AITP 2025

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Natural-Language Proofs with Higher-Order Logic

Adam Dingle

Charles University, Prague

September 4, 2025

Natty: a natural-language proof assistant

- I described Natty at AITP last year
- today: an update on Natty's capabilities, challenges ahead

Can automatic provers follow steps in real-world proofs?

- Take a proof written in natural language
- Convert each step to a logical formula
- Use an automatic prover to verify each formula
- Is this possible in general?
  - If so, life is good!
  - If not, why not?

Can automatic provers follow steps in real-world proofs?

- Take a proof [in what domain?] written in [controlled?] natural language
- Convert each step to a logical formula [in what logic?]
- Use an automatic prover to verify each formula
- Is this possible in general? [how quickly? how reliably?]
  - If so, life is good!
  - If not, why not?

Can automatic provers follow steps in real-world proofs?

- Take an undergraduate textbook proof in lightly controlled natural language
- Convert each step to a formula in first-order or higher-order logic
- Use an automatic prover to verify each formula in < 30s (ideally < 5s), 100% of the time
- Is this possible in general?
  - Hopefully yes, but verification is not as easy as you might think
  - ...especially if there are many known theorems

Natty: a natural-language proof assistant

- Input: math in controlled natural language, as natural as possible
- Natty converts proof steps to formulas of higher-order logic
- ...and proves them using internal superposition-based prover
- Can export each proof step to a THF file for comparison with other provers
- Broadly, goals are similar to Naproche

Proof assistants: a spectrum of naturalness

Metamath	HOL Light	Isabelle	Mizar	Naproche Natty

⟵ Less natural — More natural ⟶

```
$( Prove a theorem $)
    th1 $p |- t = t $=
  $( Here is its proof: $)
      tt tze tpl tt weq tt tt weq tt a2 tt tze tpl
      tt weq tt tze tpl tt weq tt tt weq wim tt a2
      tt tze tpl tt tt a1 mp mp
    $.
```

Proof assistants: a spectrum of naturalness

Metamath	HOL Light	Isabelle	Mizar	Naproche Natty

⟵ Less natural — More natural ⟶

```
definition
  let n be Nat;
  func cseq n -> Real_Sequence means :Def3: :: IRRAT_1:def 3
  for k being Nat holds it . k = (n choose k) * (n ^ (- k));
  correctness
  proof end;
end;
```

Proof assistants: a spectrum of naturalness

Metamath	HOL Light	Isabelle	Mizar	Naproche Natty

⟵ Less natural | More natural ⟶

Let us show that there exists a natural number a and nonzero natural number b such that $q = \frac{a}{b}$. Take an integer a and a nonzero integer b such that $q = \frac{a}{b}$.

Case $a = 0$ or a, b are natural numbers. Trivial.

Case $(a < 0$ and $b > 0)$ or $(a > 0$ and $b < 0)$. Then $\frac{a}{b} < 0$ and $q \geq 0$. Contradiction. End.

Case $a < 0$ and $b < 0$. Then $-a, -b \in \mathbb{N}$ and $q = \frac{-a}{-b}$. End. End. Take a natural number a and a nontrivial natural number b such that $q = \frac{a}{b}$.

Sample input 1: definition of $\mathbb{N}$

Natural numbers: definition

Axiom 1. There exists a type $\mathbb{N}$ with an element $0 \in \mathbb{N}$ and a function $s : \mathbb{N} \to \mathbb{N}$ such that

a. There is no $n \in \mathbb{N}$ such that $s(n) = 0$.
b. For all $n, m \in \mathbb{N}$, if $s(n) = s(m)$ then $n = m$.
c. Let $P : \mathbb{N} \to \mathbb{B}$. If $0 \in P$, and $k \in P$ implies $s(k) \in P$ for all $k \in \mathbb{N}$, then $P = \mathbb{N}$.

Definition. Let $1 : \mathbb{N} = s(0)$.

Lemma 1. Let $a \in \mathbb{N}$. Suppose that $a \neq 0$. Then there is some $b \in \mathbb{N}$ such that $a = s(b)$.

Sample input 2: proof of right cancellation

Right cancellation of multiplication

Theorem 5. Let a, b, c $\in \mathbb{N}$. If c $\neq$ 0 and ac = bc then a = b.

Proof. Let

$$G = \{ x \in \mathbb{N} \mid \text{for all } y, z \in \mathbb{N}, \text{ if } z \neq 0 \text{ and } xz = yz \text{ then } x = y \}.$$

Let b, c $\in \mathbb{N}$ with c $\neq$ 0 and $0 \cdot c$ = bc. Then bc = 0. Since c $\neq$ 0, we must have b = 0 by Theorem 4.1. So 0 = b, and hence 0 $\in$ G.

Now let a $\in \mathbb{N}$, and suppose that a $\in$ G. Let b, c $\in \mathbb{N}$, and suppose that c $\neq$ 0 and $s(a) \cdot c$ = bc. Then by Theorem 3.5 we deduce that ca + c = bc. If b = 0, then either s(a) = 0 or c = 0, which is a contradiction. Hence b $\neq$ 0. By Lemma 1 there is some p $\in \mathbb{N}$ such that b = s(p). Therefore ca + c = $s(p) \cdot c$, and we see that ca + c = cp + c. It follows by Theorem 2.1 that ca = cp, so ac = pc. By hypothesis it follows that a = p. Therefore s(a) = s(p) = b. Hence s(a) $\in$ G, and we deduce that G = $\mathbb{N}$.

Controlled natural language input

- plain text with Unicode characters
- axioms, definitions, theorems with or without proofs
- rich set of synonyms for typical mathematical texts
- implicit multiplication, disambiguated using type information
- set comprehension notation
- Every type (e.g. $\mathbb{N}$) is also the universal set of that type (i.e. $\lambda x.\top$)
- Proof steps can say which theorem(s) to use ("By Theorem 3.5...")
  - ...though Natty currently ignores these annotations

Proof structure inference

- In natural-language proofs, block structure is often implicit
- Natty's parser outputs a linear series of proof steps
- Natty uses heuristics to arrange these steps into a tree, indicating where assumptions will be discharged
- Roughly speaking:
  1. Each introduced variable has a scope that is as small as possible
  2. Each assumption has a scope that is as large as possible, within the bounds of the previous constraint
- Natty also notices some words such as "now", "next" indicating that an assumption should end

Structure inference: words ending an assumption

Now let

$$G = \{ x \in \mathbb{N} \mid \text{for all } y \in \mathbb{N}, x < y \text{ or } x = y \text{ or } x > y \}.$$

We will show that $x \in G$ for all $x \in \mathbb{N}$. We start by showing that $0 \in G$. Let $y \in \mathbb{N}$. By Theorem 6.2 we know that $0 \leq y$. It follows that $y = 0$ or $y > 0$. Hence $0 \in G$.

Now let $x \in \mathbb{N}$, and suppose that $x \in G$. We will show that $s(x) \in G$. Let $y \in \mathbb{N}$. By hypothesis we know that $x < y$ or $x = y$ or $x > y$.

First suppose that $x < y$. Then there is some $p \in \mathbb{N}$ such that $x + p = y$. If $p = 0$, then $x = y$, so $s(x) > y$ by Theorem 6.1. If $p \neq 0$, then by Lemma 1 there is some $r \in \mathbb{N}$ such that $p = s(r)$, which implies that $x + s(r) = y$, which implies $s(x) + r = y$, so $s(x) \leq y$, so either $s(x) < y$ or $s(x) = y$.

Next suppose that $x = y$. Then by Theorem 6.1 it follows that $s(x) > x = y$. Finally, suppose that $x > y$. We know that $s(x) > x$, and by Theorem 6.6 it follows that $s(x) > y$.

Putting the cases together, we see that $s(x) < y$ or $s(x) = y$ or $s(x) > y$ always holds. Hence $s(x) \in G$, and we conclude that $G = \mathbb{N}$.

Interactive environment

- Visual Studio Code extension
- syntax coloring
- real-time type checking
- real-time proof checking
- Demo

Proof assistants: logical foundations

- First-order logic
  - Usual ZFC axioms or extensions such as MK (Morse-Kelley), TG (Tarski-Grothendieck)
  - Everything (functions, integers, ...) is built from sets
  - Mizar, Metamath, Naproche
- Higher-order set theory
  - TG axioms in higher-order logic
  - Megalodon, Naproche-ZF
- Classical higher-order logic
  - Evolved from Alonzo Church's work on simple type theory
  - Every variable has a type
  - Functions are primitive
  - Sets are usually functions of type $\tau \rightarrow \mathbb{B}$
  - HOL Light, Isabelle, Natty
- Dependent type theory
  - Martin-Lof type theory and descendents
  - Rocq, Lean
- Choice of foundation is visible to the user to some extent, and affects automated deduction

Natty's logic / type system

- Higher-order classical logic, like in HOL Light or Isabelle/HOL
- type system allows overloading
- $+ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ and $+ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$ are distinguished
- no parametric polymorphism yet
- currently every variable must have a type
- inductive types and recursive functions must be defined axiomatically

Comparison with Naproche

	Naproche	**Natty**
Logic	first-order	higher-order
Input	LaTeX	plain text with Unicode
Proof structure	explicit	implicit
Prover	usually E	internal
Written in	Haskell	OCaml
IDE	Isabelle	Visual Studio Code
Wiedjik thms proven	10	0

Natty's internal prover

- developed because E, Vampire, other higher-order ATPs unable to prove all steps
- goal: prove easy proof steps quickly
- non-goal: prove difficult theorems (e.g. from TPTP)
- Naproche uses external first-order ATPs, which seem to do better
- open question: could Natty also use external ATPs with the right tricks?

Superposition

- superposition = inference rule for combining two formulas
- first-order superposition calculus developed in 1990s (Bachmair, Ganzinger)
- grew out of resolution + term rewriting
- higher-order superposition calculus (Blanchette et al, 2023)

Natty's internal prover

- partially implements the higher-order superposition calculus
- pragmatic, incomplete implementation
- very limited higher-order unification
- uses DISCOUNT loop as found e.g. in E
- destructive term rewriting
- lexicographic path order, with mapping from higher-order to first-order terms

Preserving formula structure

- Most provers convert all input formulas to clause normal form
- Natty (mostly) keeps formulas intact
- example: Peano induction axiom

$$\forall P : (\mathbb{N} \to \mathbb{B}).(P(0) \to \forall k : \mathbb{N}.(P(k) \to P(s(k))) \to \forall n : \mathbb{N}.P(n))$$

- Some clausification steps happen at inference time
- Attempts to imitate human-level reasoning
- Seems to help performance a bit, at the cost of some code complexity
- Makes debugging a lot easier

Main DISCOUNT loop

- DISCOUNT loop keeps formulas in two sets: $P =$ processed, $U =$ unprocessed
- Initially P is empty, U contains all premises plus negated conjecture
- On each iteration:
  1. choose a formula F from the unprocessed set U [critical step]
  2. rewrite F using formulas in P; rewrite formulas in P using F
  3. add F to P
  4. generate new formulas by combining F with each formula in P
  5. add all newly generated formulas to U

Given formula selection

- Which formula to choose next from the unprocessed set?
- Most provers keep unprocessed formulas in multiple priority queues
- Natty has a single priority queue, ordered by formula cost
- Each superposition step has a cost, determined by a heuristic formula
- A formula's cost is the sum of the costs of all superpositions in its derivation

Heuristic cost of a superposition step

- Intuitively, "downhill" steps should be cheap
- In a downhill step, a formula's length and literal count do not increase
- Natty uses a decision tree to assign each step one of the costs 0, 1, 3, or ∞
- For now, this decision tree is constructed by hand
- Very roughly:
    - If a formula has fewer literals than both parents, or is shorter than both parents, cost is 0
    - If a formula has more literals than both parents, or is longer than both parents, cost is ∞
    - Otherwise, a resolution inference has cost 1, or a paramodulation inference has cost 3
- Special rules for definitions, goal clauses, inductive formulas

Learning the cost of a superposition step

- Can we use machine learning to derive a heuristic function?
- Modified Natty to record all formulas generated during the course of a proof
- Each recorded formula has about 30 features (e.g. length, # of literals)
- We also record whether each formula was actually used in the proof
- Logistic regression model predicts probability that a formula with given features will be used
- Cost of the superposition producing ϕ is $\max(0, -L)$, where L is the logit value predicted by the regression model for ϕ
- This is roughly $-\log(P)$, where P is the predicted probability that ϕ will be used
- Lasso (ℓ_1) regularization can perform feature selection

The learned cost function

// The cost of a superposition step producing ϕ from ψ_1, ψ_2.

$$
\begin{aligned}
&\textsc{Learned-Cost}(\phi) = \\
&\quad 0.668 \\
&\quad + 0.041 \text{ if } \phi \text{ was generated by paramodulation} \\
&\quad - 0.030 \text{ if any ancestor of } \phi \text{ is a hypothesis} \\
&\quad - 0.251 \text{ if any ancestor of } \phi \text{ is the goal formula} \\
&\quad - 0.007 \text{ if } \psi_1 \text{ or } \psi_2 \text{ is a definition} \\
&\quad - 0.399 \text{ if } \psi_1 \text{ or } \psi_2 \text{ is an inductive formula} \\
&\quad - 0.009 \text{ if } \mathrm{lits}(\phi) < \min(\mathrm{lits}(\psi_1), \mathrm{lits}(\psi_2)) \\
&\quad + 0.007 \text{ if } \mathrm{lits}(\phi) > 1 \\
&\quad + 0.082 \cdot (\mathrm{lits}(\phi) - \mathrm{lits}(\psi_2)) \\
&\quad + 0.008 \cdot (\mathrm{weight}(\phi) - \max(\mathrm{weight}(\psi_1), \mathrm{weight}(\psi_2))) \\
&\quad + 0.002 \cdot (\mathrm{weight}(\phi) - \mathrm{weight}(\psi_2)) \\
&\quad - 0.182 \text{ if } \phi \text{ was generated by resolution and} \\
&\qquad\qquad \mathrm{weight}(\phi) < \min(\mathrm{weight}(\psi_1), \mathrm{weight}(\psi_2))
\end{aligned}
$$

Which cost function to use?

- Learned cost function performed about as well as the hand-generated decision tree
- Which is better? Choose your poison
- For now, Natty uses the hand-generated decision tree

What can Natty prove? How fast is it?

- Input file `nat.n`
- defines $\mathbb{N}$ axiomatically using Peano postulates
- asserts/proves many basic identities about $\mathbb{N}$ (40 theorems, 225 proof steps)
- defines $\mathbb{Z}$ axiomatically as isomorphic to an equivalence class of $\mathbb{N}$ x $\mathbb{N}$
- asserts/proves many basic identities about $\mathbb{Z}$ (28 theorems, 167 proof steps)

Performance comparison

With a 30-second time limit:

Table: Proof steps ($\mathbb{N}$)

	Natty	E	Vampire	Zipperposition
proved (of 225)	225	194	198	207
proved (%)	100%	86%	88%	92%
average time	0.17	0.45	1.01	0.88

Table: Proof steps ($\mathbb{Z}$)

	Natty	E	Vampire	Zipperposition
proved (of 167)	157	152	146	128
proved (%)	94%	91%	87%	77%
average time	1.08	0.20	1.05	1.35

Note that Natty has term no indexing yet!

Future work

- Goal: verify more math, starting with number theory
- Expand controlled natural language
- Prover enhancements
  - term index
  - use theorem references such as "By theorem 4, ..."
  - premise selection/weighting
  - experiment with lexicographic path order vs. Knuth-Bendix ordering
  - experiment with literal selection
  - possibly use E or Vampire some of the time, e.g. for first-order inference
- Create a benchmark suite of formulas derived from natural-language proof steps

Open questions

- Is superposition the best approach for proving “easy” proof steps?
- Sometimes steps that look trivial take a very long time to prove!
- Destructive term rewriting can transform an easy problem into a hard one
  - What to do about this?
- Superposition is not really goal-oriented
- Can we make it more like A*, favoring steps that take us closer to the goal?

Questions?