# Protop - Dependent Types through Topoi

Regensburg Haskell Meetup

Dr. Lars Brünjes, IOHK

2018-09-20

INPUT | OUTPUT

# Motivation

- Using *dependent types*, it is possible to model large parts of mathematics in a computational way.

# Motivation

- Using *dependent types*, it is possible to model large parts of mathematics in a computational way.
- However, classical mathematics ist mostly formulated using set theory, not type theory.

# Motivation

- Using *dependent types*, it is possible to model large parts of mathematics in a computational way.
- However, classical mathematics ist mostly formulated using set theory, not type theory.
- Many mathematical constructions use set comprehension, for example, defining a set as a subset of elements with a specified property.

# Motivation

- Using *dependent types*, it is possible to model large parts of mathematics in a computational way.
- However, classical mathematics ist mostly formulated using set theory, not type theory.
- Many mathematical constructions use set comprehension, for example, defining a set as a subset of elements with a specified property.
- It would be nice to be able to model mathematics in a style closer to what mathematicians are used to.

# Motivation

- Using *dependent types*, it is possible to model large parts of mathematics in a computational way.
- However, classical mathematics ist mostly formulated using set theory, not type theory.
- Many mathematical constructions use set comprehension, for example, defining a set as a subset of elements with a specified property.
- It would be nice to be able to model mathematics in a style closer to what mathematicians are used to.
- Many set-theoretic constructions can be done in any elementary topos.

# Elementary Topoi

# Topos

An elementary topos is a category $\mathcal{T}$ with the following properties:

- $\mathcal{T}$ has finite limits.
- $\mathcal{T}$ has exponentials.
- $\mathcal{T}$ has a subobject classifier.

# Topos

An elementary topos is a category $\mathcal{T}$ with the following properties:

- $\mathcal{T}$ has finite limits.
  - $\mathcal{T}$ has a terminal object.
  - $\mathcal{T}$ has (finite) products.
  - $\mathcal{T}$ has equalizers.
- $\mathcal{T}$ has exponentials.
- $\mathcal{T}$ has a subobject classifier.

# Equalizers

Let $f, g : X \to Y$ be two morphisms in a category $\mathcal{C}$. Then an equalizer of $f$ and $g$ in $\mathcal{C}$ is an object $\mathrm{Eq}_{f,g}$ in $\mathcal{C}$, together with a morphism $\mathrm{eq}_{f,g} : \mathrm{Eq}_{f,g} \to X$, which is universal for the property $f \circ \mathrm{eq}_{f,g} = g \circ \mathrm{eq}_{f,g}$:

$$\begin{tikzcd}
T \arrow[d, "\exists!"'] \arrow[dr] & & \\
\mathrm{Eq}_{f,g} \arrow[r, "\mathrm{eq}_{f,g}"'] & X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & Y
\end{tikzcd}$$

# Equalizers

Let $f, g : X \to Y$ be two morphisms in a category $\mathcal{C}$. Then an equalizer of $f$ and $g$ in $\mathcal{C}$ is an object $\mathrm{Eq}_{f,g}$ in $\mathcal{C}$, together with a morphism $\mathrm{eq}_{f,g} : \mathrm{Eq}_{f,g} \to X$, which is universal for the property $f \circ \mathrm{eq}_{f,g} = g \circ \mathrm{eq}_{f,g}$:

$$\begin{tikzcd}
T \arrow[d, "\exists!"'] \arrow[drr] & & & \\
\mathrm{Eq}_{f,g} \arrow[rr, "\mathrm{eq}_{f,g}"'] & & X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & Y
\end{tikzcd}$$

In $\underline{\mathrm{Set}}$, the equalizer of two functions $f, g : M \to N$ is

$$\mathrm{Eq}_{f,g} := \left\{ m \in M \;\middle|\; f(m) = g(m) \in N \right\}.$$

# Monomorphisms

In any category, a morphism $f : X \to Y$ is a monmorphism if for each pair of morphisms $i, j : T \to X$ with $f \circ i = f \circ j$, we have $i = j$:

$$
\begin{tikzcd}
T \arrow[r, shift left, "i"] \arrow[r, shift right, "j"'] \arrow[rr, bend left, "f \circ i"] \arrow[rr, bend right, "f \circ j"'] & X \arrow[r, "f"] & Y
\end{tikzcd}
$$

In $\underline{\mathrm{Set}}$, monomorphisms are exactly the injective functions, i.e. functions $f$ with $\forall x, y \in X : f(x) = f(y) \Rightarrow x = y$.

# Elementary Characterization of Monomorphisms

In any category with fibre products, there is this nice equivalent characterization of monomorphisms: A morphism $f : X \to Y$ is a monomorphism iff the following diagram commutes:

$$\begin{tikzcd}
X \times_Y X \arrow[r, "pr_2"] \arrow[d, "pr_1"'] & X \arrow[d, "f"] \\
X \arrow[r, "f"'] \arrow[ur, equal] & Y
\end{tikzcd}$$

# Elementary Characterization of Monomorphisms

In any category with fibre products, there is this nice equivalent characterization of monomorphisms: A morphism $f : X \to Y$ is a monomorphism iff the following diagram commutes:

$$\begin{tikzcd}
X \times_Y X \arrow[r, "pr_2"] \arrow[d, "pr_1"'] & X \arrow[d, "f"] \arrow[dl, equal] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

In $\underline{\mathrm{Set}}$:

$$f(x_1) = f(x_2) \Rightarrow (x_1, x_2) \in X \times_Y X \Rightarrow x_1 = x_2.$$

# Elementary Characterization of Monomorphisms

In any category with fibre products, there is this nice equivalent characterization of monomorphisms: A morphism $f : X \to Y$ is a monomorphism iff the following diagram commutes:

$$\begin{tikzcd}
X \times_Y X \arrow[r, "pr_2"] \arrow[d, "pr_1"'] & X \arrow[d, "f"] \arrow[dl, equal] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

In $\underline{\mathrm{Set}}$:

$$f(x_1) = f(x_2) \Rightarrow (x_1, x_2) \in X \times_Y X \Rightarrow x_1 = x_2.$$

If it wasn't for this, being a monomorphism would be "rank 2" in Haskell.

# Subobject classifiers

Let $\mathcal{C}$ be a category with terminal object $*$. An object $\Omega$ in $\mathcal{C}$, together with a morphism $\mathrm{true} : * \hookrightarrow \Omega$ is a subobject classifier if for all monomorphisms $m : Y \hookrightarrow X$, there is a unique morphism $\chi_m$, such that the following diagram is Cartesian:

$$\begin{tikzcd}
Y \arrow[r, "!_Y"] \arrow[d, hook, "m"'] \arrow[dr, phantom, "\square"] & * \arrow[d, hook, "\mathrm{true}"] \\
X \arrow[r, red, "\exists ! \chi_m"'] & \Omega
\end{tikzcd}$$

This means that $\mathrm{true} : * \hookrightarrow \Omega$ is an universal subobject – all subobjects in $\mathcal{C}$ are pullbacks of $\mathrm{true}$.

# Subobject classifiers

Let $\mathcal{C}$ be a category with terminal object $*$. An object $\Omega$ in $\mathcal{C}$, together with a morphism $\mathrm{true} : * \hookrightarrow \Omega$ is a subobject classifier if for all monomorphisms $m : Y \hookrightarrow X$, there is a unique morphism $\chi_m$, such that the following diagram is Cartesian:

$$\begin{tikzcd} Y \arrow[r, "!_Y"] \arrow[d, hook, "m"'] \arrow[dr, phantom, "\square"] & * \arrow[d, hook, "\mathrm{true}"] \\ X \arrow[r, "\exists! \chi_m"'] & \Omega \end{tikzcd}$$

In $\underline{\mathrm{Set}}$, the subobject classifier is

$$\mathrm{true} : \{\mathrm{true}\} \longrightarrow \mathbb{B} := \{\mathrm{true}, \mathrm{false}\}.$$

# Subobject classifiers

Let $\mathcal{C}$ be a category with terminal object $*$. An object $\Omega$ in $\mathcal{C}$, together with a morphism $\mathrm{true} : * \hookrightarrow \Omega$ is a subobject classifier if for all monomorphisms $m : Y \hookrightarrow X$, there is a unique morphism $\chi_m$, such that the following diagram is Cartesian:

$$\begin{tikzcd}
Y \arrow[r, "!_Y"] \arrow[d, hook, "m"'] \arrow[dr, phantom, "\square"] & * \arrow[d, hook, "\mathrm{true}"] \\
X \arrow[r, red, "\exists! \chi_m"'] & \Omega
\end{tikzcd}$$

Subobject classifiers are closely related to the comprehension axiom, allowing definitions like

$$M := \{n \in N \mid P(n)\}.$$

# The topos of sets

The category $\underline{\mathrm{Set}}$ of sets (and total functions) is an elementary topos:

- $\underline{\mathrm{Set}}$ has finite limits:
    - Any singleton set $\{*\}$ is a terminal object.
    - The product of sets $M$ and $N$ is the Cartesian product $M \times N$.
    - The equalizer of two functions $f, g : M \to N$ is the set

$$\mathrm{Eq}_{f,g} := \left\{ m \in M \;\middle|\; f(m) = g(m) \in N \right\}.$$

- Exponentials are sets of functions

$$N^M := \left\{ f : M \to N \right\}.$$

- Each two-element set, for example the set of Booleans

$$\mathbb{B} := \left\{ \mathrm{true}, \mathrm{false} \right\}$$

  is a subobject classifier.

# $\underline{\mathrm{Hask}}$ is not a topos

The category $\underline{\mathrm{Hask}}$ of Haskell types and (total) functions is not a topos:

- $\underline{\mathrm{Hask}}$ does not have (all) finite limits:
  - `()` is a terminal object in $\underline{\mathrm{Hask}}$.
  - The product of types `a` and `b` is `(a, b)`.
  - $\underline{\mathrm{Hask}}$ does not have arbitrary equalizers.
- Exponentials are function types `a -> b`.
- $\underline{\mathrm{Hask}}$ does not have a subobject classifier.

# Natural numbers

The category with one object $*$ and one morphism $1_*$ is a topos. In order to avoid boring cases like this, we will always consider topoi with a natural number object:

The category with one object $*$ and one morphism $1_*$ is a topos. In order to avoid boring cases like this, we will always consider topoi with a natural number object:

Let $\mathcal{C}$ be a category with final object $*$. A natural number object in $\mathcal{C}$ is a universal diagram $* \xrightarrow{\mathrm{zero}} \mathbb{N} \xrightarrow{\mathrm{succ}} \mathbb{N}$ in $\mathcal{C}$:

$$\begin{tikzcd}
* \arrow[r, "z"] & X \arrow[r, "s"] & X \\
* \arrow[u, equal] \arrow[r, "\mathrm{zero}"'] & \mathbb{N} \arrow[u, "\exists!", "\mathrm{rec}_{z,s}"'] \arrow[r, "\mathrm{succ}"'] & \mathbb{N} \arrow[u, "\mathrm{rec}_{z,s}"']
\end{tikzcd}$$

# Protop

# The goal

In order to do “computational mathematics”, we want to achieve two things:

# The goal

In order to do “computational mathematics”, we want to achieve two things:

- Syntactically, we want to use and construct objects and morphism that are defined in every topos with natural number object, i.e. that only use the axioms of elementary topoi and natural number objects.

# The goal

In order to do “computational mathematics”, we want to achieve two things:

- Syntactically, we want to use and construct objects and morphism that are defined in every topos with natural number object, i.e. that only use the axioms of elementary topoi and natural number objects.
- We want to interpret the objects and morphisms thus constructed in a model that allows us to actually compute results.

# The goal

In order to do “computational mathematics”, we want to achieve two things:

- Syntactically, we want to use and construct objects and morphism that are defined in every topos with natural number object, i.e. that only use the axioms of elementary topoi and natural number objects.
- We want to interpret the objects and morphisms thus constructed in a model that allows us to actually compute results.

If successful, we can perform many of the constructions of classical mathematics in a constructive, computational way.

# The problem

In a topos $\mathcal{T}$ with natural number object $\mathbb{N}$, consider the following situation as an example:

$$
\begin{tikzcd}
\{42\} \arrow[drr, bend left, "!_{\{42\}}"] \arrow[dr, red, "42 \mapsto 41"] \arrow[ddr, hook, bend right] & & \\
& \mathbb{N} \arrow[r, "!_{\mathbb{N}}"] \arrow[d, hook, "\mathrm{succ}"'] \arrow[dr, phantom, "\square"] & * \arrow[d, hook, "\mathrm{true}"] \\
& \mathbb{N} \arrow[r, "\chi_{\mathrm{succ}}"'] & \Omega
\end{tikzcd}
$$

If we can prove that the outer diagram commutes, then the red morphism must exist.

# The problem

In a topos $\mathcal{T}$ with natural number object $\mathbb{N}$, consider the following situation as an example:

$$\begin{tikzcd}
\{42\} \arrow[dr, red, "42\mapsto 41"] \arrow[drr, bend left, "!_{\{42\}}"] \arrow[ddr, hook, bend right] & & \\
& \mathbb{N} \arrow[r, "!_{\mathbb{N}}"] \arrow[d, hook, "\mathrm{succ}"'] \arrow[dr, phantom, "\square"] & * \arrow[d, hook, "\mathrm{true}"] \\
& \mathbb{N} \arrow[r, "\chi_{\mathrm{succ}}"'] & \Omega
\end{tikzcd}$$

If we can prove that the outer diagram commutes, then the red morphism must exist. Computationally, this means that our interpreter must be able to compute the predecessor of 42 in our model.

# The problem

In a topos $\mathcal{T}$ with natural number object $\mathbb{N}$, consider the following situation as an example:

$$
\begin{tikzcd}
\{42\} \arrow[drr, bend left, "!_{\{42\}}"] \arrow[dr, red, "42 \mapsto 41"] \arrow[ddr, hook, bend right] & & \\
& \mathbb{N} \arrow[r, "!_{\mathbb{N}}"] \arrow[d, hook, "\mathrm{succ}"'] \arrow[dr, phantom, "\square"] & * \arrow[d, hook, "\mathrm{true}"] \\
& \mathbb{N} \arrow[r, "\chi_{\mathrm{succ}}"'] & \Omega
\end{tikzcd}
$$

## Note

Equalizers are *not* such a big problem - they could be implemented by simply ignoring the extra information at runtime.

# Key idea

In the example from the last slide, we must first have proven "somehow" that the diagram commutes.

# Key idea

In the example from the last slide, we must first have proven "somehow" that the diagram commutes.

"Somewhere" in that proof, we probably mentioned the number 41.

# Key idea

In the example from the last slide, we must first have proven "somehow" that the diagram commutes.

"Somewhere" in that proof, we probably mentioned the number 41.

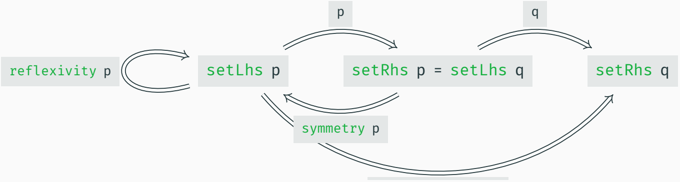
The key idea is to take proofs seriously – make the information contained in proofs explicit and give it computational content.

# Setoids

```haskell
class (Typeable a, Typeable (Proofs a))
   =>  IsSetoid a where

  type Proofs a

  reflexivity  :: a -> Proofs a
  symmetry     :: Proxy a -> Proofs a -> Proofs a
  transitivity :: Proxy a -> Proofs a -> Proofs a
               -> Proofs a
  setLhs       :: Proofs a -> a
  setRhs       :: Proofs a -> a
```



# Functoids

```haskell
data Functoid :: Type -> Type -> Type where
  Functoid :: (IsSetoid a, IsSetoid b)
           => (a -> b)
           -> (Proofs a -> Proofs b)
           -> Functoid a b

onPoints :: Functoid a b -> a -> b
onPoints (Functoid f _) = f

onProofs :: Functoid a b -> Proofs a -> Proofs b
onProofs (Functoid _ g) = g
```

$$
\begin{tikzcd}
\mathtt{x} \arrow[rr, "\mathtt{onPoints}"] \arrow[d, Rightarrow, "\mathtt{p}"'] & & \mathtt{y} \arrow[d, Rightarrow, "\mathtt{q}"] \\
\mathtt{x'} \arrow[rr, "\mathtt{onPoints}"'] & & \mathtt{y'}
\arrow[from=1-1, to=1-3, phantom, shift right=6, "\rightsquigarrow\ \mathtt{onProofs}"]
\end{tikzcd}
$$

# Objects

```haskell
class (Show x
      , Typeable x
      , IsSetoid (Domain x)
      , Singleton x
      ) => IsObject x where

  type Domain x
```

# Morphisms

```haskell
class (Show f
      , Typeable f
      , IsObject (Source f)
      , IsObject (Target f)
      , Singleton f
      ) => IsMorphism f where
  type Source f
  type Target f
  onDomains :: f -> Functoid (DSource f) (DTarget f)

type DSource f = Domain (Source f)
type DTarget f = Domain (Target f)
type PSource f = Proofs (DSource f)
type PTarget f = Proofs (DTarget f)
```

# Proofs

```haskell
class (Show p
      , Typeable p
      , IsMorphism (Lhs p)
      , IsMorphism (Rhs p)
      , Source (Lhs p) ~ Source (Rhs p)
      , Target (Lhs p) ~ Target (Rhs p)
      , Singleton p
      ) => IsProof p where
  type Lhs p
  type Rhs p
  proof :: p
        -> Domain (SOURCE p)
        -> Proofs (Domain (TARGET p))

type SOURCE p = Source (Lhs p)
type TARGET p = Target (Lhs p)

lhs :: IsProof p => p -> Lhs p
lhs _ = singleton

rhs :: IsProof p => p -> Rhs p
rhs _ = singleton
```

# Proofs (cntd.)

$$\begin{tikzcd}[column sep=huge] \texttt{SOURCE p} \arrow[r, bend left, "\texttt{lhs p}"{name=U}] \arrow[r, bend right, "\texttt{rhs p}"'{name=D}] & \texttt{TARGET p} \arrow[Rightarrow, from=U, to=D, "\texttt{p}"] \end{tikzcd}$$

$$\begin{tikzcd}[column sep=huge] & \texttt{Domain (TARGET p)} \\ \texttt{Domain (SOURCE p)} \arrow[ur, "\texttt{onPoints\$onDomains\$lhs p}"] \arrow[r, "\texttt{proof p}"] \arrow[dr, "\texttt{onPoints\$onDomains\$rhs p}"'] & \texttt{Proofs (Domain (TARGET p))} \arrow[u, "\texttt{setLhs}"'] \arrow[d, "\texttt{setRhs}"] \\ & \texttt{Domain (TARGET p)} \end{tikzcd}$$

# The Omega domain

```haskell
data OPoint :: Type where
  OPoint :: IsMorphism f => f -> DTarget f -> OPoint

data OProof :: Type where
  OProof :: (IsMorphism f, IsMorphism g)
         => f -> g -> DTarget f -> DTarget g ->
           ((DSource f, PTarget f) -> (DSource g, PTarget g)) ->
           ((DSource g, PTarget g) -> (DSource f, PTarget f)) ->
           OProof
```

$$
\begin{tikzcd}
Y \arrow[d, "f"'] & & y \arrow[d, mapsto] & y' \arrow[d, mapsto] & & Y' \arrow[d, "f'"] \\
X \arrow[r, squiggly, dash] & x & z \arrow[l, Rightarrow] & z' \arrow[r, Rightarrow] & x' \arrow[r, squiggly, dash] & X'
\end{tikzcd}
$$

# The Omega domain

```
data OPoint :: Type where
  OPoint :: IsMorphism f => f -> DTarget f -> OPoint

data OProof :: Type where
  OProof :: (IsMorphism f, IsMorphism g)
         => f -> g -> DTarget f -> DTarget g ->
            ((DSource f, PTarget f) -> (DSource g, PTarget g)) ->
            ((DSource g, PTarget g) -> (DSource f, PTarget f)) ->
            OProof
```

$$
\begin{tikzcd}
Y \arrow[d, "f"'] & & y \arrow[d, mapsto] & * \arrow[d, mapsto] & \text{true} & * \arrow[d, "1_*"] \\
X \arrow[r, squiggly, no head] & x & z \arrow[l, Rightarrow] & * \arrow[r, Leftrightarrow] & * \arrow[r, squiggly, no head] & *
\end{tikzcd}
$$