

JCST Papers

Only for Academic and Non-Commercial Use

Thanks for Reading!



[Survey](#)

[Computer Architecture and Systems](#)

[Artificial Intelligence and Pattern Recognition](#)

[Computer Graphics and Multimedia](#)

[Data Management and Data Mining](#)

[Software Systems](#)

[Computer Networks and Distributed Computing](#)

[Theory and Algorithms](#)

[Emerging Areas](#)



JCST WeChat

Subscription Account

JCST URL: <https://jcst.ict.ac.cn>

SPRINGER URL: <https://www.springer.com/journal/11390>

E-mail: jcst@ict.ac.cn

Online Submission: <https://mc03.manuscriptcentral.com/jcst>

Twitter: JCST_Journal

LinkedIn: Journal of Computer Science and Technology

Automated Conjecturing and Theorem Finding: A Survey

Jian Zhang (张 健), *Fellow, CCF, Senior Member, ACM, IEEE*, and Si-Cheng Tan (谭思成)

*Key Laboratory of System Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
University of Chinese Academy of Sciences, Beijing 100190, China*

E-mail: zj@ios.ac.cn; tansc@ios.ac.cn

Received October 16, 2025; accepted January 7, 2026.

Abstract Automated conjecturing and theorem finding represents a fundamental challenge in artificial intelligence, involving the systematic generation of mathematical conjectures and theorems from given axioms and definitions. This survey provides an overview of automated conjecturing and theorem finding methods. We categorize existing approaches into symbolic methods and data-driven methods. Symbolic methods, which include top-down lemma discovery and bottom-up theory exploration, leverage deductive reasoning and symbolic enumeration. Data-driven methods comprise statistical analogy learning and recent neural methods based on large language models, which learn patterns from large formal corpora, and also include efforts in scientific discovery from experimental data and theorem generation for synthetic data. We also discuss strategies and evaluation methods of interestingness. Finally, we identify current challenges and suggest some future research directions. This comprehensive analysis aims to guide researchers in understanding the current state of automated conjecturing and identifying opportunities for advancing this critical area of mechanized mathematics and automated reasoning.

Keywords automated conjecturing, theorem finding, automated theorem proving, neural-symbolic method, theory exploration

1 Introduction

Automated reasoning is an important area in artificial intelligence (AI). A major research problem in this area is Automated Theorem Proving (ATP). Given a set of axioms and assumptions, an automated theorem prover tries to find a formal proof which shows that a given conjecture/theorem can be deduced logically from the axioms and assumptions. In recent years, automated reasoning has experienced a significant resurgence, largely driven by advances in large language models (LLMs) and their emergent reasoning capabilities. This has once again made it a frontier of AI research.

Before proceeding, it is important to formally distinguish between two fundamental concepts: theorems and conjectures. A theorem is a mathematical statement that has been rigorously proven to be true, based on a set of axioms and previously established

theorems. In contrast, a conjecture is a mathematical statement that is proposed as potentially true but has not yet been formally proven.

A conjecture or theorem is the target of the theorem prover. Where does it come from? Usually, it is given by an experienced researcher (like a mathematician). But can we find interesting conjectures or theorems automatically? We call this problem automated conjecturing or theorem finding^[1]. The two problems are different because a conjecture may not necessarily be a theorem. However, if we have a set of conjectures, we can try to prove them using automated theorem provers. For convenience, we sometimes consider both problems as automated conjecturing.

There are many methods for automated conjecturing and theorem finding. Some are general-purpose, and some are restricted to certain domains. Some approaches are symbolic, based on logic systems; while others use machine learning techniques (like neural

networks) to learn numerical or semantic information from data. To provide a systematic organization, we categorize existing methods into two fundamental paradigms: symbolic methods and data-driven methods. Symbolic methods operate on formal logical systems, generating conjectures through symbolic enumeration or deductive reasoning from axioms and definitions, without requiring training data. In contrast, data-driven methods leverage characteristics learned from data (numerical experiments, formal proof corpus) using statistical analogy learning or neural networks. This survey gives an overview of research work based on this taxonomy.

The paper is organized as follows. Section 2 introduces background knowledge. Section 3 reviews symbolic methods for automated conjecturing, which are categorized into top-down lemma discovery and bottom-up theory exploration. Section 4 reviews data-driven approaches, as well as methods for synthetic data to address the issue of data scarcity. Section 5 presents the strategies for selecting interesting conjectures and the methods for evaluation. Section 6 identifies outstanding challenges and suggests future research directions.

2 Background

2.1 Automated Reasoning

Many general-purpose automated reasoning methods and tools are based on some kind of logical system, including the propositional logic, the first-order (predicate) logic, and the higher-order logics. The propositional logic is the simplest. Many researchers have been working on the satisfiability (SAT) problem of propositional formulas. SAT solvers can be used to prove propositional theorems. It is also interesting to note that, Hao Wang, a pioneer in ATP, developed a program (called Program II^[2]) which was able to make conjectures expressed in the propositional logic.

The book by Wos and Pieper^[3] gives an excellent introduction to automated reasoning in the first-order logic. It shows—through examples and puzzles—how to prove theorems mechanically on the computer, using the theorem prover OTTER^[4]. Typically, an automated theorem prover starts with a set of assumptions or axioms A , and tries to prove a conjecture C , using all kinds of inference rules. If we are successful, we can establish $A \implies C$. If A and C are expressed as formulas in the first-order logic, an effective ap-

proach is to negate the conjecture and try to show that $A \wedge \neg C$ can lead to inconsistency ($\perp$). For simplicity, in this paper, we assume that the set A itself is consistent. Otherwise, we may need more sophisticated techniques that can be used to perform reasoning under inconsistency.

For formulas (clauses) in the first-order logic, the inference rules include binary resolution, hyperresolution, modus ponens, paramodulation, and so on. Each rule takes one or more formulas as input and generates some formulas as consequences. Fig.1 shows a proof tree for a typical theorem proving task. Here $a_1, a_2, \dots$ denote axioms/assumptions, and C is the conjecture.

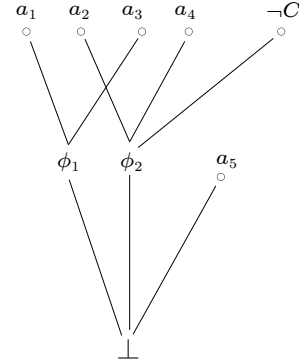


Fig.1. Proof tree.

Currently, for the first-order logic, there are many efficient automated theorem provers which can generate such proof trees automatically. Representative first-order provers include OTTER^[4], Vampire^[5], and E^[6].

A model of a formula (or a set of formulas) is an interpretation that makes the formula(s) true. For example, suppose f is a binary function symbol, and we have the formula $\forall x \forall y (f(x, y) = f(y, x))$. The following Fig.2 shows a 3-element model of the formula, because the model is symmetric.

f	0	1	2
0	0	2	1
1	2	1	0
2	1	0	2

Fig.2. Example 3-element model of formula $f(x, y) = f(y, x)$.

Informally, a model is a concrete instance of some logical formula. A theorem is true in all the models of the axioms/assumptions. From the ATP perspective, sometimes the conjecture C does not hold. In other

words, it is not a logical consequence of A . Then a model of $A \wedge \neg C$ would be a counterexample for the conjecture C . For satisfiability modulo theories (SMT), prominent solvers include Z3^[7] and CVC5^[8], which combine decision procedures for multiple theories. SMT solvers can serve dual roles: they can prove theorems by showing that a formula is unsatisfiable, or they can find models (counterexamples) when a formula is satisfiable.

For higher-order logics, mechanical theorem proving becomes much more difficult. Although some automated theorem provers implement support for higher-order logic, it is not easy to come up with a non-trivial proof fully automatically. Usually, the user works with the theorem prover interactively to obtain a machine-verified proof. This is called Interactive Theorem Proving, also known as proof assistants. Interactive theorem provers are developed to support the formal verification of complex mathematical theorems and software systems. Representative systems include Isabelle/HOL^[9, 10], Coq^[11], Lean^[12], HOL Light^[13], ACL2^[14], and Mizar^[15]. These tools have been successfully applied to verify the correctness of operating system kernels^[16], compilers^[17], and to prove challenging mathematical results such as the Four Color Theorem^[18, 19] and the Kepler Conjecture^[20]. Modern developments in proof assistants include large-scale formal mathematical libraries such as Lean's mathlib^[21], Isabelle's AFP (Archive of Formal Proofs)^①, and integration with automated reasoning tools (e.g., Sledgehammer^[22] for Isabelle). To prove a theorem with proof assistants, mathematicians may break down this goal into several sub-goals or provide some good lemmas to the tools to construct complex proofs more easily.

2.2 Inductive Reasoning and Proof Techniques

Inductive reasoning is essential in automated theorem proving for recursive data structures and properties over theories like natural numbers and trees. Mathematical induction involves proving a base case and an inductive step case. However, automated inductive proofs often fail due to the need for auxiliary lemmas or appropriate generalization^[23, 24].

2.2.1 Rippling Technique

Rippling^[25, 26] guides the rewriting process in in-

ductive proofs by manipulating the inductive conclusion to match the inductive hypothesis. It introduces special annotations to distinguish the common structure (skeleton) from the differences (wave-fronts) between the goal and hypothesis. Key concepts include the following.

- 1) *Skeleton*: the common structure shared between the inductive conclusion and the hypothesis;
- 2) *Wave-fronts*: differences introduced by the inductive step (typically constructors);
- 3) *Wave-rules*: rewrite rules that move wave-fronts outward or into arguments while preserving the skeleton.

For example, when proving list append associativity $(xs ++ ys) ++ zs = xs ++ (ys ++ zs)$, the inductive step for $x :: xs$ yields:

$$((x :: xs) ++ ys) ++ zs = (x :: xs) ++ (ys ++ zs).$$

The wave-fronts $(x ::)$ differ from the hypothesis, while the skeleton involves xs , ys , and zs . Applying append's definition as wave-rules ripples the wave-fronts outward:

$$\begin{aligned} ((x :: xs) ++ ys) ++ zs &\rightarrow x :: ((xs ++ ys) ++ zs), \\ (x :: xs) ++ (ys ++ zs) &\rightarrow x :: (xs ++ (ys ++ zs)). \end{aligned}$$

Now the inductive hypothesis directly applies to complete the proof. When all wave-fronts have been moved to the top level and the skeleton matches the inductive hypothesis, the term is said to be fully-rippled. By tracking wave-fronts and applying wave-rules systematically, rippling significantly reduces the search space in automated inductive proving. However, rippling may encounter blocking terms that prevent wave-fronts from being moved further, requiring additional techniques such as lemma discovery to proceed.

2.2.2 Generalization

Generalization^[25, 27] strengthens conjectures by replacing specific terms with fresh variables. Sometimes a conjecture is too specific to be proved by induction, and it needs to be generalized first.

Let us consider proving $length(xs ++ xs) = 2 \times length(xs)$. Direct induction fails because the inductive hypothesis about $length(xs ++ xs)$ does not match the goal $length((x :: xs) ++ (x :: xs)) = 2 \times length(x :: xs)$.

①<https://www.isa-afp.org/>, Jan. 2026.

The solution is to generalize by replacing the second xs with a fresh variable ys :

$$\text{length}(xs ++ ys) = \text{length}(xs) + \text{length}(ys).$$

This stronger lemma is provable by induction. The proof uses the following rules:

$$\begin{aligned} (x :: xs) ++ ys &= x :: (xs ++ ys), \\ \text{length}(x :: xs) &= 1 + \text{length}(xs). \end{aligned}$$

The inductive proof proceeds as follows:

$$\begin{aligned} \text{length}((x :: xs) ++ ys) &= \text{length}(x :: (xs ++ ys)) \\ &= 1 + \text{length}(xs ++ ys) \\ &= 1 + \text{length}(xs) + \text{length}(ys) \\ &= \text{length}(x :: xs) + \text{length}(ys). \end{aligned}$$

The original conjecture follows by instantiating ys with xs . Generalization strengthens the inductive hypothesis, making it applicable in more contexts—crucial for accumulator functions, multi-argument recursion, and properties with specific constants.

2.3 Large Language Models

Building upon the Transformer architecture^[28], pre-trained language models have achieved unprecedented success in natural language understanding and generation. GPT (Generative Pre-trained Transformer) series^[29–31] demonstrate the power of auto-regressive language modeling at scale, with few-shot learning capabilities across diverse tasks without task-specific fine-tuning. Recent LLMs have grown to hundreds of billions of parameters, trained on massive text corpora. These models exhibit emergent abilities, such as in-context learning and chain-of-thought reasoning^[32, 33]. Notable examples include PaLM^[34], LLaMA^[35], Claude^②, Gemini^[36], and DeepSeek^[37], each pushing the boundaries of language understanding and generation.

3 Symbolic Approaches

In this section, we focus on conjecturing based on symbolic methods in theorem proving. Based on the starting point and direction of generation, symbolic methods for generating conjectures can be divided into two categories^[24]: top-down lemma discovery and bottom-up theory exploration.

3.1 Top-Down Lemma Discovery

Top-down techniques generate lemmas from an ongoing proof attempt for a theorem to be proved. These represent goal-directed methods that systematically derive the auxiliary lemmas needed to prove the target theorem. Induction in a theorem prover is challenging, as inductive proofs often need auxiliary lemmas. Much of the research effort on lemma discovery is driven by the centrality of lemma discovery to inductive theorem proving.

The discovery of conjectured lemmas can be guided by rippling and generalization. Some studies about CLAM^[25, 38] propose to discover lemmas from failure states for rippling, introducing lemma calculation and lemma speculation. Lemma calculation rewrites the fully-rippled term in the proof goal by induction hypothesis to construct missing wave-rules as lemmas, while lemma speculation introduces higher-order meta-variables and instantiations for further conjecturing. Lemma calculation and lemma speculation are later implemented in IsaPlanner^[26, 39]. Zeno^[40] is an automated inductive prover, and can discover lemmas by generalization with a counterexample finder to avoid over-generalization. PGT (Proof Goal Transformer for Isabelle/HOL)^[41] can generate conjectures through generalization followed by mutation using rewrite rules.

The above heuristic rewrite methods for top-down lemma discovery are usually goal-directed, specifically generating lemmas that are strongly related to proof states. However, the space of possible candidates has limited expressiveness because it is dependent on a particular set of rewrite rules and the proof state. To solve this, some studies propose lemma enumeration based on generalization. AdtInd^[42], based on generalization, guides the search for lemma enumeration through templates and proves lemmas by recursively invoking AdtInd. Coq’s lfind^[43] tool reduces the lemma synthesis problem to a program synthesis problem by first generalizing the proof goal and using input-output examples to guide the synthesis of expressions to complete the generalized goal. Since enumeration-based methods generate advantageous candidate conjectures, it is necessary to design filtering and ranking techniques. AdtInd filters candidate lemmas primarily through fast counterexample testing using the SMT solver Z3^[7], while lfind employs a multi-

②<https://www.anthropic.com/news/claude-3-5-sonnet>, Jan. 2026.

stage strategy combining counterexample search, triviality and redundancy checking, and proof-based ranking. In particular, template-based methods are also a widely employed method to filter valuable conjectures, such as TBC (template-based conjecturing)^[44]. Template-based methods are also used in bottom-up theory exploration tools for conjecturing.

3.2 Bottom-Up Theory Exploration

The concept of theory exploration was originally introduced by Buchberger's Theorema system^[45]. Theory exploration^[45–47] refers to the exploration process of mathematical discovery that systematically generates conjectures and theorems from a given set of axioms and definitions within a formal mathematical theory. While traditional theorem proving focuses on verifying the truth of specific conjectures, theory exploration aims to discover new mathematical knowledge. The goal is to enrich mathematical theories by automatically discovering interesting and useful theorems, thereby expanding the knowledge base available for further mathematical reasoning and proof construction.

In this subsection, we categorize theory exploration into three types of methods, as shown in Fig.3: generate-and-test, template-based, and deductive-reasoning methods. We also introduce some studies ex-

ploring specific theories.

3.2.1 Generate-and-Test Methods

Since conjectures and theorems are formulas, a natural approach is to enumerate formulas up to a certain bound and then test the formulas on some examples/instances or provers.

Model-based conjecture searching (MCS)^[48] enumerates terms like x , $f(x)$, $f(f(x))$, ... within a certain bound and constructs equational conjecture formulas from terms. Then each formula is tested against finite models generated from the input axioms. If a formula passes the test, it is considered a candidate conjecture. To avoid the generation of too many formulas, variable renaming and rewrite rules are used to eliminate redundant conjectures.

Like MCS, HR^[49] uses models generated from axioms and proves conjectures by OTTER^[4]. The key distinction of HR is its ability to invent new concepts from existing ones through 10 production rules (PR). These production rules transform concepts to generate novel mathematical concepts similar to AM^[50]. For example, starting with the multiplication in group theory ($a \times b = c$), HR can invent the concept of idempotent elements ($a \times a = a$). The iterative formation of concepts is driven by heuristic search and pruning. HR generates conjectures for concepts based

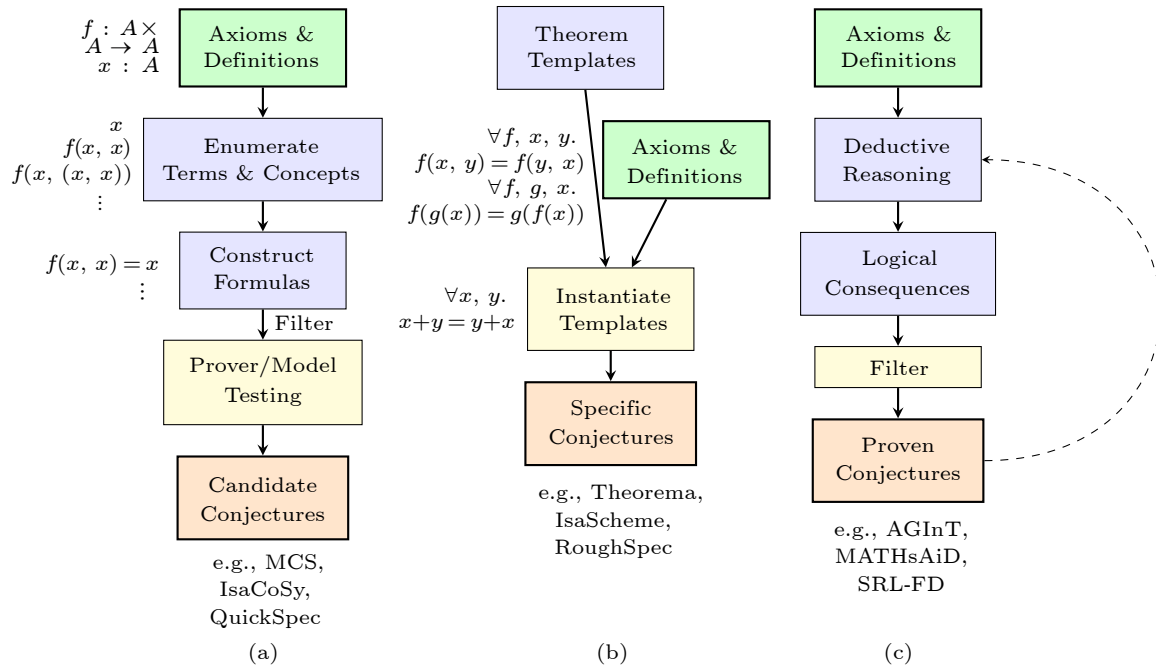


Fig.3. Three types of methods for bottom-up theory exploration. (a) The generate-and-test methods generate conjectures through enumeration and filter them using provers or model testing. (b) The template-based methods use user-defined theorem templates (shown as examples) to constrain the search space and generate specific conjectures. (c) The deductive-reasoning methods apply inference rules to derive theorems with guaranteed correctness, using filters to identify interesting ones.

on models, including non-existence, equivalence, and implication conjectures; then it proves them and adds counterexamples to the models. HR has performed case studies on algebra, graph theory, and number theory.

Similarly, bottom-up theory exploration is also employed in inductive theorem proving to generate auxiliary lemmas. IsaCoSy^[27], built on Isabelle^[10], is the first theory exploration system specifically designed to generate inductive conjectures. While IsaCoSy shares with MCS^[48] the key idea of generating only irreducible terms, it differs by employing a growing set of rewrite rules during exploration. IsaCoSy tests conjectures by Isabelle’s counter-example finder and prover IsaPlanner^[39]. IsaCoSy performs experiments on recursive theories (such as natural numbers, lists). However, IsaCoSy is bottlenecked by frequent calls to Isabelle.

QuickSpec^[51] is a lightweight theory exploration system to find equational properties. QuickSpec utilizes Haskell’s property-based testing tool^[52] to partition equivalence classes for each term. Since it only constructs equality conjectures for terms within the same equivalence class, it can filter out a large number of non-theorems. Although lacking proof capability and completeness, it significantly improves efficiency compared with other systems like IsaCoSy. QuickSpec has become a foundation for many theory exploration systems or automated theorem provers^[46, 53, 54].

Some work used generated conjectures as lemmas to prove more advanced theorems, surpassing the capabilities of provers like CLAM^[25], Zeno^[40]. Table 1 shows the experimental result of adding conjectures generated by theory exploration to theorem proving. HipSpec^[46] integrates QuickSpec into the automated theorem prover Hip^[55]. HipSpec successfully proved 47 out of 50 difficult theorems from benchmark in the TIP (Tons of Inductive Problems)^[56] library, compared with CLAM (41) and Zeno (21), making it one of the most advanced lightweight provers at the time. Similarly, Hipster^[57] uses QuickSpec as a subsystem

to generate conjectures and supports the discovery and proof in Isabelle/HOL.

Theory exploration systems like QuickSpec still face issues such as search space explosion. To optimize these enumeration-based methods, some studies introduce additional data structures or integrated deductive reasoning to reduce redundant search space and alleviate performance bottlenecks. Spectacular^[58] significantly reduces the search space by constructing equation graphs, enabling it to generate more equational conjectures compared with QuickSpec. TheSy (Theory Synthesizer)^[59] utilizes a syntax-guided enumerative method to generate formulas, but represents them using symbolic values, deriving equational conjectures through deductive inference to avoid the bottleneck of testing with models. The template-based methods mentioned below also constrain the search space by restricting the family of conjectures.

3.2.2 Template-Based Methods

Since enumeration-based methods may generate vast irrelevant conjectures, some studies introduce user-defined templates to enhance conjecture relevance and constrain the search space. These templates are often defined using higher-order logic and metavariables. Across different studies, templates may be described with varying terminology, such as schemas or patterns, which are uniformly referred to as templates in this work.

Theorema^[45] represents an early pioneering effort in employing templates for theory exploration. It defines templates in higher-order logic and instantiates the symbols of current theory to generate new concepts and conjectures. Like Theorema, IsaScheme^[60] uses the idea of user-defined templates, but automates the instantiation of templates on top of the simply-typed lambda calculus of Isabelle/HOL. IsaScheme filters out trivial conjectures (similar to IsaCoSy’s irreducibility heuristic), then filters out non-theorems by models, and finally proves convec-

Table 1. Evaluation of Combining Theory Exploration with Theorem Provers

Theory Exploration	Benchmark	Evaluation Result	
		with Exploration	w/o Exploration
HipSpec	TIP(50)	HipSpec: 47/50	CLAM: 41/50, Zeno: 21/50
Hipster	TIP(135)	Hipster: 131/135	Zeno: 103/135, CVC4: 120/135
TheSy	TIP(311)	CVC4+TheSy: 260/311	Z3: 75/311, CVC4: 70/311

Note: TIP(N) represents the evaluation subset including N theorem proving problems from the TIP dataset. In Evaluation Results, each “ $A : X/N$ ” means the experimental results of applying the prover to theorem proving: the prover A successfully proved X theorems out of N theorems. Additionally, the first column in Evaluation Results corresponds results of applying its theory exploration system to theorem proving.

tures using IsaPlanner^[39].

QuickSpec 2.0^[53] gives priority to generating the most general instances by enumerating templates to avoid more redundancy compared with QuickSpec^[51]. RoughSpec^[54], as an extension of QuickSpec, employs a template-based exploration approach like Theorema and IsaScheme. The difference is that RoughSpec proposes a hybrid method that uses QuickSpec for smaller conjectures and templates for more complex properties, thereby combining the advantages of enumeration-based and template-based methods.

Current template-based methods can effectively generate conjectures with specific forms, but providing high-quality templates still relies on users' experience. Existing research has also explored extracting template datasets from corpus, such as LOL^[61], which is extracted from Isabelle's Archive of Formal Proofs (AFP) and can be used for downstream tasks like automated conjecturing.

3.2.3 Deductive-Reasoning Methods

Deductive-reasoning methods start from axioms and previously derived theorems, repeatedly applying inference rules to generate new theorems. These methods are complete, which means that given sufficient resources, they can discover all provable theorems from a set of axioms. However, this completeness comes at the cost of computational intractability: the search space grows exponentially with inference depth, making exhaustive exploration infeasible for complex theories. The quality of generated theorems depends heavily on the reasoning engine, including inference rules, search strategies, and heuristics. Consequently, they typically serve as auxiliary tools for mathematicians to find more theorems in theory.

AGInT (Automated Generation of Interesting Theorems)^[62] is a system for first-order logic. This system generates a set of logical results from a group of axioms using a first-order prover named E^[6] as a reasoning engine, and then utilizes runtime filters to identify interesting theorems. AGInT was evaluated in the context of logical puzzles (puz) from set theory and the TPTP library^[63].

MATHsAiD^[64] is a theory exploration system designed for mathematicians, which allows users to propose various hypotheses and interesting terms for further exploration manually. It has three components: the hypothesis generator (HG), theorem generator (TG), and theorem filter (TF). TG gets a set of hy-

potheses H_k (like A is a natural number), ranging from simple to complex, generated by HG. During forward-chaining exploration, TG maintains two theories: a generating theory $\mathcal{G}_k$ and a trivial theory $\mathcal{T}_k$, and retains only conclusions that could be generated by the generating theory but could not be proven by the trivial theory as follows:

$$C_k = \{c : H_k \vdash_{\mathcal{G}_k} c \text{ and } H_k \not\vdash_{\mathcal{T}_k} c\}.$$

Then TF performs filtering by redundancy and simplicity tests. Based on the previous MATHsAiD, MATHsAiD 2.0^[65] integrates backward and forward reasoning, enabling the layered exploration of higher-level theories, such as Zariski spaces. Gao *et al.*^[66] proposed the SRL-FD method (strong relevant logic-based forward deduction method). Strong relevant logic^[67] is a non-classical logic that requires relevance between premises and conclusions. This method has been systematically applied in case studies involving NBG (von Neumann–Bernays–Gödel) set theory, Tarski's geometry, and Peano arithmetic.

3.2.4 Exploration in Specific Theories

While the above methods primarily focus on general theory and theorem proving, theory exploration has also been reflected in scientific discovery, especially in mathematics. These approaches emphasize discovering novel conjectures and invariants in specific mathematical domains such as graph theory and geometry, often combining symbolic reasoning with heuristics and specific theory knowledge.

Early pioneering systems demonstrated the potential of automated conjecturing in mathematical domains. Automated Mathematician (AM)^[50] and its successor Eurisko^[68] are among the earliest discovery systems. AM can generate concepts like disjoint sets and prime numbers. AM successfully generates conjectures including Goldbach's conjecture and the unique factorization theorem through heuristic exploration. However, these early efforts often merely rediscover existing theorems, lacking truly innovative conjectures. Tang and Lin^[69] modeled game theory, particularly (pure) Nash equilibria, using first-order logic and checked by finite models. This study discovers new classes of games and provides automated tools for the classification and structural analysis of game theory.

The exploration of graph theory has led to several successful automated conjecturing systems. The GT system^[70] can discover new mathematical con-

cepts, generate conjectures, and prove theorems in graph theory. Graffiti^[71] represents a significant advance, generating conjectures based on templates like $I \leq J$, with a graph-theoretic invariant set and validating them on a graph data library. Unlike earlier systems like AM and GT, Graffiti successfully generated novel research conjectures, with its discovered graph-theoretic quantities leading to numerous publishable mathematical papers. TxGraffiti^[72] extends Graffiti by incorporating linear programming to formulate bounds of conjectures.

Automated conjecturing in geometry has seen significant research^[73]. The pioneering work of Wu^[74] is the cornerstone of automated reasoning in geometry. Wu’s characteristic set method was used to discover geometry theorems^[75]. Following that, Bagai’s systems^[76] incrementally generate geometric situations (basic geometric objects such as points and lines), test consistency using a prover based on Wu’s algebraic method, and derive theorems from inconsistency. Similarly, GeoGebra Discovery^[77] generates conjectured properties related to given points, and these properties are verified by symbolic Gröbner-basis computations^[78]. There are also methods using deductive reasoning. Chou *et al.*^[79] proposed a deductive approach that constructs a geometric deductive database system to find all properties (fixed points) of a given geometric configuration by repeatedly applying geometric rules. Peng *et al.*^[80] proposed to construct vector equations from geometric shapes and apply the elimination rule to derive the polynomial relations.

4 Data-Driven Approaches

In recent years, much progress has been made in machine learning. We can formulate conjectures from data (experimental data or formal corpora). As a motivating example, suppose we have the following two concrete examples or facts: $3 + 4 = 7$, $4 + 3 = 7$. What kind of conjectures can we infer from these equations? One may infer that, for all integers m, n , $m + n = n + m$. Or one may conclude with a stronger conjecture like, for all natural numbers n , $n + (n + 1) = (n + 1) + n$ is a prime number. The latter is not valid.

Data-driven methods to automated conjecturing constitute a different paradigm from purely symbolic methods, as they learn potential patterns from numerical data and formal corpora. This section reviews these methods, which fall into these categories:

1) data-driven scientific discovery systems, 2) statistical analogy learning methods, 3) neural methods based on LLMs, and 4) theorem generation for synthetic data. The basis for the classification is as follows.

- *Application.* Scientific discovery systems focus on domain-specific research, while statistical analogy learning and neural methods are general-purpose across diverse domains in logical systems.

- *Data.* Scientific discovery systems primarily analyze numerical experimental data. Statistical analogy learning and neural methods learn syntax and semantics from formal corpora. Theorem generation provides synthetic data for further data-driven training.

- *Technique.* Statistical analogy learning methods use traditional techniques (e.g., clustering) for analogy discovery. Neural methods leverage deep learning architectures, particularly LLMs, to generate novel conjectures.

4.1 Data-Driven Scientific Discovery Systems

With the advancement of machine learning techniques, several studies have demonstrated the potential of data-driven discovery in mathematics and science by learning from experimental data. Table 2 shows some valuable and well-formed conjectures from scientific discovery studies.

Table 2. Example of Conjecture Generation in Specific Domains

Work	Conjecture
BACON ^[81]	Kepler’s third law: $d^3[(a - k_2)/t]^2 = k_3$
AGX ^[82]	Randic index: $\bar{l}(G) \leq Ra(G)$
Ramanujan ^[83]	$\frac{4}{3\pi - 8} = 3 - \frac{11}{6 - \frac{2 \times 3}{9 - \frac{3 \times 5}{12 - \dots}}}$
AI Feynman ^[84]	Friedman equation: $H = \sqrt{\frac{8\pi G}{3} \rho - \frac{k_f c^2}{a_f^2}}$

The BACON program^[81] automates scientific discovery based on experimental data to formulate scientific laws. This early work established the foundation for data-driven methods by showing that meaningful conjectures could emerge from systematic analysis of empirical data.

AutoGraphiX (AGX)^[82, 85] was a pioneering data-driven system for graph conjectures. It uses a vari-

able neighborhood search (VNS) metaheuristic to explore extremal values of graph invariants such as diameter, independence number, and chromatic number across a database of graphs. While early versions rely on interactive analysis, the later version proposes three methods to generate conjectures automatically^[85]: 1) a numerical method based on principal component analysis (PCA), 2) a geometric method that constructs convex hulls to yield linear inequalities, and 3) an algebraic method that derives new inequalities based on known formulas.

In the domain of mathematical constants, the Ramanujan Machine^[83] generates conjectures in continued fraction identities by enumerating polynomial coefficients and validating them based on the numerical properties of mathematical constants. A generation method based on sampling and gradient descent was also proposed. These approaches combine enumeration and data-driven learning to discover novel mathematical relationships that may not be apparent through purely symbolic reasoning.

Davies *et al.*^[86] established an interactive framework in which neural networks learn functions between mathematical objects, and attribution techniques guide intuition, leading to novel conjectures in knot theory and representation theory. Similarly, AI Feynman^[84] employs neural networks to discover latent structures in data, such as symmetry or separability properties. It then uses these properties to simplify symbolic regression problems recursively, leading to the discovery of all physics equations from the Feynman Database^③.

4.2 Statistical Analogy Learning Methods

Some researchers have applied analogy learning methods to formal mathematics corpora to discover structural or semantic patterns. Unlike methods in scientific discovery, which often rely on numerical experimental data, these methods learn relationships and analogies from the semantics of mathematical statements.

Applications of statistical methods to theorem proving demonstrate promising results. ACL2(ml)^[87, 88] uses statistical pattern recognition to suggest auxiliary lemmas by analogy, discovering families of related theorems through clustering algorithms. It analyzes proof patterns in the ACL2 corpus to identify

similar theorems and generates candidate lemmas based on structural similarities. Gauthier *et al.*^[89] used statistical analogies extracted from the Mizar Mathematical Library (MML) to generate conjectures. This work utilizes statistical concept matching^[90] as an analogical method to identify sets of similar concepts and generate conjectures based on similarity. These studies demonstrate that analogy learning techniques can effectively learn similarity and propose valuable conjectures from large formal corpora, thereby reducing search spaces in theorem proving.

4.3 Neural Methods Based on LLM

Neural methods, driven by advances in deep learning and LLMs, have emerged as powerful approaches to automated conjecturing. Compared with earlier statistical analogy learning methods, they learn more complex syntactic and semantic patterns from large-scale formal corpora, thereby generating more diverse conjectures, which complement traditional symbolic methods.

Similar to the classification of symbolic methods, neural methods for automated conjecturing can be categorized based on their purposes and starting points: 1) theory exploration methods, which generate conjectures as new knowledge from background without a specific proof goal, and 2) lemma discovery methods, which are goal-directed to generate auxiliary conjectures for proving a target theorem.

4.3.1 Neural Methods for Theory Exploration

Early work demonstrated the potential of neural methods for automated conjecturing. In the first experiments of neural methods, Urban and Jakubuv^[91] applied a GPT-2 model to a dataset derived from the Mizar Mathematical Library. The model successfully generates novel and syntactically correct conjectures in formal language, although its outputs also contain redundancies and non-theorems. Another early effort was made by Rabe *et al.*^[92], who trained a language model for mathematical reasoning on the HOList dataset^[93]. The model performs various tasks including type inference, hypothesis completion, and free-form conjecturing, achieving the result of generating 12% to 26% non-redundant and provable conjectures.

^③<https://space.mit.edu/home/tegmark/aifeynman.html>, Jan. 2026.

Recent work has further explored the capabilities of LLMs for conjecturing. Johansson and Smallbone^[94] conducted a systematic comparison of GPT-3.5 and GPT-4 against symbolic methods for theory exploration, demonstrating that augmenting these symbolic methods with LLMs could generate more diverse conjectures. MINIMO (Mathematics from Intrinsic Motivation)^[95] presents a conjecturing-proving loop: a language model generates conjectures from axioms for a prover model to attempt. Crucially, the system extracts provable conjectures and proofs from their failure proof search trees. This iterative feedback mechanism continuously improves the system’s performance in both conjecturing and proving over time.

Recognizing the complementary strengths of neural and symbolic methods, LEMMANAID^[96] uses LLMs to generate lemma templates and symbolic methods to instantiate them as Isabelle/HOL conjectures. The LLMs suggest appropriate analogous lemma patterns likely to be relevant to the theory, while symbolic instantiation ensures formal correctness. This neural-symbolic hybrid method demonstrates that combining data-driven pattern learning with symbolic reasoning can generate more diverse and relevant conjectures, representing a promising direction for future research.

4.3.2 Neural Methods for Lemma Discovery

For proving a target theorem, neural methods have been applied to generate immediate conjectures as useful auxiliary lemmas. Valuable conjectures can decompose complex theorems modularly and recursively and provide high-quality guidance for proof.

A prominent direction in neural methods for lemma discovery is the generation of planning or proof sketches, which serve as intermediate subgoal structures before proving with conjectures. LEGO-Prover^[97] prompts LLMs to generate subgoal lemmas, then proves theorems in a block-by-block manner with a growing lemma library. Draft, Sketch, and Prove (DSP)^[98] uses informal mathematical proofs to generate formal proof sketches with sequences of intermediate conjectures; these decomposed subgoals are then discharged by automated provers like Sledgehammer^[22]. DSP+^[99] revives the DSP method with fine-grained neural-symbolic enhancements at each stage, achieving performance comparable to reinforcement learning approaches without requiring additional model training. POETRY^[100] recursively generates

verifiable proof sketches with intermediate conjectures, solving them at progressively deeper levels. Compared with the DSP method, the POETRY method supports a deeper, recursive decomposition of problems or conjectures. Following this recursive decomposition paradigm, several recent studies (e.g., BFS-Prover-V2^[101], DeepSeek-Prover-V2^[102], Delta Prover^[103], and Seed-Prover^[104]) further leverage such structured reasoning within LLMs.

Recent advances have further enhanced LLMs’ lemma discovery capability through reinforcement learning. Kimina-Prover^[105] employs large-scale reinforcement learning with structured reasoning patterns, guiding models to generate intermediate steps and subgoals that simulate human problem-solving strategies. StepFun-Prover^[106] integrates tool-based reinforcement learning with Lean’s feedback, enabling iterative refinement of proof strategies through step-by-step verification and sub-lemma generation.

4.4 Theorem Generation for Synthetic Data

The datasets for theorem proving problems often come from manually written corpora, leading to a challenge of data scarcity. Some work creates datasets from existing corpora and manual curation. A direct application of automated conjecturing is to generate more theorems (and proofs), which can alleviate data scarcity in data-driven methods. It enables the generation of large-scale training data essential for neural models^[105, 107]. Previous automated conjecturing studies provide valuable theorems to corpora. For example, HR^[49] generates 184 theorems that were accepted into the TPTP library^[63]. In this subsection, we discuss recent efforts in synthesizing theorem data. Table 3 summarizes the relevant synthetic data generation work discussed in this subsection.

Rule-based generators can produce new theorem data by iteratively sampling inference rules, similar to the deductive-reasoning methods mentioned above in theory exploration. Firoiu *et al.*^[108] developed a theorem synthesizer called FwdP, which generates new theorems by applying resolution from axioms through random sampling. These synthesized theorems were used to train neural provers, achieving superior performance in both time and search steps compared with the E-prover. MetaGen^[109] employs a neural generator to create human-like handwritten theorems, iteratively applying substitution rules to derive new

Table 3. Summary of Synthetic Data Generation

Method	Synthetic Dataset	Language (Source)	Source Size ($\times 10^3$)	Data Size ($\times 10^3$)
Rule-based	FwdP ^[108]	TPTP (TPTP) ^④	1.0	500.0
	MetaGen ^[109]	MetaMath (set.mm)	21.7	10 000.0
	INT ^[110]	DSL (Inequality)	–	1 500.0
	AlphaGeometry ^[111]	DSL (Geometric)	–	100 000.0
	AIPS ^[112]	DSL (Inequality)	–	191.0
	Alchemy ^[113]	Lean (Mathlib)	110.0	6 000.0
LLM-based	LeanConjecturer ^[114]	Lean(Mathlib)	–	4.0
	STP ^[115]	Lean(Lean-Workbook)	89.0	2 000.0
Lemma-based	ATG ^[116]	MetaMath (set.mm)	3.5	70.0
	REFACTOR ^[117]	MetaMath (set.mm)	27.2	1.9
	LEGO-Prover ^[97]	Isabelle (miniF2F)	0.5	22.5

Note: Language (Source) represents the language style of synthetic data with the source where the dataset is generated from. DSL means domain-specific language. Source Size represents the number of theorems or problems used for generation in the source. Dataset Size represents the number of theorems or lemmas in synthetic data.

theorems. Alchemy^[113] utilizes the Lean mathlib corpus, expanding the number of theorems from 110k to 6 million through symbolic perturbation. The INT benchmark^[110] and AIPS (Algebraic Inequality Proving System)^[112], containing 1.5 million and 191k inequality problems, respectively, use theorem generators that sample axioms and morph trivial statements to produce non-trivial theorems. AlphaGeometry^[111], a geometric theorem prover, synthesized 100 million theorem data for training, achieving gold medal-level results in Olympic geometry proving contests. These methods systematically generate non-trivial theorems, aligning with the goals of theorem finding.

In addition to rule-based theorem generation, some studies utilize LLMs to generate new theorems from existing theorem statements. LeanConjecturer^[114] leverages context from Mathlib files and uses LLMs to generate conjectures. These candidates are filtered by Lean to discard invalid, redundant, or trivial ones. The high-quality conjectures were incorporated into the subsequent iteration. STP (Self-Play Theorem Prover)^[115], similar to MINIMO^[95], employs a conjecturing-proving self-play training method. However, STP places greater emphasis on generating similar high-quality conjectures from existing theorems, rather than simulating an increasingly advanced exploration-proof process. STP generates 2 million conjectures over 24 iterations, while its trained prover achieves state-of-the-art performance.

Furthermore, some research has focused on generating useful intermediate lemmas from successful

proofs. Mathematicians can identify modular and reusable theorems, significantly simplifying theorem proving. The REFACTOR^[117] method represents theorems as connected components in proof trees and trains neural classifiers on these components to identify valuable lemmas. The ATG (Automated Theorem Generation)^[116] benchmark from Metamath adopts a self-play method: it uses graph neural networks to represent existing theorem relationships, then employs Monte Carlo Tree Search (MCTS) to iteratively select a state and action to expand. LEGO-Prover^[97] maintains a growing library of lemmas generated by LLMs for modular proof construction and evaluates them by frequency of reuse in updated proofs.

5 Interestingness of Conjectures

In the automated conjecturing studies described in previous sections, it has been observed that automated systems frequently generate numerous meaningless or uninteresting conjectures. Consequently, the interestingness of conjectures has become equally crucial in automated conjecturing research. In this section, we will discuss the heuristic and pruning strategy, and the evaluation of the interestingness of generated conjectures.

5.1 Heuristic and Pruning Strategy

The filtering of uninteresting conjectures is employed as a pruning strategy to reduce the number of

^④<https://tptp.org/TPTP/>, Jan. 2026.

generated theorems. Colton *et al.*[118] proposed some dimensions of interestingness, including empirical plausibility and novelty. As for empirical plausibility, which means that false conjectures are uninteresting, some automated conjecturing systems, such as MCS[48], Graffiti[71], and HR[49], use data (such as graph data) or models to keep empirical plausibility by filtering out non-theorems. QuickSpec[51] also adopts a similar idea to filter out non-theorems by equivalence classes formed by the models. As for novelty, MCS[48] and HR[49] use rewrite rules to filter out redundant terms or concepts. Similarly, IsaCoSy[27] filters redundant and trivial conjectures by growing rewrite rules and also filters out non-theorems to keep plausibility by using Isabelle’s counterexample checking tool QuickCheck[119] and the prover IsaPlanner[39]. MATHsAiD[64] ensures non-triviality by maintaining a generating theory $\mathcal{G}_k$ and a trivial theory $\mathcal{T}_k$, retaining only those conjectures c that cannot be proven by the trivial theory. Based on these studies, we can summarize several characteristics of uninteresting conjectures: 1) non-theorems, 2) redundant, and 3) trivial.

Graffiti[71] pioneers several influential heuristics for filtering and evaluating conjectures based on novelty. The Dalmatian heuristic requires stronger conjectures so that they contribute new information. For example, if conjectures of the form $A \leq B_i$ already exist, a new conjecture $A \leq C$ will pass the Dalmatian heuristic if there exists at least one graph G for which $C(G) < B_i(G)$ for all i . This indicates that the new conjecture provides novel information for graph G . The Beagle heuristic focuses on detecting trivial or tautological conjectures, such as $i(G) \leq i(G) + 1$. If two graph invariants in a conjecture are semantically similar, the Beagle heuristic discards those trivial ones. Larson’s program[120], based on Graffiti and its Dalmatian heuristic, serves as a general mathematical discovery system, with applications extended to other fields such as number theory and matrix theory.

In the process of generating concepts through heuristic search like AM[50] and HR[49], each concept has an interestingness measurement, and the system prioritizes the exploration of concepts with high interestingness. In the process of deductive reasoning, AGInT[62] proposes computable measurements as run-

time filters for pruning. The following are some of the proposed measurements.

1) *Obviousness*: the number of inferences in its derivation. Formulas that are difficult to prove are interesting.

2) *Complexity*: the number of distinct function and predicate symbols. Formulas that contain many different symbols are interesting.

3) *Surprisingness*: formulas that contain symbols which are seldom seen together are interesting.

[121] also discussed the measurement of interesting conjectures, including: the degree of logical connectives V_d ^⑤, propositional schemas (templates) of formula V_p ^⑥, abstract level V_a ^⑦, and deductive distance V_e . By analyzing the theorems of NBG set theory, Peano’s arithmetic, and Tarski’s geometry, [121] draws the following conclusions about these measurements: more interesting theorems always 1) have a lower degree of logical connectives, 2) have more frequent propositional schemata, 3) have a higher abstract level, and 4) are more difficult to deduce from premises. In general, different studies possibly tend to favor different measurements, but the difficulty of proving conjectures is a common important measure of their interestingness.

5.2 Evaluation of Conjectures

Wang[2] proposed several crude metrics for evaluating the interestingness of conjectures: 1) importance and centrality: a theorem occurs frequently during many proofs, and 2) deep theorem: a short theorem that can only be proved with long proofs. However, these indicators are vague and lack universality, only providing a potentially feasible idea.

Case studies are an important method for evaluating automated conjecturing. In many studies, especially in scientific discovery, experiments involve analyzing automatically generated conjectures and theorems within a theory context through expert evaluation. Johansson *et al.*[27, 46] proposed to evaluate the theorems generated by IsaCoSy using the Isabelle library as a reference through precision and recall analysis. Precision analysis computes the portion of generated theorems that are shared with the library. Recall analysis computes the portion of the library theo-

^⑤The number of occurrences of logical connectives.

^⑥Propositional schemas are like $A \rightarrow B$, $(A \wedge B) \rightarrow C$, ...

^⑦The depth of predicate and function definitions. For example, predicate $\subseteq$ is defined on $\in$, thus predicate $\subseteq$ is higher-level predicate.

rems that are discovered. For example, Isabelle’s library contains 12 equational theorems, 10 of which are synthesized by IsaCoSy (shared theorems), and IsaCoSy synthesizes a total of 16 theorems. Let n_s , n_g , and n_l denote the number of shared theorems, generated theorems, and library theorems, respectively. The metrics are computed as follows.

$$\text{Precision} = \frac{n_s}{n_g} = \frac{10}{16} = 62.5\%,$$

$$\text{Recall} = \frac{n_s}{n_l} = \frac{10}{12} = 83.3\%.$$

Performance analysis was also conducted for some automated conjecturing systems^[27, 53, 58], especially bottom-up theory exploration systems, covering aspects such as running time, space usage, and synthesis space.

An important application of conjectures is to prove more theorems. Therefore, the studies related to automated conjecturing and theorem proving were often evaluated on theorem proving datasets by comparing provers that integrate automatically generated conjectures with other provers. It could be considered that valuable conjectures help provers to prove more complicated theorems. There are some benchmarks commonly used in theorem proving as follows.

1) TPTP[®] (Thousands of Problems for Theorem Provers)^[63]: a library of test problems for Automated Theorem Proving (ATP).

2) TIP (Tons of Inductive Problems)^[56]: benchmarks for inductive theorem provers.

3) miniF2F^[122]: a unified cross-system benchmark of 488 formal mathematics competition problems.

4) ProofNet^[123]: a benchmark of 371 undergraduate-level mathematics problems in both Lean and natural language.

5) PutnamBench^[124]: a multi-language benchmark from mathematical competition with formalizations in Lean, Isabelle, and Coq.

TIP has been used for experimental evaluation in several studies^[46, 59, 125] related to inductive theorem proving and automated conjecturing, while miniF2F, ProofNet, and PutnamBench have recently been adopted for experimental evaluation in many studies^[97, 98, 100] on neural theorem proving (NTP). ICML also proposed a challenge track[®] named Automated Theorem Generation and Proving, and provid-

ed an evaluation method using the ATG dataset^[116] from set.mm^[126]. Given a set of axioms and symbols, new theorems are evaluated by metrics like average proof reduction (APR) and precision using a random sample of problems.

6 Challenge and Future Work

6.1 Challenge

6.1.1 Limitations of Symbolic Conjecturing Methods

Most work on conjecturing relies on symbolic methods like heuristic enumeration and deductive reasoning. Many studies attempt to improve automated conjecturing systems through various heuristic strategies and filter mechanisms, such as interestingness metrics and template-based methods. However, these optimizations do not fundamentally resolve the underlying problem of exponential conjecture space. For instance, in the case of QuickSpec, exploration becomes exceedingly difficult when confronted with large theories, such as libraries containing over 30 functions (HugeLists)^[53]. The explosive growth of the search space fundamentally constrains symbolic methods in generating innovative conjectures with complex structures. Symbolic methods lack sufficiently intelligent guidance, leading to significant computational overhead in generating and validating useless and irrelevant conjectures, thereby making it difficult to identify genuinely innovative ones.

6.1.2 Limitations of Neural Conjecturing Methods

Compared with traditional symbolic methods, neural methods can effectively propose relevant conjectures. However, neural methods like LLMs also expose shortcomings compared with traditional symbolic systems^[94]. Firstly, there is the issue of controllability and extensivity. LLMs function as black-box systems and produce a limited number of conjectures with randomness and hallucinations, making it challenging to systematically explore the potential lemmas and control search strategies. Secondly, a critical challenge is balancing the use of prior knowledge in LLMs while avoiding mere recitation or memorization.

[®]<https://tptp.org/TPTP/>, Jan. 2026.

[®]<https://sites.google.com/view/ai4mathworkshopicml2024/challenges>, Jan. 2026.

This stems from the untransparency of neural models.

The neural methods introduced above often rely on LLMs’ reasoning capabilities and experience from existing proofs to generate conjectures. In other words, the ability to conjecture emerges as a byproduct of proving and memorization. A predominant limitation is their propensity to rediscover or generalize existing theorems, rather than producing conjectures of genuine novelty. How to specially improve a neural model’s capabilities for conjecturing is an open research question.

6.1.3 Limitations of Metrics and Evaluation Methods

Currently, the quality assessment of conjectures primarily focuses on case studies and theorem proving tasks. However, the selection of different theories for case studies or different theorem proving benchmarks across various studies creates significant challenges for unified evaluation and comparative analysis. Consequently, automated conjecturing still lacks unified and systematic metrics and evaluation methods.

Current evaluation methods are optimized to filter out trivial, redundant or false conjectures. However, they lack the capability to identify conjectures with genuine research potential. This situation reflects a fundamental challenge: the quantitative imbalance between formal correctness and mathematical creativity. Currently, the accurate evaluation of mathematical value relies on case studies conducted by experts and lacks automated methods.

6.2 Future Work

6.2.1 Benchmarks and Datasets

A direct application of conjecturing is to generate

more high-quality theorems, which can alleviate the scarcity in theorem proving. Table 4 shows the size of some commonly used datasets. Obviously, the size of the datasets is insufficient for LLMs. Although some existing work can synthesize theorem data through the conjecturing process, the size of the datasets is still far behind successful domains such as code generation and computer vision. Moreover, the complexity and quality of synthesized data often fail to match theorems in manually crafted corpus. The generation of high-quality datasets through conjecturing is a significant challenge in addressing data scarcity.

As previously mentioned, the field of automated conjecturing currently suffers from a lack of high-quality benchmarks and standardized evaluation methods, which significantly constrains its further development. In a related discussion, Reddy and Shojaei^[133] discussed opportunities and challenges for generative AI in scientific discovery, which can be a reference for automated conjecturing, highlighting several key directions: 1) developing benchmarks that focus on innovative scientific discovery rather than replication, 2) creating multifaceted evaluation metrics for scientific discovery to assess novelty and generalizability, and 3) involving domain experts in the design and evaluation of datasets.

6.2.2 Neural-Symbolic Methods

Neural methods for conjecturing are promising. The integration of neural methods (LLMs) with symbolic methods and symbolic tools like theorem provers represents a potentially important research direction for the future. Neural-symbolic methods effectively mitigate LLM limitations such as hallucinations and lack of controllability, leading to more accurate and trustworthy conjecturing and proving. Current neural-symbolic methods only remain at a shallow inte-

Table 4. Some Commonly Used Datasets for Different Formal Systems

Dataset	Language	Size ($\times 10^3$)	Source
CoqGym ^[127]	Coq	71	123 projects from GitHub
PISA ^[128]	Isabelle	183	AFP, Isabelle standard library
FVELER ^[129]	Isabelle	29	seL4 ^[16]
LeanDojo ^[130]	Lean	98	Mathlib
Lean-Github ^[131]	Lean	28	GitHub projects
NuminaMath-Lean ^[105]	Lean	104	NuminaMath 1.5 dataset ^[132]
HOList ^[93]	HOL Light	32	Kepler conjecture
set.mm ^[126]	MetaMath	38	Classical logic, ZFC
TPTP ^[63]	TPTP	26	–

Note: Size indicates the number of theorem proving tasks in the dataset.

gration level, such as neural generation of templates and symbolic instantiation^[96]. Ultimately, we hope that neural-symbolic architectures will eventually combine the powerful learning capability of neural models with the controllability and completeness of symbolic methods and tools for automated conjecturing.

6.2.3 Cross Domain Transfer and Intersections

A key limitation of current automated conjecturing systems is their specialization. Most methods are designed and evaluated within a closed mathematical domain (e.g., graph theory or number theory) within an experiment, rarely producing conjectures across different domains. Symbolic methods are constrained by their search space, making them difficult to apply to exploration with multiple domains. By comparison, analogy learning and neural methods still lack the capacity to grasp the deep semantic structures and abstract relations across different domains, remaining at the level of similarity rather than structural isomorphism. This is a significant barrier to achieving truly novel discovery, as major mathematical advances often stem from cross-domain analogies and conceptual intersections. Future research should prioritize cross-domain structural reasoning, such as knowledge graph and meta-learning.

7 Conclusions

This survey provided, to the best of our knowledge, the first comprehensive framework of automated conjecturing and theorem finding methods. We categorized them into symbolic methods (top-down lemma discovery and bottom-up theory exploration) and data-driven methods (statistical analogy learning and neural methods with LLMs). Data-driven methods also incorporate related data-driven scientific discovery and synthetic data generation techniques. We reviewed strategies and evaluation methods of interestingness, which are crucial for distinguishing novel conjectures from trivial results. Finally, automated conjecturing faces key challenges including the limitations of symbolic methods, neural methods, and evaluation methods. Future research directions focus on developing high-quality benchmarks and standardized evaluation methods, advancing neural-symbolic methods, and conjecturing cross different domains.

Conflict of Interest Jian Zhang is an editorial board member for the Journal of Computer Sci-

ence and Technology and was not involved in the editorial review of this article. The authors declare that there are no other competing interests.

References

- [1] Wos L. The problem of automated theorem finding. *Journal of Automated Reasoning*, 1993, 10(1): 137–138. DOI: [10.1007/BF00881868](https://doi.org/10.1007/BF00881868).
- [2] Wang H. Toward mechanical mathematics. *IBM Journal of Research and Development*, 1960, 4(1): 2–22. DOI: [10.1147/rd.41.0002](https://doi.org/10.1147/rd.41.0002).
- [3] Wos L, Pieper G W. A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning. World Scientific, 1999. DOI: [10.1142/4132](https://doi.org/10.1142/4132).
- [4] McCune W. Otter 2.0. In *Lecture Notes in Computer Science 449*, Stickel M E (ed.), Springer, 1990, pp.663–664. DOI: [10.1007/3-540-52885-7_131](https://doi.org/10.1007/3-540-52885-7_131).
- [5] Kovács L, Voronkov A. First-order theorem proving and Vampire. In *Lecture Notes in Computer Science 8044*, Sharygina N, Veith H (eds.), Springer, 2013, pp.1–35. DOI: [10.1007/978-3-642-39799-8_1](https://doi.org/10.1007/978-3-642-39799-8_1).
- [6] Schulz S. E - a Brainiac theorem prover. *AI Communications*, 2002, 15(2/3): 111–126. DOI: [10.5555/1218615.1218621](https://doi.org/10.5555/1218615.1218621).
- [7] de Moura L, Bjørner N. Z3: An efficient SMT solver. In *Lecture Notes in Computer Science 4963*, Ramakrishnan C R, Rehof J (eds.), Springer, 2008, pp.337–340. DOI: [10.1007/978-3-540-78800-3_24](https://doi.org/10.1007/978-3-540-78800-3_24).
- [8] Barbosa H, Barrett C, Brain M, Kremer G, Lachnitt H, Mann M, Mohamed A, Mohamed M, Niemetz A, Nötzli A, Ozdemir A, Preiner M, Reynolds A, Sheng Y, Tinelli C, Zohar Y. cvc5: A versatile and industrial-strength SMT solver. In *Lecture Notes in Computer Science 13243*, Fisman D, Rosu G (eds.), Springer, 2022, pp.415–442. DOI: [10.1007/978-3-030-99524-9_24](https://doi.org/10.1007/978-3-030-99524-9_24).
- [9] Paulson L C. Isabelle: A Generic Theorem Prover. Springer, 1994. DOI: [10.1007/BFb0030541](https://doi.org/10.1007/BFb0030541).
- [10] Nipkow T, Wenzel M, Paulson L C. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer, 2002. DOI: [10.1007/3-540-45949-9](https://doi.org/10.1007/3-540-45949-9).
- [11] Barras B, Boutin S, Cornes C, Courant J, Filliâtre J C, Giménez E, Herbelin H, Huet G, Muñoz C, Murthy C, Parent C, Paulin-Mohring C, Saïbi A, Werner B. The Coq proof assistant reference manual: Version 6.1. Technical Report RT-0203, INRIA, 1997. <https://inria.hal.science/inria-00069968>, Jan. 2026.
- [12] de Moura L, Ullrich S. The Lean 4 theorem prover and programming language. In *Lecture Notes in Computer Science 12699*, Platzer A, Sutcliffe G (eds.), Springer, 2021, pp.625–635. DOI: [10.1007/978-3-030-79876-5_37](https://doi.org/10.1007/978-3-030-79876-5_37).
- [13] Harrison J. HOL light: A tutorial introduction. In *Lecture Notes in Computer Science 1166*, Srivas M, Camilleri A (eds.), Springer, 1996, pp.265–269. DOI: [10.1007/BFb0031814](https://doi.org/10.1007/BFb0031814).
- [14] Brock B, Kaufmann M, Moore J S. ACL2 theorems

- about commercial microprocessors. In *Lecture Notes in Computer Science 1166*, Srivas M, Camilleri A (eds.), Springer, 1996, pp.275–293. DOI: [10.1007/BFb0031816](https://doi.org/10.1007/BFb0031816).
- [15] Naumowicz A, Kornilowicz A. A brief overview of MIZAR. In *Lecture Notes in Computer Science 5674*, Berghofer S, Nipkow T, Urban C, Wenzel M (eds.), Springer, 2009, pp.67–72. DOI: [10.1007/978-3-642-03359-9_5](https://doi.org/10.1007/978-3-642-03359-9_5).
- [16] Klein G, Elphinstone K, Heiser G, Andronick J, Cock D, Derrin P, Elkaduwe D, Engelhardt K, Kolanski R, Norrish M, Sewell T, Tuch H, Winwood S. seL4: Formal verification of an OS kernel. In *Proc. the 22nd Symposium on Operating Systems Principles*, Oct. 2009, pp.207–220. DOI: [10.1145/1629575.1629596](https://doi.org/10.1145/1629575.1629596).
- [17] Leroy X. Formal verification of a realistic compiler. *Communications of the ACM*, 2009, 52(7): 107–115. DOI: [10.1145/1538788.1538814](https://doi.org/10.1145/1538788.1538814).
- [18] Gonthier G. The four colour theorem: Engineering of a formal proof. In *Lecture Notes in Computer Science 5081*, Kapur D (ed.), Springer, 2007, Article No. 333. DOI: [10.1007/978-3-540-87827-8_28](https://doi.org/10.1007/978-3-540-87827-8_28).
- [19] Gonthier G. Formal proof—The four-color theorem. *Notices of the American Mathematical Society*, 2008, 55(11): 1382–1393.
- [20] Hales T, Adams M, Bauer G, Dang T D, Harrison J, Hoang L T, Kaliszyk C, Magron V, McLaughlin S, Nguyen T, Nguyen Q T, Nipkow T, Obua S, Pleso J, Rute J, Solovyev A, Ta T H A, Tran N T, Trieu T D, Urban J, Vu K, Zumkeller R. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 2017, 5:e2. DOI: [10.1017/fmp.2017.1](https://doi.org/10.1017/fmp.2017.1).
- [21] The Mathlib Community. The Lean mathematical library. In *Proc. the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, Jan. 2020, pp.367–381. DOI: [10.1145/3372885.3373824](https://doi.org/10.1145/3372885.3373824).
- [22] Blanchette J C, Böhme S, Paulson L C. Extending Sledgehammer with SMT solvers. *Journal of Automated Reasoning*, 2013, 51(1): 109–128. DOI: [10.1007/s10817-013-9278-5](https://doi.org/10.1007/s10817-013-9278-5).
- [23] Kapur D, Subramaniam M. Lemma discovery in automating induction. In *Lecture Notes in Computer Science 1104*, McRobbie M A, Slaney J K (eds.), Springer, 1996, pp.538–552. DOI: [10.1007/3-540-61511-3_112](https://doi.org/10.1007/3-540-61511-3_112).
- [24] Johansson M. Lemma discovery for induction: A survey. In *Lecture Notes in Computer Science 11617*, Kaliszyk C, Brady E, Kohlhasse A, Coen C S (eds.), Springer, 2019, pp.125–139. DOI: [10.1007/978-3-030-23250-4_9](https://doi.org/10.1007/978-3-030-23250-4_9).
- [25] Ireland A, Bundy A. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 1996, 16(1): 79–111. DOI: [10.1007/BF00244460](https://doi.org/10.1007/BF00244460).
- [26] Johansson M, Dixon L, Bundy A. Dynamic rippling, middle-out reasoning and lemma discovery. In *Lecture Notes in Computer Science 6463*, Siegler S, Wasser N (eds.), Springer, 2010, pp.102–116. DOI: [10.1007/978-3-642-17172-7_6](https://doi.org/10.1007/978-3-642-17172-7_6).
- [27] Johansson M, Dixon L, Bundy A. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 2011, 47(3): 251–289. DOI: [10.1007/s10817-010-9193-y](https://doi.org/10.1007/s10817-010-9193-y).
- [28] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser Ł, Polosukhin I. Attention is all you need. In *Proc. the 31st International Conference on Neural Information Processing Systems*, Dec. 2017, pp.6000–6010. DOI: [10.5555/3295222.3295349](https://doi.org/10.5555/3295222.3295349).
- [29] Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language models are unsupervised multitask learners. Technical Report, OpenAI, 2019. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- [30] Brown T B, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D M, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D. Language models are few-shot learners. In *Proc. the 34th International Conference on Neural Information Processing Systems*, Dec. 2020, Article No. 159. DOI: [10.5555/3495724.3495883](https://doi.org/10.5555/3495724.3495883).
- [31] OpenAI. GPT-4 technical report. arXiv: 2303.08774, 2023. <https://arxiv.org/abs/2303.08774>, Jan. 2026.
- [32] Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, Chi E H, Le Q V, Zhou D. Chain-of-thought prompting elicits reasoning in large language models. In *Proc. the 36th International Conference on Neural Information Processing Systems*, Nov. 28–Dec. 9, 2022, Article No. 1800. DOI: [10.5555/3600270.3602070](https://doi.org/10.5555/3600270.3602070).
- [33] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright C L, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A, Schulman J, Hilton J, Kelton F, Miller L, Simens M, Askell A, Welinder P, Christiano P, Leike J, Lowe R. Training language models to follow instructions with human feedback. In *Proc. the 36th International Conference on Neural Information Processing Systems*, Nov. 28–Dec. 9, 2022, Article No. 2011. DOI: [10.5555/3600270.3602281](https://doi.org/10.5555/3600270.3602281).
- [34] Chowdhery A, Narang S, Devlin J, Bosma M, Mishra G, Roberts A, Barham P, Chung H W, Sutton C, Gehrmann S, Schuh P, Shi K, Tsvyashchenko S, Maynez J, Rao A, Barnes P, Tay Y, Shazeer N, Prabhakaran V, Reif E, Du N, Hutchinson B, Pope R, Bradbury J, Austin J, Isard M, Gur-Ari G, Yin P, Duke T, Levskaya A, Ghemawat S, Dev S, Michalewski H, Garcia X, Misra V, Robinson K, Fedus L, Zhou D, Ippolito D, Luan D, Lim H, Zoph B, Spiridonov A, Sepassi R, Dohan D, Agrawal S, Omernick M, Dai A M, Pillai T S, Pellat M, Lewkowycz A, Moreira E, Child R, Polozov O, Lee K, Zhou Z, Wang X, Saeta B, Diaz M, Firat O, Catasta M, Wei J, Meier-Hellstern K, Eck D, Dean J, Petrov S, Fiedel N. PaLM: Scaling language modeling with pathways. *The Journal of Machine Learning Research*, 2023, 24(1): Article No. 240. DOI: [10.5555/3648699.3648939](https://doi.org/10.5555/3648699.3648939).
- [35] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux M A, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F, Rodriguez A, Joulin A, Grave E, Lample G. LLaMA:

- Open and efficient foundation language models. arXiv: 2302.13971, 2023. <https://arxiv.org/abs/2302.13971>, Jan. 2026.
- [36] Gemini Team Google. Gemini: A family of highly capable multimodal models. arXiv: 2312.11805, 2023. <https://arxiv.org/abs/2312.11805>, Jan. 2026.
- [37] DeepSeek-AI. DeepSeek-v2: A strong, economical, and efficient mixture-of-experts language model. arXiv: 2405.04434, 2024. <https://arxiv.org/abs/2405.04434>, Jan. 2026.
- [38] Bundy A, van Harmelen F, Horn C, Smaill A. The O^YS^TER-CL^AM system. In *Lecture Notes in Computer Science 449*, Stickel M E (ed.), Springer, 1990, pp.647–648. DOI: [10.1007/3-540-52885-7_123](https://doi.org/10.1007/3-540-52885-7_123).
- [39] Dixon L, Fleuriot J. IsaPlanner: A prototype proof planner in Isabelle. In *Lecture Notes in Computer Science 2741*, Baader Z (ed.), Springer, 2003, pp.279–283. DOI: [10.1007/978-3-540-45085-6_22](https://doi.org/10.1007/978-3-540-45085-6_22).
- [40] Sonnex W, Drossopoulou S, Eisenbach S. Zeno: An automated prover for properties of recursive data structures. In *Lecture Notes in Computer Science 7214*, Flanagan C, König Z (eds.), Springer, 2012, pp.407–421. DOI: [10.1007/978-3-642-28756-5_28](https://doi.org/10.1007/978-3-642-28756-5_28).
- [41] Nagashima Y, Parsert J. Goal-oriented conjecturing for Isabelle/HOL. In *Lecture Notes in Computer Science 11006*, Rabe F, Farmer W M, Passmore G O, Youssef A (eds.), Springer, 2018, pp.225–231. DOI: [10.1007/978-3-319-96812-4_19](https://doi.org/10.1007/978-3-319-96812-4_19).
- [42] Yang W, Fedyukovich G, Gupta A. Lemma synthesis for automating induction over algebraic data types. In *Lecture Notes in Computer Science 11802*, Schiex T, de Givry S (eds.), Springer, 2019, pp.600–617. DOI: [10.1007/978-3-030-30048-7_35](https://doi.org/10.1007/978-3-030-30048-7_35).
- [43] Sivaraman A, Sanchez-Stern A, Chen B, Lerner S, Millstein T. Data-driven lemma synthesis for interactive proofs. *Proceedings of the ACM on Programming Languages*, 2022, 6(OOPSLA2): 143. DOI: [10.1145/3563306](https://doi.org/10.1145/3563306).
- [44] Nagashima Y, Xu Z, Wang N, Goc D S, Bang J. Template-based conjecturing for automated induction in Isabelle/HOL. In *Lecture Notes in Computer Science 14155*, Hojjat H, Abraham E (eds.), Springer, 2023, pp.112–125. DOI: [10.1007/978-3-031-42441-0_9](https://doi.org/10.1007/978-3-031-42441-0_9).
- [45] Buchberger B, Crăciun A, Jebelean T, Kovács L, Kutsia T, Nakagawa K, Piroi F, Popov N, Robu J, Rosenkranz M, Windsteiger W. *Theorema*: Towards computer-aided mathematical theory exploration. *Journal of Applied Logic*, 2006, 4(4): 470–504. DOI: [10.1016/j.jal.2005.10.006](https://doi.org/10.1016/j.jal.2005.10.006).
- [46] Claessen K, Johansson M, Rosén D, Smallbone N. Automating inductive proofs using theory exploration. In *Lecture Notes in Computer Science 7898*, Bonacina M P (ed.), Springer, 2013, pp.392–406. DOI: [10.1007/978-3-642-38574-2_27](https://doi.org/10.1007/978-3-642-38574-2_27).
- [47] Johansson M, Smallbone N. Conjectures, tests and proofs: An overview of theory exploration. In *Proc. the 9th International Workshop on Verification and Program Transformation*, Mar. 2021, pp.1–16. DOI: [10.4204/EPTCS.341.1](https://doi.org/10.4204/EPTCS.341.1).
- [48] Zhang J. System description: MCS: Model-based conjecture searching. In *Lecture Notes in Computer Science 1632*, Ganzinger H (ed.), Springer, 1999, pp.393–397. DOI: [10.1007/3-540-48660-7_37](https://doi.org/10.1007/3-540-48660-7_37).
- [49] Colton S. The HR program for theorem generation. In *Lecture Notes in Computer Science 2392*, Voronkov A (ed.), Springer, 2002, pp.285–289. DOI: [10.1007/3-540-45620-1_24](https://doi.org/10.1007/3-540-45620-1_24).
- [50] Lenat D B. AM: An artificial intelligence approach to discovery in mathematics as heuristic search [Ph.D. Thesis]. Stanford University, Stanford, 1976.
- [51] Claessen K, Smallbone N, Hughes J. QUICKSPEC: Guessing formal specifications using testing. In *Lecture Notes in Computer Science 6143*, Fraser G, Gargantini A (eds.), Springer, 2010, pp.6–21. DOI: [10.1007/978-3-642-13977-2_3](https://doi.org/10.1007/978-3-642-13977-2_3).
- [52] Claessen K, Hughes J. QuickCheck: A lightweight tool for random testing of Haskell programs. In *Proc. the 5th ACM SIGPLAN International Conference on Functional Programming*, Sept. 2000, pp.268–279. DOI: [10.1145/351240.351266](https://doi.org/10.1145/351240.351266).
- [53] Smallbone N, Johansson M, Claessen K, Algehed M. Quick specifications for the busy programmer. *Journal of Functional Programming*, 2017, 27:e18. DOI: [10.1017/S0956796817000090](https://doi.org/10.1017/S0956796817000090).
- [54] Einarssdóttir S H, Smallbone N, Johansson M. Template-based theory exploration: Discovering properties of functional programs by testing. In *Proc. the 32nd Symposium on Implementation and Application of Functional Languages*, Sept. 2020, pp.67–78. DOI: [10.1145/3462172.3462192](https://doi.org/10.1145/3462172.3462192).
- [55] Rosén D. Proving equational Haskell properties using automated theorem provers. [Master’s Thesis], University of Gothenburg, Göteborg, 2012.
- [56] Claessen K, Johansson M, Rosén D, Smallbone N. TIP: Tons of inductive problems. In *Lecture Notes in Computer Science 9150*, Kerber M, Carette J, Kaliszyk C, Rabe F, Sorge V (eds.), Springer, 2015, pp.333–337. DOI: [10.1007/978-3-319-20615-8_23](https://doi.org/10.1007/978-3-319-20615-8_23).
- [57] Johansson M, Rosén D, Smallbone N, Claessen K. Hipster: Integrating theory exploration in a proof assistant. In *Lecture Notes in Computer Science 8543*, Watt S M, Davenport J H, Sexton A P, Sojka P, Urban J (eds.), Springer, 2014, pp.108–122. DOI: [10.1007/978-3-319-08434-3_9](https://doi.org/10.1007/978-3-319-08434-3_9).
- [58] Gissurarson M P, Roque D, Koppel J. Spectacular: Finding laws from 25 trillion terms. In *Proc. the 2023 IEEE Conference on Software Testing, Verification and Validation*, Apr. 2023, pp.293–304. DOI: [10.1109/ICST57152.2023.00035](https://doi.org/10.1109/ICST57152.2023.00035).
- [59] Singher E, Itzhaky S. Theory exploration powered by deductive synthesis. In *Lecture Notes in Computer Science 12760*, Silva A, Leino K R M (eds.), Springer, 2021, pp.125–148. DOI: [10.1007/978-3-030-81688-9_6](https://doi.org/10.1007/978-3-030-81688-9_6).
- [60] Montano-Rivas O, McCasland R, Dixon L, Bundy A. Scheme-based theorem discovery and concept invention. *Expert Systems with Applications*, 2012, 39(2):

- 1637–1646. DOI: [10.1016/j.eswa.2011.06.055](https://doi.org/10.1016/j.eswa.2011.06.055).
- [61] Einarsdóttir S H, Johansson M, Smallbone N. LOL: A library of lemma templates for data-driven conjecturing. In *Proc. the 15th Conference on Intelligent Computer Mathematics (CICM 2022) Informal Proceedings*, Aug. 2022, Article No. 22.
- [62] Puzis Y, Gao Y, Sutcliffe G. Automated generation of interesting theorems. In *Proc. the 19th International Florida Artificial Intelligence Research Society Conference*, May 2006, pp.49–54.
- [63] Sutcliffe G, Suttner C. The TPTP problem library. *Journal of Automated Reasoning*, 1998, 21(2): 177–203. DOI: [10.1023/A:1005806324129](https://doi.org/10.1023/A:1005806324129).
- [64] McCasland R L, Bundy A. MATHSAiD: A mathematical theorem discovery tool. In *Proc. the 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Sept. 2006, pp.17–22. DOI: [10.1109/SYNASC.2006.51](https://doi.org/10.1109/SYNASC.2006.51).
- [65] McCasland R L, Bundy A, Smith P F. MATHSAiD: Automated mathematical theory exploration. *Applied Intelligence*, 2017, 47(3): 585–606. DOI: [10.1007/S10489-017-0954-8](https://doi.org/10.1007/S10489-017-0954-8).
- [66] Gao H, Goto Y, Cheng J. A systematic methodology for automated theorem finding. *Theoretical Computer Science*, 2014, 554:2–21. DOI: [10.1016/J.TCS.2014.06.028](https://doi.org/10.1016/J.TCS.2014.06.028).
- [67] Cheng J. A strong relevant logic model of epistemic processes in scientific discovery. In *Lecture Notes in Computer Science 1574*, Zhong N, Zhou L (eds.), Springer, 1999, pp.489–493. DOI: [10.1007/3-540-48912-6_65](https://doi.org/10.1007/3-540-48912-6_65).
- [68] Lenat D B. EURISKO: A program that learns new heuristics and domain concepts: The nature of heuristics III: Program design and results. *Artificial Intelligence*, 1983, 21(1/2): 61–98. DOI: [10.1016/S0004-3702\(83\)80005-8](https://doi.org/10.1016/S0004-3702(83)80005-8).
- [69] Tang P, Lin F. Discovering theorems in game theory: Two-person games with unique pure Nash equilibrium payoffs. *Artificial Intelligence*, 2011, 175(14/15): 2010–2020. DOI: [10.1016/J.ARTINT.2011.07.001](https://doi.org/10.1016/J.ARTINT.2011.07.001).
- [70] Epstein S L. Learning and discovery: One system’s search for mathematical knowledge. *Computational Intelligence*, 1988, 4(1): 42–53. DOI: [10.1111/J.1467-8640.1988.TB00089.X](https://doi.org/10.1111/J.1467-8640.1988.TB00089.X).
- [71] Fajtlowicz S. On conjectures of Graffiti. *Discrete Mathematics*, 1988, 72(1/2/3): 113–118. DOI: [10.1016/0012-365X\(88\)90199-9](https://doi.org/10.1016/0012-365X(88)90199-9).
- [72] Davila R. Automated conjecturing in mathematics with TxGraffiti. arXiv: 2409.19379, 2024. <https://arxiv.org/abs/2409.19379>, Jan. 2026.
- [73] Quaresma P, Graziani P, Nicoletti S M. Considerations on approaches and metrics in automated theorem generation/finding in geometry. In *Proc. the 14th International Conference on Automated Deduction in Geometry*, Sept. 2023, pp.85–100. DOI: [10.4204/EPTCS.398.12](https://doi.org/10.4204/EPTCS.398.12).
- [74] Wu W T. Mathematics Mechanization: Mechanical Geometry Theorem-Proving, Mechanical Geometry Problem-Solving, and Polynomial Equations-Solving. Science Press, Kluwer Academic Publishers, 2000.
- [75] Wu W T. A mechanization method of geometry and its applications: I. Distances, areas and volumes. In *Select-ed Works of Wen-Tsun Wu*, Wu W T (ed.), World Scientific, pp.259–271, 2008. DOI: [10.1142/9789812791085_0016](https://doi.org/10.1142/9789812791085_0016).
- [76] Bagai R, Shanbhogue V, Żytkow J M, Chou S C. Automatic theorem generation in plane geometry. In *Lecture Notes in Computer Science 689*, Komorowski J, Raś Z W (eds.), Springer, 1993, pp.415–424. DOI: [10.1007/3-540-56804-2_39](https://doi.org/10.1007/3-540-56804-2_39).
- [77] Kovács Z, Recio T, Vélez M P. Automated reasoning tools in GeoGebra discovery. *ACM Communications in Computer Algebra*, 2021, 55(2): 39–43. DOI: [10.1145/3493492.3493495](https://doi.org/10.1145/3493492.3493495).
- [78] Kovács Z, Parisse B. Giac and GeoGebra-improved Gröbner basis computations. In *Computer Algebra and Polynomials: Applications of Algebra and Number Theory*, Gutierrez J, Schicho J, Weimann M (eds.), Springer, 2015, pp.126–138. DOI: [10.1007/978-3-319-15081-9_7](https://doi.org/10.1007/978-3-319-15081-9_7).
- [79] Chou S C, Gao X S, Zhang J Z. A deductive database approach to automated geometry theorem proving and discovering. *Journal of Automated Reasoning*, 2000, 25(3): 219–246. DOI: [10.1023/A:1006171315513](https://doi.org/10.1023/A:1006171315513).
- [80] Peng X, Chen Q, Zhang J, Chen M. Automated discovery of geometric theorems based on vector equations. *Journal of Automated Reasoning*, 2021, 65(6): 711–726. DOI: [10.1007/S10817-021-09591-2](https://doi.org/10.1007/S10817-021-09591-2).
- [81] Langley P. Data-driven discovery of physical laws. *Cognitive Science*, 1981, 5(1): 31–54. DOI: [10.1111/J.1551-6708.1981.TB00869.X](https://doi.org/10.1111/J.1551-6708.1981.TB00869.X).
- [82] Caporossi G, Hansen P. Variable neighborhood search for extremal graphs: 1 The AutoGraphX system. *Discrete Mathematics*, 2000, 212(1/2): 29–44. DOI: [10.1016/S0012-365X\(99\)00206-X](https://doi.org/10.1016/S0012-365X(99)00206-X).
- [83] Raayoni G, Gottlieb S, Manor Y, Pisha G, Harris Y, Mendlovic U, Haviv D, Hadad Y, Kammer I. Generating conjectures on fundamental constants with the Ramanujan machine. *Nature*, 2021, 590(7844): 67–73. DOI: [10.1038/S41586-021-03229-4](https://doi.org/10.1038/S41586-021-03229-4).
- [84] Udrescu S M, Tegmark M. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 2020, 6(16): eaay2631. DOI: [10.1126/sciadv.aay2631](https://doi.org/10.1126/sciadv.aay2631).
- [85] Caporossi G, Hansen P. Variable neighborhood search for extremal graphs. 5. Three ways to automate finding conjectures. *Discrete Mathematics*, 2004, 276(1/2/3): 81–94. DOI: [10.1016/S0012-365X\(03\)00311-X](https://doi.org/10.1016/S0012-365X(03)00311-X).
- [86] Davies A, Veličković P, Buesing L, Blackwell S, Zheng D, Tomašev N, Tanburn R, Battaglia P, Blundell C, Juhász A, Lackenby M, Williamson G, Hassabis D, Kohli P. Advancing mathematics by guiding human intuition with AI. *Nature*, 2021, 600(7887): 70–74. DOI: [10.1038/S41586-021-04086-X](https://doi.org/10.1038/S41586-021-04086-X).
- [87] Heras J, Komendantskaya E, Johansson M, Maclean E. Proof-pattern recognition and lemma discovery in ACL2. In *Lecture Notes in Computer Science 8312*, McMillan K, Middeldorp A, Voronkov A (eds.), Springer, 2013, pp.389–406. DOI: [10.1007/978-3-642-45221-5_27](https://doi.org/10.1007/978-3-642-45221-5_27).

- [88] Heras J, Komendantskaya E. ACL2(ml): Machine-learning for ACL2. In *Proc. the 12th International Workshop on the ACL2 Theorem Prover and Its Applications*, Jul. 2014, pp.61–75. DOI: [10.4204/EPTCS.152.5](https://doi.org/10.4204/EPTCS.152.5).
- [89] Gauthier T, Kaliszyk C, Urban J. Initial experiments with statistical conjecturing over large formal corpora. In *Joint Proc. the 2016 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*, Jul. 2016, pp.219–228.
- [90] Gauthier T, Kaliszyk C. Matching concepts across HOL libraries. In *Lecture Notes in Computer Science 8543*, Watt S M, Davenport J H, Sexton A P, Sojka P, Urban J (eds.), Springer, 2014, pp.267–281. DOI: [10.1007/978-3-319-08434-3_20](https://doi.org/10.1007/978-3-319-08434-3_20).
- [91] Urban J, Jakubův J. First neural conjecturing datasets and experiments. In *Lecture Notes in Computer Science 12236*, Benz Müller C, Miller B (eds.), Springer, 2020, pp.315–323. DOI: [10.1007/978-3-030-53518-6_24](https://doi.org/10.1007/978-3-030-53518-6_24).
- [92] Rabe M N, Lee D, Bansal K, Szegedy C. Mathematical reasoning via self-supervised skip-tree training. In *Proc. the 9th International Conference on Learning Representations*, May 2021.
- [93] Bansal K, Loos S, Rabe M, Szegedy C, Wilcox S. HOList: An environment for machine learning of higher order logic theorem proving. In *Proc. the 36th International Conference on Machine Learning*, Jun. 2019, pp.454–463.
- [94] Johansson M, Smallbone N. Exploring mathematical conjecturing with large language models. In *Proc. the 17th International Workshop on Neural-Symbolic Learning and Reasoning*, Jul. 2023, pp.62–77.
- [95] Poesia G, Broman D, Haber N, Goodman N D. Learning formal mathematics from intrinsic motivation. In *Proc. the 38th International Conference on Neural Information Processing Systems*, Dec. 2024, Article No. 1362. DOI: [10.52202/079017-1362](https://doi.org/10.52202/079017-1362).
- [96] Alhessi Y, Einarsdóttir S H, Granberry G, First E, Johansson M, Lerner S, Smallbone N. Lemmanaid: Neuro-symbolic lemma conjecturing. arXiv: 2504.04942, 2025. <https://arxiv.org/abs/2504.04942>, Jan. 2026.
- [97] Wang H, Xin H, Zheng C, Liu Z, Cao Q, Huang Y, Xiong J, Shi H, Xie E, Yin J, Li Z, Liang X. LEGO-prover: Neural theorem proving with growing libraries. In *Proc. the 12th International Conference on Learning Representations*, May 2024, pp.20212–20243.
- [98] Jiang A Q, Welleck S, Zhou J P, Li W, Liu J, Jamnik M, Lacroix T, Lample G, Wu Y. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *Proc. the 11th International Conference on Learning Representations*, May 2023, pp.1–19.
- [99] Cao C, Song L, Li Z, Le X, Zhang X, Xue H, Yang F. Reviving DSP for advanced theorem proving in the era of reasoning models. arXiv: 2506.11487, 2025. <https://arxiv.org/abs/2506.11487>, Jan. 2026.
- [100] Wang H, Xin H, Liu Z, Li W, Huang Y, Lu J, Yang Z, Tang J, Yin J, Li Z, Liang X. Proving theorems recursively. In *Proc. the 38th International Conference on Neural Information Processing Systems*, Dec. 2024, Article No. 2753. DOI: [10.52202/079017-2753](https://doi.org/10.52202/079017-2753).
- [101] Xin R, Zheng Z, Nie Y, Yuan K, Xiao X. Scaling up multi-turn off-policy RL and multi-agent tree search for LLM step-provers. arXiv: 2509.06493, 2025. <https://arxiv.org/abs/2509.06493>, Jan. 2026.
- [102] Ren Z Z, Shao Z, Song J, Xin H, Wang H, Zhao W, Zhang L, Fu Z, Zhu Q, Yang D, Wu Z F, Gou Z, Ma S, Tang H, Liu Y, Gao W, Guo D, Ruan C. DeepSeek-prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. arXiv: 2504.21801, 2025. <https://arxiv.org/abs/2504.21801>, Jan. 2026.
- [103] Zhou Y, Zhao J, Zhang Y, Wang B, Wang S, Chen L, Wang J, Chen H, Jie A, Zhang X, Wang H, Trung L, Ye R, Hoang P N, Zhang H, Sun P, Li H. Solving formal math problems by decomposition and iterative reflection. arXiv: 2507.15225, 2025. <https://arxiv.org/abs/2504.21801>, Jan. 2026.
- [104] Chen L, Gu J, Huang L, Huang W, Jiang Z, Jie A, Jin X, Jin X, Li C, Ma K, Ren C, Shen J, Shi W, Sun T, Sun H, Wang J, Wang S, Wang Z, Wei C, Wei S, Wu Y, Wu Y, Xia Y, Xin J, Yang F, Ying H, Yuan H, Yuan Z, Zhan T, Zhang C, Zhang Y, Zhang G, Zhao T, Zhao J, Zhou Y, Zhu T H. Seed-prover: Deep and broad reasoning for automated theorem proving. arXiv: 2507.23726, 2025. <https://arxiv.org/abs/2507.23726>, Jan. 2026.
- [105] Wang H, Unsal M, Lin X, Baksys M, Liu J, Dos Santos M, Sung F, Vinyes M, Ying Z, Zhu Z, Lu J, de Saxcé H, Bailey B, Song C, Xiao C, Zhang D, Zhang E, Pu F, Zhu H, Liu J, Bayer J, Michel J, Yu L, Dreyfus-Schmidt L, Tunstall L, Pagani L, Machado M, Bourigault P, Wang R, Polu S, Barroyer T, Li W D, Niu Y, Fleureau Y, Hu Y, Yu Z, Wang Z, Yang Z, Liu Z, Li J. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. arXiv: 2504.11354, 2025. <https://arxiv.org/abs/2504.11354>, Jan. 2026.
- [106] Shang S, Wan R, Peng Y, Wu Y, Chen X H, Yan J, Zhang X. StepFun-prover preview: Let's think and verify step by step. arXiv: 2507.20199, 2025. <https://arxiv.org/abs/2507.20199>, Jan. 2026.
- [107] Lin Y, Tang S, Lyu B, Yang Z, Chung J H, Zhao H, Jiang L, Geng Y, Ge J, Sun J, Wu J, Gesi J, Lu X, Acuna D, Yang K, Lin H, Choi Y, Chen D, Arora S, Jin C. Goedel-prover-V2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. arXiv: 2508.03613, 2025. <https://arxiv.org/abs/2508.03613>, Jan. 2026.
- [108] Firoiu V, Aygun E, Anand A, Ahmed Z, Glorot X, Orseau L, Zhang L, Precup D, Mourad S. Training a first-order theorem prover from synthetic data. arXiv: 2103.03798, 2021. <https://arxiv.org/abs/2103.03798>, Jan. 2026.
- [109] Wang M, Deng J. Learning to prove theorems by learn-

- ing to generate theorems. In *Proc. the 34th International Conference on Neural Information Processing Systems*, Dec. 2020, Article No. 1523. DOI: [10.5555/3495724.3497247](https://doi.org/10.5555/3495724.3497247).
- [110] Wu Y, Jiang A Q, Ba J, Grosse R. INT: An inequality benchmark for evaluating generalization in theorem proving. In *Proc. the 9th International Conference on Learning Representations*, May 2021.
- [111] Trinh T H, Wu Y, Le Q V, He H, Luong T. Solving olympiad geometry without human demonstrations. *Nature*, 2024, 625(7995): 476–482. DOI: [10.1038/S41586-023-06747-5](https://doi.org/10.1038/S41586-023-06747-5).
- [112] Wei C, Sun M, Wang W. Proving Olympiad algebraic inequalities without human demonstrations. In *Proc. the 38th International Conference on Neural Information Processing Systems*, Dec. 2024, Article No. 2633. DOI: [10.5555/3737916.3740549](https://doi.org/10.5555/3737916.3740549).
- [113] Wu S, Lu S, Gong Y, Duan N, Wei P. Alchemy: Amplifying theorem-proving capability through symbolic mutation. In *Proc. the 13th International Conference on Learning Representations*, Apr. 2025.
- [114] Onda N, Kasaura K, Oriike Y, Taniguchi M, Sannai A, Sonoda S. LeanConjecturer: Automatic generation of mathematical conjectures for theorem proving. arXiv: 2506.22005, 2025. <https://arxiv.org/abs/2506.22005>, Jan. 2026.
- [115] Dong K, Ma T. STP: Self-play LLM theorem provers with iterative conjecturing and proving. In *Proc. the 42nd International Conference on Machine Learning*, Jul. 2025, pp.14114–14136.
- [116] Lin X, Cao Q, Huang Y, Yang Z, Liu Z, Li Z, Liang X. ATG: Benchmarking automated theorem generation for generative language models. In *Proc. the 2024 Findings of the Association for Computational Linguistics: NAACL*, Jun. 2024, pp.4465–4480. DOI: [10.18653/V1/2024.FINDINGS-NAACL.279](https://doi.org/10.18653/V1/2024.FINDINGS-NAACL.279).
- [117] Zhou J P, Wu Y, Li Q, Grosse R. REFACTOR: Learning to extract theorems from proofs. In *Proc. the 12th International Conference on Learning Representations*, May 2024, pp.51382–51402. https://proceedings.iclr.cc/paper_files/paper/2024/file/b21ae5a5df83632324b61b595ab653b9-Paper-Conference.pdf
- [118] Colton S, Bundy A, Walsh T. On the notion of interestingness in automated mathematical discovery. *International Journal of Human-Computer Studies*, 2000, 53(3): 351–375. DOI: [10.1006/IJHC.2000.0394](https://doi.org/10.1006/IJHC.2000.0394).
- [119] Berghofer S, Nipkow T. Random testing in Isabelle/HOL. In *Proc. the 2nd International Conference on Software Engineering and Formal Methods*, Sept. 2004, pp.230–239. DOI: [10.1109/SEFM.2004.10049](https://doi.org/10.1109/SEFM.2004.10049).
- [120] Larson C E, Van Cleemput N. Automated conjecturing I: Fajtlowicz’s Dalmatian heuristic revisited. *Artificial Intelligence*, 2016, 231:17–38. DOI: [10.1016/j.artint.2015.10.002](https://doi.org/10.1016/j.artint.2015.10.002).
- [121] Gao H, Li J, Cheng J. Measuring interestingness of theorems in automated theorem finding by forward reasoning based on strong relevant logic. In *Proc. the 2019 IEEE International Conference on Energy Internet*, May 2019, pp.356–361. Doi: [10.1109/ICEI.2019.00069](https://doi.org/10.1109/ICEI.2019.00069).
- [122] Zheng K, Han J M, Polu S. miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. In *Proc. the 10th International Conference on Learning Representations*, Apr. 2022, pp.5223–5233.
- [123] Azerbayev Z, Piotrowski B, Schoelkopf H, Ayers E W, Radev D, Avigad J. ProofNet: Autoformalizing and formally proving undergraduate-level mathematics. arXiv: 2302.12433, 2023. <https://arxiv.org/abs/2302.12433>, Jan. 2026.
- [124] Tsoukalas G, Lee J, Jennings J, Xin J, Ding M, Jennings M, Thakur A, Chaudhuri S. PUTNAMBENCH: Evaluating neural theorem-provers on the Putnam mathematical competition. In *Proc. the 38th International Conference on Neural Information Processing Systems*, Dec. 2024, Article No. 368. DOI: [10.52202/079017-0368](https://doi.org/10.52202/079017-0368).
- [125] Einarsdóttir S H, Hajdu M, Johansson M, Smallbone N, Suda M. Lemma discovery and strategies for automated induction. In *Lecture Notes in Computer Science 14739*, Benz Müller C, Heule M J H, Schmidt R A (eds.), Springer, 2024, pp.214–232. DOI: [10.1007/978-3-031-63498-7_13](https://doi.org/10.1007/978-3-031-63498-7_13).
- [126] McGill N, Wheeler D A. Metamath: A Computer Language for Mathematical Proofs. Lulu.com, 2019.
- [127] Yang K, Deng J. Learning to prove theorems via interacting with proof assistants. In *Proc. the 36th International Conference on Machine Learning*, Jun. 2019, pp.6984–6994.
- [128] Jiang A Q, Li W, Han J M, Wu Y. LISA: Language models of Isabelle proofs. In *Proc. the 6th Conference on Artificial Intelligence and Theorem Proving*, Sept. 2021, pp.378–392.
- [129] Lin X, Cao Q, Huang Y, Wang H, Lu J, Liu Z, Song L, Liang X. FVEL: Interactive formal verification environment with large language models via theorem proving. In *Proc. the 38th International Conference on Neural Information Processing Systems*, Dec. 2024, Article No. 1742. DOI: [10.52202/079017-1742](https://doi.org/10.52202/079017-1742).
- [130] Yang K, Swope A M, Gu A, Chalamala R, Song P, Yu S, Godil S, Prenger R, Anandkumar A. LeanDojo: Theorem proving with retrieval-augmented language models. In *Proc. the 37th International Conference on Neural Information Processing Systems*, Dec. 2023, Article No. 944. DOI: [10.5555/3666122.3667066](https://doi.org/10.5555/3666122.3667066).
- [131] Wu Z, Wang J, Lin D, Chen K. LEAN-GitHub: Compiling GitHub Lean repositories for a versatile lean prover. arXiv: 2407.17227, 2024. <https://arxiv.org/abs/2407.17227>, Jan. 2026.
- [132] Li J, Beeching E, Tunstall L, Lipkin B, Soletskyi R, Huang S C, Rasul K, Yu L, Jiang A, Shen Z, Qin Z, Dong B, Zhou L, Fleureau Y, Lample G, Polu S. NuminaMath. 2024. <https://github.com/project-numina/aimo-progress-prize>, January 2026.
- [133] Reddy C K, Shojaei P. Towards scientific discovery with generative AI: Progress, opportunities, and challenges. In *Proc. the 39th AAAI Conference on Artificial Intelligence*, Feb. 25–Mar. 4, 2025, pp.28601–28609. DOI: [10.1609/AAAI.V39I27.35084](https://doi.org/10.1609/AAAI.V39I27.35084).



Jian Zhang is currently a research professor with the Institute of Software, Chinese Academy of Sciences (ISCAS), Beijing. He received his B.S. degree in software from University of Science and Technology of China (USTC), Hefei, in 1988, and his Ph.D. degree from ISCAS, Beijing, in 1994. His research interests include automated reasoning and constraint solving, software testing and static analysis.



Si-Cheng Tan received his B.S. degree in software engineering from Nanjing University, Nanjing, in 2023. He is currently pursuing his Ph.D. degree in computer science and technology at the University of Chinese Academy of Sciences (UCAS), Beijing, affiliated with the Institute of Software, Chinese Academy of Sciences, Beijing. His research interests include automated reasoning and constraint solving.