Automated Reasoning for Mathematics

Jeremy Avigad[0000−0003−1275−315X]

Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract. Throughout the history of automated reasoning, mathematics has been viewed as a prototypical domain of application. It is therefore surprising that the technology has had almost no impact on mathematics to date and plays almost no role in the subject today. This article presents an optimistic view that the situation is about to change. It describes some recent developments in the Lean programming language and proof assistant that support this optimism, and it reflects on the role that automated reasoning can and should play in mathematics in the years to come.

1 The Origins and Foundations of Automated Reasoning

The fact that IJCAR is celebrating the 30th anniversary of the CADE ATP System Competition (CASC) is a reminder that, at least by the standards of computer science, automated reasoning has a long and venerable history. Some date the field to 1956, when Allen Newell, Herbert Simon, and Cliff Shaw introduced the *Logic Theorist*, a program that used heuristic methods to prove theorems in propositional logic. Two years earlier, however, Martin Davis had implemented Presburger's decision procedure for integer arithmetic on a computer at the Institute for Advanced Study. Davis admitted that the program did not perform well but he reported that it succeeded in proving that the sum of two even numbers is even. In 1960, Henry Gelernter, J. R. Hansen, and Donald Loveland published an article on the *Geometry Machine*, a program that could prove nontrivial theorems in elementary Euclidean plane geometry. The resolution rule for propositional logic was introduced by Davis and Hilary Putnam in 1960, and John Alan Robinson's introduction of a unification algorithm in 1965 established resolution theorem proving as a powerful method for first-order logic.[1]

The theoretical foundations of automated reasoning predate even the introduction of the first electronic computers. In contemporary terms, Kurt Gödel's first incompleteness theorem says that there is no complete, consistent, computably axiomatized theory that contains (or interprets) a modicum of arithmetic. In his 1931 paper, Gödel explained that the theorem, as he stated it,

> is not in any way due to the special nature of the systems that have been set up, but holds for a wide class of formal systems; among these,

[1] All of the articles mentioned in this paragraph are found in a collection edited by Siegmann and Wrightson [50]. See also the survey by Mackenzie [36] and the references there.

> in particular, are all systems that result from the two just mentioned through the addition of a finite number of axioms ... [21,22]

It is interesting to see Gödel struggling to say "computably axiomatized theory" for the simple reason that, at the time, there was no mathematical concept of computability available. He then presented a tentative definition of computability in lectures that he gave at the Institute for Advanced Study in Princeton in 1932 precisely so that he could state the incompleteness theorems in their proper generality. Alan Turing gave his own celebrated definition of computability a few years later and titled the paper "On computable numbers, with an application to the Entscheidungsproblem," providing a negative answer to Hilbert's question as to whether there is a decision procedure for first-order logic. Gödel had expressed uncertainty as to whether his definition exhausts the general notion of computability, but he took Turing's analysis to settle the matter definitively:

> In consequence of later advances, in particular of the fact that, due to A. M. Turing's work, a precise and unquestionably adequate definition of the general concept of a formal system can now be given, the existence of undecidable arithmetical propositions and the non-demonstrability of the consistency of a system in the same system can now be proved rigorously for *every* consistent formal system containing a certain amount of finitary number theory. (quoted in [17])

The emphasis is by Gödel, who took the phrase "formal system" to mean a system with computably checkable axioms and rules.

Turing pointed out in his paper that the first incompleteness theorem is a consequence of the undecidability of theories of arithmetic, because if there were a complete, computably axiomatized theory of arithmetic one could decide the provability of a formula by simultaneously searching for a proof of the formula and its negation. Alonzo Church gave an independent proof of the undecidability of arithmetic in 1936, and Stephen Kleene, another key player in developing the modern theory of computability, was also keenly interested in applications to logic and the foundations of mathematics. So logicians were thinking about computable proof systems and proof search even before the arrival of the first digital electronic computers in the 1940s.

Fundamental decision procedures for logic and arithmetic also predate the electronic computer. As early as 1915, Leopold Löwenheim gave a decision procedure for monadic first-order logic. (The introduction to Börger et al. [11] provides an excellent overview of early work on the decidability of fragments of first-order reasoning.) Mojżesz Presburger's paper on a decision procedure for arithmetic, based on Thoralf Skolem's method of eliminating quantifiers, was published in 1929. (Presburger never earned a doctorate for that work; Andrzej Mostowski reported [15] that Alfred Tarski thought the result was too simple, a straightforward application of Thoralf Skolem's method of elimination of quantifiers.) In 1930, Tarski obtained a decision procedure for real-closed fields, that is, the first-order theory of the real numbers as an ordered field, though the result was not published until 1948. So logicians were also interested in decision procedures for aspects of mathematical reasoning even before there were computers to implement them.

2 Taking Stock

We have seen that the early history of automated reasoning was rooted in the foundations of mathematics and that many of its early pioneers were mathematicians. Even those who were not mathematicians took mathematical reasoning to be a primary target of the technology. Now, almost a century after Presburger's discovery of a decision procedure for arithmetic and almost three quarters of a century after the implementation of the Logic Theorist, it seems reasonable to ask where we stand. What has automated reasoning done for mathematics, and how is it used in mathematics today?

The answer is disappointingly negative. Automated reasoning has had almost no impact at all on mathematics and plays almost no role in the subject today. Few working mathematicians have ever touched an automated reasoning tool, let alone use automated reasoning in their daily work. The technology has contributed to very few mathematical discoveries, even minor ones.

This is surprising. One would think, as the pioneers of the subject clearly did, that mathematical reasoning is ideally suited to automation. To be sure, mathematics requires creativity, intuition, experience, and insight, but it also requires long chains of precise and sometimes tedious reasoning, and it's hard to get the details right. Computers can carry out small inferential steps much more quickly and accurately than we can, so we would expect them to be helpful for exploring and verifying mathematical results. Numeric and symbolic computation have revolutionized the sciences, even though science involves a lot more than computation. Why hasn't automated reasoning had a similar impact on mathematics?

This question is not meant as a criticism. Automated reasoning has made remarkable progress over the past 70 years, and the tools are now quite sophisticated. They have had a significant impact in several important areas, such as hardware and software verification, AI, planning, databases, knowledge representation, program synthesis, and natural language processing. Automated reasoning does not have to look to mathematics for justification. Moreover, the fact that there have been few applications to mathematics doesn't mean that there haven't been any, and the successes are worth celebrating. Finally, the fact that it has been difficult to automate mathematical reasoning is largely a reflection of the fact that mathematics isn't easy to mechanize, and those of us who love the subject wouldn't want it any other way. So my goal here is rather to review some of the successes of automated reasoning for mathematics, understand the challenges, and reflect on the role that automated reasoning can and should play in mathematics in the years to come.

In 2019, I gave a joint talk at FroCoS and TABLEAUX titled "Automated Reasoning for the Working Mathematician." In that talk, I surveyed the use of automated reasoning with interactive proof assistants in the hopes of extracting lessons that I could convey to the automated reasoning community. This article draws on that talk as well as unpublished notes, data, and experiments that I prepared at the time.[2] See also my article, “The Mechanization of Mathematics” [1], and an article by Michael Beeson with the same title [6], for additional examples of applications of automated reasoning to mathematics.

3 A Personal History

As a mathematician who has been using automated reasoning tools for more than two decades, my interest in the subject is personal. I first experimented with Isabelle and Coq in the late 1990s, and when I started using Isabelle in earnest in 2002, the automation was surprisingly mature. There were a conditional term rewriter, `simp` [44], variations on a general tableau prover (`auto`, `force`, and `clarify` [45]), and a decision procedure for linear arithmetic (`arith`). Working with students at Carnegie Mellon, I completed a proof of the Hadamard–de la Vallée-Poussin prime number theorem in September of 2004 [2]. A couple of months later, Georges Gonthier announced the verification of the four-color theorem in Coq [24], and soon after that Thomas Hales announced the verification of the Jordan curve theorem in HOL Light [26]. These were early landmarks, providing evidence that substantial mathematical theorems could be formalized.

Many of the challenges in formalizing the prime number theorem stemmed from the fact that Isabelle's libraries were young and incomplete. The automation, however, was remarkably helpful. For example, in the 4,000 lines contained in the last file in the proof, there are 390 invocations of `simp`, 397 invocations of `auto` and friends, and 246 invocations of `arith`. Even now, twenty years later, I have yet to have a better experience with automation.

I spent a sabbatical year in France with Gonthier and his team in 2009-2010, working on the formalization of the Feit–Thompson theorem, using the SSReflect proof language and Coq [25]. In designing and managing the project, Georges made the conscious decision to avoid automation entirely, other than the built-in foundational reduction of Coq expressions, which is fundamental to the methodology of SSReflect. He was skeptical that black-box automation would scale and had more faith in the power of good language design to make formalization manageable.

When I returned from France, I was ready to leave interactive theorem proving behind and turn to automated reasoning. But a talented undergraduate at Carnegie Mellon, Luke Serafin, managed to convince me to work on a project to verify the central limit theorem in Isabelle [3], and I was seduced by the excitement around homotopy type theory at the time to work on another verification project in Coq [4].

What really pulled me back to the world of proof assistants, however, was Leonardo de Moura's decision, in 2013, to launch the Lean project. Leo convinced me that even if one is primarily interested in automation for mathematics, one

[2] https://github.com/avigad/arwm

should build it on top of a secure, expressive foundation, not just to ensure that the automation is reliable, but to have a meaningful specification of what the results mean. For several years, Lean's web pages described the aim of the project as follows:

> to bridge the gap between interactive and automated theorem proving, by situating automated tools and methods in a framework that supports user interaction and the construction of fully specified axiomatic proofs.

I don't think Leo anticipated the amount of work he would have to put into implementing an elaborator for dependent type theory and supporting all the features that are needed to make that foundation usable. Work also went into the implementation of a tactic framework, in Lean 2, and the implementation of a metaprogramming language, in Lean 3, that users could use to write their own tactics [18,39]. Lean 4 is a complete rewrite of the system, most of which is now implemented in Lean 4 itself [38]. Leo and Sebastian Ullrich have put a lot of effort into making Lean 4 an efficient programming (and metaprogramming) language, and treating syntax as first-class objects, making the syntactic framework as powerful, flexible, and extensible as the tactic framework.

We are now beginning to see automation for Lean 4 that is written in Lean 4, as well as Lean-based experiments on applications of machine learning to mathematical reasoning. Thus, a decade into the Lean project, we are now in an especially good position to realize the initial vision of making it a powerful means of combining automation with user interaction. In the sections that follow, I will discuss prospects for automated reasoning for mathematics in general, but I will also focus specifically on opportunities based on recent developments in Lean.

4 Domain-General Reasoning for Verification

To prepare for the talk at FroCoS and TABLEAUX, I sent a questionnaire to colleagues who had worked on formalization of mathematics to learn about their experiences with automation. One of the interesting findings was that most of the people who I considered to be the best at formalization—people who had formalized vast amounts of interesting mathematics efficiently and with very high quality—used very little automation at all. (Larry Paulson was a notable exception; he has spoken eloquently of the power of automation in enabling him to port large amounts of measure theory and analysis from HOL Light to Isabelle.) The best explanation I could come up with is that even if automation were much better than it is now, serious users would still have to fill in some inferences by hand, which would inevitably require them to learn the library inside out and become skillful at writing explicit proofs. So even when automation is available, power users generally come to know the library and proof language well enough that they don't need to use automation to do what they want to do. If that analysis is correct, it highlights the challenge of scaling the use of formal methods to a broader mathematical audience. Even now, I get frustrated when I have to struggle with an unfamiliar part of the library to carry out an inference that seems painfully obvious. The lack of automation limits the utility of formalization to all but the most determined and dedicated practitioners.

Let me clarify that when I talk about domain-general reasoning, I am setting aside equational rewriting and simplification. It's not clear how to classify such methods. On the one hand, there is nothing more general than the equality relation: wherever there are expressions that denote objects, there are equations that govern them. On the other hand, equational rewriters handle only a small fragment of logical reasoning and the task they perform is focused and specific. If we take domain-general reasoning to encompass problems that, in full generality, are equivalent to the halting problem, it makes sense to exclude equational rewriters, which are designed to reduce expressions to canonical forms in a finite number of steps. In any case, they are incredibly useful. Tools like Isabelle's `simp` can simplify a formula to `True` and hence prove more than just equations. They can also carry out conditional rewriting and use backchaining to dispel side conditions. Users can add facts to the rewrite database as they develop new theories. As far as I know, any proof assistant that takes automation seriously has some sort of rewrite engine. Lean's version of `simp` was one of the first tactics that was implemented in that system.

To prepare for the talk at FroCoS and TABLEAUX, I also carried out some experiments with Isabelle's *Sledgehammer* [46]. This is a tool that, given a proof goal, uses a relevance filter to select a couple of hundred potentially useful theorems from the library, sends the problem to external provers, and then tries to use the information they return to reconstruct a proof internally. I set myself the task of determining the extent to which I could formalize theorems by writing a proof sketch, calling only Sledgehammer and `auto`, and then refining the sketch as needed. I formalized three theorems in this way—the mutilated chessboard problem, the intermediate value theorem, and the existence of infinitely many primes congruent to three modulo four—and I took detailed notes as I went. The data, which is still available in the GitHub repository, is not very rigorous, but it helped me understand better some of the places where the automation fell short. In particular, two of the theorems required mild forms of second-order reasoning, such as identities governing the summation of functions over finite sets or reasoning about membership in sets defined by explicit predicates. Now, provers like Vampire, Zipperposition, and E can handle higher-order reasoning natively [7,8,53], which is likely to help.

One method of proof reconstruction involves harvesting nothing more than the list of theorems the external prover needed to establish the goal and calling internal, proof-producing automation to redo the search. Isabelle uses a tool called *Metis* for that [31]. Joshua Clune, Yicheng Qian, and Alexander Bentkamp have written a proof-producing superposition theorem prover for Lean called *Duper* to serve a similar purpose, as well as to serve as generic internal automation. They invested considerable effort in adapting conventional resolution methods to dependent type theory. For example, in dependent type theory, a data type might not have any elements, and Skolemization and other components of the search calculus have to be adapted to avoid introducing unsoundness. Duper can instantiate generic type variables on the fly but that introduces additional technical complications, as does adapting unification procedures to the depedently typed setting. Because it is modeled on Zipperposition, Duper can also handle higher-order inferences, and because it operates on Lean expressions directly (rather than via translation), it is possible to handle other rules tailored specifically to dependent type theory. Testing on standard benchmarks shows that Duper's performance is roughly comparable to Metis', offering hope that we will have a Sledgehammer for Lean before long. Lean is starting to catch up with Isabelle in other respects as well, with an automated reasoner called Aesop [33], inspired by Isabelle's `auto`, as well as tools like Coq's `eauto` and PVS's `grind`.

When we consider the way that mathematicians use proof assistants, it becomes clear that support for reasoning about algebraic structures is essential. I have sometimes heard computer scientists say that there is no need to use dependent type theory because anything that can be done there can be done just as well in set theory or simple type theory. In principle, any reasonable foundation can interpret any other, possibly modulo a few axiomatic extensions, but generally speaking, the remark fails to appreciate the extent to which algebraic language and thought pervade contemporary mathematics. Any undergraduate student of mathematics can talk about the ring of $n \times n$ matrices of polynomials over $\mathbb{Z}/p\mathbb{Z}$ for a prime number p. Moreover, that student knows that multiplication is commutative on the polynomials and their coefficients but not on the matrices, and that nonzero elements of $\mathbb{Z}/p\mathbb{Z}$ have multiplicative inverses while the polynomials and matrices generally don't. In other words, mathematicians easily name complex structures and use generic notation, and they know what properties elements of those structures have. The structures themselves are mathematical objects on par with numbers, functions, and circles, yet at the same time they serve as data types, constraining what can meaningfully be said about their elements. I find it remarkable that Lean and Mathlib are so good at making a vast network of structures available to users without collapsing under the technical requirements. Doing so requires a carefully designed system of type class inference and efficient means of elaborating and unifying the complex expressions that describe the structures and their elements. Dependent type theory may not be the only possible solution, but it is the only one implemented so far that can do anything close to what mathematicians need.

Reasoning about a rich algebraic hierarchy is a challenge for automated reasoning for at least two reasons: first, because instantiating generic theorems requires determining whether the types in question are instances of the relevant algebraic structures, and, second, because carrying out unification with expressions that have algebraic components requires determining the identity of objects that have been described in different ways. Both of these tasks are too expensive to be carried out in the midst of an automated search, but, fortunately, we generally don't expect them to be: mathematicians are usually careful to establish the relevant algebraic context explicitly. Duper manages algebraic reasoning using a remarkable preprocessing tool by Qian called *Lean-Auto*, which heuristically instantiates generic theorems, infers the relevant algebraic structures, and chooses canonical representations of expressions, all before the search begins.

Sledgehammers generally work by translating the source language to a simpler one and using tools optimized for equational reasoning, propositional reasoning, and quantifier instantiation. Another approach to automating dependent type theory is to search systematically in dependent type theory itself. There are tools for Coq [16] and Agda [34,51] that take this approach, without making any attempt at completeness. At Carnegie Mellon, Chase Norman has implemented a procedure for a minimal but fully expressive dependent type theory that provides a complete solution to both the unification and type inhabitation problems, generalizing Huet's unification procedure for higher-order logic [30]. The framework is flexible enough to instantiate various heuristics to carry out the search, and the implementation performs well on examples. It will be interesting to see whether such an approach will provide automation that complements the strengths of a sledgehammer.

5 Domain-Specific Reasoning for Verification

Talking about domain-general automation reminds me of a quip that I once heard attributed to Sidney Morgenbesser that philosophers are people who know something about everything but nothing about anything. In an ornery mood, I might complain that first-order theorem provers are good at reasoning about everything but not so good at reasoning about anything in particular. At the opposite end of the spectrum are domain-specific automated reasoning tools that carry out more deterministic and focused tasks, such as reasoning about arithmetic, establishing algebraic identities, reasoning about linear and nonlinear inequalities, and so on.

Tools like these are extremely useful. Lean has benefited from the availability of a metaprogramming language, introduced in Lean 3 [18] and made vastly more powerful in Lean 4 [38], that allows users to write tactics within Lean. The ability to attract mathematical users from 2017 on was bolstered by the fact that users like Mario Carneiro were able to quickly provide them with the tactics they needed. As mathematicians gained expertise with the system, they could design tactics that would help them in their daily work. For example, Heather Macbeth, with the help of Carneiro, wrote tactics `gcongr` and `positivity` to help with common calculations, and then could easily shorten a proof like this:

```
calc ‖wp - wq‖ * ‖wp - wq‖
  _ = 2 * (‖a‖ * ‖a‖ + ‖b‖ * ‖b‖) - 4 * ‖u - half · (wq + wp)‖ *
        ‖u - half · (wq + wp)‖ := by rw [← this]; simp
  _ ≤ 2 * (‖a‖ * ‖a‖ + ‖b‖ * ‖b‖) - 4 * δ * δ :=
        (sub_le_sub_left eq₁ _)
  _ ≤ 2 * ((δ + div) * (δ + div) + (δ + div) * (δ + div)) -
        4 * δ * δ :=
        (sub_le_sub_right (mul_le_mul_of_nonneg_left
          (add_le_add eq₂ eq₂') (by norm_num)) _)
```

```
    _ = 8 * δ * div + 4 * div * div := by ring
  exact
    add_nonneg (mul_nonneg (mul_nonneg (by norm_num) zero_le_δ)
        (le_of_lt Nat.one_div_pos_of_nat))
      (mul_nonneg (mul_nonneg (by norm_num) Nat.one_div_pos_of_nat.le)
      Nat.one_div_pos_of_nat.le)
```

to this:

```
  calc ‖wp - wq‖ * ‖wp - wq‖
    _ = 2 * (‖a‖ * ‖a‖ + ‖b‖ * ‖b‖) - 4 * ‖u - half · (wq + wp)‖ *
            ‖u - half · (wq + wp)‖ := by simp [← this]
    _ ≤ 2 * (‖a‖ * ‖a‖ + ‖b‖ * ‖b‖) - 4 * δ * δ := by gcongr
    _ ≤ 2 * ((δ + div) * (δ + div) + (δ + div) * (δ + div)) -
          4 * δ * δ := by gcongr
    _ = 8 * δ * div + 4 * div * div := by ring
  positivity
```

Tomáš Skřivan recently contributed a tactic, `fun_prop`, that effectively establishes properties like continuity, differentiability, and measurability of functions.

I am grateful to Adam Topaz for writing a small Lean metaprogram to extract tactic usage statistics from a recent version of Mathlib. The data is messy because tactic variants are listed separately when they are called under separate wrappers, including variants that were written to support the port of the library from Lean 3 but are otherwise superseded by newer versions. It is also misleading in that some tactics have been around much longer than others, so the numbers do not reflect the current utility. Nor does the data say anything about the role of domain-specific automation outside of Mathlib. Finally, some tactics are used transiently and are then eliminated from the final proof document, such as those that help find theorems, display information, or write proofs. These do not appear in the list.

Nonetheless, the data is informative. Tactics used to apply theorems are most common (`apply`, `exact`, `refine`, etc., with about 60K instances in all). After that, the vast majority of tactic calls, by far, are used for equational reasoning, with more than 52K invocations of Lean's `rw` tactic, which does manual term rewriting, and more than 60K invocations of Lean's simplifier (`simp`, `simpa`, `dsimp`, and `simp-rw`). About 25K invocations are used to decompose data and existential assertions (`obtain`, `rintro`, `rcases`, `cases`, etc.), and there are about 5K calls to tactics that carry out proof by induction.

More specialized automation still manages to carry its weight. The `linarith` tactic is called more than 1,100 times; `split_ifs`, which splits a goal to simplify conditional expressions, and `ring`, which carries out ring calculations, are each called more than 1,000 times; `filter_upwards`, a simple tactic that helps reason about filters in measure theory and analysis is called almost 900 times; `norm_num`, which does numeric calculations, is called more than 800 times; a specialization of Aesop for use with category theory, `aesop_cat`, is called almost 800 times; `positivity`, a relative newcomer, is called more than 600 times; and `norm_cast`, a tactic to help mediate casts between numeric domains, and `gcongr` are each called more than 500 times.

Lean's metaprogramming language also provides flexible ways to support better interactions with automation, both domain-general and domain-specific. Lean's *Widgets* framework [42] allows users to install custom Javascript-driven displays of objects and information in Lean's "infoview" window in VS Code, and allows user interactions with these graphical displays to communicate information and actions back to Lean and the editor. When the user puts the cursor at the beginning of a tactic invocation in a proof, the infoview window highlights the part of the state that is about to change, and when the cursor is on or at the end of the invocation, it highlights what has just changed. Users can hover over constants in the infoview window to see documentation, they can click on expressions to see their types, and they can control-click on identifiers to jump to their definitions. Users can trace class inference to diagnose failures, and expand or collapse nodes of the search tree in the infoview window. In Lean, automation can return structured error messages that can be explored. Ideally, whenever automation fails, users should have the means to diagnose the problem and come up with suitable interventions to fix it. Developers of automation should keep user interfaces in mind both as targets for automated reasoning and as means for using automation more effectively.

Finally, it is worth mentioning that tools that help users find theorems and explore the library are also essential. Lean provides internal tactics like `apply?`, `exact?`, and `rw?` that suggest atomic proof steps. There is also a good symbolic search engine, *Loogle*, and there is another search tool, *Moogle*, that uses a large language model to answer natural-language queries.

6 Automation for the Discovery of New Theorems

To this point, we have focused on the use of automated reasoning to verify mathematical results that were discovered by conventional means. Mathematicians, however, tend to be much more excited about methods that help with the discovery of new mathematics. The automated reasoning community is justifiably proud of William McCune's use of his theorem prover, EQP, to settle to Robbins conjecture in 1996 [37]. The result, which shows that a certain set of equations can be taken as an alternative axiomatization of Boolean algebras, made the pages of the *New York Times*. Since then, there has also been a small industry in using automated theorem provers to prove theorems about other algebraic structures, like loops and quasigroups. One can think of a loop as like a group except without the associativity axiom, and one can think of a quasigroup as a loop without an identity. First-order theorem provers have been used to establish consequences of these and related axioms, an industry nicely surveyed by Phillips and Stanovský [47].

I have heard mathematicians express annoyance, however, at the suggestion that these results have anything to do with real mathematics. Few mathematicians have heard of the Robbins conjecture or have any interest in quasigroups. More to the point, mathematicians chafe at the implication that modern algebra is about deriving first-order consequences of axioms. Algebraists are interested in classification theorems, which characterize structures in terms of key invariants, structure theorems, which provide means of understanding structures in terms of subobjects and morphisms to other structures, and representation theorems. They are interested in introducing new structures and new spaces of structures, with applications that explain and simplify past results and provide powerful tools for future research. All these involve reasoning about structures within the context of a rich mathematical theory, rather than reasoning deductively from the axioms. As a result, to most mathematicians, the applications of automated reasoning to algebra so far are little more than recreational curiosities.

Applications of SAT solvers to mathematics fare better. For example, in 1912, Issai Schur proved that given any finite coloring of the positive integers, there is a monochromatic solution to the equation $x + y = z$. Today, this is recognized as a seminal result in both Ramsey theory and additive number theory. The theorem raises the question as to whether one can compute the largest initial segment of the positive integers $\{1, 2, \ldots, S(k)\}$ such that there is a k-coloring with no such monochromatic solution. It's not hard to establish $S(1) = 1$, $S(2) = 4$, and $S(3) = 13$. In 1965, Golomb and Baumert computed $S(4) = 44$ in a paper that contains other interesting examples of backtracking search [23]. The value $S(5) = 160$ was computed by Heule in 2017 using a SAT solver [28], a result which has drawn praise from the mathematical community. The value of $S(6)$ is still unknown.

Most mathematicians aren't interested in calculating Schur numbers, but the problem is considered at least interesting by association, given that they recognize Schur's theorem as an important theorem. The case is similar with respect to a theorem that Paul Erdős dubbed the "happy ending problem" because it led to the marriage of George Szekeres and Esther Klein. The general version of the theorem says that for every positive integer n, any sufficiently large finite set of points in general position in the plane contains a convex n-gon. Let $f(n)$ denote the smallest number of points in general position that provides such a guarantee; the value of $f(n)$ is known only for $n \leq 6$. A related problem asks for the smallest number of points in general position that guarantees the existence of an *empty* convex n-gon, that is, one without any points in its interior. There are infinite sets of points without a convex 7-gon, but Nicolás [43] and Gerken [20] proved independently that every sufficiently large set of points in general position contains an empty convex hexagon. Using a SAT solver, Heule and Scheucher [29] recently showed that 30 is the smallest number of points that provides that guarantee.

When SAT solvers are used to solve mathematical problems, it is important to have guarantees that the results are correct. Students at Carnegie Mellon are working on a SAT library for Lean that addresses that concern. Joshua Clune has written an LRAT checker that is currently in use at Amazon Web Services; Wojciech Nawrocki has verified a checker for *knowledge compilation*, a technology that, in particular, can provide precise counts of the number of satisfying assignments to a propositional formula [12]; and Cayden Codel has written a verified checker for SR, a strong proof format for SAT solvers that incorporates symmetries in “without loss of generality” reductions [13]. Perhaps the more serious concern is to verify that a problem that is sent to a SAT solver is a correct representation of the intended problem. This is pressing because the reduction of an ordinary mathematical statement to a SAT problem often relies on complex encodings as well as symmetry breaking and other reductions, and the generation of the propositional formula is further subject to subtle programming errors. Codel, Nawrocki, and James Gallicchio have been working on aspects of the library that address that problem as well [14]. Bernardo Subercaseaux, Nawrocki, Gallicchio, Codel, Carneiro, and Heule have verified the correctness of the encoding used to compute the empty hexagon number [52].

Mathematicians have yet to explore the use of automated reasoning tools to *find* objects of mathematical interests. SAT solvers, SMT solvers, constraint solvers, and model finders are all designed to find objects and structures satisfying given constraints. A popular Isabelle tool called *Nitpick* [10] uses a model finder to look for counterexamples to purported theorems, in order to prevent them from investing time and energy in trying to prove a statement that is false.

Bespoke decision procedures can also aid the process of discovery. An automated reasoning tool called *Walnut* [40,49] implements a decision procedure for an extension of Presburger arithmetic that can express properties of *automatic sequences*, which are, roughly, sequences generated by finite state automata. Consider the question as to whether there is an infinite binary sequence with no subsequence of the form xxx^R, where x is a finite sequence and x^R is its reversal. It is not hard to see that the sequence 01010101 . . . has that property, but is it possible to find one that is aperiodic? A paper by Mousavi, Schaeffer, and Shallit [41] explains how Walnut helped them construct such a sequence.

I believe that mathematicians’ general habit of dismissing “finite problems” as not properly mathematical will change over time. The entire edifice of infinitary mathematics bears on our everyday experience only through measurement and observation, and discrete problems from computer science have already begun to influence mathematical research. It also seems likely that mathematicians will find creative ways to solve infinitary problems by devising representations and reductions that are amenable to automation. The main challenge is that automated reasoning is unfamiliar to them. The history of Lean suggests that mathematicians will go to extraordinary lengths to learn a new technology once they decide that it is interesting and aligns with their goals. To facilitate adoption, it helps to have documentation and expository materials that are written with them in mind. The incentive structures in mathematics and computer science are not good at encouraging that kind of cross-disciplinary outreach, but once the door is open, it is only a matter of time before a new technology becomes part of the mathematical mainstream.

7 Machine Learning and Symbolic AI

It’s impossible to write about the prospects of automated reasoning for mathematics today without saying something about machine learning. Machine learning and symbolic AI have complementary strengths: ML is good at synthesizing vast amounts of data but isn't good at getting details right, whereas symbolic methods are good at getting the details right but are overwhelmed by combinatorial explosion in a search space. A central challenge for AI is to design systems that get the best of both worlds by combining the two approaches, and mathematics, where the problems are especially clear and well-defined, is an ideal place to make progress.

It is therefore not surprising that there is growing interest in applications of machine learning to mathematics. Researchers have long explored the use of machine learning techniques to guide symbolic search and to select premises for a sledgehammer, and with the advent of deep learning, there has been a surge of interest in using neural networks to prove theorems in conjunction with an interactive proof assistant. A recent "brief survey" of machine learning in automated reasoning by Blaauwbroek et al. [9] has 168 references, and surveys of deep learning for mathematical reasoning by Lu et al. [35] and by Li et al. [32] have well over 200 references each.

Lean is becoming recognized as an ideal platform for such work. There have been at least two projects on using machine learning for premise selection for Lean [19,48], there are tactics and code pilots based on Lean [27,54,55], and there are tools that support data extraction and interaction with Lean for machine learning experiments, including one developed by Kaiyu Yang and coauthors [55] and two developed by Kim Morrison.[3] The MiniF2F problem benchmark [56] includes versions for Lean 3, and the ProofNet benchmarks [5] have recently been ported to Lean 4 by Abhijit Chakrborty and Rahul Vishwakarma.[4] The features that make Lean a good platform for automated reasoning also make it a good platform for machine learning and support a synthesis of the two. The size and sophistication of Lean's mathematical library, Mathlib, and the involvement of the mathematical community provide powerful opportunities for progress.

8 Conclusions

Although I began with a disappointing assessment of the current state of automated reasoning for mathematics, I hope I have conveyed good reasons for optimism. Mathematicians are beginning to warm to the use of formal methods, opening up new avenues for progress. As the technology improves, the number of mathematicians making use of the automated reasoning tools will increase, providing greater incentives for computer scientists to focus on mathematical applications. This, in turn, will increase the number of mathematicians who use the the technology and can therefore provide feedback and even contribute to its development. In short, I expect that we are on the cusp of a virtuous cycle whereby technological improvements lead to more users, which, in turn, lead to further improvements.

[3] https://github.com/semorrison/lean-training-data
https://github.com/leanprover-community/repl

[4] https://github.com/rahul3613/ProofNet-lean4

In this article, I have tried to identify some of the key challenges to developing better automation for mathematics, and I have suggested specific approaches that I find promising. The biggest challenges, however, may be sociological rather than technical. Making automation useful for mathematics will require mathematicians and computer scientists working together, and neither discipline will get far on its own. Mathematicians and computer scientists have very different attitudes and outlooks. Computer scientists focus on disseminating information quickly in conference publications, and their success is measured by the number of citations they receive. With the weight of centuries behind them, mathematicians can't be rushed, are mistrustful of academic fads, and tend to look to the leading experts in their fields to determine what is important. Whereas computer scientists value measurable impact in the short term, mathematicians answer to less clear-cut assessments of the quality and depth of their work. It's not that one discipline's standards are better than the other; each has its advantages and problems. It's just that the disparity of outlooks often makes communication difficult.

I'd like to suggest to computer scientists reading this article that it might be nice to adopt a mathematical attitude every once in a while. Imagine thinking about something because you find it interesting, without caring what others think. Imagine exploring ideas to see where they take you, without worrying about whether that will result in a conference publication by the next round of deadlines. Imagine working on a problem just for the joy of taking up the challenge. If all that sounds good to you, you'll find mathematicians to be excellent companions. I am not suggesting that you should turn your back on computer science; of course, you will still have bills to pay. But I am confident that one day, when you look back over your career, any contributions you have made to mathematics will be among the things that you are most proud of, and among those that are closest to your heart. So I invite you to come to the Lean Zulip channel and start talking to mathematicians about the things that automation can do for them. I promise, you won't regret it.

References

1. Avigad, J.: The mechanization of mathematics. Notices Amer. Math. Soc. **65**(6), 681–690 (2018). https://doi.org/10.1090/noti1688
2. Avigad, J., Donnelly, K., Gray, D., Raff, P.: A formally verified proof of the prime number theorem. ACM Trans. Comput. Log. **9**(1), 2 (2007). https://doi.org/10.1145/1297658.1297660
3. Avigad, J., Hölzl, J., Serafin, L.: A formally verified proof of the central limit theorem. J. Autom. Reason. **59**(4), 389–423 (2017). https://doi.org/10.1007/S10817-017-9404-X
4. Avigad, J., Kapulkin, K., Lumsdaine, P.L.: Homotopy limits in type theory. Math. Struct. Comput. Sci. **25**(5), 1040–1070 (2015). https://doi.org/10.1017/S0960129514000498
5. Azerbayev, Z., Piotrowski, B., Schoelkopf, H., Ayers, E.W., Radev, D., Avigad, J.: Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. CoRR **abs/2302.12433** (2023). https://doi.org/10.48550/ARXIV.2302.12433

6. Beeson, M.J.: The mechanization of mathematics. In: Alan Turing: life and legacy of a great thinker, pp. 77–134. Springer, Berlin (2004)
7. Bentkamp, A., Blanchette, J., Nummelin, V., Tourret, S., Vukmirovic, P., Waldmann, U.: Mechanical mathematicians. Commun. ACM **66**(4), 80–90 (2023). https://doi.org/10.1145/3557998
8. Bentkamp, A., Blanchette, J., Tourret, S., Vukmirović, P.: Superposition for higher-order logic. Journal of Automated Reasoning **67**(1) (Jan 2023). https://doi.org/10.1007/s10817-022-09649-9
9. Blaauwbroek, L., Cerna, D., Gauthier, T., Jakubův, J., Kaliszyk, C., Suda, M., Urban, J.: Learning guided automated reasoning: A brief survey. CoRR **abs/2403.04017** (2024), https://arxiv.org/abs/2403.04017
10. Blanchette, J.C., Nipkow, T.: Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In: Kaufmann, M., Paulson, L.C. (eds.) Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6172, pp. 131–146. Springer (2010). https://doi.org/10.1007/978-3-642-14052-5_11
11. Börger, E., Grädel, E., Gurevich, Y.: The classical decision problem. Universitext, Springer-Verlag, Berlin (2001), reprint of the 1997 original
12. Bryant, R.E., Nawrocki, W., Avigad, J., Heule, M.J.H.: Certified knowledge compilation with application to verified model counting. In: Mahajan, M., Slivovsky, F. (eds.) 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy. LIPIcs, vol. 271, pp. 6:1–6:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.SAT.2023.6
13. Buss, S., Thapen, N.: DRAT and propagation redundancy proofs without new variables. Log. Methods Comput. Sci. **17**(2) (2021). https://doi.org/10.23638/LMCS-17(2:12)2021
14. Codel, C.R., Avigad, J., Heule, M.J.H.: Verified encodings for SAT solvers. In: Nadel, A., Rozier, K.Y. (eds.) Formal Methods in Computer-Aided Design, FMCAD 2023, Ames, IA, USA, October 24-27, 2023. pp. 141–151. IEEE (2023). https://doi.org/10.34727/2023/ISBN.978-3-85448-060-0_22
15. Crossley, J.N.: Reminiscences of logicians. In: Crossley, J.N. (ed.) Algebra and Logic. pp. 1–62. Springer Berlin Heidelberg, Berlin, Heidelberg (1975)
16. Czajka, L.: Practical proof search for Coq by type inhabitation. In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12167, pp. 28–57. Springer (2020). https://doi.org/10.1007/978-3-030-51054-1_3
17. Davis, M.: The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems, and Computable Functions. Dover Publications, revised edition edn. (2004)
18. Ebner, G., Ullrich, S., Roesch, J., Avigad, J., de Moura, L.: A metaprogramming framework for formal verification. Proc. ACM Program. Lang. **1**(ICFP), 34:1–34:29 (2017). https://doi.org/10.1145/3110278
19. Geesing, A.: Premise Selection for Lean 4. Master's thesis, Vrije Universiteit Amsterdam (2023)
20. Gerken, T.: Empty convex hexagons in planar point sets. Discrete Comput. Geom. **39**(1-3), 239–272 (2008). https://doi.org/10.1007/s00454-007-9018-x
21. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatsh. Math. Phys. **38**(1), 173–198 (1931). https://doi.org/10.1007/BF01700692, English translation in [22]

22. Gödel, K.: Collected works. Vol. I. Oxford University Press, New York (1986), edited by Solomon Feferman, John W. Dawson Jr., Stephen C. Kleene, Gregory H. Moore, Robert M. Solovay, and Jean van Heijenoort
23. Golomb, S.W., Baumert, L.D.: Backtrack programming. J. Assoc. Comput. Mach. **12**, 516–524 (1965). https://doi.org/10.1145/321296.321300
24. Gonthier, G.: Formal proof—the four-color theorem. Notices Amer. Math. Soc. **55**(11), 1382–1393 (2008)
25. Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Roux, S.L., Mahboubi, A., O'Connor, R., Biha, S.O., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., Théry, L.: A machine-checked proof of the odd order theorem. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7998, pp. 163–179. Springer (2013). https://doi.org/10.1007/978-3-642-39634-2_14
26. Hales, T.C.: The Jordan curve theorem, formally and informally. Amer. Math. Monthly **114**(10), 882–894 (2007). https://doi.org/10.1080/00029890.2007.11920481
27. Han, J.M., Rute, J., Wu, Y., Ayers, E.W., Polu, S.: Proof artifact co-training for theorem proving with language models. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net (2022), https://openreview.net/forum?id=rpxJc9j04U
28. Heule, M.J.H.: Schur number five. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. pp. 6598–6606. AAAI Press (2018). https://doi.org/10.1609/AAAI.V32I1.12209
29. Heule, M.J.H., Scheucher, M.: Happy ending: An empty hexagon in every set of 30 points (2024), to appear in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2024*
30. Huet, G.P.: A unification algorithm for typed lambda-calculus. Theor. Comput. Sci. **1**(1), 27–57 (1975). https://doi.org/10.1016/0304-3975(75)90011-0
31. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. In: Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003). pp. 56–68 (2003), http://www.gilith.com/papers
32. Li, Z., Sun, J., Murphy, L., Su, Q., Li, Z., Zhang, X., Yang, K., Si, X.: A survey on deep learning for theorem proving. CoRR **abs/2404.09939** (2024), https://arxiv.org/abs/2404.09939
33. Limperg, J., From, A.H.: Aesop: White-box best-first proof search for Lean. In: Krebbers, R., Traytel, D., Pientka, B., Zdancewic, S. (eds.) Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2023, Boston, MA, USA, January 16-17, 2023. pp. 253–266. ACM (2023). https://doi.org/10.1145/3573105.3575671
34. Lindblad, F., Benke, M.: A tool for automated theorem proving in Agda. In: Filliâtre, J.C., Paulin-Mohring, C., Werner, B. (eds.) Types for Proofs and Programs. pp. 154–169. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
35. Lu, P., Qiu, L., Yu, W., Welleck, S., Chang, K.: A survey of deep learning for mathematical reasoning. In: Rogers, A., Boyd-Graber, J.L., Okazaki, N. (eds.) Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023. pp. 14605–14631. Association for Computational Linguistics (2023). https://doi.org/10.18653/V1/2023.ACL-LONG.817

36. Mackenzie, D.: The automation of proof: a historical and sociological exploration. IEEE Annals of the History of Computing **17**(3), 7–29 (1995). https://doi.org/10.1109/85.397057
37. McCune, W.: Solution of the Robbins problem. J. Autom. Reason. **19**(3), 263–276 (1997). https://doi.org/10.1023/A:1005843212881
38. de Moura, L., Ullrich, S.: The Lean 4 theorem prover and programming language. In: Platzer, A., Sutcliffe, G. (eds.) Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12699, pp. 625–635. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_37
39. de Moura, L.M., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The Lean theorem prover (system description). In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9195, pp. 378–388. Springer (2015). https://doi.org/10.1007/978-3-319-21401-6_26
40. Mousavi, H.: Automatic theorem proving inWalnut. CoRR **abs/1603.06017** (2016), http://arxiv.org/abs/1603.06017
41. Mousavi, H., Schaeffer, L., Shallit, J.O.: Decision algorithms for Fibonacci-automatic words, I: basic results. RAIRO Theor. Informatics Appl. **50**(1), 39–66 (2016). https://doi.org/10.1051/ITA/2016010
42. Nawrocki, W., Ayers, E.W., Ebner, G.: An extensible user interface for Lean 4. In: Naumowicz, A., Thiemann, R. (eds.) 14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok, Poland. LIPIcs, vol. 268, pp. 24:1–24:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.ITP.2023.24
43. Nicolás, C.M.: The empty hexagon theorem. Discrete Comput. Geom. **38**(2), 389–397 (2007). https://doi.org/10.1007/s00454-007-1343-6
44. Nipkow, T.: Term rewriting and beyond - theorem proving in Isabelle. Formal Aspects Comput. **1**(4), 320–338 (1989). https://doi.org/10.1007/BF01887212
45. Paulson, L.C.: A generic tableau prover and its integration with Isabelle. J. Univers. Comput. Sci. **5**(3), 73–87 (1999). https://doi.org/10.3217/JUCS-005-03-0073
46. Paulson, L.C., Blanchette, J.C.: Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In: Sutcliffe, G., Schulz, S., Ternovska, E. (eds.) The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011. EPiC Series in Computing, vol. 2, pp. 1–11. EasyChair (2010). https://doi.org/10.29007/36DT
47. Phillips, J.D., Stanovský, D.: Automated theorem proving in quasigroup and loop theory. AI Commun. **23**(2-3), 267–283 (2010). https://doi.org/10.3233/AIC-2010-0460
48. Piotrowski, B., Mir, R.F., Ayers, E.W.: Machine-learned premise selection for Lean. In: Ramanayake, R., Urban, J. (eds.) Automated Reasoning with Analytic Tableaux and Related Methods - 32nd International Conference, TABLEAUX 2023, Prague, Czech Republic, September 18-21, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14278, pp. 175–186. Springer (2023). https://doi.org/10.1007/978-3-031-43513-3_10
49. Shallit, J.: The logical approach to automatic sequences—exploring combinatorics on words with `Walnut`, London Mathematical Society Lecture Note Series, vol. 482. Cambridge University Press, Cambridge (2023). https://doi.org/10.1017/9781108775267

50. Siekmann, J., Wrightson, G. (eds.): Automation of Reasoning 1. Springer, Berlin (1983)
51. Skystedt, L.: A New Synthesis Tool for Agda. Master's thesis, University of Gothenburg and Chalmers University of Technology (2022)
52. Subercaseaux, B., Nawrocki, W., Gallicchio, J., Codel, C., Carneiro, M., Heule, M.J.H.: Formal verification of the empty hexagon number. CoRR **abs/2403.17370** (2024), https://arxiv.org/abs/2403.17370
53. Vukmirovic, P., Blanchette, J., Schulz, S.: Extending a high-performance prover to higher-order logic. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13994, pp. 111–129. Springer (2023). https://doi.org/10.1007/978-3-031-30820-8_10
54. Welleck, S., Saha, R.: LLMSTEP: LLM proofstep suggestions in Lean. CoRR **abs/2310.18457** (2023). https://doi.org/10.48550/arxiv.2310.18457
55. Yang, K., Swope, A.M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R.J., Anandkumar, A.: Leandojo: Theorem proving with retrieval-augmented language models. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023 (2023), http://papers.nips.cc/paper_files/paper/2023/hash/4441469427094f8873d0fecb0c4e1cee-Abstract-Datasets_and_Benchmarks.html
56. Zheng, K., Han, J.M., Polu, S.: MiniF2F: a cross-system benchmark for formal Olympiad-level mathematics. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net (2022), https://openreview.net/forum?id=9ZPegFuFTFv