`mathlib`: Lean's mathematical library

Johannes Hölzl

VU VRIJE UNIVERSITEIT AMSTERDAM

EUTypes 2018

Introduction

- (classical) mathematical library for Lean
  with computable exceptions, e.g. $\mathbb{N}$, $\mathbb{Z}$, lists, ...

Introduction

- (classical) mathematical library for Lean
  with computable exceptions, e.g. $\mathbb{N}$, $\mathbb{Z}$, lists, ...
- Formerly distributed with Lean itself
  Leo wanted more flexibility

Introduction

- (classical) mathematical library for Lean
  with computable exceptions, e.g. $\mathbb{N}$, $\mathbb{Z}$, lists, ...
- Formerly distributed with Lean itself
  Leo wanted more flexibility
- Some (current) topics:
  Basic Datatypes, Analysis, Linear Algebra, Set Theory, ...

Lean

Lean DTT

- Quotient types (implies `funext`)

Lean DTT

- Quotient types (implies `funext`)
- Proof irrelevance is a definitional equality

Lean DTT

- Quotient types (implies `funext`)
- Proof irrelevance is a definitional equality
- Mark constants `noncomputable`, i.e. functions using `choice`:

$$\texttt{axiom choice} : \Pi(\alpha : \texttt{Sort}\ u), \texttt{nonempty}\ \alpha \to \alpha$$

Lean DTT

- Quotient types (implies `funext`)
- Proof irrelevance is a definitional equality
- Mark constants `noncomputable`,
  i.e. functions using `choice`:

$$\texttt{axiom choice} : \Pi(\alpha : \texttt{Sort}\ u), \texttt{nonempty}\ \alpha \to \alpha$$

- Non-commulative universes $\texttt{Prop} : \texttt{Type}\ 0 : \texttt{Type}\ 1 : \cdots$

Lean DTT

- Quotient types (implies `funext`)
- Proof irrelevance is a definitional equality
- Mark constants `noncomputable`,
  i.e. functions using `choice`:

$$\texttt{axiom choice} : \Pi(\alpha : \texttt{Sort}\ u), \texttt{nonempty}\ \alpha \to \alpha$$

- Non-commulative universes $\texttt{Prop} : \texttt{Type}\ 0 : \texttt{Type}\ 1 : \cdots$
- Basic inductives in the kernel

Lean DTT

- Quotient types (implies `funext`)
- Proof irrelevance is a definitional equality
- Mark constants `noncomputable`,
  i.e. functions using `choice`:

$$\texttt{axiom choice} : \Pi(\alpha : \texttt{Sort}\ u), \texttt{nonempty}\ \alpha \to \alpha$$

- Non-commulative universes $\texttt{Prop} : \texttt{Type}\ 0 : \texttt{Type}\ 1 : \cdots$
- Basic inductives in the kernel
  - Mutual and nested inductives are constructed

Lean DTT

- Quotient types (implies `funext`)
- Proof irrelevance is a definitional equality
- Mark constants `noncomputable`,
  i.e. functions using `choice`:

$$\texttt{axiom choice} : \Pi(\alpha : \texttt{Sort}\ u), \texttt{nonempty}\ \alpha \to \alpha$$

- Non-commulative universes $\texttt{Prop} : \texttt{Type}\ 0 : \texttt{Type}\ 1 : \cdots$
- Basic inductives in the kernel
  - Mutual and nested inductives are constructed
  - No general fixpoint operator, no general match operator
    these are derived from recursors

Type classes in Lean

- Type classes are used to fill in implicit values:

$$\texttt{add} : \Pi\{\alpha : \texttt{Type}\}[i : \texttt{has\_add}\ \alpha],\ \alpha \to \alpha \to \alpha$$

$$a + b \equiv \texttt{@add}\ \mathbb{N}\ \texttt{nat.add}\ a\ b$$

Type classes in Lean

- Type classes are used to fill in implicit values:

$$\texttt{add} : \Pi\{\alpha : \texttt{Type}\}[i : \texttt{has\_add}\ \alpha],\ \alpha \to \alpha \to \alpha$$

$$a + b \equiv \texttt{@add}\ \mathbb{N}\ \texttt{nat.add}\ a\ b$$

- Instances can depend on other instances:

$$\texttt{ring.to\_group} : \Pi(\alpha : \texttt{Type})[i : \texttt{ring}\ \alpha] : \texttt{group}\ \alpha$$

Type classes in Lean

- Type classes are used to fill in implicit values:

$$\texttt{add} : \Pi\{\alpha : \texttt{Type}\}[i : \texttt{has\_add}\ \alpha],\ \alpha \to \alpha \to \alpha$$

$$a + b \equiv \texttt{@add}\ \mathbb{N}\ \texttt{nat.add}\ a\ b$$

- Instances can depend on other instances:

$$\texttt{ring.to\_group} : \Pi(\alpha : \texttt{Type})[i : \texttt{ring}\ \alpha] : \texttt{group}\ \alpha$$

- Output parameters:

$$\begin{aligned}
&\texttt{has_mem} && : \texttt{Type} \to \texttt{out Type} \to \texttt{Type} \\
&\texttt{set.has_mem} && : \Pi\alpha, \texttt{has_mem}\ (\texttt{set}\ \alpha)\ \alpha \\
&\texttt{fset.has_mem} && : \Pi\alpha\ [i : \texttt{decidable_eq}\ \alpha], \texttt{has_mem}\ (\texttt{fset}\ \alpha)\ \alpha
\end{aligned}$$

Type classes in Lean

- Type classes are used to fill in implicit values:

$$\texttt{add} : \Pi\{\alpha : \texttt{Type}\}[i : \texttt{has\_add}\ \alpha],\ \alpha \to \alpha \to \alpha$$

$$a + b \equiv \texttt{@add}\ \mathbb{N}\ \texttt{nat.add}\ a\ b$$

- Instances can depend on other instances:

$$\texttt{ring.to\_group} : \Pi(\alpha : \texttt{Type})[i : \texttt{ring}\ \alpha] : \texttt{group}\ \alpha$$

- Output parameters:

$$\begin{aligned}
&\texttt{has_mem} && : \texttt{Type} \to \texttt{out}\ \texttt{Type} \to \texttt{Type} \\
&\texttt{set.has_mem} && : \Pi\alpha, \texttt{has_mem}\ (\texttt{set}\ \alpha)\ \alpha \\
&\texttt{fset.has_mem} && : \Pi\alpha\ [i : \texttt{decidable_eq}\ \alpha], \texttt{has_mem}\ (\texttt{fset}\ \alpha)\ \alpha
\end{aligned}$$

- Default values

Library

Library

- Basic (computable) data
- Type class hierarchies:

Orders	orders, lattices
Algebraic	(commutative) groups, rings, fields
Spaces	measurable, topological, uniform, metric

- Set theory (cardinals & ordinals)
- Analysis
- Linear algebra

Basic (computable) data

- Numbers: $\mathbb{N}$, $\mathbb{Z}$ (as datatype, not quotient), $\mathbb{Q}$, Fin n

Basic (computable) data

- Numbers: $\mathbb{N}$, $\mathbb{Z}$ (as datatype, not quotient), $\mathbb{Q}$, Fin n
- Lists, `set` $\alpha := \alpha \to$ `Prop`

Basic (computable) data

- Numbers: $\mathbb{N}$, $\mathbb{Z}$ (as datatype, not quotient), $\mathbb{Q}$, Fin *n*
- Lists, `set` $\alpha := \alpha \to$ `Prop`
- `multiset` $\alpha :=$ `list` $\alpha/_{\texttt{perm}}$

Basic (computable) data

- Numbers: $\mathbb{N}$, $\mathbb{Z}$ (as datatype, not quotient), $\mathbb{Q}$, Fin n
- Lists, $\texttt{set}\ \alpha := \alpha \to \texttt{Prop}$
- $\texttt{multiset}\ \alpha := \texttt{list}\ \alpha/_{\texttt{perm}}$
- $\texttt{finset}\ \alpha := \{m : \texttt{multiset}\ \alpha \mid \texttt{nodup}\ m\}$

Basic (computable) data

- Numbers: $\mathbb{N}$, $\mathbb{Z}$ (as datatype, not quotient), $\mathbb{Q}$, Fin n
- Lists, $\texttt{set}\ \alpha := \alpha \to \texttt{Prop}$
- $\texttt{multiset}\ \alpha := \texttt{list}\ \alpha/_{\texttt{perm}}$
- $\texttt{finset}\ \alpha := \{m : \texttt{multiset}\ \alpha \mid \texttt{nodup}\ m\}$
- Big operators for `list`, `multiset` and `finset`

Set theory: Cardinals and Ordinals (Mario Carneiro)

- Zorn's lemma, Schröder-Bernstein, ...

Set theory: Cardinals and Ordinals (Mario Carneiro)

- Zorn's lemma, Schröder-Bernstein, ...
- Isomorphism:
  `structure` $\alpha \simeq \beta :=$
  $(f : \alpha \to \beta)(g : \beta \to \alpha)(f\_g : f \circ g = id)(g\_f : g \circ f = id)$

Set theory: Cardinals and Ordinals (Mario Carneiro)

- Zorn's lemma, Schröder-Bernstein, ...
- Isomorphism:
  `structure` $\alpha \simeq \beta :=$
  $(f : \alpha \to \beta)(g : \beta \to \alpha)(f\_g : f \circ g = id)(g\_f : g \circ f = id)$
- Cardinals & ordinals are well-order

$$\begin{array}{llll} \texttt{cardinal}_u & : & \texttt{Type}_{u+1} & := & \texttt{Type}_u /_{\texttt{nonempty}} \simeq \\ \texttt{ordinal}_u & : & \texttt{Type}_{u+1} & := & \texttt{Well\_order}_u /_{\texttt{nonempty}} \simeq_{ord} \end{array}$$

Set theory: Cardinals and Ordinals (Mario Carneiro)

- Zorn's lemma, Schröder-Bernstein, ...
- Isomorphism:
  $\texttt{structure}\ \alpha \simeq \beta :=$
  $(f : \alpha \to \beta)(g : \beta \to \alpha)(f\_g : f \circ g = id)(g\_f : g \circ f = id)$
- Cardinals & ordinals are well-order

$$\begin{array}{llll} \texttt{cardinal}_u & : & \texttt{Type}_{u+1} & := & \texttt{Type}_u / _{\texttt{nonempty}\ \simeq} \\ \texttt{ordinal}_u & : & \texttt{Type}_{u+1} & := & \texttt{Well\_order}_u / _{\texttt{nonempty}\ \simeq_{ord}} \end{array}$$

- Semiring structure of `cardinal` is proved using $\simeq$ constructions

Set theory: Cardinals and Ordinals (Mario Carneiro)

- Zorn's lemma, Schröder-Bernstein, ...
- Isomorphism:
  `structure` $\alpha \simeq \beta :=$
  $(f : \alpha \to \beta)(g : \beta \to \alpha)(f\_g : f \circ g = id)(g\_f : g \circ f = id)$
- Cardinals & ordinals are well-order

$$\begin{array}{lcll} \texttt{cardinal}_u & : & \texttt{Type}_{u+1} & := \texttt{Type}_u/_{\texttt{nonempty}} \simeq \\ \texttt{ordinal}_u & : & \texttt{Type}_{u+1} & := \texttt{Well\_order}_u/_{\texttt{nonempty}} \simeq_{ord} \end{array}$$

- Semiring structure of `cardinal` is proved using $\simeq$ constructions
- $\kappa + \kappa = \kappa = \kappa * \kappa$ (for $\kappa \geq \omega$)

Set theory: Cardinals and Ordinals (Mario Carneiro)

- Zorn's lemma, Schröder-Bernstein, ...
- Isomorphism:
  `structure` $\alpha \simeq \beta :=$
  $(f : \alpha \to \beta)(g : \beta \to \alpha)(f\_g : f \circ g = id)(g\_f : g \circ f = id)$
- Cardinals & ordinals are well-order

  $$\begin{array}{llll} \texttt{cardinal}_u & : \texttt{Type}_{u+1} & := & \texttt{Type}_u/_{\texttt{nonempty}\ \simeq} \\ \texttt{ordinal}_u & : \texttt{Type}_{u+1} & := & \texttt{Well\_order}_u/_{\texttt{nonempty}\ \simeq_{ord}} \end{array}$$

- Semiring structure of `cardinal` is proved using $\simeq$ constructions
- $\kappa + \kappa = \kappa = \kappa * \kappa$ (for $\kappa \geq \omega$)
- Existence of inaccessible cardinals (i.e. in the next universe)

Analysis

- Derived from Isabelle's analysis (Filters to generalize limits)

Analysis

- Derived from Isabelle's analysis (Filters to generalize limits)
- Topology: open; nhds filter, closed, compact, interior, closure

Analysis

- Derived from Isabelle's analysis (Filters to generalize limits)
- Topology: open; nhds filter, closed, compact, interior, closure
- Uniformity: complete, totally bounded
  (compact $\leftrightarrow$ complete and totally bounded)

Analysis

- Derived from Isabelle's analysis (Filters to generalize limits)
- Topology: open; nhds filter, closed, compact, interior, closure
- Uniformity: complete, totally bounded
  (compact $\leftrightarrow$ complete and totally bounded)
- Metric spaces are only rudimentary

Analysis

- Derived from Isabelle's analysis (Filters to generalize limits)
- Topology: open; nhds filter, closed, compact, interior, closure
- Uniformity: complete, totally bounded
  (compact $\leftrightarrow$ complete and totally bounded)
- Metric spaces are only rudimentary
- Measurable spaces, Measures & Lebesgue measure

Analysis

- Derived from Isabelle's analysis (Filters to generalize limits)
- Topology: open; nhds filter, closed, compact, interior, closure
- Uniformity: complete, totally bounded
  (compact $\leftrightarrow$ complete and totally bounded)
- Metric spaces are only rudimentary
- Measurable spaces, Measures & Lebesgue measure
- Infinite sum on topological monoids α:
  $\Sigma : \forall \iota, (\iota \to \alpha) \to \alpha$

Analysis: Analytical Structures as Complete Lattices

Complete lattices, `map` & `comap` as category theory *light*

- Filters, topological spaces, uniform spaces, and measurable spaces form a complete lattices per type

$$\texttt{complete\_lattice}\,(\texttt{topology}\;\alpha)$$

Analysis: Analytical Structures as Complete Lattices

Complete lattices, `map` & `comap` as category theory *light*

- Filters, topological spaces, uniform spaces, and measurable spaces form a complete lattices per type

$$\texttt{complete\_lattice}\ (\texttt{topology}\ \alpha)$$

- (Co) induced structures allow for easy constructions:

$$\texttt{map} : \Pi\{\alpha\beta\}, (\alpha \to \beta) \to (\texttt{topology}\ \alpha \to \texttt{topology}\ \beta)$$
$$\texttt{comap} : \Pi\{\alpha\beta\}, (\alpha \to \beta) \to (\texttt{topology}\ \beta \to \texttt{topology}\ \alpha)$$

Analysis: Analytical Structures as Complete Lattices

Complete lattices, `map` & `comap` as category theory *light*

- Filters, topological spaces, uniform spaces, and measurable spaces form a complete lattices per type

$$\texttt{complete\_lattice}\ (\texttt{topology}\ \alpha)$$

- (Co) induced structures allow for easy constructions:

$$\texttt{map} : \Pi\{\alpha\beta\}, (\alpha \to \beta) \to (\texttt{topology}\ \alpha \to \texttt{topology}\ \beta)$$
$$\texttt{comap} : \Pi\{\alpha\beta\}, (\alpha \to \beta) \to (\texttt{topology}\ \beta \to \texttt{topology}\ \alpha)$$

- Easy constructions:

$$\texttt{prod}\ t_1\ t_2 := \texttt{comap}\ \pi_1\ t_1 \sqcup \texttt{comap}\ \pi_2\ t_2$$
$$\texttt{subtype}\ t\ s := \texttt{comap}\ (\texttt{subtype.val}\ s)\ t$$

Analysis: Analytical Structures as Complete Lattices

Complete lattices, `map` & `comap` as category theory *light*

- Filters, topological spaces, uniform spaces, and measurable spaces form a complete lattices per type

$$\texttt{complete\_lattice}\ (\texttt{topology}\ \alpha)$$

- (Co) induced structures allow for easy constructions:

$$\texttt{map} : \Pi\{\alpha\beta\}, (\alpha \to \beta) \to (\texttt{topology}\ \alpha \to \texttt{topology}\ \beta)$$
$$\texttt{comap} : \Pi\{\alpha\beta\}, (\alpha \to \beta) \to (\texttt{topology}\ \beta \to \texttt{topology}\ \alpha)$$

- Easy constructions:

$$\texttt{prod}\ t_1\ t_2 := \texttt{comap}\ \pi_1\ t_1 \sqcup \texttt{comap}\ \pi_2\ t_2$$
$$\texttt{subtype}\ t\ s := \texttt{comap}\ (\texttt{subtype.val}\ s)\ t$$

- Straight forward derivation of continuity rules

Analysis: Type Class Structure

$\texttt{class metric}\ (\alpha : \mathit{Type}) := \ldots$
$\texttt{instance m2t}\ (\alpha : \mathit{Type})\ [\texttt{metric}\ \alpha] : \texttt{topology}\ \alpha :=$
$\{\texttt{open}\ s := \forall x \in s, \exists \epsilon > 0, \texttt{ball}\ x\ \epsilon \subseteq s, \ldots\}$

Analysis: Type Class Structure

$\texttt{class metric } (\alpha : \mathit{Type}) := \ldots$
$\texttt{instance m2t } (\alpha : \mathit{Type})\ [\texttt{metric } \alpha] : \texttt{topology } \alpha :=$
$\{\texttt{open } s := \forall x \in s, \exists \epsilon > 0, \texttt{ball } x\ \epsilon \subseteq s, \ldots\}$

Problem: $\texttt{m2t } (m_1 \times m_2) \not\equiv (\texttt{m2t } m_1) \times (\texttt{m2t } m_2)$

Analysis: Type Class Structure

$\texttt{class metric}\ (\alpha : \mathit{Type}) := \ldots$
$\texttt{instance m2t}\ (\alpha : \mathit{Type})\ [\texttt{metric}\ \alpha] : \texttt{topology}\ \alpha :=$
$\{\texttt{open}\ s := \forall x \in s, \exists \epsilon > 0, \texttt{ball}\ x\ \epsilon \subseteq s, \ldots\}$

Problem: $\texttt{m2t}\ (m_1 \times m_2) \not\equiv (\texttt{m2t}\ m_1) \times (\texttt{m2t}\ m_2)$

$\texttt{class metric}\ (\alpha : \mathit{Type})\ \texttt{extends topology}\ \alpha :=$
$\ldots$
$(\texttt{open\_iff} : \forall s, \texttt{open}\ s \iff \forall x \in s, \exists \epsilon > 0, \texttt{ball}\ x\ \epsilon \subseteq s)$

Analysis: Type Class Structure

$\texttt{class metric}\ (\alpha : \mathit{Type}) := \ldots$
$\texttt{instance m2t}\ (\alpha : \mathit{Type})\ [\texttt{metric}\ \alpha] : \texttt{topology}\ \alpha :=$
$\{\texttt{open}\ s := \forall x \in s, \exists \epsilon > 0, \texttt{ball}\ x\ \epsilon \subseteq s, \ldots\}$

Problem: $\texttt{m2t}\ (m_1 \times m_2) \not\equiv (\texttt{m2t}\ m_1) \times (\texttt{m2t}\ m_2)$

$\texttt{class metric}\ (\alpha : \mathit{Type})\ \texttt{extends topology}\ \alpha :=$
$\ldots$
$(\texttt{open\_iff} : \forall s, \texttt{open}\ s \iff \forall x \in s, \exists \epsilon > 0, \texttt{ball}\ x\ \epsilon \subseteq s)$

Default values give a value for the topology when defining metric

Analysis: Constructing Reals

Construct $\mathbb{R}$ using completion of $\mathbb{Q}$

- For foundational reasons metric completion is not possible

Analysis: Constructing Reals

Construct $\mathbb{R}$ using completion of $\mathbb{Q}$

- For foundational reasons metric completion is not possible (alt: $\alpha \to \alpha \to \mathbb{Q} \to$ `Prop`, c.f. Krebbers & Spitters)

Analysis: Constructing Reals

Construct $\mathbb{R}$ using completion of $\mathbb{Q}$

- For foundational reasons metric completion is not possible (alt: $\alpha \to \alpha \to \mathbb{Q} \to$ `Prop`, c.f. Krebbers & Spitters)
- Formalize uniform spaces (using the filter library!)

Analysis: Constructing Reals

Construct $\mathbb{R}$ using completion of $\mathbb{Q}$

- For foundational reasons metric completion is not possible (alt: $\alpha \to \alpha \to \mathbb{Q} \to$ `Prop`, c.f. Krebbers & Spitters)
- Formalize uniform spaces (using the filter library!)
- Use completion on the uniform space $\mathbb{Q}$

Analysis: Constructing Reals

Construct $\mathbb{R}$ using completion of $\mathbb{Q}$

- For foundational reasons metric completion is not possible (alt: $\alpha \to \alpha \to \mathbb{Q} \to$ `Prop`, c.f. Krebbers & Spitters)
- Formalize uniform spaces (using the filter library!)
- Use completion on the uniform space $\mathbb{Q}$
- Is it worth it?

Analysis: Constructing Reals

Construct $\mathbb{R}$ using completion of $\mathbb{Q}$

- For foundational reasons metric completion is not possible (alt: $\alpha \to \alpha \to \mathbb{Q} \to$ `Prop`, c.f. Krebbers & Spitters)
- Formalize uniform spaces (using the filter library!)
- Use completion on the uniform space $\mathbb{Q}$
- Is it worth it?
  Mario wants to go back to Cauchy sequences...

Analysis: Constructing Reals

Construct $\mathbb{R}$ using completion of $\mathbb{Q}$

- For foundational reasons metric completion is not possible (alt: $\alpha \to \alpha \to \mathbb{Q} \to$ `Prop`, c.f. Krebbers & Spitters)
- Formalize uniform spaces (using the filter library!)
- Use completion on the uniform space $\mathbb{Q}$
- Is it worth it?
  Mario wants to go back to Cauchy sequences...
- Anyway: $\mathbb{R}$ as order & topologically complete field

Linear Algebra

$$\texttt{class}\ \texttt{module}\ (\alpha : \texttt{out}\ \texttt{Type}_u)\ (\beta : \texttt{Type}_v)\ [\texttt{out}\ \texttt{ring}\ \alpha] := \ldots$$

Linear Algebra

```
class module (α : out Type_u) (β : Type_v) [out ring α] := ...
```

- type class mechanism looks for `module _ β _`

Linear Algebra

class module (α : out Type$_u$) (β : Type$_v$) [out ring α] := ...

- type class mechanism looks for module _ β _
- only one canoncial module per type

Linear Algebra

$\texttt{class module } (\alpha : \texttt{out Type}_u)\ (\beta : \texttt{Type}_v)\ [\texttt{out ring } \alpha] := \ldots$

- type class mechanism looks for `module` _ β _
- only one canoncial `module` per type
- usually α is fixed per theory anyway

Linear Algebra

```
class module (α : out Type_u) (β : Type_v) [out ring α] := ...
```

- type class mechanism looks for `module _ β _`
- only one canoncial `module` per type
- usually α is fixed per theory anyway
- **Problem:** (multivariate) polynomials

Linear Algebra

```
class module (α : out Type_u) (β : Type_v) [out ring α] := ...
```

- type class mechanism looks for `module _ β _`
- only one canoncial `module` per type
- usually α is fixed per theory anyway
- **Problem:** (multivariate) polynomials

Constructions: Subspace, Linear maps, Quotient, Product

Linear Algebra

$\texttt{class module}\ (\alpha : \texttt{out Type}_u)\ (\beta : \texttt{Type}_v)\ [\texttt{out ring}\ \alpha] := \ldots$

- type class mechanism looks for `module` _ β _
- only one canoncial `module` per type
- usually α is fixed per theory anyway
- **Problem:** (multivariate) polynomials

Constructions: Subspace, Linear maps, Quotient, Product

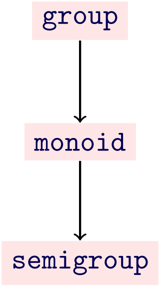
Example

Isomorphism laws:

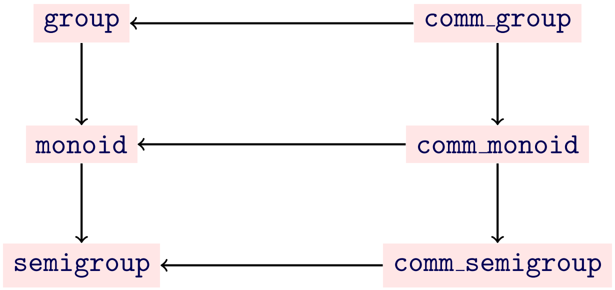
$$\frac{dom(f)}{ker(f)} \simeq_\ell im(f) \qquad \frac{s}{s \cap t} \simeq_\ell \frac{s \oplus t}{t}$$

Discussion

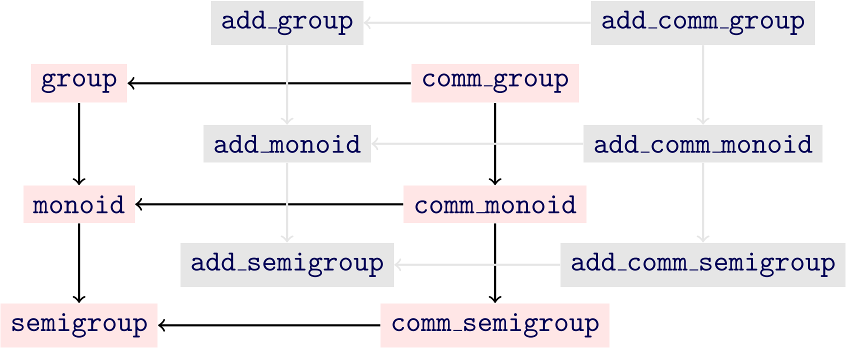
Problems with Type Classes



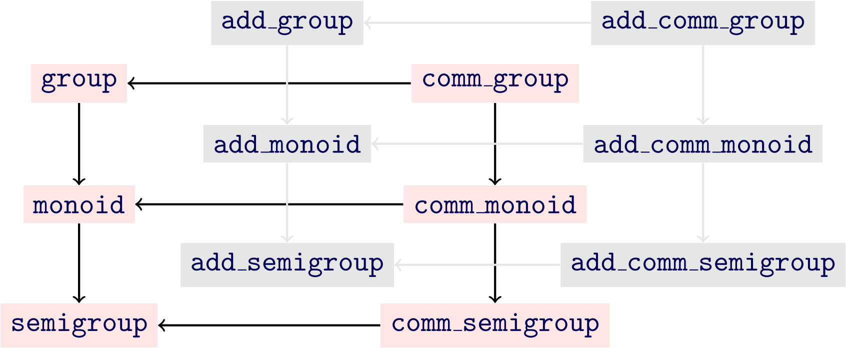
Problems with Type Classes



Problems with Type Classes



Problems with Type Classes



- Currently a automated copy from `group` to `add_group` instead: [*is_group*$(*)(/)(\square^{-1})1$] and [*is_group*$(+)(-)(-\square)0$]
- Mixin type classes
  replace `comm_monoid`, ... by [*is_commutative* $(*)$]

Problem with Universes

```
class functor (M : Type u → Type v) :=
(map : ∀(α β : Type u), (α → β) → M α → M β)
(map_comp : ∀(α β γ : Type u) f g h, map f ∘ map g = map (f ∘ g))
(map_id : ∀α, map id = id)
```

Problem with Universes

Problematic u

$\texttt{class functor}\ (M : \texttt{Type}\ u \to \texttt{Type}\ v) :=$
$(\texttt{map} : \forall (\alpha\ \beta : \texttt{Type}\ u), (\alpha \to \beta) \to M\ \alpha \to M\ \beta)$
$(\texttt{map\_comp} : \forall (\alpha\ \beta\ \gamma : \texttt{Type}\ u)\ f\ g\ h, \texttt{map}\ f \circ \texttt{map}\ g = \texttt{map}\ (f \circ g))$
$(\texttt{map\_id} : \forall \alpha, \texttt{map}\ \texttt{id} = \texttt{id})$

Problem with Universes

Problematic u

$$
\begin{aligned}
&\texttt{class functor } (M : \texttt{Type } u \to \texttt{Type } v) := \\
&(\texttt{map} : \forall (\alpha\ \beta : \texttt{Type } u), (\alpha \to \beta) \to M\ \alpha \to M\ \beta) \\
&(\texttt{map\_comp} : \forall (\alpha\ \beta\ \gamma : \texttt{Type } u)\ f\ g\ h, \texttt{map } f \circ \texttt{map } g = \texttt{map } (f \circ g)) \\
&(\texttt{map\_id} : \forall \alpha, \texttt{map id} = \texttt{id})
\end{aligned}
$$

If we only work with `functor` (`topology` α) our library is too limited, e.g. `topology.map` allows mapping between different universes.

Maintenance

- Currently maintained by Mario Carneiro, me, and Jeremy Avigad
- Contributors:

  Andrew Zipperer, Floris van Doorn, Haitao Zhang, Jeremy Avigad, Johannes Hölzl, Kenny Lau, Kevin Buzzard, Leonardo de Moura, Mario Carneiro, Minchao Wu, Nathaniel Thomas, Parikshit Khanna, Robert Y. Lewis, Simon Hudon
- Currently $\sim$ 51.000 lines of Lean code

mathlib

A (classical) mathematical library for Lean

https://github.com/leanprover/mathlib