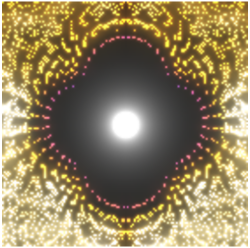
Mathematics in the Computer

Mario Carneiro

Carnegie Mellon University

April 26, 2021

Who am I?



Github: `digama0`
Zulip: Mario Carneiro

- PhD student in Logic at CMU
- Proof engineering since 2013
  - Metamath (maintainer)
  - Lean 3 (maintainer)
  - Dabbled in Isabelle, HOL Light, Coq, Mizar
  - Metamath Zero (author)
- Proved 37 of Freek's 100 theorems list in Metamath
- Lots of library code in `set.mm` and mathlib
- Say hi at `https://leanprover.zulipchat.com`

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*
- Found Metamath via a random internet search

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*
- Found Metamath via a random internet search
  - → they already formalized half of the book!

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*
- Found Metamath via a random internet search
  - → they already formalized half of the book!
  - … and there is some stuff on cofinality they don't have yet, maybe I can help

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*
- Found Metamath via a random internet search
  - $\rightarrow$ they already formalized half of the book!
  - ... and there is some stuff on cofinality they don't have yet, maybe I can help
- Got involved, did it as a hobby for a few years

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*
- Found Metamath via a random internet search
  - → they already formalized half of the book!
  - … and there is some stuff on cofinality they don't have yet, maybe I can help
- Got involved, did it as a hobby for a few years
- Got a job as an android developer, kept on the hobby

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*
- Found Metamath via a random internet search
  - → they already formalized half of the book!
  - ... and there is some stuff on cofinality they don't have yet, maybe I can help
- Got involved, did it as a hobby for a few years
- Got a job as an android developer, kept on the hobby
- Norm Megill suggested that I submit to a (Mizar) conference, it went well

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*
- Found Metamath via a random internet search
  - → they already formalized half of the book!
  - . . . and there is some stuff on cofinality they don't have yet, maybe I can help
- Got involved, did it as a hobby for a few years
- Got a job as an android developer, kept on the hobby
- Norm Megill suggested that I submit to a (Mizar) conference, it went well
- Met Leo de Moura (Lean author) at a conference, he got me in touch with Jeremy Avigad (my current advisor)

How I got involved in formalization

- Undergraduate at Ohio State University
  - Math, CS, Physics
- Reading Takeuti & Zaring, *Axiomatic Set Theory*
- Found Metamath via a random internet search
  - → they already formalized half of the book!
  - . . . and there is some stuff on cofinality they don't have yet, maybe I can help
- Got involved, did it as a hobby for a few years
- Got a job as an android developer, kept on the hobby
- Norm Megill suggested that I submit to a (Mizar) conference, it went well
- Met Leo de Moura (Lean author) at a conference, he got me in touch with Jeremy Avigad (my current advisor)
- Now I'm a PhD at CMU philosophy!

Why formalize mathematics?

- Humans make errors. Computers make fewer errors.

Why formalize mathematics?

- Humans make errors. Computers make fewer errors.
- Formalizing a proof is a really good way to learn how the proof works!
  - It is a real eye opener when you realize how much you skipped over when reading a proof “normally” vs when you have to convince a computer

Why formalize mathematics?

- Humans make errors. Computers make fewer errors.
- Formalizing a proof is a really good way to learn how the proof works!
  - It is a real eye opener when you realize how much you skipped over when reading a proof "normally" vs when you have to convince a computer
- Formalizing a lot of proofs is a good way to learn how formalization works
  - You learn tricks of the system, the way things are encoded in the library, and general techniques for structuring mathematics so that things go as smoothly as you naively thought it would at the start

Why formalize mathematics?

- Humans make errors. Computers make fewer errors.
- Formalizing a proof is a really good way to learn how the proof works!
  - It is a real eye opener when you realize how much you skipped over when reading a proof “normally” vs when you have to convince a computer
- Formalizing a lot of proofs is a good way to learn how formalization works
  - You learn tricks of the system, the way things are encoded in the library, and general techniques for structuring mathematics so that things go as smoothly as you naively thought it would at the start
- It makes you a better mathematician
  - You will lie less when doing paper mathematics
  - Cf. Terry Tao’s “post-rigorous” mathematician

Why formalize mathematics?

- Humans make errors. Computers make fewer errors.
- Formalizing a proof is a really good way to learn how the proof works!
  - It is a real eye opener when you realize how much you skipped over when reading a proof "normally" vs when you have to convince a computer
- Formalizing a lot of proofs is a good way to learn how formalization works
  - You learn tricks of the system, the way things are encoded in the library, and general techniques for structuring mathematics so that things go as smoothly as you naively thought it would at the start
- It makes you a better mathematician
  - You will lie less when doing paper mathematics
  - Cf. Terry Tao's "post-rigorous" mathematician
- Also it's the world's best puzzle game

What is it like to formalize mathematics?

Sidebar: Metamath architecture

Sidebar: Metamath architecture

- Metamath is a specification for `.mm` files that contain definitions, theorems, and proofs

Sidebar: Metamath architecture

- Metamath is a specification for `.mm` files that contain definitions, theorems, and proofs
- Metamath has many (many!) verifiers, written in dozens of languages

Sidebar: Metamath architecture

- Metamath is a specification for `.mm` files that contain definitions, theorems, and proofs
- Metamath has many (many!) verifiers, written in dozens of languages
- Proofs in an `.mm` file are expressed in a compressed format that is not intended to be written by humans

Sidebar: Metamath architecture

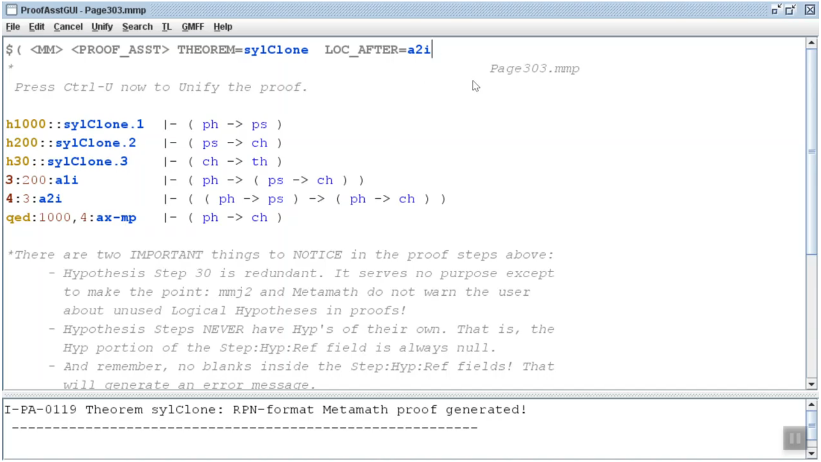
- Metamath is a specification for `.mm` files that contain definitions, theorems, and proofs
- Metamath has many (many!) verifiers, written in dozens of languages
- Proofs in an `.mm` file are expressed in a compressed format that is not intended to be written by humans
- To produce proofs, you need a *proof assistant*, and there are several of these (not as many as the verifiers).

Sidebar: Metamath architecture

- Metamath is a specification for `.mm` files that contain definitions, theorems, and proofs
- Metamath has many (many!) verifiers, written in dozens of languages
- Proofs in an `.mm` file are expressed in a compressed format that is not intended to be written by humans
- To produce proofs, you need a *proof assistant*, and there are several of these (not as many as the verifiers).
- I used `mmj2`[1] for this for many years (and now I'm the maintainer)

[1] `https://github.com/digama0/mmj2/`

Sidebar: Metamath architecture



Search “mmj2 tutorial” to see it in action

What is it like to formalize mathematics?

In Metamath:

What is it like to formalize mathematics?

In Metamath:

- You state a theorem in the formal language

What is it like to formalize mathematics?

In Metamath:

- You state a theorem in the formal language
- You apply theorems from the library, and the computer prompts you for the subgoals

What is it like to formalize mathematics?

In Metamath:

- You state a theorem in the formal language
- You apply theorems from the library, and the computer prompts you for the subgoals
- In some cases the computer can automatically prove some of the subgoals

What is it like to formalize mathematics?

In Metamath:

- You state a theorem in the formal language
- You apply theorems from the library, and the computer prompts you for the subgoals
- In some cases the computer can automatically prove some of the subgoals
- When you finish the proof, it gives you a big blob to stick in the `.mm` file and celebrate

What is it like to formalize mathematics?

In Metamath:

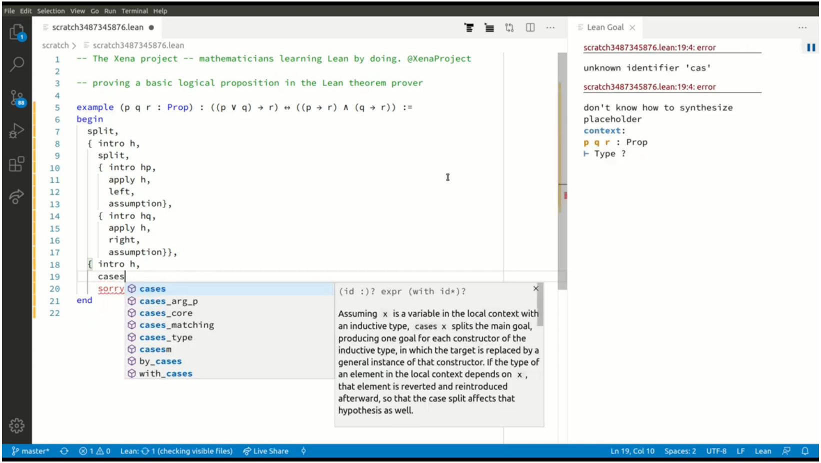
- You state a theorem in the formal language
- You apply theorems from the library, and the computer prompts you for the subgoals
- In some cases the computer can automatically prove some of the subgoals
- When you finish the proof, it gives you a big blob to stick in the `.mm` file and celebrate
- You PR your changes to the `set.mm` repository

What is it like to formalize mathematics?

In Lean:

- You state a theorem in the formal language
- You apply tactics, which transform the goals into subgoals in some defined way
- Often the tactic is simply `apply` *thm* where *thm* is a lemma from the library
- When you finish the proof, you leave the tactic script in the file
- You PR your changes to the `mathlib` repository

What is it like to formalize mathematics?



From Kevin Buzzard's "10 minute Lean tutorial: proving logical propositions"

How do Metamath and Lean differ?

- Lean has a lot more institutional support (MSR, CMU)
- Lean has a great user experience and is generally better suited to mathematician users
- Metamath is deliberately as simple as it can be, and writing a verifier for Metamath is a weekend project
  - "Batteries not included": tooling is decentralized and DIY
- There is essentially only one Lean verifier
  - Parsing lean files is nearly impossible for anything other than lean
  - Lean has an export format that has a few alternate verifiers
- Compiling `set.mm` is about 1000× faster than compiling mathlib

How do we get closer to air tight mathematics?

- “Computers make fewer errors.” Is this true?

How do we get closer to air tight mathematics?

- “Computers make fewer errors.” Is this true?
- Computers executing correct computer programs with modern error correcting hardware can perform trillions of computations without a single error

How do we get closer to air tight mathematics?

- "Computers make fewer errors." Is this true?
- Computers executing **correct** computer programs with modern error correcting hardware can perform trillions of computations without a single error

How do we get closer to air tight mathematics?

- "Computers make fewer errors." Is this true?
- Computers executing **correct** computer programs with modern error correcting hardware can perform trillions of computations without a single error
- A buggy program can be wrong 100% of the time
  - The presence of a bug is not probabilistic except in the Bayesian sense
  - The number of bugs in a program linearly correlates with the length of the program

How do we get closer to air tight mathematics?

- "Computers make fewer errors." Is this true?
- Computers executing **correct** computer programs with modern error correcting hardware can perform trillions of computations without a single error
- A buggy program can be wrong 100% of the time
  - The presence of a bug is not probabilistic except in the Bayesian sense
  - The number of bugs in a program linearly correlates with the length of the program
- The possible gains are great, but correctness makes all the difference.

How do we get closer to air tight mathematics?

- "Computers make fewer errors." Is this true?
- Computers executing **correct** computer programs with modern error correcting hardware can perform trillions of computations without a single error
- A buggy program can be wrong 100% of the time
  - The presence of a bug is not probabilistic except in the Bayesian sense
  - The number of bugs in a program linearly correlates with the length of the program
- The possible gains are great, but correctness makes all the difference.

1. The verifier should be as simple as it can be

How do we get closer to air tight mathematics?

- "Computers make fewer errors." Is this true?
- Computers executing **correct** computer programs with modern error correcting hardware can perform trillions of computations without a single error
- A buggy program can be wrong 100% of the time
  - The presence of a bug is not probabilistic except in the Bayesian sense
  - The number of bugs in a program linearly correlates with the length of the program
- The possible gains are great, but correctness makes all the difference.

1. The verifier should be as simple as it can be
2. This isn't good enough, because the chance of a bug in the program will still be larger than the computer's own error rate

How do we get closer to air tight mathematics?

- How do we ensure there are no bugs in a program?

How do we get closer to air tight mathematics?

- How do we ensure there are no bugs in a program?
- Program correctness is a mathematical property, because both the program and the target are specified abstractly (no physics needed)

How do we get closer to air tight mathematics?

- How do we ensure there are no bugs in a program?
- Program correctness is a mathematical property, because both the program and the target are specified abstractly (no physics needed)
- But program correctness proofs are long and tricky, and proving them on paper is liable to human error…

How do we get closer to air tight mathematics?

- How do we ensure there are no bugs in a program?
- Program correctness is a mathematical property, because both the program and the target are specified abstractly (no physics needed)
- But program correctness proofs are long and tricky, and proving them on paper is liable to human error...
- Hammer, meet nail

Self verifying theorem provers

Self verifying theorem provers

- Wait, didn't Gödel prove this is impossible?

Self verifying theorem provers

- Wait, didn't Gödel prove this is impossible?
  - Well, yes and no
- The important observation is that we want to prove "implementation correctness", not consistency

Self verifying theorem provers

- Wait, didn't Gödel prove this is impossible?
  - Well, yes and no
- The important observation is that we want to prove "implementation correctness", not consistency
- We will not be able to completely eliminate the circularity, though, so we still need to rely on human verification to some extent

The metamathematics of theorem provers

- Let $\mathcal{M} \subseteq \{0,1\}^* \times \{0,1\}^*$ be a machine semantics, where $\mathcal{M}(P,x)$ means that program P on input x terminates and indicates success.
  - For example, $\mathcal{M}(P,x)$ if P encodes a Turing machine that when run on input x stops in finitely many steps with a 1 on the tape
- Let $\mathcal{L} \subseteq \{0,1\}^*$ be a language of assertions
  - For example, $\varphi \in \mathcal{L}$ if φ encodes a statement in FOL
  - We generally want $\varphi \in \mathcal{L}$ to be decidable
- Let T be a theory of interest
  - For example, $T = \mathrm{ZFC}$

Implementation correctness for a theorem prover

Program V is a theorem prover (for T in $\mathcal{M}$ and $\mathcal{L}$) if for all $\varphi \in \mathcal{L}$, if there exists p such that $\mathcal{M}(V,(\varphi,p))$, then $T \vdash \varphi$.

Bootstrapping trust

1. Suppose V is a correct verifier, i.e. if $T \vdash_V A$ then $T \vdash A$ for all T, A.
2. Suppose we prove, in V + PA, the correctness theorem for W, that is, PA $\vdash_V \forall T, A \colon (T \vdash_W A \to T \vdash A)$
3. Then PA $\vdash \forall T, A \colon (T \vdash_W A \to T \vdash A)$
4. If PA is sound, then $\forall T, A \colon (T \vdash_W A \to T \vdash A)$, that is, $T \vdash_W A$ implies $T \vdash A$, so W is a correct verifier

- Bootstrapping: set $W := V$ in the above
- Note that this does not run afoul of Gödel incompleteness
- Circular proof! Need a backup to ground the argument
  - $\Rightarrow$ small verifier
  - independent bootstraps

From theory to practice

- Can we use Lean to prove Lean correct?

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated
- Can we use Metamath to prove Metamath correct?
  - Maybe, although it's kind of painful
  - Also the model theory of general Metamath databases is kind of weird

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated
- Can we use Metamath to prove Metamath correct?
  - Maybe, although it's kind of painful
  - Also the model theory of general Metamath databases is kind of weird
- Solve for V in "We use V to prove V correct"

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated
- Can we use Metamath to prove Metamath correct?
  - Maybe, although it's kind of painful
  - Also the model theory of general Metamath databases is kind of weird
- Solve for V in "We use V to prove V correct"
  - It should be as simple as possible, like Metamath

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated
- Can we use Metamath to prove Metamath correct?
  - Maybe, although it's kind of painful
  - Also the model theory of general Metamath databases is kind of weird
- Solve for V in "We use V to prove V correct"
  - It should be as simple as possible, like Metamath
  - It should have a decent finitistic metatheory

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated
- Can we use Metamath to prove Metamath correct?
  - Maybe, although it's kind of painful
  - Also the model theory of general Metamath databases is kind of weird
- Solve for V in "We use V to prove V correct"
  - It should be as simple as possible, like Metamath
  - It should have a decent finitistic metatheory
  - It should have good support for program correctness proofs

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated
- Can we use Metamath to prove Metamath correct?
  - Maybe, although it's kind of painful
  - Also the model theory of general Metamath databases is kind of weird
- Solve for V in "We use V to prove V correct"
  - It should be as simple as possible, like Metamath
  - It should have a decent finitistic metatheory
  - It should have good support for program correctness proofs
  - It should be practical on potentially very large proofs

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated
- Can we use Metamath to prove Metamath correct?
  - Maybe, although it's kind of painful
  - Also the model theory of general Metamath databases is kind of weird
- Solve for V in "We use V to prove V correct"
  - It should be as simple as possible, like Metamath
  - It should have a decent finitistic metatheory
  - It should have good support for program correctness proofs
  - It should be practical on potentially very large proofs
  - It should have basic interactive theorem prover niceties (not in the verifier but in the proof assistant)

From theory to practice

- Can we use Lean to prove Lean correct?
  - Nope, way too complicated
- Can we use Metamath to prove Metamath correct?
  - Maybe, although it's kind of painful
  - Also the model theory of general Metamath databases is kind of weird
- Solve for V in "We use V to prove V correct"
  - It should be as simple as possible, like Metamath
  - It should have a decent finitistic metatheory
  - It should have good support for program correctness proofs
  - It should be practical on potentially very large proofs
  - It should have basic interactive theorem prover niceties (not in the verifier but in the proof assistant)
- V = Metamath Zero

Metamath Zero Architecture

- MM0: The verifier, the source of trust
  - Standalone executable
  - The “small trusted kernel”

Metamath Zero Architecture

- MM0: The verifier, the source of trust
  - Standalone executable
  - The "small trusted kernel"
- MM1: The proof assistant – produces MM0 proofs
  - Runs tactics and metaprograms and exports MM0 proofs

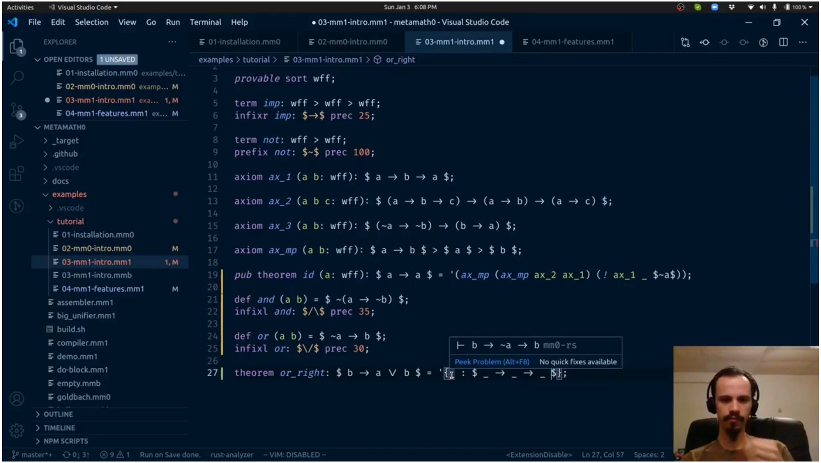
Metamath Zero Architecture

- MM0: The verifier, the source of trust
  - Standalone executable
  - The "small trusted kernel"
- MM1: The proof assistant – produces MM0 proofs
  - Runs tactics and metaprograms and exports MM0 proofs
- MMC: A verified compiler (in progress)
  - A programming language embedded as an MM1 tactic for producing x86 programs with a proof of correctness

Metamath Zero Architecture

- MM0: The verifier, the source of trust
  - Standalone executable
  - The "small trusted kernel"
- MM1: The proof assistant – produces MM0 proofs
  - Runs tactics and metaprograms and exports MM0 proofs
- MMC: A verified compiler (in progress)
  - A programming language embedded as an MM1 tactic for producing x86 programs with a proof of correctness
- The plan: write a verifier for MM0 in the MMC language

Metamath Zero

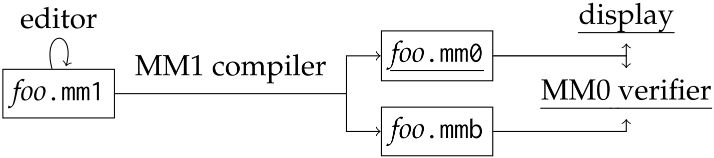


From “Metamath Zero (MM0/MM1) tutorial”

Metamath Zero Architecture

MM0 input is split into two parts:

- **`.mm0` specification file**
  - trusted, human readable
  - contains the statement of axioms and assertions
- **`.mmb` proof file**
  - untrusted, binary
  - fully elaborated, designed for efficient checking by the verifier



Underlined components are trusted.

goldbach.mm0

```
delimiter $ ( [ ~ , $ -- these tokens don't need a space after them
          $ ) ] $;    -- these tokens don't need a space before them

-- Propositional logic
strict provable sort wff;

term im: wff > wff > wff;
infixr im: $->$ prec 25;

term not: wff > wff;
prefix not: $~$ prec 41;

axiom ax_1 (ph ps: wff): $ ph -> ps -> ph $;
axiom ax_2 (ph ps ch: wff): $ (ph -> ps -> ch) -> (ph -> ps) -> ph -> ch $;
axiom ax_3 (ph ps: wff): $ (~ph -> ~ps) -> ps -> ph $;
axiom ax_mp (ph ps: wff): $ ph -> ps $ > $ ph $ > $ ps $;

def an (a b: wff): wff = $ ~(a -> ~b) $;
infixl an: $/\$ prec 34;

def iff (a b: wff): wff = $ (a -> b) /\ (b -> a) $;
infixl iff: $<->$ prec 20;

def or (a b: wff): wff = $ ~a -> b $;
infixl or: $\/$ prec 30;
```

goldbach.mm0

```
-- Predicate logic (on nat)
sort nat;
term al {x: nat} (ph: wff x): wff;
prefix al: $A.$ prec 41;

def ex {x: nat} (ph: wff x): wff = $ ~(A. x ~ph) $;
prefix ex: $E.$ prec 41;

term eq (a b: nat): wff;
infixl eq: $=$ prec 50;

axiom ax_gen {x: nat} (ph: wff x): $ ph $ > $ A. x ph $;
axiom ax_4 {x: nat} (ph ps: wff x): $ A. x (ph -> ps) -> A. x ph -> A. x ps $;
axiom ax_5 {x: nat} (ph: wff): $ ph -> A. x ph $;
axiom ax_6 (a: nat) {x: nat}: $ E. x x = a $;
axiom ax_7 (a b c: nat): $ a = b -> a = c -> b = c $;
axiom ax_10 {x: nat} (ph: wff x): $ ~ A. x ph -> A. x ~ A. x ph $;
axiom ax_11 {x y: nat} (ph: wff x y): $ A. x A. y ph -> A. y A. x ph $;
axiom ax_12 {x: nat} (a: nat) (ph: wff x): $ x = a -> ph -> A. x (x = a -> ph) $;

def sb (a: nat) {x .y: nat} (ph: wff x): wff =
  $ A. y (y = a -> A. x (x = y -> ph)) $;
notation sb (a: nat) {x: nat} (ph: wff x): wff =
($[$:41) a ($/$:0) x ($]$:0) ph;
```

goldbach.mm0

```
-- Peano's axioms
term d0: nat; prefix d0: $0$ prec max;
term suc (n: nat): nat;
axiom peano1 (a: nat): $ ~ suc a = 0 $;
axiom peano2 (a b: nat): $ suc a = suc b <-> a = b $;
axiom peano5 {x: nat} (P: wff x):
  $ [0 / x] P -> A. x (P -> [suc x / x] P) -> A. x P $;

term add: nat > nat > nat; infixl add: $+$ prec 64;
term mul: nat > nat > nat; infixl mul: $*$ prec 70;

axiom addeq (a b c d: nat): $ a = b -> c = d -> a + c = b + d $;
axiom muleq (a b c d: nat): $ a = b -> c = d -> a * c = b * d $;
axiom add0 (a: nat): $ a + 0 = a $;
axiom addS (a b: nat): $ a + suc b = suc (a + b) $;
axiom mul0 (a: nat): $ a * 0 = 0 $;
axiom mulS (a b: nat): $ a * suc b = a * b + a $;
```

goldbach.mm0

```
-- Definitions and theorems
def d1: nat = $suc 0$; prefix d1: $1$ prec max;
def d2: nat = $suc 1$; prefix d2: $2$ prec max;

def le (a b .x: nat): wff = $ E. x a + x = b $;   infixl le: $<=$ prec 50;
def lt (a b: nat): wff = $ suc a <= b $;          infixl lt: $<$ prec 50;
def dvd (a b .c: nat): wff = $ E. c c * a = b $;  infixl dvd: $|$ prec 50;
def prime (p .x: nat): wff = $ 1 < p /\ A. x (x | p -> x = 1 \/ x = p) $;

theorem goldbach (n: nat) {p q: nat}:
  $ 2 < n /\ 2 | n -> E. p E. q (prime p /\ prime q /\ n = p + q) $;
```

- This is a complete `.mm0` file that asserts that Goldbach's conjecture (GC) is derivable from the axioms of Peano Arithmetic.
- The correctness theorem for MM0 implies that if the MM0 verifier accepts any `.mmb` proof of this `.mm0` file, then GC is in fact provable in PA.

The MM0 toolchain

- MM0 verifier is currently implemented in C (2000 LOC)
  - The MM1 proof assistant is in Rust
  - MMC implementation in progress (in Rust)
- Scales well to large developments
  - Can verify `set.mm` library of ~30000 proofs in 200 ms, faster than metamath itself (`metamath.exe` – 8 s, `smm` – 900 ms)
    - That's 10,000× faster than lean, although this is an incredibly unfair comparison for a number of reasons
  - The library of supporting material from PA for this project checks in 2 ms
- Proof terms are stored in the binary format as fully elaborated and fully deduplicated terms
  - Essentially linear time verification
- Proof size is comparable to compiled proof formats in other languages (`.mm`, `.olean`)

Translation

- Even after the MM0 self-verification theorem is complete, it is only useful if people use it... or is it?

Translation

- Even after the MM0 self-verification theorem is complete, it is only useful if people use it... or is it?
- A theorem prover that can bootstrap itself can also be the foundation for others

Translation

- Even after the MM0 self-verification theorem is complete, it is only useful if people use it... or is it?
- A theorem prover that can bootstrap itself can also be the foundation for others
- Proving Lean correct in MM0 is no easier than Lean in Lean, but we have another alternative: proof translation

Translation

- Even after the MM0 self-verification theorem is complete, it is only useful if people use it... or is it?
- A theorem prover that can bootstrap itself can also be the foundation for others
- Proving Lean correct in MM0 is no easier than Lean in Lean, but we have another alternative: proof translation
- If all theorems in Lean can be translated in a proof preserving way to MM0 theorems, then Lean can be used as an MM0 IDE
  - MM0 gains all the benefits of Lean's user interface
  - We don't need to convince anyone to switch
  - Work on Lean → MM0 translation can proceed independently of new theorems to mathlib

Translation

- The MM0 formal system lies at the intersection of Metamath and second order logic, and so it has easy translation paths to each
  - MM $\to$ MM0
  - MM0 $\to$ OpenTheory
  - MM0 $\to$ Lean
- The MM $\to$ MM0 translator has been used to losslessly translate the entire Metamath ZFC library into MM0

```
theorem dirith' {n : ℕ} {a : ℤ} (n0 : n ≠ 0)
  (g1 : int.gcd a n = 1) :
  ¬ set.finite {x | nat.prime x ∧ ↑n | ↑x - a}
```

The Rosy Future

- Improved user experience in high level proof assistants like Lean means that more mathematicians can get involved
- The bottleneck on interesting formalized mathematics today is lack of mathematicians who know and care to formalize interesting mathematics
- There are no serious technical limitations on proving any branch of mathematics that I am aware of

The Rosy Future

- Improved user experience in high level proof assistants like Lean means that more mathematicians can get involved
- The bottleneck on interesting formalized mathematics today is lack of mathematicians who know and care to formalize interesting mathematics
- There are no serious technical limitations on proving any branch of mathematics that I am aware of
- By translation to MM0, the entire library of mathematics in every major theorem prover can be checked by a proven correct verifier

The Rosy Future

- Improved user experience in high level proof assistants like Lean means that more mathematicians can get involved
- The bottleneck on interesting formalized mathematics today is lack of mathematicians who know and care to formalize interesting mathematics
- There are no serious technical limitations on proving any branch of mathematics that I am aware of
- By translation to MM0, the entire library of mathematics in every major theorem prover can be checked by a proven correct verifier
  - Also, by translation *from* MM0, libraries can communicate formal content and build on material proved in other theorem provers

The Rosy Future

- Improved user experience in high level proof assistants like Lean means that more mathematicians can get involved
- The bottleneck on interesting formalized mathematics today is lack of mathematicians who know and care to formalize interesting mathematics
- There are no serious technical limitations on proving any branch of mathematics that I am aware of
- By translation to MM0, the entire library of mathematics in every major theorem prover can be checked by a proven correct verifier
  - Also, by translation *from* MM0, libraries can communicate formal content and build on material proved in other theorem provers
- The MM0 bootstrap itself can be improved independently, for example by verifying the electronic model of the hardware.

The Rosy Future

All together, these points mean that anyone will have easy access to mechanical means to verify proofs to the quality of the physical hardware (up to the probability of e.g. cosmic ray interference and our understanding of physical laws).

The Rosy Future

All together, these points mean that anyone will have easy access to mechanical means to verify proofs to the quality of the physical hardware (up to the probability of e.g. cosmic ray interference and our understanding of physical laws).

Then:

- Make it easy for programmers to write bug-free software through better verified-programming language design
- Make it easy for mathematicians to write formal mathematics, so that all the new mathematics that comes out is also verified

The Rosy Future

All together, these points mean that anyone will have easy access to mechanical means to verify proofs to the quality of the physical hardware (up to the probability of e.g. cosmic ray interference and our understanding of physical laws).

Then:

- Make it easy for programmers to write bug-free software through better verified-programming language design
- Make it easy for mathematicians to write formal mathematics, so that all the new mathematics that comes out is also verified

Correctness: solved!

Conclusion

- Verification systems are the means by which we check mathematics in the computer
- Metamath and Lean are near polar opposite designs for a theorem prover
  - (I recommend Lean for mathematicians)
- To improve on the correctness frontier, we need to verify the verifier
- The goal of the MM0 project is to prove a correctness theorem for the verifier of the form "if execution of verifier V according to the semantics of x86 machine code reports that theorem φ follows from axioms T, then $T \vdash \varphi$"
- Large parts of the project are already complete, and you can play with the MM1 proof assistant today

Resources

- Metamath: `http://us.metamath.org/`
- Lean/mathlib: `http://leanprover-community.github.io/`
- Metamath Zero: `https://github.com/digama0/mm0`
- Lean Zulip: `https://leanprover.zulipchat.com/`
  - Ask me anything on Zulip, I'm there a lot

Thanks!