathematical and logical reasoning in areas like integration [69], differential equations [19], Boolean satisfiability [36], and inferring missing proof steps [73].

Part I

Evaluations

Chapter 3

Synthetic Benchmark: INT

In this and the next chapter, we created evaluation benchmarks for machine learning models to prove theorems. We first introduce a synthetic benchmark, INT, a lightweight inequality theorem proving benchmark serving as the "MNIST" of theorem proving which also allows for controlled out-of-distribution evaluation. We then discuss two realistic benchmarks in the next chapter, based on realistic theorem proving assistant that can be used to prove most of university-level mathematics as well as some of the frontiers of mathematics research.

This chapter is largely based on the work: *INT: An Inequality Benchmark for Evaluating Generalization in Theorem Proving* by Yuhuai Wu*, Albert Jiang*, Jimmy Ba, and Roger Grosse [138], published in ICLR of 2021. Being the lead of this work, I contributed to the creation of main project ideas, including the focus on out-of-distribution generalization, theorem generation algorithms, developments of GNN, transformer baselines and their training, and MCTS generalization experiments. I strongly thank Albert Jiang's contribution for developing the codebase of theorem generation algorithms, besides his other contributions in experimental results, analysis and paper writing.

3.1 Overview

Well-designed synthetic datasets have been shown to help understand the capabilities of machine learning models [60, 103, 132]. In this chapter, we introduce a synthetic benchmark – INT, for evaluating neural networks for theorem proving. INT is an INequality Theorem proving benchmark designed for evaluating generalization. It can generate a theoretically unlimited number of theorems and proofs in the domain of algebraic equalities and inequalities.

There are two main motivations behind this work. Firstly, we want to build a benchmark that serves as the "MNIST" [70] for the theorem proving challenge. Namely, we would like to offer a cheap-to-evaluate and simple-to-iterate environment to the community for preliminary research investigations. Secondly, the assumption frequently made in machine learning that each data point is identically and independently distributed does not hold in general for theorem proving: interesting problems we want to prove are non-trivially different from those we have proofs for. Hence it is important to benchmark the ability to generalize to a new distribution of theorems.

Most existing proof assistants require a large software library to define numerous mathematical

theorems, leading to slow simulation. Taking advantage of the limited scope of inequality theorems, we load a minimal library and achieve fast simulation. Reducing the simulation overhead allows for experimentation with planning methods such as MCTS which requires many calls to a simulator. The inequality theorems generated are also less challenging than realistic theorems, hence allowing one to obtain results faster with less computational costs. Moreover, due to INT's simiplicty and its homogeneous codebase (purely in Python), there is much less overhead when it comes to setting up the benchmark.

We also develop a synthetic inequality theorem generator, which allows INT to tweak its problem distribution along 6 dimensions, enabling us to probe multiple aspects of out-of-distribution generalization.

We introduce and benchmark several baseline agents for the six types of generalization tasks in INT. We find that transformer-based agents' generalization abilities are superior when training and test data are drawn from the same distribution and inferior in out-of-distribution tasks in INT, compared to GNN-based agents. Surprisingly, despite larger generalization gaps, transformer-based agents have favorable test success rates over GNN-based ones in most cases. We find that searching with MCTS at test time greatly improves generalization.

Last but not the least, common reservation to hold for synthetic datasets is one of realism: can synthetic data help to prove realistic theorems? Polu and Sutskever [97] adopted our generation method and showed that augmentation of 1% of synthetic theorems in training helped to complete 2.3% more proofs on Metamath [82]. This demonstrates the usefulness of INT in real mathematics.

3.2 Related Works

It is worth mentioning that there have been a few approaches [121, 126] on neural theorem synthesizers, while none of them are as flexible and comprehensive as our generator in controlling theorem distribution. Our theorem generator INT is designed to be capable of creating an infinite number of theorems, as well as benchmarking the generalization ability of learning-assisted theorem provers.

3.3 The INT Benchmark Dataset and Proof Assistant

Our INT benchmark dataset provides mathematical theorems and a means to study the generalization capability of theorem provers. For this purpose, we need control over the distribution of theorems: this is achieved by a highly customizable synthetic theorem generator. We used a set of ordered field axioms [30] to generate inequality theorems and a subset of it to generate equality theorems. Details of the axiomization schemes can be found in Table 3.1. The code for generating theorems and conducting experiments is available at https://github.com/albertqjiang/INT.

3.3.1 Terminology

The axiom combination of a proof refers to the set of axioms used in constructing it. The sequence of axioms applied in order in the proof is called the axiom order. For example, let A, B, C denote three unique axioms, and their order of application in a proof be $[B, B, A, C]$. In this case, the axiom combination is the set $\{A, B, C\}$ and the axiom order is the sequence $[B, B, A, C]$. An initial

condition is a (usually trivial) logic statement (e.g. $a = a$) to initiate the theorem generation process. The degree of an expression is the number of arithmetic operators used to construct it. For example, $\text{degree}(a) = 0$ while $\text{degree}(((a * c) * b)^2) = 3$.

3.3.2 INT Assistant

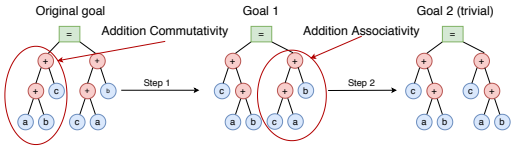
```
example (a b c : ℕ) :
(a + b) + c = (c + a) + b :=
begin
  rw add_comm (a + b);
  -- goal ->
  -- c + (a + b) = (c + a) + b
  rw add_assoc c
  -- goal ->
  -- c + (a + b) = c + (a + b)

  -- Proof closed!
end
```

(a) Lean, `rw` stands for `rewrite`

Objective: ∅ to (a + b) + c = (c + a) + b
Action: add_comm ((a + b) + c) in obj
→
Objective: ∅ to c + (a + b) = (c + a) + b
Action: add_assc ((c + a) + b) in obj
→
Objective: ∅ to (c + a) + b = (c + a) + b

Proof closed!

(b) INT, seq2seq interface



(c) INT, graph interface

Figure 3.1: A proof of $a + b + c = c + a + b$ in Lean and INT, with seq2seq and graph interfaces.

We built a lightweight proof assistant to interact with theorem provers. It has two interfaces, providing theorem provers with sequential and graph representations of the proof state, respectively.

Field axioms	Definition
AdditionCommutativity (`AC`)	$\to a + b = b + a$
AdditionAssociativity (`AA`)	$\to a + (b + c) = (a + b) + c$
AdditionSimplification (`AS`)	$a = b \to a + (-b) = 0$
MultiplicatoinCommutativity (`MC`)	$\to a \cdot b = b \cdot a$
MultiplicationAssociativity (`MA`)	$\to a \cdot (b \cdot c) = (a \cdot b) \cdot c$
MultiplicationSimplification (`MS`)	$(a \neq 0) \wedge (a = b) \to 1 = a \cdot \frac{1}{b}$
AdditionMultiplicationLeftDistribution (`AMLD`)	$\to (a + b) \cdot c = a \cdot c + b \cdot c$
AdditionMultiplicationRightDistribution (`AMRD`)	$\to a \cdot (b + c) = a \cdot b + a \cdot c$
SquareDefinition (`SD`)	$\to a^2 = a \cdot a$
MultiplicationOne (`MO`)	$\to a \cdot 1 = a$
AdditionZero (`AZ`)	$\to a + 0 = a$
PrincipleOfEquality (`POE`)	$(a = b) \wedge (c = d) \to a + c = b + d$
EquMoveTerm(Helper axiom) (`EMT`)	$a + b = c \to a = c + (-b)$
Ordered field axioms	Definition
All field axioms	
SquareGEQZero (`SGEQZ`)	$a = b \to a \cdot b \geq 0$

EquivalenceImpliesDoubleInequality (`EIDI`)	$a = b \to (a \geq b) \wedge (a \leq b)$
IneqMoveTerm (`IMT`)	$a + b \geq c \to a \geq c + (-b)$
FirstPrincipleOfInequality (`FPOI`)	$(a \geq b) \wedge (c \geq d) \to a + c \geq b + d$
SecondPrincipleOfInequality (`SPOI`)	$(a \geq b) \wedge (c \geq 0) \to a \cdot c \geq b \cdot c$

Table 3.1: Axioms used in generating INT.

Algorithm 1 Theorem Generator

function GENERATE_THEOREM(initial conditions $\mathcal{I}$, axiom order A)
  Axiom order length $L = \text{len}(A)$.
  Initialize core logic statement $C_0 \sim Uniform(\mathcal{I})$, and the set of premises $P = \{C_0\}$.
  for $t \leftarrow 1$ to L **do**
    Get axiom $a_t \leftarrow A[t]$.
    Get new logic statement and premises: C_t, $P_t \leftarrow$ MORPH (a_t, C_{t-1}).
    Add new premises to the set of all premises: $P \leftarrow P \cup P_t$.
  end for
  return C_L, P
end function

A problem in INT is represented by a goal and a set of premises (e.g. $a + 0 = a, \varnothing$), which are mathematical propositions. The INT assistant maintains a proof state composed of the goal and the proven facts. The proof state is initialized to be just the goal and premises of the theorem. A proof is a sequence of axiom-arguments tuples (e.g. $[(\text{AdditionZero}, [a + 0])]$). At each step of the proof, a tuple is used to produce a logical relation in the form of assumptions $\rightarrow$ conclusions (e.g. $\varnothing \rightarrow a + 0 = a$). Then, if the assumptions are in the proven facts, the conclusions are added to the proven facts; if the conclusions include the goal, the unproven assumptions will become the new goal. The assistant considers the theorem proven, if after all steps in the proof are applied, the goal is empty or trivial.

In Figure 3.1, we present the same proof in Lean [83] and INT assistants. They both process proofs by simplifying the goal until it is trivial. The INT assistant's seq2seq interface (Figure 3.1b) is very similar to that of Lean (Figure **??**) with the `rewrite` tactic. An action is composed of an axiom followed by argument names and their positions in the proof state. `in obj` indicates that the arguments can be found in the objective. The graph interface (Figure 3.1c) of the INT assistant allows theorem provers to chose axiom arguments from the computation graphs of the proof state by node. We can view theorem proving with this interface as a graph manipulation task.

INT assistant provides fast simulation. To demonstrate this, we produced 10,000 typical proof steps in both interfaces, 40-character-long on average. We executed them with HOL Light [52] and INT assistant. The average time it takes per step is 7.96ms in HOL Light and 1.28ms in INT, resulting in a 6.2× speedup. The correctness of the proofs is ensured by a trusted core of fewer than 200 lines of code.

3.3.3 Theorem Generator

One of the main contributions of this paper is to provide a generation algorithm that is able to produce a distribution of non-trivial synthetic theorems given an axiom order. Generating theorems by randomly sampling axiom and argument applications will often yield theorems with short proofs. Instead, we write production rules for axioms in the form of transformation and extension rules. With these production rules, we can find arguments and new premises required for longer proofs.

We provide the theorem generation algorithm in Algorithm 1. The general idea of the algorithm is to morph a trivial logic statement into one that requires a non-trivial proof; we call this statement the core logic statement. We initiate the core logic statement C_0 to be one of the initial conditions. At step t of the generation process, we are given an axiom a_t specified by the axiom order. We

apply the MORPH function associated with the axiom a_t to C_{t-1} and derive a new logic statement C_t and corresponding premises P_t. The key design idea in the MORPH function is to ensure that the newly generated logic statement and the premises form the implication $C_{t-1}, a_t, P_t \to C_t$ (see A.1 for details).

Therefore, we can chain the implications from all steps together to obtain a proof whose length is the axiom order: $C_0, \{a_t, P_t\}_{t=1}^{L} \to C_L$, where L denotes the length. The last core logic statement C_L and its premises $C_0, \{P_t\}_{t=1}^{L}$ are returned as the theorem generated. Below we show a step-by-step example of how a theorem is generated with our algorithm.

A worked example

Use Algorithm 1 to generate a theorem with initial conditions $\mathcal{I}$: $\{a = a, b = b, c = c, d = d, e = e\}$ and axiom order A: [AdditionAssociativity (`AA`), AdditionCommutativity (`AC`), EquivalenceImpliesDoubleInequality (`EIDI`), FirstPrincipleOfInequality (`FPI`)].

Core logic statement $C_0 \sim Uniform(\mathcal{I}) : a = a$.
Step 1: $a_1 =$ `AA`. C_1: $a + (b + c) = (a + b) + c$, $P_1 = \varnothing$.
Step 2: $a_2 =$ `AC`. C_2: $a + (b + c) = (b + a) + c$, $P_2 = \varnothing$.
Step 3: $a_3 =$ `EIDI`. C_3: $a + (b + c) \geq (b + a) + c$, $P_3 = \varnothing$.
Step 4: $a_4 =$ `FPI`. C_4: $(a + (b + c)) + d \geq ((b + a) + c) + e$, $P_4 = \{d \geq e\}$.

Theorem generated: Given $d \geq e$, prove $a + (b + c) + d \geq b + a + c + e$.

With recorded axiom and argument applications, we can synthesize proofs to the theorems. The proofs can be used for behavior cloning. We show generated theorem examples in Appendix A.4.

3.3.4 Dataset Statistics

Theorem Length

We compare the length of the theorems generated in characters and plot their distributions in Figure 3.2. The length of the theorem in characters is a measure for how complicated it is. As is expected, the more complicated the theorem is, the longer the proof(bigger L). It is also worth noting that as L becomes bigger, the distribution of theorem length becomes less concentrated. This is likely a consequence of a more spread-out theorem length range.

Axiom Distributions

The frequency at which each axiom is applied influences the distribution of theorems our generator is able to produce. In Figure 3.3, we present the proportions of axioms that are applied in generating 10,000 theorems. Their frequencies are a measure of how easy it is to satisfy the conditions to apply them. For the field axioms, the PrincipleOfEquality axiom is the most frequently used(9.30%) and the EquMoveTerm axiom is the most rarely used(2.38%). EquMoveTerm has a strict condition for application: the left hand side of the core logic statement has to be of the form $x + y$, therefore not frequently applied. For the ordered-field axioms, the EquivalenceImpliesDoubleInequality axiom is the most frequently used(10.18%). Since we start with a trivial equality in generation and want to end up with an inequality, a transition from equality to inequality is needed. Among the ways of

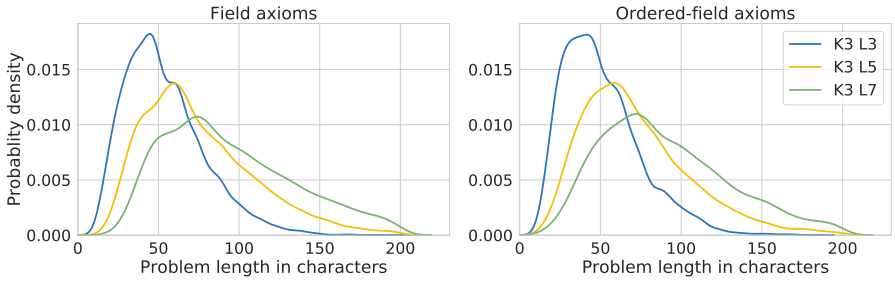
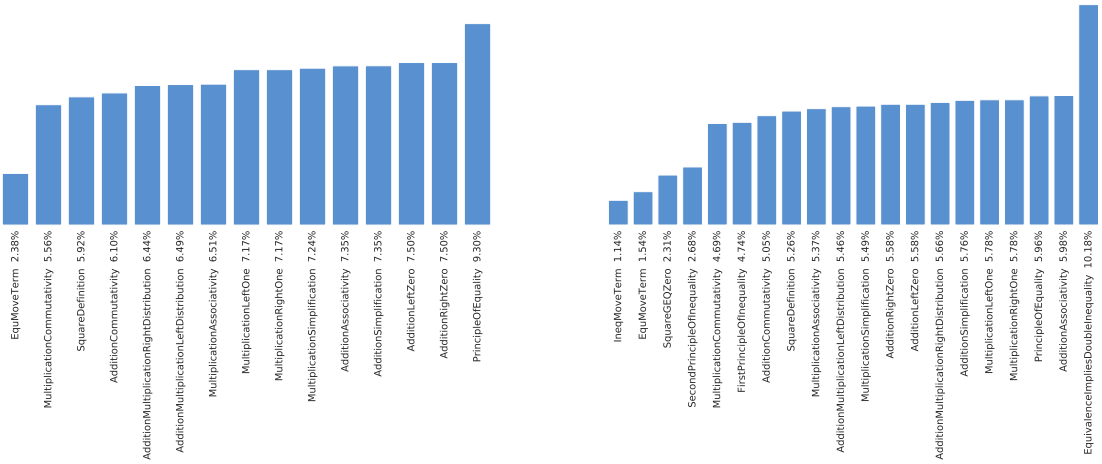


Figure 3.2: The distribution of theorem length in characters for field axioms(left) and ordered-field axioms(right) generated with parameters $K3L3$, $K3L5$, and $K3L7$. As the length of the proof is increased, so is the number of characters in the theorem, while the distribution of latter is less concentrated.



(a) Field axiom distribution

(b) Ordered-field axiom distribution

Figure 3.3

transitioning, this conditions to apply this axiom is easiest to satisfy. Its popularity is followed by the group of Field axioms, from MultiplicationCommutativity(4.69%) to AdditionAssociativity(5.98%). The rest are ordered-field axioms which define the properties of inequalities, proportions ranging from IneqMoveTerm(1.14%) to FirstPrincipleOfInequality(5.74%).

3.4 Experiments

Our experiments are intended to answer the following questions:

1. Can neural agents generalize to theorems: 1) sampled from the same distribution as training data, 2) with different initial conditions, 3) with unseen axiom orders, 4) with unseen axiom combinations, 5) with different numbers of unique axioms, 6) with shorter or longer proofs?

2. How do different architectures (transformer vs. GNN) affect theorem provers' in-distribution and out-of-distribution generalization?

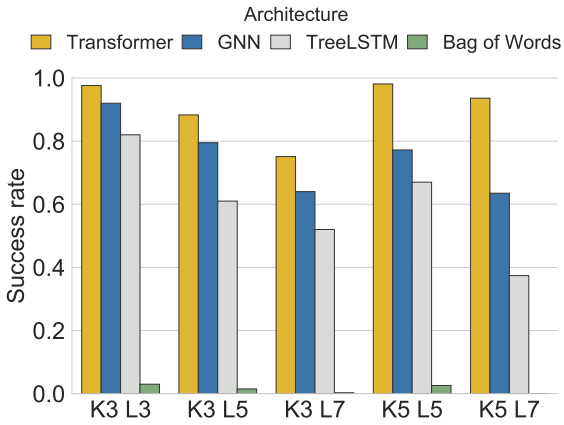


Figure 3.4: Proof success rates on test problems generated with K and L settings. Transformer and GNN perform well; TreeLSTM has mediocre performance; and Bag-of-Words performs poorly: it cannot prove more than 5% of problems.

3. Can search at test time help generalization?

3.4.1 Experiment Details

In the following experiments, we used the proofs generated by the INT generator to perform behavior cloning. We then evaluated the success rates of trained agents in a theorem proving environment. We denote the cardinality of an axiom combination as K and the length of a proof as L. In the worked example, $K = 4$ and $L = 4$. For each theorem distribution, we first generated a fixed test set of 1000 problems, and then produced training problems in an online fashion, while making sure the training problems were different from the test ones. For each experiment, we generated 1000 problems and performed 10 epochs of training before generating the next 1000. We ran 1500 such iterations in total, with 1.5 million problems generated. We used the Adam optimizer [66]. We searched over the learning rates $\{10^{-5}, 3 \cdot 10^{-5}, 10^{-4}, 3 \cdot 10^{-4}\}$ in preliminary experiments and found 10^{-4} to be the best choice, which was used for following experiments. We used one Nvidia P100 or Tesla T4 GPU with 4 CPU cores for training. For each experiment, we ran 2 random seeds, and picked the one with higher validation success rates for test evaluation. Since this paper focuses on inequalities, all figures and tables in the main text are based on results from the ordered-field axiomization.

3.4.2 Network Architectures

In this section, we introduce four baselines built on commonly used architectures: Transformers [123], Graph Neural Networks (GNNs), TreeLSTMs [120] and Bag-of-Words (BoWs). In preliminary experiments, we found Graph Isomorphism Networks (GINs) [141] to have performed the best among several representative GNN architectures. So we used GIN as our GNN of choice. Transformers interact with the INT proof assistant through the seq2seq interface while the other baselines through the graph interface.

For sequence-to-sequence training, we used a character-level transformer architecture with 6 encoding layers and 6 decoding layers. We used 512 embedding dimensions, 8 attention heads and 2048 hidden dimensions for position-wise feed-forward layers. We used dropout with rate 0.1, label

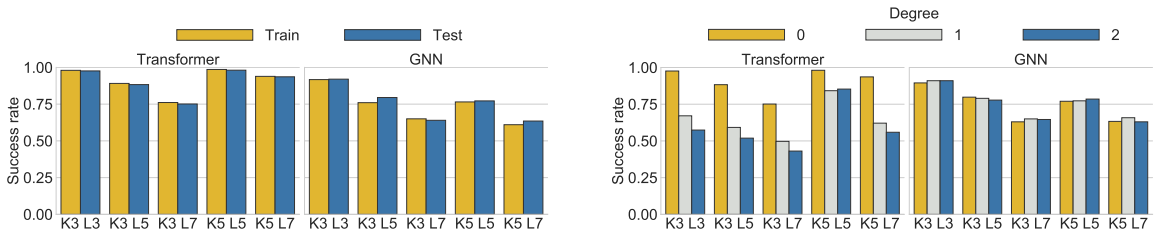


Figure 3.5: Proof success rates on problems generated with different K and L parameters. **Left:** When the IID assumption holds, the success rate decreases as the two generation parameters K and L are increased. **Right:** All agents are trained on degree-0 problems and evaluated against problems of degree 0, 1, and 2. We find that transformer-based agents deteriorate in performance as the test problems become more complex than training problems. For GNN-based agents, there are no obvious trends as to how the proof success rate changes as the degree of the initial entities is varied.

smoothing with coefficient 0.1, and a maximum 2048 tokens per batch. The library fairseq [89] was used for its implementation.

For data in the graph form, each node in computation graphs corresponds to a character in the formula. We first used a learnable word embedding of dimension 512 to represent each node. We then used 6 GIN layers to encode graph inputs into vector representations, each with 512 hidden dimensions. The graph representation was obtained by taking the sum of all the node embeddings. For the TreeLSTM and the BoW baselines, we used a bidirectional TreeLSTM with 512 hidden dimensions and a BoW architecture to compute the graph representation vectors from node embeddings. The hyper-parameters used were found to be optimal in preliminary experiments. We then proposed axioms conditioned on the graph representations, with a two-layer MLP of hidden dimension 256. Conditioning on the graph representation and axiom prediction, the arguments are selected in an autoregressive fashion. Namely, the prediction of the next node is conditioned on the previous ones. For each argument prediction, we used a one-layer MLP with a hidden size of 256. We used graph neural network libraries Pytorch Geometric [35] for the GIN implementation, and DGL [128] for the TreeLSTM implementation.

We trained agents based on architectures mentioned above by behavior cloning on theorems of various length (L) and number of axioms (K). The success rates for proving 1000 test theorems are plotted in Figure 3.4. As the BoW architecture did not utilize the structure of the state, it failed miserably at proving theorems, indicating the significance of the structural information. TreeLSTM performed worse than the graph neural network baseline. The transformer and the GNN baselines perform the best among the architectures chosen and they take inputs in sequential and graph forms, respectively. Thus, we used these two architectures in the following experiments to investigate generalization.

3.4.3 Benchmarking Six Dimensions of Generalization

IID Generalization In this experiment, the training and test data are independently and identically distributed (IID). The performances of our transformer-based and GNN-based agents are displayed on the left in Figure 3.5. As can be seen, the performance of agents examined on train and test problems are very similar. The largest difference between train and test success rates is 2% ($K3L7$). Notably, transformer-based agents complete 15.3% more test proofs than GNN-based

Table 3.2: **Top:** Proof success rates (in %) of agents trained on different numbers of axiom orders. **Bottom:** Proof success rates (in %) of agents trained on different numbers of axiom combinations. K denotes the cardinality of the axiom combination of a proof, L denotes the length of the proof.

Architecture	Axiom orders	100		500		2000		5000	
		Train	Test	Train	Test	Train	Test	Train	Test
Transformer	K3 L3	98.4	32.6	99.5	90.0	98.8	98.7	97.6	97.6
	K3 L5	95.3	6.3	94.0	56.3	94.0	94.9	96.5	94.9
	K3 L7	87.8	3.8	88.0	46.4	88.3	77.5	88.4	85.5
	K5 L5	94.7	5.6	97.0	72.9	97.4	93.1	97.5	96.9
	K5 L7	89.7	1.8	88.6	48.6	89.3	75.2	88.6	84.0
	Average	93.2	10.0	93.4	62.8	93.6	87.9	93.7	91.8
GNN	K3 L3	84.3	38.6	94.4	73.9	93.7	89.0	90.5	92.3
	K3 L5	92.7	17.1	86.3	60.0	84.4	72.9	77.7	77.1
	K3 L7	82.4	14.1	82.4	33.8	68.6	57.7	70.2	63.5
	K5 L5	91.0	23.0	89.7	61.2	81.8	75.0	78.3	80.8
	K5 L7	87.5	12.9	80.2	39.0	66.5	57.4	61.6	60.0
	Average	87.6	21.1	86.6	53.6	79.0	70.4	75.7	74.7

Architecture	Axiom combos	25		100		200		300	
		Train	Test	Train	Test	Train	Test	Train	Test
Transformer	K3 L3	99.2	34.1	99.0	72.8	99.5	96.1	98.6	98.2
	K3 L5	97.8	29.3	98.6	66.3	97.5	89.5	94.3	90.4
	K3 L7	93.6	25.0	91.9	55.9	91.5	80.0	91.9	85.9
	K5 L5	98.5	27.4	98.4	87.6	97.0	93.6	97.3	94.9
	K5 L7	91.2	30.5	92.2	76.3	91.7	82.9	90.0	87.0
	Average	96.1	29.3	96.0	71.8	95.4	88.4	94.4	91.3
GNN	K3 L3	96.3	61.6	96.0	90.1	92.7	91.2	95.3	92.0
	K3 L5	82.1	43.4	80.3	68.9	78.5	74.9	76.5	76.1
	K3 L7	72.1	34.3	68.1	57.2	62.3	63.7	62.5	62.0
	K5 L5	77.8	61.6	78.9	71.0	74.5	78.4	72.8	74.9
	K5 L7	67.2	36.8	59.7	52.7	54.9	54.0	56.7	54.5
	Average	79.1	47.5	76.6	68.0	72.6	72.4	72.8	71.9

agents on average.

Initial Condition Consider two theorems: (1) $(a+b)^2 = a^2 + b^2 + 2ab$ and (2) $(a+(b+c))^2 = a^2 + (b+c)^2 + 2a(b+c)$. The two problems take the same axioms and the same number of steps to prove. However, the axiom argument complexities are different, which can be seen as a result of varying initial conditions. Can agents trained on problems like (1) prove theorems like (2)?

For an initial condition of the form $X = X$, we use the degree of the entity X to determine the complexity. In this experiment, we trained agents on problems with initial conditions made up of entities of degree 0, and evaluated them on ones of degrees 1 and 2. The results are presented in Figure 3.5 (b) with various K and L. For transformer-based agents, the success rate drops 25.6% on degree-1 problems and 31.5% on degree-2 problems on average. However, for GNN-based agents, the largest generalization gap between training and test success rates is 3% ($K3L5$). This shows that GNN agents can generalize to problems of higher complexities while transformer agents struggle.

Axiom Orders Let A and B represent two different axioms. There are multiple orders in which they can be applied in a $K2L3$ problem. $O_1 = [A, A, B]$ and $O_2 = [B, A, B]$ are two examples.

Can an agent trained on problems generated with O_1 prove theorems generated with O_2?

For both architectures, we investigated how well agents can generalize to problems with different axiom orders than those in training. We generated 100, 500, 2000, and 5000 axiom orders to use in the training set for different K and L settings. We evaluated the test success rates on 1000 unseen axiom orders with the corresponding K and L settings and averaged them. The results averaged over different K and L settings are shown on the top of Table 3.2.

It can be observed in the table that the test success rates rise when we increase the number of axiom orders in the training set. We notice that transformer-based agents have worse generalization than GNN-based ones, as their average generalization gap is larger. This is particularly true when the number of axiom orders in the training set is 100: transformer-based agents can prove only 10.0% of test theorems. Remarkably, they still manage to complete more proofs than GNNs when the number of axiom orders in the training set exceeds 500.

Axiom Combinations Consider three problems provable in the ordered field axiomization (3.1): (1) $a^2 \geq 0$, (2) $a * (b + c) = b * a + a * c$, and (3) $a^2 + b^2 - 2ab \geq 0$. Solving (1) requires axiom SquareGEQZero (`SGEQZ`). Solving (2) requires axiom AdditionMultiplicationDistribution (`AMD`) and axiom MultiplicationCommutativity (`MC`). Solving (3) requires axiom `SGEQZ` and axiom `AMD`. Notice that all axioms used to prove (3) appear in the proofs of (1) and (2). We ask: can an agent trained on theorems like (1) and (2) prove theorems like (3)?

In this set of experiments, we investigated how well theorem provers can generalize to problems with different axiom combinations than those in training for both architectures. We used 25, 100, 200, and 300 axiom combinations to generate the training set with various K and L settings, and evaluated the agents on test sets generated with 300 unseen combinations. The results averaged over different K and L settings are displayed on the right in Table 3.2. As the number of axiom combinations in training set increases, the generalization gap decreases and test success rate improves. The transformer-based agents have larger generalization gaps than GNN-based ones. This is particularly obvious when there are 25 axiom combinations: the generalization gap is 66.8% for transformers and 31.6% for GNNs. The test success rate of transformers is 18.2% lower than that of GNNs in this setting. Yet when there are more than 100 axiom combinations in training, transformers always perform better on the test sets, completing $3.8\% - 19.6\%$ more proofs. When the data is diverse, transformers perform better; when it is insufficient, GNNs are better. This might be due to the difference in the inductive bias used by both structures and might explain the choice of neural architectures in deep learning practice.

Number of Axioms Here we investigated how well theorem provers could generalize to test problems that were generated with a different number of axioms than at training time. For instance, let A, B and C represent different axioms. Will agents trained on $K2L3$ axiom orders like $[A, B, A]$ and $[C, C, B]$ be able to prove theorems generated with $K3L3$ axiom orders like $[A, B, C]$?

We trained the agents on problems that have the same proof length ($L = 7$) and varying Ks. The results are on the left of Figure 3.6. It can be observed from the figure that in general, agents perform the best on the K they were trained on and worse when K shifts away. Transformer-based agents showed better performances in all K and L settings, completing 20.9% more proofs than GNN-based ones on average. The success rates of transformer-based agents drop 5.6% on average when the test K is shifted away by 1 from the training K. For GNN-based agents, this averages to

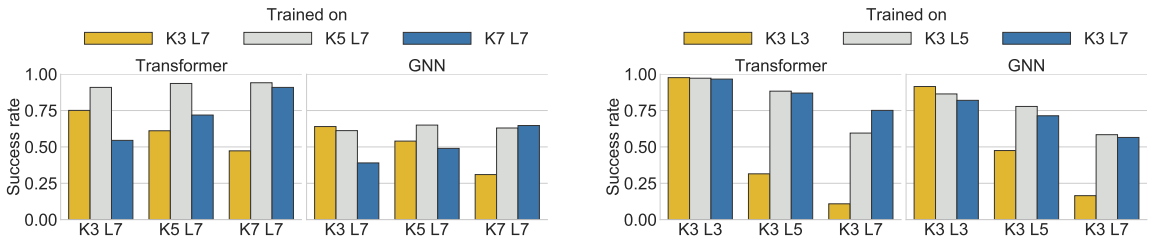


Figure 3.6: Proof success rates on problems generated with different parameters. **Left:** We keep L the same and vary K. The success rate is likely to decrease when the test problems have different K from the training problems. **Right:** We keep K the same and vary L. For all agents, the proof success rate is lower on theorems that require longer proofs.

5.1%. This shows that their generalization abilities to different number of axioms are similar.

Proof Length We tested the generalization ability of theorem provers over the dimension of proof length of the theorems. To do this, we kept the cardinality of the axiom set to be the same ($K = 3$) and varied the evaluated problems' proof length ($L = 3, 5, 7$). The result is presented on the right of Figure 3.6. For all of the agents trained, the success rate decreases as the length of the proof increases. This is due to the natural difficulty of completing longer proofs. Observing the figure, we see that the longer the training problems, the less they deteriorate in performance when proofs becomes longer: agents trained on $K3L3$ problems complete 18.8% fewer proofs when L is increased by 1, while ones trained on $K3L7$ complete 5.7% fewer. Furthermore, the performance of transformer-based agents decreases by 12.2% when the test proof length increases by 1, compared to 10.7% for GNN-based ones. This suggests that transformers have inferior proof length generalization abilities than GNNs.

3.4.4 Generalizing with Search

We investigated whether performing search at test time can help agents generalize. Specifically, we investigated the effectiveness of Monte-Carlo Tree Search (MCTS) in finding proofs for unseen theorems with GNN-based agents. We chose GNN-based agents because they are better at out-of-distribution generalization than transformer-based ones. Straightforward application of MCTS is impractical: in our theorem proving environment, the action space can be as large as 1.3M in size (see 1.2.1). Hence, it would be infeasible to expand all possible actions when constructing the MCTS trees. Thus, we only performed MCTS over the axiom space (18 distinct axioms in total), and the arguments were proposed by the behavior cloning agents. Following AlphaGo Zero/AlphaZero [114, 113], we trained a value network to estimate the value of a state. The value network is an MLP with two hidden layers of size 256, taking the GNN global representations of graphs as input. It was trained on 1000 episodes of rollouts obtained by the behavior cloning agents, with a learning rate of $3 \cdot 10^{-6}$. We also followed AlphaZero for the choice of the upper confidence bound, and the way that actions are proposed using visit counts. We used 200 simulations for constructing MCTS trees. Following [114], in the selection step of the MCTS tree construction, we use the following formula to select the next action,

$$a^* = \text{argmax}_a \left(Q(s,a) + c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right),$$

Table 3.3: The behavior cloning (BC) agents versus the MCTS-assisted (search) agents. **Left:** The average success rates (in %) of agents with and without MCTS over 1000 test theorems. **Right:** The average length of successful proofs by agents with and without MCTS over 1000 test theorems. K denotes the cardinality of the axiom combination of a proof, L denotes the length of the proof.

Train	K3L3		K3L5		K3L7	
Evaluation	BC	Search	BC	Search	BC	Search
K3 L3	92	**98**	91	97	81	96
K3 L5	50	64	80	**92**	70	**92**
K3 L7	25	40	64	78	58	**81**
Average	56	67	78	89	69	**90**

Train	K3L3		K3L5		K3L7	
Evaluation	BC	Search	BC	Search	BC	Search
K3 L3	3.83	**3.33**	4.00	3.52	5.00	3.67
K3 L5	7.54	6.82	6.2	**5.52**	6.84	5.56
K3 L7	9.05	8.54	8.01	7.53	8.39	**7.50**
Average	6.81	6.23	6.07	**5.52**	6.74	5.58

where $Q(s,a)$ represents the action value function, $N(s,a)$ denotes the visit counts, $P(s,a)$ is the prior probability, and c_{puct} is a constant hyperparameter. In all of our experiments, we used the behavior cloning policy for computing $P(s,a)$, and we used $c_{\text{puct}} = 1$. After the MCTS tree is built, the action is sampled from the policy distribution $\pi(a|s) = N(s,a)^{\frac{1}{\tau}}$, where τ is a hyperparameter and was chosen as 1 in our experiments.

We took the agents trained on "$K3L3$", "$K3L5$", and "$K3L7$" from section 3.4.3, and evaluated the agents' performance when boosted by MCTS.

Generalization The average success rates on 1000 test theorems are presented on the left in Table 3.3. We can see that search greatly improved the generalization results. It helped to solve 21% more problems on average for the agent trained on theorem distribution $K3L7$. Remarkably, when evaluating on $K3L7$ theorems, search helped the $K3L3$ agent improve its success rate from 25% to 40%: a relative improvement of 60%. It is interesting to see the $K3L7$ behavior cloning agent solved 9% fewer problems on average than the $K3L5$ agent. But search brought about much larger improvement to the $K3L7$ agent and helped it to solve the largest proportion of problems on average – 90%. This indicates that skills learned through behavior cloning can be better exploited by searching.

The average proof length for 1000 problems is presented on the right in Table 3.3 (we count those unsolved problem as 15, the step limit of an episode). We can see that by performing search, we are able to discover proofs of length closer to the ground truth proof length. For test theorems requiring 3-step proofs, the $K3L3$ agent was able to prove them in 3.33 steps on average, with a gap of 0.33 steps to the optimal value. Similarly, for test theorems requiring 5-step proofs, the $K3L5$ agent was able to prove them in 5.52 steps on average, with a gap of 0.52 steps; and for theorems requiring 7-step proofs, $K3L7$ agent achieved a gap of 0.5 steps.

3.4.5 Discussion

Experimental results suggested that transformer-based agents can complete more proofs in the IID generalization scenario but have larger out-of-distribution generalization gaps than GNN-based ones. The larger gap may be due to the lack of constraints in the sequence-to-sequence framework, in which the model can propose sequences that are invalid actions, whereas the graph interface constrains the model to propose valid actions only. However, we still see that transformers are able to complete more proofs overall. This shows the superiority of transformers in model capacity when applied to theorem proving. This insight motivates us to explore the possibility of taking the best from both worlds, combining both graph structural information and the strong transformer architecture to improve learning-assisted theorem proving. We leave it for future work.

3.5 Conclusion

We addressed the problem of diagnosing the generalization weaknesses in learning-assisted theorem provers. We constructed INT, a synthetic benchmark of inequalities, to analyze the generalization of machine learning methods. We evaluated transformer-based and GNN-based agents and a variation of GNN-based agents with MCTS at test time. Experiments revealed that transformer-based agents generalize better when the IID assumption holds while GNN-based agents generalize better in out-of-distribution scenarios. We also showed that search can boost the generalization ability of agents. We stress that proving theorems in INT is not an end in itself. A hard-coded expert system might perform well on INT but not generalize to real-world mathematical theorems. Therefore, INT should be treated as *instrumental* when diagnosing generalization of agents. The best practice is to use INT in conjunction with real mathematical datasets.

We believe our benchmark can also be of interest to the learning community, facilitating research in studying generalization beyond the IID assumption. The agents' abilities to reason and to go beyond the IID assumption are essential in theorem proving, and studying how to acquire these abilities is at the frontier of learning research. In other domains requiring out-of-distribution generalization, such as making novel dialogs [20] or confronting unseen opponents in Starcraft [124], the requirements for data and computation forbid a generally affordable research environment. The INT benchmark provides practical means of studying out-of-distribution generalization.

Chapter 4

Realistic Benchmarks: Isabelle and Lean

In this section we build two realistic theorem proving benchmarks with two interactive theorem provers: Isabelle and Lean. We choose Isabelle because it has the largest corpus of formalized proofs – arXiv of formal proofs [1], and also because of its prevalent proof style – declarative proofs which are more akin to high-level mathematical reasoning. We also choose Lean because it is the youngest and also the most active interactive theorem provers, being host to some of the most sophisticated formalized mathematics in the world, including scheme theory [17], forcing [50], perfectoid spaces [16], and condensed mathematics [109]. The reason we build more than one realistic benchmark is to encourage building generic techniques can be used with various ITPs. But this also poses one critical issue: can we compare models across benchmarks? A recent effort called MiniF2F[1] aims to resolve this issue by creating a common set of problems formalized across various ITPs, and will be discussed as well in the following chapter.

In the following, we present the environment and dataset for both Isabelle and Lean. The process of building a realistic benchmark consists of mainly two problems: 1. building an reinforcement learning environment that seamlessly interfaces machine learning models and the ITP, 2. mining existing formal human proofs written in the ITP.

This chapter is largely based on one published manuscript and one workshop publication: 1. *LISA: Language models of ISAbelle proofs* by Albert Jiang*, Wenda Li, Jesse Michael Han, and Yuhuai Wu* [59], and 2. *Proof Artifact Co-training for Theorem Proving with Language Models* by Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, Stanislas Polu [51] published in the *Mathematical Reasoning in General Artificial Intelligence Workshop* (MATHAI) at ICLR of 2021. I am the co-lead for the first manuscript, leading the main ideas, discussion, code and paper writings. I am a contributor to the second manuscript, where I contributed to the training of the first neural network models on the mined dataset, developing the Fairseq baseline (including training and evaluation), providing technical guidance on the use of machine learning models, developing proof artifact datasets (proof terms and next lemma), and suggesting mixed training over all mined data which leads to the idea of self-supervised co-training.

[1] https://github.com/openai/miniF2F

	Source length				Target length			
	min	max	mean	median	min	max	mean	median
With proof states only	7	227831	379.6	187.0	2	6522	34.2	19.0
With previous steps only	17	138581	3223.6	980.0				
With both proof states and previous steps	60	229885	3612.2	1328.2				

Table 4.1: Sequence length in characters

4.1 Isabelle

4.1.1 Environment and Dataset

We created an environment where theorem proving is modelled as a sequential decision process. Initially, the environment will load a selected theorem and we have access to the top level state. At each time-step, the agent produces a proof step of arbitrary length. The environment then applies the proof step to the top level state and iterates the process if the theorem has not been proved. We show the proof process of a simple theorem in Figure 4.1. The theorem declaration initialises the first proof state. The proof states in the middle row represent the stage of the proof progress and the proof steps in the bottom row are what the agent should produce. We support three different kinds of inputs: with proof states only, with previous steps only, and with both proof states and previous steps. For example, the previous steps when the agent should produce "done" consist of "apply (rule impI)" and "apply assumption". Because the Isabelle proof assistant provides a Partially Observable Markov Decision Process (POMDP) with the proof states being the observation, conditioning on the previous steps of the proof helps the agent to reconstruct the true state of the proof.

With this environment, we mined a total of 183K theorems from the Isabelle standard library [87] and the Archive of Formal Proofs (AFP) [1]. We then extracted a total of 2.16 million pairs of inputs and proof steps. This forms a dataset useful for theorem proving: if an agent can produce the correct proof step when prompted with an arbitrary proof state, it will be able to prove the theorem. We then used a 95%/1%/4% random split to divide the proof corpus into the training/validation/test sets. We show some dataset statistics in Table 4.1.

4.1.2 Experiments

Setup We use decoder-only Transformers similar to GPT-3 [15]. All models have 163M non-embedding parameters. We use the same BPE encoding as GPT-3 [15]. Similarly to the GPT-f models applied to Metamath [97] and Lean [51], our model is pre-trained on the WebMath dataset for 72B tokens. When finetuning on Isabelle proofsteps, we used a batch size of 2048, a learning rate of 0.005, a 100-step ramp-up, and decayed the learning rate according to a cosine schedule over 64B tokens; we early-stopped according to validation perplexity after 35B elapsed tokens. We only used the AFP part of the dataset for training and evaluation, due to time constraints.

Evaluation We used a best-first search strategy at evaluation, similar to that of [97]. We initialise and maintain a priority queue of top level states, sorted by their cumulative log probability. The cumulative log probability of a top level state is the sum of log probabilities of all the previous proof steps the agent takes to arrive at the current state. Initially, the priority queue contains only the top level state right after the declaration of the theorem, with a cumulative log probability of 0.

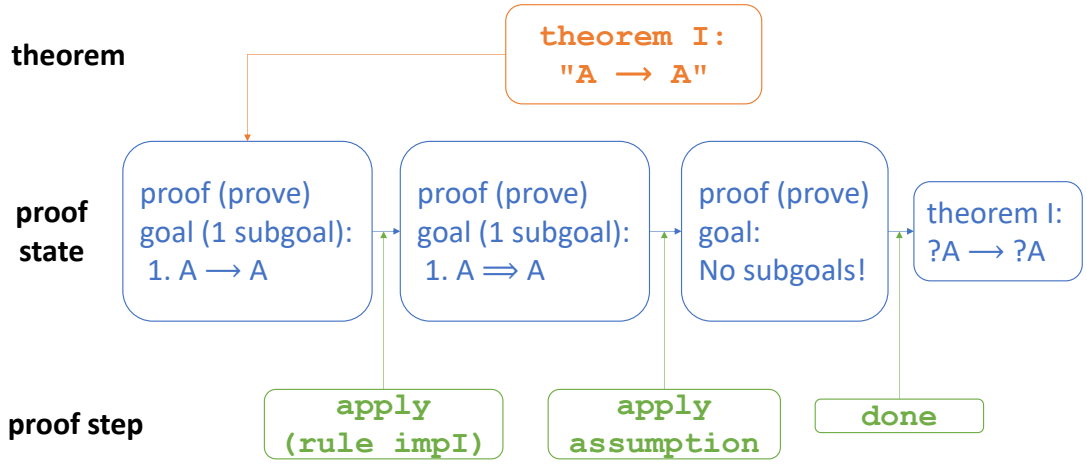


Figure 4.1: An illustration of the relationship between theorems, proof states, and proof steps.

At each search step, we pop the head of the priority queue to retrieve the top level state with the highest probability. We then query the language model for a set of 16 proof step candidates, with a temperature of 1.0. For each of the candidates, we duplicate the top level state, apply the candidate to it, and calculate the updated cumulative log probability. If the application of the candidate is successful, we add the resulted top level state to the queue. The queue has a length of 16 (i.e. it only maintains 16 entries with the highest cumulative log probabilities). If one of the resulted top level state shows that the proof is complete, we consider the proof attempt successful. If the queue is empty, or a timeout of 120s is spent on one attempt, or the number of queries exceeds 100, we consider the attempt a failure.

4.1.3 Results

We evaluated our language model with the best-first search strategy on a test set of 4000 theorems. 33.2% of the theorems were successfully proved. We analysed the failure causes of the rest of the theorems. 59.1% of the attempts failed because of the time limitation, 0.2% of the attempts failed because of the query number limitation and 7.6% of the attempts failed because the priority queue was empty at some point in the proving process. We show two successful proofs generated by our language model, and contrast them with the proofs in the AFP.

Theorem 1 is a lemma in *Utility.thy* from the AFP entry *Executable Matrix Operations on Matrices of Arbitrary Dimensions* [116]. Our proof is a one-liner and much simpler than the original proof. We checked the validity of several generated proofs manually by writing them in Isabelle with the same dependency as the original proofs.

Theorem 1 lemma foldr_foldr_concat:
"foldr (foldr f) m a = foldr f (concat m) a"
Original proof
proof (induct m arbitrary: a)
case Nil show ?case by simp
next
case (Cons v m a)
show ?case
unfolding concat.simps foldr_Cons o_def Cons
unfolding foldr_append by simp
qed
Our proof
by (induct m arbitrary: a) simp_all

Theorem 2 is a lemma in *Word_Lemmas.thy* from the AFP entry *Finite Machine Word Library* [7]. Although our proof is longer than the original, it utilises a different set of lemmas to finish the proof, and is written in a very different style compared to the original. This demonstrates that our proof search agent with language models is capable of discovering novel and interesting proofs.

Theorem 2 lemma scast_ucast_1:
"⟦ is_down (ucast :: 'a word ⇒ 'b word);
is_down (ucast :: 'b word ⇒ 'c word) ⟧ ⟹
(scast (ucast (a :: 'a::len word) :: 'b::len word) :: 'c::len word) = ucast a"
Original proof
by (metis down_cast_same ucast_eq ucast_down_wi)
Our proof
using unat_ucast
apply -
apply (simp add:ucast_def unat_ucast)+
apply (subst down_cast_same[symmetric])
apply (simp add: is_down)+
apply (rule word_eqI)
apply (simp add: nth_ucast)
apply safe
apply simp
done

As a baseline, we also considered using greedy search. This is equivalent to best-first search with the queue length = 1. This agent, as a consequence, only proved 28.3% of the theorems.

4.2 Lean

4.2.1 Human tactic proof steps

Tactics in Lean are metaprograms [31], which can construct Lean expressions, such as proof terms. A *tactic state* which tracks the partial proof term constructed so far, the list of open goals, and other metadata is threaded through each tactic invocation. Lean has special support for treating tactics as an extensible domain-specific language (DSL); this DSL is how Lean is typically used as an interactive theorem prover. The DSL amounts to a linear chain of comma-separated invocations. The Lean *proof step* task is to predict the next tactic given this goal state.

Our human tactic proof step dataset consists of source-target pairs of strings, one for each tactic invocation in the Lean core library and in `mathlib`. The source string is the pretty-printed tactic state. The target string is the tactic invocation as entered by a human author of the source code. This data is gathered by hooking into the Lean parser and Lean's compilation process. We refer to the task of predicting the next human tactic proof step given a tactic state as the *proofstep objective*.

4.2.2 The LeanStep machine learning environment

We instrument Lean for automatic theorem proving with a language model, including utilities for (1) setting the runtime environment at a particular theorem (ensuring proofs are never circular), (2) serializing the tactic state as environment observations for a theorem-proving agent, (3) exposing Lean's parser to re-parse strings emitted by a language model into tactic invocations, and (4) executing and capturing the results of the re-parsed tactics, enabling the recording of trajectories for expert iteration and reinforcement learning.

In addition to this general instrumentation, we implement a generic best-first search algorithm for theorem proving; it forms the basis for our evaluations and is written entirely in Lean itself. The algorithm is parametrized by an oracle (`Ω : tactic_state → list (string × float)`) that accepts a tactic state and returns a list of strings and heuristic scores. The search is controlled by a priority queue of *search nodes*, which consist of a tactic state (*i.e.* a partial proof) and search metadata. In the outer loop of the algorithm—which continues until either the theorem is completely proved (*i.e.* no goals are remaining on the current node), the priority queue is empty (*i.e.* the search has failed), or a pre-set timeout or budget of iterations is exceeded—we pop a node off the queue, serialize the associated tactic state and use it query the oracle, producing a list of candidates `cs : list (string × float)`. We then loop over the candidates `cs` to produce a list of new search nodes, by re-parsing each string into a tactic and adding a new node if the parsed tactic advances the proof without raising errors. These new search nodes are then re-inserted into the queue in order of decreasing priority and the search continues. We optionally constrain the search by enforcing maximum width and depth limits $w_{\max}$ and $d_{\max}$ that guard insertion into the queue. When considering nodes for insertion, any node whose depth exceeds $d_{\max}$ is ignored, and all nodes are ignored if the queue size is strictly larger than $w_{\max}$. Due to the flexibility in assigning heuristic scores and in choosing the maximum width and depth hyperparameters, our algorithm is quite general—for example, it reduces to (1) a greedy depth-first search when $w_{\max} = 0$, and (2) a naïve breadth-first search when heuristic scores are identical and $w_{\max} = d_{\max} = \infty$.

4.2.3 Experiments

Setup and evaluation The setup an evaluation follows the same protocol as for Isabelle.

4.2.4 Baseline description

The `tidy` backend is determined by a constant oracle

```
Ω : tactic_state → list (string × float)
```

which always returns the same list of tactics, namely:

```
meta def tidy_default_tactics : list (string × float) :=
list.map (flip prod.mk 0.0) [
   "refl"
,  "exact dec_trivial"
,  "assumption"
,  "tactic.intros1"
,  "tactic.auto_cases"
,  "apply_auto_param"
,  "dsimp at *"
,  "simp at *"
,  "ext1"
,  "fsplit"
,  "injections_and_clear"
,  "solve_by_elim"
,  "norm_cast"
]
```

Unlike the `gptf` backend, which generates a list of candidates in parallel independently, `tidy` enjoys the advantage that the list of tactics it emits is carefully chosen and ordered in order to optimize the proof search—this is based on the "waterfall" technique of the human-style automated theorem prover described in ([37]).

Results We evaluated our language model with the best-first search (BFS) strategy on a test set of 3071 theorems. 32.2% of the theorems were successfully proved. On the other hand, the `tidy` oracle with BFS only managed to solve 9.9% of the theorems.

4.3 Discussion

We extracted a large corpus from Isabelle proofs and examined the performance of language models in proving theorems on the dataset. We showed that a non-trivial portion of problems on AFP can be solved by the application of a language model and a best-first search. The successful proofs demonstrated the language model's ability to compose succinct, or novel proofs.

The proof assistant Isabelle provides a very convenient command that allows users to conjecture ("have"). With our environment that interacts with the proof assistant in a very flexible manner, and our rich dataset, we can set out to further study how machines can learn to conjecture, and to reason about the proof construction more generally.

On the other hand, Lean is a popular interactive theorem prover with a large user base in the mathematics community. It has been used to verify the correctness of many important mathematical results, and attracting more and more mathematicians recently. As such, it has played a critical role in advancing the field of formal mathematics. The evaluation harness we provide can serve as a valuable resource for researchers seeking to develop new techniques and tools for Lean, which in turn can have a direct impact on the practice of formal mathematics.

Part II

Capabilities

Chapter 5

Abduction: IsarStep

In this and the next chapter, we probe neural networks' abilities to do mathematical reasoning. In particular, we focused on the ability to do abduction and induction, two powerful abilities that mathematicians employ frequently in their mathematical practices. Abduction refers to the ability to conjecture a possible explanation of the existing phenomenon. In mathematics, besides making interesting mathematical conjectures, one also usually employ abduction during a proof search. Knowing that statement A is true and B is what we want to prove, what is a reasonable intermediate step that bridges A to B? By employing such reasoning, one then builds a series of high-level intermediate propositions, leading to a proof sketch. In this chapter, we delve into this capability by designing a high-level mathematical reasoning task based on a large human mathematical corpus. We demonstrate neural networks' nontrivial abilities to perform abduction on this task.

This chapter is largely based on the work: *IsarStep: a Benchmark for High-level Mathematical Reasoning* by Wenda Li, Lei Yu, Yuhuai Wu and Lawrence Paulson [73], published in ICLR of 2021. I am a contributor to this work, and my contribution includes early project discussions that lay the basis of the work, machine learning experimental designs, and paper writing.

5.1 Overview

When humans prove a theorem, a crucial step is to propose an intermediate proposition to bridge the gap between the goal and the currently known facts. This step requires complicated reasoning capabilities such as creative thinking, inference, understanding existing conditions, and symbolic manipulation of rules. For example, consider the following proof of the irrationality of $\sqrt{2}$:

Proof of irrationality of $\sqrt{2}$. Assume $\sqrt{2}$ is rational. Then there exists a pair of coprime integers a and b such that $\sqrt{2} = a/b$, and it follows that $2 = a^2/b^2$ and then $2b^2 = a^2$. Hence a is even. Thus there exists an integer c such that $a = 2c$, which combined with $2b^2 = a^2$ yields $2c^2 = b^2$: hence b is also even. So a and b are both even although they are coprime, contradiction. □

To derive $\exists c \in \mathbb{Z}.\, a = 2c$ from $2b^2 = a^2$, the intermediate proposition "a is even" would reduce the gap and lead to a successful proof. We would like to simulate the way humans prove theorems by proposing an intermediate proposition synthesis task — IsarStep. Instead of having primitive steps like $3+5=8$, the proof steps in IsarStep are at a higher-level, with much bigger steps as basic.

```
theorem "sqrt 2 ∉ ℚ"
proof
  assume "sqrt 2 ∈ ℚ"
  then obtain a b :: int where "sqrt 2 = a/b" "coprime a b"
    by (metis Rat_cases Rats_def imageE normalize_stable of_rat_divide
          of_rat_of_int_eq quotient_of_Fract quotient_of_div)
  then have "2 = a² / b²" by (smt of_int_power power_divide real_sqrt_pow2)
  then have *:"2*b² = a²"
    by (cases "b=0",auto simp:field_simps,use of_int_eq_iff in fastforce)
  then have "even a" by (metis dvd_triv_left even_mult_iff power2_eq_square)
  then obtain c::int where "a=2*c" by blast
  with * have "b² = 2*c²" by simp
  then have "even b" by (metis dvd_triv_left even_mult_iff power2_eq_square)
  with ‹even a› ‹coprime a b› show False by auto
qed
```

Figure 5.1: Full declarative proof the irrationality of $\sqrt{2}$ in Isabelle/HOL.

Therefore it usually cannot be simply solved by pattern matching and rewriting. To succeed in this task, a model is required to learn the meaning of important mathematical concepts (e.g. determinant in linear algebra, residue in complex analysis), how they are related to each other through theorems, and how they are utilised in proof derivations. Solving the IsaStep task will be potentially helpful for improving the automation of theorem provers, because proposing a valid intermediate proposition will help reduce their search space significantly. It is also a first step towards the long-term goal of sketching complete human-readable proofs automatically.

We have built the IsarStep dataset by mining arguably the largest publicly-hosted repository of mechanised proofs: the Achieve of Formal Proofs (AFP).[1] The AFP is checked by the Isabelle proof assistant [93] and contains 143K lemmas. Combining the AFP with the standard library of Isabelle/HOL yields a dataset of 204K formally-proved lemmas. The dataset covers a broad spectrum of subjects, including foundational logic (e.g. Gödel's incompleteness theorems), advanced analysis (e.g. the Prime Number Theorem), computer algebra, cryptographic frameworks, and various data structures. A nice property of the mined formal proofs is that they are mostly *declarative proofs*, a proof style very close to human prose proofs.[2] Fig. 5.1 illustrates the proof of irrationality of $\sqrt{2}$ in Isabelle. We can see that the proof is actually legible (even to people who are not familiar with the system) and it captures high-level structures like those in human proofs.

We further explore the reasoning capabilities of neural models. We frame the proposed task as a sequence-to-sequence (seq2seq) prediction problem. Beyond evaluating the existing neural seq2seq model baselines—the seq2seq with attention [3], the transformer [123]—we also propose a new architecture, the hierarchical transformer (Section 5.4). The architecture is motivated by the way humans reason about propositions; it consists of a set of local transformer layers, modelling the representation of each proposition, and a set of global layers, modelling the correlation across propositions. Experiments (Section 5.5) show that these neural models can solve 15–25% of problems on the test set, and the hierarchical transformer achieves the best result. Further analysis (Section 5.7) on the output of these models shows that while the proposition synthesis task is hard, the neural models can indeed capture mathematical reasoning. We find that the embeddings of closely related mathematical concepts are close in cosine space; models can reason about the relation between set, subset, and member, and perform more complex multi-step reasoning that is even hard for humans.

[1] https://www.isa-afp.org

[2] A comparison of proofs in different systems is available in [135]. The declarative style proof is also available in Mizar [46], where the style originates.

Our contributions are summarised as follows:

1. We mine a large non-synthetic dataset of formal proofs and propose a task for evaluating neural models' mathematical reasoning abilities. The dataset contains 820K training examples with a vocabulary size of 30K.
2. We evaluate existing neural seq2seq models on this task.
3. We introduce a hierarchical transformer model, which outperforms the baseline models.
4. We provide a comprehensive analysis of what has been learned by the neural models.
5. We provide a test suite to check the correctness of types and the validity of the generated propositions using automatic theorem provers.

5.2 The IsarStep Task

In this section, we define the task of intermediate proposition generation more concretely. We again take the proof of irrationality of $\sqrt{2}$ as an example. We will have the following derivation:

$$\underbrace{2b^2 = a^2}_{(1)} \Rightarrow \underbrace{a \text{ is even}}_{(2)} \Rightarrow \underbrace{\exists c \in \mathbb{Z}.\, a = 2c}_{(3)}.$$

In our proposed task, we would like to generate (2) given (1) and (3). When humans prove a theorem, they implicitly assume certain background knowledge, as lemmas. For example, in this case we assume that we can trivially prove (1) $\Rightarrow$ (2) based on the fact that the product of two numbers are even iff at least one of them is even. In Isabelle [93], these relevant lemmas (e.g. `even_mult_iff: even (?a * ?b) = (even ?a ∨ even ?b)` corresponding to line 10 in Fig. 5.1) can be found automatically by its built-in automation Sledgehammer [12]. In our task, we optionally provide these lemmas as extra information in addition to (1) and (3).

The derivation of (2) $\Rightarrow$ (3) in the proof above is a simple step, because only (2) is needed to arrive at (3). In most cases, multiple propositions have to be used together in order to infer a proposition, for example $P_1, P_2, P_3 \Rightarrow P_4$. For these more general cases, we also include the additional propositions (e.g. P_2 and P_1) as part of the source propositions.

To summarize, each example in the IsarStep dataset is formed by five parts:

F.1 a target proposition (e.g. a is even),

F.2 a set of used local propositions to derive **F.1** (e.g. $2b^2 = a^2$),

F.3 a local proposition derived from the target proposition **F.1** ($\exists c \in \mathbb{Z}.\, a = 2c$),

F.4 other local propositions and (library) lemmas used to justify **F.3**,

F.5 a set of used (library) lemmas to justify **F.1** (e.g. `even_mult_iff: even (?a * ?b) = (even ?a ∨ even ?b)`).

We want to synthesise **F.1** given **F.2** – **F.4** with **F.5** optional: the named lemmas in **F.5** are common knowledge and can be used as additional hints. The propositions are generated as a sequence of

tokens and therefore the search space is Σ^*: search over 30K actions (Section 5.3.3, vocabulary size for seq2seq models) at every timestep without a predefined maximum output length.

IsarStep can be considered as single step reasoning, which can be repeated to sketch more complex proofs. Good performance on this task is a crucial step for designing models that can automatically prove theorems with minimal human assistance.

5.3 Dataset Preprocesssing and Statistics

The mined raw dataset has long propositions and a large number of unique tokens. To alleviate the performance deterioration of machine learning models due to the aforementioned problems, we propose tricks to preprocess the raw dataset, including free variable normalisation and removing unnecessary parentheses. These tricks substantially reduce the sequence lengths and vocabulary size.

5.3.1 The Logic and Tokens

The core logic of Isabelle/HOL is simply-typed λ-calculus with de Bruijn indices for bound variables [131, Chapter 2.2]. A local proposition or a (library) lemma/theorem is essentially a *term* in the calculus. As types can be inferred automatically, we drop types in terms (to reduce the size of the vocabulary) and encode a term as a sequence of tokens that include lambda term constructors: `CONST`, `FREE`, `VAR`, `BOUND`, `ABS` (function abstraction), and `$` (function application). Additionally, parentheses have been used in the sequence to represent the tree structure. To give an example, we encode the proposition *even* `a` as the following sequence of tokens separated by a white space:

```
CONST HOL.Trueprop $ (CONST Parity.semiring_parity_class.even $ FREE <X0>)
```

where `CONST HOL.Trueprop` is a boilerplate function that converts from type *bool* to *prop*; `CONST Parity.semiring_pari` is the *even* predicate; `FREE <X0>` encodes the Skolem constant `a` in *even* `a`. Since `a` is a user-introduced local constant that can be arbitrary, we normalised it to the algorithmically generated name `<X0>` in order to reduce the vocabulary size (see Section 5.3.2).

Overall, every local proposition and library lemma/theorem is encoded as a sequence of tokens, and can be mostly decoded to the original term with type inference.

5.3.2 Free Variable Normalisation

Due to Isabelle's use of de Bruijn indices, bound variables have already been normalised: $\forall x.\ P\ x$ is no different from $\forall y.\ P\ y$, as both x and y are encoded as `BOUND 0`. However, arbitrary variable names can be introduced by the command **fix** in declarative proofs or unbounded variables in lemma statements (e.g. $False \Longrightarrow P$ and $False \Longrightarrow Q$ are semantically equivalent but with different free variables). To reduce the vocabulary size here, we normalised these free variables like the bound ones. For example, $False \Longrightarrow P$ would be normalised to $False \Longrightarrow$ `<V0>` as P is the first free variable in the proposition. Such normalisation reduced the vocabulary size by *one third*. The normalisation preserves the semantics, and we can always parse a normalised term back under a proper context.

5.3.3 Statistics

We have mined a total of 1.2M data points for IsarStep. We removed examples in which the length of the concatenation of the source propositions, i.e. **F.2** – **F.4** in Section 5.2, longer than 800 and the length of the target propositions, i.e. **F.1** in Section 5.2, longer than 200, which results in approximately 860K examples. From these examples we randomly sampled 10K examples for validation and test. In the training data, we removed duplicates, the examples whose target propositions exist in the held-out set, and those that are from the same theorems as the propositions in the held-out set. The final dataset split is 820K, 5000, 5000 for the training, validation, and test sets, respectively. The vocabulary size is 29,759.

5.4 Model

We define $\boldsymbol{X} = [\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^I]$ as the sequence of I source propositions, and $\boldsymbol{y} = (y_1, y_2, \ldots, y_N)$ as the target proposition containing N tokens. Let $\boldsymbol{x}^i = (x_1^i, x_2^i, \ldots, x_M^i)$ represent the ith proposition in the set, consisting of M tokens. Each source proposition $\boldsymbol{x}^i$ belongs to a category **F.2** – **F.4** defined in Section 5.2. We annotate the category corresponding to $\boldsymbol{x}^i$ as $\mathcal{C}_i$ and therefore the sequence of categories corresponding to $\boldsymbol{X}$ is $\boldsymbol{\mathcal{C}} = [\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_I]$. The generation of a target proposition $\boldsymbol{y}$ is determined by finding the proposition $\hat{\boldsymbol{y}}$, where $p(\hat{\boldsymbol{y}} \mid \boldsymbol{X}, \boldsymbol{\mathcal{C}})$ is optimal,

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\mathcal{C}}). \tag{5.1}$$

We propose two approaches to parameterising the conditional probability $p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\mathcal{C}})$, which differ in the way of modelling the sequence of source propositions. The first method is simply appending a label to each source proposition indicating their category and then concatenating the source propositions using a special token `<SEP>`, treating the resulting long sequence as the input to a seq2seq model.

Our second approach models the encoding of source propositions hierarchically. As shown in Fig. 5.2, the encoder has two types of layers. The local layers build the proposition representations by modelling the correlations of tokens within each proposition; the global layers take the proposition representations as input and model the correlations across propositions. Both the local layers and global layers are transformer layers [123]. Positional information is encoded separately for different source propositions. That is, suppose $\boldsymbol{x}^1$ has M tokens, then the position of the first token in $\boldsymbol{x}^2$ is not $M + 1$ but 1. The embedding of a token x_m^i is obtained by adding the token embedding, the positional information, and the embedding of the category that the proposition $\boldsymbol{x}^i$ belongs to. The category embedding is learnt together with the rest of the network. We call this model the *hierarchical transformer* (HAT). Intuitively, HAT models the structure of the source propositions more explicitly compared to the first approach and therefore should be better at capturing reasoning between source and target propositions. We will validate our hypothesis in Section 5.5.

5.5 Experiments

We benchmark four models on IsarStep (Section 5.2), namely the seq2seq model with attention (RNNSearch) [3, 136], the convolutional seq2seq models (Conv2Conv) [41], transformer [123], and

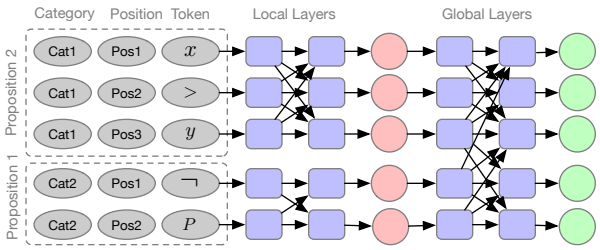


Figure 5.2: Architecture of the encoder of the hierarchical transformer (HAT). There are two types of layers, the local layers model the correlation between tokens within a proposition, and the global layers model the correlation between propositions. The input to the network is the sum of the token embedding, the positional information, and the embedding of the corresponding category.

hierarchical transformer (HAT). The input to the RNNSearch and the transformer is a concatenation of source propositions (the first parameterisation approach described in Section 5.4). We train these models with the same training data and report their performance on test sets.

5.5.1 Experimental Setup

For RNNSearch[3] [3, 136], we used 2-layer LSTMs [55] with 512 hidden units and 0.2 dropout rate. For Conv2Conv[4], we used the setup of `fconv_iwslt_de_en` to train the model. The hyperparameters for training the transformer[5] were the same as *transformer base* [123], i.e. 512 hidden size, 2048 filter size, 8 attention heads, and 6 layers for both the encoder and decoder. The hyperparameters for HAT were the same, except that the number of local context layers was 4 and global context layers is 2. We shared the source and target token embeddings for all the three models. We used beam search decoding with beam size 5 (for top1 accuracies) and 10 (for top10 accuracies). The configurations for different models were the best ones we found based on validation performance. We trained these models for 100K steps and picked the checkpoint with the best BLEU on the validation set to evaluate on the test set. Training the transformer and HAT took 72 hours on 4 Tesla-V100 GPUs.

5.5.2 Evaluation

Widely used metrics for text generation are BLEU score [92] and ROUGE score [74] that measure n-gram overlap between hypotheses and references. Both metrics are not ideal in mathematical proofs since a proposition can be invalid due to one or two incorrect tokens. Therefore, in addition to BLEU score, we also consider exact match between hypotheses and references as our evaluation metric. We report top-1 accuracy and top-10 accuracy. The top-1 accuracy is the percentage of the best output sequences that are correct in the given dataset. The top-10 accuracy is the percentage of target sequences appearing in the top 10 generated sequences.

It is possible that models generate alternative valid propositions that are not exactly the same as the references. We further implemented a test suite to bring the generated propositions back to the Isabelle environment and check their correctness using automatic theorem provers (ATPs).

[3]Codebase: https://github.com/tensorflow/nmt
[4]Codebase: https://github.com/pytorch/fairseq/tree/master/examples/conv_seq2seq
[5]Codebase: https://github.com/THUNLP-MT/THUMT/tree/pytorch

Table 5.1: Test set accurarcies (exact match) and BLEU scores of different models on the IsarStep task.

Model	Top-1 Acc.		Top-10 Acc.		BLEU	
	Base	+**F.5**	Base	+**F.5**	Base	+**F.5**
RNNSearch	13.0	16.7	26.2	32.2	42.3	52.2
Conv2Conv	8.7	8.3	21.7	20.8	48.7	46.5
Transformer	20.4	22.1	33.1	34.6	59.6	62.9
HAT	**22.8**	**24.3**	**35.2**	**37.2**	**61.8**	**65.7**

5.5.3 Results

BLEU and Exact Match Table 5.1 presents the results of different models for the IsarStep task. Overall, the neural seq2seq models achieve around 13–25% top-1 accuracies and 26–38% top-10 accuracies, which indicates that this task is non-trivial and yet not too difficult for neural networks. Of the three models, the transformer [123] outperforms the RNNSearch [3, 136] significantly and our HAT performs best. As mentioned in Section 5.2, adding **F.5** is optional and is conjectured for better performance due to exploiting used lemmas explicitly. We experimented with both cases and found that adding this extra information indeed leads to further improvement. This is consistent with the scenario when humans prove theorems: if humans are told that certain lemmas are relevant to the current proof, they will use these lemmas and have a better chance of success.

Alternative Valid Propositions We consider an output proposition P as an alternative valid intermediate proposition if 1) P is a well-formed proposition and does not match the ground truth at the surface form; 2) P does not match any substring of the source (to avoid it being a simple copy of **F.3** or an assumption in **F.2**); 3) both **F.2** $\Rightarrow P$ and $P \Rightarrow$ **F.3** can be automatically proved by ATPs.[6] Note that this will only give us a lower bound to the number of alternative propositions, due to the limitation of ATPs' automation. Table 5.2 presents the percentage of correct propositions on the test set. Correct proposition is a proposition that matches either the corresponding ground truth or one of the alternative valid propositions. We can see that alternative propositions contribute 5 percentage point more correct propositions, compared to top-1 accuracy in Table 5.1.

Table 5.2: Percentage of correct propositions.

Model	Base	+**F.5**
Transformer	25.2	26.8
HAT	27.6	29.4

Automation Improvement In lots of cases, ATPs cannot infer from one step to another automatically (i.e. **F.2** $\Rightarrow$ **F.3**) without the crucial intermediate steps proposed by humans. We found that there are about 3000 cases in our test set that **F.2** $\Rightarrow$ **F.3** cannot be proved automatically by ATPs. And within these 3000 cases, 61 cases can be proved automatically given the generated intermediate propositions from our HAT model: **F.2** $\Rightarrow P$ and $P \Rightarrow$ **F.3**. This is not a big improvement. Further progress is needed to improve seq2seq models' reasoning capability in order to improve the automation of theorem provers significantly.

[6] If **F.2** $\Rightarrow$ **F.3** is directly provable via ATPs, a trivial proposition (e.g. $1 = 1$) can be considered as an alternative. This is hard to detect automatically but could still serve as a fair comparison across seq2seq models as long as they can propose a well-formed trivial proposition in such scenario.

Better Generalisation of HAT Since the transformer and HAT have different source sequence encoding, we explore how well these two models perform on examples with various source sequence lengths. We categorise the examples on the IsarStep test set into 5 buckets based on their source lengths and calculate the top-1 accuracies for different buckets, as shown in Fig. 5.3. Interestingly, although we did not train the models with source sequences longer than 512, they can still achieve reasonable accuracies on long sequences. In particular, HAT performs significantly better than the transformer on sequences longer than 480. Especially in the length bucket of 640–800, HAT doubles the accuracy of the transformer.

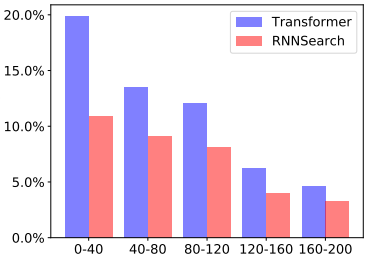


Figure 5.3: Accuracy of different source sequence lengths.

Importance of Category Information We subsequently investigate the effect of incorporating the category information for source propositions into the models by removing the category embedding for the input to the HAT encoder (Fig. 5.2), i.e. we are now modelling $p(\boldsymbol{y} \mid \boldsymbol{X})$ instead of $p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\mathcal{C}})$. We see a dramatic drop in accuracy: 14.6 versus 22.8 obtained by the HAT with category embedding included, indicating the importance of category information. This is in line with human proofs: without knowing the logical relations between propositions, we do not know what proposition is missing. This indicates that the models are not simply doing pattern matching but should have captured some reasoning information.

5.6 Test Suite Evaluation

We use the default Sledgehammer [12] method in Isabelle as our automatic theorem prover for checking derivations. To ensure fair and efficient comparisons, we shut off three of its options: "isar_proofs", "smt_proofs" and "learn". The timeout for Sledgehammer is 30s. We run the test suite on a platform with Intel 9700K CPU and 32G RAM, and it takes about 44 hours to evaluate on the test set. Due to some technical issues (e.g., the sampled example appears before Sledgehammer is introduced when booting Isabelle/HOL), 92/5000 examples from the test set are not supported by the current version of our test suite. We present the percentage of well-formed propositions (i.e., outputs that type checks in Isabelle/HOL) of Transformer and HAT in Table 5.3.

Table 5.3: Percentage of well-formed propositions

Model	Base	+**F.5**
Transformer	58.2	60.1
HAT	58.2	58.9

With such settings, Sledgehammer can automatically prove the goal **F.3** in 1984/4908 examples. By incorporating the ground truth **F.1**, the derivations (i.e., to prove both **F.2** ⇒ **F.1** and **F.1** ⇒ **F.3**) can be closed automatically in 2243 examples. Among the 1125 examples that HAT produces an exact match, 466 of them have a 'small' gap: Sledgehammer discharges the goal **F.3** directly; 61 of gaps are 'just right': the introduced intermediate step **F.1** can help Sledgehammer bridge the gap; most of the remaining 598 examples have 'large' gaps (in either **F.2** ⇒ **F.1** or **F.1** ⇒ **F.3**) that are beyond the capability of Sledgehammer. It appears that the insignificant amount of automation improvement can be attributed to the limited number of 'just right' gaps that are within the reach of Sledgehammer.

5.7 Qualitative Analysis

In this section, we present an analysis of what has been learnt by the neural network models. To summarise our findings: 1) the seq2seq models can learn the syntax of propositions correctly; 2) the learned token embeddings are comprehensive in that related mathematical concepts are close in cosine space; 3) manual inspection of the generated propositions reveal that models can learn non-trivial mathematical reasoning and even more complicated multi-step reasoning.

Token Embeddings To investigate whether the seq2seq models have learnt mathematical reasoning, we checked whether the learnt token embeddings were meaningful. We first projected the learnt embeddings for all the tokens in the vocabulary into a three-dimensional space via principal component analysis and chose random tokens and checked their 50 nearest neighbours in cosine distance. We found that the embeddings of related concepts in mathematics were close, indicating that the models have managed to learn the relations between mathematical concepts — the basic step towards reasoning mathematically. For example, in Fig. 5.4, the neighbours of 'Borel measurable' are mostly measure theory related including 'almost everywhere', 'integrable', and 'null set', while 'arrow' is close to 'isomorphism' (EpiMonoIso.category.iso), 'identity'(Category.partial_magma.ide), and 'inverse arrow'(EpiMonoIso.category.inv), which are concepts in category theory. Additionally, vector arithmetic also seems to connect related mathematical definitions: for example, the three closest tokens next to 'bounded' + 'closed' are 'bounded','closed', and 'compact', where compactness can be alternatively defined as boundedness and closedness (on a Euclidean space).

Attention Visulisations We next investigate how reasoning has been learnt by visualising attentions from transformer [123]. We find that important and related tokens are likely to attend to each other. For example, Fig. 5.5 illustrates the visulisation of the last layer of the transformer encoder for the source propositions **F.2**: **F.3**: $x_{70} \in x_{39}$ **F.4**: $x_{57} \subseteq x_{39}$. The target proposition generated from the model is $x_{70} \in x_{57}$. The interpretation of those source propositions is that combining with (**F.4**) $x_{57} \subseteq x_{39}$ we would like to infer the intermediate step so that the goal $x_{70} \in x_{39}$ can be proved. The transformer model gives the correct answer $x_{70} \in x_{57}$ which implicitly applied the lemma

$$x \in A, A \subseteq B \vdash x \in B \tag{5.2}$$

that relates $\in$ and $\subseteq$. On the last self-attention layer of the transformer encoder (Fig. 5.5), $\in$ and $\subseteq$ attend to each other. Interestingly, the above reasoning seems robust. If we swap x_{57} and x_{39} in **F.4** (i.e., the source is now **F.2**: **F.3**: $x_{70} \in x_{39}$ **F.4**: $x_{39} \subseteq x_{57}$), the answer becomes $x_{70} \in x_{39}$. This totally makes sense since (5.2) no longer applies (despite that $\in$ and $\subseteq$ still attend to each other similarly as in Fig. 5.5) and $x_{70} \in x_{39}$ can only be discharged by proving itself.

Multi-Step Reasoning By further inspecting the generated propositions, we find that the model can implicitly invoke multiple theorems as humans normally do. While this property can be found in quite a few examples, here we show one of them due to the limited space. We refer the readers to the appendix for more examples. Given the source **F.2**: $\dim(\text{span}(x_0)) \leq \text{card}(x_2)$ **F.3**: $\text{card}(x_2) = \dim(x_0)$ **F.4**: $\text{card}(x_2) \leq \dim(x_0)$, $\text{finite}(x_2)$, where dim, span and card refer to the dimensionality, the span, and the cardinality of a set of vectors, respectively, and the model gives the correct

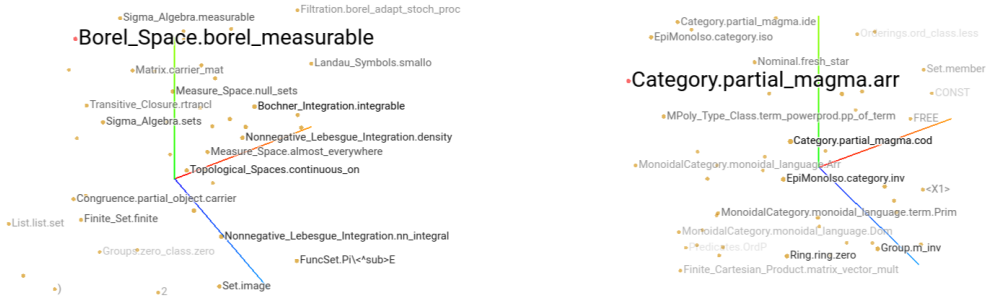


Figure 5.4: Nearest neighbours of the tokens 'Borel measurable' (left) and 'arrow' (right) in cosine space. The 512-dimensional embeddings are projected into 3-dimensional embeddings. Neighbours are found by picking the top 50 tokens whose embeddings are closest to the selected token.

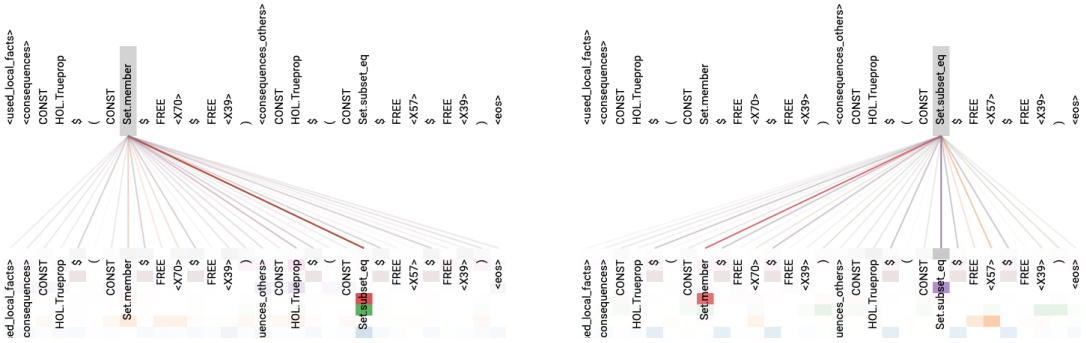


Figure 5.5: Attention visualisation of the last layer of the transformer encoder for the source propositions **F.2**: **F.3**: $x_{70} \in x_{39}$ **F.4**: $x_{57} \subseteq x_{39}$. The generated target proposition is $x_{70} \in x_{57}$.

answer $\dim(x_0) \leq \text{card}(x_2)$. Here, $\dim(x_0) \leq \text{card}(x_2)$ is derived by $\dim(\text{span}(x_0)) \leq \text{card}(x_2)$ only if the model has implicitly learned the following theorem $\vdash \dim(\text{span}(S)) = \dim(S)$, while $\dim(x_0) \leq \text{card}(x_2)$ yields $\text{card}(x_2) = \dim(x_0)$ (in conjunction of $\text{card}(x_2) \leq \dim(x_0)$) only if the model has implicitly invoked the antisymmetry lemma $x \leq y, y \leq x \vdash x = y$.

Failures We observe that incorrect propositions are well-formed and plausible propositions but they are usually a copy of parts of the source propositions.

5.8 Examples of correct synthesises

In this section, we present some correctly synthesised propositions which will be labelled as **F.1** in each case.

#605 in the validation set:

F.1 :

$$\text{additive}(\mathcal{A}_{x_1}, \mu_{x_0}) \tag{5.3}$$

F.2 :

$$\text{subalgebra}(x_0, x_1) \tag{5.4}$$

F.3 :

$$\text{measure\_space}(\mathcal{X}_{x_0}, \mathcal{A}_{x_1}, \mu_{x_0}) \tag{5.5}$$

F.4 :

$$\sigma(\mathcal{X}_{x_0}, \mathcal{A}_{x_1}) \tag{5.6}$$

$$\text{positive}(\mathcal{A}_{x_1}, \mu_{x_0}) \tag{5.7}$$

$$\text{measure\_space}(v_1, v_0, v_2) = (\sigma(v_1, v_0) \wedge \text{positive}(v_0, v_2) \wedge \text{additive}(v_0, v_2)) \tag{5.8}$$

Here, x_0 and x_1 are measure spaces. For a measure space y, $\mathcal{X}_y$, $\mathcal{A}_y$, and μ_y are the three components of y (i.e., $y = (\mathcal{X}_y, \mathcal{A}_y, \mu_y)$), where $\mathcal{X}_y$ is the carrier set, $\mathcal{A}_y$ is a collection of subsets on $\mathcal{X}_y$, and μ_y is a measure defined on $\mathcal{A}_y$. **F.2** $\Rightarrow$ **F.1**: x_1 being a subalgebra of x_0 (i.e., (5.4)) implies $\mathcal{A}_{x_1} \subseteq \mathcal{A}_{x_0}$, so that μ_{x_1} in additive$(\mathcal{A}_{x_1}, \mu_{x_1})$ (i.e., μ_{x_1} is countably addictive on $\mathcal{A}_{x_1}$ which is implied by x_1 being a measure space) can be substituted with μ_{x_0} which yields (5.3). **F.1**, **F.4** $\Rightarrow$ **F.3**: deriving (5.5) requires unfolding the definition of measure spaces (i.e., (5.8)), which requires v_0 is a sigma algebra on v_1, the measure v_2 is non-negative on v_0, and v_2 is countably additive on v_0. Two of the three requirements have already been satisfied by (5.6) and (5.7) respectively, while (5.3) entails the last one and eventually leads to (5.5).

#2903 in the validation set:

F.1 :

$$x_{29} \notin \text{path\_image}(x_7) \tag{5.9}$$

F.2 :

$$x_{29} \in \text{proots}(x_0) - \text{proots\_within}(x_0, \text{box}(x_1, x_2)) \tag{5.10}$$

$$\text{path\_image}(x_7) \cap \text{proots}(x_0) = \{\} \tag{5.11}$$

F.3 :

$$x_{29} \notin \text{cbox}(x_1, x_2) \tag{5.12}$$

F.4 :

$$\text{cbox}(x_1, x_2) = \text{box}(\text{x}_1, \text{x}_2) \cup \text{path\_image}(x_7) \tag{5.13}$$

$$x_{29} \in \text{proots}(x_0) - \text{proots\_within}(x_0, \text{box}(x_1, x_2)) \tag{5.14}$$

Here, path_image(x_7) is the image of the path function x_7 on the interval $[0, 1]$; proots(x_0) and proots_within(x_0, S) are, respectively, the roots of a polynomial x_0 and the roots (of x_0) within a set S; box$(x_1, x_2) = \{x \mid x_1 < x < x_2\}$ and cbox$(x_1, x_2) = \{x \mid x_1 \leq x \leq x_2\}$ are (bounded) boxes on an Euclidean space. **F.2** $\Rightarrow$ **F.1**: x_{29} is a root of x_0 (by (5.10)) that does not intersect with the path of x_7 (i.e., (5.11)). **F.1**, **F.4** $\Rightarrow$ **F.3**: combining with (5.13), (5.12) is equivalent to

$x_{29} \notin \mathrm{box}(\mathrm{x}_1, \mathrm{x}_2) \wedge x_{29} \notin \mathrm{path\_image}(x_7)$, which follows from joining (5.14) with (5.9).

#1514 in the validation set:

F.1 :

$$\frac{x_4(2x_{10})}{x_4(x_{10})} \leq x_9 \tag{5.15}$$

F.2 :

$$x_9 = \mathrm{Max}\left\{\frac{x_4(2y)}{x_4(y)} \mid y \leq x_8\right\} \tag{5.16}$$

$$x_{10} \leq x_8 \tag{5.17}$$

F.3 :

$$x_4(2x_{10}) \leq x_9 x_4(x_{10}) \tag{5.18}$$

F.4 :

$$0 < x_4(x_{10}) \tag{5.19}$$

F.2 $\Rightarrow$ **F.1**: (5.17) implies

$$\frac{x_4(2x_{10})}{x_4(x_{10})} \in \left\{\frac{x_4(2y)}{x_4(y)} \mid y \leq x_8\right\},$$

hence (5.15) by the definition of Max. **F.1**, **F.4** $\Rightarrow$ **F.3** by arithmetic and the positivity of the denominator (i.e., (5.19)).

#1222 in the validation set:

F.1 :

$$|ix_0 + \sqrt{1 - x_0^2}| = 1 \tag{5.20}$$

F.2 :

$$|ix_0 + \sqrt{1 - x_0^2}|^2 = 1 \tag{5.21}$$

F.3 :

$$\Im(\arcsin(x_0)) = 0 \tag{5.22}$$

F.4 :

$$\Im(\arcsin(v_0)) = -\ln(|iv_0 + \sqrt{1 - v_0^2}|) \tag{5.23}$$

$$e^{-v_0} = 1/(e^{v_0}) \tag{5.24}$$

F.2 $\Rightarrow$ **F.1** by arithmetic. **F.1**, **F.4** $\Rightarrow$ **F.3**:

$$\Im(\arcsin(v_0)) = -\ln(|iv_0 + \sqrt{1 - v_0^2}|) = -\ln 1 = 0.$$

#35 in the validation set:

F.1 :

$$x_4 = x_{10}[x_{12}] \tag{5.25}$$

F.2 :

$$\forall x.\, x_3 = x_6[x] \wedge x < \text{len}(x_6) \longrightarrow x_4 = x_{10}[x] \tag{5.26}$$

$$x_3 = x_6[x_{12}] \tag{5.27}$$

$$x_{12} < \text{len}(x_6) \tag{5.28}$$

F.3 :

$$x_4 = x_7[x_{11}] \tag{5.29}$$

F.4 :

$$x_7 = x_9 \# x_{10} \tag{5.30}$$

$$x_{12} < \text{len}[x_6] \tag{5.31}$$

$$x_{11} = x_{12} + 1 \tag{5.32}$$

Here, $x_{10}[x_{12}]$ refers to the $x_{12}{}^{\text{th}}$ element in the list x_{10}; len is the length function on a list; $x_9 \# x_{10}$ is a list where the element x_9 is concatenated to the front of the list x_{10}. **F.2** $\Rightarrow$ **F.1** by instantiating the quantified variable x in (5.26) to x_{12} and combining with (5.27 - 5.28). **F.1**, **F.4** $\Rightarrow$ **F.3**:

$$x_4 = x_{10}[x_{12}] = (x_9 \# x_{10})(x_{12} + 1) = x_7[x_{11}].$$

5.8.1 Alternative Steps

Many alternative steps are trivially equivalent to the ground truth (e.g. $A = B$ given the ground truth being $B = A$, and $P \wedge 1 = 1$ given the truth being P). However, we still manage to find a few non-trivial ones, and one of them (#954 in the test set) even identifies a redundant derivation in the Isabelle standard library:

F.1 :

$$0 = x_1(x_9)2(2\pi)\mathrm{i}n(x_3, x_9) \tag{5.33}$$

F.2 :

$$x_7 = \{w \mid w \notin \text{path\_image}(x_3) \wedge n(x_3, w) = 0\} \tag{5.34}$$

$$x_9 \in x_7 \tag{5.35}$$

F.3 :

$$\oint_{x_3} \frac{dx}{x - x_9} = 0 \tag{5.36}$$

F.4 :

$$\forall z \notin \text{path\_image}(x_3). \oint_{x_3} \frac{dx}{x - z} = \frac{n(x_3, z)}{2\pi\mathrm{i}} \tag{5.37}$$

$$x_7 = \{w \mid w \notin \text{path\_image}(x_3) \wedge n(x_3, w) = 0\} \tag{5.38}$$

$$x_9 \in x_7 \tag{5.39}$$

Here, i is the imaginary unit, path_image(x_3) returns the image of the contour x_3 on the interval $[0, 1]$, and $n(x_3, x_9)$ is the winding number of x_3 around the point x_9. **F.2** $\Rightarrow$ **F.1**: combining (5.34) and (5.35) leads to $n(x_3, x_9)$ that proves (5.33). **F.1**, **F.4** $\Rightarrow$ **F.3**: joining (5.38) and (5.39) yields

$$x_9 \notin \text{path\_image}(x_3) \tag{5.40}$$

$$n(x_3, x_9) = 0 \tag{5.41}$$

By further joining (5.40) with (5.37) we have

$$\oint_{x_3} \frac{dx}{x - x_9} = \frac{n(x_3, x_9)}{2\pi\mathrm{i}}$$

which leads to (5.36) considering $n(x_3, x_9) = 0$ (i.e., (5.41)). Note that our **F.1** is not used in the derivation above hence redundant. Instead of the redundant ground truth, HAT proposed (5.40) which is clearly a much better intermediate step.

5.9 Related Work

There have been a series of work that evaluates mathematical reasoning abilities of seq2seq models. The tasks that these works attempt to solve include school-level mathematical problems [75, 106],

function integration and ordinary differential equations [69], properties of differential systems [19], SAT formulas and temporal logic [36]. Our task is different from the previous ones in the sense that ours is non-synthetic, with realistic vocabulary size (i.e., 30K vs. less than 100) and has a broad coverage topics in research-level mathematics and computer science that have no general algorithmic solutions.

This work is closely related to the most recent work on applying language modelling to theorem proving. Urban and Jakubův [121] present initial experiments on generating conjectures using GPT-2 [100]. Polu and Sutskever [97] show that the GPT-3 language model [15] additionally pretrained with mathematical equations mined from the web can generate propositions that enable theorem provers to prove more theorems automatically. Rabe et al. [98] pretrain a masked language models on proofs mined from the HOList dataset [5] and apply the pretrained models to the downstream tasks of type inference and predicting conjectures. While both their work and ours find that transformer models have strong mathematical reasoning capabilities, they have different objectives from ours. Their objectives are to show the effectiveness of pretraining on downstream tasks; by contrast we are building benchmarks to test models' ability of solving mathematical problems. In fact, we can pretrain seq2seq models following their proposed methods and verify their effectiveness on our dataset. We will leave this for future work.

Other related work includes analogy/syntax driven conjecturing [39, 86, 126], goal classification/ranking [42, 14], proof method recommendations [84, 85], and autoformalisation of mathematics [129, 118].

Hierarchical Models Hierarchical models have been proposed to solve natural language processing tasks such as document representation [145] and document summarisation [147, 76]. Both our hierarchical transformer (HAT) and those models share the similar spirit of introducing local layers to encode local sentences (or propositions) and global layers to capture cross sentence (or proposition) information. However, our HAT is different from their hierarchical models in the way of representing sentences (or propositions): while their models encode sentences into fixed size vectors, the representation of a proposition in our model is a matrix of dynamic size. The model by Liu and Lapata [76] has a more sophisticated architecture for capturing sentence representations compared to those by Yang et al. [145] and Zhang, Wei, and Zhou [147], where they introduce multi-head pooling to encode sentences with different attention weights. Compare to [76]'s model, our model does not introduce additional parameters beyond the standard transformers. Another subtle difference between our model and the existing models is the way of doing positional encoding. Unlike documents where the order of sentences matters, propositions within each category of our IsarStep task do not require an order. Therefore, we do not encode the positional information of different propositions.

5.10 Conclusion

We mined a large corpus of formal proofs and defined a proposition generation task as a benchmark for testing machine learning models' mathematical reasoning capabilities. In our defined task, the gap between adjacent proof steps is big and therefore it cannot be simply solved by pattern matching and rewriting. We evaluated the RNN attention model and the transformer on this dataset

and introduced a hierarchical transformer that outperforms the existing seq2seq model baselines especially on long source sequences. Our analysis shows that the neural seq2seq models can learn non-trivial logical relations and mathematical concepts. We hope that our work will drive the development of models that can learn to reason effectively and eventually build systems that can generate human-readable proofs automatically.

Chapter 6

Induction – REFACTOR

Human mathematicians are often good at recognizing modular and reusable theorems that make complex mathematical results within reach. In this chapter, we propose a novel method called theoREm-from-prooF extrACTOR (REFACTOR) for training neural networks to mimic this ability in formal mathematical theorem proving. We show on a set of unseen proofs, REFACTOR is able to extract 19.6% of the theorems that humans would use to write the proofs. When applying the model to the existing Metamath library, REFACTOR extracted 16 new theorems. With newly extracted theorems, we show that the existing proofs in the MetaMath database can be refactored. The new theorems are used very frequently after refactoring, with an average usage of 733.5 times, and help to shorten the proof lengths. Lastly, we demonstrate that the prover trained on the new-theorem refactored dataset proves relatively 14-30% more test theorems by frequently leveraging a diverse set of newly extracted theorems.

This chapter is largely based on the work: *REFACTOR: Learning to Extract Theorems from Proofs* by Yuhuai Wu*, Jin Peng Zhou*, Colin Li, Roger Grosse [140], published in the *Mathematical Reasoning in General Artificial Intelligence Workshop* (MATHAI) at ICLR of 2021. Being the lead of this work, my contribution includes the original idea of REFACTOR, the theorem expansion algorithm, designs of experiments, GNN learning and training, and paper writing. I strongly thank Jin Peng Zhou's contribution for developing the codebase of GNN training, besides his other contributions in experimental results, analysis and paper writing.

6.1 Overview

In the history of calculus, one remarkable early achievement was made by Archimedes in the 3rd century BC, who established a proof for the area of a parabolic segment to be 4/3 that of a certain inscribed triangle. In the proof he gave, he made use of a technique called the *method of exhaustion*, a precursor to modern calculus. However, as this was a strategy rather than a theorem, applying it to new problems required one to grasp and generalize the pattern, as only a handful of brilliant mathematicians were able to do. It wasn't until millennia later that calculus finally became a powerful and broadly applicable tool, once these reasoning patterns were crystallized into modular concepts such as limits and integrals.

A question arises – can we train a neural network to mimic humans' ability to extract modular

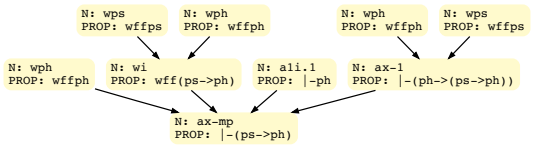
components that are useful? In this paper, we focus on a specific instance of the problem in the context of theorem proving, where the goal is to train a neural network model that can discover reusable theorems from a set of mathematical proofs. Specifically, we work under formal systems where each mathematical proof is represented by a tree called *proof tree*. Moreover, one can extract some connected component of the proof tree that constitutes a proof of a standalone theorem. Under this framework, we can reduce the problem to training a model that solves a binary classification problem where it determines whether each node in the proof tree belongs to the connected component that the model tries to predict.

To this end, we propose a method called theoREm-from-prooF extrACTOR (REFACTOR) for mimicking humans' ability to extract theorems from proofs. Specifically, we propose to reverse the process of human theorem extraction to create machine learning datasets. Given a human proof T, we take a theorem s that is used by the proof. We then use the proof of theorem s, T_s, to re-write T as T' such that T' no longer contains the application of theorem s, and replace it by using the proof T_s. We call this re-writing process the *expansion* of proof T using s. The expanded proof T' becomes the input to our model, and the model's task is to identify a connected component of T', T_s, which corresponds to the theorem s that humans would use in T.

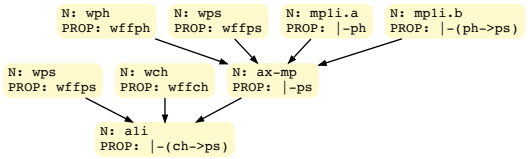
We implement this idea within the Metamath theorem proving framework – an interactive theorem proving assistant that allows humans to write proofs of mathematical theorems and verify the correctness of these proofs. Metamath is known as a lightweight theorem proving assistant, and hence can be easily integrated with machine learning models [133, 97]. It also contains one of the largest formal mathematics libraries, hence providing sufficient background for proving university-level or Olympiad mathematics. While our approach would be applicable to other formal systems (such as Lean [83], Coq [6], or HOL Light [52]), we chose Metamath for this project because of its features for reduced iteration time in the near term.

Our work establishes the first proof of concept using neural network models to extract theorems from proofs. Our best REFACTOR model is able to extract exactly the same theorem as humans' ground truth (without having seeing instances of it in the training set) about 19.6% of time. We also observe that REFACTOR's performance improves when we increase the model size, suggesting significant room for improvement with more computational resources.

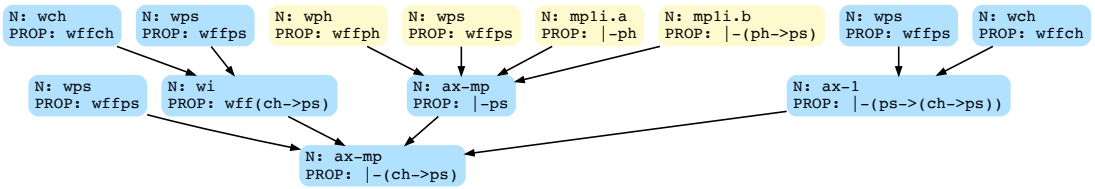
Ultimately, the goal is not to recover known theorems but to discover new ones. To analyze those cases where REFACTOR's predictions don't match the human ground truth, we developed an algorithm to verify whether the predicted component constituent a valid proof of a theorem, and we found REFACTOR extracted 1907 valid, new theorems. We also applied REFACTOR to proofs from the existing Metamath library, from which REFACTOR extracted another 16 novel theorems. Remarkably, those 16 proofs are used very frequently in the Metamath library, with an average usage of 733.5 times. Furthermore, with newly extracted theorems, we show that the human theorem library can be refactored to be more concise: the extracted theorems reduce the total size by approximately 400k nodes. (This is striking since REFACTOR doesn't explicitly consider compression as an objective.) Lastly, we demonstrate that training a prover on the refactored dataset leads to a 14-30% relative improvement on proof success rates in proving new test theorems. Out of all proved test theorems, there are 43.6% of them use the newly extracted theorems at least once. The usages also span across a diverse set of theorems: 141 unique newly extracted theorems are used, further suggesting diverse utility in new theorems we extracted.



(a) The proof tree of `a1i`.



(b) The proof tree of `mp1i`.



(c) The proof tree of `mp1i` with theorem `a1i`'s proof expanded (colored in blue).

Figure 6.1: In (a) and (b), we show proof tree visualizations of the theorem `a1i` and `mp1i`. Each node contains two pieces of information: `N` refers to the name associated with the node, and `PROP` refers to the proved proposition that is obtained by applying all theorem applications above that node. In (c), we also show the expanded proof tree of `mp1i` with `a1i`'s proof being expanded and colored in blue, namely, the set of nodes $\mathcal{V}_{target}$ that are the targets for our proposed learning task.

Our main contributions are as follows: 1. We propose a novel method called REFACTOR to train neural network models for the theorem extraction problem, 2. We demonstrate REFACTOR can extract unseen human theorems from proofs with a nontrivial accuracy of 19.6%, 3. We show REFACTOR is able to extract frequently used theorems from the existing human library, and as a result, shorten the proofs of the human library by a substantial amount. 4. We show new-theorem refactored dataset can improve baseline theorem prover performance significantly with newly extracted theorem being used frequently and diversely.

6.2 Metamath and Proof Representation

In this section, we describe how one represents proof in the Metamath theorem proving environment. We would like to first note that even though the discussion here specializes in the Metamath environment, most of the other formal systems (Isabelle/HOL, HOL Light, Coq, Lean) have very similar representations. The fundamental idea is to think of a theorem as a function, and the proof tree essentially represents an abstract syntax tree of a series of function applications that lead to the intended conclusion.

Proof of a theorem in the Metamath environment is represented as a tree. For example, the proof of the theorem `a1i` is shown in Figure 6.1 (a). Each node of the tree is associated with a *name* (labeled as `N`), which can refer to a premise of the theorem, an axiom, or a proved theorem from the existing theorem database. Given such a tree, one can then traverse the tree from the top to bottom, and iteratively prove a true proposition (labeled as `PROP`) for each node by making a step of *theorem application*. The top-level nodes usually represent the premises of the theorem, and the resulting proposition in the bottom node matches the conclusion of the theorem. In such a way, the theorem is proved.

We now define one step of theorem application. When a node is connected by a set of parent nodes, it represents a step of theorem application. In particular, one can think of a theorem as a function that maps a set of hypothesis to a conclusion. Indeed, a node in the tree exactly represents such function mapping, that is to map the set of propositions of the parent nodes, to a new conclusion specified by the theorem. Formally, given a node c whose associated name refers to a theorem T, we denote its parent nodes as $\mathcal{P}_c$. We can then prove a new proposition by applying the theorem T, to all propositions proved by nodes in $\mathcal{P}_c$.

The proof of the theorem `a1i` in Figure 6.1 (a) consists of 3 theorem applications. In plain language, the theorem is a proof of the fact that if `ph` is true, then `(ps->ph)` is also true. The top-level nodes are the hypotheses of the theorem. Most of the hypotheses state that some expression is a well-formed formula so that the expression can be used to form a syntactically correct sentence. The more interesting hypothesis is `a1i.1`, which states `|-ph`, meaning `ph` is assumed to be true. In the bottom node, the theorem invokes the theorem `ax-mp`, which takes in four propositions as hypotheses, and returns the conclusion `|-(ps->ph)`.

6.3 Method

In this section, we describe our approach to training neural network models for extracting useful theorems from proofs. Our approach inspects one proof at a time and this intuition comes from the fact that human mathematicians do not need to look at multiple proofs and can instead determine whether a proof segment is broadly applicable just from the current proof. As one can represent mathematical proofs as trees, we first discuss how to identify a connected component of the tree with a valid proof of another theorem. We then formalize the problem of theorem extraction as a node-level binary classification problem on the proof tree. Next, we propose an algorithm that expands a theorem's proof inside of another proof, to create suitable targets for learning theorem extraction. Finally, we give an algorithm that verifies if the component predicted by the model constitutes a valid proof of a theorem, and if so, turns the component into a theorem.

6.3.1 Sub-component of a Proof Tree as a Theorem

We have discussed how one can represent a mathematical proof as a proof tree in section 6.2. Interestingly, one can also identify some components of the proof tree with an embedded proof of another theorem. To start with, given a node in a proof tree, one can treat the entire subtree above that node as a proof of the node (more precisely, the proposition contained in the node, i.e., `PROP`). For example, in the proof of `a1i`, the subtree above the node `ax-1` consists of two hypotheses `wffph` and `wffps`, and they constitute a proof of the proposition `|-(ph->(ps->ph))` contained in the node `ax-1`.

In addition to the entire subtree above a node, one may identify some connected component of the tree with a valid theorem. For example, in Figure 6.1 (c), we show that the proof of the theorem `mp1i` contains an embedded proof of the theorem `a1i`. The embedded proof is colored in blue, and there is a one-to-one correspondence between these blue nodes and the nodes in the proof of `a1i` shown in Figure 6.1 (a). One can hence refactor the proof with an invocation of the theorem `a1i`, resulting in a much smaller tree shown in Figure 6.1 (b).

In general, there are certain criteria a component needs to satisfy to be identified as a valid proof of a theorem. In Appendix 6.3.4, we develop such an algorithm in more detail that performs the verification. We will use that to verify the prediction given by a neural network model.

To conclude, in this section, we establish the equivalence between theorem extraction from a proof as to the extraction of a sub-component from a proof tree. This allows us to formalize the problem as a node-level prediction problem on graphs as we introduce next.

6.3.2 Supervised Prediction Task

The model is given a proof tree $\mathcal{G}$ with a set of nodes $\mathcal{V}$, edges $\mathcal{E}$, and node features x_v which correspond to the name `N` and the proposition `PROP` associated with each node. The task of the model is to output a subset of nodes $\mathcal{V}_{\text{target}} \subset \mathcal{V}$ that correspond to an embedded proof of a useful theorem. We cast the problem as a node-level binary classification problem that predicts whether each node belongs to $\mathcal{V}_{\text{target}}$. Without loss of generality, we let all nodes in $\mathcal{V}_{\text{target}}$ to have labels of 1 and the rest 0.

We use a graph neural network parametrized by θ to take a single graph and its node feature as input, and outputs a scalar $\hat{P}_v$ between 0 and 1 for each node $v \in \mathcal{V}$, representing the probability belonging to $\mathcal{V}_{\text{target}}$. Our objective is a binary cross entropy loss between the node level probabilities and the ground truth target for a graph. Because the number of nodes usually varies significantly across proofs, we normalize the loss by the number of nodes in the graph[1]:

$$\mathcal{L}(G,\theta) = -\frac{1}{|\mathcal{V}|}\sum_{v\in\mathcal{V}_{\text{target}}} \log P(\hat{P}_v = 1|\mathcal{G},\theta) \tag{6.1}$$

$$-\frac{1}{|\mathcal{V}|}\sum_{v\notin\mathcal{V}_{\text{target}}} \log P(\hat{P}_v = 0|\mathcal{G},\theta) \tag{6.2}$$

We then seek the best parameters by minimizing the loss over all proof trees:

$$\operatorname*{argmin}_{\theta} \sum_{G} \mathcal{L}(G,\theta). \tag{6.3}$$

6.3.3 REFACTOR: Theorem-from-Proof Extractor

With the prediction task formulated, we now describe how to generate training data points of proof trees $\mathcal{G}$ with suitable targets $\mathcal{V}_{\text{target}}$ defined. Even though we specialize our discussion in the context of Metamath, the same technique can be applied to other formal systems for creating datasets of theorem extraction, such as Lean [83].

It is worth noting that even though the existing human proofs from the Metamath library cannot be used directly, they offer us hints as to how to construct training data points. To illustrate, in Figure 6.1 (b), the proof of `mp1i` invokes a theorem application with `a1i`, which is a theorem that human considered useful and stored in the library. Our idea is to reverse the process of theorem extraction, by expanding the proof of `a1i` in the proof of `mp1i` to obtain a synthetic proof shown in 6.1 (c). In this expanded proof of `mp1i`, one can see the proof of `a1i` is embedded as a component colored in blue, hence creating a suitable target for theorem extraction.

[1] In our preliminary experiments we found that the normalized loss gave better performance than weighting all nodes in the database equally.

Algorithm 2 Theorem Expansion Algorithm Pseudocode

```
1: procedure EXPANSION
2:     Input: proof tree T that uses theorem s at node c.
3:     Input: proof tree of theorem s: T_s.
4:     nominalArguments = GetArguments(T_s)
5:     contextualArguments = [GetSubtree(p) for p in GetParents(c)]
6:     allNodeNames = GetAllNodeNames(T_s)
7:     f : nominalArguments → contextualArguments.
8:     f(i^th element of nominalArguments) ≜ i^th element of contextualArguments
9:     for each name N ∈ allNodeNames do
10:        if N ∈ nominalArguments then
11:            replace N with f(N)
12:        end if
13:    end for
14:    replacedProof = GetProof(allNodeNames)
15:    replace entire subtree above node c with replacedProof
16:    return T
17: end procedure
```

We explain how we perform the proof expansion in detail. We think of the theorem as a function whose arguments are a set of hypotheses and the output is a conclusion, as mentioned in 6.2. Instead of calling the theorem by its name, we intentionally duplicate the body of its proof tree, and manually replace their nominal arguments with the arguments we wish to pass in context. There are three key steps: 1. identifying the proof tree associated to the theorem, substituting nominal arguments with the ones in the proof context, and finally copy and replace it to where the expanded node is located. The pseudocode of the algorithm can be found in Algorithm 2. The algorithm takes input of two proof trees where the first proof tree uses the theorem that the second proof tree shows as one of the steps.

We illustrate our algorithm with the example from Figure 6.1. Specifically, proof tree T corresponds to Figure 6.1 (b) and proof tree T_s corresponds to Figure 6.1 (a). The theorem we want to expand is `a1i` and we first obtain all its arguments using `GetArguments` function. We treat each theorem as a function and its arguments are the hypothesis of the theorem used to compute the conclusion. Consequently, the nominal arguments are `wph`, `wps` and `a1i.1`. Next, we obtain contextual arguments, which are those specific hypotheses used in the context of the proof. Each hypothesis are represented by the entire subtree above each parent of c. Concretely, the contextual arguments of the `a1i` node in (b) are `wps`, `wch` and [`wph`, `wps`, `mp1i.a`, `mp1i.b`, `ax-mp`]. Here, we use square bracket to enclose a subtree that has more than one node, which is treated holistically as the third contextual argument. Note that we can clearly see a one-to-one correspondence between the nominal arguments and the contextual arguments: (`wph`→`wps`, `wps`→`wch` and `a1i.1`→[`wph`, `wps`, `mp1i.a`, `mp1i.b`, `ax-mp`]). We then simply replace all nodes in the proof tree of `a1i` using this mapping. This gives us [`wps`, `wch`, `wps`, `wi`, `wph`, `wps`, `mp1i.a`, `mp1i.b`, `ax-mp`, `wps`, `wch`, `ax-1`, `ax-mp`]. We generate its proof tree representation with `GetProof` function. Finally we replace the subtree above `a1i` with the new proof tree which in this case happens to be the entire proof of `mp1i` and this leads to the final expanded proof in Figure 6.1 (c).

Lastly, note that there are many options for theorem expansion. Firstly, one single proof can contain multiple theorems, and each theorem can be expanded either simultaneously or one by

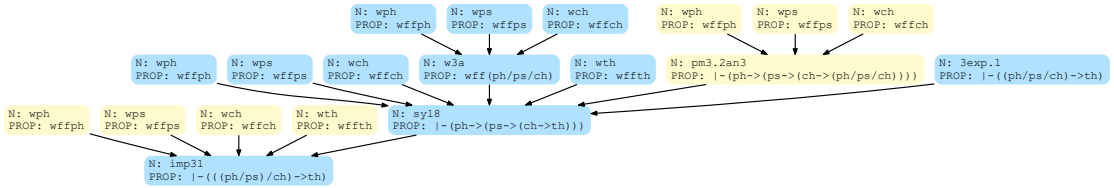
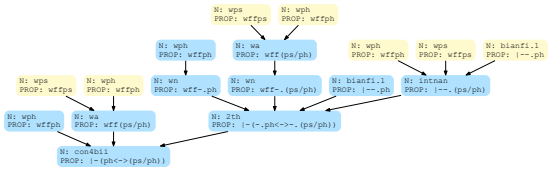
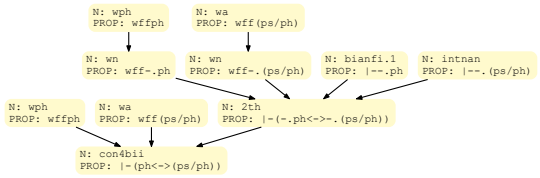


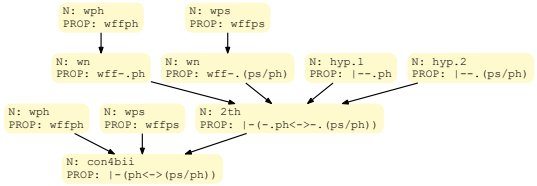
Figure 6.2: A proof tree prediction where nodes with output probability greater than 0.5 have been colored blue. This proof tree does not satisfy the constraint to be a valid theorem because only one of the parent nodes of the root are predicted to be in $\mathcal{V}_{target}$.



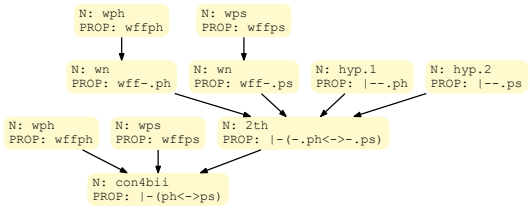
(a) A prediction made by REFACTOR with $\hat{\mathcal{V}}_{target}$ in blue.



(b) $\hat{\mathcal{V}}_{target}$ extracted from (a).



(c) $\hat{\mathcal{V}}_{target}$ extracted as in (b) with leaf node name and proposition replaced.



(d) A valid proof tree extracted and verified.

Figure 6.3: Visualization of theorem verification algorithm.

one. In addition, one can even recursively expand theorems by expanding the theorem inside of an expanded proof. For simplicity, in this work, we only expand one theorem at a time, and for every theorem in a proof. Hence, for a proof that contains M total number of theorem applications, we create M data points for learning theorem extraction. We leave investigations of more sophisticated expansion schemes to future work.

6.3.4 Verification of Theorem Extraction Prediction

In this section, we present our algorithm to determine whether a predicted component made by REFACTOR constitutes a valid theorem. On a high level, our algorithm checks two necessary conditions and performs standardization before feeding the node names of extracted component to a verifier which we describe next.

We describe how we can verify Metamath proofs represented by a conclusion and a list of node names such as the ones seen in the previous section. This can be easily achieved by calling `GetProof` from Algorithm 2 on the list of nodes names which follow a Reverse Polish Notation (RPN), and the function call returns a proof tree labelled with propositions (i.e., `PROP`) . We then compare

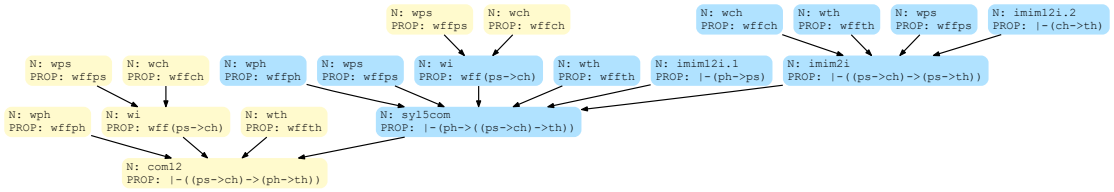


Figure 6.4: An example prediction that fails to be extracted as a new theorem due to no valid substitution plan in standardization. Specifically, the blue node `wi` cannot be substituted to a basic argument allowed in Metamath while still keeping the proof tree valid.

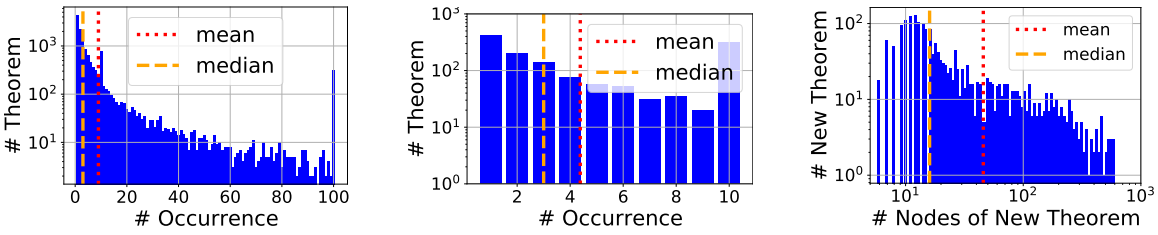
between the proposition given in the bottom node (conclusion) to the given conclusion specified by the theorem. The proof is verified if and only if the two conclusions are the same. We refer to this simple procedure as Metamath verifier.

For the theorem verification algorithm, We first take all node prediction with value greater than 0.5 as the set of extraction nodes, which we represent as $\hat{\mathcal{V}}_{target}$ (see Figure 6.3 (a) and (b)). We first check if $\hat{\mathcal{V}}_{target}$ forms a connected component i.e. a tree structure, as disjoint set of nodes cannot be a valid new theorem. Secondly, one necessary constraint for a valid extracted theorem is that for each node in $\hat{\mathcal{V}}_{target}$, either none or all of its parent nodes need to be present in $\hat{\mathcal{V}}_{target}$. If only some but not all parents are present, this corresponds to a step of theorem application with an incorrect number of arguments. We illustrate one example that violates this constraint in Figure 6.2. As seen in this example, only one parent of the root node is in $\hat{\mathcal{V}}_{target}$ and similarly one parent node of `syl8` is not in $\hat{\mathcal{V}}_{target}$. Because of these missing arguments, this will not be a valid new theorem. We note that although the extraction algorithm can be implemented in a way such that it "auto-completes" the arguments by adding additional necessary nodes into the set of extracted nodes, we choose not to do so in order to make sure the submodule is entirely identified by REFACTOR.

Once the extracted nodes pass these checks, we perform a so-called standardization. Here we once again leverage functions defined in Algorithm 2. Specifically, we replace all node names of leaf nodes with a pre-defined set of node names allowed in Metamath such as `wph`, `wps`. This can be achieved by first obtaining arguments of the extracted component via `GetArguments` and replacing these arguments in a fashion similar to Algorithm 2 except this time the nominal arguments are from the extracted component and contextual arguments will be the pre-defined arguments from Metamath convention. As seen in Figure 6.3 (c), we replace all leaf node names `wa` with `wps`.

After standardization, we simply feed all the node names of the extracted component into the verifier we have described to determine whether it is a valid theorem. For example, node names in (c) [`wph`, `wps`, `wph`, `wn`, `wps`, `wn`, `hyp.1`, `hyp.2`, `2th`, `con4bii`] are fed into the verifier and we arrive at Figure 6.3 (d).

Intuitively, this standardization process can be thought of as an reverse process of the steps performed in proof expansion algorithm. Instead of replacing simple and basic nominal arguments with complex contextual ones, we use pre-defined simple contextual arguments from Metamath to replace the complex nodes in the extracted proof tree. We note that verifying a proof after standardization is not always possible. Consider an example in Figure 6.4 where the two parent nodes of blue node `wi` are not included in $\hat{\mathcal{V}}_{target}$ but in fact included in $\mathcal{V}_{target}$. Because of this, we need to replace `wi` with a basic argument in Metamath such as `wta`. However, with this replacement,



(a) (b) (c)

Figure 6.5: Number of theorems vs number of occurrences in entire dataset (a) and test set (b). Both (a) and (b) show noticeable occurrence imbalance with (b) being less due to our further subsampling of a maximum 10 occurrence. (c) Distribution of number of nodes in new theorems extracted. The model mostly extracts short theorems but is also capable of extracting theorems that have hundreds of nodes.

the arguments of `syl5com` will no longer be valid because it needs an expression with two `wff` variables in the node we substituted. Therefore, there will be no valid substitution and this proof tree prediction cannot be extracted as a new theorem. We discard the extracted components that cannot be verified after standardization and only consider the ones that can be verified as new theorems.

6.4 Experiments

In this section, we evaluate the performance of our theorem extraction method via a variety of experiments. We begin by describing our dataset and experimental setup and then analyze the results to address the following research questions:

- **Q1**: How does REFACTOR perform when evaluating against ground truth theorem under a variety of ablations of data and model architectures?
- **Q2**: Are newly extracted theorems by REFACTOR used frequently?
- **Q3**: With newly extracted theorems, can we (a) compress the existing theorem library and (b) improve theorem proving?

6.4.1 Dataset and Pre-processing

We applied REFACTOR to create datasets from the main and largest library of Metamath, `set.mm`. In order to fairly compare prover performance reported from Whalen [133], we used their version of `set.mm`, which contains 27220 theorems. We also filtered out all expanded proofs with more than 1000 nodes or contain nodes features of character length longer than 512. This gave rise to 257264 data points for training theorem extraction before theorem maximum occurrence capping, which we describe next.

We noted that the distribution of theorem usage in `set.mm` is highly imbalanced. To prevent the model from learning to only extract a few numbers of common theorems due to their pervasiveness, we employed a subsampling of the data with respect to theorem occurrence to balance the dataset. Specifically, in the training set, for those theorems that occur more than 100 times as extraction targets, we subsampled 100 data points per theorem. In Figure 6.5 (a), we plot a histogram of

theorem occurrence versus the number of theorems. As seen in the figure, the distribution roughly follows a power-law distribution with 4000 theorems only used once in `set.mm`, and a substantial number of theorems that occur beyond 100 times. For the validation and test set, as we wanted to evaluate the model on a diverse set of extraction targets, we capped the maximum number of occurrences as 10 using subsampling. The occurrence histogram of the test dataset is shown in Figure 6.5 (b) and the total number of expanded proofs in our dataset after capping theorem maximum occurrence is 124294.

To evaluate the model's generalization ability, we performed a target-wise split on the dataset. That is, we split the dataset in a way that the prediction targets, namely, the theorems to be extracted, are different for the train, valid and test set. By doing so, we discouraged simple memorization of common theorems and extracting them from unseen proofs.

6.4.2 Model Architecture and Training Protocol

In this section, we describe our neural network architecture parameters and other training details. We used a character-level tokenization for the node feature, which is a concatenation of texts in the fields `N` and `PROP` (see Figure 6.1). For each node, we first embedded all the characters with an embedding matrix, followed by two fully connected layers. We then averaged over all embeddings to obtain a vector representation of a node. We used these vector representations as the initial node embeddings to a graph neural network. We used K GraphSage convolution [48] layers with size d and two more fully connected layers with sigmoid activation at the end to output the scalar probability. The size of the character embedding was set to 128 and the number of hidden neurons in all the fully connected layers was set to 64. Both K and d are hyperparameters.

For all of our model training, we used a learning rate of 1e-4 with Adam optimizer [66]. All methods were implemented in Pytorch[2] and Pytorch Geometric library [3]. We ran all experiments on one NVIDIA Quadro RTX 6000, with 4-core CPUs.

6.4.3 Q1 - How many human-defined theorems does the model extract?

On the theorem extraction dataset obtained from Section 6.4.1, REFACTOR was able to correctly classify 85.6% (Node Accuracy) of the nodes. For 19.6% (Proof Accuracy) of the proofs, REFACTOR was able to correctly classify all of the nodes and fully recover the theorem that the human use. We also show that our approach scales well with the model size (Table 6.2). As we increase the model by around 50x from 80k to 4M, both node and proof accuracy improve. In particular, the proof accuracy goes up significantly from 2.3% to 19.6%. This shows promise that the accuracy can be further improved by using a larger model with a larger dataset.

To understand what mechanism in the GNN made the theorem extraction possible, we re-trained the model, but with different configurations compared to the original training procedure. In particular, we examined the case where all the edges are removed (No edge) as well as two types of uni-directional connections: 1) only edges that go from leaves to root are included (Leaves→Root) and 2) only edges that go from root to leaves are included (Leaves←Root). In addition, we were curious to see whether the graph structure alone is sufficient for theorem prediction when no node features are provided.

[2] https://pytorch.org/
[3] https://pytorch-geometric.readthedocs.io/en/latest/

Table 6.1: Node level and proof level accuracy of REFACTOR with different input configurations. **No edge**: all the edges in the graph are removed; **Leaves→Root**: only keep the edges are in the same direction of the paths that go from leaves to their parents; **Leaves←Root**: same as Leaves→Root except all the edges are all reversed; **Leaves↔Root**: the original graph with bidirectional edges. **Node Features**: whether or not the node features are fed as input to the model. All the experiments are run with $K = 10$ and $d = 256$.

	Train Node Acc.	Train Proof Acc.	Test Node Acc.	Test Proof Acc.
No edge + Node Features	86.8%	0.1%	74.9%	0.1%
Leaves→Root + Node Features	87.1%	0.5%	75.2%	0.1%
Leaves←Root + Node Features	96.6%	6.0%	88.1%	3.5%
Leaves↔Root	86.3%	0%	74.2%	0%
Leaves↔Root + Node Features (**REFACTOR**)	97.5%	37.5%	84.3%	13.3%

Table 6.2: Node level and proof level accuracy of REFACTOR with various model sizes.

K, d, # Params	Train Node Acc.	Train Proof Acc.	Test Node Acc.	Test Proof Acc.
5, 64, 80k	89.4%	5.1%	77.4%	2.3%
5, 128, 222k	91.3%	9.9%	78.6%	3.0%
5, 256, 731k	93.7%	17.3%	80.1%	4.4%
10, 256, 1206k	97.5%	37.5%	84.3%	13.3%
10, 512, 4535k	97.9%	42.7%	85.6%	19.6%

For all the experiments, we used a model with $K = 10$ and $d = 256$. We summarize the results of these data configurations in Table 6.1 and report node level and proof level accuracy on training and test set. It can be seen that both edge connection and input node feature information is crucial in this task as both (No edge + Node Features) and (Leaves↔Root) achieved minimum proof level accuracy. Interestingly, the direction of edge led to a drastically different performance. Leaves→Root + Node Features performs poorly in proof level accuracy whereas Leaves←Root + Node Features achieved comparable performance with bidirectional edges (Leaves↔Root + Node Features).

This phenomenon can be explained by recognizing the fact that there are many identical hypothesis nodes in a proof due to MetaMath's low-level nature. For example, there are three identical leaf nodes `wps` in Figure 6.1 (c). If the edges only point from hypothesis to conclusion, the message for two identical hypothesis leaves will always be the same due to no incoming messages. Hence, it is theoretically impossible to make correct predictions on the proof level. On the other hand, the opposite direction of edges does not suffer from this limitation as there is only one root in the proof tree. Empirically, this configuration is able to achieve decent performance, but still far behind the performance of the model with bi-directional edges.

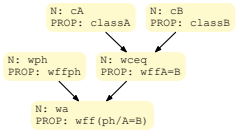
6.4.4 Q2 - Are newly extracted theorems by REFACTOR used frequently?

In this section, we investigate whether theorems extracted by REFACTOR are used frequently. We used the best model (i.e., the largest model) in Table 6.2 for the results analyzed in this section. We explored two ways of extracting new theorems. We first investigated the incorrect predictions of REFACTOR on the theorem extraction dataset. When the prediction differs from the ground truth, it can correspond to a valid proof. We also applied REFACTOR on the human proofs of nodes less than 5000 from the library `set.mm`. In both cases, we first need to verify the validity of the extracted components using the algorithm developed in details in Appendix 6.3.4.

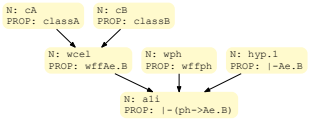
Table 6.3: An analysis of incorrect predictions on the theorem extraction dataset. We observe there are still substantial amount of predictions that lead to valid theorems.

Dataset	Total	Not Tree & Invalid	Tree & Invalid	Tree & Valid
Training	64349	13368	47521	3460
Validation	4766	1175	3238	353
Test	4822	1206	3348	328
`set.mm`	22017	8182	13470	365

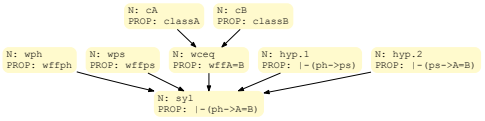
The number of valid theorems from the incorrect predictions on the theorem extraction dataset, and the predictions on `set.mm` are listed under *Tree & Valid* in Table 6.3. We observe that there were a non-trivial amount of predictions that led to valid theorems. Remarkably, we see REFACTOR was able to extract valid theorems in the real human proofs (`set.mm`), despite the fact that human proof distribution may be very different from the training distribution. Adding up all extracted theorems from both approaches, we arrived at 4204 new theorems. We notice that among them, some new theorems were duplicates of each other due to standardization and we kept one copy of each by removing all other duplicates. We also removed 302 theorems extracted on `set.mm` that corresponded to the entire proof tree. In the end, we were left with 1923 unique new theorems with 1907 and 16 from the expanded and original dataset respectively.



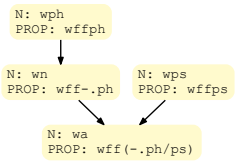
(a) Used 60705 times, from expanded dataset



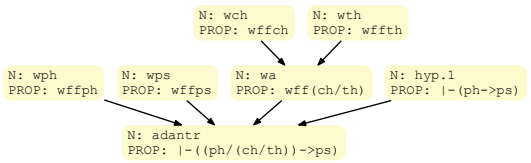
(b) Used 11375 times, from expanded dataset



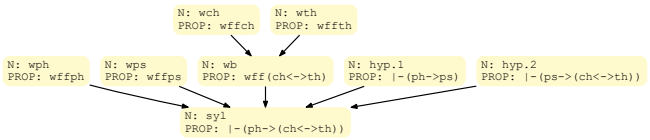
(c) Used 11125 times, from expanded dataset



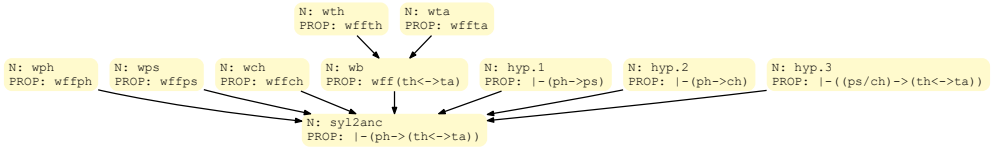
(d) Used 8594 times, from `set.mm`



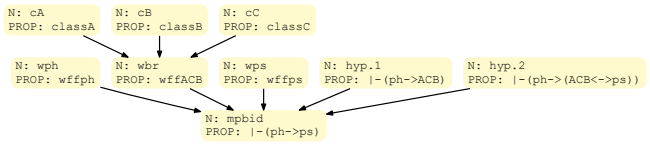
(e) Used 7241 times, from expanded dataset



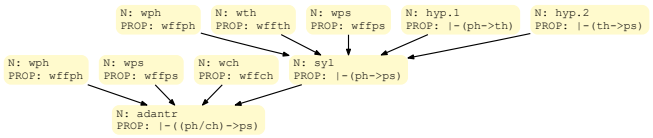
(f) Used 4693 times, from expanded dataset



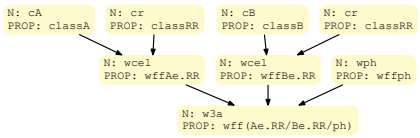
(g) Used 4437 times, from expanded dataset



(h) Used 3428 times, from expanded dataset



(i) Used 3376 times, from expanded dataset



(j) Used 2933 times, from `set.mm`

Figure 6.6: Top 10 most frequently used theorems in refactoring.

Table 6.4: Theorem usage and their contribution to refactoring

	# Theorems Used	Total	Average	Max	Avg. # Nodes Saved	Total # Nodes Saved
Expanded	670	147640	77.4	60705	196.7	375126
Original	14	11736	733.5	8594	2025.8	32413
Total	684	159376	82.9	60705	211.9	407539

In Figure 6.6, we show the top 10 most frequently used new theorems in refactoring. Among them, two are extracted from the original `set.mm` and the rest are extracted from the expanded dataset. It is worth noting that although these theorems generally have fewer than 10 nodes each, they in total contribute to more than 78% of total number of nodes saved in refactoring, suggesting the pervasiveness and reusability of these extracted theorems in `set.mm`.

We also plot the distribution of number of proof nodes of the extracted theorems in Figure 6.5 (c). We can see the newly extracted theorems are of various sizes, spanning almost two orders of magnitudes.

We then computed the number of usages in `set.mm` for each newly extracted theorem, reported in Table 6.4. The average number of uses is 83 times, showing nontrivial utility of these theorems. Notably, the theorems extracted on `set.mm` are even more frequently used – 733.5 times on average. We think that because the human library is already quite optimized, it is harder to extract new theorems from existing proofs. But a successful extraction is likely to be of higher quality as the proof tree input represents a true human proof rather than a synthetically expanded proof.

We additionally performed a more detailed analysis on the predictions, by classifying them into three categories. The first category is denoted by *Non-Tree & Invalid* where the prediction is a disconnected set of nodes and hence it is impossible to form a new theorem. In the second category *Tree & Invalid*, the prediction is a connected component and hence forming a sub-tree, but it still does not satisfy other conditions outlined in our algorithm description to be a valid proof of a theorem. The last category *Tree & Valid* corresponds to a prediction that leads to an extraction of new theorem previously not defined by humans. We present the number of predictions for each category in Table 6.3. Surprisingly, we noticed the model predicted a substantial amount of disconnected components. We hypothesize this may be because our current model makes independent node-level predictions. We believe an autoregressive model has a great potential to fix this problem by encouraging contiguity, a direction which we leave for future work.

6.4.5 Q3a - How much can we compress the existing library using the extracted theorems?

When the newly extracted theorems are broadly reusable, we would expect the proofs in the library could be shortened by using the new theorems as part of the proofs. In this paper, we consider a specific re-writing procedure, which alternates between 1) matching the extracted theorems against the proofs in the library and 2) replacing the matched proportion of the proofs with the application of the new theorems (See more details in the Appendix). We call this procedure the *refactoring* procedure and the resulting shortened proof the *refactored* proof. We want to highlight that compression is only one of the downstream tasks we used to evaluate the usefulness of our extracted theorems. One may pursue a compression objective for this purpose, to find the most frequently appeared fragments across all proofs. Our single-proof prediction approach puts its main focus on

Table 6.5: Proof success rate comparison. New theorem usage for REFACTOR is averaged across 1 and 5 min setting.

Setting	1 min	5 min	New Theorem Usage
Holophrasm [133]	-	14.3%	-
Holophrasm (ours)	11.5%	15.1%	-
REFACTOR	**14.9**%	**17.2**%	43.0%

human preferences and could potentially be combined with compression as future work.

With the 16 new extracted theorems from the original dataset, the new library obtained from refactoring was indeed smaller (See Table 6.4). These new theorems on average saved 2025.8 nodes which is an order of magnitude more than those from the expanded dataset (196.7 nodes). Nevertheless, this shows that extracted theorems from both expanded and human datasets are frequently used in refactoring the theorem library. In total, we were able to refactor 14092 out of 27220 theorems in the MetaMath database. This improvement in compression is striking, as REFACTOR didn't explicitly consider compression as an objective.

6.4.6 Q3b - Are newly extracted theorems useful for theorem proving?

We further demonstrated the usefulness of our new theorems with an off-the-shelf neural network theorem prover, Holophrasm [133]. We trained two Holophrasm provers, one with the original dataset, and the other with the dataset augmented with the newly extracted and refactored proofs.

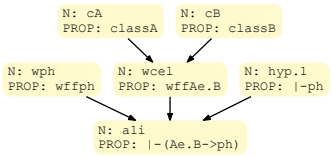
We evaluated the proof success rate in Table 6.5. We used the default values for all hyperparameters of the prover, and we evaluated proof success rates on a hold-out suit of test theorems. We report the results with the time limit of each proof search set to 1 and 5 minutes. Compared to the reported result in Whalen [133] under a 5-minute limit, our re-implementation was able able to obtain a slightly higher success rate (15.1%). It can be seen that by training on the refactored dataset, the prover's proof success rate improved relatively by 14-30% under 1 and 5 min limits, demonstrating the usefulness of REFACTOR in theorem proving.

To investigate how newly extracted theorems contributed to the improvement, we calculated the percentage of proved theorem that used new theorem at least once in its proof, i.e. new theorem usage as shown in Table 6.5. The usage for 1 and 5 min cases are 42.3% and 43.6% respectively, indicating newly extracted theorems were used very frequently by the prover. More remarkably, the newly extracted theorems used in proving test theorems did not concentrate on few theorems as one might predict. Instead, there was a diverse set of newly extracted theorems that were useful in theorem proving: for the 5 min setting, there were in total 141 unique new theorems used for proving test theorems, and the most frequently used one was used 17 times.

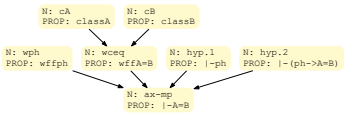
In Figure 6.7, we show the top 10 most frequently used new theorems in theorem proving. All of them are extracted from the expanded dataset. It can be seen that the top 5 mostly used new theorems have fewer nodes than the other 5, suggesting these shorter new theorems are less proof specific and hence are used more frequently than those that are much longer and more applicable in niche proof context.

6.5 Related Work

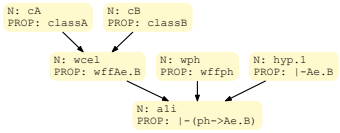
Lemma Extraction Our work is mostly related to the work of Kaliszyk and Urban [62] and Kaliszyk, Urban, and Vyskocil [63]. The authors propose to do lemma extraction on the synthetic proofs generated by Automated Theorem Provers (ATP) on the HOL Light and Flyspeck libraries. They showed the lemma extracted from the synthetic proofs further improves the ATP performances for premise selection. However, their proposed lemma selection methods require human-defined metrics and feature engineering, whereas we propose a novel way to create datasets for training a neural network model to do lemma/theorem selection. Unfortunately, as the Metamath theorem prover is not equipped with ATP automation to generate synthetic proofs, we could not easily compare our method to these past works. We leave more thorough comparisons on the other formal systems to future work.



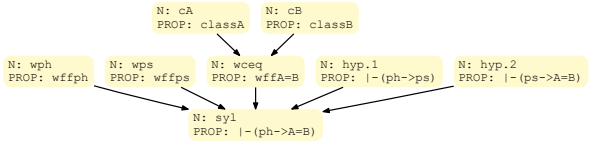
(a) Used 17 times, from expanded dataset



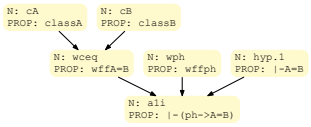
(b) Used 16 times, from expanded dataset



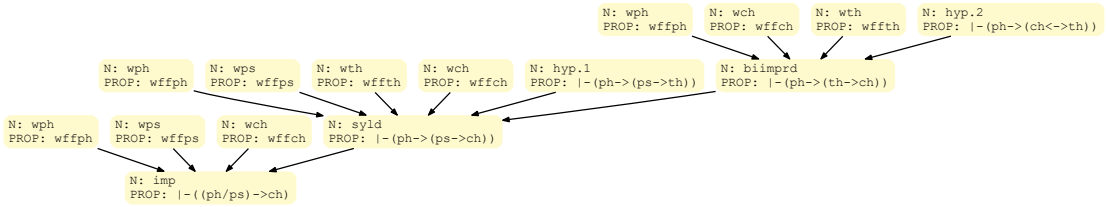
(c) Used 9 times, from expanded dataset



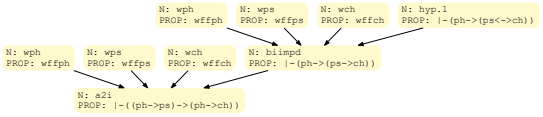
(d) Used 6 times, from expanded dataset



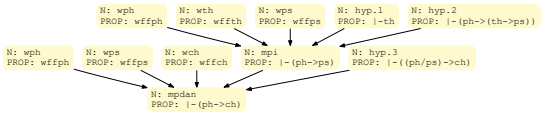
(e) Used 5 times, from expanded dataset



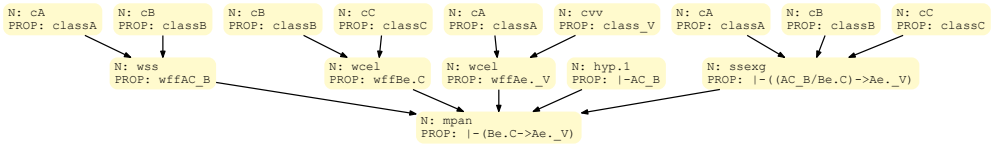
(f) Used 4 times, from expanded dataset



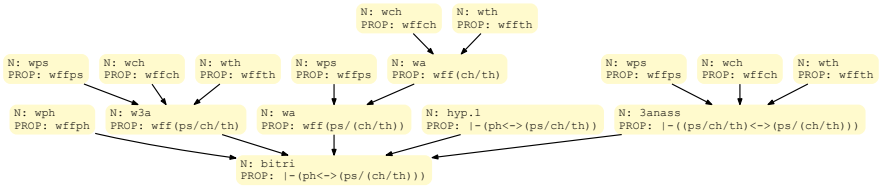
(g) Used 4 times, from expanded dataset



(h) Used 3 times, from expanded dataset



(i) Used 3 times, from expanded dataset



(j) Used 3 times, from expanded dataset

Figure 6.7: Top 10 most frequently used theorems in theorem proving.

Discovering Reusable Structures Our work also is related to a broad question of discovering reusable structures and sub-routine learning. One line of the work that is notable to mention is the Explore-Compile-style (EC, EC2) learning algorithms [25, 32, 33]. These works focus on program synthesis while trying to discover a library of subroutines. As a subroutine in programming serves a very similar role as a theorem for theorem proving, their work is of great relevance to us. However they approach the problem from a different angle: they formalize sub-routine learning as a compression problem, by finding the best subroutine that compresses the explored solution space. However, these works have not yet been shown to be scalable to realistic program synthesis tasks or theorem proving. We, on the other hand, make use of human data to create suitable targets for subroutine learning and demonstrate the results on realistic formal theorem proving. Another related line of work build inductive biases to induce modular neural networks that can act as subrountines [2, 38, 56, 79, 18, 137]. These works usually require domain knowledge of sub-routines for building neural architectures hence not suitable for our application.

6.6 Conclusion

In this paper, we study the problem of extracting useful theorems from mathematical proofs in the Metamath framework. As proofs are represented as *proof trees* in formal systems, we formalize theorem extraction as a node-level binary classification problem on proof trees. We propose one way to create datasets for the problem and additionally develop an algorithm to verify the validity of the prediction. We demonstrate that our best graph neural network model was able to extract unseen human theorems 19.6% of the time. When the model's prediction did not match the human theorem ground truth, we can additionally extract 1907 theorems from the dataset. We further applied the model on the existing Metamath library and found it was able to extract 16 new theorems, each was used 733.5 times on average in the entire Metamath database. After theorem refactoring, those 16 new theorems saved 32413 proof nodes of the entire dataset. Finally, by training the refactored proofs, we show a prover achieved better proof success rate on test theorems.

Our work represents the first proof-of-concept of theorem extraction using neural network models. We see there are various ways to improve the existing model, such as scaling up the model size, or using more powerful architectures such as transformers to autoregressively predict the target, all of which are left to future works. Lastly, we would like to note that our methodology is not only generic for formal mathematical theorem extraction, but also has the potential to be applied to other applications, such as code refactoring.

Part III

Techniques

Chapter 7

LIME:

This part of the thesis focuses on developing better techniques for neural networks in mathematical reasoning. We mainly focuses on three problems: 1. neural architecture inductive biases for reasoning, 2. high-level flexible reasoning, and 3. data scarcity problem in formal libraries. In this chapter, we first take a look at how to design better neural network inductive biases for reasoning. Inspired by Peirce's view that deduction, induction, and abduction are the primitives of reasoning, we design three synthetic tasks that are intended to require the model to have these three abilities. We specifically design these tasks to be synthetic and devoid of mathematical knowledge to ensure that only the fundamental reasoning biases can be learned from these tasks. This defines a new pre-training methodology called "LIME" (Learning Inductive bias for Mathematical rEasoning). Models trained with LIME significantly outperform vanilla transformers on four very different large mathematical reasoning benchmarks. Unlike dominating the computation cost as traditional pre-training approaches, LIME requires only a small fraction of the computation cost of the typical downstream task.

This chapter is largely based on the work: *LIME: Learning Inductive Bias for Primitives of Mathematical Reasoning* by Yuhuai Wu, Markus Rabe, Wenda Li, Jimmy Ba, Roger Grosse and Christian Szegedy [139], published in ICML of 2021.

7.1 Overview

Inductive bias is essential for successful neural network learning. Many of the breakthroughs in machine learning are accompanied by new neural architectures with better inductive biases, such as locality bias in convolutional neural networks [70], recurrence and memory in LSTMs [55], and structural bias in graph neural networks [107]. However, explicitly encoding inductive biases as new neural architectures can be difficult for abstract concepts such as *mathematical reasoning*. Attempts to design elaborate architectures for reasoning often fall short of the performance of the more generic transformer architecture. In this work, we aim to avoid the search for new architectures and investigate whether one can *learn useful inductive bias for mathematical reasoning through pretraining*.

Large-scale unsupervised pretraining of language models revolutionized the field of natural language processing (NLP), improving the state-of-the-art in question answering, name entity recogni-

tion, text classification, and other domains, e.g. [100, 26, 144, 77, 101, 15]. As a result, pretraining has become a common practice for modern neural network based NLP. A popular explanation for the benefit of pretraining is that the model can learn world knowledge by memorizing the contents of the natural language corpus, which can be useful in downstream tasks, such as question answering and text classification. However, there is another potential advantage of pretraining—it may distill inductive biases into the model that are helpful for training on downstream tasks [15, 130]. We focus on the latter and design pretraining tasks that are intentionally devoid of world knowledge and only allow the model to learn inductive bias for reasoning.

Inspired by the logician Charles Peirce [94], we consider the following three reasoning primitives:

1. **Deduction**: the ability to deduce new truths from given facts and inference rules.
2. **Induction**: the ability to induce general inference rules from a set of known facts.
3. **Abduction**: the ability to explain the relationship between the evidences and inference rules.

To endow the models with an inductive bias for mathematical reasoning, we design a synthetic task for each of the three reasoning primitives. We hypothesize that the transformer networks are flexible enough to learn strong inductive bias from the three synthetic reasoning tasks, which helps to improve the performance on downstream tasks. Although such inductive bias may be useful in general reasoning tasks (e.g., NLP tasks), in this work, we focus on mathematical reasoning benchmarks, for which we expect to observe the largest gains. We call training on these tasks LIME – an acronym for "Learning Inductive Bias for Mathematical rEasoning". Note that there is only a limited amount of pretraining data available for formal mathematical benchmarks, therefore the study of generic pre-training techniques is particularly important for the success of machine learning in mathematical reasoning.

We demonstrate that LIME pretrained models provide significant gains across four large mathematical reasoning benchmarks: IsarStep [73], HOList Skip-tree [98], MetaMathStep [97], and LeanStep []. Notably, LIME improved the top-1 accuracy from 20.4% to 26.9% IsarStep, and from 15.5% to 29.8% on LeanStep. Compared to traditional pretraining tasks, LIME has two major differences. First, LIME requires only a fraction of the computational cost of downstream tasks. With only about two hours of training on a single modern GPU, one already obtains all the benefits, in contrast to days of training on a large natural language corpus with hundreds of GPUs/TPUs. Secondly, LIME does not load the input embeddings or the weights in the output layer for finetuning on downstream tasks. This allows one to use the same pretrained model for a variety of downstream tasks, which can have vastly different vocabularies due to language or tokenization differences.

7.2 Methods

In this section, we first discuss the primitives of reasoning, inspired by Peirce's views, and design one synthetic task for each reasoning primitive.

7.2.1 Reasoning Primitives

In Peirce's view, there are exactly three kinds of reasoning: deduction, abduction, and induction. Deduction is known as the workhorse for mathematics. It is the process of deriving new facts by

applying logical inference rules to known facts or premises. On the other hand, abduction and induction can be thought of as the inverses of deduction. If we call the premise used in deduction as *Case*, its logical rule as *Rule*, and its conclusion as *Result*, then abduction is equivalently the inference of a Case from a Rule and a Result, while induction may be said to be the inference of a Rule from a Case and a Result. We summarize the three reasoning primitives in the following table:

Reasoning Primitives	**Inference Map**
Deduction	Rule, Case $\rightarrow$ Result
Abduction	Rule, Result $\rightarrow$ Case
Induction	Case, Result $\rightarrow$ Rule

To give an example, we let Rule be "All the beans in this bag are white", Case be "These beans are from this bag", and Result be "These beans are white". Deduction is to derive the fact that these beans are white (Re) from knowing all the beans from this bag are white (R) and these beans are from this bag (C). Abduction explains why the beans are white (Re) from knowing that all the beans in the bag are white (R) – because these beans must be from the bag (C). Lastly, induction aims to provide a general principle to observing the fact that the beans are white (Re) and they come from this bag (C), which is that all the beans in the bag must be white (R). We refer to Peirce [94] and Bellucci and Pietarinen [8] for more elaborate discussions on the primitives of reasoning.

Mathematical reasoning exhibits nontrivial uses of these reasoning primitives. Deduction happens when one needs to derive new valid statements from the given premise (Case) and theorems in the library (Rule). Abduction is used to postulate conjectures from the known facts and theorems, allowing one to decompose the challenging theorem into subgoals for proof. Induction, the ability to extract general principles from known facts and theorems is also one of the major activities of mathematical reasoning. It is used when one derives theorems from special cases and proposes new definitions and general frameworks to encapsulate existing knowledge.

7.2.2 LIME Synthetic Tasks For Reasoning Primitives

We design three synthetic tasks inspired by the three reasoning primitives. As discussed in the previous section, all of the reasoning primitives consist of three essential elements: Rule, Case, and Result. Inspired by this, we first design a method to generate those elements. Once they are generated, we can construct tasks that predict one element from the other two. In the following, we describe one simple way to generate those three elements, though we acknowledge that there are many other possible approaches.

We require two types of symbols: 1. *math symbols*, 2. *rule symbols*. In general, these symbols can take any forms (e.g., integer representations). But for the ease of discussion, we will think of math symbols as the union of those operators used in mathematics (e.g., "$+ - * = ()\&$") and lower case letters (e.g., a, b, c ...), and rule symbols as upper case letters (e.g., A, B, C ...). We now construct Rule, Case, and Result in order:

1. **Rule** is a randomly sampled string that consists of i) rule symbols and ii) math symbols. The length of the string is randomly sampled from a range. For instance, a randomly sampled rule can be: $A * A + B = C$ with rule symbols A, B, and C.

2. **Case** is a dictionary that represents substitutions. For each rule symbol used in the Rule string, we sample a random string of random length that consists of math symbols. This forms a dictionary, whose keys are all rule symbols, and the values are the corresponding sampled string. To illustrate, following the previous example, for each A, B and C, we sample a random string to form a dictionary as: $\{A : a,\ B : b,\ C : d + e\}$.

3. **Result** is the outcome of the substitution. For each rule symbol in the Rule string, we replace it with the corresponding value stored in the Case dictionary. This gives rise to the Result string. As per the previous example, we now substitute A with a, B with b, and C with $d + e$ into the Rule string, generating the Result string: $a * a + b = d + e$.

After Rule, Case, and Result are generated, we can construct three tasks for deduction, abduction, and induction respectively. We define the three synthetic tasks as follows:

- `Deduct`: **Source:** Rule string and Case dictionary.

  Target: Result string.

- `Abduct`: **Source:** Rule string and Result string.

  Target: Case dictionary.

- `Induct`: **Source:** Case dictionary and Result string.

  Target: Rule string.

We also consider a task called `Mix`, which is a uniform mix of three tasks. Namely, during generation, we randomly select a task and sample an example from that task. To formulate them as sequence to sequence tasks, we represent the Case dictionary also as a string, e.g., "$\{A : a,\ B : b,\ C : d + e\}$". An example of `Abduct` using the examples of Rule, Case, and Result above is to predict the target $\{A : a,\ B : b,\ C : d + e\}$ from the source $A * A + B = C$ `<s>` $a * a + b = d + e$.

Pre-training on our synthetic tasks can be seen as a form of skip-component learning. There are three essential components: Rule, Case and Result, and we skip one of them and use the remaining two elements to reconstruct the missing one. Past work has shown that learning to predict missing words [26], subsequences [115, 101], or subtrees [98] are strong pre-training tasks.

7.2.3 Symbol-Agnostic Representation

In order to solve the synthetic tasks, the model needs to distinguish which set of symbols can be substituted (rule symbols). As a result, the model may memorize information about the symbols that is irrelevant to the inductive biases encoded in the task. To prevent such memorization, we propose a way to make the synthetic tasks agnostic to the choice of symbols.

We first note that the choice of symbols is irrelevant to our synthetic tasks. To avoid symbol-specific memorization, for each training and evaluation example, we randomly sample two sets of symbols to be used in Rules and in the rest of the example. But for the `Abduct` task, the model needs to know which symbols are replaced by the Rule part of the example and which symbols are in the Result language. We simply list the split of the symbols used in the example at the beginning of the input string, marked by two special symbols, `<Rule>` and `<Math>`. They are followed by the

Algorithm 3

function GENERATE_TUPLE(Vocabulary size S)
  Vocabulary $\mathcal{V} \leftarrow \{1, 2, \ldots, S\}$. ▷ Use an integer representation of symbols.
  Math symbol set $\mathcal{M} \leftarrow$ SAMPLE($\mathcal{V}$, n=44, replacement=False). ▷ Sample 44 distinct symbols.
  Rule symbol set $\mathcal{R} \leftarrow$ SAMPLE($\mathcal{V} \backslash \mathcal{M}$, n=20, replacement=False). ▷ Sample 20 distinct symbols.
  Rule $R \leftarrow$ SAMPLE($\mathcal{M} \bigcup \mathcal{R}$, n=RANDOM(5,20), replacement=False). ▷ Sample a sequence of symbols of length between 5 and 20.
  Case dictionary $C \leftarrow \{\}$.
  for s in $\mathcal{R}$ **do**
    Case dictionary $C[s] \leftarrow$ SAMPLE($\mathcal{M}$, n=RANDOM(2,8), replacement=True). ▷ Sample a sequence of symbols for each rule symbol, of length of length between 2 and 8.
  end for
  Result $R' \leftarrow$ Rule R. ▷ Set result string R' to be the same as rule string R.
  for s in $\mathcal{R}$ **do**
    SUBSTITUTE(R', s, $C[s]$). ▷ Substitute every rule symbol s in result string R' with previously randomly sampled string $C[s]$.
  end for
  return Math symbol set $\mathcal{M}$, Rule symbol set $\mathcal{R}$, Rule R, Case C, Result R'.
end function

original source string. The target string remains unchanged. For example, the previous example in the `Abduct` task becomes,

Source: `<Rule>` $A\ B\ C$ `<Math>` $*\ +\ =\ a\ b\ d\ e$ `<s>` $A * A + B = C$ `<s>` $a * a + b = d + e$

Target: $\{A : a,\ B : b,\ C : d + e\}$

In our implementation, we use integers to represent symbols. Specifically, for each example, we sample two disjoint sets of integers from the set $\{1, \ldots, S\}$ to represent the math symbols and the rule symbols, where S is the size of the vocabulary. In our experiments, we sample 44 math symbols and 24 rule symbols for each problem. The complete pseudo-code of generating the symbols, Rule, Case, and Result for one task example is provided in Algorithm 3.

7.2.4 Other synthetic task variants

Besides the three main tasks, we also explored other variants of the pre-training tasks, results shown in Section 7.4.1.

`Rewrite` and `Rewrite_multistep`

We propose a rewrite task, inspired by the rewrite tactic used in interactive theorem provers. The `Rewrite` task requires the model to rewrite a string according to a rule transformation. One example of the task is:

Source: $a + b - c$ `<s>` $A + B = B + A$

Target: $b + a - c$

"$A + B = B + A$" is the rule transformation, which is applied to the LHS string "$a + b - c$". The model needs to predict the RHS string as the result of the rule application, i.e., $b + a - c$. Besides rule symbols and math symbols, we also require the third set of symbols, named as "string symbols". For the ease of our discussion, we we will think of math symbols as the union of those operators

used in mathematics (e.g., "$+ - * = ()\&$"), rule symbols as upper case letters (e.g., A, B, C ...), and string symbols as lower case letters (e.g., a, b, c ...). We first sample a random string as the LHS string, consisting of math symbols and string symbols (e.g., $a+b-c$). We sample a sub-string of the LHS string, and replace the string symbols in the sub-string with rule symbols. For example, we sample and obtain the substring $a+b$ from $a+b-c$, and we replace a, b with rule symbols A, B. This then forms the LHS of the rule transformation, $A+B$, with the substitution dictionary $\{A:a, B:b\}$. We then sample the RHS of the rule transformation from the union of rule symbols A and B, and all math symbols, e.g., $B+A$. This gives the rule transformation $A+B=B+A$. We substitute the value of the substitution dictionary for each rule symbol in the RHS rule, and then substitute back to the original LHS string to obtain $b+a-c$. The task example is constructed by using the LHS string and the rule transformation as the source input, and use the result of the rule transformation as the target.

We further introduce a multi-step version of the rewrite task: `Rewrite_multistep`. In this task, the source may contain more than one rewrite rule, and the target is the result of applying all the rewrite rules in a sequence. This task is motivated from the need to perform multi-step planning in mathematical reasoning tasks. During pre-training, for each training example, we uniformly sample the number of rewrite steps from 1 to 5.

Other variants of `Induct` Task

We introduce three other variants of the `Induct` task.

1. `Induct_v2`: We move the Case dictionary from the source input to the target output. This makes the task significantly harder, which requires the agent to synthesize a rule and a possible explanation (Case) to explain the Result.

2. `Induct_v3`: Instead of providing the Case dictionary, we provide two Result strings, coming from the same Rule. Namely, we sample two Case dictionaries, and applying each to the Rule string to obtain two Result strings. Both Result strings are used as source, and the target is the Rule string.

3. `Induct_rewrite`: We also create a "induction" version of the `Rewrite` task. In this task, the source is the LHS string concatenated with the RHS string, that is the result of the rewrite. The target is the rewrite rule that is used to do the rewrite.

7.3 Experiments

In this section, we present results on four large mathematical reasoning tasks that are especially useful in the context of automated theorem proving. Our results show significant gains in learning inductive biases from synthetic tasks. We have selected four tasks to cover various different styles of interactive theorem provers: The HOL-Light (skip-tree) corpus was created from very high-level tactic-based proofs, but it is less interpretable than IsarStep's declarative style corpus. We also evaluate the next proof-step prediction task on the `set.mm` library of MetaMath, which consists of very granular, basic proof steps. Namely, the proof steps are more predicable and average proof lengths have significantly increased.

7.3.1 Experiment Details

LIME Pretraining We generate datasets of our synthetic tasks for pretraining: `Deduct`, `Abduct`, `Induct`, `Mix`. For pretraining of IsarStep, we used a vocabulary size S of 1000. For the other two downstream tasks, we used a vocabulary size of 100. The reason we used different vocabulary sizes was that we found (cf. appendix) the discrepancy in vocabulary size affects the performance of a downstream task if it has a very large vocabulary size (IsarStep has 28K). We use 44 math symbols and 24 rule symbols. The length of the Rule string is sampled from 5 to 20, the length of the string for each substitution (the values of Case dictionary) is sampled from 2 to 8. We used word-level tokenization for all the tasks. We pretrained the model for 20K updates. For tasks with larger vocabulary size (i.e., 1000), we found the learning became more difficult. Hence we used a curriculum learning scheme: we first trained the model for 10K steps on the same task with a vocabulary size of 100, then continue training for another 10K step on vocabulary size of 1000. The pretraining was done on a single Nvidia Tesla T4 GPU with 4 CPU cores for 2 hours. We set the maximum number of tokens in a batch to 4096, and accumulate four batches of gradients for one parameter update. We used the Adam optimizer [66] with learning rate $3 \cdot 10^{-4}$. We used a dropout rate of 0.1 and label smoothing [119] with a coefficient 0.1.

Fine-tuning For all the downstream tasks in this section, when loading the pretrained models for fine-tuning, we do not load in the vocabulary embeddings nor the output layer weights. For the downstream task IsarStep and MetaMathStep, we used four Nvidia Tesla T4 GPU with 16 CPU cores for training. We set the maximum number of tokens in a batch to 4096, and accumulated four batches of gradients for one parameter update. We trained the model for 200K updates. We used the Adam optimizer, and we searched over the learning rates $\{3 \cdot 10^{-4}, 7 \cdot 10^{-4}\}$, and warmup steps $\{4000, 8000\}$. We used a dropout rate of 0.1 and label smoothing with a coefficient 0.1. For the HOList skip-tree task, we used TPUs for running the experiments. We used a batch size of 256 sequences and trained the model for 1 million updates.

Architecture All experiments used the transformer base model from [**transformer17**], i.e. 512 hidden size, 2048 filter size, 8 attention heads. For the IsarStep and MetaMathStep task, we used 6 layers for both the encoder and decoder, implemented using fairseq [89]. For the HOList skip-tree experiment, we used a somewhat modified transformer architecture with 8 encoder and 4 decoder layers of the same size as above in which the self-attention and attention over the encoder output were merged.

Evaluation During training, we kept track of the best validation tokenized BLEU score [1], and we used the model with validation BLEU for evaluation on the test set. We report top-1 and top-10 accuracies. We consider an output sequence as correct if it matches the target sequence exactly. We performed a beam search with width 10. The top-1 accuracy is then defined as the percentage of the best output sequences that are correct. The top-n accuracy is defined as the percentage of target sequences appearing in the top n generated sequences.

[1] https://github.com/pytorch/fairseq/blob/master/fairseq/tasks/translation.py#L396

Table 7.1: Test top-1, top-10 (%) accuracy on the IsarStep task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain [73]	20.4	33.1
HAT [73]	22.8	35.2
LIME `Deduct`	24.7	37.7
LIME `Abduct`	26.7	**41.0**
LIME `Induct`	23.9	38.8
LIME `Mix`	**26.9**	40.4

Table 7.2: Test top-8 Accuracy on Skip-Tree HOList (%).

Model	Equation completion	Hard type inference	Missing assumptions	Easy type inference
No pretrain [98]	46.3	95.0	41.8	95.9
LIME `Deduct`	50.3	94.8	**47.9**	97.0
LIME `Abduct`	48.4	94.8	46.1	96.3
LIME `Induct`	44.8	94.9	42.6	96.4
LIME `Mix`	**51.7**	**95.6**	46.1	**97.6**

7.3.2 IsarStep

The IsarStep task is taken from [73]. IsarStep is a task of predicting the missing intermediate propositions given surrounding propositions to bridge the gap between the goal and the current state of the proof. The dataset was mined from the public repository of formal proofs of the Isabelle proof assistant (Paulson, 1994). Unlike HOList and MetaMath, IsarStep contains mostly declarative proofs, a proof style close to humans' prose proofs. The dataset has a broad coverage of undergraduate and research-level mathematics and computer science theorems. There are 820K, 5000, 5000 sequence pairs for the training, validation, and test sets with a maximum of 800 tokens in source sequences and 200 tokens in the target sequences. Following [73], during training, we use 512 as the maximum length for both the source and target, and truncated those that exceed the length to 512. For reporting, we evaluate all 5000 test examples regardless of their lengths.

The results on the IsarStep task for four pretrained models and the baseline transformer model without pretraining is shown in Table 7.1. We also include another baseline, HAT transformer introduced in [73], which is a specially designed hierarchical transformer architecture tailored to this task. We see the pretrained model achieved substantial improvement over the model trained from scratch as well as HAT. Notably, the model that was pretrained on `Abduct` improved the top-10 accuracy from 33.1% to 41.0%, for almost 8% absolute improvement. The model pretrained on `Mix` performed the best on top-1 accuracy, improving the baseline by 6.5% accuracy. We also showed the validation BLEU scores along training in Figure 7.1. We can see that the pretrained models learned much faster than the model trained from scratch. With around 50K steps of updates, the pretrained model already obtained better BLEU scores than the best score achieved by the un-pretrained model. Moreover, since the downstream task requires 200K steps of training with 4 GPUs, the amount of computation spent on pretraining is only 2.5% of the downstream task, strongly demonstrating the efficiency of the proposed pretraining method.

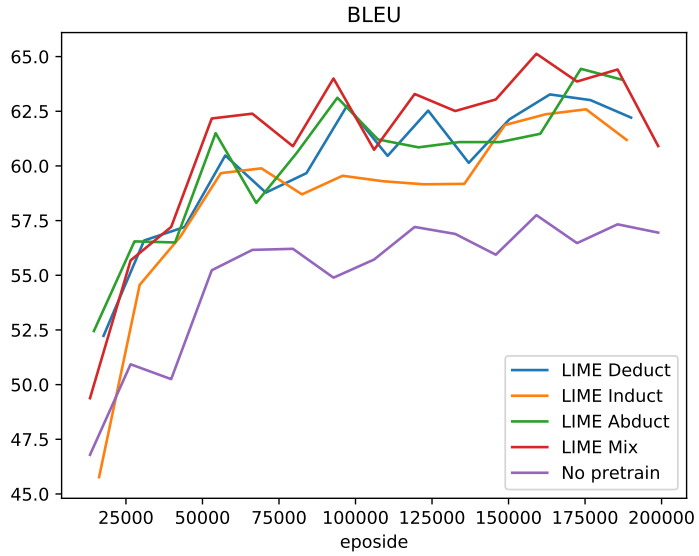


Figure 7.1: Validation BLEU along with training on the IsarStep task.

Table 7.3: Test top-1, top-10 (%) accuracy on the MetaMathStep task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain	67.7	76.5
LIME `Deduct`	68.8	77.4
LIME `Abduct`	68.8	76.1
LIME `Induct`	**69.9**	**78.0**
LIME `Mix`	69.1	77.9

7.3.3 HOList Skip-Tree

As the second mathematical reasoning benchmark, we consider the HOList skip-tree evaluation tasks by Rabe et al. [98]. These tasks include two variants of type inference, predicting under which assumptions theorems hold, and completing equalities. All source expressions for these tasks are taken from the validation set of the theorem database of the HOList proof logs [5]. The evaluations are done on a random sample of 1000 instances from the full evaluation sets. We initialized the model parameters with the pretrained weights and then repeated the experiments by Rabe et al. [98]. That is, we trained the models for up to 1M parameter updates on the training set with batch size 256 and repeat the evaluation every 100K steps. In Table 7.2 we present the best result from these 10 evaluation runs. We see a significant improvement in these reasoning tasks when the models are initialized with the pretrained weights. Notably, on equation completion and missing assumptions task, we improved the beam search (with width 8) exact match rate performance from 46.3% to 51.7% and 41.8% to 47.9%. Note that this is despite the amount of pretraining compute cost being negligible: it takes less than 1 percent of the cost of the downstream task training. Pretraining used 1/20 number of the update steps (50K vs 1M) with 8 (and 4) times smaller batches (pretraining has much shorter sequence lengths, 128 vs. 1024 and 512, respectively).

Table 7.4: Test top-1, top-10 (%) accuracy on the LeanStep unseen lemma prediction task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain	15.8	27.4
LIME `Deduct`	25.8	38.0
LIME `Abduct`	26.0	38.6
LIME `Induct`	25.0	38.2
LIME `Mix`	**29.8**	**41.8**

7.3.4 MetaMathStep

Compared to other ITPs, MetaMath is a low-level proving system: each proof step makes only a small step towards the goal. As such, each proof contains many more proof steps than in other ITPs: with 37,000 theorems in the human-written theorem library, there are around 3 million proof steps. We extract the proof steps and use them to construct a sequence-to-sequence task following Polu and Sutskever [97] (their proof step training objective).

In this task, the model is asked to generate PROOFSTEPS given a GOAL, namely, the GOAL string is the source input, and PROOFSTEPS is the target output. We follow [97] and use their string representation for the GOAL and the PROOFSTEPS. Instead of using subword tokenization in Polu and Sutskever [97], we use a character-level representation for our task. Following Polu and Sutskever [97], we split theorems into train/valid/test theorems of size 35K, 1K, 1K, and associate all proof steps of a theorem with that split. For each dataset, we filter examples with lengths longer than 1024. This reduced the total number of proof steps to 1.4 million. For validation and test set, we randomly sample 3000 examples out of 40K (after filtering) and perform validation and test evaluations on them. In Table 7.3 we present the impact of LIME on MetaMathStep. We also observe gains from LIME on this dataset, with the model trained on `Induct` task achieving 2.2% top-1 and 1.5% top-10 test accuracy improvement. Similarly, as for the IsarStep task, the computation spent on pretraining is only 2.5% of the downstream task.

7.3.5 LeanStep: Unseen Next Lemma Prediction Task

Lastly, we look at a mathematical benchmark based on Lean 3 theorem prover. Lean has an extremely active community and is host to some of the most sophisticated formalized mathematics in the world, including scheme theory [17], forcing [50], perfectoid spaces [16], and condensed mathematics [109]. We extracted a similar style of dataset as MetaMathStep from Lean, that is, we predict the next lemma to apply given the current goal state (or commonly known as the tactic state in Lean). Unlike MetaMathStep, we focus on predicting unseen lemmas that have not been seen during training time. Namely, in this task, we evaluate the model's capability of conjecturing a novel lemma string given a goal. Specifically, we extracted 498,624 number of goal, next lemma pairs from Lean mathlib library [21]. We found there are 34,867 lemmas that appeared only once in the entire dataset. We then randomly sampled 8k of lemmas from this set and used the corresponding goal lemma pairs for the validation and the tests (each $4k$). As such, during validation and testing, the model needs to predict lemmas that have not been seen during training. We present the results on LIME and the baseline in Table 7.4. We observed a huge gain with LIME pretraining. Remarkably,

Table 7.5: Test top-1, top-10 (%) accuracy on the IsarStep task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain [73]	20.4	33.1
HAT [73]	22.8	35.2
LIME `Deduct`	24.7	37.7
LIME `Abduct`	26.7	41.0
LIME `Induct`	23.9	38.8
LIME `Mix`	26.9	40.4
LIME `Rewrite`	26.0	38.6
LIME `Rewrite_multistep`	**28.6**	**43.9**
LIME `Induct_v2`	25.6	39.8
LIME `Induct_v3`	25.0	38.8
LIME `Induct_rewrite`	25.8	39.5

Table 7.6: Comparisons to other pretraining tasks on IsarStep task.

Model	Top-1 Acc.	Top-10 Acc
No pretrain [73]	20.4	33.1
LIME `Mix`	26.9	40.4
Pretrain on MetaMathStep	23.1	35.7
Pretrain on WMT En-De	17.2	30.3

LIME `MIX` doubled the top-1 accuracy compared to the baseline unpretrained model, improving the accuracy from 15.8% to 29.8%.

7.4 Ablation Studies

7.4.1 A full comparison of all synthetic tasks

In this section we present a full comparison for all synthetic tasks. We followed the training protocol in Section 7.3.1 and evaluate all pretrained variants on IsarStep. The results are reported in Table 7.5. We can see that the `Rewrite_multistep` achieved the best performance across all synthetic tasks, surpassing the baseline by 8.2% for Top-1 accuracy and 10.8% for Top-10 accuracy. This indicates the inductive bias for long horizon reasoning encoded in `Rewrite_multistep` may be useful for the reasoning task. We leave investigations of improving pretraining tasks beyond LIME to the future works.

7.4.2 Pretraining on Formal Reasoning and Natural Language Tasks

Here we investigate how LIME compares to pretraining on natural language or existing formal reasoning datasets. In this set of experiments, we pretrained three models on `Mix`, MetaMathStep, and on the WMT 2016 English-to-Germany (WMT En-De) translation task, and then we fine-tuned and evaluated these models on the IsarStep task. We pretrained the model on MetaMathStep and WMT EN-DE for 200K steps with 4 GPUs, which is 40 times more computation spent than on

Table 7.7: Pretraining on IsarStep for the MetaMathStep task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain	67.7	76.5
LIME `Mix`	69.1	77.9
Pretrain on IsarStep	67.0	76.1

Table 7.8: Whether one needs to load vocabulary embeddings and output layer weights on IsarStep tasks.

Model	Top-1 Acc.	Top-10 Acc
No pretrain [73]	20.4	33.1
LIME `Mix`	26.9	40.4
LIME `Mix` + Loading All Weights	26.7	40.6

LIME. Due to the mismatch between vocabularies of the pretraining task and the downstream task, we do not load the vocabulary embeddings nor output layer weights. The results in Table 7.6 show that pretraining on MetaMathStep did provide gains, though significantly smaller than gains provided by LIME `Mix`, despite their 40 times higher computational cost. Moreover, pre-training on WMT translation had even a negative effect on the performance. We also conducted an analogous experiment with an evaluation on the MetaMathStep. The result is shown in Table 7.7. In contrast to MetaMath helping IsarStep, we see that pretraining on IsarStep task did not help the downstream task MetaMathStep. We hypothesize that this could be due to MetaMathStep task is closer to the LIME tasks than IsarStep, and hence providing more gains than the opposite direction.

7.4.3 Do we need vocabulary embeddings for fine-tuning?

As mentioned earlier, we did not load in the vocabulary embeddings from the pretrained models when we switched to fine-tuning on downstream tasks. Even without loading the vocab embeddings, the pretrained models still improved the performance. In this ablation study, we investigate how much this decision has affected the results and whether vocabulary embeddings can help improve the performance even further. We performed the comparisons on IsarStep. The task contains a token vocabulary of size 28336. We generated new synthetic tasks for the same vocabulary size, such that we can load the vocabulary embeddings and output layers when initializing the model for IsarStep. Table 7.8 shows that this led to similar performance. This aligns with our expectation that the model should not learn content specific knowledge that is potentially stored in the vocabulary. These weights turn out to be non-essential for the final performance, supporting the evidence that the transformer learns inductive biases from the pretraining task.

7.4.4 Does LIME help LSTMs?

In this section, we investigate if LIME also helps other architectures than transformers. In particular, we applied LIME to two LSTM based architectures: 1. vanilla LSTM, 2. LSTM with attention mechanism. The vanilla LSTM is a stacking LSTM with 4 layers, each with 1000 cells, and 1000-dimensional embeddings. The LSTM with attention architecture is taken from [78], also with 4

Table 7.9: Comparing LIME's benefits on LSTMs on the IsarStep Task

Model	Top-1 Acc.	Top-10 Acc.
LSTM	5.5	11.3
LSTM + LIME `Abduct`	6.9	14.3
LSTM + attention	12.3	22.7
LSTM + attention + LIME `Abduct`	13.4	26.3
Transformer	20.4	33.1
Transformer + LIME `Abduct`	26.7	41.0

layers, 1000 cells and 1000-dimensional embeddings. We evaluate on the IsarStep task, and compared a model trained from scratch and a model pre-trained on LIME `abduct` task. We used the same training protocol as described in 7.3.1. The results are shown in Table 7.9, along with the results on transformer. We observe that LIME improved LSTM as well as LSTM with attention, but the improvements were small compared to transformer. Specifically, if we compare Top-1 accuracy, we can see that LIME improved LSTM from 5.5% to 6.9%, LSTM with attention from 12.3% to 13.4%, and transformer from 20.4% to 26.7%. This observation is aligned with our hypothesis that the transformer is a malleable architecture and hence it is capable of learning architectural inductive biases from datasets. This is mainly attributed to the potential of learning dynamic attention graphs in self-attention layers. We note that this still warrants further investigation as the performance of these architectures are not at the same level, and that may also lead to different improvements.

7.4.5 Does the vocabulary size matter?

In this section, we investigate whether the vocabulary size S in the synthetic task generation algorithm has an effect on the performance. We used the `REWRITE` task for the experiment in this section. We generated datasets of various vocabulary sizes, 100, 512, 1000, 5000, 25000. We used the same curriculum learning for pre-training as described in 7.3.1 on larger vocabulary sizes: first training on the `Rewrite` task of vocabulary size 100 for 10K steps, then training on each individual dataset for another 10K steps. We compare the performance on the downstream task Isarstep. The results are presented in Table 7.10. We see that when the vocabulary size is equal or larger than 512, the performance were similar. The smallest vocabulary size 100 obtained the worst performance among all, and all the other four models achieved similar BLEU scores. The model trained on the largest vocabulary achieved best performance on top-1 accuracy and top-10 accuracy. The results show there is a non-trivial effect of the vocabulary size of the synthetic task to the performance of the downstream task. Hence we use vocabulary size of 1000 for all the experiments in the main paper. We leave investigations of the causes to future work.

7.5 Related Work

Language Model Pretraining The advent of the transformer architecture [123] and the BERT style pretraining [26] represented a huge improvement in the quality of language modeling. Since then, an explosion of research activity in the area pushed the quality of language models through better pretraining tasks. Where BERT [26] masks out a fraction of the input tokens, later works

Table 7.10: Vocabulary sizes' effects on the IsarStep task.

Model	Top-1 Acc.	Top-10 Acc.
No pretrain	20.4	33.1
LIME on `Rewrite`, $S = 100$	24.1	37.5
LIME on `Rewrite`, $S = 512$	25.4	38.8
LIME on `Rewrite`, $S = 1000$	26.0	38.6
LIME on `Rewrite`, $S = 5000$	25.8	38.5
LIME on `Rewrite`, $S = 25000$	27.4	40.9

demonstrated the advantages of masking out subsequences [115, 27, 61, 101, 22] and whole sentences [146].

Besides the choice of pretraining tasks, the scale of language models is also an important factor. Language models improve in quality and develop new abilities as they grow larger while trained on the same data [100, 101, 15].

Inductive Biases in General There have been works studying learning inductive biases in other contexts. In particular, McCoy et al. [80] studied whether one can learn linguistic inductive biases on synthetic datasets via meta-learning. Papadimitriou and Jurafsky [91] shows inductive biases learned in music data can be useful for natural language. They further designed several synthetic tasks and showed similar kind of improvements for natural language tasks. From a more theoretical point of view, Xu et al. [142] formalize an aspect of inductive (architectural) bias under the context of GNNs, with a notation called *architectural alignment*. The architecture is aligned when the architecture can perfectly simulates the ground truth solution. But their work is limited to showing alignment in combinatorial problems, whose ground truth solutions are known. In contrast, our work tries to learn architectural bias by relying on the flexible Transformer architecture and training on synthetic datasets.

Inductive Biases for Mathematics Previous work studying inductive biases for logical reasoning has focused on encoding bias in the neural architecture. Initial works focused on encoding the tree structure of expressions using TreeRNNs [34]. Graph neural networks are shown to provide a much stronger performance than tree models in premise selection [127] and theorem proving [90]. GNNs also scale to larger formulas in SAT [111, 110, 49], QBF [71], and #SAT [122]. Crouse et al. [23] have shown that pooling mechanisms can have an impact on the performance of GNNs on logical formulas as well. Closely related, Hellendoorn et al. [53] have shown that it can be helpful to hard-code the tree structure of programs in the attention mask of transformers. Schlag et al. [108] developed an architecture for encoding relational information using tensor product representation for mathematical reasoning.

7.6 Does LIME encode Induction, deduction and abduction?

Although LIME has shown to achieve substantial improvements across various benchmarks, it is not entirely clear that the specific synthetic tasks necessarily enforce the reasoning ability of induction, deduction and abduction. We would like to note that deduction, induction, and abduction are

high-level and philosophical concepts, and serve only as an inspiration for us to design the synthetic tasks. We do not expect the model will necessarily learn exactly these three capabilities. After all, we have chosen a particular implementation of "Case", "Rule" and "Result". Furthermore, we also design tasks mimic proof steps in formal theorem proving (see the rewrite task in Appendix 7.2.4), which also achieved excellent results. Nevertheless, we believe LIME is a first step towards building reasoning inductive biases, and provides many inspirations and directions for future work.

7.7 Conclusion

In this work, we encoded inductive biases for mathematical reasoning in the form of datasets. We created three synthetic tasks inspired by three reasoning primitives of deduction, induction, and abduction. We demonstrated that pretraining on these tasks (LIME) significantly improved the performances across four mathematical reasoning benchmarks. Notably, LIME requires negligible computation compared to the downstream task, unlike being the dominating factor in previous pretraining methods. Our work naturally poses many future research questions. Could the primitive tasks provide similar gains for NLP tasks? Are there similar primitive tasks for natural language reasoning? We also look forward to disentangling the effects of pretraining between learning content knowledge and inductive bias for all downstream tasks to better understand pre-training.

Chapter 8

PACT: self-supervised learning for theorem proving

Labeled data for imitation learning of theorem proving in large libraries of formalized mathematics is scarce as such libraries require years of concentrated effort by human specialists to be built. This is particularly challenging when applying large Transformer language models to tactic prediction, because the scaling of performance with respect to model size is quickly disrupted in the data-scarce, easily-overfitted regime. In this chapter we introduce PACT (**P**roof **A**rtifact **C**o-**T**raining), a general methodology for extracting abundant self-supervised data from kernel-level proof terms for co-training alongside the usual tactic prediction objective. We apply this methodology to Lean, a proof assistant host to some of the most sophisticated formalized mathematics to date. We instrument Lean with a neural theorem prover driven by a Transformer language model and show that PACT improves theorem proving success rate on a held-out suite of test theorems from 32% to 48%.

This chapter is largely based on the work: *Proof Artifact Co-training for Theorem Proving with Language Models* by Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, Stanislas Polu [51], published in the *Mathematical Reasoning in General Artificial Intelligence Workshop* (MATHAI) at ICLR of 2021. My contribution to this work includes the training of the first neural network models on the mined dataset, developing the Fairseq baseline (including training and evaluation), providing technical guidance on the use of machine learning models, developing proof artifact datasets (proof terms and next lemma), and suggesting mixed training over all mined data which leads to the idea of self-supervised co-training.

8.1 Overview

Deep learning-driven automated theorem proving in large libraries of formalized mathematics (henceforth "neural theorem proving") has been the focus of increased attention in recent years. Labeled data for imitation learning of theorem proving is scarce—formalization is notoriously labor-intensive, with an estimated cost of 2.5 man-years per megabyte of formalized mathematics [134], and complex projects require years of labor from human specialists. Within a fixed corpus of (possibly unproven) theorem statements, it is possible to augment a seed dataset of human proofs with new successful

trajectories using reinforcement learning or expert iteration. However, this is quite computationally intensive, and without a way to expand the curriculum of theorems, the agent will inevitably saturate and suffer from data starvation.

Data scarcity is a particularly thorny obstruction for applying large language models (LLMs) to neural theorem proving. LLMs have achieved spectacular success in data-rich regimes such as plain text [15], images [28], and joint text-image modeling [99], and the performance of decoder-only Transformers has been empirically shown to obey scaling power laws in model and data size [54]. However, existing datasets of human proof steps for neural theorem proving are extremely small and exist at scales at which overfitting occurs extremely rapidly, disrupting the scaling of performance with respect to model size [64].

We make two contributions towards addressing the problem of data scarcity in the context of formal mathematics. First, we introduce PACT (**P**roof **A**rtifact **C**o-**T**raining), a general methodology for extracting self-supervised auxiliary tasks for co-training a language model alongside a tactic prediction objective for interactive theorem proving. Second, we present LeanStep, a collection of datasets and a machine learning environment for the Lean 3 theorem prover with support for PACT, supervised learning of tactic prediction, theorem proving evaluation, and reinforcement learning.

We train large language models on these data and demonstrate that PACT significantly improves theorem proving success rate on a held-out suite of test theorems, from 32% to 48%. We then embark on a careful study of the effects of pretraining vs. co-training and show that PACT combined with *WebMath* pretraining [97] achieves the best validation loss and theorem proving success rate. Finally, on an out-of-distribution collection of thousands of theorems (some involving novel definitions) added to Lean's mathematical library after we extracted our train/test data, we achieve a theorem proving success rate of 37%, suggesting strong generalization and usefulness at the frontier of formalized mathematics.

8.2 Proof artifact co-training

In this section, we describe the PACT task suite and how data for these tasks are extracted.

For every proof term τ, we record the type Γ of τ, its name `nm`, and a list `ps` of all premises (*i.e.* named references to other lemmas in the library) which are used in τ. We then recurse through τ, tracking a list `bs` of bound variables which we update whenever navigating into the body of a λ-expression. At every sub-term $\tau' \subseteq \tau$ we record τ', its type Γ', the current state of `bs`, and the following data:

1. A *tactic state*, where the goal is set to be Γ' and the list of hypotheses in the local context is set to be the list `bs`, *i.e.* those bound variables in scope at τ'.
2. A *partial proof term*, *i.e.* τ with τ' masked out.
3. A *premise selection bitmask*, *i.e.* Boolean labels for every `p` in `ps` indicating whether `p` is used in τ'.
4. A *local context bitmask*, *i.e.* similar Boolean labels for every `b` in `bs` indicating whether `b` is used in τ'.
5. An optional *next lemma*: if the first step of τ' is to apply a premise `p` in `ps`, we record `p`.

Whenever we record a term, we record both *pretty-printed* and far more explicit *fully elaborated* versions of it. The fully elaborated terms explicitly display enormous amounts of type information which are usually silently inferred by Lean. From these data, we assemble the following language modeling tasks:

1. **Next lemma prediction.** Given the tactic state, predict the next lemma to be applied.
2. **Proof term prediction.** Given the tactic state, predict the entire proof term τ'.
3. **Skip-proof.** Given the partial proof term, predict the masked-out subterm τ'.
4. **Type prediction.** Given the partial proof term, predict the type Γ' of the masked-out subterm τ'.
5. **Tactic state elaboration.** Given the tactic state, predict the fully elaborated tactic state.
6. **Proof term elaboration.** Given τ, predict the fully elaborated version of τ.
7. **Premise classification.** Given the tactic state and a premise `p` $\in$ `ps`, predict either `<TRUE>` or `<FALSE>` according to the premise selection bitmask.
8. **Local context classification.** Given the tactic state (which consists of a list of local assumptions `bs` and the goal Γ'), predict the sublist of `bs` which is true on the local context bitmask.
9. **Theorem naming.** Given the type Γ of the top-level proof term τ, predict the name `nm`.

We remark that our next lemma prediction task is precisely the low-level `PROOFSTEP` objective studied in [97], and our skip-proof task superficially resembles, but is much more difficult than the skip-tree task studied in [98], as proof terms tend to be far more complex than the syntax trees of theorem statements.

Dataset sizes

- **Next lemma prediction**: ≈2.5M examples
- **Proof term prediction**: ≈2.9M examples
- **Skip-proof**: ≈1.7M examples
- **Type-prediction**: ≈1.7M examples
- **Tactic state elaboration**: ≈346K examples
- **Proof term elaboration**: ≈1.0M examples
- **Premise classification**: ≈9.3M examples
- **Local context classification**: ≈2.0M examples
- **Theorem naming**: ≈32K examples.

8.3 Experiments

8.3.1 Training protocol

In all of our experiments, we use decoder-only Transformers similar to GPT-3 [15]. Unless mentioned otherwise, all of our models have 24 layers with $d_{\text{model}} = 1536$ and 24 heads, accruing to $837m$ trainable parameters. They are also pretrained on `WebMath` [97] for $72B$ tokens. We use the standard `BPE` encoding [15], a batch size of 512 and a learning rate of 0.00025 with a cosine schedule and a 100-step ramp-up.

We use an 80-5-15 train-validation-test split. We split all datapoints deterministically by *theorem name*, by hashing each name to a float in $(0, 1)$. This ensures, for example, that proof steps used to prove a test theorem never appear in the training data and vice-versa.

When fine-tuning a model we load its saved parameters but re-initialize the optimizer. We start each training for a fixed number of tokens (defining the cosine schedule) and record the number of tokens consumed as we reach a minimal validation loss. We use the minimum validation loss snapshot to evaluate each model on our held-out test set.

We partition our datasets into three groups:

1. `tactic`: the dataset described in Section 4.2.1.
2. `mix1`: the union of the PACT tasks **next lemma prediction** and **proof term prediction** (Section 8.2), selected because of their close relation to `tactic`.
3. `mix2`: all other datasets described in Section 8.2.

This grouping is motivated by the impossibility to ablate each dataset separately given our compute budget. They nonetheless enable us to study the effect of tasks that are very close to the `tactic` objective in comparison to others. Our choice of **next lemma prediction** and **proof term prediction** for `mix1` is motivated by the observation that these tasks are closely related to the theorem proving objective: a proof can be given entirely in terms of a sequence of lemmas to apply (as in Metamath), or the proof can be finished in one step by supplying the entire proof term. Despite their logical similarity to the `PROOFSTEP` objective, we nevertheless use different keywords in the prompt to the model to disambiguate (`NEXTLEMMA` and `PROOFTERM`) from (`PROOFSTEP`) because the data is noisy and represents a significant distribution shift: during pretty-printing, subtrees of proof terms beyond a certain depth are dropped entirely, there is generally no guarantee that they can be re-parsed, and the data is much more verbose than what humans typically supply in source code.

8.3.2 Evaluation protocol

We run theorem-proving evaluations on our held-out `test` set, comprising 3071 theorems. Since the split was conducted by theorem name, the proofs of these theorems never appear in the training data. For each theorem in the test set, we set the runtime environment to the location where the theorem is proved in the source code, preventing the use of theorems defined later in `mathlib` and ensuring that we never derive circular proofs. We run the proof search algorithm using either the `tidy` or the `gptf` backend. In all of our experiments, we use a maximum width of $w_{\max} = 16$, a maximum depth of $d_{\max} = 128$, a maximum budget of 512 iterations of the outer loop, a timeout of

`tactic`	
tactic proof steps	`GOAL <TacticState> PROOFSTEP <Tactic>`
`mix1`	
next lemma prediction	`GOAL <TacticState> NEXTLEMMA apply (<NextLemma>)`
proof term prediction	`GOAL <TacticState> PROOFTERM exact (<ProofTerm>)`
`mix2`	
skip proof	`RESULT <MaskedProofTerm> SKIPPROOF <ProofTerm>`
type prediction	`RESULT <MaskedProofTerm> PREDICTTYPE <Type>`
tactic state elaboration	`GOAL <TacticState> ELABGOAL <ElaboratedTacticState>`
proof term elaboration	`PROOFTERM <ProofTerm> ELABPROOFTERM <ElaboratedProofTerm>`
premise classification	`GOAL <TacticState> CLASSIFYPREMISE <Premise> <True|False>`
local context classification	`GOAL <TacticState> CLASSIFYLOCALS <LocalsList>`
theorem naming	`TYPE <Type> NAME <Name>`

Figure 8.1: Auto-regressive objectives used for each task described in **??**. Placeholders represented with brackets (such as `<TacticState>`) are substituted by the context-completion pairs from each datasets in the prompts above. Each task is presented to the model with its respective keyword (`PROOFSTEP`, `NEXTLEMMA`,...). We wrap the completions of `mix1` tasks (with `apply(...)` and `exact(...)` respectively) as a hint that they are related to the respective Lean tactics; this is not directly possible for the other tasks.

5 seconds per tactic execution, and a global timeout of 600 seconds per theorem. Because sampling completions from our models is much slower ($\approx$ 1 second) than querying the constant `baseline` oracle (instantaneous), the `baseline` proof search runs through many more rounds of proof search than `gptf` before timeout.

We report the percentage of theorems proved from the held-out test set, averaging over three evaluation runs when reporting the pass rate (*i.e.* percentage of theorems proved).

8.3.3 Effect of co-training vs pretraining

We first study the effects of pretraining versus co-training with the `mix1` and `mix2` datasets. We pretrain using the methodology described above (potentially pretraining first on `WebMath`, and then on a PACT dataset in sequence). For co-training, we simply concatenate and shuffle the datasets together without applying any particular weight to a given dataset.

The main results are presented in Figure 8.2. Pretraining exhibits an effective transfer from `mix-1` and/or `mix-2` but the best result is achieved by co-training with both these datasets. With this setup, we are able to train for much longer (71B tokens vs 22B+18B for the best pretraining setup) before overfitting on the `PROOFSTEP` task. We hypothesize that PACT regularizes overfitting to the `PROOFSTEP` task while still imparting useful knowledge to the model due to large amounts of mutual information, and that this is the main driver of increased performance.

8.3.4 Ablation Studies

Ablating `WebMath` pretraining Next, we ablate the effect of `WebMath` pretraining. As expected, co-trained models suffer from a performance drop without `Webmath` pretraining. but we were more interested in measuring the effect on pretrained models on `mix-1` and `mix-2`, as they may

Model	Tokens elapsed	`mix1`	`mix2`	`tactic`	Pass-rate
Baselines					
`refl`					1.1%
`tidy-bfs`					9.9%
`WebMath > tactic`	1B			1.02	32.2%
Pretraining					
`WebMath > mix1`	11B	*0.08*			
`WebMath > mix2`	16B		*0.08*		
`WebMath > mix1 + mix2`	22B	*0.11*	*0.08*		
`WebMath > mix1 > tactic`	1B			1.00	39.8%
`WebMath > mix1 + mix2 > tactic`	1B			0.97	44.0%
Co-training (PACT)					
`WebMath > mix1 + tactic`	18B	*0.08*		0.94	40.0%
`WebMath > mix1 + mix2 + tactic`	71B	*0.09*	*0.09*	**0.91**	**48.4**%
Pretraining and co-training					
`WebMath > mix2 > mix1 + tactic`	18B	*0.08*		0.93	46.9%

Figure 8.2: Comparison of pretraining and co-training on `mix-1` and `mix-2`. `>` denotes a pretraining step and `+` denotes a co-training. As an example, `WebMath > mix2 > mix1 + tactic` signifies a model successively pretrained on `WebMath` then `mix2` and finally co-trained as a fine-tuning step on `mix1` and `tactic`. Columns `mix1`, `mix2`, `tactic` report the optimal validation loss achieved on these respective datasets. We provide a detailed description of experiment runtime and computing infrastructure in the appendix.

not benefit from `WebMath` as much due to the two successive pretraining steps.

We report the optimal validation losses in Figure 8.3 (we plan to report evaluation pass-rates as well in a later version of this paper). `WebMath` appears as substantially beneficial even in the sequential pretraining setup. This indicates that PACT is not a replacement for `WebMath` pretraining, but rather a complementary method for enhancing the performance of language models for theorem proving.

Ablating regularization We rule out the possibility that the benefits from PACT come from simply regularizing our models on the scarce `tactic` data alone. We checked that a `WebMath > tactic` model trained with 15% residual dropout achieved a minimum validation loss of 1.01 and 33.6% pass rate, far below the 48.4% PACT pass rate.

Effect of model size Finally, we study how performance scales with respect to model size. We use the best training setup reported in Figure 8.2, `WebMath > mix1 + mix2 + tactic`. The `837m` model is our main model. The `163m` and `121m` models respectively have 12 and 6 layers, with $d_{\text{model}} = 768$. The learning rates are respectively adjusted to 0.0014 and 0.0016.

As demonstrated by Figure 8.4, performance is highly correlated with model size, with larger models generally achieving better generalization even in the overfitted regime. We leave as future work a careful study of how evaluation performance is affected when scaling to multi-billion parameter models, as well as the feasibility of deploying them for interactive use by Lean users.

Model	Tokens total	Early-stop	`mix1`	`mix2`	`tactic`
Baselines					
`tactic`	32B	1B			1.59
pretraining					
`mix1`	32B	20B	*0.12*		
`mix2`	32B	25B		*0.10*	
`mix1 + mix2`	32B	27B	*0.13*	*0.10*	
`mix1 > tactic`	32B	1B			1.26
`mix1 + mix2 > tactic`	32B	1B			1.16
Co-training					
`mix1 + tactic`	32B	27B	*0.11*		1.12
`mix1 + mix2 + tactic`	96B	71B	*0.10*	*0.11*	**1.07**
pretraining and co-training					
`mix2 > mix1 + tactic`	32B	26B	*0.11*		1.09

Figure 8.3: Validation losses achieved in the pretraining and co-training setups without `WebMath` pretraining. See Figure 8.2 for a description of the columns and the models nomenclature used.

Model	Tokens total	Early-stop	`mix1`	`mix2`	`tactic`	Pass-rate
`121m`	96B	82B	*0.13*	*0.10*	1.23	35.1%
`163m`	96B	80B	*0.12*	*0.09*	1.11	39.8%
`837m`	96B	71B	*0.09*	*0.09*	**0.91**	**48.4**%

Figure 8.4: Validation losses and pass-rates achieved for various model sizes using PACT. See Figure 8.2 for a description of the columns. The setup used is `WebMath > mix1 + mix2 + tactic`.

8.3.5 Time-stratified evaluation

In the 5 week period that separated our last dataset extraction and the writing of this paper, `mathlib` grew by 30K lines of code, adding 2807 new theorems. Evaluating our models on these new theorem statements gives a unique way to assess their capability to assist humans in formalizing proofs and to test their generalization to completely unseen theorems and definitions. This evaluation set also addresses one of the weaknesses of using a random split of theorems from a formal mathematics library, namely that the split is non-chronological; *e.g.* test theorems can appear as lemmas in proofs of train theorems.

We call this temporally held-out test set `future-mathlib` and evaluate our best model as well as the `refl` and `tidy-bfs` baselines on it. In contrast to evaluation on our `test` split, the `refl` baseline (simply attempting a proof by the `refl` tactic) closes 328 proofs (11.6%), demonstrating an important skew towards trivial boilerplate lemmas generally defined to provide alternate interfaces to new definitions. The `tidy-bfs` baseline closed 611 proofs (21.8%), and our best model `wm-tt-m1-m2` closed 1043 proofs (37.1%), proving 94% of the `refl` lemmas. We attribute the weaker performance to heavy distribution shift: by the nature of the dataset, the `future-mathlib` theorems frequently involve new definitions and concepts which the model was never exposed to during training. Nevertheless, the success rate remains high enough to suggest strong generalization and usefulness at the frontier of formalized mathematics.

8.3.6 Chained tactic prediction

Individual Lean tactics are chained together with commas. However, the Lean interactive tactic DSL also includes a number of other tactic combinators for creating composite tactics. A frequently used combinator is the infix semicolon `t; s` which will perform the tactic `t` and then apply the tactic `s` to each of the resulting subgoals produced by `t`. Our data pipeline for human tactic proof steps treats these semicolon-chained tactics as a single string for the language modeling objective. Thus, our models learn to occasionally emit multiple-step tactic predictions using semicolons. For example, `wm-to-tt-m1-m2` solved the following lemma in category theory with a single prediction chaining four tactics in a row:

```
theorem category_theory.grothendieck.congr
  {X Y : grothendieck F} {f g : X ⟶ Y} (h : f = g) :
  f.fiber = eq_to_hom (by subst h) ≫ g.fiber :=
begin
  rcases X; rcases Y; subst h; simp
end
```

One way of measuring the sophistication of predicted tactics is to consider the number of successful proofs on the evaluation set which have this composite form using semicolon-chaining. We display this analysis in Table 8.1, which shows that training with PACT in addition to the human-made tactics causes longer semicolon-chained tactics to be successfully predicted during theorem proving. This is remarkable because the semicolon idiom is specific to the tactic DSL which does not occur in the PACT data whatsoever, and yet the co-training causes longer and more frequent successful composite tactic predictions.

Table 8.1: Counting the number of semicolon-chained tactics predicted by our models that appear *in successful proofs*. Each column headed by a number n; indicates the number of times that a suggestion appeared with n occurrences of ';'.

Model	1;	2;	3;	4;	Mean
`wm-to-tt`	215	49	2	0	1.199
`wm-to-tt-m1`	186	39	5	1	1.225
`wm-to-tt-m1-m2`	**328**	**82**	**12**	**3**	**1.271**

8.3.7 Theorem naming case study

We included *theorem naming* as part of the PACT task suite. By `mathlib` convention, theorem names are essentially snake-cased, natural language summaries of the type signature of a theorem, and so the theorem naming task is analogous to a formal-to-informal translation task. We evaluate the ability of our best model (in terms of theorem proving success rate) `wm-to-tt-m1-m2` on its ability to guess theorem names on the completely unseen `future-mathlib` set of theorems. The distribution shift inherent in the `future-mathlib` dataset particularly impacts the theorem naming task, because many of the ground-truth names will involve names for concepts that were only defined in `mathlib` *after* we extracted our training data.

On the ≈2.8K `future-mathlib` theorems, we queried `wm-to-tt-m1-m2` for up to $N = 16$ candidates. We order these candidates into a list `xs` by decreasing cumulative log-probability and calculate the top-K accuracy by checking if any of the first K candidates of `xs` match the ground truth exactly. The model `wm-to-tt-m1-m2` was able to achieve 20.1% top-1 accuracy, 21.1% top-3 accuracy, 26.7% top-10 accuracy, and 30.0% top-16 accuracy. We display a sample of correct top-1 guesses (Figure 8.5) and a sample of failed guesses in (Figure 8.6). We note that the failed guesses, while containing no syntactic matches, are both semantically reasonable and syntactically very similar to the ground truth.

8.3.8 Test set evaluation breakdown by module

Lean's `mathlib` is organized into top-level modules, which roughly organize theorems into mathematical subject area. In Figure 8.7, we break down the evaluation results on our `test` set between our PACT-trained models `wm-to-tt-m1-m2` and `wm-to-tt-m1` and our baselines `wm-to-tt` and `tidy`. We see that full PACT mostly dominates over co-training on just the `mix1` tasks over all subject areas, and that `wm-to-tt-m1` dominates the model `wm-to-tt` trained on human tactic proof steps only.

8.4 Computational resource estimates

For each evaluation loop over the `test` set, we distributed the theorems over a pool of 32 CPU workers whose inference requests were load-balanced over 4 `V100` GPUs. Each evaluation required ≈10 hours with ≈30% GPU utilization. We observed that our evaluation was bottlenecked by inference and in practice, we hosted up to three evaluation loops at once on a VM with 80 logical cores without achieving full CPU utilization. In addition to the wall-clock timeout of 600s, we also limited the proof search to a logical timeout of `512` iterations, where one iteration corresponds to

Correct top-1 guesses	
Theorem statement	∀ {α : Type u_1} {β : Type u_2} [_inst_1 : decidable_eq α] [_inst_2 : decidable_eq β] (s : finset α) (t : finset β), s.product t = s.bUnion (λ (a : α), finset.image (λ (b : β), (a, b)) t)
Ground truth	finset.product_eq_bUnion
Theorem statement	∀ {α : Type u_1} {β : Type u_2} [_inst_1 : topological_space α] [_inst_2 : topological_space β] {f : α → β}, quotient_map f → function.surjective f
Ground truth	quotient_map.surjective
Theorem statement	∀ {α : Type u_1} {β : Type u_2} (f : α → option β) (x : option α), x.pbind (λ (a : α) (_x : a ∈ x), f a) = x.bind f
Ground truth	option.pbind_eq_bind
Theorem statement	∀ {C : Type u$_1$} [_inst_1 : category_theory.category C] {G : C ⥤ C} [_inst_2 : category_theory.comonad G] {A B : category_theory.comonad.coalgebra G} (h : A.A ≅ B.A) (w : A.a ≫ G.map h.hom = h.hom ≫ B.a), (category_theory.comonad.coalgebra.iso_mk h w).hom.f = h.hom
Ground truth	category_theory.comonad.coalgebra.iso_mk_hom_f
Theorem statement	∀ {k : Type u_1} {E : Type u_2} [_inst_1 : is_R_or_C ,k] [_inst_2 : inner_product_space k E] [_inst_4 : normed_space ℝ E] [_inst_5 : is_scalar_tower ℝ k E] (p x : E × E), ⇑(fderiv_inner_clm p) x = has_inner.inner p.fst x.snd + has_inner.inner x.fst p.snd
Ground truth	fderiv_inner_clm_apply

Figure 8.5: A sample of correct top-1 guesses by our best model `wm-to-tt-m1-m2` on the *theorem naming* task. We performed this experiment on the `future-mathlib` evaluation set, which comprises entirely unseen theorems added to `mathlib` only after we last extracted training data.

Incorrect guesses	
Theorem statement	∀ {α : Type u_1} (t : ordnode α) (x : α), t.dual.find_min′ x = ordnode.find_max′ x t
Guesses (top 8)	ordinal.find_min′_eq, ordinal.find_min′_eq_max′, ordinal.find_min′_def, ordinal.find_min′_eq_max, ordinal.find_min′, ordinal.dual_find_min′, ordinal.find_min′_gt, ordinal.find_min′_q
Ground truth	ordnode.find_min′_dual
Theorem statement	∀ {α : Type u_1} {β : Type u_3} {γ : Type u_5} [_inst_1 : measurable_space α] [_inst_3 : measurable_space β] [_inst_5 : measurable_space γ] {μ : measure_theory.measure α} {ν : measure_theory.measure β} [_inst_8 : measure_theory.sigma_finite ν] {f : α × β → γ}, ae_measurable f (μ.prod ν) → (∀ᵐ(x : α) ∂μ, ae_measurable (λ (y : β), f (x, y)) ν)
Guesses (top 8)	measure_theory.ae_prod, measure_theory.ae_of_ae_prod, measure_theory.ae_eq_prod_of_ae, measure_theory.ae_ae_of_ae_prod, measure_theory.ae_measure_prod_mk_left, measure_theory.ae_prod_of_ae_prod, measure_theory.ae_measure_prod, measure_theory.ae_eq_refl
Ground truth	ae_measurable.prod_mk_left
Theorem statement	∀ {α : Type u_1} {β : Type u_2} {γ : Type u_3} {f : filter α} {h : set α → set β} {m : γ → β} {l : filter γ}, filter.tendsto m l (f.lift′ h) ↔ ∀ (s : set α), s ∈ f → (∀ᶠ (a : γ) in l, m a ∈ h s)
Guesses (top 8)	filter.tendsto_lift′_iff, filter.tendsto_lift′_def
Ground truth	filter.tendsto_lift′
Theorem statement	∀ {R : Type} [_inst_1 : comm_ring R] {d : ℤ} (f : ℤ√d →+* R), ↑(⇑(zsqrtd.lift.symm) f) = ⇑f zsqrtd.sqrtd
Guesses (top 8)	zsqrtd.coe_lift_symm, zsqrtd.coe_lift.symm, zsqrtd.lift.coe_symm_apply, zsqrtd.lift_symm_apply, zsqrtd.lift.coe_coe_symm, zsqrtd.lift.coe_symm_coe, zsqrtd.lift.symm_coe_zsqrtd, zsqrtd.lift_symm_to_zsqrtd
Ground truth	zsqrtd.lift_symm_apply_coe

Figure 8.6: A sample of incorrect guesses by our best model `wm-to-tt-m1-m2` on the *theorem naming* task. We performed this experiment on the `future-mathlib` evaluation set, which comprises entirely unseen theorems added to `mathlib` only after we last extracted training data. Most of the top-8 guesses displayed in the above table are very similar to the ground truth, in some cases being equivalent up to permutation of underscore-separated tokens. Note that for the first example, the concept of `ordnode` was not in the training data whatsoever and all predictions are in the syntactically similar `ordinal` namespace.

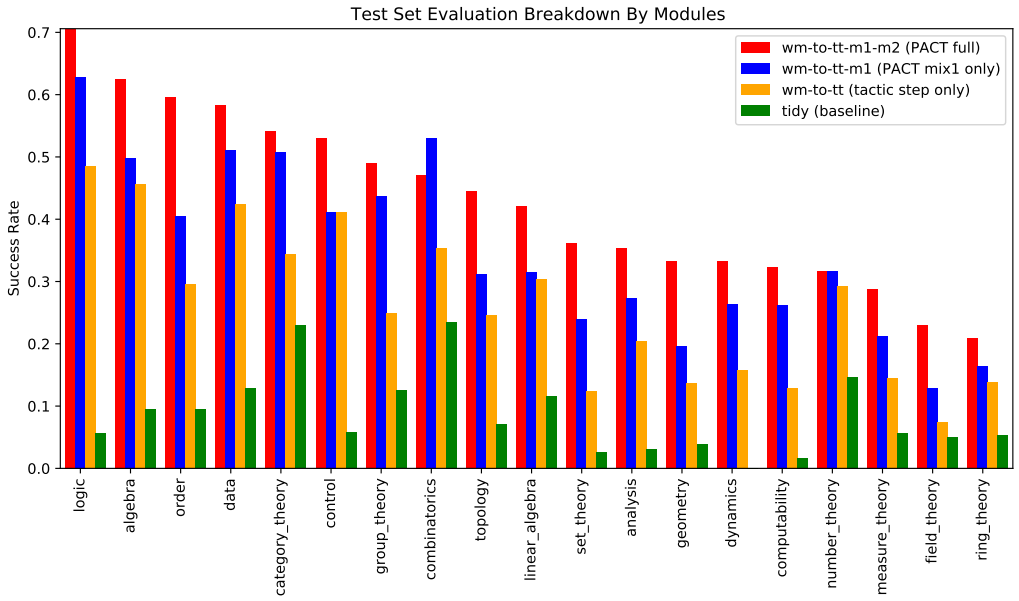


Figure 8.7: A breakdown of theorem proving success rate on the `test` set for `wm-to-tt-m1-m2`, `wm-to-tt-m1`, `wm-to-tt`, and the `tidy` baseline across top-level modules in Lean's `mathlib`. We see that `wm-to-tt-m1-m2` mostly dominates `wm-to-tt-m1` and the models trained using PACT dominate the model `wm-to-tt` trained on human tactic proof steps.

a single expansion of a node of the BFS search tree. In practice, so much time was spent either blocked on inference or performing the tactic executions in the inner loop of each iteration that we rarely exceeded the logical timeout, usually exceeding the wall-clock timeout instead.

Fine-tuning on our largest dataset `mix1 + mix2 + tactic` required 26 hours using 64 `A100` GPUs exhibiting high `FP16` usage, totalling an estimated ≈1.5K `A100(FP16)`-hours. This gives an estimated cost of 17.33 `A100(FP16)`-hours per billion elapsed tokens during training. We note that when calculating the number of elapsed tokens for training, we overestimate the actual number of tokens effectively trained on by summing full context windows (in this case, 2048 tokens).

8.5 Example proofs

Lean's `mathlib` is one of the most active open-source software projects in the world. More than one-third of the proofs found by our models are shorter and produce smaller proof terms than the ground truth, leading to dozens of GPT-f co-authored commits to `mathlib`. We examine some of the proofs found by our models in more detail.

- `lie_algebra.morphism.map_bot_iff`

  This proof produces a proof term which is 4X smaller than the original:

  ```
  lemma map_bot_iff : I.map f = ⊥ ↔ I ≤ f.ker :=
  by { rw  le_bot_iff, apply lie_ideal.map_le_iff_le_comap }
  ```

  The original, human-written proof is much longer, viz.

```
lemma map_bot_iff : I.map f = ⊥ ↔ I ≤ f.ker :=
begin
  rw le_ker_iff, unfold lie_ideal.map, split; intros h,
  { rwa [eq_bot_iff, lie_submodule.lie_span_le, set.image_subset_iff,
    lie_submodule.bot_coe] at h,},
  { suffices : f '' I = ↑(⊥ : lie_ideal R L'), { rw [this, lie_submodule.lie_span_eq],
    },
    ext x, rw [lie_submodule.bot_coe, set.mem_singleton_iff, set.mem_image],
    split,
    { rintros ⟨y, hy, hx⟩, rw  hx, exact h y hy, },
    { intros hx, use 0, simp [hx], }, },
end
```

- `primrec.of_equiv`

  This proof produces a proof term which is 12X smaller than the original:

```
theorem of_equiv {β} {e : β ≃ α} :
  by haveI := primcodable.of_equiv α e; exact
  primrec e :=
by letI : primcodable β := primcodable.of_equiv α e; exact encode_iff.1 primrec.encode
```

  The author of the original proof and maintainer of that package commented:

  > `encode_iff.1 primrec.encode` is clever, it's a way to translate `primrec` across an equivalence when the encode function is defined as `encode x = encode (e x)` where `e` is the isomorphism.

  As far as they knew, this trick was never used before in the `computability` package.

- `real.tan_eq_sin_div_cos`

  This proof demonstrates our model's library knowledge and ability at premise selection.

```
lemma real.tan_eq_sin_div_cos (x : ℝ) : tan x = sin x / cos x :=
begin
  rw  of_real_inj,
  simp only [complex.tan_eq_sin_div_cos, of_real_sin, of_real_cos, of_real_div,
    of_real_tan]
end
```

  Our model was able to predict this entire list of `simp` lemmas in one shot. Note that the lemma `complex.tan_eq_sin_div_cos` in this list is the *complex number* version of the result, i.e. `∀ (x : ℂ), tan x = sin x / cos x`. The previous human-written version of the proof did not use the more general version of the lemma on complex numbers, demonstrating our model's ability to find more general cases of lemmas. We contrast this with the human-written ground truth, which is more complex and performs a case analysis using the complex cosine:

```
  lemma tan_eq_sin_div_cos : tan x = sin x / cos x :=
if h : complex.cos x = 0 then by simp [sin, cos, tan, *, complex.tan, div_eq_mul_inv]
    at *
```

```
else
  by rw [sin, cos, tan, complex.tan,  of_real_inj, div_eq_mul_inv, mul_re];
  simp [norm_sq, (div_div_eq_div_mul _ _ _).symm, div_self h]; refl
```

- `sym2.is_diag_iff_proj_eq`

  The proof of this lemma is longer than the ground truth and was not contributed to `mathlib`, but we describe it here because the proof is original and includes a nontrivial instantiation of an existential quantifier.

```
theorem sym2.is_diag_iff_proj_eq (z : α × α) :
  is_diag ⟦z⟧ ↔ z.1 = z.2 :=
begin
    intros,
    simp only [is_diag, prod.ext_iff, quot.exists_rep, iff_true, not_true,
    eq_self_iff_true],
    simp [diag], split,
    { rintros ⟨y, hy⟩, cases hy; refl },
    intro h, cases z, existsi z_snd,
    cases h, refl,
end
```

  Before `existsi z_snd`, the goal state is

```
z_fst z_snd: α
h: (z_fst, z_snd).fst = (z_fst, z_snd).snd
⊢ ∃ (y : α), (y, y) ≈ (z_fst, z_snd)
```

  This goal state never appeared in `mathlib`.

- `norm_le_zero_iff`

  The following proof is remarkable because it uses fewer tactic steps and takes a different route to the proof than the ground truth, uses a complex idiom `simpa [...] using @...`, and was predicted in one shot.

```
lemma norm_le_zero_iff {α : Type u_1} [_inst_1 : normed_group α]
  {g : α} : ||g|| ≤ 0 ↔ g = 0 :=
by { simpa [le_antisymm_iff, norm_nonneg] using @norm_eq_zero α _ g }
-- ground truth:
-- by { rw[dist_zero_right],
--      exact dist_le_zero }
```

  The lemmas supplied between the square brackets are used to simplify the main goal. The lemma supplied after the keyword `using` can further simplify the lemmas supplied between the square brackets. The `@` modifier makes all arguments explicit. The string `@norm_eq_zero` never appeared in our training data but the prediction includes the correct number of correctly typed arguments, and even replaces the second argument with a placeholder `_`, correctly guessing that it can be inferred by the elaborator. Finally, this again showcases the strength of our models as premise selectors: all three lemmas `le_antisymm_iff`, `norm_nonneg`, and `norm_eq_zero` were not used in the human-supplied proof but are necessary for this proof.

Moving forward, we hope that our neural theorem provers will continue to find ways to improve `mathlib` and assist in creating new proofs. More generally, we hope neural theorem proving will one day be become a routine part of the formalization workflow.

8.6 Discussions

Impact on Lean community Lean's `mathlib` is one of the most active open-source software projects in the world, achieving explosive growth in recent years [21]. Our work has been welcomed by members of this community, with Lean power users describing some of the new proofs found by GPT-f as "nontrivial" and "clever". More than one-third of the proofs found by our models are shorter and produce smaller proof terms (sometimes by several orders of magnitude) than the ground truth. Manually inspecting a small, non-cherry picked sample of these shorter proofs has led to 19 GPT-f co-authored commits to `mathlib`, some of which reduce proof term sizes and theorem compilation times by an order of magnitude. We supply more detail in the appendix.

Future directions There are many elaborations on the training data, training methodology, and tree search wrapping `lean-gptf` which can be reasonably expected to improve its performance at theorem proving. Our dataset can be synthetically augmented using similar methods as [97]. Our dataset could be cleaned further, and proofs minimized. Merely making the decoded rewrites robust by only using the largest prefix of successful rewrites significantly boosts the success rate of suggested rewrites. In a similar vein, predicted lemmas generated as arguments to unsuccessful tactic applications could be cached and re-used as hints for an intermittently-queried hammer. The increased success rate of chained tactic predictions mentioned above shows the feasibility of having language models perform multiple reasoning steps in a single query, potentially improving the efficiency of the proof search. From the experiments described in Section 5.5, it is clear that the composition of the dataset used for co-training significantly affects performance on theorem proving. Although we uniformly sampled across all co-training tasks, it would be interesting to optimize a dynamic mixture schedule, perhaps annealing towards a desired task.

Conclusion There is a sense in which PACT is merely an application of the well known principle that compute in the form of search should be exchanged for training signal whenever possible. In Lean, typeclass inference relies on a backtracking Prolog-style search; the elaborator performs search to disambiguate overloaded notation and infer types; Lean tactics have complex semantics precisely because they can perform search to find subproofs automatically. The work done by these subroutines is preserved in the proof artifacts, and PACT can be viewed as a way of extracting this information offline for more training signal.

We have presented PACT as a way of addressing the data scarcity issue for learning theorem proving from human tactic scripts in proof assistant libraries. Another well-studied solution for this is expert iteration and reinforcement learning. In the setting of HOL Light, and under the assumption of a hardcoded finite action space of tactics, DeepHOLZero in conjunction with supervised seed data was able to achieve up to 70% proof success rate on the HOList theorem proving task. Similarly, in a set-up much closer to ours, MM GPT-f demonstrated the feasibility of expert iteration when using generative language models for theorem proving.

Within a fixed corpus of theorems (and hence proof terms), however, both PACT and RL are fundamentally constrained by a lack of exploration—as the performance of the theorem proving agent improves, it will eventually saturate and become starved for data, and its curriculum will need to be expanded. Although self-supervised methods such as PACT represent a way to significantly improve the data-efficiency of reinforcement learning loops over existing theorem prover libraries, the development of continuously self-improving and infinitely scalable neural theorem provers remains contingent on sufficiently powerful exploration and automated curriculum generation; we consider these challenges to be of paramount importance.

Chapter 9

Conclusion and Future

In this thesis, we have explored the potential of neural networks for reasoning, specifically in the domain of mathematical reasoning. We have shown that the use of neural networks can bring significant improvements in the performance of automated and interactive theorem provers. By leveraging statistical structures and modeling arbitrary action distributions, neural networks can learn effective heuristics for theorem proving, leading to faster and more accurate results. Furthermore, we have established benchmarks to evaluate our progress and to provide a common ground for future research in this area.

Our research have investigated the capabilities of neural networks in mathematical reasoning, particularly in abduction and induction. Our experiments reveal that neural networks can indeed perform non-trivial mathematical reasoning tasks with high accuracy, and we have further improved their performance with various techniques. These techniques include designing better inductive biases and utilizing self-supervised learning. Through our work, we have demonstrated the potential of neural networks as a promising tool for mathematical reasoning, and we believe that our findings will contribute to the ongoing efforts to bridge the gap between machine learning and automated reasoning. In this concluding chapter, we outline future research in this exciting and rapidly evolving field.

9.1 General mathematical reasoning

We briefly discuss our vision of the *general* mathematical reasoning problem.

Even though the formal reasoning program provides us with a well-defined problem to start our investigation, it faces one main challenge as a problem domain: a large body of mathematics has not yet been formalized. For a model trained on the existing human corpus, its mathematical knowledge is limited to the formalized fragment of mathematics – this is certainly unsatisfactory if one's goal is to build a versatile mathematician. One approach to tackle this limitation is to design an exploration algorithm that enable the model to discover new mathematics beyond the formalized ones. However, it is unclear how to (re)create the fragment of mathematics that is also interesting to humans. In fact, we believe that, without extra human guidance, a random exploration approach is most likely doomed due to the massive space of possibilities.

Another approach could be to completely abandon the formal reasoning program and focus on

informal mathematics. After all, the majority of mathematics is written informally in textbooks, journals, and arXiv papers. The downside of this approach is the lack of principled evaluation protocol for developing machine learning models. Once a proof is written informally, we have to resort to human mathematicians for help verify the proof. The cost of evaluation strongly hinders research progress and hence we do not favor this approach.

Instead, we believe that one promising approach is to combine the best of both worlds: a verifiable formal environment for principled evaluation, and a large amount of information from the corpus of informal mathematics.

If we say that the *narrow* definition of mathematical reasoning problem is to build a versatile mathematician, we believe that to tackle this challenge, one needs to tackle a more *general* problem: build a model that can formalize informal mathematical statements and proofs into formalized programs. The implications of such success are huge. Not only could one potentially prove all the useful mathematical theorems, but we could build a machine that can uncover the ambiguity in natural language (informal proofs) and crystallize it to form a formalized program (formal proofs).

9.2 Future Directions

With the general mathematical reasoning problem in mind, we will now discuss the promising directions forward to tackle it.

9.2.1 Auto-formalization

Autoformalization, the process of automatically transforming informal mathematical statements into formal ones, has the potential to revolutionize mathematical reasoning. One promising direction for future research is to use autoformalization to enhance general mathematical reasoning. By grounding natural language mathematical provers with direct feedback signals from formal proof assistants, we can harness the power of neural networks to guide and accelerate the discovery of new mathematical theorems while maintaining the rigor and precision of formal proofs. This would allow mathematicians to take advantage of the best of both worlds - the intuition and creativity of informal math, combined with the rigor and accuracy of formal math. In addition, autoformalization could also enable the automated generation of new conjectures and hypotheses based on patterns discovered in formalized mathematical knowledge, providing a powerful tool for exploring and advancing mathematical knowledge. Further research in this direction is needed to fully explore the potential of autoformalization for general mathematical reasoning.

9.2.2 Exploration / Search

Current search and exploration techniques are still rudimentary, largely following traditional methods such as Breadth-First Search (BFS). Here, we highlight some promising directions that can be pursued to improve these techniques.

Search from a higher level of abstraction One such direction is to search from a higher level of abstraction. The human mind does not search for tactics but rather for jumpy thoughts, which is of a much higher level of abstraction. There are various ways to represent these concepts. One

exciting approach is to use natural language, which is highly adaptable in terms of the objects it can represent. By using natural language, one can easily search for ideas at various levels of abstraction, which allows for more efficient exploration.

Another promising direction is to conduct searches in the latent space. By representing actions and states in this latent space, it becomes possible to conduct searches more efficiently and flexibly.

Search by repair When it comes to writing, mistakes are an inevitable part of the process. Even professional mathematicians cannot always complete a proof in a single attempt. However, using error messages from proof assistants, one can pinpoint the reasons for the failure and make necessary adjustments. Through this iterative process of refinement and editing, we can naturally arrive at the final proof.

In other words, the search by repair approach involves iteratively refining a solution until it meets certain criteria or fulfills certain requirements. This can be done by detecting errors or flaws in the solution and correcting them through a series of edits or adjustments. As the process continues, the solution becomes more refined, until it ultimately achieves the desired outcome.

Exploration guided by informal proofs Up to this point in our thesis, our focus has been on searching for formal proofs starting from scratch. However, it is worth noting that most formal proofs have counterparts described in natural language. As a result, one potential approach to proof search in formal theorem proving is to leverage the natural language proof as a guide.

By relying on the natural language proof, we can gain insights into the key intermediate steps, conjectures, and hints that can be used to fill in the gaps in the formal proof. This approach is closely related to the concept of autoformalization, which we have discussed earlier. If we can effectively use informal proofs to aid in our exploration, this serves as a compelling demonstration of autoformalization and its powerful implications.

9.2.3 Scratchpad

It also seems plausible that we can let our AIs to use scratchpad, greatly enhancing their ability to investigate and reason about mathematical problems. A scratchpad can be thought of as a virtual whiteboard where the AI can write down and manipulate mathematical expressions, equations, and diagrams.

One way scratchpad can help with mathematical reasoning is by allowing the AI to investigate special examples. Often in mathematics, working with specific values can be helpful to gain insights into the general case. By using scratchpad to input specific values and observe the resulting calculations, the AI can develop intuition about the problem and potentially discover new patterns or relationships.

Scratchpad can also be useful in developing partial proofs and organizing thoughts. In many cases, mathematical proofs can be complex and require multiple steps. By using scratchpad to write down intermediate steps and observations, the AI can keep track of its progress and avoid overlooking important insights. Additionally, scratchpad can be used to organize and structure thoughts, which can help the AI to communicate their reasoning more effectively.

9.2.4 Multimodal model

Another promising direction for advancing mathematical reasoning AIs is the integration of multimodal models that can both generate and understand images. Much of our mathematical intuition is pictorial in nature. Using visual representations can often lead to more efficient problem solutions. Therefore, the ability to reason about both textual and visual representations of mathematical concepts can significantly improve the performance of mathematical reasoning AIs.

The use of multimodal models can facilitate the development of novel approaches to tackle complex mathematical problems. By leveraging image understanding and generation capabilities, these models can represent mathematical problems and concepts more accurately and intuitively. Additionally, they can provide more effective representations of mathematical structures and relationships, which can enhance the performance of reasoning algorithms.

Furthermore, incorporating multimodal models into mathematical reasoning AIs can enable new forms of collaboration between humans and machines. For example, a human mathematician could provide a visual representation of a complex problem, and the AI could then reason about it and provide potential solutions. This could help reduce the time and effort required to solve complex mathematical problems.

9.2.5 Towards an extendable theorem proving system

One promising direction for future research in automated reasoning is the development of an extendable knowledge system that can handle large amounts of information and constantly incorporate new knowledge. One key challenge in this area is determining how to effectively utilize newly learned facts. One approach is to develop algorithms that can automatically recognize which previously unproven theorems can be proved using the newly acquired knowledge. Another approach is to use machine learning techniques to predict which new facts are most likely to be useful in proving previously unsolved theorems.

Another important question is how to update the theorem library with new facts. One possible solution is to develop methods for integrating new facts into existing theories in a way that preserves consistency and logical coherence. This could involve using proof assistants to validate the new facts before they are incorporated into the library. Alternatively, we can use machine learning algorithms to identify which new facts are most relevant to existing theories and then automatically integrate them in a way that maximizes the coherence of the overall system.

Finally, retrieving knowledge from a large library in a way that is faithful to the original source material poses another key challenge. One approach is to use machine learning algorithms to develop natural language processing tools that can automatically identify the most relevant pieces of information in large collections of text. Another approach is to develop graphical user interfaces that make it easy for users to navigate complex libraries of mathematical knowledge, allowing them to quickly find the information they need and incorporate it into their own work. Ultimately, the goal is to develop a system that can handle large amounts of information, update its knowledge in real-time, and provide users with the tools they need to easily access and utilize this knowledge for their own research and problem solving.

Appendix A

INT generation algorithm

A.1 The morph Function

We detail the morphing of C at each step as follows. For each theorem a, we define two symbolic patterns: $\mathcal{L}_a$ and $\mathcal{R}_a$, each represented by an expression (see A.2 for full details). For example, if a is AdditionCommutativity, we use $\mathcal{L}_a = x_1 + x_2$ to denote any formula that is a sum of two terms (x_1 and x_2 can be arbitrary terms). We check if one of the nodes in the computation graph of C has the structure defined by $\mathcal{L}_a$. If so, we then transform that node to a formula specified by $\mathcal{R}_a$. For example, if C is $(p+q)+l = (p+(q+l))$, $p+q$ is a node that matches the pattern specified by $\mathcal{L}_a$, in which $x_1 = p$ and $x_2 = q$. Let $\mathcal{R}_a = x_2 + x_1$. We hence transform the node $p+q$ to $q+p$ as specified by $\mathcal{R}_a$. As a result, C' becomes $(q+p)+l = (p+(q+l))$. If there is no node in the computation graph, we morph the core logic statement using the extension function $\mathcal{E}$, defined in A.3 . We sample nodes in available computation graphs and combine them with C, coming up with C' and optionally a non-empty set of new premises P_{new}.

Algorithm 4 Theorem Generator (complete)

function GENERATE_THEOREM(initial conditions $\mathcal{I}$, axiom order A)
  Axiom order length $L = \text{len}(A)$.
  Initialize core logic statement $C_0 \sim Uniform(\mathcal{I})$, and the set of premises $P = \{C_0\}$.
  for $t \leftarrow 1$ to L **do**
    Get axiom $a_t \leftarrow A[t]$.
    Get new logic statement and premises: C_t, $P_t \leftarrow$ MORPH (a_t, C_{t-1}).
    Add new premises to the set of all premises: $P \leftarrow P \cup P_t$.
  end for
  return C_L, P
end function

function MORPH(axiom a, core logic statement C)
  Collect $\mathcal{N}_t = \{n|$ n is a node in C and matches the pattern specified by $\mathcal{L}_a\}$
  if $\mathcal{N}_t \neq \varnothing$ **then**
    Sample node $n \sim Uniform(\mathcal{N}_t)$.
    Transform n into new node n' using the mapping from $\mathcal{L}_a$ to $\mathcal{R}_a$.
    $C' \leftarrow$ Replace n with n' in the graph of C. $P_{new} \leftarrow \varnothing$.
  else
    Collect $\mathcal{N}$, the set of all nodes in the graphs.
    Extend C and get the set of premises: $C', P_{new} \leftarrow \mathcal{E}(a, C, \mathcal{N})$.
  end if
  return C', P_{new}.
end function

The reasons that we have two sets of rules for morphing are as follow: 1) Transformation rules can only be applied when the axiom will produce an equality, while extension rules can be applied to any axiom. So in order to generate theorems with all the axioms, we need the extension rules. 2) Almost all the extension rules will complicate the core logic statement while none of the transformation rules will. If we only have extension rules, the goal generated can be very complex even the proof is of moderate length. In order to generate compact theorems (goal not too complicated) with long proofs, the transformation rules are preferred. Therefore we only apply extension rules when transformation rules are not applicable.

A.2 Transformation Rules

The implementations of the transformation rules $\mathcal{L}$ and $\mathcal{R}$.

Axiom (a)	$\mathcal{L}_a$	$\mathcal{R}_a$
AdditionCommutativity	$x_1 + x_2$	$x_2 + x_1$
AdditionAssociativity	$x_1 + (x_2 + x_3)$	$(x_1 + x_2) + x_3$
AdditionSimplification	$x_1 + (-x_1)$	0
MultiplicatoinCommutativity	$x_1 \cdot x_2$	$x_2 \cdot x_1$
MultiplicationAssociativity	$x_1 \cdot (x_2 \cdot x_3)$	$(x_1 \cdot x_2) \cdot x_3$
MultiplicationSimplification	$x_1 \cdot \frac{1}{x_1}$	1
AdditionMultiplicationLeftDistribution	$(x_1 + x_2) \cdot x_3$	$x_1 \cdot x_3 + x_2 \cdot x_3$
AdditionMultiplicationRightDistribution	$x_1 \cdot (x_2 + x_3)$	$x_1 \cdot x_2 + x_1 \cdot x_3$
SquareDefinition	x_1^2	$x_1 \cdot x_1$
MultiplicationOne	$x_1 \cdot 1$ or $1 \cdot x_1$	x_1
AdditionZero	$x_1 + 0$ or $0 + x_1$	x_1
SquareGEQZero	NA	NA
PrincipleOfEquality	NA	NA
EquMoveTerm	NA	NA
EquivalenceImpliesDoubleInequality	NA	NA
IneqMoveTerm	NA	NA
FirstPrincipleOfInequality	NA	NA
SecondPrincipleOfInequality	NA	NA

Table A.1

A.3 Extension Function

For these axioms, the core logic statement C needs to be of the form $\text{LHS}(C) = \text{RHS}(C)$.

Axiom (a)	Extension function $\mathcal{E}(C, a, \mathcal{N})$
AdditionCommutativity	Sample node $n \sim$ `Uniform` ($\mathcal{N}$) **return** $\text{RHS}(C) + n = n + \text{LHS}(C)$, $\varnothing$
AdditionAssociativity	Sample nodes $n_1, n_2 \sim$ `Uniform` ($\mathcal{N}$) **return** $\text{RHS}(C) + (n_1 + n_2) = \text{LHS}(C) + n_1 + n_2$, $\varnothing$
AdditionSimplification	**return** $0 = \text{LHS}(C) + (-\text{RHS}(C))$, $\varnothing$
MultiplicatoinCommutativity	Sample node $n \sim$ `Uniform` ($\mathcal{N}$) **return** $\text{RHS}(C) \cdot n = n \cdot \text{LHS}(C)$, $\varnothing$
MultiplicationAssociativity	Sample nodes $n_1, n_2 \sim$ `Uniform` ($\mathcal{N}$) **return** $\text{RHS}(C) \cdot (n_1 \cdot n_2) = \text{LHS}(C) \cdot n_1 \cdot n_2$, $\varnothing$
MultiplicationSimplification	**return** $1 = \text{LHS}(C) \cdot \frac{1}{\text{RHS}(C)}$, $\varnothing$
AdditionMultiplicationLeftDistribution	Sample nodes $n_1, n_2 \sim$ `Uniform` ($\mathcal{N}$) **return** $(n_1 + n_2) \cdot \text{RHS}(C) =$ $n_1 \cdot \text{LHS}(C) + n_2 \cdot \text{LHS}(C), \varnothing$
AdditionMultiplicationRightDistribution	Sample nodes $n_1, n_2 \sim$ `Uniform` ($\mathcal{N}$) **return** $\text{RHS}(C) \cdot (n_1 + n_2) =$ $\text{LHS}(C) \cdot n_1 + \text{LHS}(C) \cdot n_2, \varnothing$
SquareDefinition	**return** $\text{LHS}(C) \cdot \text{RHS}(C) = \text{LHS}(C)^2$, $\varnothing$
MultiplicationOne	**return** `Uniform` ($\{\text{LHS}(C) \cdot 1 = \text{RHS}(C)$, $1 \cdot \text{LHS}(C) = \text{RHS}(C)$ $\}$), $\varnothing$
AdditionZero	**return** `Uniform` ($\{\text{LHS}(C) + 0 = \text{RHS}(C)$, $0 + \text{LHS}(C) = \text{RHS}(C)$ $\}$), $\varnothing$
SquareGEQZero	**return** $\text{LHS}(C) \cdot \text{RHS}(C) \geq 0$, $\varnothing$
PrincipleOfEquality	Sample nodes $n_1, n_2 \sim \mathcal{N}$, where $n_1 = n_2$ **return** $\text{LHS}(C) + n_1 = \text{RHS}(C) + n_2$, $\{n_1 = n_2\}$
EquMoveTerm	Only execute when $\text{LHS}(C)$ is of the form $x + y$ **return** $x = \text{RHS}(C) + (-y)$, $\varnothing$
EquivalenceImpliesDoubleInequality	**return** $\text{LHS}(C) \geq \text{RHS}(C)$, $\varnothing$

Table A.2

For these axioms, the core logic statement C needs to be of the form $\text{LHS}(C) \geq \text{RHS}(C)$.

Axiom (a)	Extension function $\mathcal{E}(C, a, \mathcal{N})$
IneqMoveTerm	Only execute when $\text{LHS}(C)$ is of the form $x + y$ **return** $x \geq \text{RHS}(C) + (-y)$, $\varnothing$
FirstPrincipleOfInequality	Sample nodes $n_1, n_2 \sim \mathcal{N}$, where $n_1 \geq n_2$ **return** $\text{LHS}(C) + n_1 \geq \text{RHS}(C) + n_2$, $\{n_1 \geq n_2\}$
SecondPrincipleOfInequality	Sample node $n \sim \mathcal{N}$, where $n \geq 0$ **return** $\text{LHS}(C) \cdot n \geq \text{RHS}(C) \cdot n$, $\{n \geq 0\}$

Table A.3

A.4 Example problems

Equality theorems

Theorem 1
Goal: $((0 \cdot 1) \cdot ((-(a^2)) \cdot c)) = (((-(a^2)) \cdot ((a \cdot a) + (-(a^2)))) \cdot c)$

Theorem 2
Goal: $(((((0 + c) + a) \cdot a) \cdot 1) \cdot (b \cdot (0 + c))) = ((((c \cdot a) + (a \cdot a)) \cdot (0 + c)) \cdot b)$

Theorem 3
Goal: $0 = ((((c + 0) \cdot (a + a)) \cdot (\frac{1}{((c \cdot a) + (c \cdot a))})) + (-(0 + 1)))$

Theorem 4
Premises: $(b + d) = b$
Goal: $(1 + (-((b + b) \cdot (\frac{1}{((b + (b + d)) \cdot 1)})))) = (0 + 0)$

Theorem 5
Premises: $(a + d) = b$
Goal: $1 = (((d \cdot ((a + d) + ((c + (a + d)) + 0))) \cdot ((d \cdot (a + d)) + (d \cdot (c + b)))) \cdot (\frac{1}{((d \cdot ((a + d) + ((c + (a + d)) + 0)))^2)}))$

Theorem 6
Premises: $((b \cdot b) + d) = (b \cdot b)$
Goal: $(0 + ((b \cdot b) + d)) = (((1 \cdot ((b + b) \cdot b)) + (-(((b \cdot b) + (b \cdot b)) \cdot 1))) + (b \cdot b))$

Theorem 7
Goal: $((a \cdot (a + 0)) + ((-(0 + a)) \cdot (a + 0))) = ((a \cdot 0) + (0 \cdot 0))$

Theorem 8
Goal: $(((c \cdot c) + c) \cdot ((c^2) \cdot 1)) = (((c \cdot c) \cdot (0 + (c \cdot c))) + (c \cdot (0 + (c \cdot c))))$

Theorem 9
Goal: $1 = ((((a \cdot c) + ((b \cdot (a \cdot b)) \cdot c)) \cdot (a + (a \cdot c))) \cdot (\frac{1}{((((a + ((b \cdot a) \cdot b)) \cdot c) \cdot (a \cdot c)) + (((a + ((b \cdot a) \cdot b)) \cdot c) \cdot a))}))$

Theorem 10
Goal: $((((b \cdot c) + (c \cdot c)) + (-(0 + ((b + c) \cdot c)))) \cdot (c \cdot c)) = ((c^2) \cdot 0)$

Theorem 11
Goal: $(1 \cdot (b + a)) = ((0 + (a + b)) + 0)$

Theorem 12
Goal: $(((-c) \cdot (-c)) + (((-c) \cdot c) + ((-c) \cdot (-c)))) = (((-c) \cdot (-c)) + (0 \cdot (-c)))$

Theorem 13
Goal: $(((a^2) \cdot (a \cdot (a + 0))) + (a \cdot (a \cdot (a + 0)))) = ((((a^2) \cdot (a^2)) + (a \cdot (a^2))) + 0)$

Theorem 14
Goal: $((((b \cdot 1) \cdot (a \cdot c)) \cdot (b \cdot a)) + (((b \cdot 1) \cdot (a \cdot c)) \cdot (b \cdot a))) = (((((b \cdot a) \cdot c) \cdot (b \cdot a)) + (((b \cdot a) \cdot c) \cdot (b \cdot a))) \cdot 1)$

Theorem 15
Goal: $1 = ((\frac{1}{((\frac{1}{(b + 0)}) \cdot b)}) \cdot 1)$

Theorem 16

Goal: $0 = ((0 + (-((a \cdot b) + (-(b \cdot a))))) + (-(0 \cdot 1)))$

Theorem 17
Premises: $(a + d) = c$; $((b + c) + e) = (a + d)$
Goal: $(((b \cdot d) + (b \cdot (b + (a + d)))) + ((b + c) + e)) = ((((b \cdot d) + (b \cdot (b + c))) \cdot 1) + (a + d))$

Theorem 18
Goal: $((((\frac{1}{b}) \cdot b) \cdot b) \cdot 1) = ((b \cdot 1) \cdot 1)$

Theorem 19
Goal: $(((1 \cdot (b \cdot (c + a))) + (b \cdot a)) + 1) = (1 \cdot ((1 \cdot ((b \cdot c) + (b \cdot a))) + ((b \cdot a) + 1)))$

Theorem 20
Premises: $(b + d) = c$; $((1 \cdot a) + e) = a$
Goal: $(((a + (b + d)) \cdot (\frac{1}{((1 \cdot a) + c)})) + ((1 \cdot a) + e)) = ((1 \cdot 1) + a)$

Theorem 21
Goal: $((((c^2) \cdot ((c^2) \cdot c)) + (-(((c \cdot c) \cdot (c^2)) \cdot c))) + (b + b)) = ((1 \cdot ((0 + b) + b)) + (-0))$

Theorem 22
Premises: $(b + d) = (a \cdot b)$
Goal: $(1 \cdot ((((c + c) \cdot (((a \cdot b) \cdot c) + (c + c))) + ((c + c) \cdot (c + c))) + (a \cdot b))) = (((((c + c) \cdot ((((a \cdot (b \cdot c)) + c) + c) + (c + c))) + (b + d)) \cdot 1) + 0)$

Theorem 23
Premises: $((0 \cdot 1) + d) = (1 \cdot 0)$
Goal: $(((((a + (0 \cdot 1)) \cdot (1 \cdot 0)) + (-b)) + (1 \cdot 0)) + (1 \cdot 0)) = ((((((a \cdot (1 \cdot 0)) + ((b + (-b)) \cdot (1 \cdot 0))) + ((-b) + (1 \cdot 0))) + ((0 \cdot 1) + d)) + 0)$

Theorem 24
Premises: $(a + d) = (1 + c)$
Goal: $(((((1 \cdot b) + (c \cdot b)) + (1 + c))^2) \cdot ((1 + c) \cdot b)) = ((((((1 \cdot b) + (c \cdot b)) + (1 + c)) \cdot (((b \cdot (1 + (1 \cdot c))) + (a + d)) \cdot 1)) \cdot (1 + c)) \cdot b)$

Theorem 25
Premises: $(a + d) = (b \cdot 1)$
Goal: $0 = ((b + (a + d)) + (-((b \cdot 1) + (b \cdot 1))))$

Theorem 26
Premises: $(c + d) = a$
Goal: $(0 + ((((a + a) \cdot 1) + a) \cdot 1)) = (1 \cdot ((1 \cdot ((a + (c + d)) + 0)) + (1 \cdot a)))$

Theorem 27
Premises: $(c + d) = (b + c)$
Goal: $(1 \cdot ((((b + c) \cdot c) + (b \cdot (b + c))) + (c + d))) = ((((b + c)^2) + (b + c)) \cdot 1)$

Theorem 28
Premises: $((1 \cdot b) + d) = b$
Goal: $(((((1 \cdot b) + b) \cdot (a \cdot 1)) \cdot (((b + ((1 \cdot b) + d)) \cdot a) \cdot 1)) + 0) = ((((b + (1 \cdot b)) \cdot (a \cdot 1))^2) \cdot 1)$

Theorem 29
Goal: $(((b \cdot 1) + 0) \cdot (1 \cdot 0)) = (((b \cdot 1) \cdot ((-(0 + b)) + (1 \cdot b))) + (0 \cdot ((-(0 + b)) + (1 \cdot b))))$

Theorem 30

Goal: $(1 \cdot 1) = (((((a \cdot (c + c)) + 0) \cdot (b \cdot (c + c))) \cdot (\frac{1}{((((a \cdot c) + (a \cdot c)) \cdot b) \cdot (c + c))})) + 0)$

Theorem 31
Goal: $((1 \cdot (b \cdot b)) \cdot b) = (1 \cdot (0 + (((0 + b) \cdot b) \cdot b)))$

Theorem 32
Goal: $(((c \cdot (c \cdot 1)) + 0) \cdot 1) = (((c \cdot c) + 0) \cdot 1)$

Theorem 33
Goal: $1 = (1 \cdot (\frac{1}{((1+0) \cdot (\frac{1}{((b \cdot (\frac{1}{b})) + 0)}))}))$

Theorem 34
Goal: $(((((((c+a) \cdot a) \cdot (c+a)) \cdot c) \cdot (a+c)) \cdot (c+a)) \cdot (c+a)) = (((((((a+c) \cdot (c+a)) \cdot a) \cdot c) \cdot (a+c)) \cdot (c+a)) \cdot (c+a))$

Theorem 35
Goal: $0 = ((-(1 \cdot 0)) + ((-(c + c)) + ((1 \cdot c) + c)))$

Theorem 36
Goal: $1 = (1 \cdot (\frac{1}{(a \cdot (\frac{1}{(((a+c)+a)+(-(c+a)))}))}))$

Theorem 37
Premises: $(a + d) = a$; $((\frac{1}{c}) + e) = b$
Goal: $(((1 \cdot (1 \cdot (\frac{1}{(c \cdot (\frac{1}{c}))}))) + a) + b) = (1 \cdot (((1 \cdot 1) + (a + d)) + ((\frac{1}{c}) + e)))$

Theorem 38
Goal: $0 = ((b \cdot (b + (-b))) + (-(((0 + 0) \cdot b) + 0)))$

Theorem 39
Goal: $(((1 \cdot c) + (-(1 \cdot (c \cdot 1)))) \cdot 1) = ((0 \cdot 1) \cdot 1)$

Theorem 40
Goal: $((a + b) \cdot (1 \cdot ((b \cdot c) + (c \cdot c)))) = ((a \cdot ((c \cdot c) + (b \cdot c))) + (b \cdot ((c \cdot c) + (b \cdot c))))$

Theorem 41
Goal: $(0 + ((0 + ((c + c) \cdot c)) \cdot (a \cdot b))) = (0 + ((((c \cdot c) \cdot a) + ((c \cdot c) \cdot a)) \cdot b))$

Theorem 42
Premises: $(0 + d) = 1$
Goal: $((((1 \cdot 0) + (a + (a \cdot 1))) + 0) + d) = (((((1 \cdot a) + (-(a \cdot 1))) + a) + (a \cdot 1)) + 1)$

Theorem 43
Premises: $(b + d) = 0$
Goal: $0 = ((((((0 + b) \cdot 0) + ((0 + b) \cdot b)) \cdot 1) + 0) + (-((((b \cdot 0) + (b \cdot b)) + (b + d)) \cdot 1)))$

Theorem 44
Goal: $((0 + c) \cdot ((-c) + (((c \cdot 1) + 0) + (-c)))) = (((0 + c) \cdot (-c)) + ((0 + c) \cdot 0))$

Theorem 45
Goal: $0 = (0 + (-(((0 \cdot 0) + (a \cdot 0)) + (-(((((((a \cdot b) + (a \cdot b)) + ((b + b) + b)) + (-(((a \cdot (b + b)) + (b + b)) + b))) + a) \cdot 0) \cdot 1)))))$

Theorem 46

Premises: $((a + b) + d) = (a + b)$; $(b + e) = a$
Goal: $(a \cdot a) = (1 \cdot (a \cdot a))$

Theorem 47
Premises: $(c + d) = c$
Goal: $((b \cdot (1 + 0)) + (b \cdot (c + d))) = (0 + (b \cdot ((0 + (1 \cdot (\frac{1}{((b+(c+d))\cdot(\frac{1}{(b+c)}))}))) + (c + d))))$

Theorem 48
Goal: $((b + ((((a + a) \cdot 1) \cdot a) + 0)) \cdot a) = ((b \cdot a) + ((((a \cdot a) \cdot 1) + (a \cdot (a \cdot 1))) \cdot a))$

Theorem 49
Goal: $(((1 + b) \cdot (((a \cdot ((c \cdot 1) + (c^2))) + 1) + b)) + ((1 + b) \cdot (((a \cdot ((c \cdot 1) + (c^2))) + 1) + b))) = ((((1 + b) + (1 + b)) \cdot ((((a \cdot (c \cdot 1)) + (a \cdot (c \cdot (c \cdot 1)))) + (1 + b)) \cdot 1)) \cdot 1)$

Theorem 50
Goal: $0 = (((((0+(((c\cdot c)+(c\cdot c))+((c+c)+(c\cdot c))))+0)+c)\cdot a)+(-(((0+((((c+c)\cdot c)+(c+c))+(c\cdot c)))\cdot a)+(c\cdot a))))$

Inequality theorems

Theorem 1
Premises: $(1 + d) \geq 0$; $(b + e) \geq 0$
Goal: $((((1 + 1) \cdot (a \cdot (\frac{1}{a}))) \cdot (1 + d)) + (b + e)) \geq ((((1 \cdot 1) + (1 \cdot 1)) \cdot (1 + d)) + 0)$

Theorem 2
Goal: $(b^2) \geq (0 + (b \cdot (1 \cdot b)))$

Theorem 3
Premises: $((c + 0) + d) \geq 0$; $(d + e) \geq b$
Goal: $((c \cdot ((c + 0) + d)) + (d + e)) \geq (((0 + c) \cdot ((c + 0) + d)) + b)$

Theorem 4
Goal: $(b + 0) \geq ((((0 + b) + c) + c) + (-(c + c)))$

Theorem 5
Premises: $(1 + d) \geq 0$
Goal: $((((c \cdot c) + c) + a) \cdot (1 + d)) \geq ((((c^2) + (c + a)) \cdot 1) \cdot (1 + d))$

Theorem 6
Premises: $(b + d) = b$
Goal: $1 \geq ((((a + b) + (-(b + d))) \cdot ((a + b) + b)) \cdot (\frac{1}{((a\cdot(a+b))+(a\cdot b))}))$

Theorem 7
Premises: $((0 + a) + d) = 0$
Goal: $(((0 + a) \cdot a) + ((0 + a) + d)) \geq ((a^2) + 0)$

Theorem 8
Premises: $(b + d) = a$
Goal: $((c \cdot b) + (b \cdot b)) \geq (1 \cdot ((((c + a) + (-(b + d))) + b) \cdot b))$

Theorem 9
Goal: $(1 \cdot ((((b \cdot (\frac{1}{b})) \cdot a) \cdot (1 \cdot 1)) + a)) \geq ((1 \cdot ((1 \cdot 1) \cdot (a \cdot (1 \cdot 1)))) + (1 \cdot a))$

Theorem 10
Premises: $(c + d) \geq 0$
Goal: $(b \cdot (c + d)) \geq ((((b + b) + 0) + (-b)) \cdot (c + d))$

Theorem 11
Goal: $(((b + 0) + (b + c)) + 0) \geq (((b + b) + c) + 0)$

Theorem 12
Goal: $((c \cdot (c + 0)) + 0) \geq ((c^2) + 0)$

Theorem 13
Goal: $(1 \cdot (b \cdot 1)) \geq ((1 \cdot b) \cdot 1)$

Theorem 14
Goal: $1 \geq ((((b \cdot (\frac{1}{b})) + (\frac{1}{b})) + 1) \cdot (\frac{1}{((1+(\frac{1}{b}))+1)}))$

Theorem 15
Goal: $1 \geq ((\frac{1}{((c \cdot a) \cdot (\frac{1}{(a \cdot c)}))}) \cdot 1)$

Theorem 16
Goal: $((c \cdot (a \cdot a)) + (((a \cdot a) + (c \cdot a)) \cdot (a \cdot a))) \geq (0 + ((c + (0 + ((a + c) \cdot a))) \cdot (a \cdot a)))$

Theorem 17
Goal: $(((c \cdot b) + a) \cdot ((c \cdot b) + (c \cdot b))) \geq ((a \cdot ((c \cdot b) + (c \cdot b))) + ((c \cdot b) \cdot ((c \cdot b) + (c \cdot b))))$

Theorem 18
Goal: $((a \cdot b) \cdot 1) \geq ((((a \cdot 1) \cdot b) \cdot 1) \cdot 1)$

Theorem 19
Goal: $a \geq ((a + c) + (-c))$

Theorem 20
Goal: $((c \cdot b) \cdot b) \geq (b \cdot (b \cdot c))$

Theorem 21
Premises: $(a + d) = a$; $((a + d) + e) \geq 0$; $(b + f) \geq (0 \cdot 0)$
Goal: $((((((c \cdot 0) + (0 \cdot 0)) + (a + d)) \cdot ((0 + ((c + 0) \cdot (a + (-a)))) + a)) \cdot ((a + d) + e)) + (b + f)) \geq ((0 \cdot ((a + d) + e)) + (0 \cdot 0))$

Theorem 22
Premises: $(c + d) \geq 0$; $((0 + 0) + e) \geq (0 + 0)$
Goal: $((((((0 + (c + (-c))) \cdot (-c)) \cdot (\frac{1}{((0 \cdot (-c)) + (0 \cdot (-c)))})) \cdot (0 + 1)) \cdot (c + d)) + ((0 + 0) + e)) \geq ((0 \cdot (c + d)) + (0 + 0))$

Theorem 23
Premises: $((a^2) + d) \geq 0$
Goal: $((((a \cdot a) + c) \cdot (0 + (1 \cdot (a \cdot a)))) \cdot ((a^2) + d)) \geq ((((a \cdot a) \cdot ((a^2) + 0)) + (c \cdot ((a^2) + 0))) \cdot ((a^2) + d))$

Theorem 24
Premises: $(c + d) = c$; $((0 + a) + e) \geq a$

Goal: $((((a+b)\cdot(((a+(-a))+(a+b))+(c+d)))\cdot((((((0+a)+b)+c)\cdot(a+b))\cdot 1))+((0+a)+e))\geq(0+a)$

Theorem 25
Goal: $1\geq((a\cdot(c+b))\cdot(\frac{1}{((a\cdot c)+(a\cdot b))}))$

Theorem 26
Premises: $(a+d)\geq b$
Goal: $((0\cdot((((((a+c)+a)\cdot(a\cdot c))\cdot(a\cdot c))+((a\cdot c)\cdot(a\cdot c)))+(-((((((a+(c+a))\cdot a)\cdot c)+(a\cdot c))\cdot(a\cdot c))+0))))+(a+d))\geq(0+b)$

Theorem 27
Premises: $((c\cdot b)+d)=(b\cdot b)$; $((b\cdot b)+e)\geq a$
Goal: $((((b+b)+(b+b))\cdot((((c\cdot(b\cdot b))+b)+b)+(b^2)))+((b\cdot b)+e))\geq((((b+b)\cdot((((c\cdot b)\cdot b)+(b+b))+((c\cdot b)+d)))+((b+b)\cdot((((c\cdot b)\cdot b)+(b+b))+((c\cdot b)+d))))+a)$

Theorem 28
Premises: $((b\cdot 0)+d)\geq c$
Goal: $((((b+(((0+c)+(0+c))+0))\cdot 0)\cdot((b\cdot 0)+(((0+c)+(0+c))\cdot 0)))+((b\cdot 0)+d))\geq(0+c)$

Theorem 29
Premises: $(a+d)\geq 0$
Goal: $((0\cdot((((c\cdot c)+(c\cdot 0))\cdot a)+(-(((c+0)\cdot((c+0)\cdot a))\cdot 1))))+(a+d))\geq(0+0)$

Theorem 30
Premises: $(a+d)\geq c$
Goal: $(((b\cdot(b\cdot 1))+(b\cdot c))+(a+d))\geq((0+(b\cdot((b\cdot 1)+c)))+c)$

Theorem 31
Goal: $(0+(0+(c+b)))\geq(0+((b+c)+0))$

Theorem 32
Goal: $(a+(a+0))\geq((((0+a)+0)+a)+0)$

Theorem 33
Premises: $((c+c)+d)\geq a$; $(d+e)\geq 0$; $((c+c)+f)\geq(0+a)$; $(b+g)\geq 0$
Goal: $(((((((c+c)+(c+c))\cdot((c+c)+(c+c)))+((c+c)+d))+(d+e))+((c+c)+f))+(b+g))\geq((((0+a)+0)+(0+a))+0)$

Theorem 34
Goal: $(((0+b)+c)+a)\geq(0+(0+(b+(c+a))))$

Theorem 35
Premises: $(a+d)\geq 0$; $(a+e)\geq(c\cdot c)$; $(e+f)\geq 0$; $(c+g)\geq 0$; $(c+h)\geq(c+g)$; $(c+i)\geq 0$
Goal: $(((((((c\cdot c)\cdot(a+d))+(a+e))\cdot(e+f))\cdot(c+g))+(c+h))\cdot(c+i))\geq((((((0\cdot(a+d))+(c\cdot c))\cdot(e+f))\cdot(c+g))+(c+g))\cdot(c+i))$

Theorem 36
Goal: $(1\cdot(1\cdot(1\cdot a)))\geq(1\cdot((a+0)+0))$

Theorem 37
Premises: $(b+d)\geq b$; $((c+b)+e)\geq c$; $(b+f)\geq a$; $(e+g)\geq(b+f)$
Goal: $(((c+(b+d))+(b+f))+(e+g))\geq(((((c+b)+c)+(-((c+b)+e)))+a)+(b+f))$

Theorem 38
Goal: $((a + (((b + c) \cdot (b + c)) + ((c + b) \cdot b))) \cdot ((c + b) + (c + b))) \geq ((((((b + c) \cdot (c + b)) + ((b + c) \cdot b)) + a) \cdot (c + b)) + ((((((b + c) \cdot (c + b)) + ((b + c) \cdot b)) + a) \cdot (c + b)))$

Theorem 39
Premises: $(c + d) = b$; $((c + b) + e) = (c + d)$; $(a + f) \geq 0$; $(0 + g) \geq 0$; $(g + h) \geq 0$; $(d + i) \geq 0$
Goal: $((((((c+(c+d))+((c+b)+e))\cdot(a+f))\cdot(0+g))\cdot(g+h))\cdot(d+i)) \geq ((((((c+b)+(c+d))\cdot(a+f))\cdot(0+g))\cdot(g+h))\cdot(d+i))$

Theorem 40
Goal: $((((c + a) \cdot b) \cdot b) + (a + c)) \geq ((a + c) + (((a + c) \cdot b) \cdot b))$

Theorem 41
Goal: $(((c + b) + (a + (c + b))) \cdot (\frac{1}{((((1\cdot c)+b)+a)+(c+b))})) \geq (1 \cdot 1)$

Theorem 42
Premises: $(c + d) = b$
Goal: $(((((c\cdot b)+(c^2))\cdot((b+c)\cdot(c\cdot b)))+(c+d))\cdot((((((c\cdot(b+c))\cdot(b+c))\cdot c)\cdot b)+b)) \geq ((((((c\cdot b)+(c^2))\cdot((b+c)\cdot(c\cdot b)))+(c+d))^2)$

Theorem 43
Premises: $(a + d) = b$; $(d + e) = a$; $(c + f) \geq 0$; $((b + b) + g) \geq 0$
Goal: $((1 \cdot (c + f)) \cdot ((b + b) + g)) \geq (((((b + b) + a) \cdot (\frac{1}{(0+((b+(a+d))+(d+e)))})) \cdot (c + f)) \cdot ((b + b) + g))$

Theorem 44
Goal: $(((((a \cdot 1) \cdot a) \cdot 1) \cdot b) + (((a \cdot 1) \cdot (a \cdot 1)) \cdot (a \cdot a))) \geq (1 \cdot ((((a \cdot a) \cdot 1) \cdot b) + (((a \cdot a) \cdot 1) \cdot (a \cdot a))))$

Theorem 45
Premises: $((c + 0) + d) \geq b$; $(1 + e) \geq a$
Goal: $((0 + ((c + 0) + d)) + (1 + e)) \geq (((0 + (-((c \cdot 1) + (-(c + 0))))) + b) + a)$

Theorem 46
Premises: $(c + d) \geq (a \cdot c)$
Goal: $(((1 \cdot (1 \cdot (a \cdot (a \cdot c)))) \cdot ((1 \cdot ((a \cdot a) \cdot c)) + 0)) + (c + d)) \geq (0 + (a \cdot c))$

Theorem 47
Premises: $(c + d) \geq c$
Goal: $((c \cdot (0 + c))^2) \geq (((0 + ((c \cdot (0 + c)) \cdot (c^2))) + c) + (-(c + d)))$

Theorem 48
Premises: $(a + d) = b$
Goal: $(1 \cdot ((b + b) + (-(1 \cdot (b + (a + d)))))) \geq (1 \cdot (0 \cdot 1))$

Theorem 49
Premises: $((c \cdot b) + d) = a$; $((c \cdot b) + e) \geq b$
Goal: $(((b \cdot b) \cdot (a \cdot (c \cdot b))) + ((c \cdot b) + e)) \geq ((((b \cdot b) \cdot a) \cdot (c \cdot b)) + b)$

Theorem 50
Goal: $(((a + c) \cdot (c + a)) + ((a \cdot (c + a)) + ((c \cdot c) + (c \cdot a)))) \geq (((a + c) \cdot ((c + a) + (c + a))) \cdot 1)$

Bibliography

[1] Manuel Eberl, Gerwin Klein, Andreas Lochbihler, Tobias Nipkow, Larry Paulson, and René Thiemann, eds. *Archive of Formal Proofs*. 2021. URL: https://www.isa-afp.org/index.html.

[2] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. "Neural Module Networks". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 39–48.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *Proceedings of ICLR*. 2015.

[4] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, and Christian Szegedy. "Learning to Reason in Large Theories without Imitation". In: *CoRR* abs/1905.10501 (2019). arXiv: 1905.10501. URL: http://arxiv.org/abs/1905.10501.

[5] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. "HOList: An Environment for Machine Learning of Higher Order Logic Theorem Proving". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 454–463. URL: http://proceedings.mlr.press/v97/bansal19a.html.

[6] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Yann Coscoy, David Delahaye, Daniel de Rauglaudre, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, et al. "The Coq proof assistant reference manual". In: *INRIA, version* 6.11 (1999).

[7] Joel Beeren, Matthew Fernandez, Xin Gao, Gerwin Klein, Rafal Kolanski, Japheth Lim, Corey Lewis, Daniel Matichuk, and Thomas Sewell. "Finite Machine Word Library". In: *Archive of Formal Proofs* (June 2016). https://isa-afp.org/entries/Word_Lib.html, Formal proof development. ISSN: 2150-914x.

[8] Francesco Bellucci and Ahti-Veikko Pietarinen. "Charles Sanders Peirce: Logic". In: *The Internet Encyclopedia of Philosophy*. 2015. URL: https://iep.utm.edu/peir-log/.

[9] Emily M. Bender and Alexander Koller. "Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Ed. by Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault. Association for Computational Linguistics, 2020, pp. 5185–5198. DOI: 10.18653/v1/2020.acl-main.463. URL: https://doi.org/10.18653/v1/2020.acl-main.463.

[10] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: A methodological tour d'horizon". In: *Eur. J. Oper. Res.* 290.2 (2021), pp. 405–421. DOI: 10.1016/j.ejor.2020.07.063. URL: https://doi.org/10.1016/j.ejor.2020.07.063.

[11] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. "The Tactician - A Seamless, Interactive Tactic Learner and Prover for Coq". In: *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*. Ed. by Christoph Benzmüller and Bruce R. Miller. Vol. 12236. Lecture Notes in Computer Science. Springer, 2020, pp. 271–277. DOI: 10.1007/978-3-030-53518-6_17. URL: https://doi.org/10.1007/978-3-030-53518-6%5C_17.

[12] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C Paulson. "Extending Sledgehammer with SMT solvers". In: *Proceedings of International Conference on Automated Deduction*. 2011.

[13] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. "Hammering towards QED". In: *J. Formaliz. Reason.* 9.1 (2016), pp. 101–148. DOI: 10.6092/issn.1972-5787/4593. URL: https://doi.org/10.6092/issn.1972-5787/4593.

[14] Chad E Brown, Bartosz Piotrowski, and Josef Urban. "Learning to Advise an Equational Prover". In: (2020). URL: http://aitp-conference.org/2020/abstract/paper_32.pdf.

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[16] Kevin Buzzard, Johan Commelin, and Patrick Massot. "Formalising perfectoid spaces". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*. Ed. by Jasmin Blanchette and Catalin Hritcu. ACM, 2020, pp. 299–312. DOI: 10.1145/3372885.3373830. URL: https://doi.org/10.1145/3372885.3373830.

[17] Kevin Buzzard, Chris Hughes, Kenny Lau, Amelia Livingston, Ramon Fernández Mir, and Scott Morrison. "Schemes in Lean". In: *arXiv preprint arXiv:2101.02602* (2019).

[18] Michael Chang, Abhishek Gupta, Sergey Levine, and Thomas L. Griffiths. "Automatically Composing Representation Transformations as a Means for Generalization". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=B1ffQnRcKX.

[19] François Charton, Amaury Hayat, and Guillaume Lample. "Deep Differential System Stability - Learning advanced computations from examples". In: *CoRR* abs/2006.06462 (2020). arXiv: 2006.06462. URL: https://arxiv.org/abs/2006.06462.

[20] Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. "A survey on dialogue systems: Recent advances and new frontiers". In: *Acm Sigkdd Explorations Newsletter* 19.2 (2017), pp. 25–35.

[21] The mathlib community. "The lean mathematical library". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*. Ed. by Jasmin Blanchette and Catalin Hritcu. ACM, 2020, pp. 367–381. DOI: 10.1145/3372885.3373824. URL: https://doi.org/10.1145/3372885.3373824.

[22] Alexis Conneau and Guillaume Lample. "Cross-lingual Language Model Pretraining". In: *Advances in Neural Information Processing Systems, NeurIPS 2019, Vancouver, BC, Canada, December 8-14, 2019*. 2019, pp. 7057–7067. URL: http://papers.nips.cc/paper/8928-cross-lingual-language-model-pretraining.

[23] Maxwell Crouse, Ibrahim Abdelaziz, Cristina Cornelio, Veronika Thost, Lingfei Wu, Kenneth Forbus, and Achille Fokoue. "Improving Graph Neural Network Representations of Logical Formulae with Subgraph Pooling". In: *arXiv preprint arXiv:1911.06904* (2019).

[24] Ádám Darvas, Reiner Hähnle, and David Sands. "A theorem proving approach to analysis of secure information flow". In: *International Conference on Security in Pervasive Computing*. Springer. 2005, pp. 193–209.

[25] Eyal Dechter, Jonathan Malmaud, Ryan P. Adams, and Joshua B. Tenenbaum. "Bootstrap Learning via Modular Concept Discovery". In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. Ed. by Francesca Rossi. IJCAI/AAAI, 2013, pp. 1302–1309. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6890.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/n19-1423. URL: https://doi.org/10.18653/v1/n19-1423.

[27] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. "Unified Language Model Pre-training for Natural Language Understanding and Generation". In: *Advances in Neural Information Processing Systems, NeurIPS 2019, Vancouver, BC, Canada, December 8-14, 2019*. 2019, pp. 13063–13075.

[28] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for

Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[29] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL: https://openreview.net/forum?id=YicbFdNTTy.

[30] David Steven Dummit and Richard M Foote. *Abstract algebra*. Vol. 3. Wiley Hoboken, 2004.

[31] Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. "A metaprogramming framework for formal verification". In: *Proc. ACM Program. Lang.* 1.ICFP (2017), 34:1–34:29. DOI: 10.1145/3110278. URL: https://doi.org/10.1145/3110278.

[32] Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Josh Tenenbaum. "Learning Libraries of Subroutines for Neurally-Guided Bayesian Program Induction". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett. 2018, pp. 7816–7826. URL: https://proceedings.neurips.cc/paper/2018/hash/7aa685b3b1dc1d6780bf36f7340078c9-Abstract.html.

[33] Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Luc Cary, Lucas Morales, Luke B. Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. "DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning". In: *CoRR* abs/2006.08381 (2020). arXiv: 2006.08381. URL: https://arxiv.org/abs/2006.08381.

[34] Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. "Can Neural Networks Understand Logical Entailment?" In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=SkZxCk-0Z.

[35] Matthias Fey and Jan Eric Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *CoRR* abs/1903.02428 (2019). arXiv: 1903.02428. URL: http://arxiv.org/abs/1903.02428.

[36] Bernd Finkbeiner, Christopher Hahn, Markus N. Rabe, and Frederik Schmitt. "Teaching Temporal Logics to Neural Networks". In: *CoRR* abs/2003.04218 (2020). arXiv: 2003.04218. URL: https://arxiv.org/abs/2003.04218.

[37] M. Ganesalingam and W. T. Gowers. "A Fully Automatic Theorem Prover with Human-Style Output". In: *J. Autom. Reason.* 58.2 (2017), pp. 253–291. DOI: 10.1007/s10817-016-9377-1. URL: https://doi.org/10.1007/s10817-016-9377-1.

[38] Alexander L. Gaunt, Marc Brockschmidt, Nate Kushman, and Daniel Tarlow. "Differentiable Programs with Neural Libraries". In: *ICML*. 2017.

[39] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. "Initial Experiments with Statistical Conjecturing over Large Formal Corpora". In: *Joint Proceedings of the FM4M, MathUI, and ThEdu Workshops, Doctoral Program, and Work in Progress at the Conference on Intelligent Computer Mathematics 2016 co-located with the 9th Conference on Intelligent Computer Mathematics (CICM 2016), Bialystok, Poland, July 25-29, 2016*. Ed. by Andrea Kohlhase, Paul Libbrecht, Bruce R. Miller, Adam Naumowicz, Walther Neuper, Pedro Quaresma, Frank Wm. Tompa, and Martin Suda. Vol. 1785. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 219–228. URL: http://ceur-ws.org/Vol-1785/W23.pdf.

[40] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. "Learning to Prove with Tactics". In: *CoRR* abs/1804.00596 (2018). arXiv: 1804.00596. URL: http://arxiv.org/abs/1804.00596.

[41] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. "Convolutional Sequence to Sequence Learning". In: *Proceedings of ICML*. 2017.

[42] Zarathustra Goertzel and Josef Urban. "Usefulness of Lemmas via Graph Neural Networks". In: (2019). URL: http://aitp-conference.org/2019/abstract/AITP_2019_paper_32.pdf.

[43] Victor BF Gomes, Martin Kleppmann, Dominic P Mulligan, and Alastair R Beresford. "Verifying strong eventual consistency in distributed systems". In: *Proceedings of the ACM on Programming Languages* (2017).

[44] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, et al. "A machine-checked proof of the odd order theorem". In: *International Conference on Interactive Theorem Proving*. Springer. 2013, pp. 163–179.

[45] Sébastien Gouëzel. "Subadditive cocycles and horofunctions". In: *Proceedings of the International Congress of Mathematicians*. 2017.

[46] Adam Grabowski, Artur Kornilowicz, and Adam Naumowicz. "Mizar in a nutshell". In: *Journal of Formalized Reasoning* 3.2 (2010), pp. 153–245.

[47] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. "A formal proof of the Kepler conjecture". In: *Forum of mathematics, Pi*. Vol. 5. Cambridge University Press. 2017.

[48] William L Hamilton, Rex Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *arXiv preprint arXiv:1706.02216* (2017).

[49] Jesse Michael Han. "Enhancing SAT solvers with glue variable predictions". In: *arXiv preprint arXiv:2007.02559* (2020).

[50] Jesse Michael Han and Floris van Doorn. "A formal proof of the independence of the continuum hypothesis". In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*. Ed. by Jasmin Blanchette and Catalin Hritcu. ACM, 2020, pp. 353–366. DOI: 10.1145/3372885.3373826. URL: https://doi.org/10.1145/3372885.3373826.

[51] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. "Proof Artifact Co-training for Theorem Proving with Language Models". In: *The First Mathematical Reasoning in General Artificial Intelligence Workshop, ICLR 2021* (2021). URL: https://mathai-iclr.github.io/papers/papers/MATHAI_23_paper.pdf.

[52] John Harrison. "HOL Light: A tutorial introduction". In: *International Conference on Formal Methods in Computer-Aided Design*. Springer. 1996, pp. 265–269.

[53] Vincent J. Hellendoorn, Charles Sutton, Rishabh Singh, Petros Maniatis, and David Bieber. "Global Relational Models of Source Code". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=B1lnbRNtwr.

[54] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. "Scaling Laws for Autoregressive Generative Modeling". In: *CoRR* abs/2010.14701 (2020). arXiv: 2010.14701. URL: https://arxiv.org/abs/2010.14701.

[55] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[56] Drew A Hudson and Christopher D Manning. "Compositional Attention Networks for Machine Reasoning". In: *International Conference on Learning Representations (ICLR)*. 2018.

[57] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. "DeepMath - Deep Sequence Models for Premise Selection". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett. 2016, pp. 2235–2243. URL: http://papers.nips.cc/paper/6280-deepmath-deep-sequence-models-for-premise-selection.

[58] Jan Jakubův and Josef Urban. "Hammering Mizar by Learning Clause Guidance". In: *arXiv preprint arXiv:1904.01677* (2019).

[59] Albert Q. Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. "LISA: Language models of ISAbelle proofs". In: *Under submission to AITP 2021* (2021).

[60] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. "CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2901–2910.

[61] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. "SpanBERT: Improving Pre-training by Representing and Predicting Spans". In: *Transactions of the Association for Computational Linguistics* 8 (2020), pp. 64–77. DOI: 10.1162/tacl_a_00300. URL: https://doi.org/10.1162/tacl_a_00300.

[62] Cezary Kaliszyk and Josef Urban. "Learning-assisted theorem proving with millions of lemmas". In: *J. Symb. Comput.* 69 (2015), pp. 109–128. DOI: 10.1016/j.jsc.2014.09.032. URL: https://doi.org/10.1016/j.jsc.2014.09.032.

[63] Cezary Kaliszyk, Josef Urban, and Jirı Vyskocil. "Lemmatization for Stronger Reasoning in Large Theories". In: *Frontiers of Combining Systems - 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015. Proceedings*. Ed. by Carsten Lutz and Silvio Ranise. Vol. 9322. Lecture Notes in Computer Science. Springer, 2015, pp. 341–356. DOI: 10.1007/978-3-319-24246-0_21. URL: https://doi.org/10.1007/978-3-319-24246-0%5C_21.

[64] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. "Scaling Laws for Neural Language Models". In: *CoRR* abs/2001.08361 (2020). arXiv: 2001.08361. URL: https://arxiv.org/abs/2001.08361.

[65] Christoph Kern and Mark R Greenstreet. "Formal verification in hardware design: a survey". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 4.2 (1999), pp. 123–193.

[66] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[67] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, et al. "seL4: Formal verification of an OS kernel". In: *Proceedings of the ACM SIGOPS symposium on Operating systems principles*. 2009.

[68] Marie-Anne Lachaux, Baptiste Rozière, Lowik Chanussot, and Guillaume Lample. "Unsupervised Translation of Programming Languages". In: *CoRR* abs/2006.03511 (2020). arXiv: 2006.03511. URL: https://arxiv.org/abs/2006.03511.

[69] Guillaume Lample and François Charton. "Deep Learning For Symbolic Mathematics". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=S1eZYeHFDS.

[70] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. "Object Recognition with Gradient-Based Learning". In: *Shape, Contour and Grouping in Computer Vision*. Berlin, Heidelberg: Springer-Verlag, 1999, p. 319. ISBN: 3540667229.

[71] Gil Lederman, Markus N. Rabe, Sanjit Seshia, and Edward A. Lee. "Learning Heuristics for Quantified Boolean Formulas through Reinforcement Learning". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL: https://openreview.net/forum?id=BJluxREKDB.

[72] Xavier Leroy. "Formal Verification of a Realistic Compiler". In: *Communications of the ACM* 52.7 (2009), pp. 107–115.

[73] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C. Paulson. "IsarStep: a Benchmark for High-level Mathematical Reasoning". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=Pzj6fzU6wkj.

[74] Chin-Yew Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.

[75] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. "Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems". In: *Proceedings of ACL*. 2017.

[76] Yang Liu and Mirella Lapata. "Hierarchical Transformers for Multi-Document Summarization". In: *Proceedings of ACL*. 2019.

[77] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[78] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. Ed. by Llus Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, and Yuval Marton. The Association for Computational Linguistics, 2015, pp. 1412–1421. DOI: 10.18653/v1/d15-1166. URL: https://doi.org/10.18653/v1/d15-1166.

[79] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. "The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=rJgMlhRctm.

[80] R. Thomas McCoy, E. Grant, P. Smolensky, T. Griffiths, and Tal Linzen. "Universal linguistic inductive biases via meta-learning". In: *Proceedings of CogSci* abs/2006.16324 (2020).

[81] William McCune. "Solution of the Robbins' problem". In: *Journal of Automated Reasoning* 19.3 (1997), pp. 263–276.

[82] Norman Megill and David A Wheeler. *Metamath: A Computer Language for Mathematical Proofs*. Lulu. com, 2019.

[83] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. "The Lean Theorem Prover (System Description)". In: *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*. Ed. by Amy P. Felty and Aart Middeldorp. Vol. 9195. Lecture Notes in Computer Science. Springer, 2015, pp. 378–388. DOI: 10.1007/978-3-319-21401-6_26. URL: https://doi.org/10.1007/978-3-319-21401-6%5C_26.

[84] Yutaka Nagashima. "Simple Dataset for Proof Method Recommendation in Isabelle/HOL". In: *International Conference on Intelligent Computer Mathematics*. 2020.

[85] Yutaka Nagashima and Yilun He. "PaMpeR: Proof Method Recommendation System for Isabelle/HOL". In: *CoRR* (2018). URL: http://arxiv.org/abs/1806.07239.

[86] Yutaka Nagashima and Julian Parsert. *Goal-Oriented Conjecturing for Isabelle/HOL*. 2018.

[87] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*. Vol. 2283. Springer Science & Business Media, 2002.

[88] OpenAI. *OpenAI Five*. https://blog.openai.com/openai-five/. 2018.

[89] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. "fairseq: A Fast, Extensible Toolkit for Sequence Modeling". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*. Ed. by Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh. Association for Computational Linguistics, 2019, pp. 48–53. DOI: 10.18653/v1/n19-4009. URL: https://doi.org/10.18653/v1/n19-4009.

[90] Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. "Graph Representations for Higher-Order Logic and Theorem Proving". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 2967–2974. URL: https://aaai.org/ojs/index.php/AAAI/article/view/5689.

[91] Isabel Papadimitriou and Dan Jurafsky. "Learning Music Helps You Read: Using Transfer to Study Linguistic Structure in Language Models". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 6829–6839. URL: https://www.aclweb.org/anthology/2020.emnlp-main.554.

[92] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "BLEU: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://www.aclweb.org/anthology/P02-1040/.

[93] Lawrence C Paulson. *Isabelle: A generic theorem prover*. Vol. 828. Springer Science & Business Media, 1994.

[94] Charles Sanders Peirce. *Reasoning and the logic of things: The Cambridge conferences lectures of 1898*. Harvard University Press, 1992.

[95] Frank Pfenning and Christine Paulin-Mohring. "Inductively Defined Types in the Calculus of Constructions". In: *Mathematical Foundations of Programming Semantics, 5th International Conference, Tulane University, New Orleans, Louisiana, USA, March 29 - April 1, 1989, Proceedings*. Ed. by Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt. Vol. 442. Lecture Notes in Computer Science. Springer, 1989, pp. 209–228. DOI: 10.1007/BFb0040259. URL: https://doi.org/10.1007/BFb0040259.

[96] George Pırlea and Ilya Sergey. "Mechanising blockchain consensus". In: *Proceedings of ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2018.

[97] Stanislas Polu and Ilya Sutskever. "Generative Language Modeling for Automated Theorem Proving". In: *CoRR* abs/2009.03393 (2020). arXiv: 2009.03393. URL: https://arxiv.org/abs/2009.03393.

[98] Markus N. Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. "Mathematical reasoning via self-supervised skip-tree training". In: *ICLR* (2021). URL: https://openreview.net/forum?id=YmqAnY0CMEy.

[99] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. "Learning Transferable Visual Models From Natural Language Supervision". In: ().

[100] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language models are unsupervised multitask learners". In: *OpenAI Blog*. 2019. URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[101] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. URL: http://jmlr.org/papers/v21/20-074.html.

[102] Tim Rocktäschel and Sebastian Riedel. "End-to-end Differentiable Proving". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, pp. 3788–3800. URL: https://proceedings.neurips.cc/paper/2017/hash/b2ab001909a8a6f04b51920306046ce5-Abstract.html.

[103] Germán Ros, Laura Sellart, Joanna Materzynska, David Vázquez, and Antonio M. López. "The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 3234–3243. DOI: 10.1109/CVPR.2016.352. URL: https://doi.org/10.1109/CVPR.2016.352.

[104] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson, 2020. ISBN: 9780134610993. URL: http://aima.cs.berkeley.edu/.

[105] Alex Sanchez-Stern, Yousef Alhessi, Lawrence K. Saul, and Sorin Lerner. "Generating correctness proofs with neural networks". In: *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2020, London, UK, June 15, 2020*. Ed. by Koushik Sen and Mayur Naik. ACM, 2020, pp. 1–10. DOI: 10.1145/3394450.3397466. URL: https://doi.org/10.1145/3394450.3397466.

[106] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. "Analysing mathematical reasoning abilities of neural models". In: *Proceedings of ICLR*. 2019.

[107] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20.1 (2008), pp. 61–80.

[108] Imanol Schlag, Paul Smolensky, Roland Fernandez, Nebojsa Jojic, Jürgen Schmidhuber, and Jianfeng Gao. "Enhancing the Transformer with Explicit Relational Encoding for Math Problem Solving". In: *CoRR* abs/1910.06611 (2019). arXiv: 1910.06611. URL: http://arxiv.org/abs/1910.06611.

[109] Peter Scholze. *Liquid tensor experiment*. https://xenaproject.wordpress.com/2020/12/05/liquid-tensor-experiment/. Formalization available at https://github.com/leanprover-community/lean-liquid. 2020.

[110] Daniel Selsam and Nikolaj Bjørner. "Guiding High-Performance SAT solvers with Unsat-Core Predictions". In: *International Conference on Theory and Applications of Satisfiability Testing*. Springer. 2019, pp. 336–353.

[111] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. "Learning a SAT Solver from Single-Bit Supervision". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019. URL: https://openreview.net/forum?id=HJMC%5C_iA5tm.

[112] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[113] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144. ISSN: 0036-8075. DOI: 10.1126/science.aar6404. eprint: https://science.sciencemag.org/content/362/6419/1140.full.pdf. URL: https://science.sciencemag.org/content/362/6419/1140.

[114] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of Go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359.

[115] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. "MASS: Masked Sequence to Sequence Pre-training for Language Generation". In: *36th International Conference on Machine Learning, ICML 2019, Long Beach, California, USA, June 9-15, 2019*. 2019. URL: http://proceedings.mlr.press/v97/song19d.html.

[116] Christian Sternagel and René Thiemann. "Executable Matrix Operations on Matrices of Arbitrary Dimensions". In: *Archive of Formal Proofs* (June 2010). https://isa-afp.org/entries/Matrix.html, Formal proof development. ISSN: 2150-914x.

[117] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*. 2014.

[118] Christian Szegedy. "A Promising Path Towards Autoformalization and General Artificial Intelligence". In: *Intelligent Computer Mathematics*. Ed. by Christoph Benzmüller and Bruce Miller. 2020.

[119] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the Inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[120] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1556–1566. DOI: 10.3115/v1/P15-1150. URL: https://www.aclweb.org/anthology/P15-1150.

[121] Josef Urban and Jan Jakubův. "First Neural Conjecturing Datasets and Experiments". In: *Intelligent Computer Mathematics*. Ed. by Christoph Benzmüller and Bruce Miller. Cham: Springer International Publishing, 2020, pp. 315–323. ISBN: 978-3-030-53518-6.

[122] Pashootan Vaezipoor, Gil Lederman, Yuhuai Wu, Chris J. Maddison, Roger B. Grosse, Edward A. Lee, Sanjit A. Seshia, and Fahiem Bacchus. "Learning Branching Heuristics for Propositional Model Counting". In: *CoRR* abs/2007.03204 (2020). arXiv: 2007.03204. URL: https://arxiv.org/abs/2007.03204.

[123] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[124] Oriol Vinyals, Igor Babuschkin, Wojciech Marian Czarnecki, Michaël Mathieu, Andrew Joseph Dudzik, Junyoung Chung, Duck Hwan Choi, Richard W. Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* (2019), pp. 1–5.

[125] Mingzhe Wang and Jia Deng. "Learning to Prove Theorems by Learning to Generate Theorems". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/d2a27e83d429f0dcae6b937cf440aeb1-Abstract.html.

[126] Mingzhe Wang and Jia Deng. "Learning to Prove Theorems by Learning to Generate Theorems". In: *CoRR* abs/2002.07019 (2020). arXiv: 2002.07019. URL: https://arxiv.org/abs/2002.07019.

[127] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. "Premise selection for theorem proving by deep graph embedding". In: *Advances in Neural Information Processing Systems*. 2017, pp. 2786–2796.

[128] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander J. Smola, and Zheng Zhang. "Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs". In: *CoRR* abs/1909.01315 (2019). arXiv: 1909.01315. URL: http://arxiv.org/abs/1909.01315.

[129] Qingxiang Wang, Chad Brown, Cezary Kaliszyk, and Josef Urban. "Exploration of neural machine translation in autoformalization of mathematics in Mizar". In: *Proceedings of ACM SIGPLAN International Conference on Certified Programs and Proofs* (2020).

[130] Alex Warstadt and Samuel R. Bowman. "Can neural networks acquire a structural bias from raw linguistic data?" In: *Proceedings of CogSci* (2020).

[131] Makarius Wenzel. *The Isabelle/Isar Implementation*. https://isabelle.in.tum.de/dist/Isabelle2020/doc/isar-ref.pdf. [Online; accessed 31-May-2020]. 2020.

[132] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1502.05698.

[133] Daniel Whalen. "Holophrasm: a neural Automated Theorem Prover for higher-order logic". In: *CoRR* abs/1608.02644 (2016). arXiv: 1608.02644. URL: http://arxiv.org/abs/1608.02644.

[134] Freek Wiedijk. *The De Bruijn Factor*. 2000. URL: http://www.cs.ru.nl/F.Wiedijk/factor/factor.pdf.

[135] Freek Wiedijk. *The seventeen provers of the world: Foreword by Dana S. Scott*. Vol. 3600. Springer, 2006.

[136] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* (2016). URL: http://arxiv.org/abs/1609.08144.

[137] Yuhuai Wu, Honghua Dong, Roger B. Grosse, and Jimmy Ba. "The Scattering Compositional Learner: Discovering Objects, Attributes, Relationships in Analogical Reasoning". In: *CoRR* abs/2007.04212 (2020). arXiv: 2007.04212. URL: https://arxiv.org/abs/2007.04212.

[138] Yuhuai Wu, Albert Jiang, Jimmy Ba, and Roger Grosse. "INT: An Inequality Benchmark for Evaluating Generalization in Theorem Proving". In: (2021). URL: https://openreview.net/forum?id=O6LPudowNQm.

[139] Yuhuai Wu, Markus N. Rabe, Wenda Li, Jimmy Ba, Roger B. Grosse, and Christian Szegedy. "LIME: Learning Inductive Bias for Primitives of Mathematical Reasoning". In: (2021).

[140] Yuhuai Wu, Jin Peng Zhou, Colin Li, and Roger Grosse. "REFACTOR: Learning to Extract Theorems from Proofs". In: *The First Mathematical Reasoning in General Artificial Intelligence Workshop, ICLR 2021* (2021). URL: https://mathai-iclr.github.io/papers/papers/MATHAI_22_paper.pdf.

[141] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How Powerful are Graph Neural Networks?" In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=ryGs6iA5Km.

[142] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. "What Can Neural Networks Reason About?" In: *ICLR 2020*. 2020.

[143] Kaiyu Yang and Jia Deng. "Learning to Prove Theorems via Interacting with Proof Assistants". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6984–6994. URL: http://proceedings.mlr.press/v97/yang19a.html.

[144] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. "Xlnet: Generalized autoregressive pretraining for language understanding". In: *Advances in Neural Information Processing Systems, NeurIPS 2019, Vancouver, BC, Canada, December 8-14, 2019*. 2019.

[145] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. "Hierarchical Attention Networks for Document Classification". In: *Proceedings of NAACL HLT*. Ed. by Kevin Knight, Ani Nenkova, and Owen Rambow. 2016.

[146] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization". In: *37th International Conference on Machine Learning, ICML 2020, Vienna, Austria, 2020*. Vol. 119. PMLR, 2020.

[147] Xingxing Zhang, Furu Wei, and Ming Zhou. "HIBERT: Document Level Pre-training of Hierarchical Bidirectional Transformers for Document Summarization". In: *Proceedings of ACL*. 2019.