UC SANTA CRUZ

On Proofs and Progress

Eoin Mackall

October 28, 2026

University of California, Santa Cruz

Table of Contents

1. Introduction
   - Background
2. Proofs
   - Introduction
   - Inference
   - Design
   - Verification
   - Refutation and repair
   - Future Directions

Introduction

Background

Set-up:

- Let X be a smooth and projective complex variety.
- Let $\mathcal{A}$ be an Azumaya algebra on X (a flat coherent $\mathcal{O}_X$-algebra whose fibers are matrix algebras).
- There is a scheme $\pi : \mathrm{SB}(\mathcal{A}) \to X$ (points in a fiber correspond to minimal left ideals in $\mathcal{A}$) called *the Severi–Brauer scheme of* $\mathcal{A}$.

Background

Set-up:

- Let X be a smooth and projective complex variety.
- Let $\mathcal{A}$ be an Azumaya algebra on X (a flat coherent $\mathcal{O}_X$-algebra whose fibers are matrix algebras).
- There is a scheme $\pi : \mathrm{SB}(\mathcal{A}) \to X$ (points in a fiber correspond to minimal left ideals in $\mathcal{A}$) called *the Severi–Brauer scheme of* $\mathcal{A}$.

Definition

A Severi–Brauer scheme $\mathrm{SB}(\mathcal{A})$ *as above is* **topologically trivial** *if it is homeomorphic over* X *to* $\mathbb{P}(V)$ *for a complex topological vector bundle* V *over* X.

Background

Perry and de Jong observed that for topologically trivial Severi–Brauer schemes, there is an isomorphism

$$\theta^* : \mathrm{H}^*(\mathrm{SB}(\mathcal{A}), \mathbb{Z}) \xrightarrow{\sim} \mathrm{H}^*(\mathbb{P}(V), \mathbb{Z}) \cong \bigoplus_{i=0}^{\mathrm{rk}(V)-1} \mathrm{H}^{*-2i}(X, \mathbb{Z}),$$

induced by a homeomorphism $\theta : \mathbb{P}(V) \to \mathrm{SB}(\mathcal{A})$.

Background

Perry and de Jong observed that for topologically trivial Severi–Brauer schemes, there is an isomorphism

$$\theta^* : \mathrm{H}^*(\mathrm{SB}(\mathcal{A}), \mathbb{Z}) \xrightarrow{\sim} \mathrm{H}^*(\mathbb{P}(V), \mathbb{Z}) \cong \bigoplus_{i=0}^{\mathrm{rk}(V)-1} \mathrm{H}^{*-2i}(X, \mathbb{Z}),$$

induced by a homeomorphism $\theta : \mathbb{P}(V) \to \mathrm{SB}(\mathcal{A})$.

If $[\mathcal{A}] \in \mathrm{Br}(X)[n] \twoheadleftarrow \mathrm{H}^2_{\acute{e}t}(X, \mathbb{Z}/n\mathbb{Z})$, then the rational Hodge structure on $\mathrm{H}^*(\mathrm{SB}(\mathcal{A}), \mathbb{Q})$ is determined from a class $B = b/n \in \mathrm{H}^2(X, \mathbb{Q})$ also,

$$\mathrm{H}^*(\mathrm{SB}(\mathcal{A}), \mathbb{Q}) \xleftarrow{\sim} \bigoplus_{i=0}^{\mathrm{rk}(V)-1} \mathrm{H}^{*-2i}(X, \mathbb{Q})(-i)$$

$$\sum_i \pi^*(x_i) \cup (h + \pi^* B)^i \longleftarrow\!\shortmid \quad \sum_i x_i$$

where $h = c_1(\mathcal{O}_{\mathbb{P}(V)(1)})$.

Background

An invariant of $[\mathcal{A}] \in \mathrm{Br}(X)[n]$ called *the index* is detected by the existence of algebraic cycles on $\mathrm{SB}(\mathcal{A})$ with linear fibers over X.

Background

An invariant of $[\mathcal{A}] \in \mathrm{Br}(X)[n]$ called *the index* is detected by the existence of algebraic cycles on $\mathrm{SB}(\mathcal{A})$ with linear fibers over X.

For an integral Hodge class $\gamma \in H^{2d}(\mathrm{SB}(\mathcal{A}), \mathbb{Q})$ to exist with this fiber degree, there must be rational Hodge classes $c_j \in \mathrm{H}^{2j}(X, \mathbb{Q})$ such that

$$p_i^{B,d}(c_1, ..., c_i) = \binom{d}{i} B^i + \sum_{j=1}^{i} \binom{d-j}{i-j} B^{i-j} c_j \in \mathrm{H}^{2i}(X, \mathbb{Q})$$

are integral for each $i = 1, ..., d$.

Background

An invariant of $[\mathcal{A}] \in \mathrm{Br}(X)[n]$ called *the index* is detected by the existence of algebraic cycles on $\mathrm{SB}(\mathcal{A})$ with linear fibers over X.

For an integral Hodge class $\gamma \in H^{2d}(\mathrm{SB}(\mathcal{A}), \mathbb{Q})$ to exist with this fiber degree, there must be rational Hodge classes $c_j \in \mathrm{H}^{2j}(X, \mathbb{Q})$ such that

$$p_i^{B,d}(c_1, ..., c_i) = \binom{d}{i} B^i + \sum_{j=1}^{i} \binom{d-j}{i-j} B^{i-j} c_j \in \mathrm{H}^{2i}(X, \mathbb{Q})$$

are integral for each $i = 1, ..., d$.

Perry and de Jong find counterexamples to the integral Hodge conjecture by producing such Hodge classes on $\mathrm{SB}(\mathcal{A})$ for $\mathcal{A}$ with a known index.

Background

Theorem

Suppose that there exists an algebraic cycle on $\mathrm{SB}(\mathcal{A})$ *linear of codimension* d *on the fibers.*

Background

Theorem

Suppose that there exists an algebraic cycle on $\mathrm{SB}(\mathcal{A})$ *linear of codimension* d *on the fibers. Then there exist rational Hodge classes* $c_j \in \mathrm{H}^{2j}(X, \mathbb{Q})$ *such that:*

- *the classes* $C_i = p_i^{B,d}(c_1, ..., c_i)$ *are integral,*

Background

Theorem

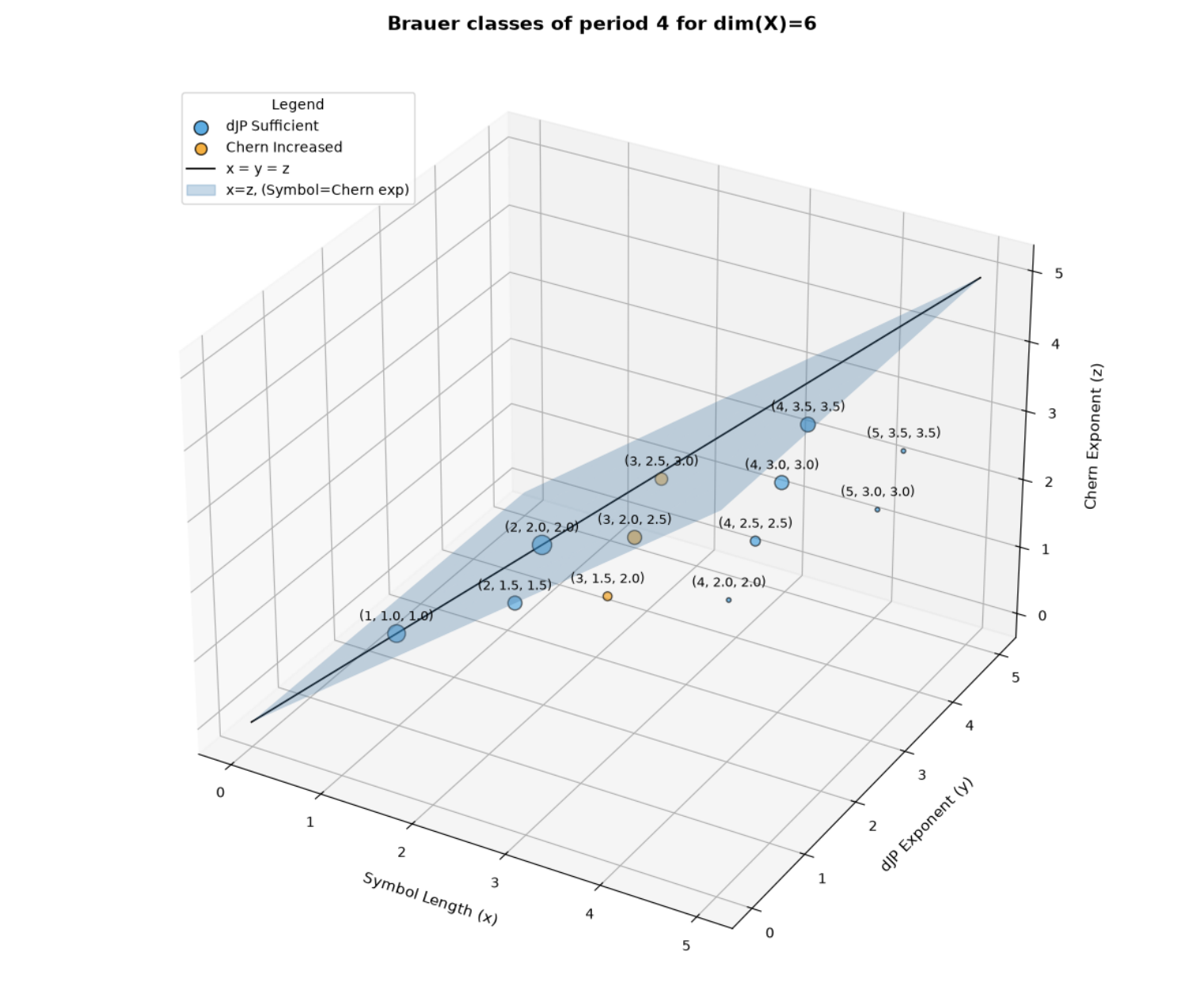
Suppose that there exists an algebraic cycle on $\mathrm{SB}(\mathcal{A})$ *linear of codimension* d *on the fibers. Then there exist rational Hodge classes* $c_j \in \mathrm{H}^{2j}(X, \mathbb{Q})$ *such that:*

- *the classes* $C_i = p_i^{B,d}(c_1, ..., c_i)$ *are integral,*
- *and for all* $l \geq 1$ *we have*

$$\mathrm{Sq}^{2l}(C_i) \equiv \sum_{k=0}^{l} \binom{i-l+k-1}{k} C_{l-k} C_{i+k} \pmod 2$$

for the mod-2 Steenrod operations Sq^{2l}.

Background


Brauer classes of period 4 for dim(X)=6
Legend
dJP Sufficient
Chern Increased
x = y = z
x=z, (Symbol=Chern exp)
(1, 1.0, 1.0)
(2, 1.5, 1.5)
(2, 2.0, 2.0)
(3, 1.5, 2.0)
(3, 2.0, 2.5)
(3, 2.5, 3.0)
(4, 2.0, 2.0)
(4, 2.5, 2.5)
(4, 3.0, 3.0)
(4, 3.5, 3.5)
(5, 3.0, 3.0)
(5, 3.5, 3.5)
Symbol Length (x)
dJP Exponent (y)
Chern Exponent (z)

Background

HODGE THEORETIC LOWER BOUNDS ON INDICES OF ELEMENTS OF WEIL–CHÂTELET GROUPS

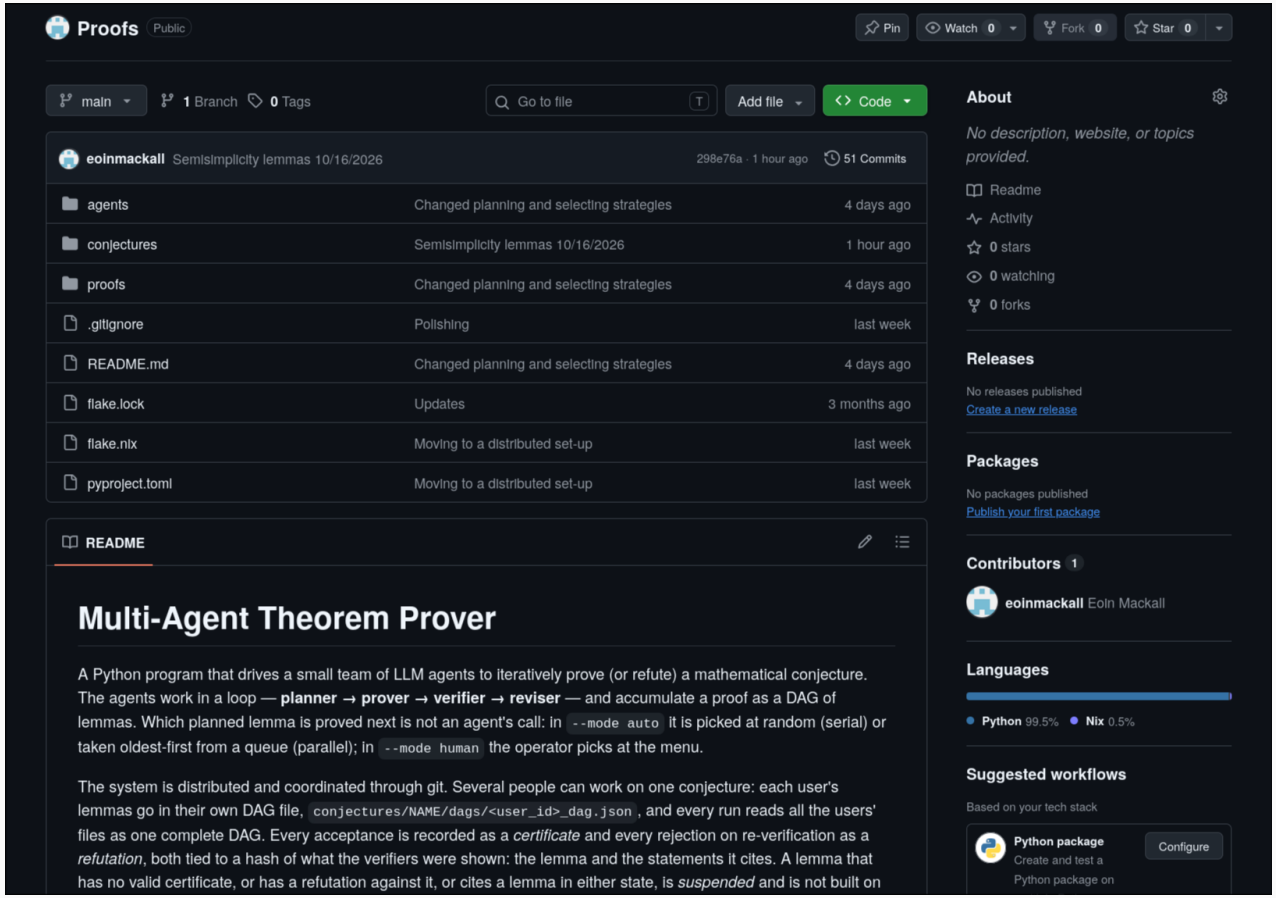
EOIN MACKALL

ABSTRACT. We develop a technique for producing lower bounds on the indices of torsors under an abelian scheme A over a smooth and proper complex scheme X. Our technique is the analog of one due to Hotchkiss, de Jong and Perry that can be used for producing lower bounds on indices of unramified Brauer classes over X. We conclude with some data analysis for specific examples of abelian schemes A isotrivial over a base X.

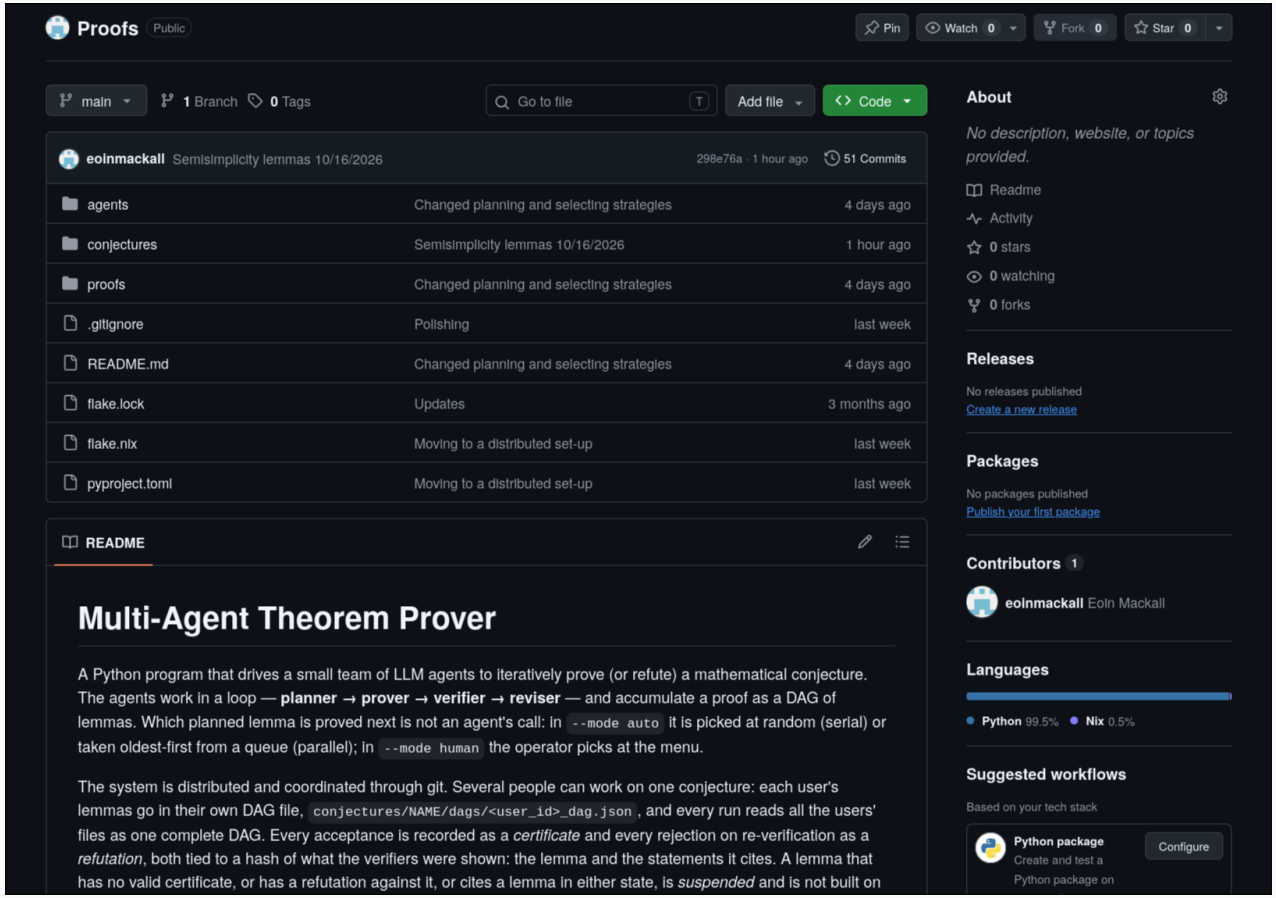
https://claude.ai/share/f8025cdf-df45-47c9-892a-0d54c90d3e38

Proofs

Proofs

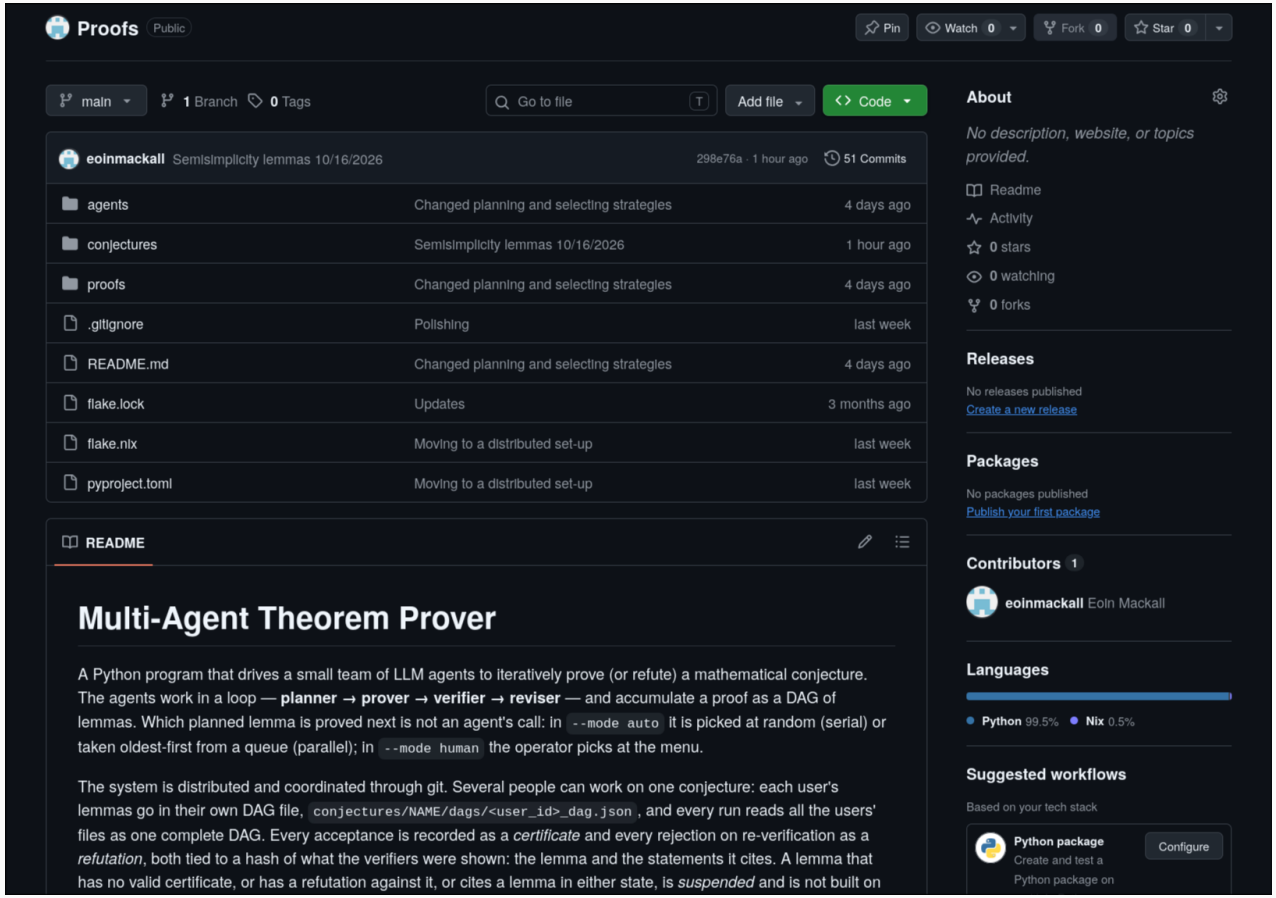

Proofs
Public
Pin
Watch 0
Fork 0
Star 0
main
1 Branch
0 Tags
Go to file
Add file
Code
eoinmackall Semisimplicity lemmas 10/16/2026
298e76a · 1 hour ago
51 Commits
agents
Changed planning and selecting strategies
4 days ago
conjectures
Semisimplicity lemmas 10/16/2026
1 hour ago
proofs
Changed planning and selecting strategies
4 days ago
.gitignore
Polishing
last week
README.md
Changed planning and selecting strategies
4 days ago
flake.lock
Updates
3 months ago
flake.nix
Moving to a distributed set-up
last week
pyproject.toml
Moving to a distributed set-up
last week
README
Multi-Agent Theorem Prover
A Python program that drives a small team of LLM agents to iteratively prove (or refute) a mathematical conjecture. The agents work in a loop — planner → prover → verifier → reviser — and accumulate a proof as a DAG of lemmas. Which planned lemma is proved next is not an agent's call: in --mode auto it is picked at random (serial) or taken oldest-first from a queue (parallel); in --mode human the operator picks at the menu.
The system is distributed and coordinated through git. Several people can work on one conjecture: each user's lemmas go in their own DAG file, conjectures/NAME/dags/<user_id>_dag.json, and every run reads all the users' files as one complete DAG. Every acceptance is recorded as a certificate and every rejection on re-verification as a refutation, both tied to a hash of what the verifiers were shown: the lemma and the statements it cites. A lemma that has no valid certificate, or has a refutation against it, or cites a lemma in either state, is suspended and is not built on
About
No description, website, or topics provided.
Readme
Activity
0 stars
0 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Contributors 1
eoinmackall Eoin Mackall
Languages
Python 99.5%
Nix 0.5%
Suggested workflows
Based on your tech stack
Python package
Create and test a Python package on
Configure

Proofs



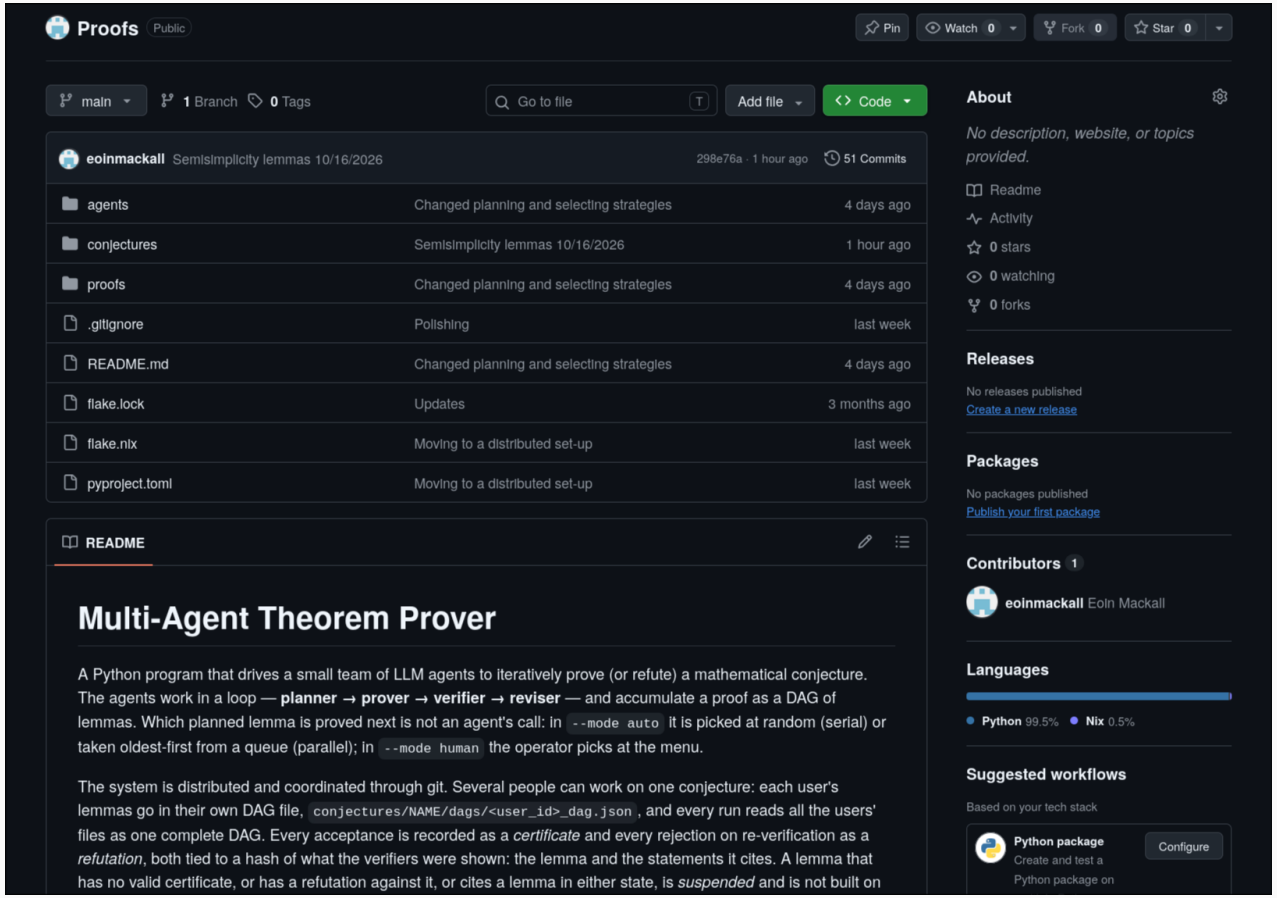
- harness for proving theorems in natural language (English)

Proofs



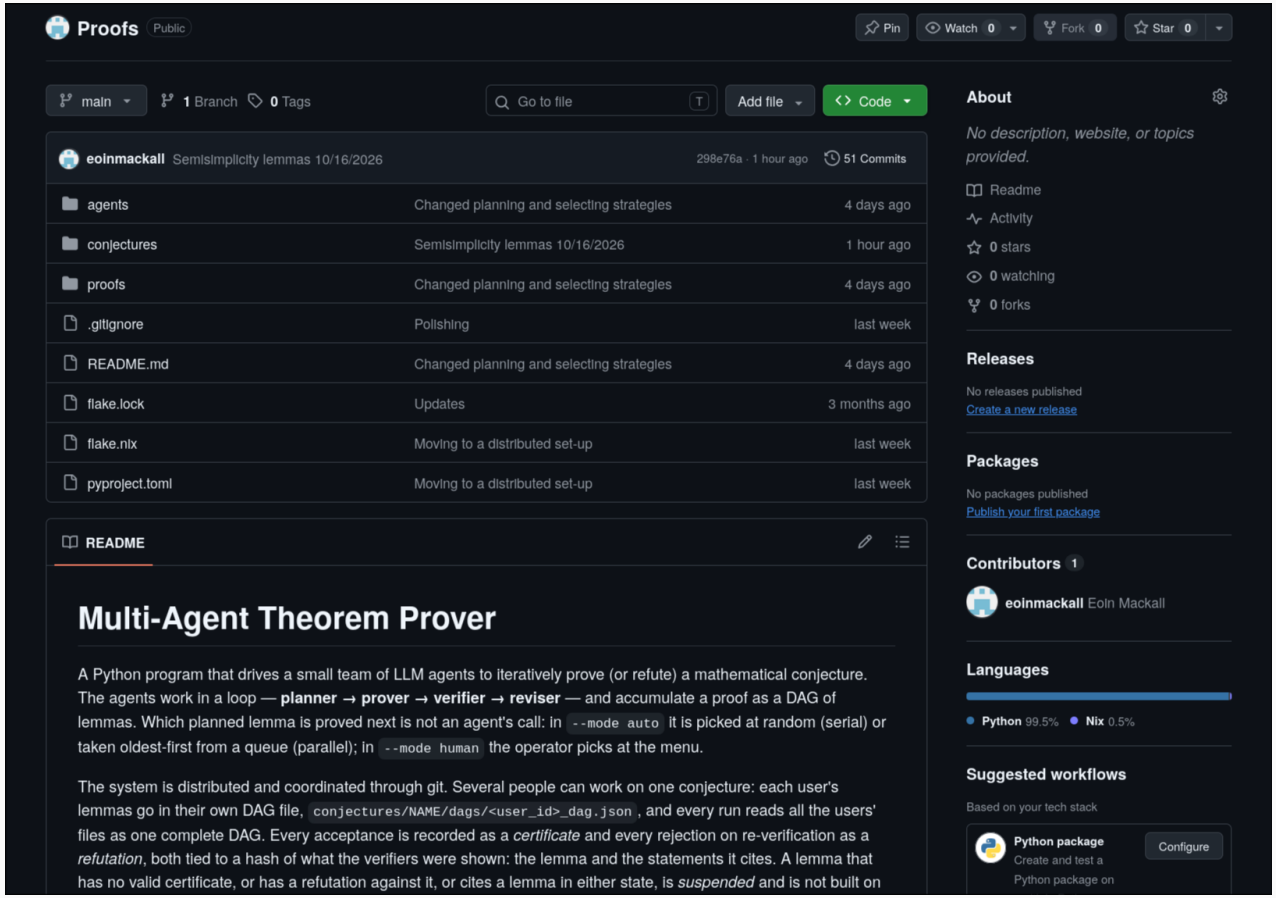
- harness for proving theorems in natural language (English)
- open-source (will probably add GPLv3 license)

Proofs



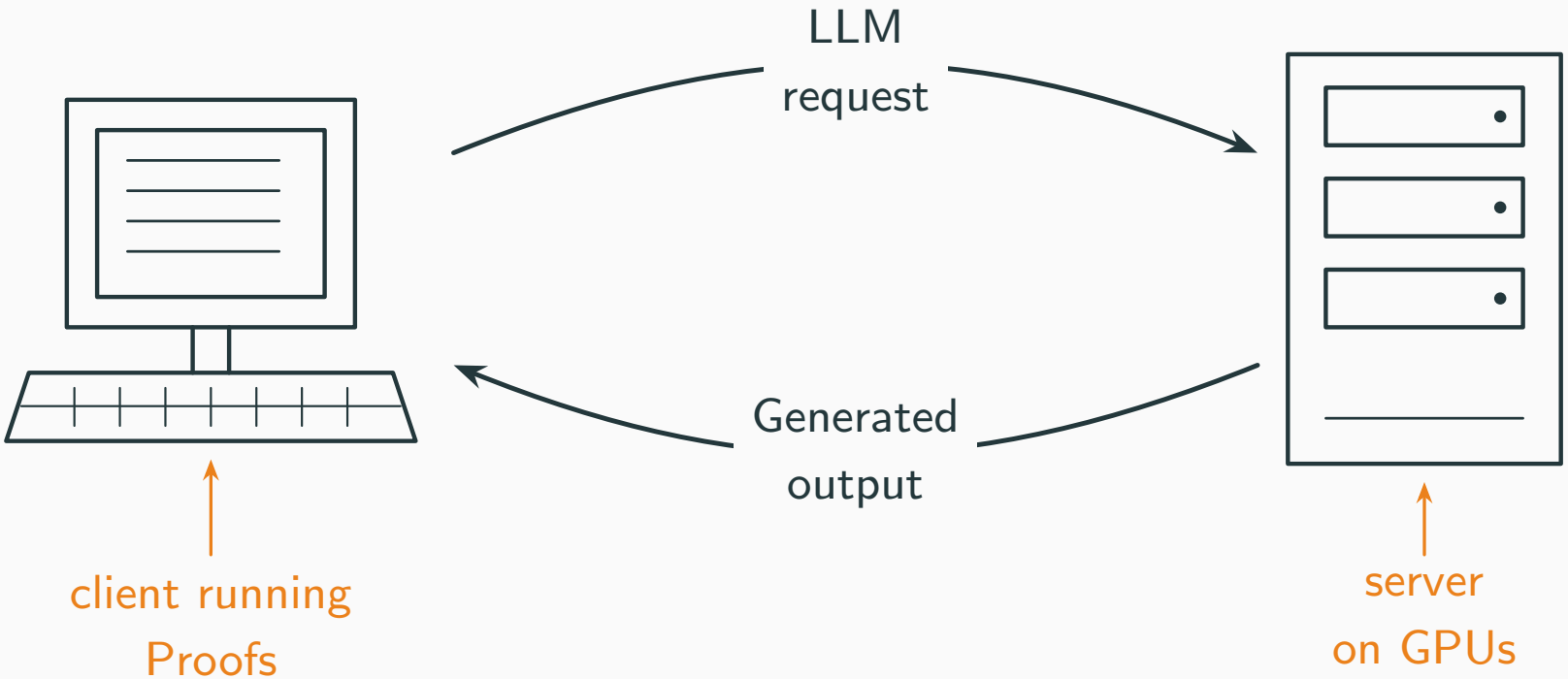
- harness for proving theorems in natural language (English)
- open-source (will probably add GPLv3 license)
- OpenAI API compatible

Proofs

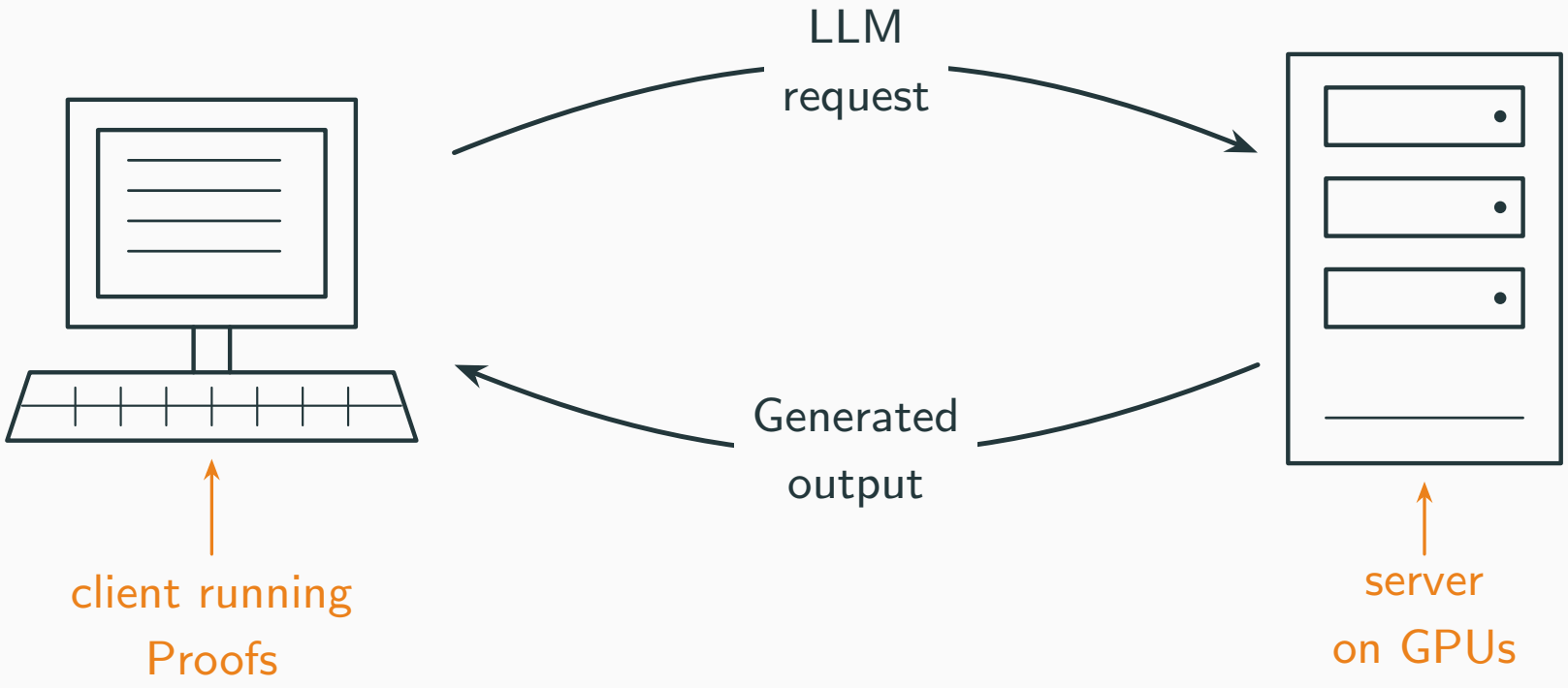


- harness for proving theorems in natural language (English)
- open-source (will probably add GPLv3 license)
- OpenAI API compatible
- distributed design with parallel LLM requests at runtime

Proofs: LLM inference



Proofs: LLM inference



CLI command

```
proofs run conjectures/semisimplicity/
  --model qwen3
  --parallel 3
  --host https://ellm.nrp-nautilus.io
  --api-key sk-Abd3...
```

System prompt vs. user prompt

Request to the server (e.g. vLLM or llama-server)

```
client.chat.completions.create(
 model="Qwen/Qwen3.8-27B",
 messages=[
  {"role": "system", "content": prover_md},
  {"role": "user",   "content": lemma_prompt},
 ])
```

} system prompt
} user prompt

What the model sees (with the chat template)

```
<|im_start|>system
Reasoning effort is set to xhigh. Please think carefully
through the task, validate key assumptions, consider plausible
alternatives, and prioritize correctness, consistency, and
clarity in the final answer.
<prover_md><|im_end|>
<|im_start|>user
<lemma_prompt><|im_end|>
<|im_start|>assistant
<think>
```

} system prompt = template's line + yours
} user prompt

Proofs: LLM inference

Example (A "prover agent" system prompt)

You are a rigorous mathematical prover agent. Given the target lemma, the overall conjecture, the statements of the lemmas already proved, and the parsed reference collection, write a complete, rigorous, step-by-step proof of the target lemma.

You decide which of the proved lemmas your argument needs; nothing has been chosen for you. Read their statements, use the ones that help, and ignore the rest.

The proved lemmas are listed with two identifier fields each: user_id and lemma_id. A lemma_id is unique only within one user's work, so a lemma is identified by the pair {"user_id", "lemma_id"} — always cite a lemma by the pair, copying both fields from its entry verbatim.

If a "Known references" list is provided, its entries are theorem-level results from a parsed reference collection: treat them exactly like proved lemmas. You may cite any of them by id and rely on them without proving them; a proof that re-proves a reference...

Proofs: design

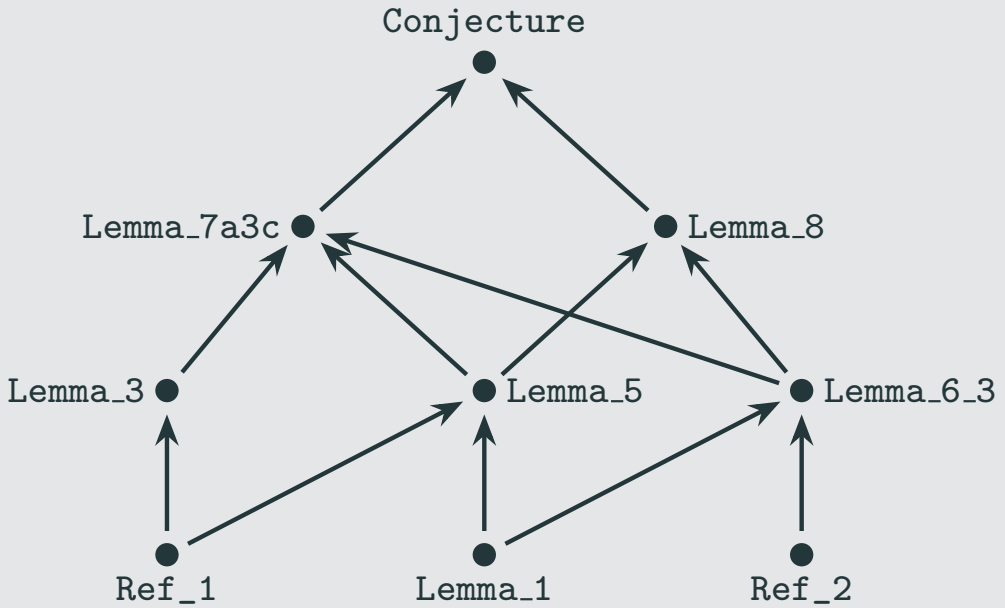
Input

- `conjecture.md`
- `references.md`
- `comments.md`
- + work so far

$\Rightarrow$

Output

A directed acyclic graph of lemmas and their dependencies



Proofs: design

Agent loop

1. A planner reads the conjecture, additional context (references, lemmas from DAG, and comments) and comes up with possible directions to attack the conjecture

Proofs: design

Agent loop

1. A planner reads the conjecture, additional context (references, lemmas from DAG, and comments) and comes up with possible directions to attack the conjecture
2. A generator is given the planner's plan and constructs $N + 5$ possible formal lemma statements in this direction

Proofs: design

Agent loop

1. A planner reads the conjecture, additional context (references, lemmas from DAG, and comments) and comes up with possible directions to attack the conjecture
2. A generator is given the planner's plan and constructs $N + 5$ possible formal lemma statements in this direction
3. N proof loops are spawned

Proofs: design

Agent loop

1. A planner reads the conjecture, additional context (references, lemmas from DAG, and comments) and comes up with possible directions to attack the conjecture
2. A generator is given the planner's plan and constructs $N + 5$ possible formal lemma statements in this direction
3. N proof loops are spawned
   - 3.1 A prover takes one of the $N + 5$ formal lemma statements and attempts to prove it

Proofs: design

Agent loop

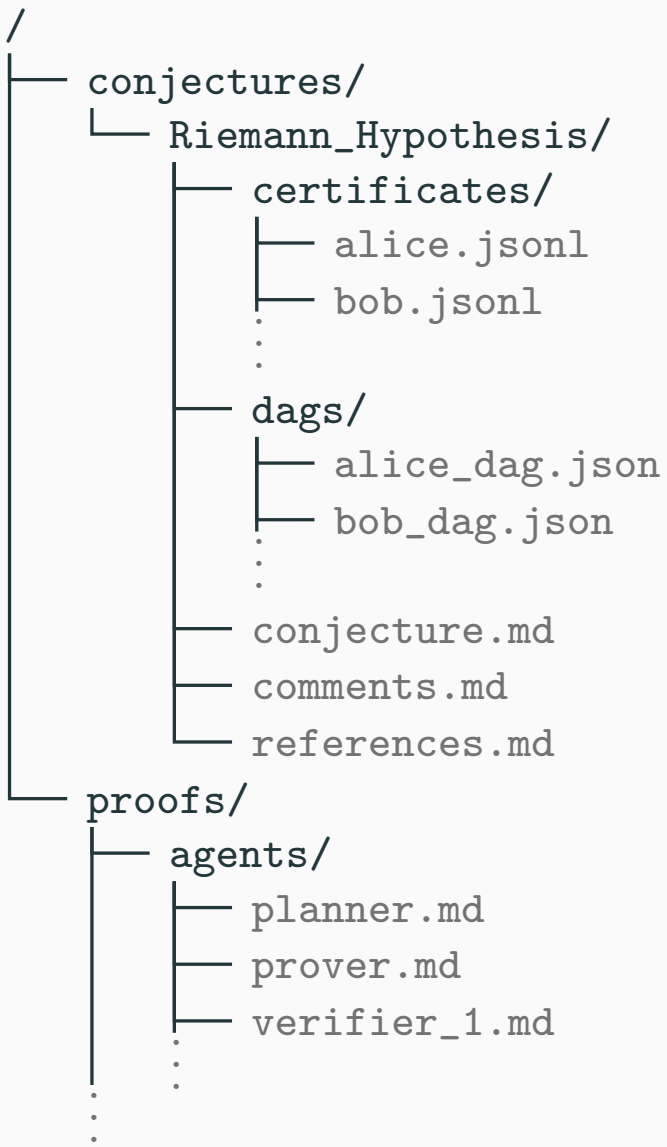
1. A planner reads the conjecture, additional context (references, lemmas from DAG, and comments) and comes up with possible directions to attack the conjecture
2. A generator is given the planner's plan and constructs $N + 5$ possible formal lemma statements in this direction
3. N proof loops are spawned
   - 3.1 A prover takes one of the $N + 5$ formal lemma statements and attempts to prove it
   - 3.2-4 A sequence of three verifiers reviews the lemma; a lemma that successfully passes all verifiers goes to the DAG

Proofs: design

Agent loop

1. A planner reads the conjecture, additional context (references, lemmas from DAG, and comments) and comes up with possible directions to attack the conjecture
2. A generator is given the planner's plan and constructs $N + 5$ possible formal lemma statements in this direction
3. N proof loops are spawned
   - 3.1 A prover takes one of the $N + 5$ formal lemma statements and attempts to prove it
   - 3.2-4 A sequence of three verifiers reviews the lemma; a lemma that successfully passes all verifiers goes to the DAG
   - 3.5 A rejected lemma is passed to a reviser who tries to salvage the attempt by adjusting the statement or suggesting revisions to the proof

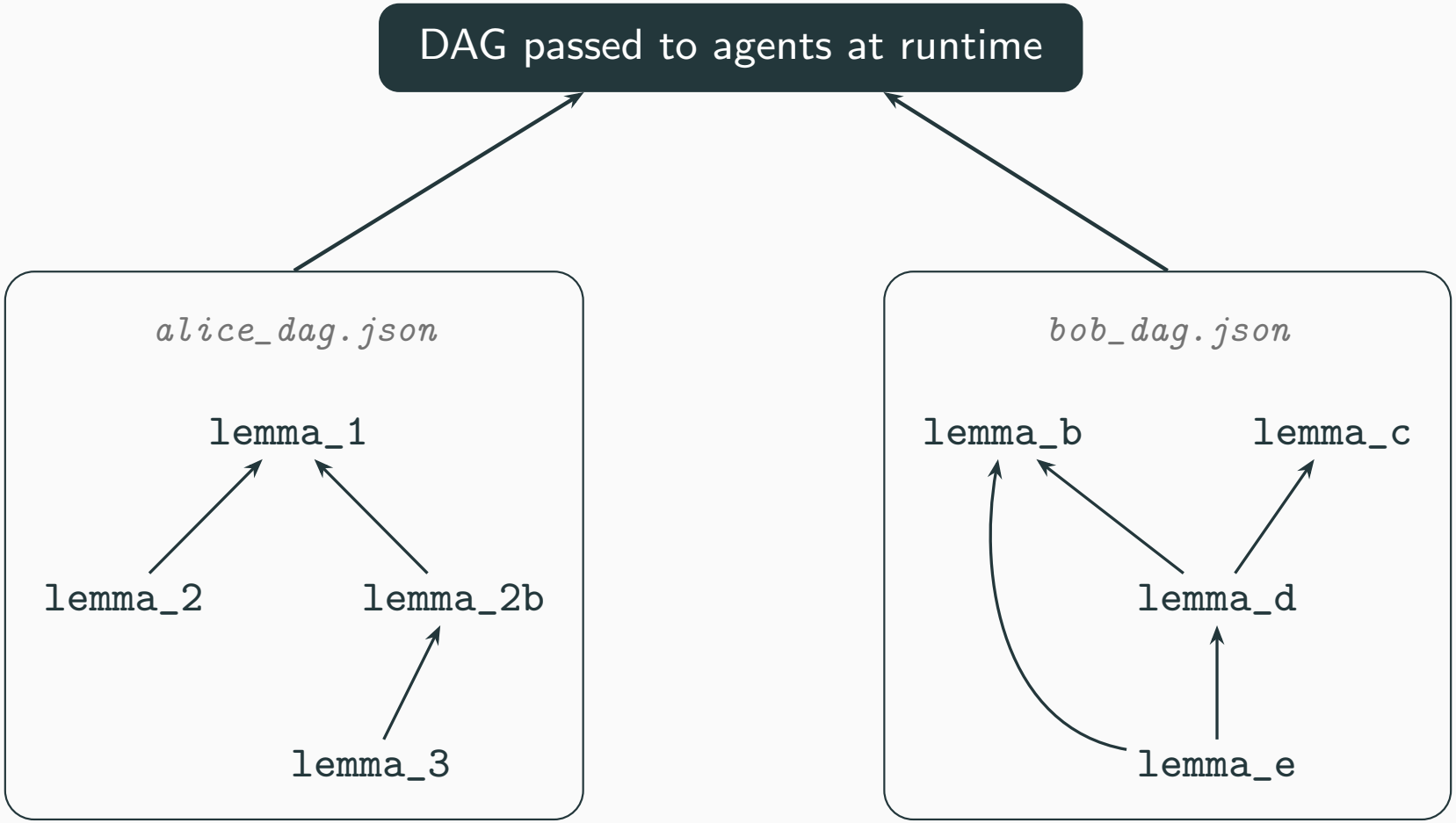
Proofs: design



Proofs: design

Distributed design

- Each user sets global variables `user_id`, `name`, `email`
- DAGs are created under ‘`conjectures/CONJ/dags/user_id.dag`‘



Proofs: design

Proofs are stored with some additional metadata.

Proofs: design

Proofs are stored with some additional metadata.

Example

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/semisimplicity/dags/eoinm_dag.json`

Proofs: design

Proofs are stored with some additional metadata.

Example

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/semisimplicity/dags/eoinm_dag.json`

Problem

1. Can't pass the whole DAG to an LLM at runtime without risking context bloat

Proofs: design

Proofs are stored with some additional metadata.

Example

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/semisimplicity/dags/eoinm_dag.json`

Problem

1. Can't pass the whole DAG to an LLM at runtime without risking context bloat
2. Different algorithms for selecting which lemmas are "important"

Proofs: design

Proofs are stored with some additional metadata.

Example

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/semisimplicity/dags/eoinm_dag.json`

Problem

1. Can't pass the whole DAG to an LLM at runtime without risking context bloat
2. Different algorithms for selecting which lemmas are "important"
   - The n most recently proved lemmas?

Proofs: design

Proofs are stored with some additional metadata.

Example

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/semisimplicity/dags/eoinm_dag.json`

Problem

1. Can't pass the whole DAG to an LLM at runtime without risking context bloat
2. Different algorithms for selecting which lemmas are "important"
   - The n most recently proved lemmas?
   - The n most cited lemmas?

Proofs: design

Proofs are stored with some additional metadata.

Example

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/semisimplicity/dags/eoinm_dag.json`

Problem

1. Can't pass the whole DAG to an LLM at runtime without risking context bloat
2. Different algorithms for selecting which lemmas are "important"
   - The n most recently proved lemmas?
   - The n most cited lemmas?
   - Some n randomly selected lemmas?

Proofs: verification

The trust problem

How do you verify contributions or a specific claim?

Proofs: verification

The trust problem

How do you verify contributions or a specific claim?

A user can point the system at any lemma in any user's DAG to verify that lemma, or that lemma together with everything it cites.

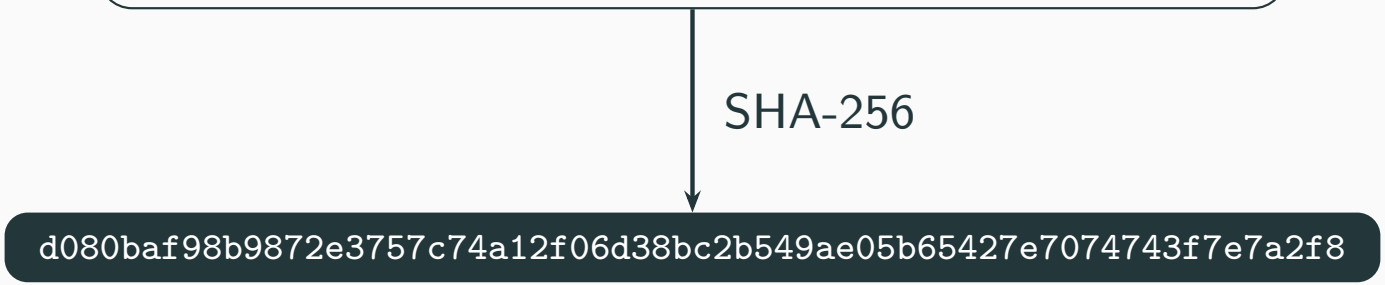
Proofs: verification

The trust problem

How do you verify contributions or a specific claim?

A user can point the system at any lemma in any user's DAG to verify that lemma, or that lemma together with everything it cites.

```
{
  "lemma_id_1": {
    "lemma_statement": "Let X, Y, ... Then ...",
    "proof": "We have ...",
    "citations": [
      {"alice": "lemma_1", "statement": "..."},
      {"bob": "lemma_b", "statement": "..."}
    ]
  }
}
```



Proofs: verification

Certificates for verified lemmas are stored in a `verifiers_user_id.jsonl` file under `certificates/`.

Example

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/semisimplicity/certificates/eoinm.jsonl`

Proofs: verification

Certificates for verified lemmas are stored in a `verifiers_user_id.jsonl` file under `certificates/`.

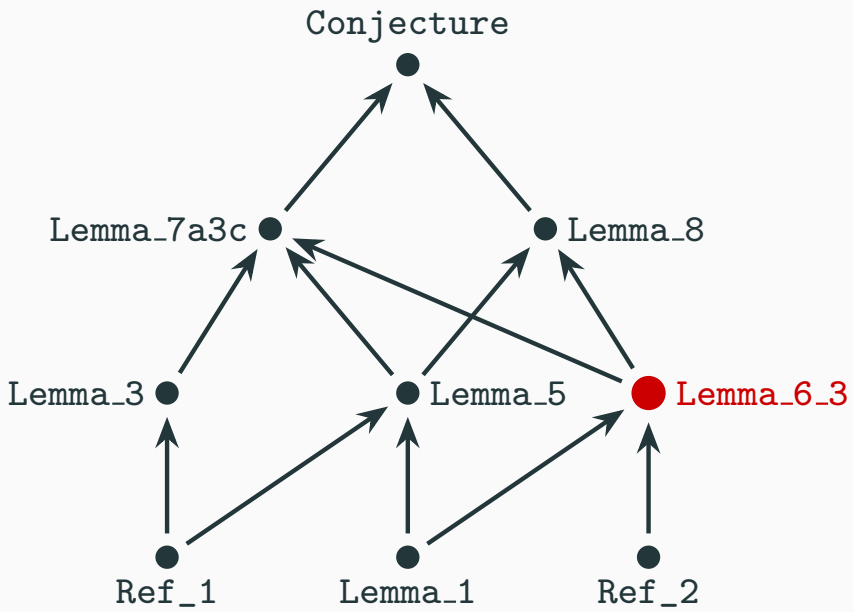
Example

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/semisimplicity/certificates/eoinm.jsonl`

Just like in mathematics today, the more times that a lemma is verified the more likely that it is correct(?).

Proofs: refutations

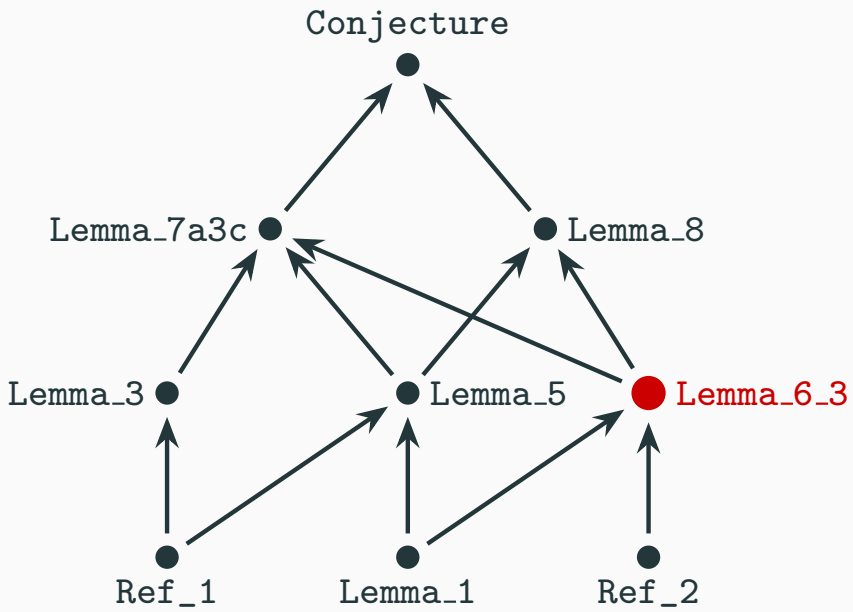
If a lemma in a user's DAG is rejected by the verifiers during a verification, a refutation is added.



Proofs: refutations

If a lemma in a user's DAG is rejected by the verifiers during a verification, a refutation is added.

Refutations are stored in a special `refutations/` directory.

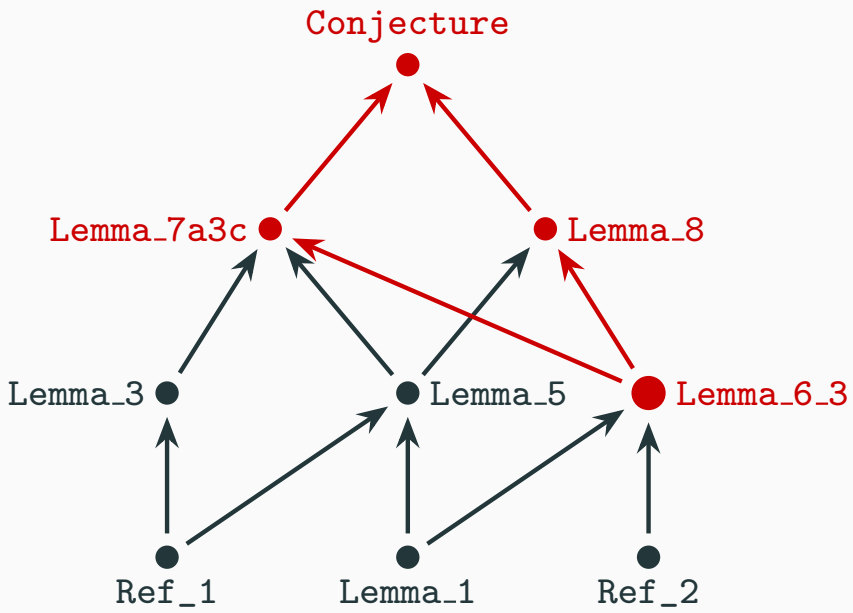


Proofs: refutations

If a lemma in a user's DAG is rejected by the verifiers during a verification, a refutation is added.

Refutations are stored in a special `refutations/` directory.

At runtime, any lemma whose hash matches one listed in refutations is "suspended". All dependents are suspended too.



Proofs: refutations

A lemma with a refutation can be repaired.

Proofs: refutations

A lemma with a refutation can be repaired. Then:

1. If the verifiers accept the proof, the refutation is removed (false positive, maybe?).

Proofs: refutations

A lemma with a refutation can be repaired. Then:

1. If the verifiers accept the proof, the refutation is removed (false positive, maybe?).
2. If the proof is fixed, the refutation is removed.

Proofs: refutations

A lemma with a refutation can be repaired. Then:

1. If the verifiers accept the proof, the refutation is removed (false positive, maybe?).
2. If the proof is fixed, the refutation is removed.
3. If the statement is amended, then the refutation is removed and all certificates for every dependent become invalid. The program walks up the chain re-verifying dependents.

Proofs: refutations

A lemma with a refutation can be repaired. Then:

1. If the verifiers accept the proof, the refutation is removed (false positive, maybe?).
2. If the proof is fixed, the refutation is removed.
3. If the statement is amended, then the refutation is removed and all certificates for every dependent become invalid. The program walks up the chain re-verifying dependents.
4. If none of the above happen, then there may be an error and the refutation stands.

Proofs: semisimplicity conjecture

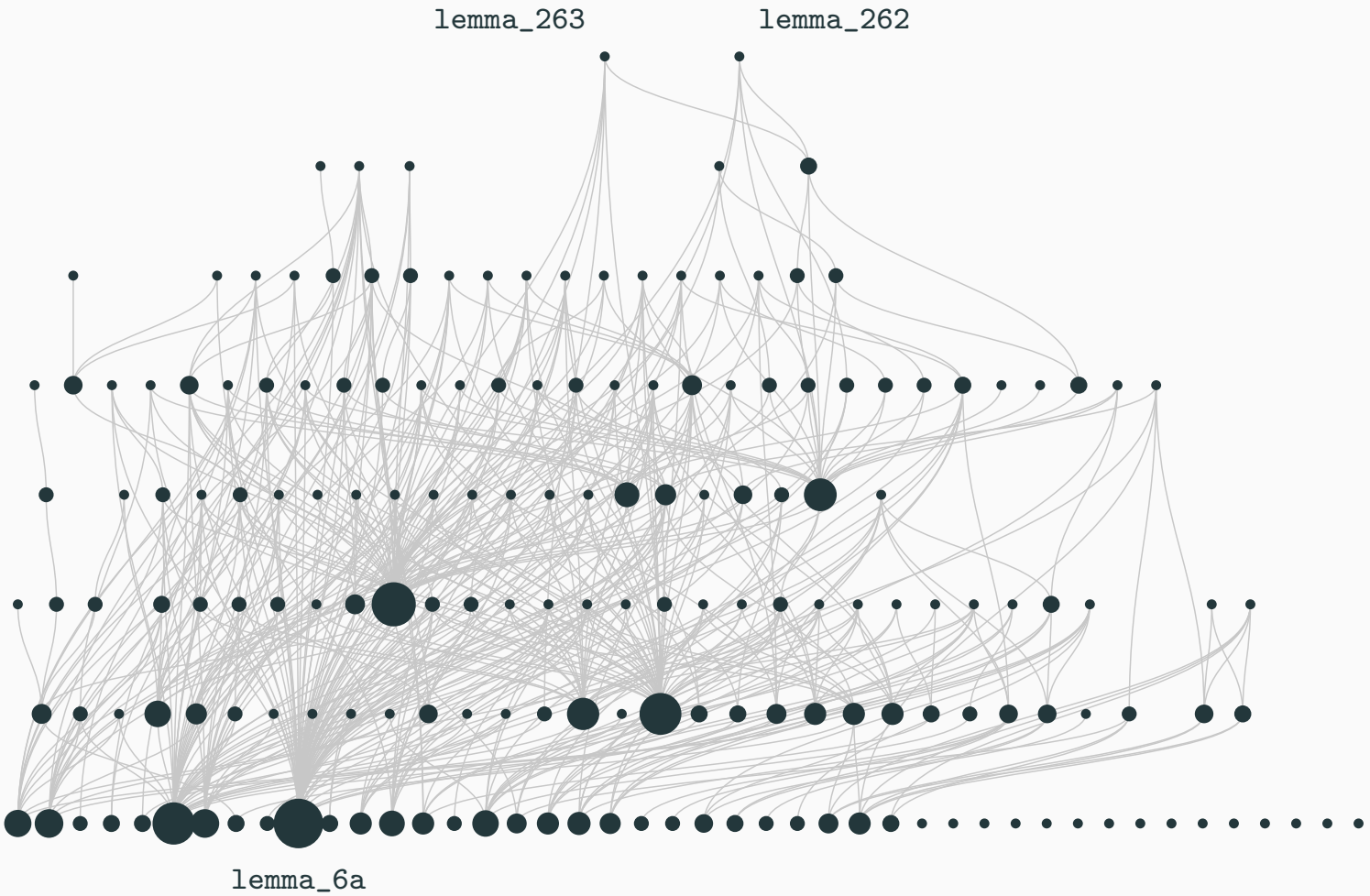
Conjecture

Let $q = p^a$ be a prime power, let X be a smooth projective variety over $\mathbb{F}_q$, and let $\ell \neq p$ be a prime. For every integer $i \geq 0$, the geometric Frobenius acts semisimply on $H^i_{\text{ét}}(X_{\overline{\mathbb{F}}_q}, \mathbb{Q}_\ell)$.

Proofs: semisimplicity conjecture

Conjecture

Let $q = p^a$ be a prime power, let X be a smooth projective variety over $\mathbb{F}_q$, and let $\ell \neq p$ be a prime. For every integer $i \geq 0$, the geometric Frobenius acts semisimply on $H^i_{\text{ét}}(X_{\overline{\mathbb{F}}_q}, \mathbb{Q}_\ell)$.



Proofs: semisimplicity conjecture

- 181 lemmas in the DAG

Proofs: semisimplicity conjecture

- 181 lemmas in the DAG
- Mostly has reproved known results or special cases (e.g. curves, H^2 and H^{d-2}, abelian varieties)

Proofs: semisimplicity conjecture

- 181 lemmas in the DAG
- Mostly has reproved known results or special cases (e.g. curves, H^2 and H^{d-2}, abelian varieties)
- Mostly used Qwen3.8-flash-next on the NRP LLM server with 3 proof loops ($\sim$194 million tokens; 108 million input, 85.6 million output)

Proofs: semisimplicity conjecture

- 181 lemmas in the DAG
- Mostly has reproved known results or special cases (e.g. curves, H^2 and H^{d-2}, abelian varieties)
- Mostly used Qwen3.8-flash-next on the NRP LLM server with 3 proof loops ($\sim$194 million tokens; 108 million input, 85.6 million output)
- Through an inference provider this would have cost \$58.22.

Proofs: p-crossed products of degree p^2

Conjecture

Let k be a field of characteristic $p > 2$. Let A be a central simple k-algebra of degree p^2 (i.e. a p-algebra of degree p^2). Then A is a crossed product algebra for some group G of order p^2.

Proofs: p-crossed products of degree p^2

Conjecture

Let k be a field of characteristic $p > 2$. Let A be a central simple k-algebra of degree p^2 (i.e. a p-algebra of degree p^2). Then A is a crossed product algebra for some group G of order p^2.

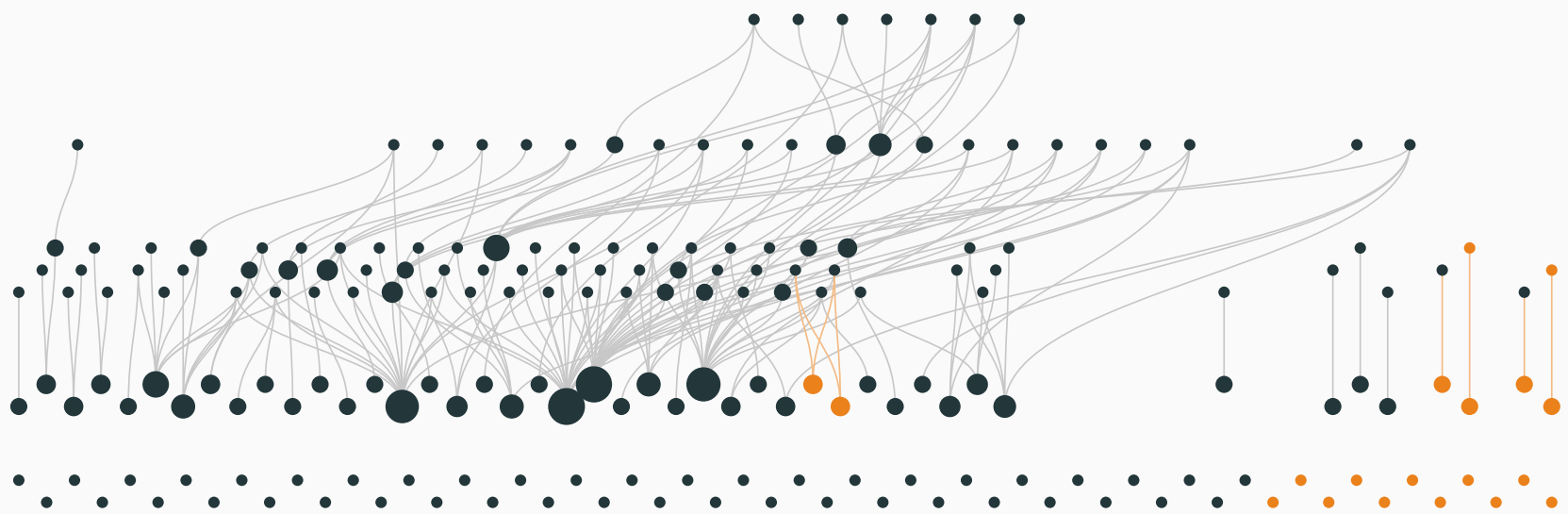
`https://github.com/eoinmackall/Proofs/blob/main/conjectures/algebra_example/conjecture.md`

Proofs: p-crossed products of degree p^2

Conjecture

Let k be a field of characteristic $p > 2$. Let A be a central simple k-algebra of degree p^2 (i.e. a p-algebra of degree p^2). Then A is a crossed product algebra for some group G of order p^2.

`https://github.com/eoinmackall/Proofs/blob/main/conjectures/algebra_example/conjecture.md`



Proofs: p-crossed products of degree p^2

- 204 lemmas in total

Proofs: p-crossed products of degree p^2

- 204 lemmas in total
- Mostly run on the Nvidia DGX Spark for the week leading up to the workshop

Proofs: p-crossed products of degree p^2

- 204 lemmas in total
- Mostly run on the Nvidia DGX Spark for the week leading up to the workshop
- At 240W max power draw, this translates to between \$13.31 and \$16.53 for one week (at PG&E rates)

Proofs: future directions

1. Add additional tooling (e.g. access to SageMath, OSCAR, other CAS's)

Proofs: future directions

1. Add additional tooling (e.g. access to SageMath, OSCAR, other CAS's)

Some problems

- Given access to run code, need to sandbox

Proofs: future directions

1. Add additional tooling (e.g. access to SageMath, OSCAR, other CAS's)

Some problems

- Given access to run code, need to sandbox
- Add code-runs to certification (probably shouldn't execute code submitted by an anonymous contributor)

Proofs: future directions

1. Add additional tooling (e.g. access to SageMath, OSCAR, other CAS's)

Some problems

- Given access to run code, need to sandbox
- Add code-runs to certification (probably shouldn't execute code submitted by an anonymous contributor)
- Maintaining reproducibility and ease of use (some solutions, e.g. Nix?)

Proofs: future directions

1. Add additional tooling (e.g. access to SageMath, OSCAR, other CAS's)

Some problems

- Given access to run code, need to sandbox
- Add code-runs to certification (probably shouldn't execute code submitted by an anonymous contributor)
- Maintaining reproducibility and ease of use (some solutions, e.g. Nix?)

2. Add the ability to translate a lemma into Lean (and a "permanent" certification)

Proofs: future directions

1. Add additional tooling (e.g. access to SageMath, OSCAR, other CAS's)

Some problems

- Given access to run code, need to sandbox
- Add code-runs to certification (probably shouldn't execute code submitted by an anonymous contributor)
- Maintaining reproducibility and ease of use (some solutions, e.g. Nix?)

2. Add the ability to translate a lemma into Lean (and a "permanent" certification)
3. Obtain concrete (new) results, tests, and benchmarks for variations

On Proofs and Progress

Thanks for your time