Porting Mathlib

Mario Carneiro

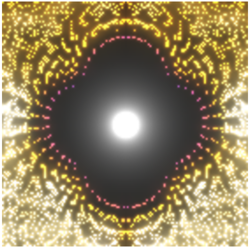
Carnegie Mellon University

July 30, 2021

Acknowledgements

- This is reporting on an unfinished project
- Joint work with Daniel Selsam, Aurélien Saue, Gabriel Ebner, Wojciech Nawrocki, Kevin Buzzard, David Renshaw, Jeremy Avigad, Deniz Aydin, Shing Tak Lam
- and many more are likely to enter the project later
- Funded by Microsoft Research

Who am I?

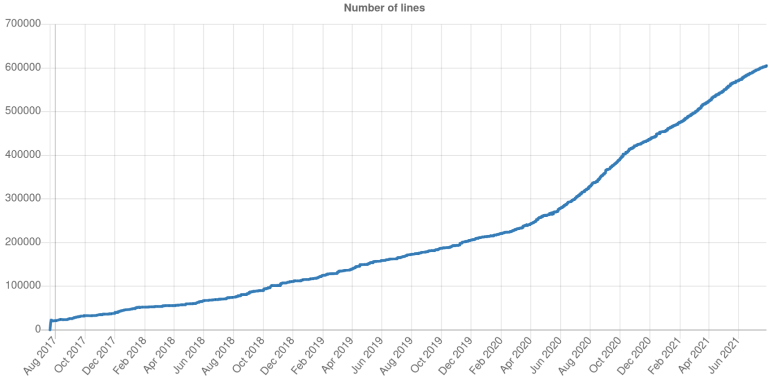


Github: `digama0`
Zulip: Mario Carneiro

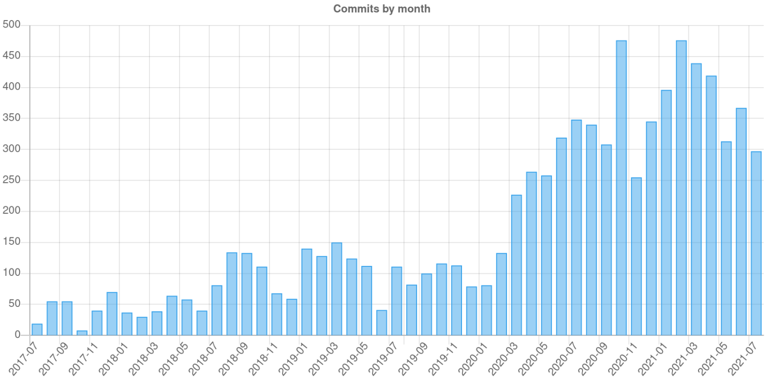
- PhD student in Logic at CMU
- Proof engineering since 2013
  - Metamath (maintainer)
  - Lean 3 mathlib (founder, maintainer)
  - Dabbled in Isabelle, HOL Light, Coq, Mizar
  - Metamath Zero (author)
- Proved 37 of Freek's 100 theorems list in Metamath
- Lots of library code in mathlib
- Say hi at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

Mathlib

- Library of formalized mathematics and computer science
- Lean 3 proof assistant
- de-facto standard library
- Based on Dependent type theory (CIC)
- Uses classical logic (and choice)
- Active community
  - **github**.com/leanprover-community/mathlib
  - leanprover.**zulip**chat.com



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



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

Mathlib goals

Two main goals:

- (CS) To build a standard library for lean as a programming language
  - To support verified programming and proven-correct algorithms
  - To support and provide tactics and decision procedures for proof automation
- (Math) To build a library of formalized mathematics, and support users doing the same

These goals complement each other, it is not just two libraries in one

Mathlib goals: 3 years later

I wrote those goals in a presentation three years ago, and the Math goal has seen considerably more attention than the CS goal, for several reasons:

- The name
- Effective marketing to mathematicians by Kevin Buzzard et al. have skewed the composition of mathlib contributors much farther towards pure mathematicians than comparable formal libraries in other communities
- Poor performance of the lean 3 interpreter means that optimized and verified data structures aren't really worth it

I expect that Lean 4 will push the balance towards more CS applications.

More Mathlib projects

A number of notable projects have been done within or using mathlib:

- Schemes in Lean (v1: Kevin Buzzard, Chris Hughes, Kenny Lau; v2: Amelia Livingston, Ramon Fernández Mir; v3: Scott Morrison)
- Formalising perfectoid spaces (Kevin Buzzard, Johan Commelin, Patrick Massot)
- A formal proof of the independence of the continuum hypothesis (Jesse Michael Han, Floris van Doorn)
- Liquid Tensor Experiment (Johan Commelin, Peter Scholze, Patrick Massot, Adam Topaz, Riccardo Brasca, Kevin Buzzard, Bhavik Mehta, Scott Morrison, Damiano Testa, Heather Macbeth, Filippo A.E. Nuccio, et al)
  - This one has even been the subject of several news articles

A Short History of Lean and mathlib

Several major versions:

- Lean 1 (2013 - no public release)
- Lean 2 (2015) – includes HoTT mode
- Lean 3 (2017)
- Lean 4 (2021, alpha)

- The Lean 2 math library was developed by Jeremy Avigad, Floris van Doorn, Rob Lewis et al.
- Lean 3 is not backwards compatible with Lean 2, and the decision was made to start again taking advantage of significant new features
- `mathlib` is the latest version of the Lean 3 math library, developed by hundreds of contributors
- Lean 4 is not backwards compatible with Lean 3, and the decision was made to **port the library**

The Lean 2 → Lean 3 transition

- Lean 3 added a number of features around the metaprogramming framework
- But the elaborator was scaled back significantly to avoid excessive backtracking that caused bad error messages and flaky scripts
- Reinterpreted Lean 2 files would generally have an error on every other lemma, and there were some small syntax differences as well, so manual porting required heavy touch-ups
- Mathlib was loosely based on the lean 2 library, but was written by hand from scratch
- Lean 2 library is about 40K lines; this was all eventually ported to mathlib within the first year (important stuff right away, other stuff later for completeness)

The Lean 2 → Lean 3 transition

- Lean 3 added a number of features around the metaprogramming framework
- But the elaborator was scaled back significantly to avoid excessive backtracking that caused bad error messages and flaky scripts
- Reinterpreted Lean 2 files would generally have an error on every other lemma, and there were some small syntax differences as well, so manual porting required heavy touch-ups
- Mathlib was loosely based on the lean 2 library, but was written by hand from scratch
- Lean 2 library is about 40K lines; this was all eventually ported to mathlib within the first year (important stuff right away, other stuff later for completeness)
- Mathlib is now *600*K lines, up from 450K at Jan 2021

Lean 4

- A complete reimplementation of Lean in Lean by Leonardo de Moura and Sebastian Ullrich
- Implementation started in 2018, first stable version released Jan 2021
- Endlessly extensible - third party libraries like mathlib can override or extend almost any part of the system
- Compiles to C, so no more slow interpreted tactics
- A much more powerful macro / syntax engine, for first class DSL support
- `do` notation is significantly more powerful

Lean 3 → Lean 4 differences

- Lean 4 is "inspired" by lean 3, it is not a direct upgrade and there is no backward compatibility
  - The basic concepts of theorems, definitions, expressions, tactics, attributes all have equivalents, but many things are moved around
- The lean 4 kernel is (mostly) an extension of the lean 3 kernel
  - Nested and mutual inductives
  - Natural number and string literals (with builtin bignum arithmetic)
  - Lean 3 "macros" are handled differently
  - Opaque definitions (`constant`)
  - A kernel primitive for trusting the evaluation of a compiled lean expression

Lean 3 → Lean 4 differences

- Lots of minor syntax changes
  - `by` *tacs* instead of `begin` *tacs* `end`
  - `match` doesn't have an `end`
  - `fun` x `=>` e instead of λ x`,` e
  - Much more reliance on whitespace sensitivity over punctuation delimiters
- Tactics are now integrated with macros
- Elaboration is sometimes stronger, sometimes weaker
- Typeclass inference is different
- Many lean 3 tactics don't exist yet in lean 4
  - Some are omitted on purpose, either because they can be implemented by users (aka mathlib) or we have a better but incompatible design now

Porting strategies

`binport`: Porting kernel terms

- The plan:
  - Run lean 3
  - Get elaborated expressions
  - Typecheck them with the lean 4 kernel

`binport`: Porting kernel terms

- The plan:
  - Run lean 3
  - Get elaborated expressions
  - Typecheck them with the lean 4 kernel
- This works, although it is very sensitive to unfolding heuristics matching between lean 3 and lean 4

`binport`: Porting kernel terms

- The plan:
  - Run lean 3
  - Get elaborated expressions
  - Typecheck them with the lean 4 kernel
- This works, although it is very sensitive to unfolding heuristics matching between lean 3 and lean 4
- We have ported all of mathlib this way! 🎉

`binport`: Porting kernel terms

- The plan:
  - Run lean 3
  - Get elaborated expressions
  - Typecheck them with the lean 4 kernel
- This works, although it is very sensitive to unfolding heuristics matching between lean 3 and lean 4
- We have ported all of mathlib this way! 🎉
- The result is a set of compiled `.olean` files that can be imported into lean 4 files

Drawbacks of `binport`

- It doesn't produce source files

Drawbacks of `binport`

- It doesn't produce source files
  - Hybrid build process?

Drawbacks of `binport`

- It doesn't produce source files
  - Hybrid build process?
  - but lean 3 files can't refer to lean 4 files

Drawbacks of `binport`

- It doesn't produce source files
  - Hybrid build process?
  - but lean 3 files can't refer to lean 4 files
  - Potential for gradual top-down manual translation

Drawbacks of `binport`

- It doesn't produce source files
  - Hybrid build process?
  - but lean 3 files can't refer to lean 4 files
  - Potential for gradual top-down manual translation
- A lot of very important extra-logical metadata is lost
  - Tactics
  - Elaboration hints for definition unfolding
  - Attributes / simp lemmas
  - Notations

Drawbacks of `binport`

- It doesn't produce source files
  - Hybrid build process?
  - but lean 3 files can't refer to lean 4 files
  - Potential for gradual top-down manual translation
- A lot of very important extra-logical metadata is lost
  - Tactics
  - Elaboration hints for definition unfolding
  - Attributes / simp lemmas
  - Notations
- Lean 4 already has definitions for builtins, like `Nat`, that we want to align
  - This is necessary to get the benefits of e.g. `Nat.add`

Drawbacks of `binport`

- It doesn't produce source files
  - Hybrid build process?
  - but lean 3 files can't refer to lean 4 files
  - Potential for gradual top-down manual translation
- A lot of very important extra-logical metadata is lost
  - Tactics
  - Elaboration hints for definition unfolding
  - Attributes / simp lemmas
  - Notations
- Lean 4 already has definitions for builtins, like `Nat`, that we want to align
  - This is necessary to get the benefits of e.g. `Nat.add`
  - Every alignment has to be defeq or it can break the translation

Drawbacks of `binport`

- It doesn't produce source files
  - Hybrid build process?
  - but lean 3 files can't refer to lean 4 files
  - Potential for gradual top-down manual translation
- A lot of very important extra-logical metadata is lost
  - Tactics
  - Elaboration hints for definition unfolding
  - Attributes / simp lemmas
  - Notations
- Lean 4 already has definitions for builtins, like `Nat`, that we want to align
  - This is necessary to get the benefits of e.g. `Nat.add`
  - Every alignment has to be defeq or it can break the translation
  - Some necessary alignments are not defeq

Drawbacks of `binport`

- It doesn't produce source files
  - Hybrid build process?
  - but lean 3 files can't refer to lean 4 files
  - Potential for gradual top-down manual translation
- A lot of very important extra-logical metadata is lost
  - Tactics
  - Elaboration hints for definition unfolding
  - Attributes / simp lemmas
  - Notations
- Lean 4 already has definitions for builtins, like `Nat`, that we want to align
  - This is necessary to get the benefits of e.g. `Nat.add`
  - Every alignment has to be defeq or it can break the translation
  - Some necessary alignments are not defeq
  - Some alignments are completely different, e.g. `+` is heterogeneous in lean 4

`olean-port`: Reconstructed syntax porting

- The plan:
  - Run lean 3
  - Get compiled `olean` files
  - Reconstruct lean 4 syntax which would have the same effect

`olean-port`: Reconstructed syntax porting

- The plan:
  - Run lean 3
  - Get compiled `olean` files
  - Reconstruct lean 4 syntax which would have the same effect
- Lean 3 uses `olean` files to communicate from one file to the next, so they have much better coverage of notations, attributes, etc.

`olean-port`: Reconstructed syntax porting

- The plan:
  - Run lean 3
  - Get compiled `olean` files
  - Reconstruct lean 4 syntax which would have the same effect
- Lean 3 uses `olean` files to communicate from one file to the next, so they have much better coverage of notations, attributes, etc.
- Can get a result similar to the lean 3 `#print` command

`olean-port`: Reconstructed syntax porting

- The plan:
  - Run lean 3
  - Get compiled `olean` files
  - Reconstruct lean 4 syntax which would have the same effect
- Lean 3 uses `olean` files to communicate from one file to the next, so they have much better coverage of notations, attributes, etc.
- Can get a result similar to the lean 3 `#print` command
- Tactics generally have characteristic proofs, so we can reconstruct the tactics that produced the terms

`olean-port`: Reconstructed syntax porting

- The plan:
  - Run lean 3
  - Get compiled `olean` files
  - Reconstruct lean 4 syntax which would have the same effect
- Lean 3 uses `olean` files to communicate from one file to the next, so they have much better coverage of notations, attributes, etc.
- Can get a result similar to the lean 3 `#print` command
- Tactics generally have characteristic proofs, so we can reconstruct the tactics that produced the terms
  - And if they don't, we can make them leave more explicit annotations

Drawbacks of `olean-port`

- Tactics are completely erased in proofs stored in oleans, and reconstruction is hard

Drawbacks of `olean-port`

- Tactics are completely erased in proofs stored in oleans, and reconstruction is hard
- This does not capture file-local structuring commands:
  - local notations
  - variables
  - sections and namespaces
  - local attributes, or attributes that remove themselves

Drawbacks of `olean-port`

- Tactics are completely erased in proofs stored in oleans, and reconstruction is hard
- This does not capture file-local structuring commands:
  - local notations
  - variables
  - sections and namespaces
  - local attributes, or attributes that remove themselves
- Many autogenerated definitions are mixed in with the “real” definitions
  - Definitional lemmas
  - Theorems generated by the `inductive` command
  - Theorems generated by tactics in a definition
  - Theorems generated by attributes

Drawbacks of `olean-port`

- Tactics are completely erased in proofs stored in oleans, and reconstruction is hard
- This does not capture file-local structuring commands:
  - local notations
  - variables
  - sections and namespaces
  - local attributes, or attributes that remove themselves
- Many autogenerated definitions are mixed in with the "real" definitions
  - Definitional lemmas
  - Theorems generated by the `inductive` command
  - Theorems generated by tactics in a definition
  - Theorems generated by attributes
- Definitions by pattern matching are already compiled

`lean3-port`: Lean 3 re-parsing

- The plan:
  - Use lean 4 to write a lean 3 parser
  - Use it on lean 3 files or snippets

`lean3-port`: Lean 3 re-parsing

- The plan:
  - Use lean 4 to write a lean 3 parser
  - Use it on lean 3 files or snippets
- This is a very flexible approach, since it means we can just have `#lang lean3` sections in a lean 4 file

`lean3-port`: Lean 3 re-parsing

- The plan:
  - Use lean 4 to write a lean 3 parser
  - Use it on lean 3 files or snippets
- This is a very flexible approach, since it means we can just have `#lang lean3` sections in a lean 4 file
- Or we can target it on full files, and either translate them to lean 4 syntax or just run and import them

`lean3-port`: Lean 3 re-parsing

- The plan:
  - Use lean 4 to write a lean 3 parser
  - Use it on lean 3 files or snippets
- This is a very flexible approach, since it means we can just have `#lang lean3` sections in a lean 4 file
- Or we can target it on full files, and either translate them to lean 4 syntax or just run and import them
- Lean 4 has a focus on versatile syntax parsing, and it is already being used to implement a parser very similar to lean 3 (namely lean 4)

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed
- The parser can also run VM code, even IO actions

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed
- The parser can also run VM code, even IO actions
  - The VM code needs an emulation of the lean 3 environment

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed
- The parser can also run VM code, even IO actions
  - The VM code needs an emulation of the lean 3 environment
- Yes, parsing is definitely Turing-complete

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed
- The parser can also run VM code, even IO actions
  - The VM code needs an emulation of the lean 3 environment
- Yes, parsing is definitely Turing-complete

In short, to parse lean 3 you have to be lean 3

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed
- The parser can also run VM code, even IO actions
  - The VM code needs an emulation of the lean 3 environment
- Yes, parsing is definitely Turing-complete

In short, to parse lean 3 you have to be lean 3

- Lean 4 parsers are more restricted

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed
- The parser can also run VM code, even IO actions
  - The VM code needs an emulation of the lean 3 environment
- Yes, parsing is definitely Turing-complete

In short, to parse lean 3 you have to be lean 3

- Lean 4 parsers are more restricted
  - parsing happens all at once before execution

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed
- The parser can also run VM code, even IO actions
  - The VM code needs an emulation of the lean 3 environment
- Yes, parsing is definitely Turing-complete

In short, to parse lean 3 you have to be lean 3

- Lean 4 parsers are more restricted
  - parsing happens all at once before execution
  - parsers can't do IO

Drawbacks of `lean3-port`

- The lean 3 grammar is not at all context free
  - There is an approximate BNF description but it lies a lot
  - The expression grammar is extensible, even locally to an expression
- The parser is very stateful: each command is executed as soon as it is parsed
- The parser can also run VM code, even IO actions
  - The VM code needs an emulation of the lean 3 environment
- Yes, parsing is definitely Turing-complete

In short, to parse lean 3 you have to be lean 3

- Lean 4 parsers are more restricted
  - parsing happens all at once before execution
  - parsers can't do IO
  - parsers can't change the environment (they are non-monadic)

`synport`: AST syntax parsing

- The plan:
  - Modify the lean 3 parser to construct an AST on the side
  - Export the AST in a common format
  - Load the AST into lean 4 and translate it to lean 4 syntax

`synport`: AST syntax parsing

- The plan:
  - Modify the lean 3 parser to construct an AST on the side
  - Export the AST in a common format
  - Load the AST into lean 4 and translate it to lean 4 syntax
- This ensures that we perfectly mimic any lean 3 parser oddities, since lean 3 is doing the parsing

`synport`: AST syntax parsing

- The plan:
  - Modify the lean 3 parser to construct an AST on the side
  - Export the AST in a common format
  - Load the AST into lean 4 and translate it to lean 4 syntax
- This ensures that we perfectly mimic any lean 3 parser oddities, since lean 3 is doing the parsing
- We can get access to all sorts of syntax not available with previous approaches:
  - Tactic block structure
  - Pattern matching definitions
  - Variables, sections, local notations

Drawbacks of `synport`

- We have to instrument lean 3

Drawbacks of `synport`

- We have to instrument lean 3 (done 🎉)

Drawbacks of `synport`

- We have to instrument lean 3 (done 🎉)
- How to adjust to elaboration changes?
  - Lean 3 uses a different elaboration strategy for recursors than lean 4, so most applications of e.g. `Nat.rec` fail

Drawbacks of `synport`

- We have to instrument lean 3 (done 🎉)
- How to adjust to elaboration changes?
  - Lean 3 uses a different elaboration strategy for recursors than lean 4, so most applications of e.g. `Nat.rec` fail
- How to adjust to tactic changes?
  - e.g. We know the lean 3 code used `simp` here but it doesn't work in lean 4

Drawbacks of `synport`

- We have to instrument lean 3 (done 🎉)
- How to adjust to elaboration changes?
  - Lean 3 uses a different elaboration strategy for recursors than lean 4, so most applications of e.g. `Nat.rec` fail
- How to adjust to tactic changes?
  - e.g. We know the lean 3 code used `simp` here but it doesn't work in lean 4
- We need implementations for all lean 3 tactics
  - The meta framework is very different, so we are not planning to port meta code directly

Drawbacks of `synport`

- We have to instrument lean 3 (done 🎉)
- How to adjust to elaboration changes?
  - Lean 3 uses a different elaboration strategy for recursors than lean 4, so most applications of e.g. `Nat.rec` fail
- How to adjust to tactic changes?
  - e.g. We know the lean 3 code used `simp` here but it doesn't work in lean 4
- We need implementations for all lean 3 tactics
  - The meta framework is very different, so we are not planning to port meta code directly
- Some commands manipulate lean 3 state that is expressed differently in lean 4, like `precedence`
  - Should we just delete these?

The manual approach: `mathlib4`

- The plan:
  - Take inspiration from lean 3 mathlib
  - Write lean 4 files by hand

The manual approach: `mathlib4`

- The plan:
  - Take inspiration from lean 3 mathlib
  - Write lean 4 files by hand
- Best quality results

The manual approach: `mathlib4`

- The plan:
  - Take inspiration from lean 3 mathlib
  - Write lean 4 files by hand
- Best quality results
- Able to adapt to areas where lean 4 is just too different

The manual approach: `mathlib4`

- The plan:
  - Take inspiration from lean 3 mathlib
  - Write lean 4 files by hand
- Best quality results
- Able to adapt to areas where lean 4 is just too different
- No startup cost besides training people to write lean 4 code

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

What to do about ongoing changes? Mathlib gets 10 PRs a day

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

What to do about ongoing changes? Mathlib gets 10 PRs a day

- Stop the world and port everything?

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

What to do about ongoing changes? Mathlib gets 10 PRs a day

- Stop the world and port everything?
  - Could be done if we get everyone together to work on it

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

What to do about ongoing changes? Mathlib gets 10 PRs a day

- Stop the world and port everything?
  - Could be done if we get everyone together to work on it
  - The skillset needed for porting is not the same as for authoring

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

What to do about ongoing changes? Mathlib gets 10 PRs a day

- Stop the world and port everything?
  - Could be done if we get everyone together to work on it
  - The skillset needed for porting is not the same as for authoring
- Freeze parts of mathlib and port bottom-up

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

What to do about ongoing changes? Mathlib gets 10 PRs a day

- Stop the world and port everything?
  - Could be done if we get everyone together to work on it
  - The skillset needed for porting is not the same as for authoring
- Freeze parts of mathlib and port bottom-up
  - Easy for theorems to get lost in the shuffle

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

What to do about ongoing changes? Mathlib gets 10 PRs a day

- Stop the world and port everything?
  - Could be done if we get everyone together to work on it
  - The skillset needed for porting is not the same as for authoring
- Freeze parts of mathlib and port bottom-up
  - Easy for theorems to get lost in the shuffle
- Fork mathlib, keep both versions in sync

Drawbacks of manual porting

- Obviously doesn't scale: the porting process itself will take a month or more

What to do about ongoing changes? Mathlib gets 10 PRs a day

- Stop the world and port everything?
  - Could be done if we get everyone together to work on it
  - The skillset needed for porting is not the same as for authoring
- Freeze parts of mathlib and port bottom-up
  - Easy for theorems to get lost in the shuffle
- Fork mathlib, keep both versions in sync
  - Easy for theorems to get lost in the shuffle

What we are actually doing

- `binport`: Porting kernel terms
- `olean-port`: Reconstructed syntax porting
- `lean3-port`: Lean 3 re-parsing
- `synport`: AST syntax parsing
- `mathlib4`: Manual porting

What we are actually doing

- `binport`: Porting kernel terms
- `olean-port`: Reconstructed syntax porting
- `lean3-port`: Lean 3 re-parsing
- `synport`: AST syntax parsing
- `mathlib4`: Manual porting

What we are actually doing

A combination of several strategies:

- `mathlib4`: A from scratch implementation of mathlib foundations in lean 4
  - Tactics implemented here
  - Foundational theories like `Data.Nat.Basic` that are useful and not too hard to port
  - Setting up syntax to be used by the porting tools

What we are actually doing

A combination of several strategies:

- `mathlib4`: A from scratch implementation of mathlib foundations in lean 4
  - Tactics implemented here
  - Foundational theories like `Data.Nat.Basic` that are useful and not too hard to port
  - Setting up syntax to be used by the porting tools
- `binport`: Translating lean 3 proof data into lean 4 `oleans`
  - Useful for getting a context for files in the middle or top of the dependency graph for added parallelism

What we are actually doing

A combination of several strategies:

- `mathlib4`: A from scratch implementation of mathlib foundations in lean 4
  - Tactics implemented here
  - Foundational theories like `Data.Nat.Basic` that are useful and not too hard to port
  - Setting up syntax to be used by the porting tools
- `binport`: Translating lean 3 proof data into lean 4 `oleans`
  - Useful for getting a context for files in the middle or top of the dependency graph for added parallelism
- `synport`: Translating lean 3 AST data into lean 4 source files
  - Uses `lean --ast`, implemented in a fork of lean 3
  - Provides a starting point for manual editing

Improvements

- Idea: translate elaboration info
  - Attach info about elaborated exprs to AST nodes in lean 3
  - Produce syntax in `synport` with the lean 4 elaborator available

Improvements

- Idea: translate elaboration info
  - Attach info about elaborated exprs to AST nodes in lean 3
  - Produce syntax in `synport` with the lean 4 elaborator available
  - When a syntax is not going to elaborate the way we want, select a more explicit syntax

Improvements

- Idea: translate elaboration info
  - Attach info about elaborated exprs to AST nodes in lean 3
  - Produce syntax in `synport` with the lean 4 elaborator available
  - When a syntax is not going to elaborate the way we want, select a more explicit syntax
    - ... or not

Improvements

- Idea: translate elaboration info
  - Attach info about elaborated exprs to AST nodes in lean 3
  - Produce syntax in `synport` with the lean 4 elaborator available
  - When a syntax is not going to elaborate the way we want, select a more explicit syntax
    - . . . or not
- Add more backward compatible syntax to ease manual translations

Improvements

- Idea: translate elaboration info
  - Attach info about elaborated exprs to AST nodes in lean 3
  - Produce syntax in `synport` with the lean 4 elaborator available
  - When a syntax is not going to elaborate the way we want, select a more explicit syntax
    - …or not
- Add more backward compatible syntax to ease manual translations
  - Requires a post processing step to remove the syntax if we want to change the style guide

Improvements

- Idea: translate elaboration info
  - Attach info about elaborated exprs to AST nodes in lean 3
  - Produce syntax in `synport` with the lean 4 elaborator available
  - When a syntax is not going to elaborate the way we want, select a more explicit syntax
    - . . . or not
- Add more backward compatible syntax to ease manual translations
  - Requires a post processing step to remove the syntax if we want to change the style guide
- Manage alignments through `#align` annotations in ported files
  - Useful for `binport` to be able to stay in sync

Improvements

- Idea: translate elaboration info
  - Attach info about elaborated exprs to AST nodes in lean 3
  - Produce syntax in `synport` with the lean 4 elaborator available
  - When a syntax is not going to elaborate the way we want, select a more explicit syntax
    - ... or not
- Add more backward compatible syntax to ease manual translations
  - Requires a post processing step to remove the syntax if we want to change the style guide
- Manage alignments through `#align` annotations in ported files
  - Useful for `binport` to be able to stay in sync
  - Still usable even if the alignments are not defeq as long as downstream uses are also realigned

Translation examples: lt_or_ge

```
-- Lean 3
protected lemma lt_or_ge : ∀ (a b : ℕ), a < b ∨ b ≤ a
| a 0     := or.inr (zero_le a)
| a (b+1) :=
  match lt_or_ge a b with
  | or.inl h := or.inl (le_succ_of_le h)
  | or.inr h :=
    match nat.eq_or_lt_of_le h with
    | or.inl h1 := or.inl (h1 ▸ lt_succ_self b)
    | or.inr h1 := or.inr h1
    end
  end

-- Lean 4
protected theorem lt_or_ge : (a b : ℕ) → a < b ∨ b ≤ a
| a, 0 => Or.inr (zero_le a)
| a, b + 1 =>
  match lt_or_ge a b with
  | Or.inl h => Or.inl (le_succ_of_le h)
  | Or.inr h =>
    match nat.eq_or_lt_of_le h with
    | Or.inl h1 => Or.inl (h1 ▸ lt_succ_self b)
    | Or.inr h1 => Or.inr h1
```

Translation examples: div_inv_monoid

```
-- Lean 3
/-- A `div_inv_monoid` is a `monoid` with operations `/` and `⁻¹`... -/
@[protect_proj, ancestor monoid has_inv has_div]
class div_inv_monoid (G : Type u) extends monoid G, has_inv G, has_div G :=
(div := λ a b, a * b⁻¹)
(div_eq_mul_inv : ∀ a b : G, a / b = a * b⁻¹ . try_refl_tac)
(gpow : ℤ → G → G := gpow_rec)
(gpow_zero' : ∀ (a : G), gpow 0 a = 1 . try_refl_tac)
(gpow_succ' :
  ∀ (n : ℕ) (a : G), gpow (int.of_nat n.succ) a = a * gpow (int.of_nat n) a  . try_refl_tac)
(gpow_neg' :
  ∀ (n : ℕ) (a : G), gpow (-[1+ n]) a = (gpow n.succ a) ⁻¹ . try_refl_tac)

-- Lean 4
/-- A `DivInvMonoid` is a `Monoid` with operations `/` and `⁻¹`... -/
@[protectProj]
class DivInvMonoid (G : Type u) extends Monoid G, Inv G, Div G where
  div := fun a b => a * b⁻¹
  div_eq_mul_inv : (a b : G) → a / b = a * b⁻¹ := by try_refl_tac
  gpow : ℤ → G → G := gpow_rec
  gpow_zero' : (a : G) → gpow 0 a = 1 := by try_refl_tac
  gpow_succ' : (n : ℕ) → (a : G) → gpow (int.of_nat n.succ) a = a * gpow (int.of_nat n) a
    := by try_refl_tac
  gpow_neg' : (n : ℕ) → (a : G) → gpow -[1+ n] a = (gpow n.succ a)⁻¹ := by try_refl_tac
```

Translation examples: nat.mod_lt

```
-- Lean 3
lemma mod_lt (x : nat) {y : nat} (h : 0 < y) : x % y < y :=
begin
  induction x using nat.case_strong_induction_on with x ih,
  { rw zero_mod, assumption },
  { by_cases h₁ : succ x < y,
    { rwa [mod_eq_of_lt h₁] },
    { have h₁ : succ x % y = (succ x - y) % y := mod_eq_sub_mod (not_lt.1 h₁),
      have : succ x - y ≤ x := le_of_lt_succ (sub_lt (succ_pos x) h),
      have h₂ : (succ x - y) % y < y := ih _ this,
      rwa [← h₁] at h₂ } }
end

-- Lean 4
theorem mod_lt (x : Nat) {y : Nat} (h : 0 < y) : x % y < y := by
  induction' x using Nat.case_strong_induction_on with x ih
  · rw [zero_mod]; assumption
  · byCases h₁ : succ x < y
    · rwa [mod_eq_of_lt h₁]
    · have h₁ : succ x % y = (succ x - y) % y := mod_eq_sub_mod (not_lt.1 h₁)
      have : succ x - y ≤ x := le_of_lt_succ (sub_lt (succ_pos x) h)
      have h₂ : (succ x - y) % y < y := ih _ this
      rwa [← h₁] at h₂
```

Conclusion

- This is quite possibly the largest source-level proof porting project ever
- Mathlib's high (and growing) activity rate and many contributors lead to some logistical challenges
- The techniques discussed here apply generally to any source-level translations
  - Lean 3 is in many ways a worst case for this kind of job
  - even translating Coq or Isabelle to Lean would follow a similar path

Conclusion

- This is quite possibly the largest source-level proof porting project ever
- Mathlib's high (and growing) activity rate and many contributors lead to some logistical challenges
- The techniques discussed here apply generally to any source-level translations
  - Lean 3 is in many ways a worst case for this kind of job
  - even translating Coq or Isabelle to Lean would follow a similar path
- We really hope we don't have to do this again in lean 5

Resources

- Lean/mathlib: `http://leanprover-community.github.io/`
- Lean 4: `https://github.com/leanprover/lean4/`
- Mathport: `https://github.com/dselsam/mathport`
- Zulip: `https://leanprover.zulipchat.com/`
  - Porting discussions are on **#lean4** and **#mathlib4** streams

Thanks!