The Future of Formalized Mathematics

Florian Rabe

Friedrich-Alexander-University Erlangen-Nuremberg, Computer Science

January 2023

About Me and this Talk

Florian Rabe

http://uniformal.github.io/

My vision: UniFormal

- a universal framework for formal mathematical knowledge
- developed since 2006, $\sim 100,000$ loc, ~ 500 pages of publications

My interests

- structure and organize complex divergent theories
  logic, specification, computation, enginerring, ...
- connect disparate systems
  standard languages, interoperable tools, shared libraries, ...
- understand and bridge across communities and disciplines
  set/type theory, proof/computation, foundations/applications, ...

My methods

- breadth: mathematical approach to knowledge of all kinds
- depth: mathematical rigor from foundation to applications

The Future of Formalized Mathematics

Three Parts in this Talk

1. History, challenges, and opportunities
2. My research on understanding formal systems for math
   - languages
   - tools
   - libraries
   - communities
3. Vision and strategy for the future

Formal Mathematics: History and Future

Revolutions in Mathematical History

Formation of Empires in Antiquity

- invention of logical deduction, proof

Scientific Revolution

- mathematization of science

Foundational Crisis (around 1900)

- paradoxa, careful axiomatizations, formal systems
- Hilbert's program, search for the best formal system
- birth of computer science

Computerization ($\approx$ 1970–today)

- automated proofs, e.g., Automath 1967, Four-Color Theorem 1976
- enormous diversification of formal systems
- large-scale collaborations, applications large formal libraries

The Formalization of Mathematics

search for a formal system for mathematics

1879	Frege: formal logic
1889	Peano: rigorous axiom system
1901	Peano, Russell: paradoxa
1908, 1913	Russell, Whitehead: type theory
1908, 1922	Zermelo, Fraenkel: axiomatic set theory
1920s	Hilbert: reduction of all math to effective means
1929, 1936	Gödel, Gentzen: modern logic
1931	Gödel: incompleteness

no single best formal system — computer science arises

1930s	Turing, Church, Curry: formal computation
1958, 1969	Curry, Howard: proofs as formal objects
1967	de Bruijn: Automath — first major formalization
1973–today	Trybulec: Mizar — first formal mathematical library
1980s–today	HOL, Nuprl, Coq, . . . : increasingly strong, diverse

ongoing shift from foundations to applications

Situation Today

Flagship Successes

- 2003–2014: Hales et. al., Kepler conjecture (Flyspeck)
  $> 5,000$ processor hours needed to check proof
- 2006–2012: Gonthier et al., Feit-Thompson theorem
  170,000 lines of human-written formal logic
- 2020–2022: Scholze, Commelin et. al., Liquid Tensor challenge
  major formalization in step with frontline research

potential to massively transform future mathematics
— but how exactly?

Future Prospects

Opportunities

- Computer support for large proofs, computations, databases
  beyond one person's ability to comprehend
- Complex distributed collaborations
  polymath, finite simple groups, ...
- Faster understanding — smart editors, interactive documents, ...
- AI for mathematics??? — huge potential but very hard to predict

Challenges

- Scale: current formalizations still small — systems still experiments
- Cost-benefit ratio: most applications still too expensive
  and even harder to maintain
- Integration: about 10 major systems — mutually incompatible
  1994 QED manifesto call for integration: failed despite support
- Ecosystem: hard to build peripheral tools
  current tools designed to be standalone

"Next generation Deduction systems"

Dagstuhl Seminar November 2023, quoted from the abstract:

Techniques in deduction are often implemented only as short-lived prototypes whose transfer to stable systems is uncertain.

Over time systems may become too big, monolithic, and unwieldy.

Portfolios of techniques may be useful for experiments, but do not seem to facilitate a conceptual synthesis or a breakthrough.

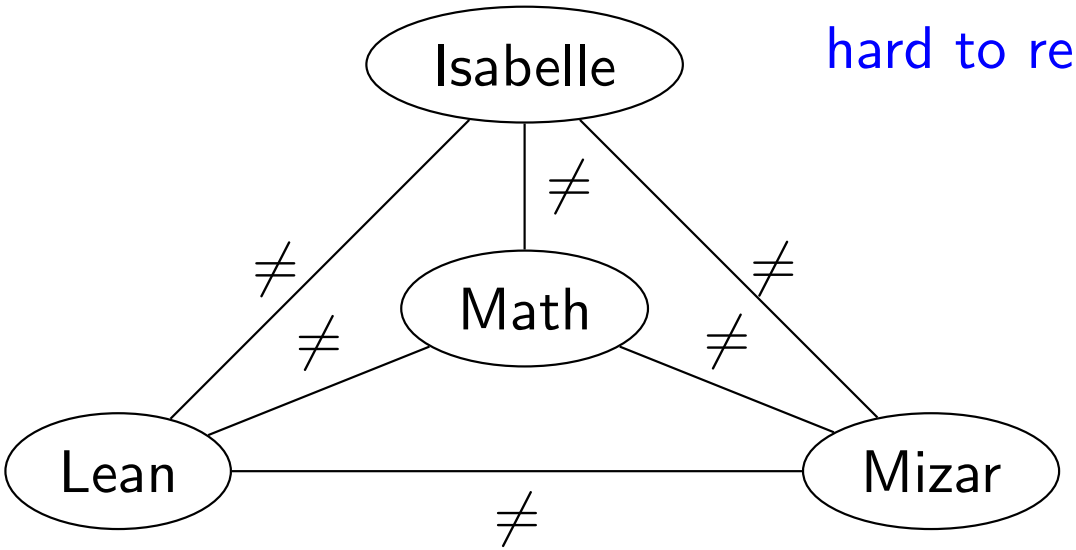
The field faces a "software crisis" that is also a "techniques crisis".

Gap between Formal Math and Standard Math

Formal math systems weak at capturing mathematical practice

- convey intuitions, motivation
- change (and abuse) of definitions and notations
- critical deep technical language features

subtyping, quotient typing, undefinedness
identification via isomorphism, flexible records/structures



hard to remedy as an afterthought

moreover: also very different from each other

incompatible languages, tools, libraries; disjoint communities

Definition of Derivative in Isabelle, Lean, Coq, Mizar

```
definition has_derivative :: "('a::real_normed_vector ⇒ 'b::real_normed_vector) ⇒
    ('a ⇒ 'b) ⇒ 'a filter ⇒ bool"  (infix "(has'_derivative)" 50)
  where "(f has_derivative f') F ⟷
    bounded_linear f' ∧
    ((λy. ((f y - f (Lim F (λx. x))) - f' (y - Lim F (λx. x))) /R norm (y - Lim F (λx. x))) ⟶ 0) F"
```

```
def has_fderiv_at_filter (f : E → F) (f' : E →L[𝕜] F) (x : E) (L : filter E) :=
(λ x', f x' - f x - f' (x' - x)) =o[L] (λ x', x' - x)
```

```
Definition D_x (D:R -> Prop) (y x:R) : Prop := D x /\ y <> x.
Definition D_in (f d:R -> R) (D:R -> Prop) (x0:R) : Prop :=
  limit1_in (fun x:R => (f x - f x0) / (x - x0)) (D_x D x0) (d x0) x0.
```

```
definition
  let f be PartFunc of REAL,REAL;
  let x0 be Real;
  pred f is_differentiable_in x0 means :: FDIFF_1:def 4
  ex N being Neighbourhood of x0 st
  ( N c= dom f & ex L being LinearFunc ex R being RestFunc st
  for x being Real st x in N holds
  (f . x) - (f . x0) = (L . (x - x0)) + (R . (x - x0)) );
end;
```

Why the gap exists and why it won't go away

Current proof systems not developed for mathematics

- proof assistants are extremely hard to build
  ≈ 100 person-years to be competitive
- languages optimized for software/hardware verification
  decidability, efficiency
- tools optimized for power users in large projects

formal math often an afterthought, overshadowed in practice
(some exceptions)

Why the gap exists and why it won't go away

Current proof systems not developed for mathematics

- proof assistants are extremely hard to build
  ≈ 100 person-years to be competitive
- languages optimized for software/hardware verification
  decidability, efficiency
- tools optimized for power users in large projects

formal math often an afterthought, overshadowed in practice
(some exceptions)

Counter-point: languages developed by/for mathematicians

- computer algebra systems mostly untyped
- recent approaches based on natural language
  Koepke's Naproche, Hales's formal abstracts language

Paper: A Survey of Languages for Formalizing Mathematics
Intelligent Computer Mathematics, 2020

Strategic Questions

Should we

- change mathematics in a way that is easier to formalize, or
- build systems that capture better how mathematics is actually done?

Should we

- formalize math multiple times in each system, or
- insist on a compatibility layer between systems?

Strategic Questions

Should we

- change mathematics in a way that is easier to formalize, or
- build systems that capture better how mathematics is actually done?

Should we

- formalize math multiple times in each system, or
- insist on a compatibility layer between systems?

Kevin Buzzard's answer

- CS must build systems that work for math
- he alone has neither resources nor interest to do it

Strategic Questions

Should we

- change mathematics in a way that is easier to formalize, or
- build systems that capture better how mathematics is actually done?

Should we

- formalize math multiple times in each system, or
- insist on a compatibility layer between systems?

Kevin Buzzard's answer

- CS must build systems that work for math
- he alone has neither resources nor interest to do it

My answer

- math as a whole does have the resources
- bring formal math home or watch it be an afterthought

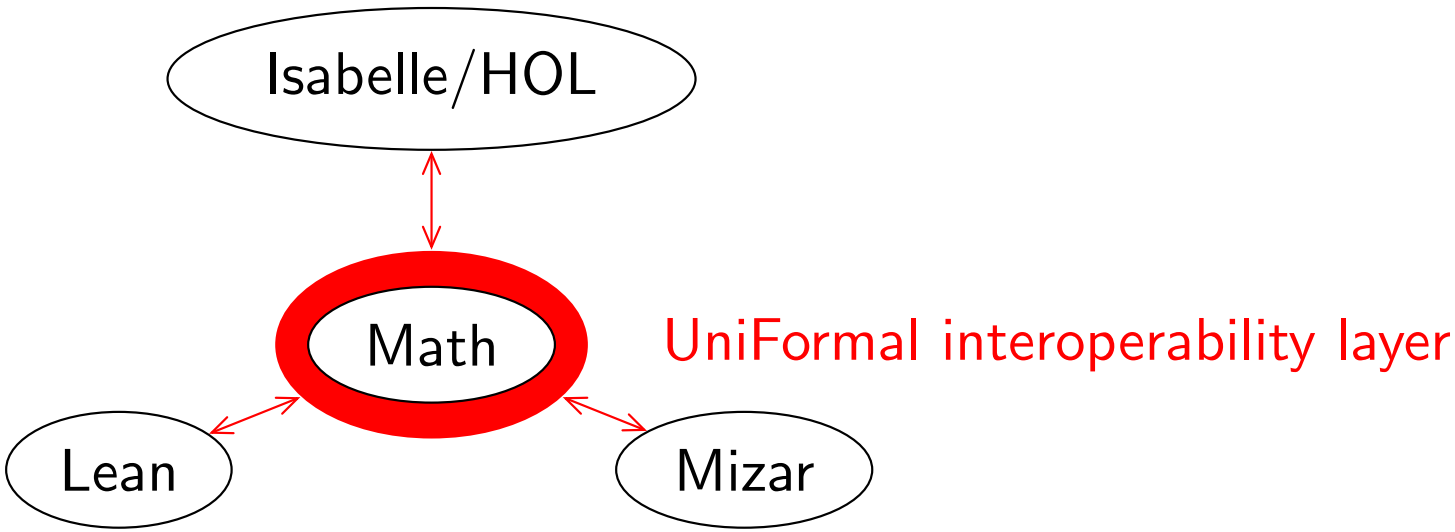
What To Do Concretely?

Evolve current optimized systems

- educate CS about math needs
- educate math about potential
- reach new scales to discover new challenges

Also develop new systems specifically for math

- optimized for adoption in math
- integrate proof, computation, natural language
- compatibility layers and open standards for system interoperability



My UniFormal Research: Understanding Languages, Tools, Libraries, Communities

The UniFormal Library

2 dedicated DFG projects on formalization

- LATIN (2009–2012): modular formalization of formal systems
  logics, type theories, set theories, ...
  by now $\sim$ 1000 modules, 20,000 lines
  3-10 contributors coordinated by me
- OAF (2014–2020): uniform representation of formal libraries
  in particular: 6 major proof assistants
  overall >200 Users >250 Repositories, >100 GB

One EU project on math (15 sites, 7.5M Euro)

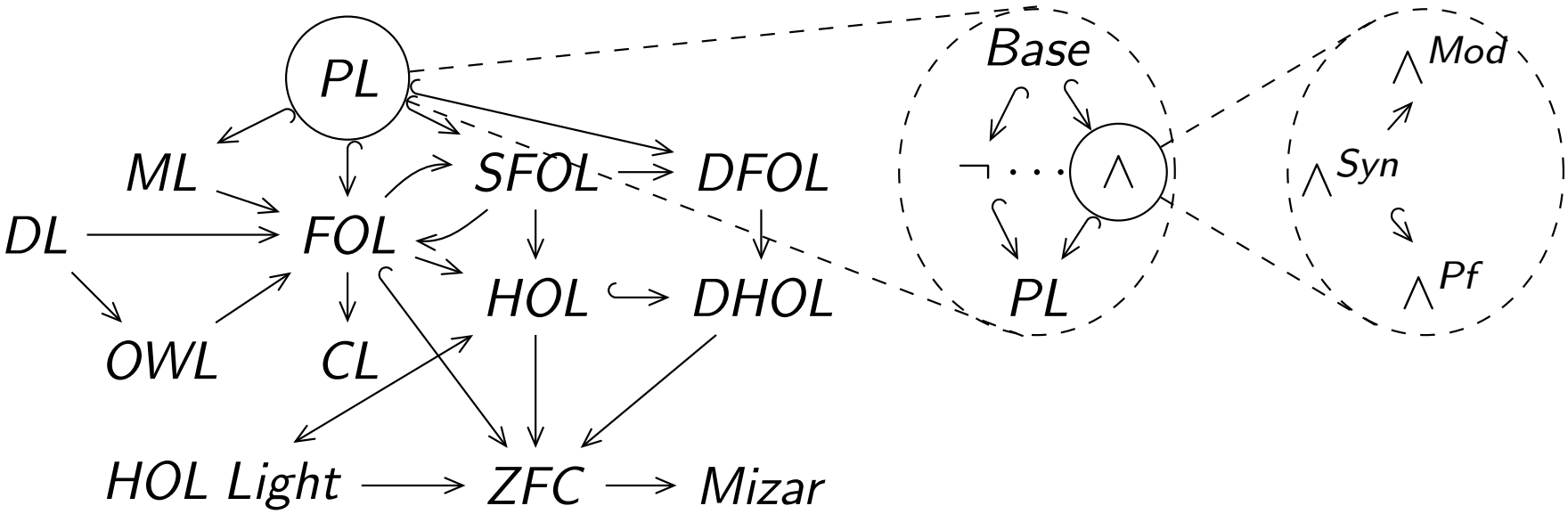
- OpenDreamKit (2015–2019): integrate computational mathematics
- deep case studies on system integration
- prototyped UniFormal system as interface layer

Central part: My UniFormal framework and system
100k loc, 5-10 contributors coordinated by me

Me pointing at first-order logic in the LATIN library

Example: Modularity and Translations in UniFormal

An example fragment of the logic library:



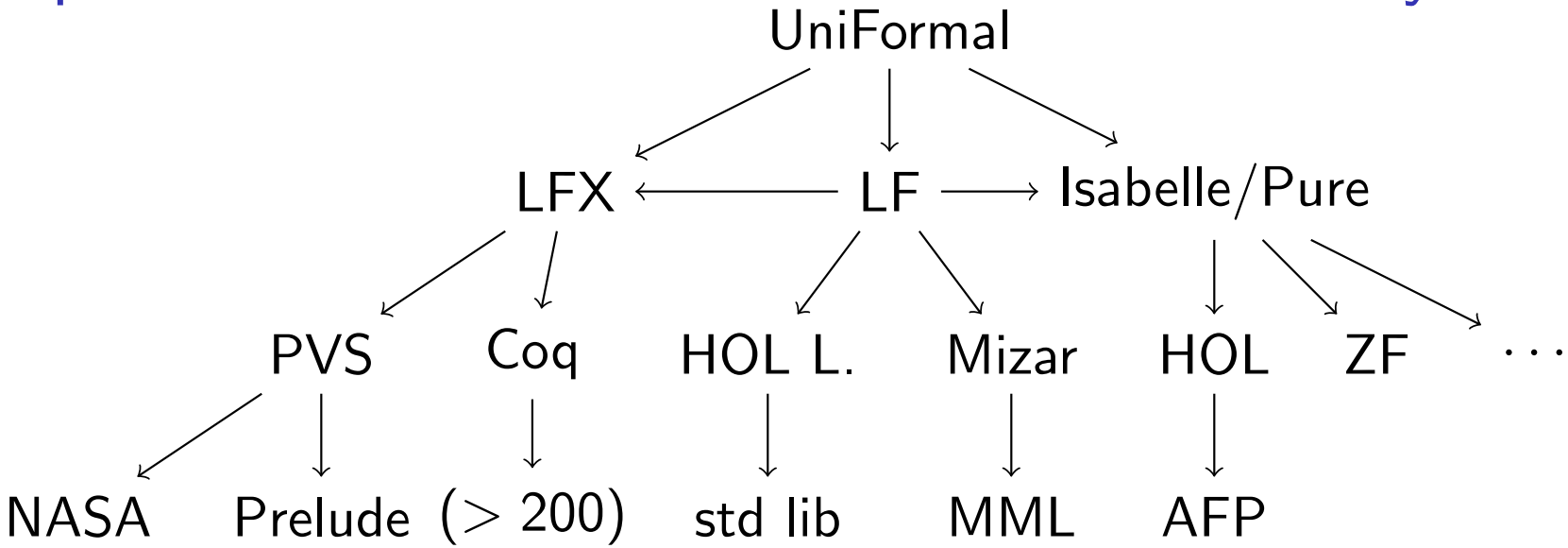
Other Highlights

- type theory, category theory, variants of all logics
- controversial features in separate modules

  choice axiom/operators, non-empty types, ...
- advanced language features in separate modules

  classes, inductive types, record types, ...

Example: Prover Libraries in the UniFormal Library



Language	Library	Modules	Declarations
LF	LATIN	529	2,824
PVS	Prelude+NASA	974	24,084
Isabelle	Distribution+AFP	12,318	2,116,638
HOL Light	Basic	189	22,830
Coq	$>$ 200 in total	$> 10k$	$> 1M$
Mizar	MML	1,194	69,710

Paper: Experiences from Exporting Major Proof Assistant Libraries
Journal of Automated Reasoning, 2021

The Lean Logic in the UniFormal Library

```
theory Lean : LF =
  // universe hierarchy (maximum operators not shown)
  level: type
  Zero: level
  Succ: level → level

  // terms, types, universes (only main constructors shown)
  expr: type
  Sort: level → expr
  App:  expr → expr → expr              # 1 @ 2
  Lam:  expr → (expr → expr) → expr     # λ 1 2
  Pi:   expr → (expr → expr) → expr     # Π 1 2

  // typing and equality of expressions (only a few rules shown)
  typing : expr → expr → type           # 1 % 2
  equal : expr → expr → type            # 1 = 2 prec 0

  typing_Sort: {l} Sort l % Sort (Succ l)
  typing_App:  {A,B,f,a} f % (Π A [x] B x) → a % A → f@a % B a
  typing_Lam:  {l,A,B,F} A % Sort l → ({x} x%A → F x % B x) → (λ A [x] F x) % Π A [x] B x
  equal_Beta:  {a,A,F} (λ A F) @ a = F a
```

Paper on the framework: How to Identify, Translate, and Combine Logics?
Journal of Logic and Computation, 2017

Example: Lean library in UniFormal

Lean $\longrightarrow$ Lean kernel $\longrightarrow$ UniFormal

- Lean surface:
  def set $(\alpha$: Type u) $:= \alpha \to$ Prop
- Lean kernel export:

```
5848 #NS 0 set
...
#DEF 5848 184818 184819 1
```

- UniFormal kernel:

```
<constant name="set">
 <definition>
  <operator module="LF" name="apply">
   <symbol module="Lean" name="Lam" />
     ...
```

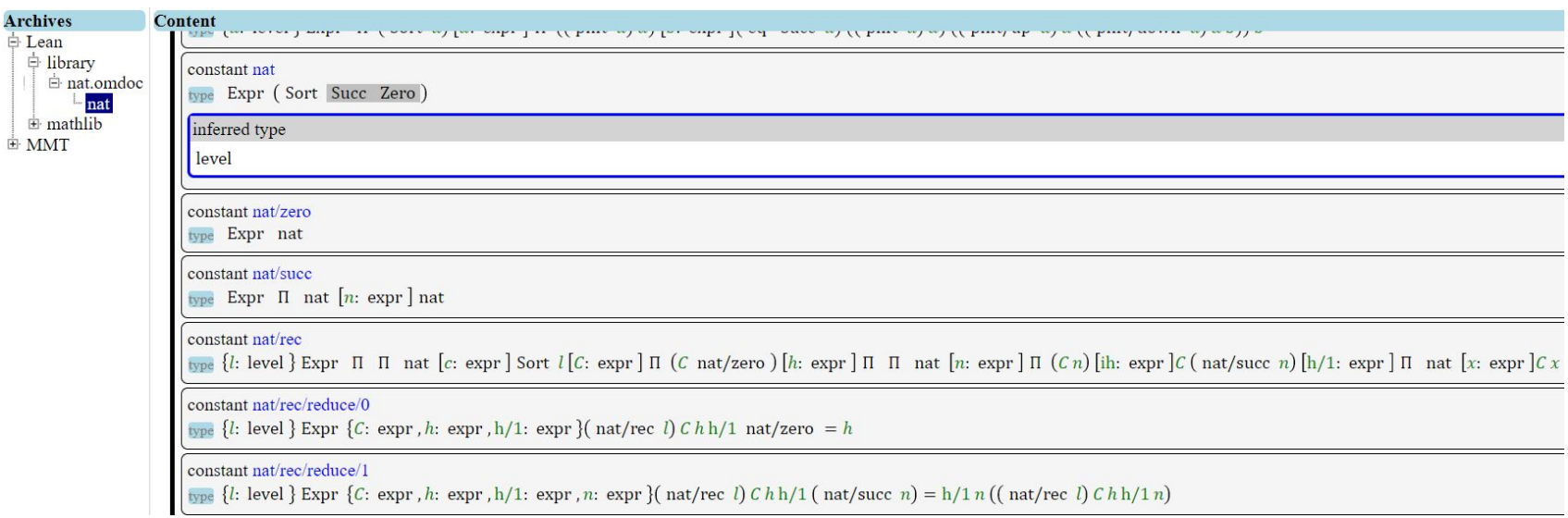
- UniFormal surface:
  set $= \lambda\ [\alpha$: Type u] $\alpha \to$ Prop

Example: Using Lean in UniFormal

Importing Lean theories in the UniFormal IDE:

```
theory Test : latin:/lean?Lean =
  include ☞latin:/lean?nat
  double = λ nat [n] nat/add @ n @ n
```

Browsing the Lean library in the UniFormal web browser:



Support Tool Example: Search Service

Enter Java regular expressions to filter based on the URI of a declaration

Namespace

Theory

Name

Enter an expression over theory http://code.google.com/p/hol-light/source/browse/trunk

$x,y,p: x MOD p = y MOD p

Use $x,y,z:query to enter unification variables.

Search

type of MOD_EQ

$$\vdash \forall m{:}\text{num} . \forall n{:}\text{num} . \forall p{:}\text{num} . \forall q{:}\text{num} . m = n + q * p \Longrightarrow m \,\text{MOD}\, p = n \,\text{MOD}\, p$$

type of MOD_MULT_ADD

$$\vdash \forall m{:}\text{num} . \forall n{:}\text{num} . \forall p{:}\text{num} . (m * n + p) \,\text{MOD}\, n = p \,\text{MOD}\, n$$

Extremely challenging

Deep understanding needed of

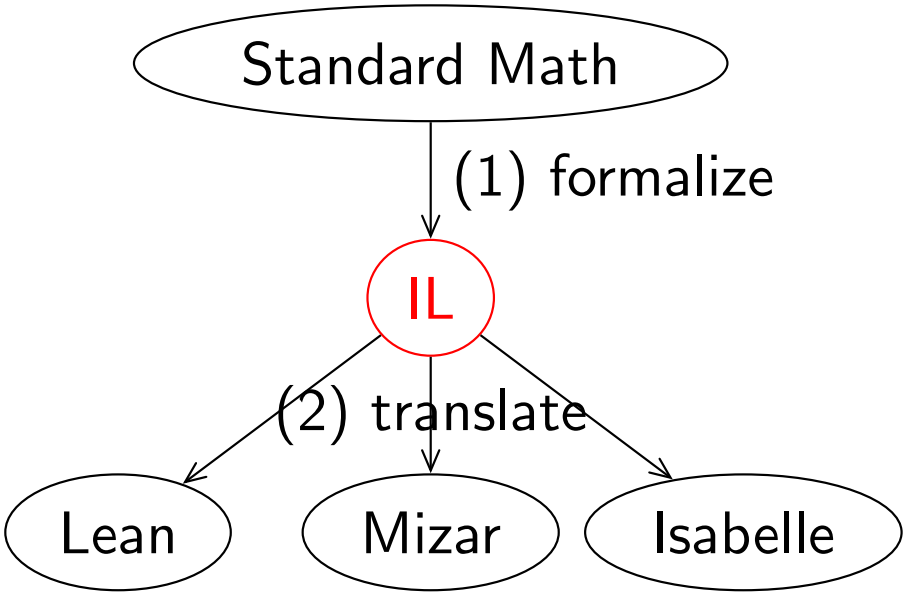
- languages
  type systems, proof systems, computation systems, module systems
- tools: provers, knowledge management
  implementation, scalability, maintenance
  and how to change them
- libraries and knowledge structuring
  translations, interfaces, modularity
- communities
  e.g., Coq: Sacerdoti Coen, Isabelle: Wenzel, Metamath: Carneiro
  Mizar: Urban, PVS: Shankar, TPTP: Sutcliffe
- future users in math
  computer algebra systems, mathematical databases
  others, e.g., Weierstrass Institute, zbMath

My UniFormal Vision and Goals for the Next 5-10 Years

Goal 1: Natural Formalization

Design an Intermediate Language (IL) to bridge the gap

- abstract syntax close to standard mathematical language
  much more expressive than current formal systems
- but designed with full formalization in mind
  more rigorous formal semantics than current languages



Advantages

- step (1) for mathematicians
  intuitive, readable, fun, scalable
- step (2) for prover-specific engineering trade-offs
- step (1) often sufficient
  e.g, for search, interaction, type-checking, machine learning

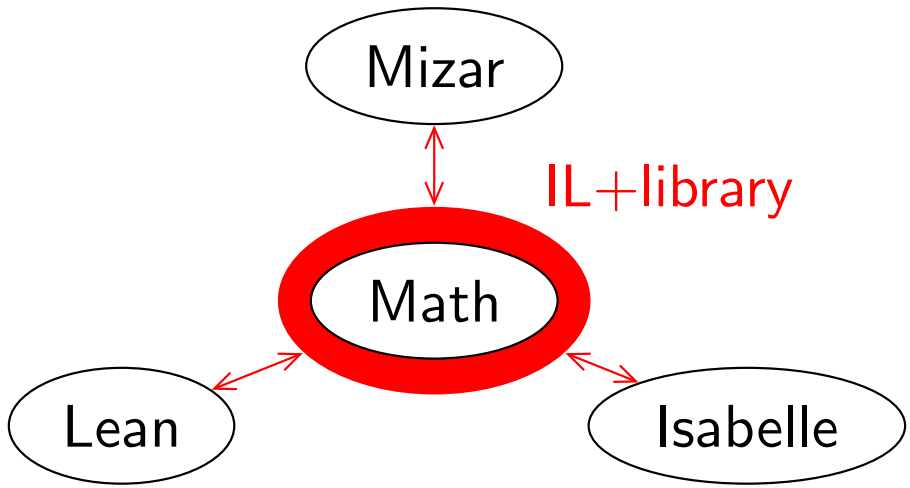
massively lowers entrance barrier for formalization

Goal 2: A Standard Math Library

IL as standard library language for Math

- no commitment to single foundation or tool — accessible to humans, robust under change of foundations
- formal syntax, semantics — accessible to provers
- modern, scalable implementations

My team's contribution



Universal standard libraries highly successful in low-math sciences
but we need IL to get the same for math

Concrete Case Studies for the Next 5 Years

Formal Lecture Notes

- Formalize Bonn's undergraduate mathematics lecture notes
  surprisingly feasible if we omit proofs!
- Coherent formalization across all courses
  handling incompatible definitions is my specialty

Realize Hales's Formal Abstracts Vision

- curated formal library of 5,000 definitions 2,000 theorems
- widely adopted service for mathematicians to search and contribute
- offers many collaboration options
  e.g., Scholze's proposed change to definition of topological space

Long-Term Goal: Hausdorff Library of Mathematics

- interface between standard and formal math
- adopted by major math stakeholders

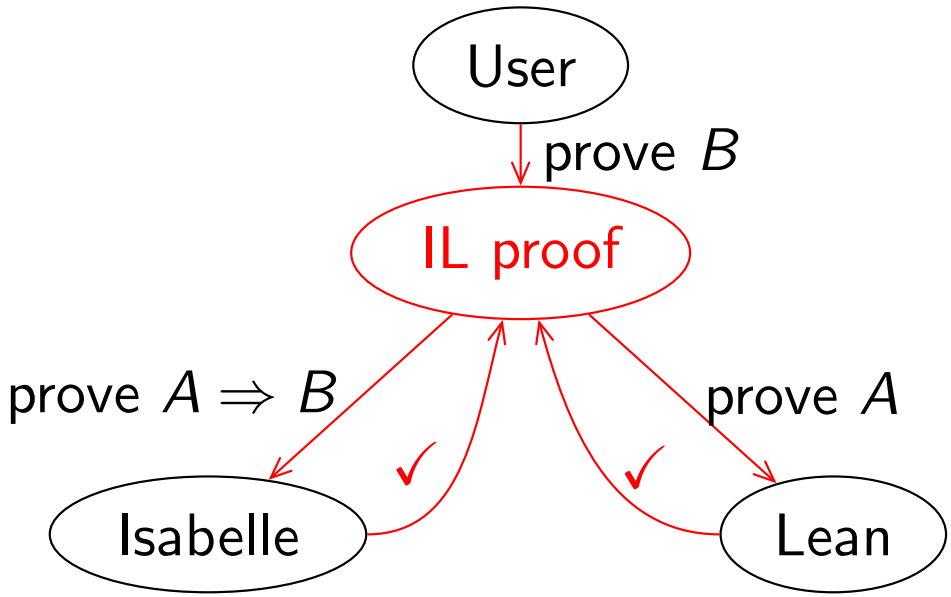
Goal 3: Cross-System Proofs via IL

IL does not replace existing proof systems

- IL optimized for creating formalizations
- provers optimized for processing formalizations

But IL enables high-level cross-systems proofs

- split big proof into subproofs that are attacked by different, optimized provers e.g., needed for Flyspeck proof
- accordingly for cross-system computation



Side Note: The Other Kind of Formalization

Computer Algebra Systems are also formal systems

- focus on algorithms instead of proofs
- developed in parallel to formal proof systems

Macsyma/Maxima since 1968, Axiom family since 1977

But they differ from provers in pragmatic aspects

- mostly untyped systems, no logic unaffected by foundational issues
- easier to apply, integrate

computation systems useful as black-box tools

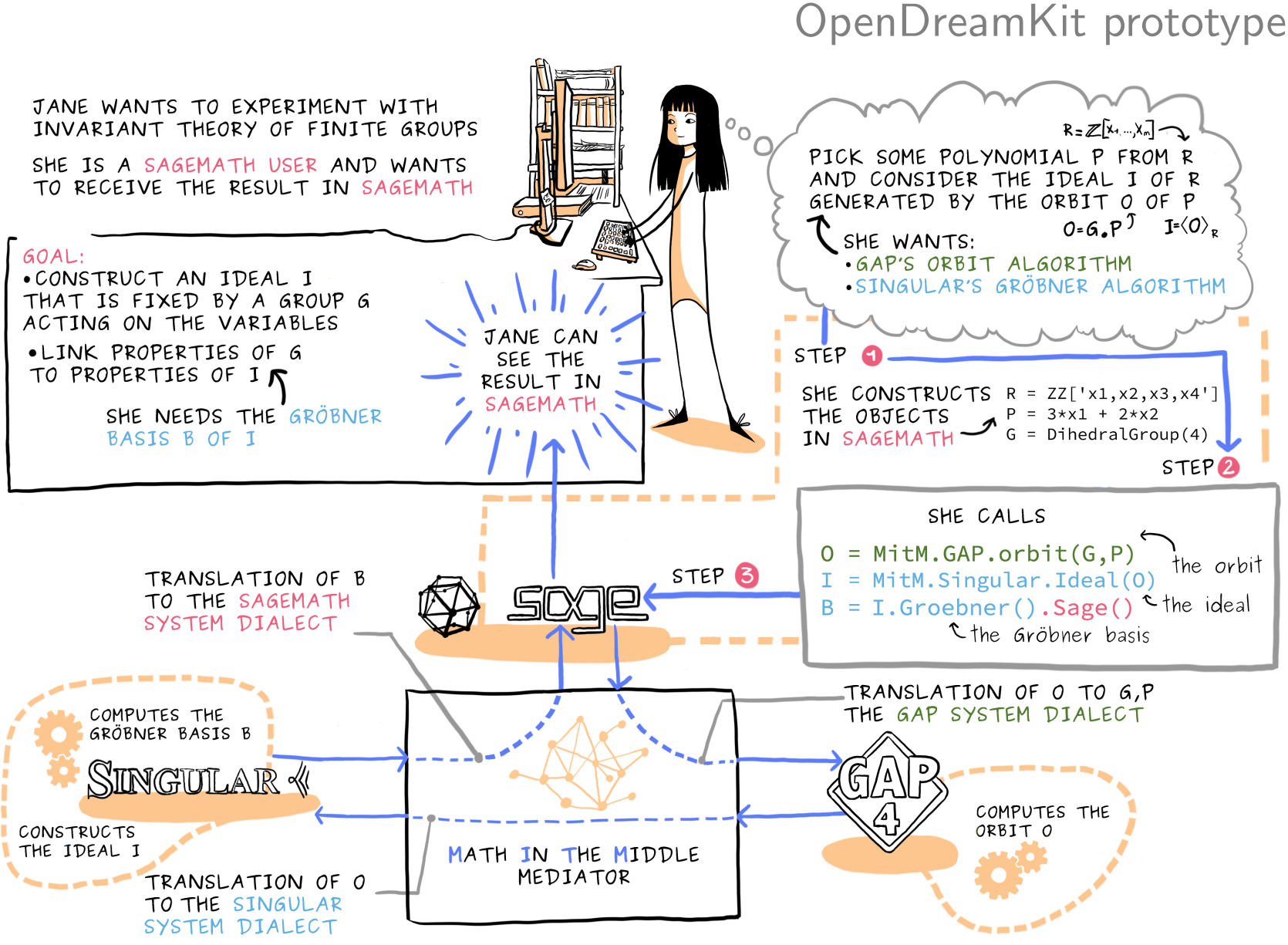
And that has made them much more successful

- major commercial systems e.g., Mathematica, Maple
- many mathematicians among users and developers
- integrative force on mathematical community

e.g., SageMath platform

despite shared history, today mostly disconnected from provers

Cross-System Computation via IL



Conclusion

- Computerization+formalization will be transformative
- My dual strategy:
  - embrace and evolve current systems
  - build new systems by and for math
- Intermediate language with math-owned standard library