Thesis for The Degree of Doctor of Philosophy

Theory Exploration

Automated Conjecturing for Programs and Proofs

Sólrún Halla Einarsdóttir

Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg
Gothenburg, Sweden, 2025

Theory Exploration
Automated Conjecturing for Programs and Proofs

Sólrún Halla Einarsdóttir

ISBN 978-91-8103-319-9
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 5776.
ISSN 0346-718X
DOI 10.63959/chalmers.dt/5776

Department of Computer Science and Engineering
Division of Data Science and AI
Chalmers University of Technology | University of Gothenburg
SE-412 96 Göteborg,
Sweden
Phone: +46(0)31 772 1000

Printed by Chalmers Digitaltryck,
Gothenburg, Sweden 2025.

"Certainly, let us learn proving, but also let us learn guessing."
- G. Pólya, Mathematics and Plausible Reasoning

Theory Exploration
Automated Conjecturing for Programs and Proofs

Sólrún Halla Einarsdóttir
Department of Computer Science and Engineering
Chalmers University of Technology | University of Gothenburg

Abstract

Theory exploration is an approach to automating the discovery of interesting and useful properties about computer programs and mathematical structures. Such properties can be used to guide automated and interactive reasoning. Coming up with new lemmas is often crucial in proof automation, and can provide vital assistance to a user of an interactive proof system. Generating properties that specify the behavior of a program is beneficial for software verification, testing, and debugging. Automated conjecturing is a challenging endeavor due to the vast search space and the difficulty in identifying the most interesting and useful properties. Developing effective conjecturing techniques is therefore critical for advancing both automated and interactive formal reasoning about programs and proofs.

In this thesis, we present novel symbolic and neuro-symbolic methods for theory exploration, along with the design, development, and evaluation of associated tools. First, we present a coinductive lemma discovery tool, the first system designed to automatically discover and prove lemmas about potentially infinite structures. Then, we integrate theory exploration and automated theorem proving in a state-of-the-art inductive proof system. Next, we introduce template-based theory exploration, which narrows the conjecturing search space and makes theory exploration faster and more targeted. In addition, we provide empirical evidence for the effectiveness of template-based theory exploration in finding interesting and useful lemmas for mathematical formalizations. Finally, we use Large Language Models (LLMs) for lemma conjecturing, both directly and as part of a neuro-symbolic template-based tool. We present the first neuro-symbolic lemma conjecturing tool that can automatically conjecture lemmas across all formalization domains.

Keywords

Theory Exploration, Conjecturing, Theorem Proving, Formalization, Automated Reasoning, Functional Programming, Proof Assistants, AI for Math, Induction, Coinduction

Acknowledgments

Firstly, to my supervisor Moa, thank you for taking me on and providing so much guidance and inspiration on this long journey! To my co-supervisor Nick, thank you for all the collaboration an support! And to Aarne, thank you for your advice and support as examiner.

In the later half of my PhD time I had the privilege of working with excellent external collaborators who came with great perspectives and skills. Thank you to Yousef, Emily, Martin, and Márton for your contributions to this thesis.

I would also like to thank my excellent local collaborators: Thank you to Johannes for your guidance and collaboration way back when, and to George for helping to lift the Lemmanaid load!

Many great colleagues, past and present, have brightened my workdays at CSE this past decade (or so). Special thanks go to my colleagues in the DSAI division and in the Formal Methods and Functional Programming units.

I am deeply grateful to all the friends who have helped me experience brief glimpses of life outside of being a doktorand and a småbarnsförälder in the past few years.

Finally, I would like to thank my family for their immeasurable support: My mother Eyja, my father Einar, Áslaug, Arngrímur, Védís, Iðunn, Edda, Una, and my grandparents and extended family in Iceland. Kærar þakkir!

During my PhD studies, my family has grown significantly. I would like to thank my husband Koen for always being so supportive, encouraging, and understanding. Thank you to Vincent and Sander, it's an honor to get to be your bonusmamma. Last but not least: Pieter Snæbjörn and Anna Eydís, thank you for all the joy and inspiration you give me every day. Þið eruð best!

This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation. The computation was enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

List of Publications

Appended publications

This thesis is based on the following publications:

Paper 1. Sólrún Halla Einarsdóttir, Moa Johansson, Johannes Åman Pohjola. *Into the Infinite - Theory Exploration for Coinduction*. Proceedings of AISC 2018.

Paper 2. Sólrún Halla Einarsdóttir, Márton Hajdu, Moa Johansson, Nicholas Smallbone, Martin Suda. *Lemma Discovery and Strategies for Automated Induction*. IJCAR 2024.

Paper 3. Sólrún Halla Einarsdóttir, Nicholas Smallbone, Moa Johansson. *Template-based Theory Exploration: Discovering Properties of Functional Programs by Testing*. IFL '20.

Paper 4. Sólrún Halla Einarsdóttir, Moa Johansson, Nicholas Smallbone. *LOL: A Library Of Lemma templates for data-driven conjecturing*. CICM 2022, WIP track.

Paper 5. Yousef Alhessi, Sólrún Halla Einarsdóttir, George Granberry, Emily First, Moa Johansson, Sorin Lerner, Nicholas Smallbone. *Lemmanaid: Neuro-Symbolic Lemma Conjecturing*. 2nd AI for Math Workshop @ ICML 2025.

Summary of contributions

The contributions to the appended papers by the author of this thesis are listed below.

Paper 1. All of the implementation work, the majority of the experimental/evaluation work, co-writing the text of the paper in collaboration with co-authors.

Paper 2. Implementation of the experimental pipeline, running experiments and analysis of results. Writing most of section 1 (Introduction) and section 4 (Evaluation), and overseeing the assembly of the different sections into a cohesive paper.

Paper 3. All of the implementation work, most of the experimental/evaluation work, and co-writing the text of the paper in collaboration with co-authors.

Paper 4. All the implementation and experimental/evaluation work. Co-writing the text of the paper.

Paper 5. Shared first-authorship with Yousef Alhessi. Designing the updated template language and the code used to extract templates to compile the training data set. Implementation of the symbolic engine for template instantiation in Isabelle. Co-writing the text of the paper in collaboration with co-authors.

Contents

Part I

Introductory Chapters

Chapter 1

Introduction

How can we automate the discovery of interesting and useful properties about computer programs and mathematical structures? Why would we want to know about such properties? And what do we mean by interesting and useful?

The topic of this thesis is automated conjecturing through theory exploration. Theory exploration is an approach to automatically discovering interesting and useful properties about the functions and data structures that appear in computer programs and mathematical theories. In the work presented in this thesis, we use theory exploration techniques to discover lemmas for mathematical formalization and proof automation, and to discover specifications for functional programs.

Mathematical formalization is the process of defining mathematical concepts and proving theorems about them within a formal computer system, where the system verifies that each step of reasoning is logically valid. In the context of mathematical formalization and theorem proving, automatically coming up with new conjectures and lemmas can help with proof automation or to guide an interactive proof process. In the context of software development, a program specification is necessary in order to verify program correctness using formal methods. Knowing what properties hold about the functions and data structures in a program can help us with specifying, verifying, testing, debugging, and extending the program.

1.1 What makes a conjecture interesting or useful?

To define what makes a conjecture interesting, we draw inspiration from the measures of interestingness suggested by Colton, Bundy, and Walsh in their survey on the notion of interestingness in automated mathematical discovery [22] from 2000, and consider the following characteristics:

1. *Empirical Plausibility and Applicability*. We are primarily interested in conjectures that are actually true and satisfied by a non-trivial subset of their domain. The more easily a counterexample that falsifies a conjecture can be found, the less interesting it is. Conversely, if a conjecture has preconditions, the more seldom they are satisfied the less interesting it is.

2. *Novelty and Surprisingness* Interesting conjectures should introduce new knowledge or present knowledge in a novel way. If a conjecture is redundant or trivial with respect to previously discovered conjectures, we deem it uninteresting. Instances of tautologies are totally unsurprising and therefore make for uninteresting conjectures.

3. *Comprehensibility* If the conjecturing system is intended to output conjectures for a user to read, conjectures that are easier to read and understand are of more interest.

4. *Utility*. If the user of a conjecturing system explicitly expresses interest in particular concepts or a particular kind of conjectures, conjectures about those concepts and of that kind must be interesting.

These characteristics guide the design of the conjecturing methods presented in this thesis. All of these methods are designed to generate conjectures about specific concepts, ensuring the utility of generated conjectures. The QuickSpec conjecturing tool [77], which we use in Papers 1 and 2, uses term generation and equivalence testing to establish the plausibility and applicability of generated conjectures. QuickSpec incorporates pruning methods to boost novelty and surprisingness and heuristics to improve comprehensibility. The template-guided method implemented in our tool RoughSpec, presented in Paper 3, places increased emphasis on the utility of generated conjectures by only searching for conjectures of specified shapes. This may in turn generate conjectures that are somewhat less novel, surprising, or applicable. In Paper 5 we present a neuro-symbolic conjecturing tool, Lemmanaid, which relies on a Large Language Model to implicitly capture notions of interestingness from its training data, without explicit guidance.

In a theorem proving setting, a conjecture's usefulness can be defined in the following way: *If the conjecture holds, does adding it as a lemma enable the proof of a goal theorem that we couldn't prove without it with the available proof methods?* We make use of this definition in Papers 1 and 2 included in this thesis, where our conjecturing system is embedded in a theorem prover. But how can we evaluate the quality of conjectures that are generated in a context where there is no goal of proving specific theorems? A measure we use to evaluate interestingness and usefulness throughout the work in this thesis is the following: *Does a human expert consider it interesting or useful?* We assume that the lemmas included in a formalization by an expert user were considered interesting or useful by that expert. If our system manages to automatically rediscover a lemma that an expert user chose to include in their formalization, we judge that it has succeeded in synthesizing some of the intuition and motivation that led the expert user to include that lemma.

1.2 The Conjecturing System

We present our vision of a conjecturing system: A user is working on a mathematical formalization, or writing a program, and has defined some functions, some datatypes, perhaps some theorems about those functions and datatypes. Our conjecturing system takes these definitions as input, and outputs conjectures that are likely to be true and useful lemmas in this context. These conjectures help the user to make progress in their formalization, or better understand the behavior of the program they've defined. Such a conjecturing system, alongside automatic proof methods for the conjectured lemmas, could be an important piece of a formalization copilot system, where human mathematicians and AI systems can collaborate to formalize mathematical theories (and discover new mathematical theories) efficiently and effectively, as envisioned by mathematician Terence Tao and others [29].

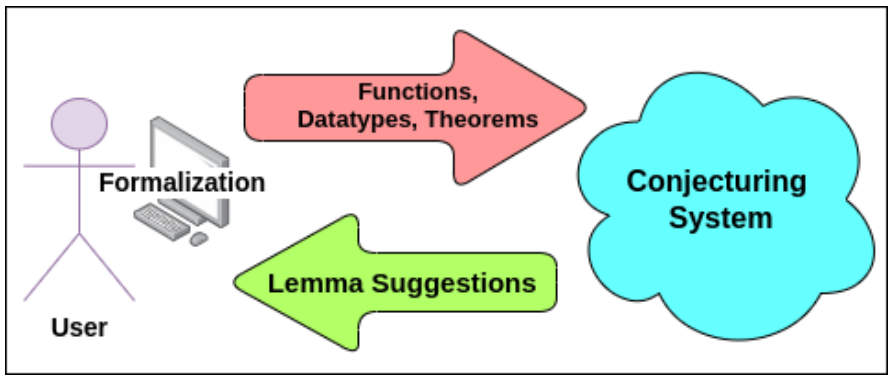

Figure 1.1: Our vision of a conjecturing system.

As a simple example, when asked to generate conjectures about addition and multiplication on natural numbers, our theory exploration system RoughSpec will produce the output shown in Figure 1.2 in less than half a second. The generated conjectures include identity, commutativity, distributivity, and associativity properties. In Paper 2 we show how RoughSpec can be used to discover not only simple arithmetic properties, but specifications for various Haskell software, such as a window manager and a large library of list operations.

```
Searching for fix-point/id properties...
  1. x * 0 = 0
Searching for left-id-elem properties...
  2. 0 + x = x
  3. 1 * x = x
Searching for right-id-elem properties...
  4. x + 0 = x
  5. x * 1 = x
Searching for commutative properties...
  6. x + y = y + x
  7. x * y = y * x
Searching for operator-commutative properties...
  8. x + (y + z) = y + (x + z)
  9. x * (y * z) = y * (x * z)
Searching for distributive properties...
 10. x * (y + z) = (x * y) + (x * z)
Searching for associative properties...
 11. (x + y) + z = x + (y + z)
 12. (x * y) * z = x * (y * z)
```

Figure 1.2: Arithmetic properties discovered by RoughSpec.

For further examples, consider some of the lemmas conjectured by our neuro-symbolic conjecturing tool LEMMANAID, which conjectures lemmas for formalizations in the Isabelle proof assistant. Given the function symbols, definitions and types used in a verified implementation of Schönhage-Strassen multiplication and verified proof of its asymptotic complexity [72], LEMMANAID conjectures the correctness of the multiplication implementation, as well as various lemmas that are used in the proofs of the implementation's correctness and its complexity, a few of which are shown in Figure 1.3.

$$\textit{to-nat}\ (\textit{schoenhage-strassen-mul}\ a\ b) = \textit{to-nat}\ a * \textit{to-nat}\ b$$
$$Idl_{\mathcal{Z}}\{m\} \cap Idl_{\mathcal{Z}}\{n\} = Idl_{\mathcal{Z}}\{lcm\ m\ n\}$$

$$a \in \textit{nt-lsbf-fermat.fermat-non-unique-carrier}\ m \Longrightarrow$$
$$b \in \textit{nt-lsbf-fermat.fermat-non-unique-carrier}\ m \Longrightarrow$$
$$\textit{val}\ (\textit{schoenhage-strassen-tm}\ m\ a\ b) = \textit{schoenhage-strassen}\ m\ a\ b$$

$$n > 0 \Longrightarrow \textit{real}\ (\textit{bitsize}\ n) \leq \textit{log}\ 2\ (\textit{real}\ n) + 1$$

Figure 1.3: Some lemmas conjectured by LEMMANAID.

1.3 Research Questions

The overarching research questions addressed in this thesis are the following:

1. How can we develop and apply theory exploration techniques to discover interesting and useful properties in the context of programming and proving?

2. Can template-based theory exploration be used effectively for functional program specification and mathematical formalization?

3. How can we combine neural and symbolic methods for theory exploration?

1.4 Contributions of this thesis

In the work compiled in this thesis, we design, develop, and evaluate tools to automatically discover interesting and useful properties. We present novel methods, both symbolic and neuro-symbolic. We demonstrate how our tools can be used to discover the lemmas needed to automate inductive and coinductive proofs, to automatically specify properties of functional programs, and to discover lemmas in formalizations for a wide variety of domains. We use large language models (LLMs) for lemma conjecturing both directly and as part of a neuro-symbolic tool in a novel approach, and develop the first neuro-symbolic lemma conjecturing tool that can automatically conjecture lemmas for any theory formalized in Isabelle/HOL. The contributions of the five papers included in the thesis are listed below:

The first coinductive lemma discovery tool

In Paper 1, we present Cohipster, a coinductive lemma discovery tool for the Isabelle proof assistant. Cohipster uses theory exploration to discover equational lemmas about corecursive functions and proves them using automated proof tactics. Cohipster is the first system to automatically discover and prove coinductive lemmas. Its design and implementation required the development of new techniques for automatically testing the equivalence of infinite structures, as well as for automating coinductive proofs.

A state of the art inductive proof system

In Paper 2, we integrate the state-of-the-art theory exploration system QuickSpec with the state-of-the-art automated theorem prover Vampire to create a hybrid inductive proof system. We present a method that adds lemmas conjectured by theory exploration to inductive problems as part of an automated proof pipeline. We train strategy schedules specialized for inductive proofs with and without auxiliary lemmas to optimize the prover's parameter selection. By combining strategy training and theory exploration we achieve state of the art performance on inductive proof benchmarks, finding more proofs than any previous method.

Template-based theory exploration

In Papers 3-5 we present and develop template-based theory exploration, where the search space is restricted to specific shapes of properties. A template is an abstraction describing a family of properties that have a particular common shape, such as distributivity or commutativity. Limiting the search space to specific shapes of properties makes theory exploration faster, more tractable, targeted and versatile.

- In Paper 3 we present template-based theory exploration and the tool RoughSpec. RoughSpec is a template-based theory exploration system for Haskell programs that allows users to specify what kinds of properties they are interested in finding by using templates. RoughSpec generates specifications that are small and easy for the user to understand. It allows the user to target specific families of properties they are interested in, or use built-in default templates to quickly generate a rough specification of their program's behavior. RoughSpec scales up to larger APIs much better than previous systems. RoughSpec is also complementary to the previous theory exploration system QuickSpec, and using them in combination efficiently produces specifications of a manageable size that contain interesting properties.

- In Paper 4 we provide a more robust empirical foundation for template-based theory exploration and move in the direction of a data-driven approach to finding good templates. We compile a collection of over 20,000 lemmas from the Isabelle Archive of Formal Proofs (AFP) and their corresponding templates. The AFP contains formalizations on a wide range of topics from mathematics, computer science, and logic, developed by experts, and is therefore a good source of interesting and diverse lemmas. The assumption behind RoughSpec was that certain lemma shapes are common and can be used to efficiently conjecture many (but not all) lemmas by analogy to known shapes. Our analysis of the dataset supports this assumption.

Neuro-symbolic theory exploration with an LLM

In Paper 5, we combine template-based theory exploration with LLMs in the neuro-symbolic lemma conjecturing tool Lemmanaid. We fine tune an LLM to come up with a good template for a given set of definitions, using a training data set similar to the one presented in Paper 4. We develop a new symbolic tool, similar to RoughSpec as presented in Paper 3, but implemented in the Isabelle proof assistant, to generate lemmas using the template from the LLM. We also develop a purely neural baseline, by using the LLM to directly conjecture lemmas instead of templates.

Lemmanaid successfully discovers interesting and useful lemmas in a wide variety of formalization domains from mathematics, computer science, and logic. It can be used to automatically generate conjectures about any set of Isabelle definitions and unlike our previous tools it is not restricted to generating

equational properties, nor does it require user supplied templates. LEMMANAID outperforms both purely neural and purely symbolic methods on our test sets, demonstrating the value of a neuro-symbolic approach.

1.5 Structure of thesis

The remainder of this thesis is organized as follows: In Chapter 2 we introduce relevant background material. In Chapter 3 we review the existing literature on lemma discovery and identify research gaps addressed by our work. In Chapter 4 we summarize the included papers. In Chapter 5 we summarize our main conclusions and discuss some ideas for future research directions. The papers are appended in the second part of this thesis, ordered as listed in Section 1.4.

Chapter 2

Background

2.1 What is Artificial Intelligence?

Traditionally, artificial intelligence (AI) has been divided into two approaches. On the one hand, there are symbolic methods, which are explicitly programmed to reason about their input to come up with a correct output. They are explicitly embedded with symbolic representations of knowledge and reasoning, in the form of logical inference rules and logical encodings of facts. One example of this is classical automated theorem proving, where the prover attempts to use a system of logical inference rules to deduce a proof of a given logical statement based on a set of logical axioms. On the other hand, there are data-driven machine learning methods. Such methods learn to solve tasks without explicit instructions about how to find the correct outcome, by using statistical algorithms to recognize patterns in a large set of examples.

In recent years much of the discussion of AI, both in popular media and in the research community, has been centered around a particular category of machine learning systems, namely generative AI models and in particular Large Language Models (LLMs). These models use deep neural networks based on the transformer architecture [85] to learn patterns from huge amounts of training data, and use the learned patterns to generate new output data based on given input prompts. They have achieved very impressive results in generating all kinds of text output as well as images, audio and video.

LLMs have also shown a capability to solve reasoning tasks, particularly when chain-of-thought prompting [93] is used to generate a series of intermediate reasoning steps, when continued pre-training on specialized technical datasets is used [56, 3], or when the model is fine-tuned for specific reasoning tasks with supervised fine-tuning on relevant data. More recently we have seen the release of so-called Large Reasoning Models, specialized LLMs specifically trained for reasoning tasks using reinforcement learning [67, 24]. Models specialized for reasoning are often modified to spend more time performing multi-step reasoning internally before outputting a response compared to a standard LLM. There is a great deal of ongoing development in improving the LLM performance on reasoning tasks, and they have already achieved remarkable

results in code generation tasks [46] as well as mathematical reasoning [91] and theorem proving [57].

Training a large generative AI model requires a huge training data set and a significant amount of computing power. For example, consider DeepSeek-V3 [25], a state of the art LLM released in December 2024. It contains 671 billion parameters, was trained on a dataset of 14.8 trillion tokens and consumed 56,000 petaflop/s-days of compute during its training. Each query to the model also incurs a computational cost, so larger scale or more complex querying is only available through a paid API or to users with access to the computational resources and know-how to host their own version of the model.

In contrast, specialized symbolic tools usually require very little computational power for their implementation and usage, but have a much more limited utility. For example, the tools we present in Papers 1 and 3 run on a standard laptop and produce results in a matter of seconds, and did not require any specialized hardware for their development. However, they are highly constrained in the kinds of output they can generate and the kind of input they can process. The input must be provided in a specific syntax and belong to the input space the tool is designed to reason about, and the output will never come from outside the predefined search space of outputs. For example, the tool Cohipster we present in Paper 1 will only produce equational properties of a limited size, about the subset of Isabelle/HOL functions and datatypes that can be translated to Haskell and for which we can automatically generate test data.

While large neural models are powerful and impressively versatile, they can not provide correctness guarantees for their results. They will come up with the most likely output based on the given input and the patterns learned from the training data, without verifying that the output is in fact correct. In order to guarantee correct output they must be combined with symbolic methods for verification. Various neuro-symbolic approaches have been developed to leverage deep neural models while using symbolic tools to verify output, particularly for mathematical reasoning and theorem proving tasks.

What then is the best approach to design an AI system that comes up with interesting and useful properties or lemmas? In the work presented in this thesis we use several different approaches ranging from mostly symbolic to using a LLM, detailed in Section 2.3. In Chapter 3 we examine approaches to lemma discovery from the literature.

2.2 AI for mathematics and theorem proving

In 1955, AI pioneers Newell and Simon presented a specification of the logic theorist [65], a system that generated proofs for logic theorems using heuristic methods *"similar to those that have been observed in human problem solving activity"* that could successfully prove many theorems from *Principia Mathematica* [94]. Newell and Simon were very optimistic about the development of AI that would surpass human abilities in mathematics as well as other domains, as were many of their peers at the time. In 1958 they predicted that *"within ten*

years a digital computer will discover and prove an important new mathematical theorem." [73].

This prediction certainly did not come true in the ten following years. Even now, 70 years later, it has not yet been realized in its entirety, although various qualified versions of the statement have come true. The early AI visionaries believed it would be straightforwardly possible to create systems that could simulate the reasoning abilities of a human, by programming them with the correct heuristic rules. This turned out not to be the case.

Despite not realizing Newell and Simon's prediction, many great advancements have been made since 1955 in how computers can be used to automate and assist mathematical reasoning. In Chapter 3 we examine previous approaches to lemma discovery more thoroughly. Here we briefly introduce some other concepts relevant to the work in this thesis.

2.2.1 Automated theorem proving

Automated theorem provers (ATPs) are systems designed to check whether a given logical conjecture follows from a given set of axioms. In Paper 2 we work with Vampire, a saturation-based ATP [81, 53]. Vampire is a state-of-the-art ATP and has won many awards CADE ATP System Competition (CASC) [82], the world cup of ATP systems, winning at least one division every year since 1999. In the 2025 CASC, it won first place in all eight theorem proving divisions for the first time. While Vampire was originally built to prove theorems in first-order logic, it has been extended for various other applications including inductive reasoning [39], which we make use of in our work.

2.2.2 Interactive theorem proving

While ATPs can reason very efficiently, the expressivity of the logic they can reason about is limited, so using them to prove more advanced mathematical theorems is a challenge. An interactive theorem prover (ITP), also called a proof assistant, is a tool for formalizing proofs about mathematics or other topics that can be expressed in a formal language. The ITP provides an interface between the user and a logical verification engine. This allows the user to define concepts, such as datatypes and functions, and verify statements about these concepts by building a proof of logical steps verified by the system. In this way the user can build verified formalizations about advanced mathematical concepts.

In Papers 1, 4 and 5 we work with the ITP Isabelle [66]. The instance we work with, Isabelle/HOL, includes built in tools and libraries for specification and reasoning in higher-order logic. A vast and constantly growing collection of Isabelle formalizations is available in the Archive of Formal Proofs [2], which makes a great source for data-driven applications which we take advantage of in Papers 4 and 5.

2.2.3 LLMs for math

Since the emergence of LLMs in this decade, we have a new way of using AI systems to engage with mathematical reasoning. We can give an LLM a query, in natural language, asking for a solution to a given problem or the proof of a given theorem, or even a formal proof in the language of an ITP like Isabelle. This has lead to some impressive results, for example an AI system achieved gold medal performance on the International Mathematical Olympiad [58] for the first time this year, which seemed far out of reach only a few years ago. If the rate of progress continues as it has, perhaps LLMs will be making advancements in research mathematics soon enough.

Much of the research on using LLMs for mathematical reasoning has been focused on autoformalization [92, 79, 95, 44], or translating natural language statements to a formal proof language used by an ITP. Another line of research is the generation of proofs in a formal language, either incrementally [45, 70], or by generating whole proofs at once [34]. However, there has not been much work on lemma discovery and conjecturing using LLMs. Although some success has been made on generating lemmas in a constrained setting [36, 68], based on a seed statement [28], and building a library of lemmas based on encountered proof goals [89], our tool LEMMANAID is the first to conjecture lemmas in a generalized setting and without any seed statement.

2.3 Our AI approach

The work in this thesis showcases a few different flavors of AI, ranging from fully symbolic to using LLMs. In Papers 1 and 3, we use AI for conjecturing with an approach that is not machine learning but also not purely symbolic, described in more detail in Section 3.1.3. It is in part data-driven but does not require us to provide a dataset for training and testing – rather we generate random test data using methods that are fast and efficient. A great benefit of this approach is that it allows us to generate new knowledge while using tools that can be run on a normal laptop and come up with results in seconds. In Paper 1 we also use purely symbolic methods to automatically prove the discovered lemmas.

In Paper 2, we combine the previous approach to conjecturing with the strong symbolic AI built in to the Vampire ATP, as well as a data-driven approach using random sampling and optimization to find an optimal strategy schedule for Vampire. Combining these methods made for a powerful automated inductive proof system.

While we achieved good results with these symbolic techniques, we were constrained to conjecturing equational properties. We could only generate conjectures about functions and datatypes that could be expressed in Haskell and for which we could automatically generate test data. We were therefore motivated to develop more versatile neural and neuro-symbolic conjecturing methods, to expand the range of concepts we could conjecture about and the shapes of conjectures we could generate. We hypothesized that the learned patterns embedded in deep neural models could help us find interesting and useful

conjectures, without us having to explicitly define what makes a conjecture interesting.

In Paper 4, we move in the direction of machine learning based conjecturing by compiling a potential training data set and examining the patterns that occur in the data. In Paper 5, we use an LLM to generate lemma conjectures directly, and also to generate lemma templates in combination with a symbolic system that instantiates templates into well-formed lemma conjectures.

Chapter 3

Literature on lemma discovery

There are many different possible approaches to the challenging task of discovering lemmas. A major dichotomy is whether the approach is top-down or bottom-up [48]. A top-down approach starts from a goal conjecture and aims to discover lemmas that will aid in the proof of that conjecture. A bottom-up approach is what we refer to as *theory exploration*, where lemmas about a given set of concepts are constructed without an explicit goal. Here the aim is rather to create a richer theory with new lemmas that may be useful in future proofs or help to unveil new theorems and concepts.

Another dichotomy is whether the approach inherently generates only true lemmas or whether it generates conjectures that may actually be false. The first case, where reasoning from known facts is used to generate provably true lemmas, we refer to as *lemma synthesis*. The second, where the generating system does not know whether the generated statements are true and they must be verified or falsified later on, we refer to as *conjecturing*.

Following is an overview of systems that generate conjectures and lemmas using various different methods, developed over the past 50 years. We discuss the conceptualization of these systems, how they determine what properties are interesting, and what they have been used for. We identify research gaps that are addressed by the work in this thesis.

3.1 Symbolic methods

First we consider various symbolic approaches to lemma discovery. We divide these up into purely symbolic methods based on heuristics and reasoning, methods that generate conjectures matching specific templates, and those that enumeratively generate terms or properties. Filtering out false conjectures by testing is commonly done in all three categories.

3.1.1 Conjecturing with heuristics and symbolic reasoning

Various systems have implemented purely symbolic conjecturing techniques. Many of these techniques use heuristic rules to mutate known facts or proof goals to generate new conjectures. Others involve logical deduction used to reason forward from given facts to synthesize new lemmas.

Bottom-up

The AM system [55] was an AI system designed to develop new mathematical concepts, developed in the 1970s. Guided by a large body of heuristic rules, it incrementally extends its knowledge base, eventually rediscovering various mathematical concepts and theorems. For example, starting with knowledge about elementary concepts such as sets, operations, and equality, AM discovered concepts corresponding to natural numbers, multiplication, and factors, and conjectured known relationships including the unique factorization theorem.

The GT system [30] for automating research in graph theory could discover, conjecture and prove theorems about properties of graphs. The conjectures were limited to certain set-theoretic relations between classes of graphs that had certain properties, such as one property subsuming another or two properties being equivalent. It could operate both independently, inventing properties from scratch starting from a graph definition, or with interactive guidance, conjecturing and proving theorems about user-defined properties.

Bagai et al. presented a discovery program for plane geometry in [4]. Geometric diagrams, involving points and lines and relations between points and/or lines, were represented with first order logical statements. Starting from an empty set, the system would add new points and relations to a previous diagram, and conjecture that the diagram resulting from each added relation was impossible to draw. The conjecture was sent to a theorem prover, and if the prover could not prove the conjecture the diagram was considered consistent and used to develop further concepts.

The HR system [21] was designed to generate concepts and conjectures in mathematical domains. It used a model finder combined with heuristic search rules to generate conjectures of interest, starting from a set of axioms. Its applications included lemma generation as well as generating benchmark theorems to test automatic theorem provers. It displayed some ability to invent interesting new mathematical knowledge, for example when applied to number theory it discovered interesting and previously undefined sequences [20].

MATHsAiD [60] is an automated theorem-discovery tool designed to assist a working mathematician, which has mainly been applied in the context of abstract algebra. It takes a set of axioms and definitions provided by the user and discovers new knowledge using a forward reasoning process, instantiating templates called theorem shells and generating theorems in parallel with their proofs. It uses a variety of heuristics constructed to reflect human mathematical intuition and knowledge. This system managed to rediscover a known but nontrivial theorem on Zariski spaces.

Top-down

Many automated theorem provers have incorporated goal-directed lemma discovery, in particular for proof by induction. Inductive proofs often require the introduction of auxiliary lemmas in order to be proved by automated proof methods. The reasoning required to prove the induction step, given the base case and the induction hypothesis, is often too complex for the automated proof techniques. The right lemma, often itself requiring a proof by induction, can bridge the gap by being more easily provable from the given facts, while enabling the proof of the original inductive step. In many cases a modified version of the inductive step proof goal makes for a useful lemma.

Boyer and Moore introduced the waterfall model to automate inductive proofs [9] and implemented it in their prover in the 1970s. The model includes lemma conjecturing by *generalization*, or replacing subterms in open proof goals by fresh variables. Generalization heuristics have been improved and refined in various systems [1, 42, 78]. These heuristics are used to determine when in the proof process generalization should be applied and which subterms should be generalized. Counterexample checking can be used to filter out false conjectures before trying to prove them. The waterfall model is the core concept in the design of the industrial inductive prover ACL2 [52].

Proof planning [13] was introduced by Bundy, as a method of planning a strategy of when and how various heuristics should be applied in a proof search. When a proof attempt fails, a *proof critic* technique is triggered to analyze the failed proof and try to come up with a useful lemma. The rippling [14] method for planning inductive proofs, implemented in the OYSTER-CLAM system [15, 43], includes lemma discovery critics that go beyond the capabilities of subterm generalization. These critics enable the construction of generalizations containing an additional new variable, as well as speculating lemmas more broadly if the proof gets stuck before applying the inductive hypothesis. These lemma discovery techniques were also implemented in the IsaPlanner system for automating inductive proofs in Isabelle/HOL [26, 27], but experimental evaluation found that they were rarely applicable and often inefficient [50].

The PGT system [63] is a goal-oriented conjecturing tool for Isabelle/HOL. Starting from a proof goal, generalization and mutation of the goal's subterms are used to generate conjectures. Counterexample checking is then used to filter out false conjectures.

Several approaches have been taken to extending saturation-based first-order theorem provers to support inductive reasoning [23, 87, 39]. These techniques have used generalization to conjecture lemmas. In [38], Hajdu et al. further improve the inductive reasoning capabilities of the saturation-based theorem prover Vampire by extending the superposition calculus with rewriting-based techniques. This method slightly relaxes the efficiency requirements of the calculus, allowing for equational rewriting to conjecture more lemmas that are proved by induction during the proof search by equational rewriting and proved by induction. However, this conjecturing method is still quite limited.

3.1.2 Template-based conjecturing

Several systems have incorporated some form of template-based conjecturing, where conjectures are based on user-provided or built-in templates. This is a straightforward approach to limiting the search space of bottom-up conjecturing.

Bottom-up

Graffiti [31] was a conjecture generation system for graph theory, developed in the 1980s. It attempts to generate conjectures of specific predetermined shapes, comparing them against a library of graphs such that a formula for which none of the library graphs provides a counterexample is considered a conjecture. Various heuristics are used to trim away trivial and uninteresting conjectures.

Buchberger [11] introduced the term "theory exploration" (earlier work used the term "theory formation") to describe the process of a mathematician at the beginning of work on a new theory, and his team implemented it in the Theorema [12] system. Theorema provides tools to assist the user in their theory exploration but does not automate the process, rather it is intended as a framework for mathematicians to more effectively work with an interactive theorem prover. Theorema introduced the concept of *knowledge schemas* that represent interesting mathematical knowledge and could be instantiated in new theories, but did not automate the instantiation process.

IsaScheme [61] is a theory exploration system for Isabelle/HOL. It makes use of user-provided templates (schemas) similarly to our approach in RoughSpec, presented in Paper 3. Users provide the schemas as well as a set of terms to instantiate the schemas with, and the instantiation is performed automatically. The generated conjectures are then automatically refuted using Isabelle/HOL's counter-example finders, or proved using the IsaPlanner [26, 27] prover.

The TBC system for template-based conjecturing [64] is another Isabelle tool that comes up with lemmas from an inductive proof goal. There, conjectures are generated by instantiating a template from a list of 16 predefined templates, using the functions that appear in the goal or those that appear in their definitions. The conjectures are refuted using counterexample checking, or proved using automated proof tactics.

3.1.3 Conjecturing by term generation

A number of systems have incorporated methods of conjecturing by enumerating terms or equations, often in combination with testing to filter out false conjectures.

Bottom-up – QuickSpec and Hipster

The work presented in this thesis builds on two theory exploration systems developed at Chalmers, QuickSpec [77] and Hipster [51, 47].

QuickSpec QuickSpec [77, 17] discovers equational properties about Haskell programs. It takes in a set of functions given by the user and generates all correctly typed terms, up to a given size limit, as potential left- or right-hand sides of equational properties. The terms are evaluated for a large set of randomly generated inputs, using the QuickCheck [16] property based testing tool. Terms that have the same value for every test input are then conjectured to be equal. While QuickSpec does not provide proofs that the generated properties hold, the extensive and sophisticated testing that takes place in the discovery process means that they are unlikely to be falsifiable.

Several clever design features help QuickSpec to be efficient and avoid generating redundant output. Pruning is interleaved with testing – a property that follows from previously discovered properties by equational reasoning will be pruned away before testing the relevant terms. QuickSpec imports the functionality for this reasoning from the equational theorem prover Twee [76]. QuickSpec also uses *schemas* to prioritize testing the most general versions of properties before considering their more specific instances. In this way QuickSpec avoids generating and considering redundant terms and avoids testing redundant properties, helping it live up to its name and work very quickly in many cases. The pruning also limits the amount of output so the user is not overwhelmed by redundant output properties, and each output property is more likely to be interesting to the user.

While QuickSpec was originally intended to help functional programmers generate formal specifications, it has demonstrated usefulness in a theorem proving context in the HipSpec [18] and Hipster systems. In HipSpec, QuickSpec is integrated into an inductive theorem prover. HipSpec uses QuickSpec to conjecture lemmas about the functions occurring in a given theorem statement and proves the lemmas before attempting to prove the goal theorem with the help of the lemmas. HipSpec achieved state of the art results on inductive proof benchmarks.

Hipster Hipster is a theory exploration system for Isabelle/HOL that uses QuickSpec as an external tool. Hipster comes up with conjectures about a set of functions given by the Isabelle user, and then uses proof tactics implemented in Isabelle to attempt to prove the generated lemma. The discovered lemmas and their proofs can then be imported into the user's formalization and used in further proofs.

Hipster discards a lemma as too trivial to be of interest if its proof is "too easy" or if it's a consequence of a previously discovered lemma. The definition of simple and complicated proof can be set by the user by selecting a suitable proof tactics to represent simple and complicated reasoning. For example, proofs that only require rewriting might be considered simple while proofs that require induction are considered to indicate an interesting lemma.

Other Bottom-up Approaches

IsaCoSy [49] is a theory exploration systems for Isabelle/HOL that uses term generation and testing. It generates equational terms iteratively, starting from

a minimal size and increasing to a given maximal size. In each iteration, it generates all possible type-correct terms of the current size that cannot be reduced by rewriting using known equations. At the end of the iteration, the generated conjectures are refuted using Isabelle/HOL's counter-example finders or proved using the IsaPlanner [26, 27] prover. Those that are proved are used to generate constraints for the next iteration. IsaCoSy demonstrated the ability to generate many useful and interesting lemmas, but often had a long runtime.

The SMT solver CVC4 [7] was extended to support inductive reasoning [71], including the generation of subgoals or lemmas in the proof search. The lemma conjecturing method is inspired by an older version of QuickSpec [17]. Potential equalities are enumerated and those that are irrelevant, trivial, or contradictory to known facts are filtered out. This extension was inherited by the current version cvc5 [6].

Speculate [10] is a theory exploration system designed to discover interesting properties about Haskell programs, like QuickSpec. It enumerates expressions involving a given collection of functions and uses testing to separate them into equivalence classes, using enumerative testing methods [59] rather than random testing. Speculate does not have built-in support for polymorphism like QuickSpec does, and has somewhat less effective pruning methods causing its performance to be slower. However, Speculate has better support for discovering inequalities and conditional properties than QuickSpec.

The TheSy [74] system uses comprehensive term generation to come up with equational conjectures. However rather than using testing their system is purely symbolic, evaluating the terms on symbolic inputs to form conjectures. It then uses an inductive theorem prover to attempt to prove the generated conjectures.

Top-down

The lfind system [75], implemented as a tactic in the Rocq proof assistant, generates conjectures using data-driven program synthesis techniques. Starting from a stuck proof, the proof goals are generalized, mutated, and evaluated on random concrete values in order to generate input-output examples for an example-guided synthesizer. The numerous candidate lemmas produced are then passed through several filters, including counterexample search to filter out false conjectures. The remaining candidates are ranked using an automated prover, with lemmas that can be used to automate the proof goal getting the highest ranking.

The CCLemma prover [54] uses goal-directed conjecturing to generate lemmas for equational inductive proofs. By using e-graphs and equality saturation to represent the space of all stuck proof states, their methods only suggest lemmas that will lead to progress in the proof.

3.1.4 Research Gaps

- While many of the methods listed above deal with the discovery of inductive lemmas and automating inductive proof, none of them are designed

to handle codatatypes and corecursive functions. Paper 1 addresses this gap and presents the first theory exploration system capable of handling infinite structures and discovering coinductive properties about them as well as generating coinductive proofs. Paper 1 also presents an approach to testing codatatypes and corecursive functions, the challenges of which are not addressed in previous work.

- In the above sections we list various systems that have been used to automate inductive proofs. However, improving the automation of inductive proof is still an open problem, with one of the main challenges being finding the right lemmas. The systems listed above all have limitations that cause them to fail to prove some relatively simple benchmark problems. In particular, while the cutting edge ATP Vampire has recently added support for inductive proving, it has very limited built-in support for lemma conjecturing. Paper 2 addresses this gap, presenting a hybrid automated inductive prover we created by combinining a state-of-the-art theory exploration system (QuickSpec) with the state-of-the-art ATP (Vampire) to create a hybrid inductive proof system. Our hybrid VampireSpec inductive prover manages more proofs from benchmark sets than any other prover we're aware of, including HipSpec, CVC4, Vampire without external lemma conjecturing, TheSy, and CCLemma.

- Previous theory exploration systems for Haskell programs, namely QuickSpec and Speculate, have lacked support for user control over the kinds of properties they find, and scaled poorly due to search space growth. We address this gap in Paper 3 with RoughSpec, which solves the problem of determining what conjectures are interesting by only considering conjectures matching templates defined by the user. RoughSpec is far more tractable and scales better to larger libraries than QuickSpec and Speculate.

  Previous methods for template-based conjecturing, described in Section 3.1.2, have been implemented within proof assistants. RoughSpec instead targets functional programs written in Haskell, and makes use of existing tool QuickSpec's highly optimized techniques for term generation, testing, and pruning. RoughSpec includes a library of default templates in addition to allowing the user to define their own templates, while previous template-based approaches have done one or the other.

- The methods for template-based conjecturing described in Section 3.1.2 are based on the assumption that templates are useful for lemma conjecturing, but lack empirical evidence supporting this hypothesis. They rely on user-provided templates without providing assistance to the user in knowing what template will be useful, or use a small set of default templates that may not generate the most interesting or useful conjectures. We address this gap in Paper 4 by compiling and analyzing a large and diverse dataset of lemmas and their corresponding templates. In Paper 5, we further address the open problem of knowing what template will

generate useful lemmas in a particular setting, by using a fine-tuned LLM to predict a useful template for a given setting.

- Symbolic lemma discovery methods as described above are generally restricted in the kinds of lemmas they can find, by heuristics as in Section 3.1.1, templates as in Section 3.1.2, or in the size and shape (mostly equational) of conjectures as in Section 3.1.3 We address this gap in Paper 5 with Lemmanaid, a conjecturing system that is unrestricted in the size and shapes of conjectures it can discover.

3.2 Methods based on machine learning

As machine learning techniques have improved and become ubiquitous in the past 15 years, various approaches to lemma discovery by machine learning have been developed. Here the distinction between top-down and bottom-up approaches becomes less clear, since a machine learning algorithm may in many cases be easily modified to either include or not include a proof goal in its input when it is trained to generate a lemma as output.

3.2.1 Conjecture generation by statistical machine learning

Top-down

Heras et al. [40] present a conjecturing method based on statistical machine learning, implemented in the ACL2(ml) system [41], an extension to ACL2 [52]. Clustering algorithms are used to create families of similar theorems from a proof library based on the term structure of the theorem statements. A symbolic lemma analogy system will conjecture lemmas that may be useful in the proof of a given theorem statement, based on lemmas that were used in proofs of similar theorem statements, called source lemmas. A mapping between symbols present in the current proof context and those in the source lemma is created with the help of clusters of the available definitions. The symbols in the source lemma are replaced by the analogous current symbols according to the mapping, with some potential mutations to the source lemma's term structure. Counter-example checking is used to filter out false conjectures. A user attempting to prove a goal theorem can let the system suggest auxiliary lemmas that are analogous to lemmas used in proofs of theorems similar to their goal.

Gauthier et al. [35] experimented with using statistical concept matching for conjecturing. The most similar sets of concepts in a given library were found by concept matching, and conjectures were generated by replacing concepts in a given theorem with their matching counterparts. This resulted in some useful lemmas for automated first-order theorem proving.

3.2.2 Conjecture generation by neural networks

Since deep neural networks have shown a great ability to recognize all kinds of different patterns, various approaches have been used to train such models to capture the intuition of what makes a useful lemma in a proof or what makes an interesting lemma about a given set of concepts.

Bottom-up

Urban and Jakubův [84] trained the language model GPT-2 on four different formats of the Mizar Mathematical Library [5]. They generated some novel and well-typed conjectures but the output also included duplicates from the training set as well as false statements. They found that the output was sensitive to temperature settings. When the temperature was low (i.e. less randomness in predictions) the model output only known lemmas from the training data, when the temperature increased the model started to also come up with novel true lemmas, but making the temperature too high lead to statements that were not syntactically valid.

Rabe et al. [69] trained a language model with a transformer architecture on mathematical formulas with a novel skip-tree training method, where the formulas are viewed as trees of subexpressions and the model's task is to predict a missing subset of a given tree. They attempted mathematical reasoning tasks such as generating a missing precondition for a given conditional statement, generating one side of an equation given the other side, and "free-form" conjecturing where the model is simply asked to generate an entire theorem statement. Between 13-30% of statements their system generated were both provable and new, while the remainder were either false or trivial consequences (ie. alpha-renamings or even exact copies) of statements from the training set.

Top-down

LEGO-Prover [89] builds a library of lemmas to improve the formalization capability of an LLM-based prover. A system combining two LLMs, the prover and the evolver, has the goal of generating formal Isabelle proofs based on a formal theorem statement paired with an informal natural language statement and a human-written informal proof. The prover model decomposes a goal theorem into possible subgoal lemmas and generates a proof step by step, potentially retrieving lemmas from the skill library. The evolver generates lemmas to store in the library, either by refining a subgoal lemma generated by the prover to make it more general and reusable, or by augmenting a lemma from the library.

Synthesizing theorems for training data

Since a lack of training data is often an obstacle for training a neural theorem prover in a particular domain, theorem synthesis has been used as a source of data to train on. One approach to this is to randomly generate parameters

to instantiate predefined templates for the relevant kinds of theorems. For example, Zombori et al. [97] randomly generate arithmetic statements of a specified shape, and Fawzi et al. [32] randomly generate graphs about which their prover learns to prove polynomial equalities.

Another approach is to synthesize theorems by inferring new statements from a given set of facts. Firoiu et al. [33] generate synthetic training data for a first order theorem prover. Their forward proposer algorithm infers new clauses from a given set of axiom clauses, using random sampling to select a branch of the deduction tree. Wang and Deng [90] propose a neural generator that learns to synthesize both theorems and their proofs, and evaluate it in the setting of intuitionistic logic and set theory in the Metamath proof language. The synthesized theorems are generated via forward reasoning from previously known theorems, using neural networks to select suitable proof trees and substitutions for the generation.

The synthesized training data for the AlphaGeometry system [83] for proving theorems about Euclidean geometry combined constrained random selection and symbolic logical deduction. A dataset of 100 million synthetic theorems and their proofs was constructed by randomly generating theorem premises from a set of possibilities, inputting them into a specialized symbolic deduction proof engine, and then extracting a theorem statement and its proof from the resulting dependency graph. Constructs that appear in the proof but are independent of the extracted goal theorem statement are identified as auxiliary constructs. Auxiliary construct definitions are moved from the theorem's set of premises to its proof, in order to train a generative model to come up with such constructs based on the relevant premises.

3.2.3 Reinforcement-based conjecturing

Several recent works use *reinforcement learning* to train a conjecturer. Such a system must implement a reasonable *reward function* to reward the generation of good (interesting or useful) conjectures.

Bottom-up

The Minimo system [68] learns conjecturing and proving as a reinforcement game. Using a language model to generate conjectures and a Monte Carlo Tree Search to search for proofs, the system bootstraps a mathematical theory from a set of basic axioms. The conjecturing agent is rewarded for generating lemmas that are challenging to prove but still provable by the proving agent.

The STP system [28] also uses a reinforcement loop to train a conjecturer and a prover. The conjecturer, based on a fine-tuned LLM, generates a conjecture from a given seed statement and its proof. The conjecturer's reward function is designed to reward conjectures that are provable while also challenging, diverse, and relevant. The goal of the conjecturer-prover loop is to improve the performance of the prover, and the end product is a whole-prove generating neural prover. Different instances of the system were trained on Lean syntax and Isabelle syntax and evaluated on theorem proving benchmarks.

Top-down

Gauthier and Urban [36] use a self-learning loop to train a neural machine translation model to generate useful lemmas for inductive proofs. Their domain is a large dataset of problems where the goal is to prove the equivalence of two different programs synthesized to describe the same integer sequence from the Online Encyclopedia of Integer Sequences [37]. They train the model to propose useful predicates for a given problem, based on a training set of problems paired with the predicates that lead to their successful proof. If the generated predicates enable a proof to be found using an SMT solver, the problem and predicates are added to the training data for the next iteration of the self-learning loop. This algorithm finds proofs for many more of the benchmark problems than comparable baseline proof methods.

3.2.4 Research Gaps

Conjecturing with neural methods is a fairly young and underexplored research area. A position paper published earlier this year, authored by several experts in the field of AI for mathematical reasoning [96], identified learning to construct new abstractions, such as lemmas, as a challenge and a promising future research direction. Previous work on neural conjecturing using LLMs has generated lemmas in a specialized setting, such as reasoning about programs that generate integer sequences [36] or bootstrapping mathematical theories in a simple proof environment [68]. Other recent work has used LLMs to generate related conjectures based on a given seed statement [28, 89].

In [84], the (somewhat primitive compared to available models now, 5 years later) LLM GPT-2 was used for bottom-up conjecturing in the Mizar language with some success. Since then, there has not been further work on broad neural conjecturing about mathematical formalizations in a bottom-up theory exploration setting. We address this gap in Paper 5 with our tool Lemmanaid, the first to conjecture lemmas in a generalized setting across all Isabelle/HOL formalizations and without any seed statement.

Chapter 4

Summary of Included Papers

4.1 Paper 1: Into the Infinite - Theory Exploration for Coinduction

We present an extension of theory exploration to coinductive theories, and a tool called Cohipster that can discover and prove coinductive properties in Isabelle/HOL. Cohipster is the first lemma discovery system designed to handle codatatypes and coinductive properties. We also present an approach to testing codatatypes and corecursive functions, the challenges of which are not addressed in previous work.

Coinduction and corecursion are the mathematical duals of induction and recursion and allow the specification of potentially infinite structures, for example streams, and functions that operate on such structures. They have many applications in theoretical computer science, having been used to define and verify the behavioral equivalence of processes, Hoare logic for non-terminating programs, total functional programming in the presence of non-termination, and properties of lazy datatypes in functional languages like Haskell. Developing techniques to test and reason about corecursive functions and coinductive properties is therefore a contribution to further enable the development and verification of such programs. The Isabelle codatatypes and corecursive functions have also been considered convenient for use in mathematical formalizations of complex numbers, so our techniques can also be used to further develop such formalizations.

Cohipster builds on and extends the framework of the previous theory exploration system Hipster in Isabelle/HOL. Prior to this work Hipster only had capabilities to discover properties about recursive functions and prove them by induction. As the underlying conjecture finding method relies on testing, we had to extend the testing methods involved to enable evaluating and testing data structures that are potentially infinite in size. To prove the conjectured properties, we developed automated proof methods for coinductive properties. For this we made use of the coinduction tactics built-in to Isabelle/HOL, with added techniques to automatically determine the correct parameters and to

automate the proofs of subgoals.

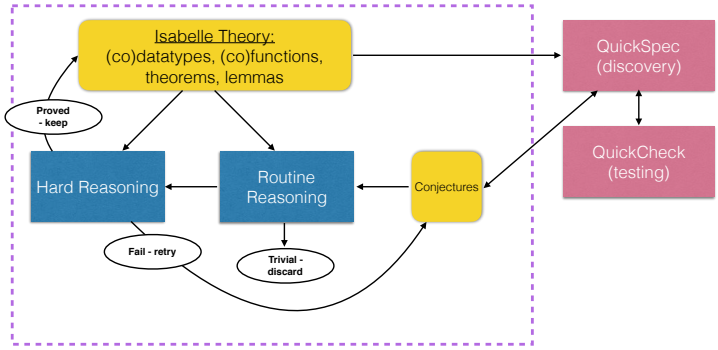

Figure 4.1: The architecture of the (Co)Hipster system.

Cohipster takes a set of functions as an argument and generates conjectures of equational properties. It then attempts to prove the generated conjectures automatically, discarding those that are trivial to prove, presenting the conjectures it finds nontrivial proofs of to the user as lemmas, and any unproved conjectures are left for the user to try to prove or disprove.

For example, when we call on Cohipster for a theory of infinite binary trees, containing a function *mirror* for reflecting a tree and a function *tmap* for mapping a function over the tree elements, Cohipster will discover the lemmas

$$mirror\,(mirror\;t) = t$$

and

$$tmap\;f\;(mirror\;t) = mirror\,(tmap\;f\;x)$$

Cohipster has conjectured these lemmas with the use of QuickSpec, generated formal machine-checked proofs in Isabelle, and determined that the lemmas are not too trivial to be of interest. This process takes just under a minute. The Isabelle user can then import these lemmas into their theory with a mouse-click.

We evaluate Cohipster on several theories about different codatatypes found in the literature: lazy lists, extended natural numbers, streams, and two kinds of infinite trees. Cohipster demonstrates the ability to automatically and efficiently discover and prove useful and interesting properties about these codatatypes. Cohipster is the first theory exploration tool to be capable of handling infinite structures and discovering coinductive properties about them as well as generating coinductive proofs.

4.2 Paper 2: Lemma Discovery and Strategies for Automated Induction

Automating inductive proofs is a long standing challenge for automated theorem provers. One of the reasons for the difficulty is that inductive proofs often require auxiliary lemmas in order to complete the proof. Proof by induction is commonly used in mathematics to reason about natural numbers and in computer science to reason about inductive datatypes. Improved techniques for automated inductive proving are therefore needed to achieve further automation in mathematics and software verification.

We combine the state-of-the-art theory exploration system QuickSpec with the state-of-the-art automated theorem prover (ATP) Vampire to create a hybrid inductive proof system. QuickSpec has previously been used to discover useful lemmas for inductive proofs in the specialized inductive prover HipSpec and in the Isabelle/HOL lemma discovery system Hipster. Vampire is primarily aimed at proving theorems in first-order logic but has been extended to also perform various other tasks, including proofs by induction in recent years. Vampire is a very powerful and efficient prover but has very limited built-in support for lemma conjecturing, so combining it with an external conjecturing tool is a promising prospect. We also employ specialized strategy schedule training to optimize the parameter selection for Vampire, which can have a great effect on whether or not a proof is found.

Starting from a goal theorem intended to be proved using induction, we have QuickSpec conjecture lemmas about the relevant functions and datatypes. These conjectures are added to the problem file given as input to Vampire, using a special keyword to notify Vampire that any lemma that is used in the result proof must itself also be proved. This kind of speculative lemma use is straightforward and efficient in Vampire. The whole process – starting from the original problem, conjecturing lemmas with QuickSpec, adding conjectures to the problem file and attempting to prove the modified problem with Vampire – is automated.

Another important piece of the puzzle when optimizing ATP performance is choosing the right strategy for the problem. A strategy is an assignment of concrete values to the various parameters or options that control how an ATP like Vampire searches for a proof. Vampire has more than 100 options (parameters) to configure the proof search, so there are very many strategies to choose from, and different strategies work well on different problems. A strategy schedule is an arrangement of different proving strategies of complementary characteristics into a sequence with assigned time budgets, to be executed in sequence (or in parallel). A good strategy schedule can greatly improve ATP performance. In this work, we constructed a strategy schedule specifically targeting inductive theorem proving on the TIP benchmarks [19], following the recipe for strategy discovery pioneered by the Spider system [86] and using a novel approach presented in [8].

In this work we compare the number of proofs found by Vampire on the TIP benchmark problems using six different approaches:

1. Our default strategy for Vampire had flags for structural induction active. This was meant to simulate what kinds of results a non-expert user who wanted to use Vampire to prove these inductive problems might achieve.

2. We then added lemmas conjectured by QuickSpec to the problems before sending them to Vampire, to see what benefits we could get by adding externally conjectured lemmas.

3. We gave Vampire a specialized strategy schedule for inductive problems, trained on the TIP problems.

4. We used the strategy schedule from 3. and added lemmas conjectured by QuickSpec to the problems before sending them to Vampire.

5. We gave Vampire a specialized strategy schedule for inductive problems, trained on the TIP problems with conjectures from QuickSpec added.

6. Finally, we used the strategy schedule from 5. and added conjectured lemmas to the input problems.

	Default Strategy	Trained Strategy Schedule	Schedule trained with lemmas
no lemmas	102	236	237
with lemmas	143	263	288
Total proofs found	153	269	289

Table 4.1: Number of proofs found using different strategies.

The results of these experiments can be seen in Table 4.1. We can see that adding conjectured lemmas improves the performance of the default strategy by 40% (from 102 to 143 proofs). Strategy schedule training alone improves the performance greatly, by 130% (from 103 to 236 proofs). Our best method was the strategy schedule trained on problems with lemmas, when given the problems with added lemmas, which found 288 proofs, an improvement of 183% compared to the baseline.

Certain subsets of the TIP benchmark set have been used in previous work to evaluate the inductive proof performance of CVC4 [71] and HipSpec [18]. Our best method, which leveraged conjectured lemmas and strategy schedule training in combination, outperformed both of these previous state-of-the-art approaches.

Shortly after this paper was published, [54] presented the CCLemma inductive prover, previously described in Section 3.1.3. The authors evaluate their tool on a subset of the TIP benchmark set, while also introducing some new benchmark problems meant to provide a greater challenge, and provide a comparison other inductive proof systems including CVC4 and HipSpec. On the common evaluation benchmarks, CCLemma found slightly fewer proofs than HipSpec, and is therefore also outperformed by our best hybrid method.

4.3 Paper 3: Template-based Theory Exploration: Discovering Properties of Functional Programs by Testing

We introduce RoughSpec, a tool for template-based theory exploration. Previous state-of-the-art theory exploration tools do not scale well to larger libraries of concepts, and may produce too much output that is uninteresting to a user. RoughSpec makes theory exploration more tractable and scalable, and solves the problem of determining what conjectures are interesting, by only considering conjectures matching templates defined by the user.

A template is an expression describing a family of properties such as commutativity or distributivity. We represent a template as a Haskell equation containing functions, variables, and *metavariables*. For example the following is a template for commutative properties (in our syntax, variables are written in uppercase, and a metavariable is written as a variable with a leading question mark): `?F X Y = ?F Y X`.

RoughSpec takes in a set of functions along with a set of templates, and generates conjectures about the functions that match one of the given templates. For each given template, RoughSpec instantiates the metavariables in the template with the functions in scope to create type-correct equations which are then tested. For example, if we call RoughSpec using the above template along with addition and subtraction on integers it will come up with the conjecture `x+y = y+x` (and no further output), in less than 0.1 seconds. In the example shown at the beginning of the chapter in Figure 1.2 we used this commutativity template and an additional six templates describing identity properties, distributivity and associativity, shown in Figure 4.2:

```
?F(?X)   = ?X          ?F(?G(X))      = ?G(?F(X))
?F(?Y,X) = X           ?F(?G(X,Y))    = ?G(?F(X),?F(Y))
?F(X,?Y) = X           ?F(?F(X,Y),Z) = ?F(X,?F(Y,Z))
?F(X,Y)  = ?F(Y,X)
```

Figure 4.2: Templates used for the example in Figure 1.2.

RoughSpec builds on the previous theory exploration tool QuickSpec, making use of QuickSpec's highly optimized techniques for term generation, testing, and pruning. Since RoughSpec only generates terms and tests conjectures that match the given templates, it potentially has a much smaller search space than QuickSpec and is well suited to exploring larger libraries of functions and performing fast and targeted searches for properties. We have also combined RoughSpec with QuickSpec, using QuickSpec to perform a complete search for smaller term sizes, while using templates for larger, more complex properties, in order to leverage the strengths of both systems.

We evaluate RoughSpec in case studies on several Haskell libraries. We demonstrate that RoughSpec can be useful in understanding an unfamiliar library, it can be used to target specific kinds of useful properties, and that

it makes theory exploration tractable on larger libraries that were previously infeasible to explore. We find that a hybrid approach combining QuickSpec and RoughSpec often produces higher-quality output than either tool manages on its own.

4.4 Paper 4: LOL: A Library Of Lemma templates for data-driven conjecturing

Previous work on template-based conjecturing has relied on the hypothesis that a small number of templates can be used to generate interesting and useful conjectures in various different contexts, without providing empirical evidence for the hypothesis. This paper describes a dataset of lemmas and their structural *templates* extracted from Isabelle's Archive of Formal Proofs (AFP) [2], and data analysis showing that a small number of templates can be used to generate many different lemmas. This dataset can be used for data-driven template generation, in place of relying on user provided templates or a limited number of built in templates as previous methods have done.

We extract lemma statements and their corresponding templates from the Isabelle/HOL theory files in the AFP. We analyze the frequency of different templates among the assembled lemmas and examine the most commonly occurring templates. The entries in the AFP consist of proof formalizations related to various topics in Computer Science, Logic and Mathematics. Since the lemmas found in the AFP are invented by human experts as part of proofs that those experts found reason to formalize, we believe they make good examples of lemmas that are interesting and useful in a wide variety of contexts.

Our dataset contains 22767 equational lemmas captured by 6567 different templates. They were extracted from 2169 different theory files from 611 AFP entries. The 10 most frequent templates together describe 3057 lemmas or 13.5% of the lemmas in our set, while more than 50% of the lemmas can be described using only 266 of the 6567 templates. This supports our hypothesis that a smaller number of templates is sufficient to discover many interesting and useful lemmas using template-based conjecturing.

4.5 Paper 5: Lemmanaid: Neuro-Symbolic Lemma Conjecturing

We present LEMMANAID, a neuro-symbolic lemma conjecturing tool designed to discover conjectures by analogy. Previous symbolic lemma discovery tools have been restricted in the shapes of lemmas they can discover and the input they can handle. Previous neural lemma discovery methods have also been limited in their application scope. LEMMANAID combines Large Language Models (LLMs) and symbolic methods to conjecture lemmas about any given Isabelle formalization.

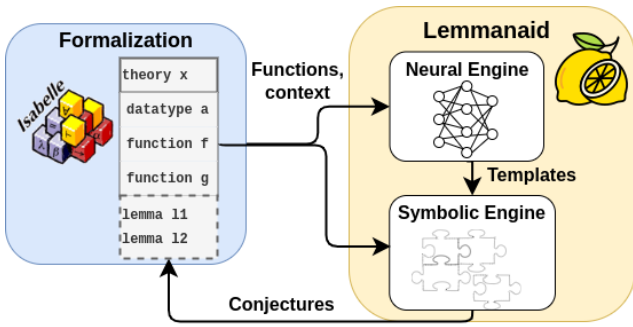

Figure 4.3: High-level overview of LEMMANAID.

LEMMANAID uses a fine tuned LLM to generate *lemma templates* that describe the shape of a lemma, given some function symbols along with their definitions. A symbolic engine implemented in Isabelle then takes the generated template as input, along with the function symbols and definitions. Instantiating the template using the given function symbols, the symbolic engine outputs lemma conjectures.

For evaluation, we measure the percentage of unseen test-set lemmas that can be discovered by LEMMANAID given the relevant functions and definitions. Our training and test data is extracted from Isabelle's HOL library and from formalizations in the Archive of Formal Proofs (AFP) [2]. Using human-written formalizations as a gold-standard gives us a good approximation of how many interesting lemmas LEMMANAID can produce for a novel theory. To compare our neuro-symbolic approach to fully neural conjecturing, we fine-tune the same LLM used in LEMMANAID's neural engine to instead generate complete lemma statements, bypassing the symbolic engine.

LEMMANAID outperforms both purely neural and purely symbolic methods on our test sets, discovering between 34.6-48.2% of the gold standard human written lemmas, which is 8.5-14% more lemmas than the neural-only method. LEMMANAID greatly outperforms the symbolic conjecturing tool QuickSpec in our comparison case study, discovering more than 3 times as many of the relevant gold standard lemmas. By leveraging the best of both symbolic and neural methods we can generate useful lemmas across a wide range of domains, facilitating computer-assisted theory development and formalization.

Chapter 5

Concluding Remarks and Future Directions

In this thesis, we show that theory exploration can be used effectively to conjecture interesting and useful lemmas in a variety of settings. Our techniques can discover valuable lemmas for users of proof assistants, generate program specifications, and help to achieve state-of-the-art results in automated inductive theorem proving. We demonstrate the potential of a template-based approach to limit the conjecturing search space and thereby make theory exploration faster and more tractable. We demonstrate the feasibility of using an LLM for theory exploration, both on its own and as a component of a neuro-symbolic template-based conjecturing system. Lemma conjecturing is a challenging task. Developing useful conjecturing techniques is a crucial aspect of advancing both automated and interactive formal reasoning about programs and proofs.

The research represented in this thesis was completed over a span of years from 2017-2025. During this time we saw the arrival and evolution of LLMs, and the impressive results they achieved in all sorts of domains, including mathematics and theorem proving. These new technological advancements have brought with them a new research frontier in exploring how LLMs can be applied, extended, evolved, and combined with other techniques. It has been a unique experience to work on research in this field in parallel to this development. We believe that there is still much to learn and discover in how to best harness the formidable capabilities of LLMs to develop tools that are powerful and effective, but also safe and efficient.

5.1 Future Directions

We have identified several potential future directions to continue the work in this thesis. Some are research ideas that could be examined in the near future, while others are broader impact goals that could be pursued in the longer term.

5.1.1 Next Steps

In Paper 5 we demonstrate the potential of a neuro-symbolic approach to theory exploration. In the current work we focus solely on lemma conjecturing, and not on proving the generated lemmas or other goal theorems. Combining the LEMMANAID conjecturing tool with automated proof techniques, either symbolic or neural, would make for a more holistic tool and enable evaluation on proof benchmarks.

Recent work has shown the potential of combining LLMs, reinforcement learning and formal proof methods to automate complex reasoning tasks [88]. In [36], a self-learning loop combining a neural machine translation (NMT) model and an SMT solver successfully generates lemmas that enable many challenging inductive problems to be proved. Further exploring the application of reinforcement learning in automated neuro-symbolic lemma conjecturing is a promising direction. Integrating LEMMANAID with proof techniques would also enable the creation of a reinforcement loop that rewards conjectures for being provable or useful in proofs, but a an alternative reward function based on the interestingness measures described in Section 1.1 could also be examined.

Another interesting direction for future work would be extending the LEMMANAID approach to more languages and different proof assistants such as Lean [62] and Rocq [80]. The current work only considers Isabelle formalizations and is fine-tuned on Isabelle syntax. A multi-lingual approach has shown promise in the context of autoformalization [44], and could potentially be useful to create a more robust and flexible conjecturing tool.

In Paper 2 we demonstrate how theory exploration with QuickSpec can be combined with Vampire's automated induction methods to achieve top results on inductive benchmark problems. Developing this approach further and applying it to more advanced and specialized inductive problems is another promising direction. An interesting place to start would be a to combine the approaches from Papers 2 and 3, and apply template-based theory exploration to inductive problem sets, perhaps incorporating the neuro-symbolic approach from Paper 5. Our method in Paper 2 only conjectures equational lemmas, and many inductive proofs require conditional lemmas. Better support for conditional conjecturing is likely to improve outcomes and increase the applicability of our methods.

5.1.2 Further Prospects

Looking to the future, we pose some bigger goals for researchers and engineers to work towards in coming years. We would like to see theory exploration made more usable and useful to users working on both mathematical formalization and software development. We also believe theory exploration techniques could be used to improve automated reasoning methods, which in turn would be beneficial to automated techniques for software verification.

One goal is the development of theory exploration tools that are useful to a mathematician working on developing and formalizing a new theory. This requires smooth integration with a proof assistant and a user-friendly interface. User studies could help to better define what kinds of lemmas and

conjectures would be the most helpful in this context, and further development of conjecturing methods may be needed to fulfill those wishes.

Another goal is to develop tools that integrate theory exploration methods in a software development workflow. In this context, theory exploration can be used to generate software specifications and tests. The ongoing escalation of AI-assisted software development and the use of generative AI to generate code brings with it an increased need for usable methods for specification, testing, and verification, to ensure that the generated code works as intended.

Third, improved automated reasoning methods incorporating theory exploration could be advantageous to further automation in both mathematical proof and software verification. These kinds of automated methods could be integrated with interactive tools as described above, or used in fully automated systems. Stronger and more widely applicable automation in software verification would lower the bar for developing robust and secure software.

Bibliography

[1] Markus Aderhold. "Improvements in Formula Generalization". In: *Automated Deduction – CADE-21*. Ed. by Frank Pfenning. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 231–246. ISBN: 978-3-540-73595-3 (cit. on p. 19).

[2] *Archive of Formal Proofs*. https://www.isa-afp.org/index.html (cit. on pp. 13, 35, 36).

[3] Zhangir Azerbayev et al. "Llemma: An Open Language Model for Mathematics". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=4WnqRR915j (cit. on p. 11).

[4] Rajiv Bagai et al. "Automatic Theorem Generation in Plane Geometry". In: *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems*. ISMIS '93. Berlin, Heidelberg: Springer-Verlag, 1993, pp. 415–424. ISBN: 3540568042 (cit. on p. 18).

[5] Grzegorz Bancerek et al. "The Role of the Mizar Mathematical Library for Interactive Proof Development in Mizar". In: *J. Autom. Reason.* 61.1-4 (2018), pp. 9–32. DOI: 10.1007/s10817-017-9440-6. URL: https://doi.org/10.1007/s10817-017-9440-6 (cit. on p. 25).

[6] Haniel Barbosa et al. "cvc5: A Versatile and Industrial-Strength SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*. Ed. by Dana Fisman and Grigore Rosu. Vol. 13243. Lecture Notes in Computer Science. Springer, 2022, pp. 415–442. DOI: 10.1007/978-3-030-99524-9_24. URL: https://doi.org/10.1007/978-3-030-99524-9%5C_24 (cit. on p. 22).

[7] Clark W. Barrett et al. "CVC4". In: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*. Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 171–177. DOI: 10.1007/978-3-642-22110-1_14. URL: https://doi.org/10.1007/978-3-642-22110-1%5C_14 (cit. on p. 22).

[8] Filip Bártek, Karel Chvalovský and Martin Suda. "Regularization in Spider-Style Strategy Discovery and Schedule Construction". In: *Automated Reasoning*. Ed. by Christoph Benzmüller, Marijn J.H. Heule and Renate A. Schmidt. Cham: Springer Nature Switzerland, 2024, pp. 194–213. ISBN: 978-3-031-63498-7 (cit. on p. 31).

[9] Robert S. Boyer and J.S. Moore. *A Computational Logic*. ACM Monographs in Computer Science, 1979 (cit. on p. 19).

[10] Rudy Braquehais and Colin Runciman. "Speculate: discovering conditional equations and inequalities about black-box functions by reasoning from test results". In: *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*. 2017, pp. 40–51 (cit. on p. 22).

[11] Bruno Buchberger. "Theory exploration with Theorema". In: *Analele Universitatii Din Timisoara, ser. Matematica-Informatica* 38.2 (2000), pp. 9–32 (cit. on p. 20).

[12] Bruno Buchberger et al. "Theorema: Towards computer-aided mathematical theory exploration". In: *Journal of Applied Logic* 4 (Dec. 2006), pp. 470–504. DOI: 10.1016/j.jal.2005.10.006 (cit. on p. 20).

[13] Alan Bundy. "The use of explicit plans to guide inductive proofs". In: *9th International Conference on Automated Deduction*. Ed. by Ewing Lusk and Ross Overbeek. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 111–120. ISBN: 978-3-540-39216-3 (cit. on p. 19).

[14] Alan Bundy et al. *Rippling: meta-level guidance for mathematical reasoning*. USA: Cambridge University Press, 2005. ISBN: 052183449X (cit. on p. 19).

[15] Alan Bundy et al. "The OYSTER-CLAM system". In: *10th International Conference on Automated Deduction*. Ed. by Mark E. Stickel. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 647–648. ISBN: 978-3-540-47171-4 (cit. on p. 19).

[16] Koen Claessen and John Hughes. "QuickCheck: a lightweight tool for random testing of Haskell programs". In: *Proceedings of ICFP*. 2000, pp. 268–279 (cit. on p. 21).

[17] Koen Claessen, Nicholas Smallbone and John Hughes. "QuickSpec: guessing formal specifications using testing". In: *Proceedings of the 4th International Conference on Tests and Proofs*. TAP'10. Málaga, Spain: Springer-Verlag, 2010, pp. 6–21. ISBN: 3642139760 (cit. on pp. 21, 22).

[18] Koen Claessen et al. "Automating Inductive Proofs using Theory Exploration". In: *Proceedings of CADE*. Vol. 7898. LNCS. Springer, 2013, pp. 392–406 (cit. on pp. 21, 32).

[19] Koen Claessen et al. "TIP: Tons of Inductive Problems". In: *Proceedings of CICM*. Springer, 2015, pp. 333–337 (cit. on p. 31).

[20] Simon Colton. "Refactorable Numbers - A Machine Invention". In: *Journal of Integer Sequences* 2 (1999). ISSN: 1530-7638. URL: https://cs.uwaterloo.ca/journals/JIS/colton/joisol.html (cit. on p. 18).

[21] Simon Colton. "The HR Program for Theorem Generation". In: *Automated Deduction—CADE-18*. Ed. by Andrei Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 285–289. ISBN: 978-3-540-45620-9 (cit. on p. 18).

[22] Simon Colton, Alan Bundy and Toby Walsh. "On the notion of interestingness in automated mathematical discovery". In: *Int. J. Hum.-Comput. Stud.* 53.3 (Sept. 2000), pp. 351–375. ISSN: 1071-5819. DOI: 10.1006/ijhc.2000.0394. URL: https://doi.org/10.1006/ijhc.2000.0394 (cit. on p. 3).

[23] Simon Cruanes. "Superposition with Structural Induction". In: *Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings*. Ed. by Clare Dixon and Marcelo Finger. Vol. 10483. Lecture Notes in Computer Science. Springer, 2017, pp. 172–188 (cit. on p. 19).

[24] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. arXiv: 2501.12948 [cs.CL]. URL: https://arxiv.org/abs/2501.12948 (cit. on p. 11).

[25] DeepSeek-AI et al. *DeepSeek-V3 Technical Report*. 2025. arXiv: 2412.19437 [cs.CL]. URL: https://arxiv.org/abs/2412.19437 (cit. on p. 12).

[26] Lucas Dixon and Jacques D. Fleuriot. "IsaPlanner: A Prototype Proof Planner in Isabelle". In: *Automated Deduction – CADE-19*. Vol. 2741. Springer Berlin Heidelberg, July 2003, pp. 279–283. DOI: 10.1007/978-3-540-45085-6_22 (cit. on pp. 19, 20, 22).

[27] Lucas Dixon and Moa Johansson. *IsaPlanner 2: A Proof Planner for Isabelle*. 2007 (cit. on pp. 19, 20, 22).

[28] Kefan Dong and Tengyu Ma. "STP: Self-play LLM Theorem Provers with Iterative Conjecturing and Proving". In: *Forty-second International Conference on Machine Learning*. 2025. URL: https://openreview.net/forum?id=zWArMedNuW (cit. on pp. 14, 26, 27).

[29] Christoph Drösser. *AI Will Become Mathematicians' 'Co-Pilot'*. 2024. URL: https://www.scientificamerican.com/article/ai-will-become-mathematicians-co-pilot/. (accessed: 2025-10-13) (cit. on p. 5).

[30] Susan L. Epstein. "Learning and discovery: one system's search for mathematical knowledge". In: *Computational Intelligence* 4.1 (1988), pp. 42–53. DOI: https://doi.org/10.1111/j.1467-8640.1988.tb00089.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8640.1988.tb00089.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8640.1988.tb00089.x (cit. on p. 18).

[31] S. Fajtlowicz. "On Conjectures of Graffiti". In: *Discrete Math.* 72.1–3 (Dec. 1988), pp. 113–118. ISSN: 0012-365X. DOI: 10.1016/0012-365X(88)90199-9. URL: https://doi.org/10.1016/0012-365X(88)90199-9 (cit. on p. 20).

[32] Alhussein Fawzi et al. “Learning dynamic polynomial proofs”. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019 (cit. on p. 26).

[33] Vlad Firoiu et al. *Training a First-Order Theorem Prover from Synthetic Data*. 2021. arXiv: 2103.03798 [cs.AI]. URL: https://arxiv.org/abs/2103.03798 (cit. on p. 26).

[34] Emily First et al. “Baldur: Whole-Proof Generation and Repair with Large Language Models”. In: *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2023. San Francisco, CA, USA: Association for Computing Machinery, Nov. 2023, pp. 1229–1241. DOI: 10.1145/3611643.3616243. URL: https://doi.org/10.1145/3611643.3616243 (cit. on p. 14).

[35] 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*. 2016, pp. 219–228. URL: http://ceur-ws.org/Vol-1785/W23.pdf (cit. on p. 24).

[36] Thibault Gauthier and Josef Urban. “Learning Conjecturing from Scratch”. In: *Automated Deduction - CADE-30*. 2025 (cit. on pp. 14, 27, 38).

[37] Thibault Gauthier et al. “A Mathematical Benchmark for Inductive Theorem Provers”. In: *LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023*. Ed. by Ruzica Piskac and Andrei Voronkov. Vol. 94. EPiC Series in Computing. EasyChair, 2023, pp. 224–237 (cit. on p. 27).

[38] Márton Hajdu, Laura Kovács and Michael Rawson. “Rewriting and Inductive Reasoning”. In: *Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning*. Ed. by Nikolaj Bjørner, Marijn Heule and Andrei Voronkov. Vol. 100. EPiC Series in Computing. EasyChair, 2024, pp. 278–294. DOI: 10.29007/vbfp. URL: /publications/paper/T1mjX (cit. on p. 19).

[39] Márton Hajdu et al. “Getting Saturated with Induction”. In: *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*. Ed. by Jean-François Raskin et al. Cham: Springer Nature Switzerland, 2022, pp. 306–322. ISBN: 978-3-031-22337-2. DOI: 10.1007/978-3-031-22337-2_15. URL: https://doi.org/10.1007/978-3-031-22337-2_15 (cit. on pp. 13, 19).

[40] Jonathan Heras et al. “Proof-Pattern Recognition and Lemma Discovery in ACL2”. In: *Proceedings of LPAR*. 2013. DOI: 10.1007/978-3-642-45221-5_27 (cit. on p. 24).

[41] Jónathan Heras and Ekaterina Komendantskaya. "ACL2(ml): Machine-Learning for ACL2". In: *Electronic Proceedings in Theoretical Computer Science* 152 (June 2014), pp. 61–75. ISSN: 2075-2180. DOI: 10.4204/eptcs.152.5. URL: http://dx.doi.org/10.4204/EPTCS.152.5 (cit. on p. 24).

[42] Birgit Hummel. "An investigation of formula generalization heuristics for induction proofs". In: *Interner Bericht. Fakultät für Informatik, Universität Karlsruhe* 86-87 (1987) (cit. on p. 19).

[43] Andrew Ireland and Alan Bundy. "Productive Use of Failure in Inductive Proof". In: *Journal of Automated Reasoning* 16 (1996), pp. 79–111 (cit. on p. 19).

[44] Albert Q. Jiang, Wenda Li and Mateja Jamnik. "Multi-language diversity benefits autoformalization". In: *Proceedings of the 38th International Conference on Neural Information Processing Systems*. NIPS '24. Vancouver, BC, Canada: Curran Associates Inc., 2025. ISBN: 9798331314385 (cit. on pp. 14, 38).

[45] Albert Qiaochu Jiang et al. "Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs". In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=SMa9EAovKMC (cit. on p. 14).

[46] Juyong Jiang et al. "A Survey on Large Language Models for Code Generation". In: *ACM Trans. Softw. Eng. Methodol.* (July 2025). Just Accepted. ISSN: 1049-331X. DOI: 10.1145/3747588. URL: https://doi.org/10.1145/3747588 (cit. on p. 12).

[47] Moa Johansson. "Automated theory exploration for interactive theorem proving: An introduction to the Hipster system". In: *Proceedings of ITP*. Vol. 10499. LNCS. Springer, 2017, pp. 1–11. ISBN: 978-331966106-3 (cit. on p. 20).

[48] Moa Johansson. "Lemma Discovery for Induction". In: *Intelligent Computer Mathematics*. Ed. by Cezary Kaliszyk et al. Cham: Springer International Publishing, 2019, pp. 125–139. ISBN: 978-3-030-23250-4 (cit. on p. 17).

[49] Moa Johansson, Lucas Dixon and Alan Bundy. "Conjecture Synthesis for Inductive Theories". In: *Journal of Automated Reasoning* 47.3 (Oct. 2011), pp. 251–289. ISSN: 1573-0670. DOI: 10.1007/s10817-010-9193-y. URL: https://doi.org/10.1007/s10817-010-9193-y (cit. on p. 21).

[50] Moa Johansson, Lucas Dixon and Alan Bundy. "Dynamic Rippling, Middle-Out Reasoning and Lemma Discovery". In: *Verification, Induction, Termination Analysis: Festschrift for Christoph Walther on the Occasion of His 60th Birthday*. Ed. by Simon Siegler and Nathan Wasser. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 102–116. ISBN: 978-3-642-17172-7. DOI: 10.1007/978-3-642-17172-7_6. URL: https://doi.org/10.1007/978-3-642-17172-7_6 (cit. on p. 19).

[51] Moa Johansson et al. “Hipster: Integrating Theory Exploration in a Proof Assistant”. In: *Proceedings of CICM*. Springer, 2014, pp. 108–122 (cit. on p. 20).

[52] Matt Kaufmann, J. Strother Moore and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach*. USA: Kluwer Academic Publishers, 2000. ISBN: 0792377443 (cit. on pp. 19, 24).

[53] Laura Kovács and Andrei Voronkov. “First-Order Theorem Proving and Vampire”. In: *Computer Aided Verification*. Ed. by Natasha Sharygina and Helmut Veith. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–35. ISBN: 978-3-642-39799-8 (cit. on p. 13).

[54] Cole Kurashige et al. “CCLemma: E-Graph Guided Lemma Discovery for Inductive Equational Proofs”. In: *Proc. ACM Program. Lang.* 8.ICFP (Aug. 2024). DOI: 10.1145/3674653. URL: https://doi.org/10.1145/3674653 (cit. on pp. 22, 32).

[55] Douglas B. Lenat. “AM, an artificial intelligence approach to discovery in mathematics as heuristic search”. PhD thesis. Stanford University, 1976 (cit. on p. 18).

[56] Aitor Lewkowycz et al. *Solving Quantitative Reasoning Problems with Language Models*. 2022. arXiv: 2206.14858 [cs.CL]. URL: https://arxiv.org/abs/2206.14858 (cit. on p. 11).

[57] Zhaoyu Li et al. “A Survey on Deep Learning for Theorem Proving”. In: *First Conference on Language Modeling*. 2024. URL: https://openreview.net/forum?id=zlw6AHwukB (cit. on p. 12).

[58] Thang Luong and Edward Lockhart. *Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad*. https://deepmind.google/discover/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/. Last accessed 12 October 2025. 2025 (cit. on p. 14).

[59] Rudy Matela Braquehais. “Tools for Discovery, Refinement and Generalization of Functional Properties by Enumerative Testing”. PhD thesis. University of York, Oct. 2017. URL: https://etheses.whiterose.ac.uk/id/eprint/19178/ (cit. on p. 22).

[60] R. L. McCasland, A. Bundy and P. F. Smith. “MATHsAiD: Automated mathematical theory exploration”. In: *Applied Intelligence* (June 2017). ISSN: 1573-7497. DOI: 10.1007/s10489-017-0954-8. URL: https://doi.org/10.1007/s10489-017-0954-8 (cit. on p. 18).

[61] Omar Montano-Rivas et al. “Scheme-based theorem discovery and concept invention”. In: *Expert systems with applications* 39.2 (2012), pp. 1637–1646 (cit. on p. 20).

[62] Leonardo de Moura et al. “The Lean Theorem Prover (System Description)”. In: *Automated Deduction - CADE-25*. Ed. by Amy P. Felty and Aart Middeldorp. Cham: Springer International Publishing, 2015, pp. 378–388. ISBN: 978-3-319-21401-6 (cit. on p. 38).

[63] Yutaka Nagashima and Julian Parsert. "Goal-Oriented Conjecturing for Isabelle/HOL". In: *Intelligent Computer Mathematics*. Ed. by Florian Rabe et al. Cham: Springer International Publishing, 2018, pp. 225–231. ISBN: 978-3-319-96812-4 (cit. on p. 19).

[64] Yutaka Nagashima et al. "Template-Based Conjecturing for Automated Induction in Isabelle/HOL". In: *Fundamentals of Software Engineering*. Ed. by Hossein Hojjat and Erika Ábrahám. Cham: Springer Nature Switzerland, 2023, pp. 112–125. ISBN: 978-3-031-42441-0 (cit. on p. 20).

[65] A. Newell and H. Simon. "The logic theory machine–A complex information processing system". In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 61–79. DOI: 10.1109/TIT.1956.1056797 (cit. on p. 12).

[66] Tobias Nipkow, Markus Wenzel and Lawrence C. Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Berlin, Heidelberg: Springer-Verlag, 2002. ISBN: 3540433767 (cit. on p. 13).

[67] OpenAI et al. *OpenAI o1 System Card*. 2024. arXiv: 2412.16720 [cs.AI]. URL: https://arxiv.org/abs/2412.16720 (cit. on p. 11).

[68] Gabriel Poesia et al. "Learning Formal Mathematics From Intrinsic Motivation". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=uNKlTQ8mBD (cit. on pp. 14, 26, 27).

[69] Markus Norman Rabe et al. "Mathematical Reasoning via Self-supervised Skip-tree Training". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=YmqAnY0CMEy (cit. on p. 25).

[70] Z. Z. Ren et al. *DeepSeek-Prover-V2: Advancing Formal Mathematical Reasoning via Reinforcement Learning for Subgoal Decomposition*. 2025. arXiv: 2504.21801 [cs.CL]. URL: https://arxiv.org/abs/2504.21801 (cit. on p. 14).

[71] Andrew Reynolds and Viktor Kuncak. "Induction for SMT Solvers". In: *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*. Ed. by Deepak D'Souza, Akash Lal and Kim Guldstrand Larsen. Vol. 8931. Lecture Notes in Computer Science. Springer, 2015, pp. 80–98 (cit. on pp. 22, 32).

[72] Jakob Schulz. "Schönhage-Strassen Multiplication". In: *Archive of Formal Proofs* (May 2024). https://isa-afp.org/entries/Schoenhage_Strassen.html, Formal proof development. ISSN: 2150-914x (cit. on p. 6).

[73] Herbert A. Simon and Allen Newell. "Heuristic Problem Solving: The Next Advance in Operations Research". In: *Operations Research* 6.1 (1958), pp. 1–10. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/167397 (cit. on p. 13).

[74] Eytan Singher and Shachar Itzhaky. "Theory Exploration Powered by Deductive Synthesis". In: *Computer Aided Verification*. Ed. by Alexandra Silva and K. Rustan M. Leino. Cham: Springer International Publishing, 2021, pp. 125–148. ISBN: 978-3-030-81688-9 (cit. on p. 22).

[75] Aishwarya Sivaraman et al. "Data-driven lemma synthesis for interactive proofs". In: *Proc. ACM Program. Lang.* 6.OOPSLA2 (Oct. 2022). DOI: 10.1145/3563306. URL: https://doi.org/10.1145/3563306 (cit. on p. 22).

[76] Nicholas Smallbone. "Twee: An Equational Theorem Prover." In: *CADE*. 2021, pp. 602–613 (cit. on p. 21).

[77] Nicholas Smallbone et al. "Quick specifications for the busy programmer". In: *Journal of Functional Programming* 27 (2017). DOI: 10.1017/S0956796817000090 (cit. on pp. 4, 20, 21).

[78] William Sonnex, Sophia Drossopoulou and Susan Eisenbach. "Zeno: An Automated Prover for Properties of Recursive Data Structures". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Cormac Flanagan and Barbara König. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 407–421. ISBN: 978-3-642-28756-5 (cit. on p. 19).

[79] Christian Szegedy. "A Promising Path Towards Autoformalization and General Artificial Intelligence". In: *Intelligent Computer Mathematics*. Ed. by Christoph Benzmüller and Bruce Miller. Cham: Springer International Publishing, 2020, pp. 3–20. ISBN: 978-3-030-53518-6 (cit. on p. 14).

[80] The Coq Development Team. *The Coq Proof Assistant*. Version 8.20. Sept. 2024. DOI: 10.5281/zenodo.14542673. URL: https://doi.org/10.5281/zenodo.14542673 (cit. on p. 38).

[81] Vampire Team. *Vampire*. http://vprover.github.io (cit. on p. 13).

[82] *The CADE ATP System Competition*. https://tptp.org/CASC/30/ (cit. on p. 13).

[83] Trieu H. Trinh et al. "Solving olympiad geometry without human demonstrations". In: *Nature* 625 (2024), pp. 476–482. URL: https://doi.org/10.1038/s41586-023-06747-5 (cit. on p. 26).

[84] 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 (cit. on pp. 25, 27).

[85] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on p. 11).

[86] Andrei Voronkov. "Spider: Learning in the Sea of Options". In: *Vampire23: The 7th Vampire Workshop*. To appear. 2023. URL: https://easychair.org/smart-program/Vampire23/2023-07-05.html%5C#talk:223833 (cit. on p. 31).

[87] Daniel Wand. "Superposition: types and induction". PhD thesis. Saarland University, 2017 (cit. on p. 19).

[88] Haiming Wang et al. *Kimina-Prover Preview: Towards Large Formal Reasoning Models with Reinforcement Learning*. 2025. arXiv: 2504.11354 [cs.AI]. URL: https://arxiv.org/abs/2504.11354 (cit. on p. 38).

[89] Haiming Wang et al. "LEGO-Prover: Neural Theorem Proving with Growing Libraries". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=3f5PALef5B (cit. on pp. 14, 25, 27).

[90] Mingzhe Wang and Jia Deng. "Learning to prove theorems by learning to generate theorems". In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546 (cit. on p. 26).

[91] Peng-Yuan Wang et al. *A Survey on Large Language Models for Mathematical Reasoning*. 2025. arXiv: 2506.08446 [cs.AI]. URL: https://arxiv.org/abs/2506.08446 (cit. on p. 12).

[92] Qingxiang Wang, Cezary Kaliszyk and Josef Urban. "First Experiments with Neural Translation of Informal to Formal Mathematics". In: *Intelligent Computer Mathematics*. Ed. by Florian Rabe et al. Cham: Springer International Publishing, 2018, pp. 255–270. ISBN: 978-3-319-96812-4 (cit. on p. 14).

[93] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS '22. New Orleans, LA, USA: Curran Associates Inc., 2022. ISBN: 9781713871088 (cit. on p. 11).

[94] Alfred North Whitehead and Bertrand Arthur William Russell. *Principia mathematica; 2nd ed.* Cambridge: Cambridge Univ. Press, 1927. URL: https://cds.cern.ch/record/268025 (cit. on p. 12).

[95] Yuhuai Wu et al. "Autoformalization with Large Language Models". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. 2022. URL: https://openreview.net/forum?id=IUikebJ1Bf0 (cit. on p. 14).

[96] Kaiyu Yang et al. "Position: Formal Mathematical Reasoning—A New Frontier in AI". In: *Forty-second International Conference on Machine Learning Position Paper Track*. 2025. URL: https://openreview.net/forum?id=HuvAM5x2xG (cit. on p. 27).

[97] Zsolt Zombori et al. "Towards Finding Longer Proofs". In: *Automated Reasoning with Analytic Tableaux and Related Methods: 30th International Conference, TABLEAUX 2021, Birmingham, UK, September 6–9, 2021, Proceedings*. Birmingham, United Kingdom: Springer-Verlag, 2021, pp. 167–186. ISBN: 978-3-030-86058-5. DOI: 10.1007/978-3-030-86059-2_10. URL: https://doi.org/10.1007/978-3-030-86059-2_10 (cit. on p. 26).