

The SorryDB project

Lenny Taelman

University of Amsterdam

Autoformalization for the working mathematician

ICERM, April 2025

work-in-progress with Austin Letson and Oliver Dressler

<https://github.com/SorryDB/SorryDB>

Introduction

About me

- «Working mathematician» (in algebraic geometry)
- Used to build experimental chess engines as an undergrad
- Tenured → can afford to try something different

Neural tools in formal mathematics

Dream: if neural provers help bring the formal overhead factor below 1, then formal methods may become a transformative tool in mathematical research.

Neural tools in formal mathematics

Dream: if neural provers help bring the formal overhead factor below 1, then formal methods may become a transformative tool in mathematical research.

More immediately: if neural provers can fill some mathematically «trivial» proofs, formalization may become more accessible.

Neural tools in formal mathematics

Dream: if neural provers help bring the formal overhead factor below 1, then formal methods may become a transformative tool in mathematical research.

More immediately: if neural provers can fill some mathematically «trivial» proofs, formalization may become more accessible.

Reality check: today a *tiny* fraction of mathematicians use formal methods, and a *tiny* fraction of those mathematicians have adopted neural tools (beyond copilots).

Lack of adoption

Possible reasons for lack of adoption by mathematicians:

- Skepticism about capabilities
- Frustration with technical hurdles
- Inconvenience of integrating tools in existing workflow

Incentives of computer scientists do not always align with needs of mathematicians.

See also: Jason Rute's *The last mile* at Lean Together 2025.

Two cultures

1. *Mathematicians* interested in using proof automation
2. *Computer scientists* designing new algorithms and models

~~Two~~ Three cultures

1. *Mathematicians* interested in using proof automation
2. *Computer scientists* designing new algorithms and models
3. *Makers* building tools, reproducing papers, ...

Two Three cultures

1. *Mathematicians* interested in using proof automation
2. *Computer scientists* designing new algorithms and models
3. *Makers* building tools, reproducing papers, ...

In computer chess, *makers* reproduced AlphaZero in a large collaborative effort (cf. *Leela Zero*).

Hundreds of *makers* compete in the Kaggle AIMO competitions in «competition» mathematics.

Two Three cultures

1. *Mathematicians* interested in using proof automation
2. *Computer scientists* designing new algorithms and models
3. *Makers* building tools, reproducing papers, ...

In computer chess, *makers* reproduced AlphaZero in a large collaborative effort (cf. *Leela Zero*).

Hundreds of *makers* compete in the Kaggle AIMO competitions in «competition» mathematics.

The *SorryDB project* builds *incentives* and *infrastructure* for makers to help bring automated theorem proving to «research» mathematics.

Sorries

What is a sorry?

Lean has a «forbidden» axiom stating that every statement has a proof:

```
axiom sorryAx :  $\forall P : \text{Prop}, \exists p : P$ 
```

What is a sorry?

Lean has a «forbidden» axiom stating that every statement has a proof:

```
axiom sorryAx :  $\forall P : \text{Prop}, \exists p : P$ 
```

The tactic `sorry` applies this axiom and is often used to temporarily admit a statement, deferring the proof to later.

```
theorem not_now (x : Nat) : x + 1 > x := by  
  sorry
```

What is a sorry?

Lean has a «forbidden» axiom stating that every statement has a proof:

```
axiom sorryAx :  $\forall P : \text{Prop}, \exists p : P$ 
```

The tactic `sorry` applies this axiom and is often used to temporarily admit a statement, deferring the proof to later.

```
theorem not_now (x : Nat) : x + 1 > x := by  
  sorry
```

Also for sub-statements:

```
theorem not_now (x : Nat) : x + 1 > x := by  
  induction x with  
  | zero => exact Nat.zero_lt_succ 0  
  | succ y ih => rw [Nat.add_assoc]; sorry
```

Examples in the wild: 1/3

Sample sorry from the *Fermat Last Theorem* project:

```
theorem GL2.localFullLevel.isCompact
  (v : HeightOneSpectrum (O F)) :
  IsCompact (GL2.localFullLevel v).carrier := by
  sorry
```

Examples in the wild: 1/3

Sample sorry from the *Fermat Last Theorem* project:

```
theorem GL2.localFullLevel.isCompact
  (v : HeightOneSpectrum (O F)) :
  IsCompact (GL2.localFullLevel v).carrier := by
  sorry
```

Translation: let F be a number field, v a finite place, then the subgroup $GL_2(\mathcal{O}_{F_v})$ of the topological group $GL_2(F_v)$ is compact.

Examples in the wild: 2/3

Sample sorry from the *Cambridge combinatorics* project:

```
lemma q13 [Finite  $\alpha$ ] (g :  $\beta \hookrightarrow \alpha$ ) (f :  $\beta \rightarrow \alpha \rightarrow \alpha$ )
  (hf : IsLatin f) :
   $\exists f', f' \circ g = f \wedge \text{IsLatin } f := \text{by}$ 
sorry
```

Examples in the wild: 2/3

Sample sorry from the *Cambridge combinatorics* project:

```
lemma q13 [Finite  $\alpha$ ] (g :  $\beta \hookrightarrow \alpha$ ) (f :  $\beta \rightarrow \alpha \rightarrow \alpha$ )
  (hf : IsLatin f) :
   $\exists f', f' \circ g = f \wedge \text{IsLatin } f := \text{by}$ 
  sorry
```

Translation: any latin rectangle can be extended to a latin square.

Examples in the wild: 3/3

From the *Carleson project*:

```
lemma dist_dirichletApprox_le {f : ℝ → ℂ} {x : ℝ}
  (hf : MemLp f ∞volume)
  (periodic_f : f.Periodic (2 * π))
  {r : ℝ} (hr : r ∈ Ioo 0 1)
  {n : ℕ} (hn : n = ⌈r-1⌉) :
  dist (dirichletApprox n x)
    ({y : ℂ | ‖y‖ ∈ Ioo r 1}.indicator 1 x) ≤
  2 ^ (5 : ℝ) * niceKernel r x := by
```

sorry

Examples in the wild: 3/3

From the *Carleson project*:

```
lemma dist_dirichletApprox_le {f : ℝ → ℂ} {x : ℝ}
  (hf : MemLp f ∞volume)
  (periodic_f : f.Periodic (2 * π))
  {r : ℝ} (hr : r ∈ Ioo 0 1)
  {n : ℕ} (hn : n = ⌈r-1⌉) :
  dist (dirichletApprox n x)
    ({y : ℂ | ‖y‖ ∈ Ioo r 1}.indicator 1 x) ≤
  2 ^ (5 : ℝ) * niceKernel r x := by
  sorry
```

Translation: small step in establishing that some operator is bounded.

The SorryDB project

The SorryDB project

GitHub repositories of ongoing formalization projects are littered with sorried statements.

Each such «sorry» is a real-world challenge for an automated theorem prover. Being able to fill such a sorry is almost by definition useful to some mathematician.

The SorryDB project

GitHub repositories of ongoing formalization projects are littered with *sorry* statements.

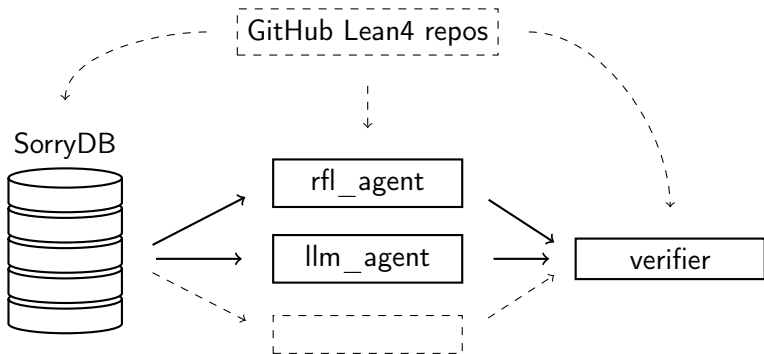
Each such «*sorry*» is a real-world challenge for an automated theorem prover. Being able to fill such a *sorry* is almost by definition useful to some mathematician.

We scrape *sorry*s from public repositories, and build the infrastructure to make it as easy as possible to develop, compare, and use neural theorem provers on those *sorry*s.

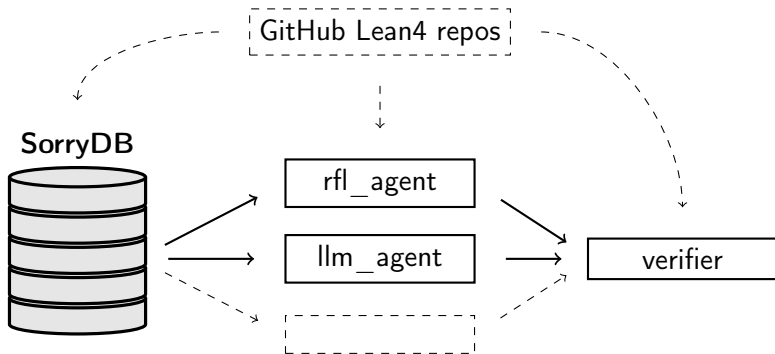
See also:

- *LeanAgent* by Kumarappan, Tiwari, et. al.
- *miniCTX* by Hu, Zhu, Welleck
- *SWE-bench* Jimenez, Yang, et. al.

What is built?



What is built?



Database

The database consists of JSON files with coordinates of verified (Prop-valued) sorries in public repositories. It contains all information needed to reproduce those sorries locally.

Updated nightly:

- Cumulative list of sorries
- Deduplicated list of sorries

Also includes:

- Static lists of sorries for development and testing

See <https://github.com/SorryDB/sorrydb-data>

Sample SorryDB item

```
{
  "repo": {
    "remote": "https://github.com/ImperialCollegeLondon/FLT",
    "branch": "quat-alg-weight-2",
    "commit": "00bee2838ab99689836c6396ea310e0bedfbcddd",
    "lean_version": "v4.18.0-rc1"
  },
  "location": {
    "path": "FLT/GlobalLanglandsConjectures/GLzero.lean",
    "start_line": 127,
    "start_column": 19,
    "end_line": 127,
    "end_column": 24
  },
  "debug_info": {
    "goal": "z : C\\nn : N\\nρ : Weight n\\nhρ : ρ.IsTrivial\\n⊢ IsSmooth fun x => z",
    "url": "https://github.com/ImperialCollegeLondon/FLT/blob/00bee2838ab99689836c6396ea310e0bedfbcddd/FLT/GlobalLanglandsConjectures/GLzero.lean#L127"
  },
  "metadata": {
    "blame_email_hash": "1679c78ca90b",
    "blame_date": "2024-06-12T14:33:04+02:00",
    "inclusion_date": "2025-03-14T12:00:00+00:00"
  },
  "id": "a7f9b3c5d2e1"
}
```

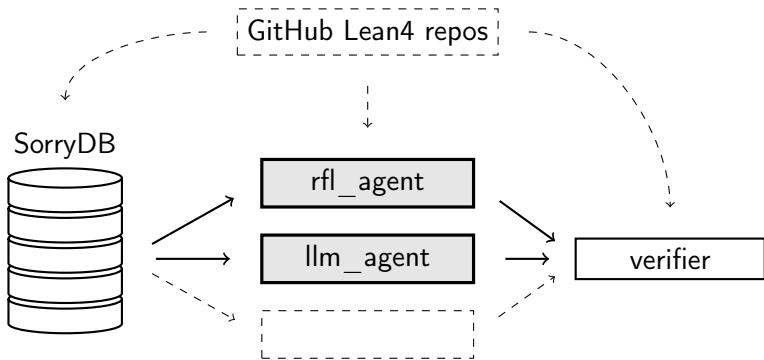
link

Statistics

As of April 23, 2025 the database

- contains 936 sorries with unique pretty-printed goal
- contains sorries from 19 repositories
- adds about 10 sorries with a new blame date per day
- takes about 2 hours to update

What is built?



Agents

«Sorry-filling» agents

- take a list of SorryDB entries as input
- reproduce the sorries locally (`git clone`, `lake build`)
- attempt to prove them
- output list of proof strings

Agents

«Sorry-filling» agents

- take a list of SorryDB entries as input
- reproduce the sorries locally (`git clone`, `lake build`)
- attempt to prove them
- output list of proof strings

Two sample agents have been implemented:

- `rfl_agent`: tries the tactic `rfl`
- `llm_agent`: tries one-shot LLM-generated proof

Agents

«Sorry-filling» agents

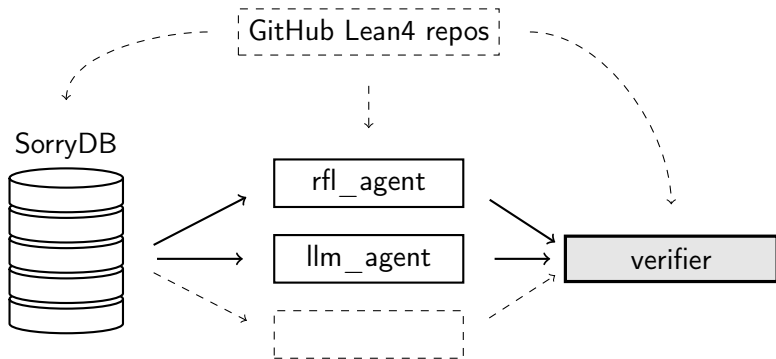
- take a list of SorryDB entries as input
- reproduce the sorries locally (`git clone`, `lake build`)
- attempt to prove them
- output list of proof strings

Two sample agents have been implemented:

- `rfl_agent`: tries the tactic `rfl`
- `llm_agent`: tries one-shot LLM-generated proof

Not for consumption, but basis for developing proper agents.

What is built?



Verifier

The verifier takes as input:

1. An entry of the sorryDB
2. A candidate proof string (replacing "sorry")

... and verifies that the candidate proof string

1. indeed fills the sorry
2. does not break anything else

Some technical comments

- We use the Lean reservoir as list of repositories to scrape.
- Requires a little bit of manual curation.
- We clone and build repositories locally, and extract the Lean version from `toolchain`.
- We use the corresponding Lean REPL version to extract sorry statements, and to verify that their target is of type `Prop`.
- `llm_agent` uses `langchain` and can work with many models (bring your own API key)
- The verifier uses Lean REPL again to check that the code compiles with the candidate proof string, and that no new sorries are introduced.

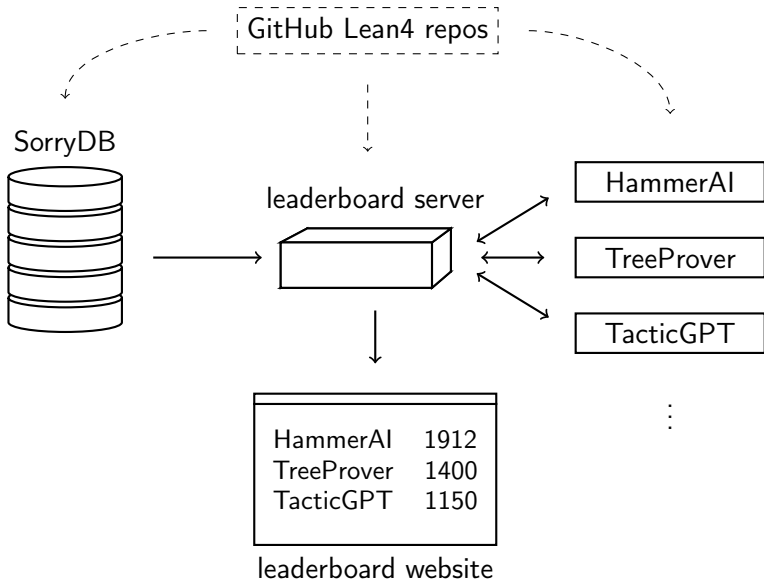
What is next?

Future plans

Next steps, in rough order of priority:

- Scale up scraping
- Build some «serious» agents: symbolic, LLM-based, . . .
- Reproduce one of the SOTA provers in an agent
- Run «static» competitions between agents
- Set up a continuous sorry-filling competition and leaderboard

Future competition ecosystem



Relative rating

Absolute performance («11% solved») is not so relevant:

- database is very inhomogeneous, no attempt to curate
- nature of typical sorries may evolve over time
- many sorries are unprovable: missing hypotheses, sorried definitions, ...

Relative rating

Absolute performance («11% solved») is not so relevant:

- database is very inhomogeneous, no attempt to curate
- nature of typical sorries may evolve over time
- many sorries are unprovable: missing hypotheses, sorried definitions, ...

Rather, we propose to focus on relative performance:

- agents poll server for sorries to fill
- server selects random recent sorries
- update ratings if one competitor succeeds and another fails

Future plans

Next steps, in rough order of priority:

- Scale up scraping
- Build some «serious» agents: symbolic, LLM-based, . . .
- Reproduce one of the SOTA provers in an agent
- Run «static» competitions between agents
- Set up a continuous sorry-filling competition and leaderboard

Future plans

Next steps, in rough order of priority:

- Scale up scraping
- Build some «serious» agents: symbolic, LLM-based, . . .
- Reproduce one of the SOTA provers in an agent
- Run «static» competitions between agents
- Set up a continuous sorry-filling competition and leaderboard

You can help! For example:

- Build a simple agent using your favourite Lean interaction tool
- Test your favourite model on sorries from the SorryDB
- Help us build and maintain a stable, and easy-to-use ecosystem
- Give us feedback!

Questions

- **Other uses** for the database? How can we support them?
- **Lean interaction tools?** Which ones are good candidates for basing agents on?
- **Theorem proving models?** Which ones are good candidates to reproduce in an agent?
- **Obstacles?**