How to Speak Logic with Your Computer

The unreasonable effectiveness of a machine assistant

Asta Halkjær From, postdoc, SDPS, 2025-01-07

Who Am I

- ~~Magician~~ Logician
- PhD from DTU
- Postdoc in the SDPS section: Software, Data, People & Society
- I live with my fiancée and Betty Rambo (pictured)

~~Magician~~

Logician?

- Domain that we care about
    - Facts
- Language for describing the domain
    - $\wedge$, $\Rightarrow$
- Meaning of the language
    - A $\wedge$ B means “A and B”
    - A $\Rightarrow$ B means “A implies B”
- **Reasoning transforms knowledge**

$$\frac{A \qquad B}{A \wedge B}$$

$$\frac{A \qquad A \Rightarrow B}{B}$$

What Do I Do?

Hybrid Logic

"All gamers want to sit next to a gamer"

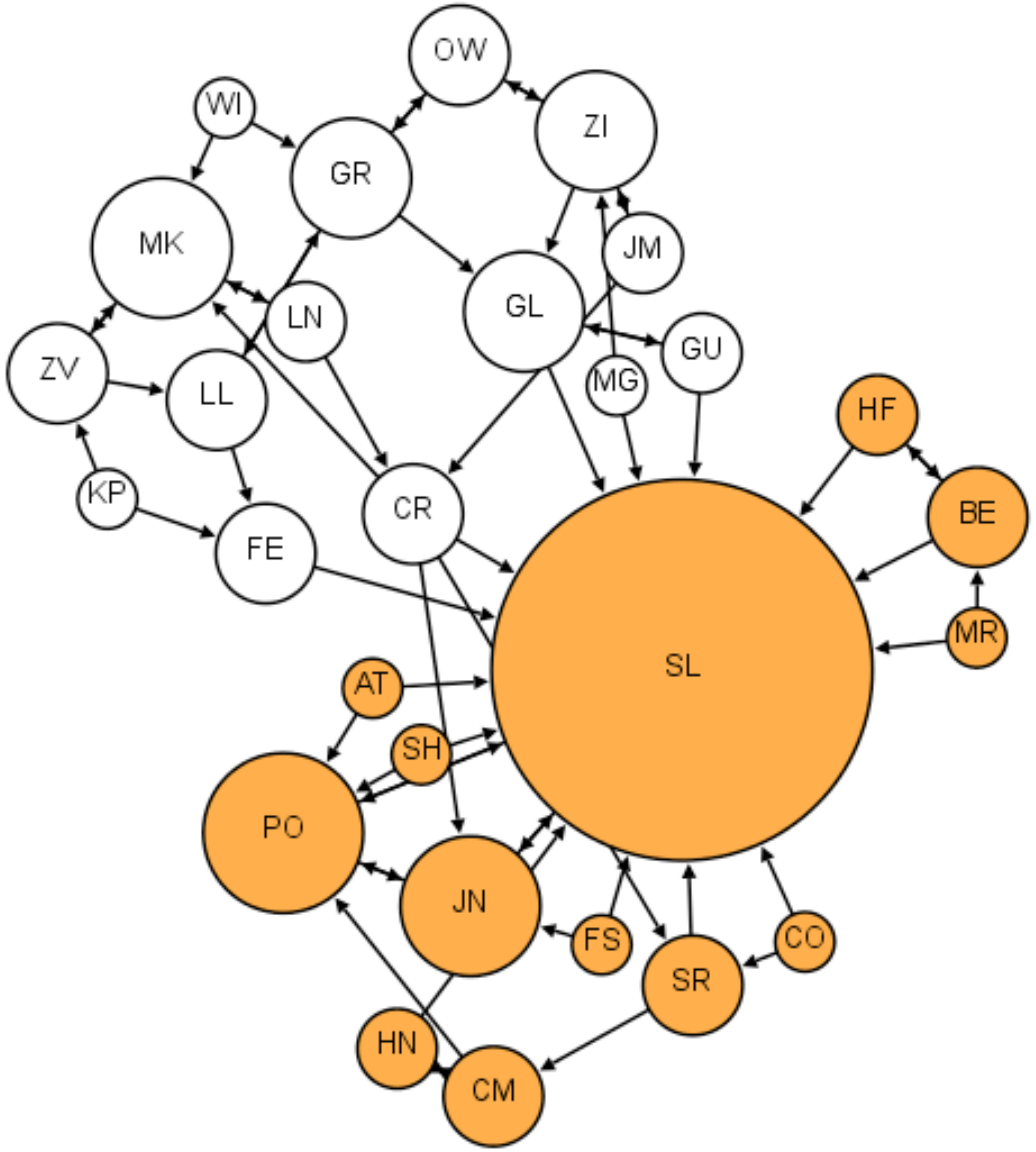
A (gamer ⇒ ◇ gamer)

Ida wants to sit next to Karla

Karla is a gamer

$$\frac{@i\ \Diamond k \qquad @k\ A}{@i\ \Diamond A}$$

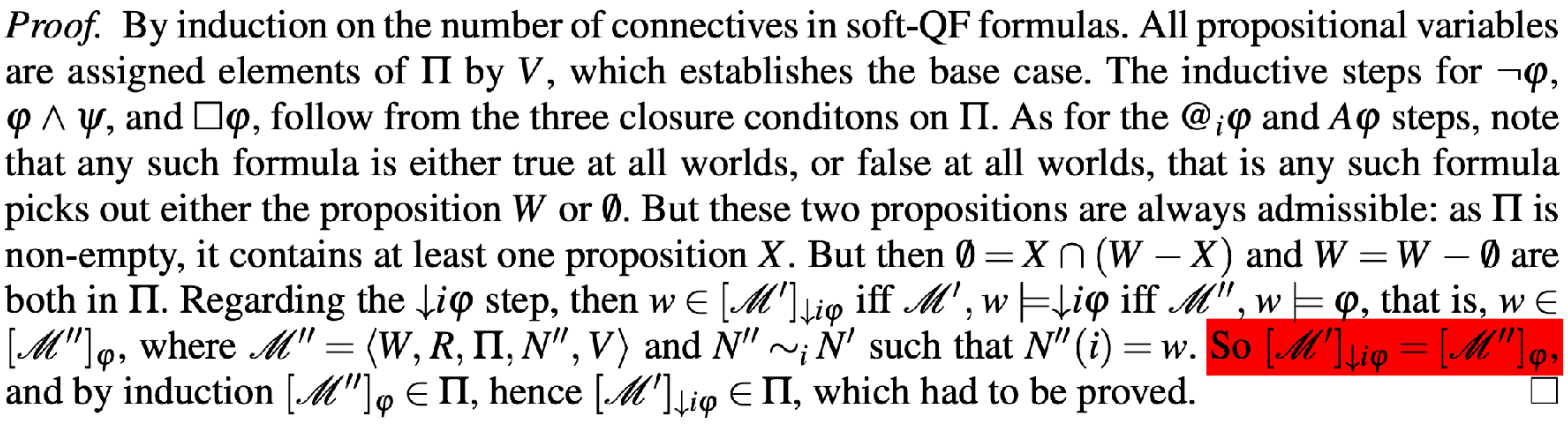
Ida wants to sit next to a gamer



Martin Grandjean. https://www.martingrandjean.ch/social-network-analysis-visualization-morenos-sociograms-revisited/

Spot the Error

(I highlighted it for your convenience)



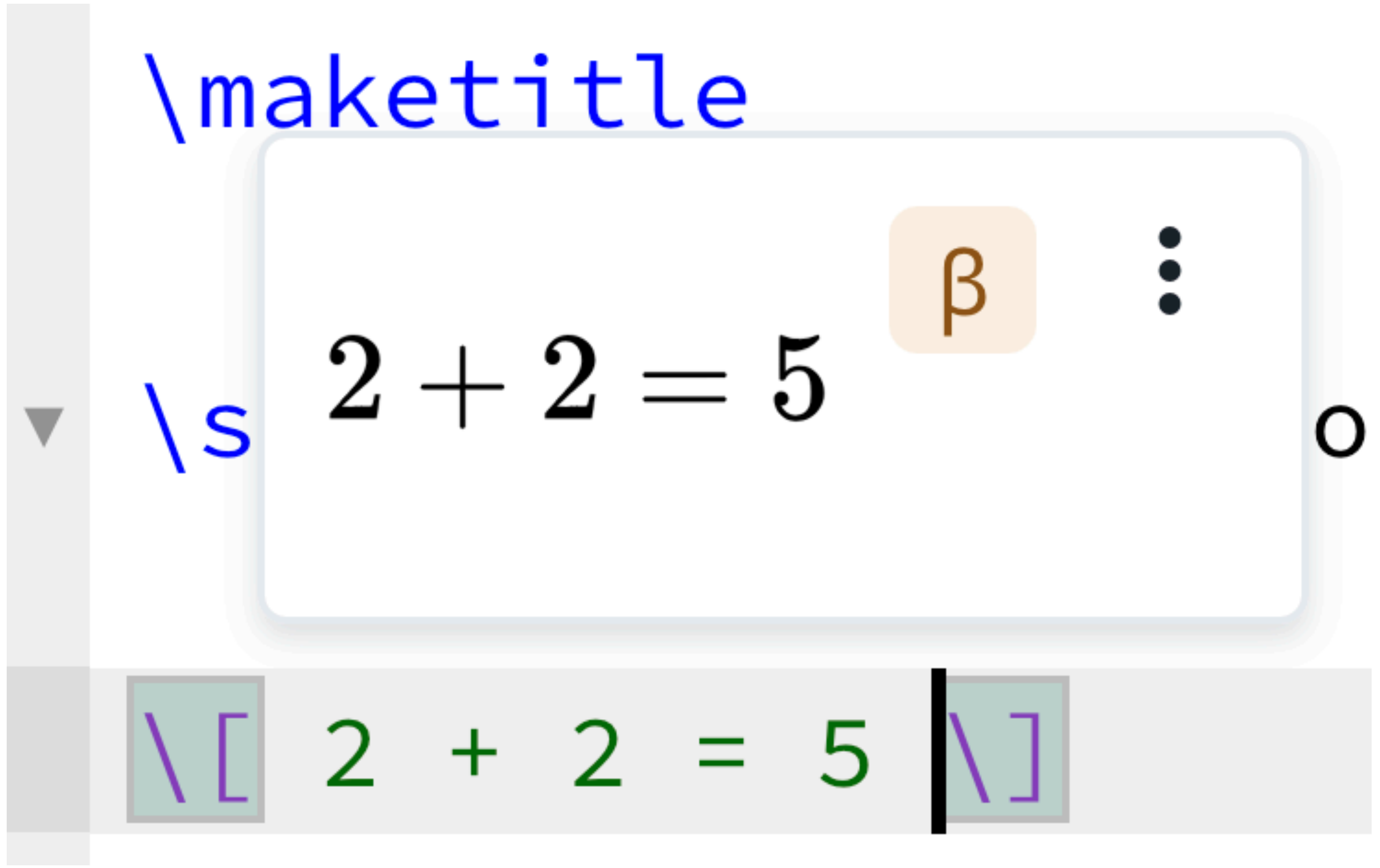
Proof. By induction on the number of connectives in soft-QF formulas. All propositional variables are assigned elements of Π by V, which establishes the base case. The inductive steps for $\neg\varphi$, $\varphi \wedge \psi$, and $\Box\varphi$, follow from the three closure conditons on Π. As for the $@_i\varphi$ and $A\varphi$ steps, note that any such formula is either true at all worlds, or false at all worlds, that is any such formula picks out either the proposition W or $\emptyset$. But these two propositions are always admissible: as Π is non-empty, it contains at least one proposition X. But then $\emptyset = X \cap (W - X)$ and $W = W - \emptyset$ are both in Π. Regarding the $\downarrow i\varphi$ step, then $w \in [\mathscr{M}']_{\downarrow i\varphi}$ iff $\mathscr{M}', w \models \downarrow i\varphi$ iff $\mathscr{M}'', w \models \varphi$, that is, $w \in [\mathscr{M}'']_{\varphi}$, where $\mathscr{M}'' = \langle W, R, \Pi, N'', V \rangle$ and $N'' \sim_i N'$ such that $N''(i) = w$. So $[\mathscr{M}']_{\downarrow i\varphi} = [\mathscr{M}'']_{\varphi}$, and by induction $[\mathscr{M}'']_{\varphi} \in \Pi$, hence $[\mathscr{M}']_{\downarrow i\varphi} \in \Pi$, which had to be proved. $\Box$

Private preprint.

Landesman & Litt and the Putnam-Wieland Conjecture

https://mathstodon.xyz/@littmath/113755426360011273

- "There was **an error**, in one of the last couple of paragraphs of the paper"
- "We … found **an error**, in a reasonably well-cited paper published in 2016"
- "a similar **erroneous lemma** appeared in **4 different papers, with 3 distinct wrong proofs**"
- "there was **an error in a key lemma**"
- "Most of these errors were found because the theorem being proved was wrong. **In all cases, the proofs as written were very plausible.**"



```
\maketitle

\s                                o

\[ 2 + 2 = 5 \]
```

$$2+2=5$$

littmath

@littmath@mathstodon.xyz

@jannem I think formal verification is ultimately where we’re heading!

https://mathstodon.xyz/@littmath/113756062568413960

Large Language Models?

“Everyone gets bidirectional BFS wrong”

“For fun, I also tested a few easily available LLMs with simple prompts. All of those I tested generated wrong (albeit original) code:

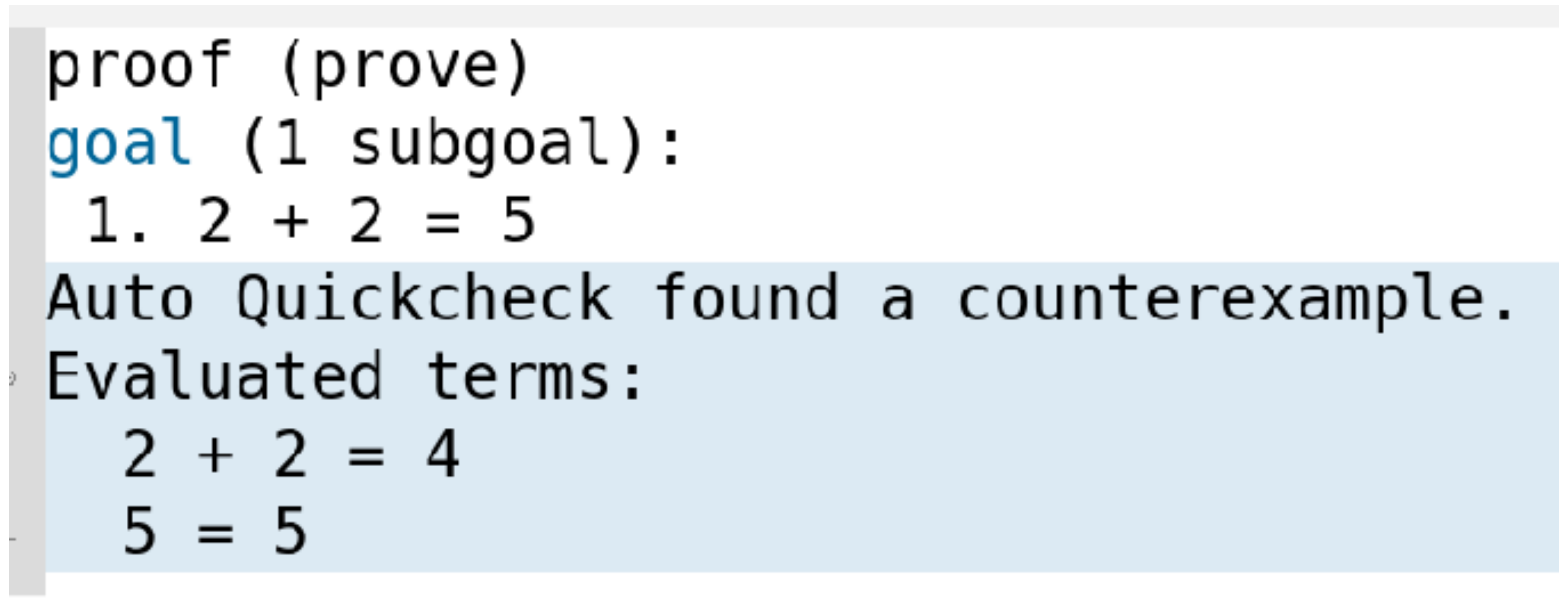
- (Python) ChatGPT 4o (has our bug)
- (C++) ChatGPT 4o (has our bug)
- (Python) Claude Haiku (has our bug, but also returns a completely wrong path sometimes)
- GitHub Copilot (Codex) guessed the wrong path not once but twice while trying to help me write this blogpost”

https://zdimension.fr/everyone-gets-bidirectional-bfs-wrong/

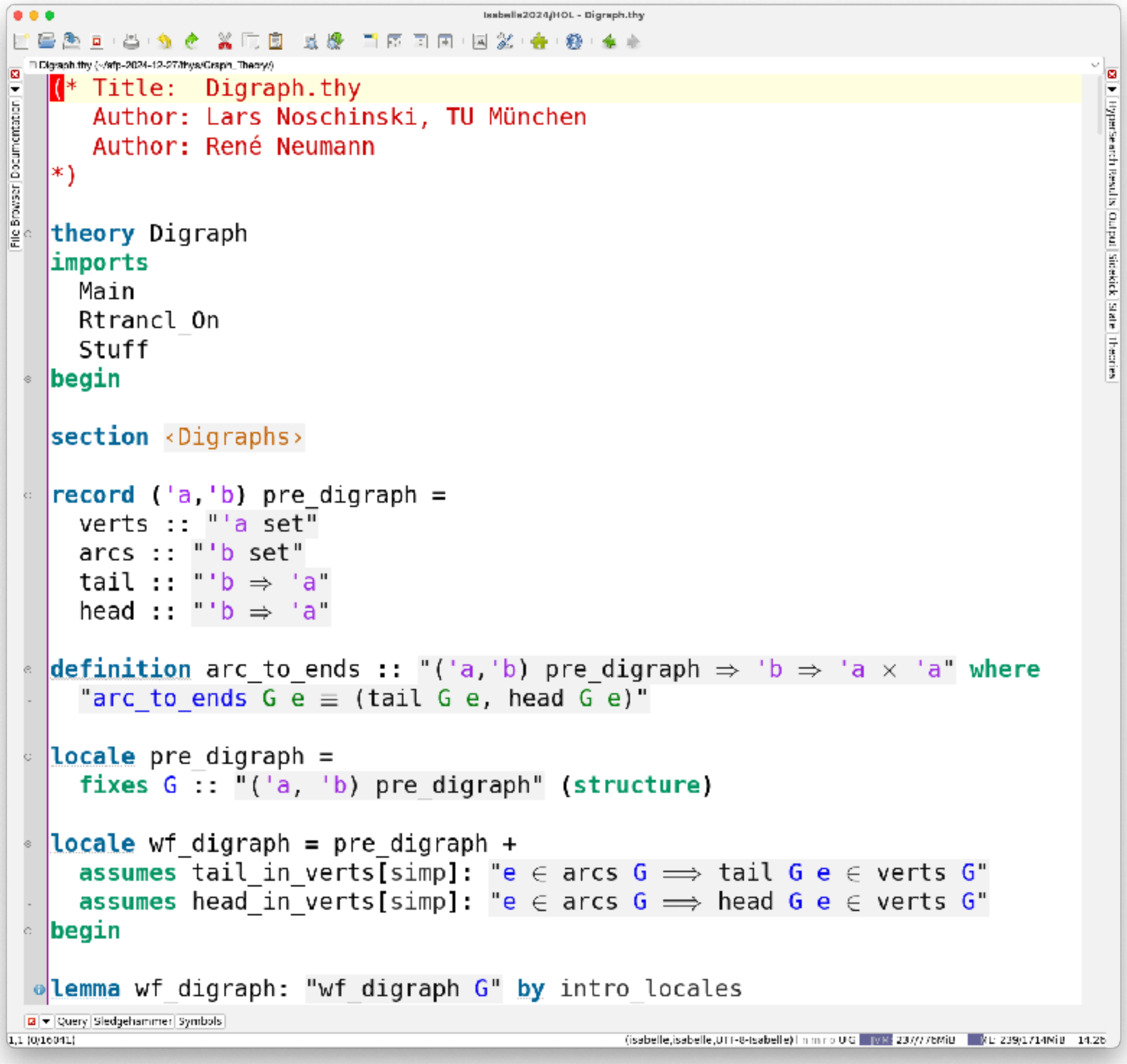
Formal Verification?

Isabelle/HOL

- English is ambiguous
- Logic is completely formal
    - We can teach a computer logic
- “HOL = Functional Programming + Logic”



```
proof (prove)
goal (1 subgoal):
 1. 2 + 2 = 5
Auto Quickcheck found a counterexample.
Evaluated terms:
  2 + 2 = 4
  5 = 5
```



```
(* Title:  Digraph.thy
   Author: Lars Noschinski, TU München
   Author: René Neumann
*)

theory Digraph
imports
  Main
  Rtrancl_On
  Stuff
begin

section ‹Digraphs›

record ('a,'b) pre_digraph =
  verts :: "'a set"
  arcs :: "'b set"
  tail :: "'b ⇒ 'a"
  head :: "'b ⇒ 'a"

definition arc_to_ends :: "('a,'b) pre_digraph ⇒ 'b ⇒ 'a × 'a" where
  "arc_to_ends G e ≡ (tail G e, head G e)"

locale pre_digraph =
  fixes G :: "('a, 'b) pre_digraph" (structure)

locale wf_digraph = pre_digraph +
  assumes tail_in_verts[simp]: "e ∈ arcs G ⟹ tail G e ∈ verts G"
  assumes head_in_verts[simp]: "e ∈ arcs G ⟹ head G e ∈ verts G"
begin

lemma wf_digraph: "wf_digraph G" by intro_locales
```

Rules for propositional connectives:

$$\frac{P \qquad Q}{P \wedge Q}\,conjI \qquad \frac{P \wedge Q \qquad [\![P; Q]\!] \Longrightarrow R}{R}\,conjE$$

$$\frac{P \wedge Q}{P}\,conjunct1 \qquad \frac{P \wedge Q}{Q}\,conjunct2$$

$$\frac{P}{P \vee Q}\,disjI1 \qquad \frac{Q}{P \vee Q}\,disjI2 \qquad \frac{P \vee Q \qquad P \Longrightarrow R \qquad Q \Longrightarrow R}{R}\,disjE \qquad \frac{P \Longrightarrow Q}{P \longrightarrow Q}\,impI$$

$$\frac{P \longrightarrow Q \qquad P \qquad Q \Longrightarrow R}{R}\,impE \qquad \frac{P \longrightarrow Q \qquad P}{Q}\,mp$$

$$\frac{P \Longrightarrow False}{\neg P}\,notI \qquad \frac{\neg P \qquad P}{R}\,notE \qquad \frac{\neg P \Longrightarrow False}{P}\,ccontr \qquad \frac{P \Longrightarrow Q \qquad Q \Longrightarrow P}{P \longleftrightarrow Q}\,iffI$$

Rules for quantifiers:

$$\frac{\bigwedge x.\ P\ x}{\forall x.\ P\ x}\,allI \qquad \frac{\forall x.\ P\ x \qquad P\ x \Longrightarrow R}{R}\,allE \qquad \frac{\forall x.\ P\ x}{P\ x}\,spec$$

$$\frac{P\ x}{\exists x.\ P\ x}\,exI \qquad \frac{\exists x.\ P\ x \qquad \bigwedge x.\ P\ x \Longrightarrow Q}{Q}\,exE$$

Dmitriy Traytel. IPA course.

Functional Data Structures and Algorithms

Tobias Nipkow, Jasmin Blanchette, Manuel Eberl, Alejandro Gómez-Londoño, Peter Lammich, Christian Sternagel, Simon Wimmer, Bohua Zhan

This book is an introduction to data structures and algorithms for functional languages, with a focus on proofs. It covers both functional correctness and running time analysis. It does so in a unified manner with inductive proofs about functional programs and their running time functions. All proofs have been machine-checked by the proof assistant Isabelle. The pdf contains links to the corresponding Isabelle theories.

https://functional-algorithms-verified.org/

2.4 Top-Down Merge Sort

Merge sort is another prime example of a divide-and-conquer algorithm, and one whose running time is guaranteed to be $O(n \lg n)$. We will consider three variants and start with the simplest one: split the list into two halves, sort the halves separately and merge the results.

```
merge :: 'a list ⇒ 'a list ⇒ 'a list

merge [] ys = ys
merge xs [] = xs
merge (x # xs) (y # ys)
= (if x ≤ y then x # merge xs (y # ys) else y # merge (x # xs) ys)

msort :: 'a list ⇒ 'a list

msort xs
= (let n = |xs|
   in if n ≤ 1 then xs
      else merge (msort (take (n div 2) xs)) (msort (drop (n div 2) xs)))
```

https://functional-algorithms-verified.org/functional_data_structures_algorithms.pdf#page=23

2.4.1 Functional Correctness

We start off with multisets and sets of elements:

$$\mathit{mset}\ (\mathit{merge}\ \mathit{xs}\ \mathit{ys}) = \mathit{mset}\ \mathit{xs} + \mathit{mset}\ \mathit{ys} \tag{2.10}$$

$$\mathit{set}\ (\mathit{merge}\ \mathit{xs}\ \mathit{ys}) = \mathit{set}\ \mathit{xs} \cup \mathit{set}\ \mathit{ys} \tag{2.11}$$

Proposition (2.10) is proved by induction on the computation of *merge* and (2.11) is an easy consequence.

Lemma 2.2. $\mathit{mset}\ (\mathit{msort}\ \mathit{xs}) = \mathit{mset}\ \mathit{xs}$

Proof by induction on the computation of *msort*. Let $n = |\mathit{xs}|$. The base case ($n \leq 1$) is trivial. Now assume $n > 1$ and let $\mathit{ys} = \mathit{take}\ (n \text{ div } 2)\ \mathit{xs}$ and $\mathit{zs} = \mathit{drop}\ (n \text{ div } 2)\ \mathit{xs}$.

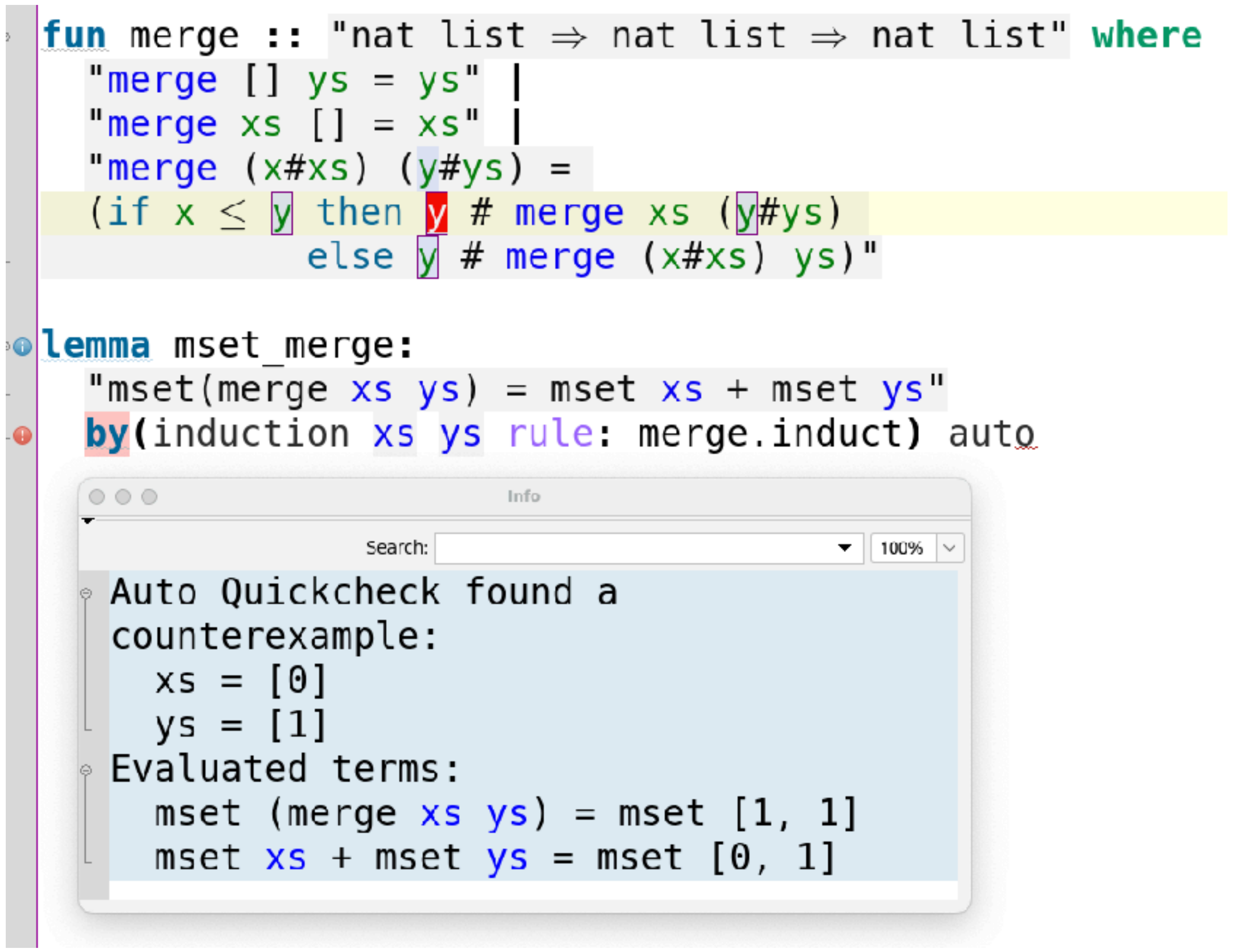
$$\begin{aligned} & \mathit{mset}\ (\mathit{msort}\ \mathit{xs}) = \mathit{mset}\ (\mathit{msort}\ \mathit{ys}) + \mathit{mset}\ (\mathit{msort}\ \mathit{zs}) && \text{by (2.10)} \\ & = \mathit{mset}\ \mathit{ys} + \mathit{mset}\ \mathit{zs} && \text{by IH} \\ & = \mathit{mset}\ (\mathit{ys}\ @\ \mathit{zs}) \\ & = \mathit{mset}\ \mathit{xs} && \square \end{aligned}$$

```
lemma mset_merge: "mset(merge xs ys) = mset xs + mset ys"
  by(induction xs ys rule: merge.induct) auto

lemma mset_msort: "mset (msort xs) = mset xs"
proof(induction xs rule: msort.induct)
  case (1 xs)
  let ?n = "length xs"
  let ?ys = "take (?n div 2) xs"
  let ?zs = "drop (?n div 2) xs"
  show ?case
  proof cases
    assume "?n ≤ 1"
    thus ?thesis by(simp add: msort.simps[of xs])
  next
    assume "¬ ?n ≤ 1"
    hence "mset (msort xs) = mset (msort ?ys) + mset (msort ?zs)"
      by(simp add: msort.simps[of xs] mset_merge)
    also have "... = mset ?ys + mset ?zs"
      using ‹¬ ?n ≤ 1› by(simp add: "1.IH")
    also have "... = mset (?ys @ ?zs)" by (simp del: append_take_drop_id)
    also have "... = mset xs" by simp
    finally show ?thesis .
  qed
qed
```

So What?

We have a machine assistant

- Common language
  - Immediate feedback
  - Eases collaboration
- Search for counter-examples
  - Good for your sanity
- No “easy exercise left for the reader”
  - (oops it was actually hard)



```
fun merge :: "nat list ⇒ nat list ⇒ nat list" where
  "merge [] ys = ys" |
  "merge xs [] = xs" |
  "merge (x#xs) (y#ys) =
  (if x ≤ y then y # merge xs (y#ys)
            else y # merge (x#xs) ys)"

lemma mset_merge:
  "mset(merge xs ys) = mset xs + mset ys"
  by(induction xs ys rule: merge.induct) auto
```

```
Auto Quickcheck found a
counterexample:
  xs = [0]
  ys = [1]
Evaluated terms:
  mset (merge xs ys) = mset [1, 1]
  mset xs + mset ys = mset [0, 1]
```

Archive of Formal Proofs

https://www.isa-afp.org/topics/

Computer science (3)

- Algorithms (51)
- Artificial intelligence (3)
- Automata and formal languages (58)
- Concurrency (11)
- Data management systems (6)
- Data structures (69)
- Functional programming (25)
- Hardware (2)
- Machine learning (2)
- Networks (7)
- Programming languages (4)
- Security (54)
- Semantics and reasoning (26)
- System description languages (8)

Logic (2)

- Computability (10)
- General logic (9)
- Philosophical aspects (12)
- Proof theory (23)
- Rewriting (23)
- Set theory (15)

Mathematics (2)

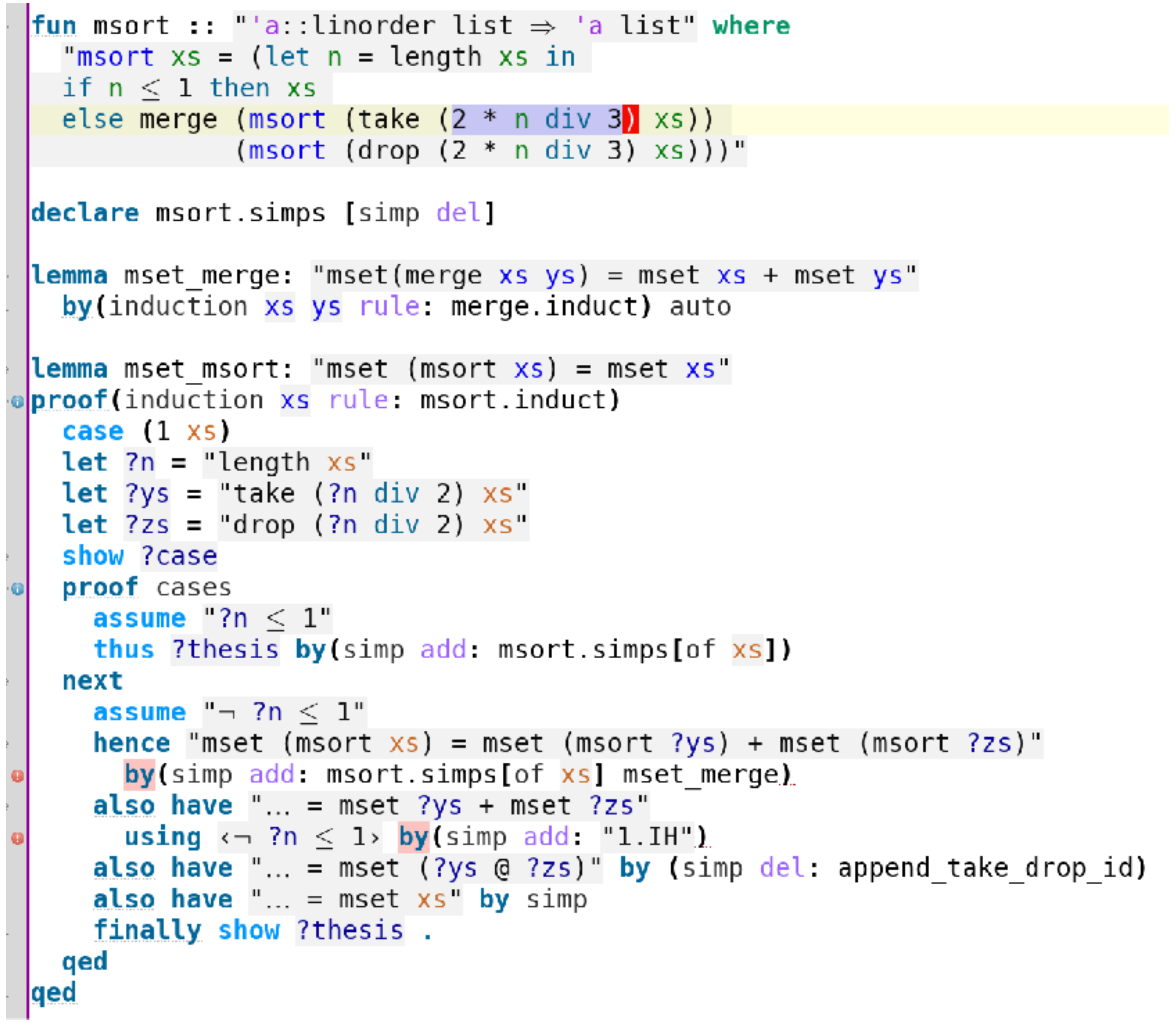
- Algebra (99)
- Analysis (59)
- Category theory (16)
- Combinatorics (46)
- Games and economics (18)
- Geometry (25)
- Graph theory (27)
- Measure and integration (6)
- Misc (4)
- Number theory (50)
- Order (10)
- Physics (6)
- Probability theory (28)
- Topology (10)

Tools (27)

Terence Tao’s Experience I

https://mathstodon.xyz/@tao/111360298114925842

- “formalization is generally **much slower**”
- “If … I found it necessary to tweak some parameter or hypothesis …, **the Lean compiler would quickly identify all the other parts of the argument that might need changing**, and it was a fairly quick matter to modify the proof to get back to a working state.”
- “in Mathematical English, it is all too common to introduce one or more small errors which tended to be quite **tedious to detect and eradicate.**”

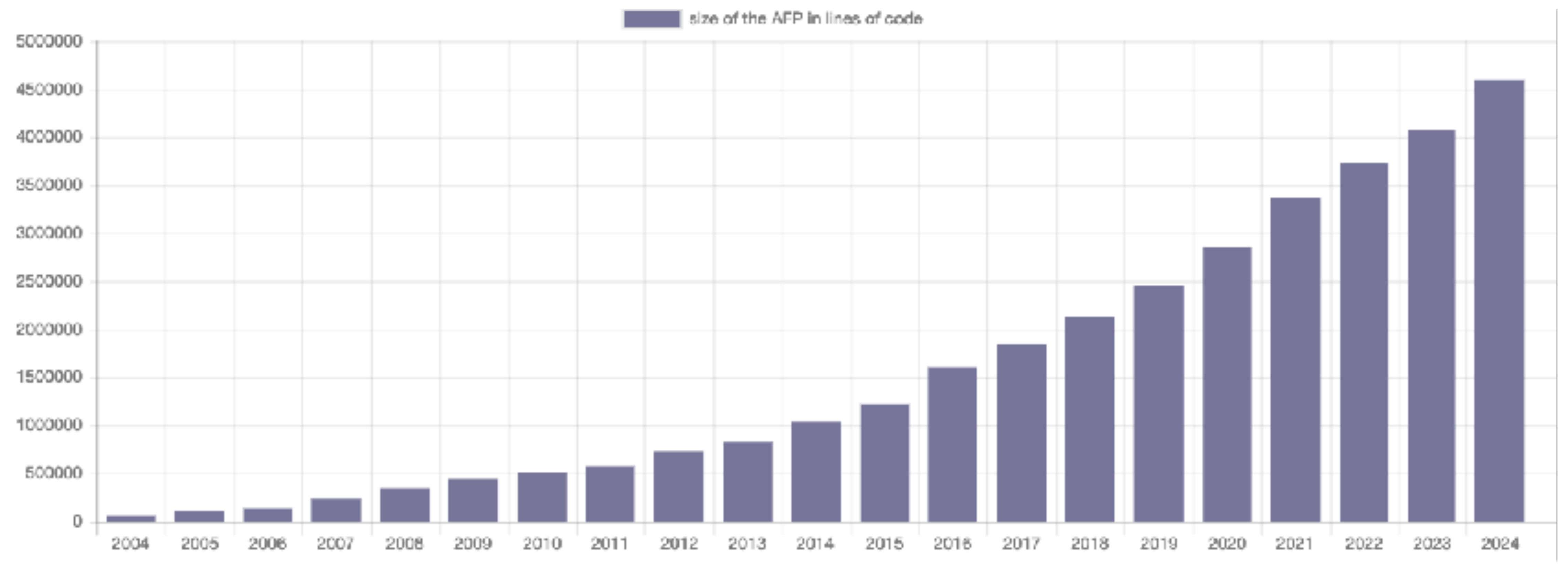


```
fun msort :: "'a::linorder list ⇒ 'a list" where
  "msort xs = (let n = length xs in
  if n ≤ 1 then xs
  else merge (msort (take (2 * n div 3) xs))
             (msort (drop (2 * n div 3) xs)))"

declare msort.simps [simp del]

lemma mset_merge: "mset(merge xs ys) = mset xs + mset ys"
  by(induction xs ys rule: merge.induct) auto

lemma mset_msort: "mset (msort xs) = mset xs"
proof(induction xs rule: msort.induct)
  case (1 xs)
  let ?n = "length xs"
  let ?ys = "take (?n div 2) xs"
  let ?zs = "drop (?n div 2) xs"
  show ?case
  proof cases
    assume "?n ≤ 1"
    thus ?thesis by(simp add: msort.simps[of xs])
  next
    assume "¬ ?n ≤ 1"
    hence "mset (msort xs) = mset (msort ?ys) + mset (msort ?zs)"
      by(simp add: msort.simps[of xs] mset_merge)
    also have "... = mset ?ys + mset ?zs"
      using ‹¬ ?n ≤ 1› by(simp add: "1.IH")
    also have "... = mset (?ys @ ?zs)" by (simp del: append_take_drop_id)
    also have "... = mset xs" by simp
    finally show ?thesis .
  qed
qed
```

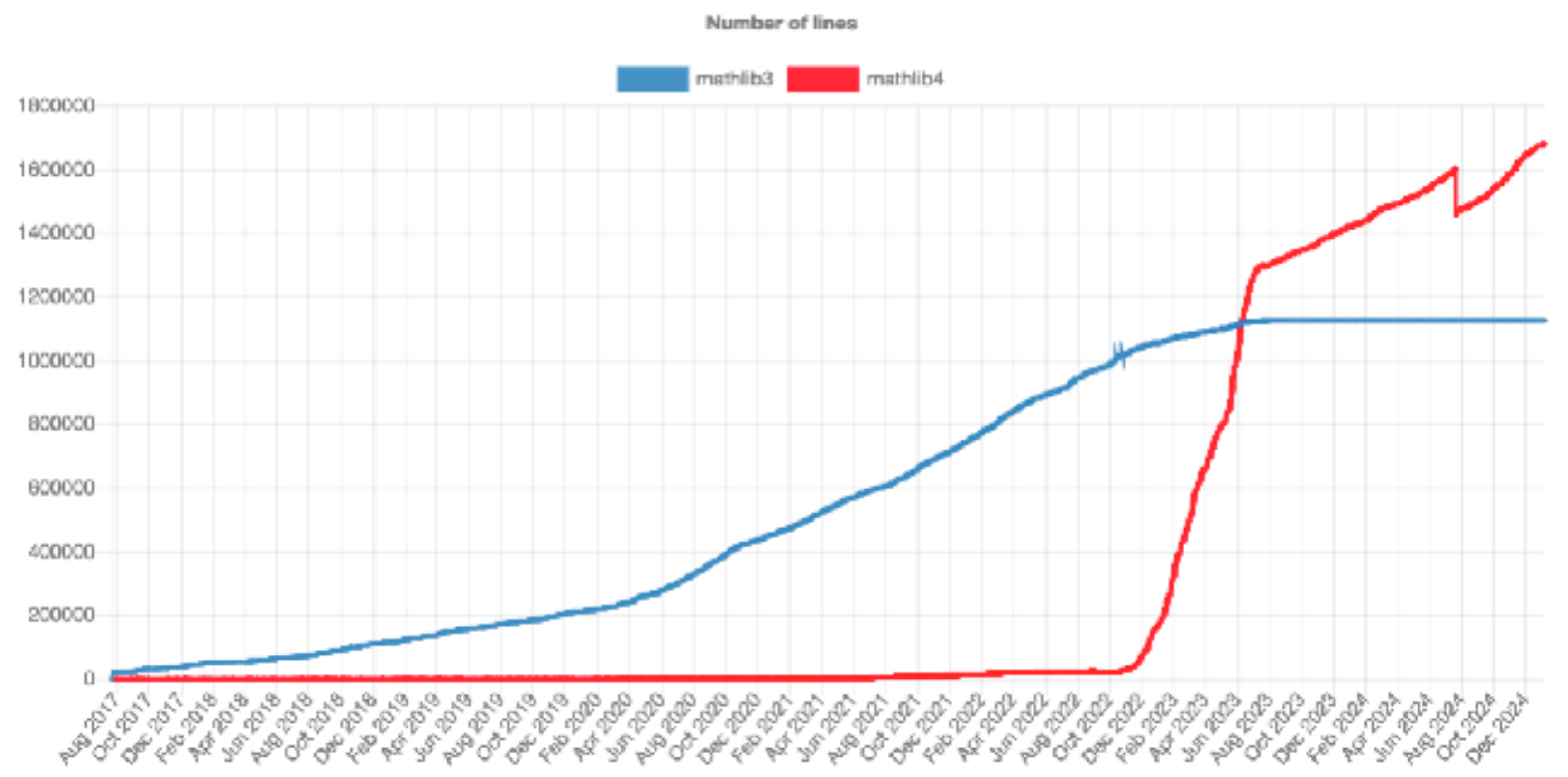
Terence Tao’s Experience II

https://mathstodon.xyz/@tao/111256629476328055

- “[proof assistants] facilitate **large-scale collaboration** with individuals without requiring high pre-established levels of trust”
- “I have already received a number of **pull requests** from Lean experts”
- “The fact that I could verify that the suggested code compiled without breaking anything else made it **far easier to accept these changes than if this were a traditional LaTeX-based proof writing project**.”



https://www.isa-afp.org/statistics/

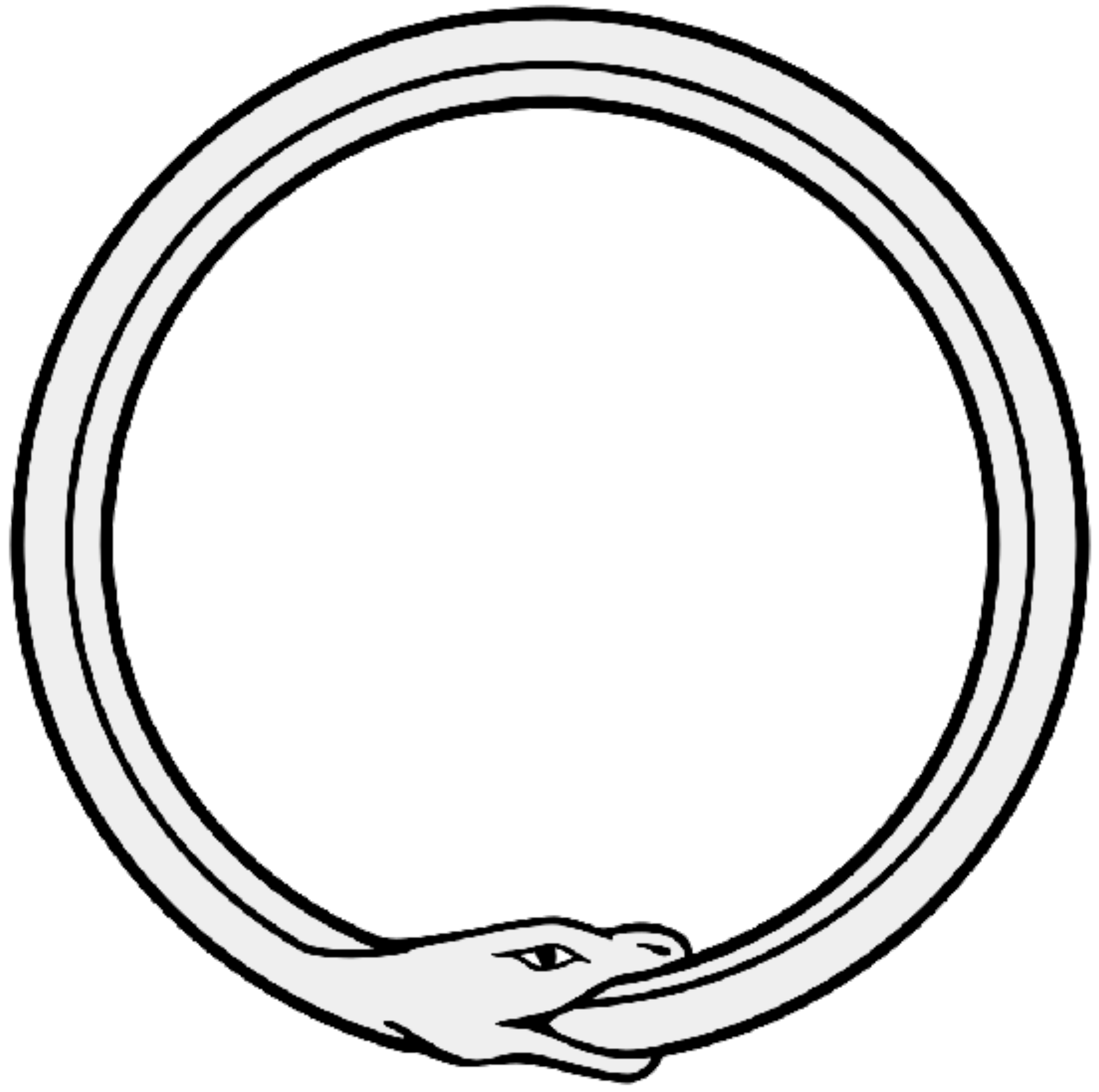


https://leanprover-community.github.io/mathlib_stats.html

What Do I Do?

Logic in Logic

- A lot of proofs in logic use the same recipe
  - Paper 1: A, B, C, D
  - Paper 2: A, B, C’, D
  - Paper 3: A, B, C”, D
- I like to write down that recipe in Isabelle/HOL
  - f(X): A, B, X, D
- And bake new things: f(1), f(betty), f(@), …



https://commons.wikimedia.org/wiki/File:Ouroboros-simple.svg

Where to Start

- “**Logic in Computer Science (LICS) in B2** is run by an experienced teacher team. Our goal is to give you *a firm foundation of formal logic, with a focus on logics relevant in computer science*, using a very good textbook from Cambridge University Press.”
  https://kurser.ku.dk/course/ndab16011u/

- **Interactive Proof Assistants (IPA) in B1** is “a hands-on course about using a proof assistant to construct formal models of algorithms, protocols, and programming languages and to reason about their properties. The focus is on applying logical methods to concrete problems. The course will demonstrate the challenges of formal rigour and the *benefits of machine support in modeling, proving and validating*.”
  https://kurser.ku.dk/course/ndak23006u/

Coolclusion

- Computers are cool
- Logic is cool
- Computers + logic is very cool

- Let your computer help you code
    - (Maybe we can actually write some software that works?)

https://openclipart.org/detail/317151/computer-crash