

—

On AI-Assisted Mathematical Research

Tips from a Mathematician-Turned-Developer

Sonat Süer

Presented at the University of Wrocław Model Theory Seminar
on 14 October 2026

Introduction

Introduction

Thank you for the invitation. I am *very* rusty at giving talks to mathematicians, so I apologize in advance.

I would also like to thank Piotr Kowalski for giving valuable feedback on the initial drafts of this talk.

About the Speaker

I

- got a PhD in mathematical logic in a past life,
- have been working as a software developer for about nine years,
- have been using AI professionally since it became useful in my line of work.

Introduction

— About the Speaker

Full disclosure: I am giving this talk because I solved a few problems in model theory on a shoestring budget (20 Euros per month subscription), using AI with a methodology not usually adopted by mathematicians, in a short amount of time. And I happen to know Piotr.

About the Talk

The talk will be mostly about technical tips, inspired by software development practice, on using AI for mathematical research.

More importantly, the talk is *not* about the future of AI and mathematics, especially not from ethical or societal points of view. The talk is also not about the mathematical theory behind large language models.

Introduction

— About the Talk

We are at a point in time where predictions are especially difficult –I revised this talk a few times in the last month due to the rapid progress.

The talk is an answer to the question “If *I* were to do mathematics today, how would I incorporate AI?” This doesn’t mean you should do it in the same way or do it at all. It also doesn’t mean that I would do mathematics today.

The reasonable takeaway would be having an introductory understanding of what is possible if you decide to use AI beyond prompting.

The Plan

Here is the rough plan:

- *AI for the Working Mathematician*: A soft introduction to the basic notions of practical AI without assuming any background.
- *Automation and Tuning*: Notes on technology assisting large language models.
- *Formalization*: Why and how to formalize. Pitfalls around formalization.
- *Case studies*: There will be three case studies, two of which are about model theory.

Introduction — The Plan

AI for the Working Mathematician

—

Feel free to stop me and ask questions, especially if I forget that I am talking to mathematicians.

AI for the Working Mathematician

What I Mean by AI

AI used to mean a lot of things; nowadays people mostly mean Large Language Models (LLMs) when they say AI. We will also focus on LLMs. Two properties of LLMs are important to understand:

- LLMs are stateless,
- LLMs are probabilistic.

AI for the Working Mathematician — What I Mean by AI

More can be said about LLMs, but for practical purposes these two are the most important. Also, LLMs are almost never used “naked”, but we will come to that.

What Does Stateless Mean?

At heart, an LLM is a mathematical function with some randomness. It does not *learn* from you; learning is a stage which happens before a model is released.

When you are having a conversation with an AI assistant, each time you send the *entire* dialog to the assistant, not just your latest message. The input you send to the LLM is called the context, and it is a limited resource. You cannot let it grow arbitrarily.

AI for the Working Mathematician — What Does Stateless Mean?

The context window is a limited resource! Also, it is not homogeneous. LLMs tend to pay more attention to the head and tail of what is in the context.

Sessions to Tackle Statelessness

Typically one interacts with an LLM in sessions. Different sessions share no context, so context needs to be communicated explicitly to the next session. In mathematics, this can be done by

- a research plan,
- a handoff document from one agent to another,
- a research “journal”,
- a ledger of statements proven, disproven, conjectured, partially verified computationally,
- etc.

Note these are generated *by* an AI, *for* an AI.

AI for the Working Mathematician — Sessions to Tackle Statelessness

In general, it is a good idea to record both success and failure. We will see concrete examples of this in the case studies.

What Does Probabilistic Mean?

The output of an LLM is sampled, so it is not usually reproducible in exact form. (The popular expression is that LLMs are stochastic parrots.)

More importantly, LLMs *hallucinate*, though not as much as they used to.

AI for the Working Mathematician — What Does Probabilistic Mean?

This makes LLMs unreliable. In the context of mathematics, it is actually dangerous: unattended, they can produce plausible *and* wrong proofs.

Pushing Stochasticity Upstream

Naive approach:



Optimized approach:



Motto: If a problem can be solved by a non-LLM tool, use it. If necessary, use an LLM to generate the non-LLM tool.

AI for the Working Mathematician — Pushing Stochasticity Upstream

This is something already done by a lot of AI assistants. If you ask an assistant to find something in a text, it does not put the entire text in its context. It runs a specific tool. The intermediary artifact it generates is the search parameters fed to the tool. Assistants also write small scripts to handle tasks. **The design space of intermediary artifacts is rich.**

In the context of mathematics, the intermediary artifact can be a formal specification that can be processed by a symbolic engine (SMT solver, symbolic algebra software, proof checker, etc.) or a custom *domain-specific language* with a custom semantics. Again, we will see concrete examples.

Automation and Tuning

We have seen that LLMs need support to be useful on their own. This part is about automating that support.

Automation and Tuning

Skills, Output Modes and the Like

AI assistants are highly configurable. I will not go into much detail here because this aspect of AI assistants is well documented. For the record, you can

- “record” a process as a skill,
- modify the way an LLM “speaks”,
- create a process on the spot,
- etc.

Automation and Tuning — Skills, Output Modes and the Like

What Is an Agent?

An agent is an autonomous system with *agency* driven by an LLM. Agents can make decisions, explore their environments and, more importantly, change them.

The most prominent examples are coding agents, which can read a code repository, decide on an implementation plan, do the implementation and run tests on the implementation.

Automation and Tuning — What Is an Agent?

There are other examples of agents which can do customer support, market research, etc. Coding agents are the most relevant examples for us.

What Is a Harness?

A harness is infrastructure in which the agents live. A harness can do any number of the following and possibly more:

- sandboxing execution for security,
- managing resources,
- injecting prompts,
- monitoring outputs,
- enforcing policy.

For instance, Prove2Me is a collaborative autoformalization harness used in the formalization of Fermat's Last Theorem.

Automation and Tuning — What Is a Harness?

The notion of a harness may look scary to nontechnical people, but actually it is not. It is just a higher level of automation.

Creating a Custom Harness

You can create your own harness! Remember that LLMs are very good at generating software from a well-written specification.

Here is an example project:

Li, A., Saha, R., Xue, A., Chaudhuri, S., Klivans, A., Kothari, P. K., & Meka, R. (2026). Long-Horizon AI Research for Grothendieck Constant: A Case Study in Human–AI Mathematical Collaboration. *arXiv preprint arXiv:2608.11195*. <https://arxiv.org/abs/2608.11195>

Automation and Tuning — Creating a Custom Harness

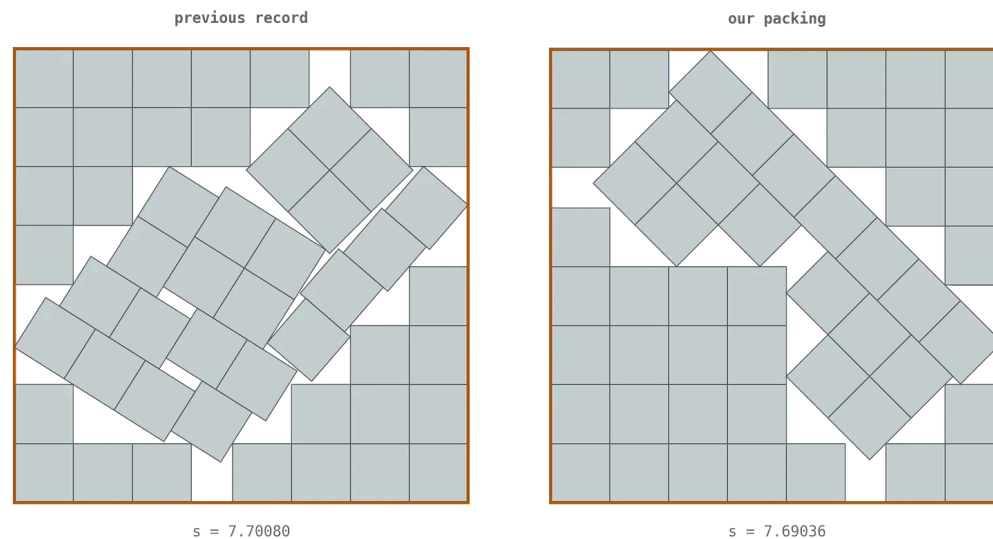
This paper is one of the rare papers on methodology. It explains how the authors organize coding agents and reasoning models using a “bulletin” and session reports.

This sort of automation is now within reach even for relatively nontechnical mathematicians.

Creating a Custom Harness

Another custom harness which improved the square packing lower bound.

<https://slightlysupervised.substack.com/p/improving-25-square-packing-upper>



Automation and Tuning — Creating a Custom Harness

This is admittedly more advanced and involved “program evolution” and requires more technical background. It broke around 20 existing records in about two days –apparently this was a weekend project.

Formalization

Formalization

Formalization is a broad topic which deserves a separate talk. I will mostly focus on Lean and its role in AI-assisted mathematics.

Formalization is becoming a part of mathematics, and LLMs are getting better at formalizing theorems from scratch and also helping mathematicians do formalization in a more interactive way.

It seems at least the ability to read Lean code is becoming a useful skill for mathematicians.

Why Formalize?

There are different motivations to formalize:

- the formalization itself is the goal,
- having a statement formalized is the goal.

Formalization — Why Formalize?

These are not mutually exclusive, of course. A formal proof is a very important kind of intermediary artefact in this context.

Formalization Itself as a Goal

In this case the end product is the code itself. The Lean library `mathlib` is an example of this. `mathlib` has a community, design standards, a review process, etc. The point is to *organize* mathematics. The formalization is meant to be read by humans. Such formalizations tend to contain custom tactics encoding some form of *insight*.

Kevin Buzzard's project on Fermat's Last Theorem is another example. OpenAI produced a formalization in a matter of days by brute force, but that did not make the project obsolete: its goal is the organized library, not just the proof.

— Formalization Itself as a Goal

A side issue: I think `mathlib`'s coverage of model theory is very weak – not even multi-sorted structures are supported.

To Have a Statement Formalized as a Goal

In this case the quality of the formalization is not important and the formalization is almost always generated by AI, sometimes by swarms of agents.

These formalizations can be “supporting evidence” when a big company drops a controversial proof. They signal that the proof is not “slop”. This kind of formalization also occurs as an “intermediary artifact” in a proof search (most likely by an AI) where progressing without a certificate of correctness is risky.

— To Have a Statement Formalized as a Goal

We usually do not learn much from these formalizations. Also, it is not always clear the natural language proof is translated faithfully. For instance, an erroneous natural language proof may be fixed silently by an AI during the translation process. There is a recent preprint claiming that the OpenAI proof of Navier-Stokes is mistranslated into Lean.

Interesting Pitfalls of Formalization

When you have a formal proof, you have a proof of *something*, not necessarily of what you think you proved. This is called the semantic gap. The problem shifts to checking definitions.

Theorems are proved from definitions, and the proofs can be verified. Definitions are the entry point, so there is nothing against which they can be verified.

Formalization

— Interesting Pitfalls of Formalization

The semantic gap already exists in traditional mathematics. *Proofs and Refutations* by Imre Lakatos is essentially about this gap.

For instance, Fermat's Last Theorem has almost no room for a semantic gap since the definitions are simple and the statement is elementary.

But there can be more mundane problems: Say you proved a deep theorem in algebraic geometry. Deep down, you should have a definition of a ring. Did you add $0 \neq 1$ to your assumptions?

So having a formal proof does not eliminate the need for auditing.

Boring Pitfalls of Formalization

Less likely, but still legitimate, reasons to doubt a formal proof:

- the underlying theory may be inconsistent (happened to Coq),
- the software may have bugs (recently happened to Lean).

However, I am not aware of any formalization withdrawn after an inconsistency or a bug. These tend to be amendable.

Lean as a Foundation

Lean is *not* set theory. Actually, one can build a model of ZFC inside Lean where each level in the universe hierarchy acts like a strongly inaccessible cardinal.

You can prove things in Lean that you cannot prove in ZFC, such as the existence of Grothendieck universes.

— Lean as a Foundation

Of course, this does not mean anything to most mathematicians. I am mentioning this because the audience consists of logicians.

A recent result by Mario Carneiro: $\text{CIC} + \text{EM} \vdash \text{Con}(\text{ZF})$. Here CIC is the calculus of inductive definitions, and EM is the excluded middle principle.

A Small Pedagogical Difference Between Set Theory and Lean

In Lean, *all* functions are total. The derivative *function* defined in Lean is total, so it is defined even at points where the original function is not differentiable. At those points the derivative function takes the value 0 by a convention of `mathlib`. For the Weierstrass function `w`, which is nowhere differentiable:

```
theorem weierstrass_deriv (x : ℝ) : deriv W x = 0
```

So instead of functions, one usually works with predicates. For derivatives, one works with `HasDerivAt f a x`, which says that `f` is differentiable at `x` with derivative `a`.

— A Small Pedagogical Difference Between Set Theory and Lean

A Big Pedagogical Difference Between Set Theory and Lean

In Lean, one can express a statement as a `Prop` or as a `Type`, which means that proof irrelevance is a choice you make. There are two notions of equality: definitional and propositional.

A more technical remark for the curious: `Prop` has no runtime counterpart; you cannot decide it at runtime without an explicit `Decidable` instance.

— A Big Pedagogical Difference Between Set Theory and Lean

I will not go into more details. The bottom line is that, if you want to be able to read Lean code, you should keep in mind that not everything can be translated to set theory in a straightforward way.

Formalizing Without Prior Art

If you decide to formalize a result, missing prior art will be an issue, especially if you are working in a niche field like model theory. Some standard theorem or definition you need will not be available. The ideal solution is to fill the gap by, say, contributing to `mathlib`, but that is usually a big and time-consuming project.

The non-ideal solution is to use opaque predicates and extra axioms. Axioms replace missing theorems and opaque predicates replace missing definitions. This may not always be feasible. And even if it is, there is the danger of introducing semantic gaps.

— Formalizing Without Prior Art

We will see examples of opaque predicates and extra assumptions. The choice of these is usually important, and even if you leave the decision to an AI assistant, it is important to understand the generated proof.

The Role of `mathlib`

There are two important questions one should ask about `mathlib`:

- *Is `mathlib` trustworthy?*

Yes. It is heavily used and reviewed, so a semantic gap in it is very unlikely to go unnoticed.

- *Is it the only option?*

Essentially yes: it is the one general mathematics library for Lean. This may sound like a disadvantage, but in reality it is not.

— The Role of `mathlib`

These points were raised by Piotr during a private conversation.

Lean is software, so we should judge it as software. Software development teaches us that battle-testing is very powerful, meaning heavily used software tends to have fewer unknown bugs. `mathlib` has a big community, and it is used in many big projects. Having one `mathlib` avoids having an *ecosystem fragmentation*. If we had a `mathlib2`, it would be very difficult to use code relying on `mathlib2` in `mathlib` or vice versa. Having the same underlying language is not enough.

Case Study 1: A Minimal Type of Infinite Transcendence Degree

Case Study 1: A Minimal Type of Infinite Transcendence Degree

This was an open problem in the theory of differentially closed fields with several commuting derivations. When the AI-assisted proofs boomed, I tried my hand at it just to see what would happen. Frankly, I was expecting the project to fail, but it didn't and here we are.

Rank Inequalities in Differentially Closed Fields

We will be working in the theory of differentially closed fields of characteristic zero with several commuting derivations. For $p \in S_n(K)$, where K is a differential field, define

$$\chi_p(n) = \text{tr-deg}_K K(\theta(\eta) : \theta \text{ a monomial in the derivations, } \text{ord}(\theta) \leq n)$$

to be the dimension function of p where η is a realization of p .

It is a classical fact that $\chi_p(n)$ is given by a polynomial for large values of n .

Case Study 1: A Minimal Type of Infinite Transcendence Degree — Rank Inequalities in Differentially Closed Fields

Rank Inequalities in Differentially Closed Fields

McGrail proved the following relation in her thesis:

$$U(p) < \omega^\tau(a_\tau + 1),$$

where τ is the degree of the dimension polynomial and a_τ is the leading coefficient in the binomial basis.

She also claimed that

$$\omega^\tau a_\tau \leq U(p).$$

This turned out to be false.

Case Study 1: A Minimal Type of Infinite Transcendence Degree — Rank Inequalities in Differentially Closed Fields

A Linear Counterexample

Proposition (S.). Consider the case of 2 derivations, say δ_1 and δ_2 . For a linear δ_2 -operator L define p_L to be the type of the equation

$$\delta_1 y = L(y).$$

Then $U(p_L) = \omega$ yet $\tau = 1$ and $a_\tau = \text{ord}(L)$, the order of L in δ_2 .

This proves that McGrail's claim about the lower bound was false, but it does not rule out a lower bound of the form ω^τ .

Case Study 1: A Minimal Type of Infinite Transcendence Degree — A Linear Counterexample

A Nonlinear Counterexample

León Sánchez asked whether

$$\delta_1 x = x^3 + c, \quad c \text{ generic},$$

has finite U -rank in the case of 2 derivations. Note that it has infinite transcendence degree, because nothing constrains δ_2 , so τ is 1.

The answer turns out to be yes. Actually, Freitag, Kowalski and León Sánchez prove that all equations in a certain class, including the one above, are strongly minimal.

By sheer coincidence, around the same time I also found a proof using AI.

Case Study 1: A Minimal Type of Infinite Transcendence Degree — A Nonlinear Counterexample

Methodology

The problem is to show that there is no intermediate locus. Consider a solution a of $\delta_1 x = x^3 + c$ and its jets in the δ_2 -direction:

$$x_0 = a, \quad x_1 = \delta_2 a, \quad x_2 = \delta_2^2 a, \quad \dots$$

A sensible first step is to understand the action of δ_1 on the jets x_k ? Since the derivations commute,

$$\delta_1 x_k = \delta_2^k (\delta_1 x_0) = \delta_2^k (x_0^3 + c).$$

Let us expand this more.

Case Study 1: A Minimal Type of Infinite Transcendence Degree — Methodology

Methodology

I could have done the computation by hand, but I asked Claude to write a symbolic differential algebra library built on SymPy and made a few computations:

Experiment. Compute $\delta_1 x_k$ for $k = 1, \dots, 4$. Writing $c_{0k} = \delta_2^k c$:

$$\delta_1 x_1 = 3x_0^2 x_1 + c_{01},$$

$$\delta_1 x_2 = 3x_0^2 x_2 + 6x_0 x_1^2 + c_{02},$$

$$\delta_1 x_3 = 3x_0^2 x_3 + 18x_0 x_1 x_2 + 6x_1^3 + c_{03},$$

$$\delta_1 x_4 = 3x_0^2 x_4 + 24x_0 x_1 x_3 + 18x_0 x_2^2 + 36x_1^2 x_2 + c_{04}.$$

Case Study 1: A Minimal Type of Infinite Transcendence Degree — Methodology

Methodology

This revealed a pattern, namely at every level

$$\delta_1 x_k = 3x_0^2 \cdot x_k + (\text{terms in } x_0, \dots, x_{k-1}) + \delta_2^k c,$$

with the *same multiplier* $3x_0^2 = f_x$ in front of the new jet.

This became a lemma of the paper, later proved by hand, and helped prove propositions uniform in k .

Case Study 1: A Minimal Type of Infinite Transcendence Degree — Methodology

Note that this could have been done by hand easily, but Claude did it in a matter of minutes. The only token cost was building the software; running the experiments themselves used none. And the software was reused for other experiments, so its cost is marginal. There were many other experiments and/or symbolic computations.

I asked the assistant to at least computationally verify (to a certain extend) any claim without using it and also keep a ledger.

Since this was the first project I did, it is not very tidy so I did not put it on GitHub however I can share it privately if you are interested.

Case Study 2: Generic Differential Polynomials

<https://github.com/sonatsuer/generic-diff-eq-are-strongly-minimal-by-ai>

Case Study 2: Generic Differential Polynomials

The second case study is a genuinely new result in the model theory of differentially closed fields answering a question asked by DeVilbiss and Freitag. By today's standards it could be published, but I expect the standards to change pretty soon. Again, the point of this project is to give an example of a workflow where symbolic computation can guide AI in hypothesis testing.

Statement of the Problem

Let $(\mathcal{U}, \delta)$ be a saturated differentially closed field of characteristic 0. Fix $h \geq 1$, $d \geq 2$, and let $\mathcal{M}(h, d)$ be the set of monomials of total degree at most d in $x, x', \dots, x^{(h)}$.

Conjecture (DeVilbiss–Freitag). Let $f = \sum_{\mu \in \mathcal{M}(h, d)} \alpha_{\mu} \mu$ with $\alpha_{\mu} \in \mathcal{U}$ independent differential transcendentals over $\mathbb{Q}$. Then $Z(f) = \{a \in \mathcal{U} : f(a) = 0\}$ is strongly minimal.

DeVilbiss and Freitag proved it for $d \geq 6$.

Case Study 2: Generic Differential Polynomials — Statement of the Problem

$d \geq 2$ is optimal: for $d = 1$ the equation is linear, $Z(f)$ is a $\mathbb{C}$ -vector space of dimension h , where $\mathbb{C}$ is the field of constants, and so it is not strongly minimal for $h > 1$.

Methodology

Let f be a differential polynomial with coefficients $\bar{a}$. For a solution a of $f = 0$, the *linearization* of f at a is $L_a = \sum_{i=0}^h \frac{\partial f}{\partial x^{(i)}}(a) \partial^i$. Let p be the generic type of $Z(f)$.

Theorem (DeVilbiss–Freitag). Let $m \geq \text{nmdeg}(p) + 1$. Suppose that for every tuple $(a_1, \dots, a_m)$ of solutions of $f = 0$, algebraically independent over $\mathbb{Q}\langle\bar{a}\rangle$, the linear system

$$L_{a_1} z_1 = L_{a_2} z_2 = \dots = L_{a_m} z_m$$

has no definable proper infinite-rank $\mathcal{C}$ -subspace of solutions (the *subspace condition* (S)). Then $Z(f)$ is strongly minimal.

Since $\text{nmdeg}(p) \leq 2$ (Freitag–Jaoui–Moosa), $m = 3$ suffices.

Case Study 2: Generic Differential Polynomials — Methodology

$\mathcal{C}$ is the field of constants, and nmdeg is the Freitag–Moosa degree of non-minimality. DeVilbiss and Freitag state the theorem for the differential tangent space of an auxiliary variety; eliminating one variable turns it into the linear system on the slide.

Methodology

Let $K = \mathbb{Q}\langle \bar{a}, \bar{a} \rangle$ and let the *system matrix* M be the $(m-1) \times m$ matrix of the system over the Ore algebra $K[\partial]$: row k has L_{a_1} in column 1, $-L_{a_{k+1}}$ in column $k+1$, and zeros elsewhere.

Proposition, proved by AI. Assume $L_{a_2}, \dots, L_{a_m} \neq 0$. The subspace condition (S) holds *if and only if* every invariant factor of the Jacobson normal form of M is a unit of $K[\partial]$ (the *Jacobson criterion* (J)). For $m=2$ the condition is $\text{gcd}(L_{a_1}, L_{a_2}) = 1$.

The Jacobson criterion (J) makes symbolic experimentation easy.

Case Study 2: Generic Differential Polynomials — Methodology

DeVilbiss and Freitag verify (S) through a sufficient condition, the genericity condition (G) on the next slides, which requires $d \geq 2m = 6$. The Jacobson criterion (J) is equivalent to (S), and decidable by the Euclidean algorithm. Claude proved it after I asked it to derive necessary and sufficient conditions.

This went into a ledger where I keep all statements that came up while trying to solve the problem, and it was marked as proven.

Methodology

The Jacobson criterion (J) is essentially an invertibility condition on the system matrix M and can be checked by computing minors. Claude computed *a lot* of minors. For this, it developed (by my prompt) a custom symbolic algebra library based on SymPy to do linear algebra over an Ore domain. The experiments resulted in explicit conjectures with overwhelming evidence, and then Claude gave actual proofs.

I will give details about such a computation, but there were many more. These are all documented in the project repository.

Case Study 2: Generic Differential Polynomials — Methodology

In the end, no proof depended on the symbolic computations carried out by the custom software, but they guided the research.

Experiment

DeVilbiss and Freitag check the subspace condition (S) through a sufficient condition, the *genericity condition* (G): the coefficients $\beta_{i,j} = \frac{\partial f}{\partial x^{(i)}}(a_j)$ of the linearizations should be independent differential transcendentals. Each $\beta_{i,j}$, and each constraint $f(a_j) = 0$, is *linear* in the coefficients of f . Using the fact that those coefficients are generic, one can reduce (G) to linear algebra: the *genericity matrix* G , of size $m(h+2) \times N$, with rows

$$v_j = \left(\mu(a_j) \right)_{\mu} \quad \text{and} \quad w_{i,j} = \left(\frac{\partial \mu}{\partial x^{(i)}}(a_j) \right)_{\mu}$$

indexed by the N monomials μ , must have full rank.

Case Study 2: Generic Differential Polynomials — Experiment

Experiment

So I asked Claude to do some computations to see if the genericity matrix G had indeed full rank for small values.

Experiment. The rank was computed for $h = 2, \dots, 5$, $d = 2, \dots, 7$, $m = 2, 3$.

Result. Full rank for every $d \geq 3$. At $d = 2$ the rank is short.

So the genericity condition (G) fails at $d = 2$, for every h : the DeVilbiss–Freitag method cannot reach $d = 2$. This says nothing about (S): at $d = 2$ the Jacobson criterion (J) still holds, and the paper proves the case $d = 2$ through (J).

Case Study 2: Generic Differential Polynomials — Experiment

A technical remark: The computation was done modulo a prime, at random integer points, to keep it fast and exact. A full rank mod p implies full rank over the rationals, so this is OK.

The table resulting from the experiment is recorded and later it was marked as evidence while making conjectures.

Formalization

Formalization is done by Claude. However, it is not a *clean* formalization in the sense that it has opaque predicates and non-standard axioms. Basically, everything about model theory is assumed due to the weak `mathlib` support. The algebra is mostly proven on top of `mathlib`, which already had decent algebra support.

Here are examples of opaque predicates used in the formalization:

```
opaque StronglyMinimal (Z : Set U) : Prop
opaque IsSaturatedDCF0 (U : Type*) [Field U] [Differential U] : Prop
```

Case Study 2: Generic Differential Polynomials — Formalization

Formalization

And here is an axiom the formalization had to assume:

```
axiom criterion_implies_stronglyMinimal
  (hU : IsSaturatedDCF0 U)
  (hh : 1 ≤ h) (hd : 2 ≤ d) (α : Coeffs h d U)
  (horder : HasOrder α) (hnonlin : IsNonlinear α)
  (hconst : ConstCoeffTranscendental α) (hirr : AbsIrreducible α)
  (hcrit : ∀ a :
    Fin 3 → U, Admissible α a → OrePoly.IsRightInvertible (Msys α a)) :
    StronglyMinimal (Zeros α)
```

Case Study 2: Generic Differential Polynomials — Formalization

The hypothesis `hcrit` is the Jacobson criterion (J), in the equivalent form “ M is right invertible”.

The axioms originally generated semantic gaps by mistake. One axiom was not even satisfiable, rendering the whole proof invalid. This was fixed by Claude after an audit.

Formalization also found small issues and Claude fixed them in its write-up.

None of the errors found during the process were fatal. They were fixed with minor tweaks.

Case Study 3: Visual Proofs to Lean Proofs

<https://github.com/sonatsuer/visual-proof-engine>

Case Study 3: Visual Proofs to Lean Proofs

The third project is a toy project in recreational mathematics about visual proofs in algebra –after Piotr reminded me politely that I might accidentally solve someone’s PhD problem. It demonstrates what a nontrivial pipeline can look like.

The ideas in the previous studies are easy for most mathematicians to grasp and adopt: each is a familiar research process, automated by AI and AI-generated tools. This project is of a different kind. Its purpose is to showcase some nontrivial and new ideas and give you a glimpse of what else is possible. I would also argue that the end result is a mathematical object and producing it was a mathematical activity.

Project Description

This project is a pipeline which generates Lean proofs and animations from visual proof goals. The actual mathematical content is recreational in nature. It generates visual proofs, or “proofs without words” as some people call them. The point is the pipeline itself.

The pipeline has no step using AI. However, the automation software making up the pipeline is produced by Claude. It depends on nontrivial mathematical facts like cell decomposition for Presburger arithmetic, or some basic insight encoded as Lean tactics.

Case Study 3: Visual Proofs to Lean Proofs — Project Description

A pipeline as the end product of a project is something alien to mathematicians. But bear with me.

An Example

Here is an example of a goal:

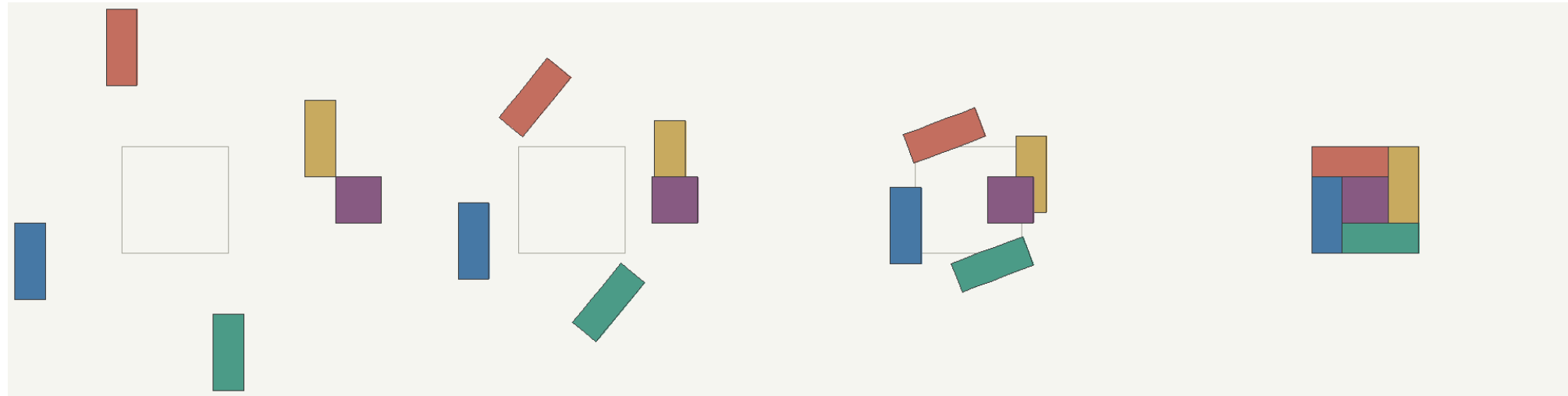
```
{
  "goal": "tilebox", "name": "pinwheel", "dim": 2,
  "params": ["a", "b"],
  "preconditions": ["a >= 1", "b >= a + 1"],
  "pieces": [
    {"shape": {"kind": "box", "lo": ["0", "0"], "len": ["a", "b"]},
      "multiplicity": 4, "name": "ab"},
    {"shape": {"kind": "box", "lo": ["0", "0"], "len": ["b - a", "b - a"]},
      "name": "(b-a)^2"}
  ],
  "box": ["a + b", "a + b"]
}
```

Case Study 3: Visual Proofs to Lean Proofs — An Example

The goal: $(a + b)^2 = 4ab + (b - a)^2$ for $b > a \geq 1$. Four $a \times b$ rectangles and one $(b - a) \times (b - a)$ square must tile the $(a + b) \times (a + b)$ square.

An Example

The animation it produces, rendered at $a = 2, b = 5$:



Case Study 3: Visual Proofs to Lean Proofs — An Example

Search found this pinwheel in about 8 seconds, and Lean proved the tiling for every $b > a \geq 1$ by checking a single small case, $a = 1, b = 2$, and a theorem that transfers it to all allowed (a, b) , in 2.5 seconds. The animation is drawn from the certificate that Lean checks. The image shows four key frames but the actual output is an animation. So we have a proof that this visual proof works *and* it proves the correct algebraic identity.

An Example

Lean reads the certificate and generates a theorem saying that the pieces tile the square. The algebraic identity then follows by counting cells on both sides.

```
theorem pinwheel_nat (a b : ℕ) (ha : 1 ≤ a) (hab : a < b) :
  ((a : ℤ) + b) ^ 2 = 4 * a * b + ((b : ℤ) - a) ^ 2 := by
  have h := pinwheel_card (fun v => if v = "a" then (a : ℤ) else if v = "b" then
(b : ℤ) else 0)
    (by simp; omega) (by simp; omega)
  simp at h
  rw [sq, sq, mul_assoc 4]
  exact h
```

Case Study 3: Visual Proofs to Lean Proofs — An Example

`pinwheel_card` is the counting theorem; its proof takes the generated theorem `pinwheel.tiling` and applies `card_eq`.

An Example

The theorem generated by elaboration from a certificate:

```
pinwheel.tiling : ∀ (ρ : Env),
  ρ "a" ≥ 1 → ρ "b" ≥ ρ "a" + 1 →
    Tiling
      [(boxAt ![var "a", var "b"]).denote ρ,
        (place pinwheel.g0 (boxAt ![var "a", var "b"])).denote ρ,
        (place pinwheel.g1 (boxAt ![var "a", var "b"])).denote ρ,
        (place pinwheel.g2 (boxAt ![var "a", var "b"])).denote ρ,
        (place pinwheel.g3 (boxAt ![(var "b").add (var "a").neg,
                                          (var "b").add (var "a").neg])).denote ρ]
      ((boxAt ![(var "a").add (var "b"), (var "a").add (var "b")]).denote ρ)
```

Case Study 3: Visual Proofs to Lean Proofs — An Example

ρ is an assignment of integer values to the parameters a and b . `boxAt` is a box with the given side lengths at the origin, `var "a"` is the parameter a , `place g` moves a shape by the rigid motion g (a rotation or reflection of the grid followed by a translation), and `denote ρ` is the set of cells a shape occupies at the values ρ . The motions $g_0, \dots, g_3$ were found by the search, not written by anyone. `pinwheel_card`, not shown, derives the identity over $\mathbb{Z}$ from the tiling by counting cells on both sides. The whole tiling is proved by one check at $a = 1, b = 2$ plus a theorem that transfers it to every allowed (a, b) .

Pipeline

The user states what is to be proved as a *goal*, written in JSON. This goal can be handwritten or be generated as the output of another mechanism.

Solve small cases.

Input: the goal at a few concrete parameter values.

Output: all tilings at those values.

Tool: an exact cover solver (dancing links, or the SAT solver CaDiCaL). It tries every placement of every piece, rotated, reflected and translated, and finds the choices that cover each cell of the target exactly once.

Case Study 3: Visual Proofs to Lean Proofs — Pipeline

This corresponds to what most mathematicians do with pen and paper, though more systematically and exhaustively.

Pipeline

The small cases serve as “inspiration” for the actual construction.

Generalize.

Input: the concrete tilings.

Output: a candidate *certificate*: a JSON file listing the pieces and how each one is moved into place, with the moves given as functions of the parameters.

Tool: a Python script. It matches the same piece across the different values and fits its position as a linear function of the parameters.

Case Study 3: Visual Proofs to Lean Proofs — Pipeline

The candidate certificate is essentially a conjecture.

Pipeline

Add a coverage witness.

Input: the certificate.

Output: the certificate with a witness: for each cell of the target, which piece covers it.

Tool: a Python script that reads it off the shapes. Lean needs it because “every cell is covered by some piece” is a $\forall\exists$ statement, which Lean’s arithmetic tactic cannot decide; with the witness, each case is plain linear arithmetic.

Case Study 3: Visual Proofs to Lean Proofs — Pipeline

This is a technical step. I included it because you *will* face technical issues if you start using technology.

Pipeline

Falsify.

Input: the certificate.

Output: either “no counterexample”, or the smallest parameter values at which it fails, in which case the candidate is dropped.

Tool: a Python checker that evaluates the tiling exactly at small parameter values. It is cheap and runs before anything reaches Lean.

Case Study 3: Visual Proofs to Lean Proofs — Pipeline

This is a final sanity check to avoid wasting time.

Pipeline

Verify in Lean.

Input: the certificate.

Output: a Lean theorem “for all parameters satisfying the preconditions, the pieces tile the target”.

Tools: a custom elaborator stating the theorem. Then either Lean tactics (both standard and custom) prove it, or an SMT solver splits the parameter space into cases that Lean checks one by one. Either way, Lean’s kernel re-checks the final proof.

Case Study 3: Visual Proofs to Lean Proofs — Pipeline

An elaborator generates a theorem statement from an external object.

Pipeline

Render.

Input: the verified certificate.

Output: an animation of the pieces moving into place.

Tool: a Python renderer. Before drawing, it re-checks that the picture matches the certificate, so the animation cannot disagree with what Lean proved.

Case Study 3: Visual Proofs to Lean Proofs — Pipeline

Thank you for your attention!

Thank you for your attention!

—