

# A Sledgehammer for Dependent Type Theory

Jeremy Avigad

Department of Philosophy

Department of Mathematical Sciences

Carnegie Mellon University

Joint work with Thomas (Hanwen) Zhu, Joshua Clune,  
Albert (Qiaochu) Jiang, Sean Welleck, and others

October 31, 2025

# Outline

- Automated vs. Interactive Theorem Proving
- The Sledgehammer Ethos
- The Challenges of Dependent Type Theory
- Our Sledgehammer
- Premise Selection
- Other Automation
- Institute for Computer-Aided Reasoning in Mathematics
- Neurosymbolic AI

A vertical blue textured bar on the left side of the slide, with a fine, woven fabric-like pattern.

# **Automated vs. Interactive Theorem Proving**



# History

Automated reasoning has been around since the 1950s:

- Martin Davis implemented Presburger's decision procedure at the IAS in 1954.
- Allen Newell, Herbert Simon, and Cliff Shaw introduced the Logic Theorist in 1956.
- Hao Wang implemented good provers for propositional and predicate logic in 1958.
- Henry Gelernter, J. R. Hansen, and Donald Loveland published an article on the Geometry Machine in 1960.
- Davis and Hilary Putnam introduced the propositional resolution rule in 1960.
- John Alan Robinson introduced a unification algorithm in 1965.

# History

The first proof checkers appeared later:

- De Bruijn's Automath appeared in the late 1960s.
- Mizar appeared in 1973.

It's not surprising, considering that interactive theorem proving requires interaction.

See Harrison, Urban, and Wiedijk, “History of Interactive Theorem Proving.”

# Verification

## Using automation:

- Write formal specifications and theorems.
- Call automation to verify inferences.
- Add intermediate assertions if necessary.

## Pros:

- Users only need to learn the assertion language.
- Faster, friendlier development.

## Cons:

- It doesn't scale.
- It's not robust.

# Verification

## Using interaction:

- Write formal specifications and theorems.
- Write detailed formal proofs interactively.

## Pros:

- Anything is provable in principle.
- Have fine control over proofs.

## Cons:

- It requires a lot of expertise.
- It requires a lot of work.

# Synthesis

View automation / interaction as a continuum:

- Any automatic proof system can benefit from user interaction.
- Any interactive proof system can benefit from automation.

A sledgehammer starts on the interactive side.



A vertical blue textured bar on the left side of the slide, resembling a fine-grained fabric or paper texture.

# **The Sledgehammer Ethos**

# Sledgehammers

**The context:** given some hypotheses, a conclusion, and a library with tens of thousands of theorems.

**The task:** formally verify the inference.

**The approach:**

- Use heuristics to extract a modest set of promising premises from the library.
- Translate the problem to the language of one or more ATPs.
- Call external provers to try to prove the goal.
- If any succeeds, harvest information about the proof (possibly only the premises used).
- Use the information to reconstruct a formal proof internally.

# Sledgehammers

There are variations at every step:

- One can use symbolic heuristics, lightweight ML, or neural methods for premise selection.
- One can use first-order provers, higher-order provers, or SMT solvers as external provers.
- One can use various translations.
- One can harvest various types of information.
- One can use different methods of proof reconstruction.



# Sledgehammers

The most successful one to date is Isabelle's Sledgehammer, originally developed by Larry Paulson and Jia Meng (c. 2006), and later by Jasmin Blanchette and many others.

The next slide shows the sledgehammer web page for Isabelle 2009.





# Sledgehammer

[Home](#)

[Overview](#)

[Installation](#)

[Download](#)

[Documentation](#)

[Community](#)

## Site Mirrors:

[Cambridge \(.uk\)](#)  
[Munich \(.de\)](#)  
[Sydney \(.au\)](#)

## The Sledgehammer: Let Automatic Theorem Provers write your Isabelle scripts!



The sledgehammer can be used, at any point in a backward proof, with one mouse click. Your first subgoal will be converted into clause form and given to automatic theorem provers (ATPs), together with perhaps hundreds of other clauses representing currently known facts. Because jobs are run in the background, you can continue to work on your proof by other means. Provers can be run in parallel, the first successful result is displayed, and the other provers are terminated. Any reply (which may arrive minutes later) will appear in the Proof General response buffer. If a subgoal is proved, the response consists of a list of Isabelle commands to insert into the proof script. These will invoke the [Metis](#) prover.

## Installation

Supported provers include [E](#), [SPASS](#), [Vampire](#). Additionally, provers from [System on TPTP](#) can be queried over the internet. The standard [Isabelle installation](#) already includes bundled versions of E and SPASS. Remote Vampire is also preconfigured. Note that remote provers require Perl with the World Wide Web Library *libwww-perl* installed.

## How to Invoke It

The sledgehammer is part of Isabelle/HOL. To call it, merely invoke the menu item *Isabelle > Commands > sledgehammer* (see [screenshot](#)). Alternatively, issue the `sledgehammer` Isar command.

For best results, first simplify your problem by calling *auto* or at least *safe* followed by *simp\_all*. None of the ATPs contain arithmetic decision procedures. They are not especially good at heavy rewriting, but because they regard equations as undirected, they often prove theorems that require the reverse orientation of a rewrite rule. Higher-order problems can be tackled, but the success rate is better for first-order problems. You may get better results if you first simplify the problem to remove higher-order features.

Note that problems can be easy for *auto* and difficult for ATPs, but the reverse is also true, so don't be discouraged if your first attempts fail. Because the system refers to all theorems known to Isabelle, it is particularly suitable when your goal has a short proof from lemmas that you don't know about.

## Additional Commands

Several Isar commands are available to control the sledgehammer.

- `sledgehammer prover1 ... proverN` invokes the specified automated theorem provers in parallel on the first subgoal. The first successful prover will terminate the others.
- The `print_atps` command tells about admissible prover names. Provers with the prefix `remote` query SystemOnTPTP, so an active internet connection is needed.
- If no provers are given as arguments to sledgehammer, the system refers to the default which is set to `"e spass remote_vampire"`.
- `atp_info` prints information about presently running provers.
- `atp_kill` terminates all running provers.

# The Challenges of Dependent Type Theory

# Axiomatic foundations

Set theory is an excellent foundation for mathematics:

- The rules of first-order logic are easy to explain.
- There are nine axioms (with choice).
- It is expressive and strong enough to represent any standard mathematical argument.

For implementation in a proof assistant, a type discipline is essential:

- for disambiguating statements.
- for inferring information.
- for error messages.



# Structural reasoning

Since the early twentieth century, axiomatically characterized structures have been central to mathematics.

Mathematicians now

- take products and powers of structures,
- take limits of structures,
- build quotients of structures,
- and much more.

In short, they calculate with structures as easily as they calculate with numbers.

Structures have to be first-class objects in a proof assistant.



```

def quadraticChar ( $\alpha$  : Type) [MonoidWithZero  $\alpha$ ] (a :  $\alpha$ ) :  $\mathbb{Z}$  :=
  if a = 0 then 0 else if IsSquare a then 1 else -1

def legendreSym (p :  $\mathbb{N}$ ) (a :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  := quadraticChar (ZMod p) a

variable {p q :  $\mathbb{N}$ } [Fact p.Prime] [Fact q.Prime]

theorem quadratic_reciprocity (hp : p  $\neq$  2) (hq : q  $\neq$  2) (hpq : p  $\neq$  q) :
  legendreSym q p * legendreSym p q = (-1) ^ (p / 2 * (q / 2))

```

```
def Padic (p : ℕ) [Fact p.Prime] :=  
  CauSeq.Completion.Cauchy (padicNorm p)
```

```
def PadicInt (p : ℕ) [Fact p.Prime] :=  
  {x : ℚ_[p] // ||x|| ≤ 1}
```

```
variable {p : ℕ} [Fact p.Prime] {F : Polynomial ℤ_[p]} {a : ℤ_[p]}
```

```
theorem hensels_lemma :
```

```
  (hnorm : ||Polynomial.eval a F|| <  
    ||Polynomial.eval a (Polynomial.derivative F)|| ^ 2)
```

```
  ∃ z : ℤ_[p],
```

```
  F.eval z = 0 ∧
```

```
  ||z - a|| < ||F.derivative.eval a|| ∧
```

```
  ||F.derivative.eval z|| = ||F.derivative.eval a|| ∧
```

```
  ∀ z', F.eval z' = 0 → ||z' - a|| < ||F.derivative.eval a|| → z' = z
```

```

variable {R S : Type} (K L : Type) [EuclideanDomain R]
variable [CommRing S] [IsDomain S]
variable [Field K] [Field L]
variable [Algebra R K] [IsFractionRing R K]
variable [Algebra K L] [FiniteDimensional K L] [IsSeparable K L]
variable [algRL : Algebra R L] [IsScalarTower R K L]
variable [Algebra R S] [Algebra S L]
variable [ist : IsScalarTower R S L]
variable [iic : IsIntegralClosure S R L]
variable (abv : AbsoluteValue R  $\mathbb{Z}$ )

/-- The main theorem: the class group of an integral closure `S` of `R`
in a finite extension `L` of `K = Frac(R)` is finite if there is an
admissible absolute value. -/
noncomputable def fintypeOfAdmissibleOfFinite : Fintype (ClassGroup S) :=
  ...

```

```

variable { $\alpha$   $\beta$   $\iota$  : Type} {m : MeasurableSpace  $\alpha$ }
variable [MetricSpace  $\beta$ ] { $\mu$  : Measure  $\alpha$ }
variable [SemilatticeSup  $\iota$ ] [Nonempty  $\iota$ ] [Countable  $\iota$ ]
variable {f :  $\iota \rightarrow \alpha \rightarrow \beta$ } {g :  $\alpha \rightarrow \beta$ } {s : Set  $\alpha$ }

```

```

/-- Egorov's theorem: A sequence of almost everywhere convergent functions
    converges uniformly except on an arbitrarily small set. -/

```

```

theorem tendstoUniformlyOn_of_ae_tendsto
  (hf :  $\forall$  n, StronglyMeasurable (f n))
  (hg : StronglyMeasurable g)
  (hsm : MeasurableSet s) (hs :  $\mu$  s  $\neq \infty$ )
  (hfg :  $\forall^m$  x  $\partial\mu$ , x  $\in$  s  $\rightarrow$  Tendsto (fun n => f n x) atTop ( $\mathcal{N}$  (g x)))
  { $\varepsilon$  :  $\mathbb{R}$ } (h $\varepsilon$  : 0 <  $\varepsilon$ ) :
   $\exists$  t  $\subseteq$  s, MeasurableSet t  $\wedge$   $\mu$  t  $\leq$  ENNReal.ofReal  $\varepsilon$   $\wedge$ 
    TendstoUniformlyOn f g atTop (s  $\setminus$  t) :=
  ...

```



# Dependent type theory

Dependent type theory is great for ITP, but challenging for ATP:

- Types can depend on parameters.
- Types can be empty.
- Parameters are instantiated in theorems and function application.
- Type class inference can depend on parameters and hypotheses.
- An expression as simple as  $x < 2$  can be written in many different forms, depending on where the  $<$  comes from.
- Applying a theorem about  $\mathbb{Z} \bmod p$  may require knowing that  $p$  is prime just to make sense of notation.
- Unification and automation have to recognize definitional equality.

# Our Sledgehammer

# Our sledgehammer

Thomas Zhu, Josh Clune, Albert Jiang, Sean Welleck, and I have implemented a prototype sledgehammer that uses:

- **premise selection:** a neural method we implemented, and an implementation of the Meng-Paulson heuristic by Kim Morrison
- **translation:** Lean-auto (Qian, Clune, Barrett, A) to translate to TPTP / SMT
- **external prover:** Zipperposition
- **proof reconstruction:** Aesop (Limperg, From) and Duper (Clune, Qian, Bentkamp, A)

We are working with the cvc5 team to add cvc5 and LeanSMT.



# LeanHammer

LeanHammer is an automated reasoning tool for Lean that brings together multiple proof search and reconstruction techniques and combines them into one tool. The `hammer` tactic provided by LeanHammer uses a variety of techniques to search for a proof of the current goal, then constructs a suggestion for a tactic script which can replace the `hammer` invocation.

LeanHammer is in an early stage of its development and is therefore subject to breaking changes. There are currently versions of the hammer that are compatible with the stable versions of Lean from `v4.20.0` through `v4.23.0` (and the corresponding versions of Mathlib).

Pull requests and issues are welcome.

## Adding LeanHammer to Your Project

To add LeanHammer for v4.23.0 to an existing project with a `lakefile.toml` file, replace the Mathlib dependency in `lakefile.toml` with the following:

```
[[require]]
name = "Hammer"
git = "https://github.com/JOSHCLUNE/LeanHammer"
rev = "v4.23.0"

[[require]]
name = "mathlib"
scope = "leanprover-community"
rev = "v4.23.0"
```



The file `lean-toolchain` should contain the following:

```
leanprover/lean4:v4.23.0
```





# Lean-auto





Lean-auto was developed principally by Yicheng Qian.

We use Lean-auto to export problems to Zipperposition.

- It instantiates parameters and type classes (monomorphization).
- It chooses canonical representatives of definitionally equivalent terms (canonicalization).
- It collects instances of axioms for inductive types.
- It lifts everything to a common universe.
- It expresses problems as instances of inferences in simple type theory.



# Lean-Auto: An Interface Between Lean 4 and Automated Theorem Provers

Yicheng Qian<sup>1(✉)</sup> , Joshua Clune<sup>2</sup> , Clark Barrett<sup>1</sup> ,  
and Jeremy Avigad<sup>2</sup> 



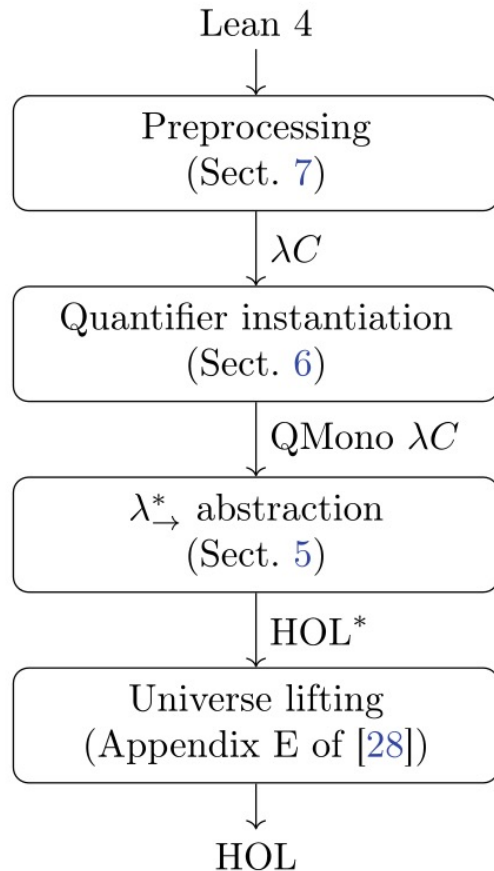
<sup>1</sup> Stanford University, Stanford, USA  
`pratherc@stanford.edu`

<sup>2</sup> Carnegie Mellon University, Pittsburgh, USA



**Abstract.** Proof automation is crucial to large-scale formal mathematics and software/hardware verification projects in ITPs. Sophisticated tools called hammers have been developed to provide general-purpose proof automation in ITPs such as Coq and Isabelle, leveraging the power of ATPs. An important component of a hammer is the translation algorithm from the ITP’s logical system to the ATP’s logical system. In this paper, we propose a novel translation algorithm for ITPs based on dependent type theory. The algorithm is implemented in Lean 4 and the Lean 4 ecosystem. We combined it with ATPs like

cussed in Sect. 4.3.



**Fig. 1.** Translation workflow of Lean-auto.

In our paper, we work backwards in Lean-auto’s translation workflow. We start from  $\lambda^*_{\rightarrow}$  abstraction (Sect. 5), then quantifier instantiation (Sect. 6), and end with preprocessing (Sect. 7). This is because it is easier to begin with the simpler logical system and progressively take into account more features of the highly expressive Lean 4 language. We leave universe lifting to Appendix E of [28] since it is relatively straightforward compared to the other steps.

## 1.1 Related Work

Hammers are not restricted to ITPs with expressive logical systems. Several ITPs based on FOL or HOL also have their hammers, for example, the hammer of Mizar [19], the hammer of MetaMath [9], and HOL(y)Hammer [18]. Apart from hammers, there are various

# Duper

Duper (developed by Clune, Qian, and Bentkamp) is our version of Metis.

- It is a superposition prover, modeled after Zipperposition.
- It includes higher-order inferences.
- It produces formal proofs.
- It has inhabitation reasoning for soundness in the presence of empty types.
- It extends Zipperposition's rules and unification to dependent types.
- It deals with universe levels.
- It currently uses Lean-auto for monomorphization (but see below).

# Duper: A Proof-Producing Superposition Theorem Prover for Dependent Type Theory

Joshua Clune   

Carnegie Mellon University, Pittsburgh, PA, USA

Yicheng Qian 

Peking University, Beijing, China

Alexander Bentkamp   

Heinrich-Heine-Universität Düsseldorf, Germany

Jeremy Avigad   

Carnegie Mellon University, Pittsburgh, PA, USA

---

## Abstract

We present Duper, a proof-producing theorem prover for Lean based on the superposition calculus. Duper can be called directly as a terminal tactic in interactive Lean proofs, but is also designed with proof reconstruction for a future Lean hammer in mind. In this paper, we describe Duper's underlying approach to proof search and proof reconstruction with a particular emphasis on the challenges of working in a dependent type theory. We also compare Duper's performance to Metis' on pre-existing benchmarks to give evidence that Duper is performant enough to be useful for proof reconstruction in a hammer.



# Premise Selection

# Premise selection for a hammer

Thomas, Josh, Albert, Sean, and I:

- implemented a neural method of premise selection.
- implemented the prototype hammer.
- set up evaluation procedures.
- tested, tuned, and evaluated the combination:
  - variations on the training
  - variations on the model size
  - variations on the core hammer algorithm.
  - variations on the number of premises used.
  - alternate methods for premise selection.

# Premise Selection for a Lean Hammer

Thomas Zhu<sup>1,\*</sup>, Joshua Clune<sup>1,\*</sup>,  
Jeremy Avigad<sup>1,†</sup>, Albert Q. Jiang<sup>2,†</sup>, Sean Welleck<sup>1,†</sup>

<sup>1</sup>Carnegie Mellon University, <sup>2</sup>Mistral AI

\*Equal contribution, <sup>†</sup>Equal advising

{thomaszh,jclune,avigad,swelleck}@andrew.cmu.edu, qj213@cam.ac.uk

## Abstract

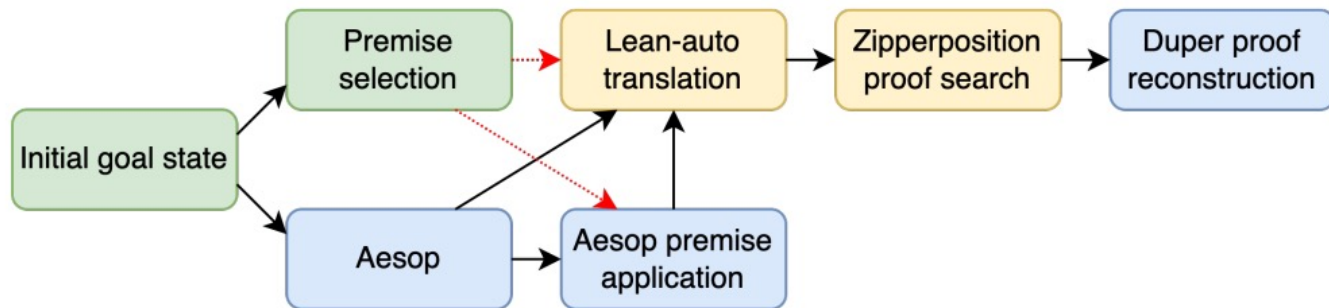
Neural methods are transforming automated reasoning for proof assistants, yet integrating these advances into practical verification workflows remains challenging. Hammers are tools that interface with external automatic theorem provers to automate tedious reasoning steps. They have dramatically improved productivity in proof assistants, but the Lean proof assistant still does not have a hammer despite its growing popularity. We present LeanHammer, the first end-to-end domain-general hammer for Lean, built on a novel neural premise selection system for a hammer in dependent type theory. Unlike existing Lean premise selectors, our approach dynamically adapts to user-specific contexts and combines with symbolic proof search and reconstruction to create a practical hammer. With comprehensive evaluations, we show that our premise selector enables LeanHammer to solve 21% more goals relative to existing premise selectors, and generalize well to diverse domains. Our work bridges the gap between neural retrieval and symbolic reasoning, making formal verification more accessible to researchers and practitioners.<sup>1</sup>



# The algorithm

The hammer uses Aesop (an internal backtracking search) with additional rules that enable it to use premise selection and call Lean-auto on subgoals.

Aesop may succeed on its own, but it can also call Lean-auto on subgoals with the premises, and that, in turn, calls Zipperposition.





# Premise selection

Our premise selector uses methods introduced in Mikula et al., “Magnushammer: A Transformer-Based Approach to Premise Selection.”

## Method:

- Collect positive and negative instances of (goal, premise) pairs.
- Use a transformer model and embed both into the same latent space.
- Nudge positive instances closer, negative instances further.

We include information like docstrings, namespaces in the training.

Premise selector	Model size	Recall (%)		Proof rate (%)				
		@16	@32	aesop	auto	aesop+auto	full	cumul
None		0.0	0.0	16.9	9.4	16.9	16.9	16.9
Random forest*		22.1	22.3	19.1	11.9	19.1	19.1	19.1
MePo		38.4	42.1	23.3	14.5	21.5	26.3	27.5
ReProver	218M	35.1 <sup>†</sup>	38.7 <sup>†</sup>	11.4	12.9	20.5	12.0	22.3
<b>Ours</b> (small)	23M	59.2	67.8	23.9	19.1	25.9	27.9	31.9
<b>Ours</b> (medium)	33M	58.6	68.1	23.1	20.1	26.1	28.5	30.7
<b>Ours</b> (large)	82M	<b>63.5</b>	<b>72.7</b>	<b>24.1</b>	<b>21.3</b>	<b>28.5</b>	<b>30.1</b>	<b>33.3</b>
<b>Ours</b> (large) $\cup$ MePo				28.9	23.9	30.3	35.9	37.6
<b>Ours</b> (cumulative)				27.5	25.5	31.1	34.5	37.3
<b>Ours</b> (cumulative) $\cup$ MePo				30.3	27.1	32.3	38.2	39.6
Ground truth				27.7	33.1	37.8	41.0	43.0

\*Performance upper bound, excluding errors. <sup>†</sup>Our definition is slightly different from [Yang et al. \(2023\)](#). See Appendix C.

# Premise selection

We host premise selection with an AWS server with a GPU.

We retrain the model with each release of Mathlib. The embeddings of Mathlib theorems are there.

Theorems from a local project have to be embedded when used. They are cached (even between users).

We provide instructions for setting up your own server.

# premise-selection

This repository provides a cloud-based Lean premise selector,

`Lean.PremiseSelection.Cloud.premiseSelector`. It sends the current goal state and any new user-defined premises (in current file or imported) to a cloud server, and returns the top `k` premises recommended by the server.

To use the selector:

```
import PremiseSelection

theorem add_comm_nat (a b : Nat) : a + b = b + a := Nat.add_comm ..

example (a b : Nat) : a + b = b + a := by
  premises -- prints premises including `add_comm_nat` and `Nat.add_comm`
```



The premise selector extends the `Lean.PremiseSelection` API introduced in Lean 4 core.

It is developed as part of [LeanHammer](#), which uses the cloud-based premise selector.

## Overview

The premise selection server backend runs a selector model on from Mathlib, Batteries, and Lean core. It uses an encoder-only transformer to embed premises and the goal state, and retrieves the top-`k` premises by cosine similarity.

For performance reasons (see below), the number of new premises that can be uploaded has an upper limit set by the server (e.g. 8192). A warning will be issued if this limit is surpassed, and extra new premises are truncated. This truncation prioritizes the new premises in the current module, and then premises in more recently imported modules.












# Other Automation

# Other automation

Other things that should be part of the hammer:

- Lean-SMT (the cvc5 team) calls cvc5 and constructs a Lean proof directly.
- Grind is a powerful new internal Lean prover.
- Canonical (Norman, A) does exhaustive search in dependent type theory and produces small, explicit terms.
- QuerySMT (Clune, Barbosa, A) has cvc5 generate theory-specific hints, inserts them in the Lean source, and uses Grind and Duper to construct a Lean proof.

# LEAN-SMT: An SMT Tactic for Discharging Proof Goals in Lean

Abdalkhman Mohamed<sup>1</sup> , Tomaz Masearenhas<sup>4</sup> , Harun Khan<sup>1</sup> ,  
Haniel Barbosa<sup>4</sup> , Andrew Reynolds<sup>2,3</sup> , Yicheng Qian<sup>1</sup> ,  
Cesare Tinelli<sup>2</sup> , and Clark Barrett<sup>1</sup> () 



<sup>1</sup> Stanford University, Stanford, USA  
barrett@cs.stanford.edu

<sup>2</sup> The University of Iowa, Iowa City, USA

<sup>3</sup> Amazon Web Services, Seattle, WA, USA

<sup>4</sup> Universidade Federal de Minas Gerais,  
Belo Horizonte, Brazil



**Abstract.** Lean is an increasingly popular proof assistant based on dependent type theory. Despite its success, it still lacks important automation features present in more seasoned proof assistants, such as the Sledgehammer tactic in Isabelle/HOL. A key aspect of Sledgehammer is the use of proof-producing SMT solvers to prove a translated proof goal and the reconstruction of the resulting proof into valid justifications for the original goal. We present LEAN-SMT, a tactic providing this functionality in Lean. We detail how the tactic converts Lean goals into SMT problems and, more importantly, how it reconstructs SMT proofs



# Canonical for Automated Theorem Proving in Lean

Chase Norman   

Carnegie Mellon University, Pittsburgh, PA, USA

Jeremy Avigad   

Carnegie Mellon University, Pittsburgh, PA, USA

---

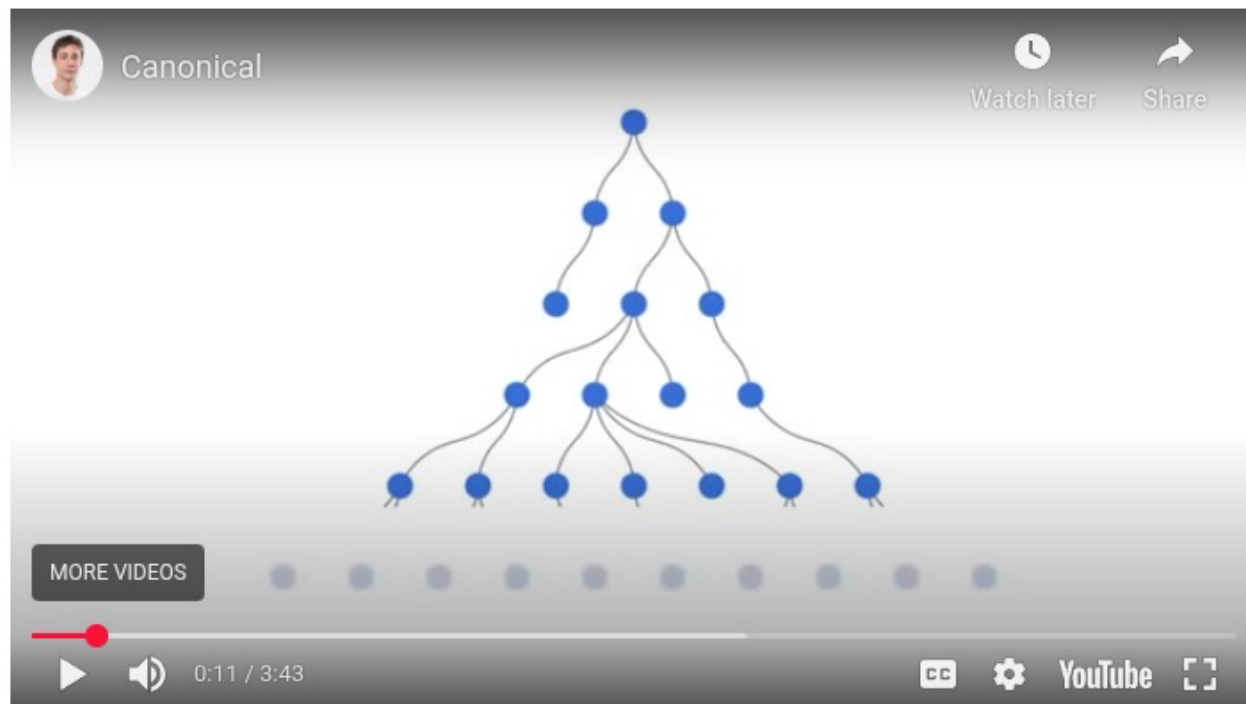
## Abstract

Canonical is a solver for type inhabitation in dependent type theory, that is, the problem of producing a term of a given type. We present a Lean tactic which invokes Canonical to generate proof terms and synthesize programs. The tactic supports higher-order and dependently-typed goals, structural recursion over indexed inductive types, and definitional equality. Canonical finds proofs for 84% of Natural Number Game problems in 51 seconds total.



# Canonical

Automated Theorem Proving for General Mathematics.



# Research

LeanHammer

QuerySMT

Duper

Keller Reduction

PLF

Zeus

## Hint-Based SMT Proof Reconstruction

```
example (Even Odd : Int → Prop)
(h1 : ∀ x : Int, ∀ y : Int, Odd (x) → Odd (y) → Even (x + y))
(h2 : ∀ x : Int, ∀ y : Int, Odd (x) → Even (y) → Odd (x + y))
(h3 : ∀ x : Int, Even (x) → ~ Odd (x))
(h4 : Odd (1)) : Even (10) := by
  querySMT
```

▼Tactic state  
No goals  
▼Suggestions  
Try this:  
apply @Classical.byContradiction  
Intro negGoal  
have sntLemma0 : Int.ofNat 5 + Int.ofNat 5 = Int.ofNat 10 := by grind  
have sntLemma1 : Int.ofNat 1 + Int.ofNat 4 = Int.ofNat 5 := by grind  
have sntLemma2 : Int.ofNat 1 + Int.ofNat 2 = Int.ofNat 3 := by grind  
have sntLemma3 : Int.ofNat 1 + Int.ofNat 1 = Int.ofNat 2 := by grind  
have sntLemma4 : Int.ofNat 1 + Int.ofNat 3 = Int.ofNat 4 := by grind  
duper [h1, h2, h3, h4, negGoal, sntLemma0, sntLemma1, sntLemma2, sntLemma3, sntLemma4] []

There are several paradigms for integrating interactive and automated theorem provers, combining the convenience of powerful automation with strong soundness guarantees. These paradigms vary both in how they translate problems between the technologies' respective languages, and in how they leverage proofs discovered by automation.

My research with [Haniel Barbosa](#) and [Jeremy Avigad](#) explores a new approach to reconstructing proofs found by SMT solvers. Rather than verifying or replaying a full proof produced by the SMT solver, or at the other extreme, rediscovering the solver's proof from just the set of premises it uses, we explore an approach which helps guide an interactive theorem prover's internal automation by leveraging derived facts during solving. This enables our Lean tactic, [QuerySMT](#), to use hints discovered by [cvc5](#) to generate structured proof scripts which don't depend on the external solver.

# Challenges



# Engineering

## Maintenance

- All the components of the pipeline (Duper, Lean-auto, the premise selection) must be kept up to date with Lean.
- The model has to be retrained regularly.
- Someone has to keep an eye on the server.

## Quality of life

- We should set up precompiled binaries for Duper.
- The server should handle different versions of Mathlib more gracefully.
- Kim Morrison has asked for a fork of Mathlib that uses the hammer.
- The sledgehammer should support parallel calls.



# Performance

We need to:

- Try more examples by hand.
- Study Lean-auto translation failures.
- Study Aesop failures.
- Experiment with other means of premise selection, and combinations thereof.
- Are there things that premise selection systematically misses?
- Try other back-end provers (like Vampire, cvc5).
- Try other means of proof reconstruction.
- Collect user feedback and examples.



Institute for Computer-Aided  
Reasoning in *Mathematics*



# **NSF invests over \$74 million in 6 mathematical sciences research institutes**

From improving medical care to detecting planets in other solar systems, the institutes will explore mathematical sciences with a broad range of applications

August 4, 2025

---

The U.S. National Science Foundation is investing over \$74 million in six research institutes focused on the mathematical sciences and their broad applications in all fields of science, technology and many industries.

For over 40 years, NSF has funded Mathematical Sciences Research Institutes to serve as catalysts for U.S. research in mathematics and statistics and to produce mathematical innovations to rapidly address new and emerging challenges and opportunities. The institutes collectively investigate a wide range of mathematical research areas with potential impacts, including better patient outcomes in hospital emergency rooms, enhanced safety of semiautonomous vehicles, and detection of exoplanets using quantum physics. Previous research conducted at the institutes has had broad impacts, such as improved speed and accuracy of MRI imaging and the development of mathematical foundations of artificial intelligence-based technologies.



# Mission

The mission of the *Institute for Computer-Aided Reasoning in Mathematics* is to:

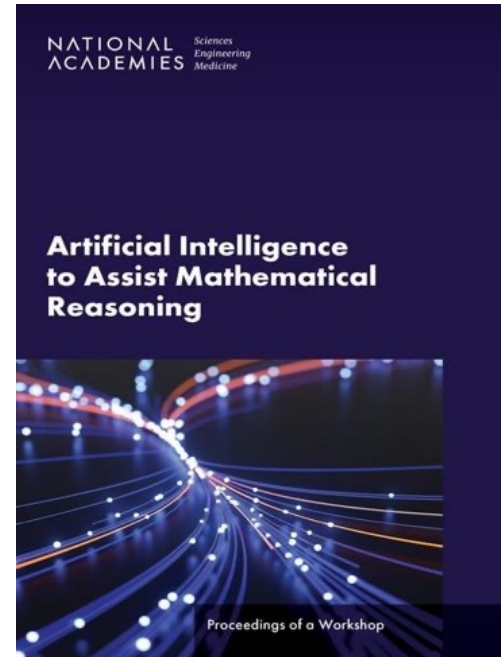
- empower mathematicians to take advantage of new technologies for mathematical reasoning and keep mathematics central to everything we do;
- unite mathematicians of all kinds, computer scientists, students, and researchers, to achieve that goal; and
- ensure that mathematics and the new technologies are accessible to everyone.



# Motivation

In 2023, a workshop by the *National Academies for Science, Engineering, and Medicine* explored the promise of these technologies and the challenges that lie ahead.

The Institute for Computer-Aided Reasoning in Mathematics is designed to meet the challenges.



# Empowering mathematicians

Some challenges:

- Existing tools aren't designed for mathematicians.
- Documentation isn't written for mathematicians.
- Mathematicians don't have the relevant expertise.
- Mathematicians don't have time to learn to use AI.
- Collaborations are needed between computer scientists and mathematicians.
- Nobody "owns" AI for mathematics; it falls through the cracks.
- Some of the work is tedious, doesn't yield academic credit.
- The mathematics community doesn't know how to support/assess mathematicians using AI.

# Empowering mathematicians

We will maintain a staff of innovation engineers who will:

- help mathematicians learn to use the technologies
- answer questions and provide technical support
- maintain documentation, tools, infrastructure, and other community resources
- serve as liaisons to computer science and industry
- carry out essential tasks that academics don't have time or incentives to do
- be community leaders in the use of technology
- gather resources and coordinate efforts.

# Bringing us together

We will also provide:

- workshops
- summer schools
- collaborative visits
- an annual conference

These will build a community of students, researchers, mathematicians, computer scientists, engineers, and others to address the challenges together.

We need a combination of perspectives and expertise.



# Improving access

AI and the digitization of mathematics can lead to greater democratization but it can also lead to greater inequities.

A central goal of ICARM is to ensure that all communities have the resources they need to participate in mathematics and take advantage of the new technologies.

Our original proposal included a summer school for college students, a workshop for graduate students, and an after-school program for high school students to address this challenge head on.

# Current status

The institute has been launched as a three-year pilot:

- 2-3 administrative staff
- 3 innovation engineers
- At least two workshops each year
- At least one summer school each year
- A conference in the second year
- Collaborative visits

# Current status

We are:

- Setting up administrative and financial infrastructure within CMU
- Constituting our governing boards
- Setting up our space
- Setting up our web pages and computing infrastructure
- Hiring staff and innovation engineers
- Starting to plan our first activities and events
- Collaborating with the other institutes

At the Joint Mathematics Meetings, we will hold tutorials, participate in the institutes' reception, and have a booth.



# Neurosymbolic AI



# Machine learning vs. formal methods

## Formal methods and symbolic AI

- based on logic and rules
- precise semantics
- easily get lost in a search

## Machine learning and neural AI

- synthesizes huge amounts of data
- can explore and learn
- can detect subtle patterns and develop intuitions
- it can be hard to tell what the answers mean

Both are important.

# A synthesis

A sledgehammer is a good example of neurosymbolic AI.

- Use neural AI to learn what facts are likely to be relevant.
- Use symbolic AI to fill in and check the details.

Understanding how to combine the two traditions effectively is arguably the most important challenge to AI today.