# Universes of data types in constructive type theory

## Lecture 1: Martin-Löf Type Theory

Fredrik Nordvall Forsberg

University of Strathclyde
https://fredriknf.com/pc22/

Proof and Computation 2022 Autumn School, Fischbachau

# Data types in constructive type theory

Constructive type theory is both a foundational mathematical system and an expressive programming language.

Data types play an important role from both points of view.

Can we study such data types systematically?

For example, what definitions are semantically meaningful? And can we prove theorems or provide constructions for whole classes of data types, rather than one by one?

# Plan

Lecture 1 (now) Introduction to Martin-Löf Type Theory

Lecture 2 (Thursday) Inductive data types

Lecture 3 (Friday) More advanced data types

# Brief and incomplete history leading to type theory

# Brief and incomplete history leading to type theory

- BHK interpretation: informal explanation of what a constructive proof is (Heyting 1934, Kolmogorov 1932)

L.E.J. Brouwer

Arend Heyting

Andrey Kolmogorov

# Brief and incomplete history leading to type theory

- BHK interpretation: informal explanation of what a constructive proof is (Heyting 1934, Kolmogorov 1932)
- Curry-Howard correspondence: constructive propositional logic corresponds precisely to the simply typed lambda calculus/typed combinatory logic (Curry 1958, Howard 1969)

L.E.J. Brouwer

Arend Heyting

Andrey Kolmogorov

Haskell Curry

William Howard

# Brief and incomplete history leading to type theory

- BHK interpretation: informal explanation of what a constructive proof is (Heyting 1934, Kolmogorov 1932)
- Curry-Howard correspondence: constructive propositional logic corresponds precisely to the simply typed lambda calculus/typed combinatory logic (Curry 1958, Howard 1969)
- Constructive Type Theory extends correspondence to predicate logic by introducing dependent types (Martin-Löf 1972)

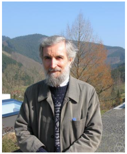

L.E.J. Brouwer


Arend Heyting


Andrey Kolmogorov


Haskell Curry


William Howard


Per Martin-Löf

# Informal vs formal

Constructive type theory is meant to be a foundational system for constructive mathematics.

# Informal vs formal

Constructive type theory is meant to be a foundational system for constructive mathematics.

As such, it is a formal system presented by rules.

However when working *in* type theory, arguments can be presented informally (cf. "working in ZFC").

# Informal vs formal

Constructive type theory is meant to be a foundational system for constructive mathematics.

As such, it is a formal system presented by rules.

However when working *in* type theory, arguments can be presented informally (cf. “working in ZFC”).

Will try to be a little more formal today — one take-away is that there is structure in the rules ready to be exploited.

# Judgements

Fundamental underlying concept: judgements.

$$\text{“context”} \vdash \text{“statement”}$$

# Judgements

Fundamental underlying concept: judgements.

$$\text{“context”} \vdash \text{“statement”}$$

$$n : \mathbb{N}, m : \mathbb{N} \vdash \mathsf{Bool}^{n+m} \text{ type}$$

# Judgements

Fundamental underlying concept: judgements.

$$\text{“context”} \vdash \text{“statement”}$$

$$n : \mathbb{N}, m : \mathbb{N} \vdash \mathsf{Bool}^{n+m} \text{ type}$$

| | |
|---|---|
| $\Gamma$ valid | $\Gamma$ is a well formed context |
| $\Gamma \vdash A$ type | $A$ is a well formed type (in context $\Gamma$) |
| $\Gamma \vdash a : A$ | $a$ is of type $A$ |
| $\Gamma \vdash A \equiv B$ | $A$ and $B$ are the same type |
| $\Gamma \vdash a \equiv a' : A$ | $a$ and $a'$ are the same term (in type $A$) |

# Judgements

Fundamental underlying concept: judgements.

$$\text{“context”} \vdash \text{“statement”}$$

$$n : \mathbb{N}, m : \mathbb{N} \vdash \mathsf{Bool}^{n+m} \text{ type}$$

| | |
|---|---|
| $\Gamma$ valid | $\Gamma$ is a well formed context |
| $\Gamma \vdash A$ type | $A$ is a well formed type (in context $\Gamma$) |
| $\Gamma \vdash a : A$ | $a$ is of type $A$ |
| $\Gamma \vdash A \equiv B$ | $A$ and $B$ are the same type |
| $\Gamma \vdash a \equiv a' : A$ | $a$ and $a'$ are the same term (in type $A$) |

**Convention:** We normally suppress mentioning $\Gamma$, and only show context extensions $x : A \vdash \ldots$.

# Inference rules

Formally type theory is given by a collection of *inference rules*

$$\frac{\mathcal{J}_1 \quad \dots \quad \mathcal{J}_n}{\mathcal{J}}$$

# Inference rules

Formally type theory is given by a collection of *inference rules*

$$\frac{\mathcal{J}_1 \quad \ldots \quad \mathcal{J}_n}{\mathcal{J}}$$

A judgement $\mathcal{J}$ is derivable if we can construct a derivation tree with conclusion $\mathcal{J}$ using the inference rules. For example:

$$\dfrac{\dfrac{}{n : \mathbb{N}, m : \mathbb{N} \vdash \mathsf{Bool}\ \mathsf{type}} \qquad \dfrac{\dfrac{}{n : \mathbb{N}, m : \mathbb{N} \vdash n : \mathbb{N}} \quad \dfrac{}{n : \mathbb{N}, m : \mathbb{N} \vdash m : \mathbb{N}}}{n : \mathbb{N}, m : \mathbb{N} \vdash n + m : \mathbb{N}}}{n : \mathbb{N}, m : \mathbb{N} \vdash \mathsf{Bool}^{n+m}\ \mathsf{type}}$$

# Inference rules

Formally type theory is given by a collection of *inference rules*

$$\frac{\mathcal{J}_1 \quad \dots \quad \mathcal{J}_n}{\mathcal{J}}$$

A judgement $\mathcal{J}$ is derivable if we can construct a derivation tree with conclusion $\mathcal{J}$ using the inference rules. For example:

$$\dfrac{\dfrac{}{n : \mathbb{N}, m : \mathbb{N} \vdash \mathsf{Bool}\ \mathsf{type}} \qquad \dfrac{\dfrac{}{n : \mathbb{N}, m : \mathbb{N} \vdash n : \mathbb{N}} \quad \dfrac{}{n : \mathbb{N}, m : \mathbb{N} \vdash m : \mathbb{N}}}{n : \mathbb{N}, m : \mathbb{N} \vdash n + m : \mathbb{N}}}{n : \mathbb{N}, m : \mathbb{N} \vdash \mathsf{Bool}^{n+m}\ \mathsf{type}}$$

Of course, when working in type theory, we never explicitly construct derivation trees!

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory).

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory). ✗ No, "$17 : \mathbb{N}$" is a judgement, not a statement.

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory). ✗ No, "$17 : \mathbb{N}$" is a judgement, not a statement.
- 17 is even.

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory). ✗ No, "$17 : \mathbb{N}$" is a judgement, not a statement.
- 17 is even. ✓ Yes, would expect "$y : \mathbb{N} \vdash \mathrm{Even}[y]$ type".

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory). ✗ No, "$17 : \mathbb{N}$" is a judgement, not a statement.
- 17 is even. ✓ Yes, would expect "$y : \mathbb{N} \vdash \mathsf{Even}[y]$ type".
- `"hi"` is even.

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory). ✗ No, "$17 : \mathbb{N}$" is a judgement, not a statement.
- 17 is even. ✓ Yes, would expect "$y : \mathbb{N} \vdash \text{Even}[y]$ type".
- `"hi"` is even. ✗ No, statement does not type-check, since `"hi"` is not a natural number.

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory). ✗ No, "$17 : \mathbb{N}$" is a judgement, not a statement.
- 17 is even. ✓ Yes, would expect "$y : \mathbb{N} \vdash \mathrm{Even}[y]$ type".
- `"hi"` is even. ✗ No, statement does not type-check, since `"hi"` is not a natural number.
- $\pi \in \cos$ (in set theory).

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory). ✗ No, "$17 : \mathbb{N}$" is a judgement, not a statement.
- 17 is even. ✓ Yes, would expect "$y : \mathbb{N} \vdash \mathrm{Even}[y]$ type".
- `"hi"` is even. ✗ No, statement does not type-check, since `"hi"` is not a natural number.
- $\pi \in \cos$ (in set theory). ✓ Yes.

# Caveats

Judgements are "external" — they cannot be proven inside the language.

In contrast, "$a \in A$" in set theory is an "internal" statement.

**Quiz: what makes sense to prove or disprove?**

- 17 is a natural number (in type theory). ✗ No, "$17 : \mathbb{N}$" is a judgement, not a statement.
- 17 is even. ✓ Yes, would expect "$y : \mathbb{N} \vdash \mathrm{Even}[y]$ type".
- `"hi"` is even. ✗ No, statement does not type-check, since `"hi"` is not a natural number.
- $\pi \in \cos$ (in set theory). ✓ Yes.

Note that $A \equiv B$ and $a \equiv a' : A$ also are "external" statements; we will see an internal version that can be (dis)proven later.

# Basic rules

Variables:

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A}$$

Conversion:

$$\frac{t : A \qquad A \equiv B}{t : B}$$

Judgemental equality:

$$\frac{t : A}{t \equiv t : A} \qquad \frac{t \equiv s : A}{s \equiv t : A} \qquad \frac{s \equiv t : A \qquad t \equiv u : A}{s \equiv u : A}$$

Congruence rules: for example

$$\frac{A \equiv A' \qquad B \equiv B'}{(A \to B) \equiv (A' \to B')}$$

(many more!)

# A pattern for introducing types

A type is usually given by four (five) groups of rules:

**Formation** What is needed to construct the type?

**Introduction** What is needed to construct canonical elements of the type?

**Elimination** How can elements of the type be used?

**Computation** What happens when you eliminate canonical elements? (“$\beta$-rules”)

**Uniqueness (sometimes)** How are functions into or out of the type determined? (“$\eta$-rules”)

# Pair types

Formation

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}}$$

Introduction

$$\frac{a : A \qquad b : B}{(a, b) : A \times B}$$

Elimination

$$\frac{p : A \times B}{\mathsf{fst}\, p : A} \qquad \frac{p : A \times B}{\mathsf{snd}\, p : B}$$

Computation $\mathsf{fst}\,(a, b) \equiv a : A$ and $\mathsf{snd}\,(a, b) \equiv b : B$.

Uniqueness

$$\frac{p : A \times B}{p \equiv (\mathsf{fst}(p), \mathsf{snd}(p)) : A \times B}$$

# Pair types: alternative elimination and computation rules

Elimination

$$\frac{p : A \times B}{\mathsf{fst}\, p : A} \qquad \frac{p : A \times B}{\mathsf{snd}\, p : B}$$

Computation $\mathsf{fst}\,(a, b) \equiv a : A$ and $\mathsf{snd}\,(a, b) \equiv b : B$.

Uniqueness

$$\frac{p : A \times B}{p \equiv (\mathsf{fst}(p), \mathsf{snd}(p)) : A \times B}$$

# Pair types: alternative elimination and computation rules

Elimination

$$\frac{p : A \times B}{\mathsf{fst}\, p : A} \qquad \frac{p : A \times B}{\mathsf{snd}\, p : B}$$

Computation $\mathsf{fst}\,(a, b) \equiv a : A$ and $\mathsf{snd}\,(a, b) \equiv b : B$.

Uniqueness

$$\frac{p : A \times B}{p \equiv (\mathsf{fst}(p), \mathsf{snd}(p)) : A \times B}$$

Alternatively:

Elimination'

$$\frac{z : A \times B \vdash C \text{ type} \quad x : A, y : B \vdash c : C[z \mapsto (x, y)] \quad p : A \times B}{\mathsf{elim}_\times(C, c, p) : C[z \mapsto p]}$$

Computation'

$$\mathsf{elim}_\times(C, c, (a, b)) \equiv c[x \mapsto a, y \mapsto b] : C[z \mapsto (a, b)].$$

# Pair types: alternative elimination and computation rules

Elimination

$$\frac{p : A \times B}{\mathsf{fst}\, p : A} \qquad \frac{p : A \times B}{\mathsf{snd}\, p : B}$$

Computation $\mathsf{fst}\,(a, b) \equiv a : A$ and $\mathsf{snd}\,(a, b) \equiv b : B$.

Uniqueness

$$\frac{p : A \times B}{p \equiv (\mathsf{fst}(p), \mathsf{snd}(p)) : A \times B}$$

Alternatively:

Elimination'

$$\frac{z : A \times B \vdash C \text{ type} \quad x : A, y : B \vdash c : C[z \mapsto (x, y)] \quad p : A \times B}{\mathsf{elim}_{\times}(C, c, p) : C[z \mapsto p]}$$

Computation'

$$\mathsf{elim}_{\times}(C, c, (a, b)) \equiv c[x \mapsto a, y \mapsto b] : C[z \mapsto (a, b)].$$

Exercise

Show that E + C follows from E' + C', and that E' + C' follows from E + C + Uniqueness. Does Uniqueness follow from E' + C'?

# Function types

Formation

$$\frac{A \text{ type} \qquad B \text{ type}}{A \to B \text{ type}}$$

Introduction

$$\frac{x : A \vdash t : B}{\lambda(x : A).t : A \to B}$$

Elimination

$$\frac{f : A \to B \qquad a : A}{f\, a : B}$$

Computation $(\lambda(x : A).t)\, a \equiv t[x \mapsto a] : B$

Uniqueness

$$\frac{f : A \to B}{f \equiv (\lambda(x : A).f\, x) : A \to B}$$

# Example: swap function

Given types $A$ and $B$, let us write swap : $A \times B \to B \times A$.

$$\mathsf{swap} : A \times B \to B \times A$$
$$\mathsf{swap} = \boxed{?_0 : A \times B \to B \times A}$$

# Example: swap function

Given types $A$ and $B$, let us write $\mathsf{swap} : A \times B \to B \times A$.

$$\mathsf{swap} : A \times B \to B \times A$$
$$\mathsf{swap} = \lambda(x : A \times B).\ ?_1 : B \times A$$

# Example: swap function

Given types $A$ and $B$, let us write $\mathsf{swap} : A \times B \to B \times A$.

$$\mathsf{swap} : A \times B \to B \times A$$

$$\mathsf{swap} = \lambda(x : A \times B).(\boxed{?_2 : B}, \boxed{?_3 : A})$$

# Example: swap function

Given types $A$ and $B$, let us write $\mathsf{swap} : A \times B \to B \times A$.

$$\mathsf{swap} : A \times B \to B \times A$$
$$\mathsf{swap} = \lambda(x : A \times B).(\mathsf{snd}\, x, \; ?_3 : A \;)$$

# Example: swap function

Given types $A$ and $B$, let us write $\mathsf{swap} : A \times B \to B \times A$.

$$
\begin{aligned}
&\mathsf{swap} : A \times B \to B \times A \\
&\mathsf{swap} = \lambda(x : A \times B).(\mathsf{snd}\, x, \mathsf{fst}\, x)
\end{aligned}
$$

# Example: swap function

Given types $A$ and $B$, let us write $\mathsf{swap} : A \times B \to B \times A$.

$$\begin{aligned}
&\mathsf{swap} : A \times B \to B \times A \\
&\mathsf{swap} = \lambda(x : A \times B).(\mathsf{snd}\, x, \mathsf{fst}\, x)
\end{aligned}$$

$$\dfrac{\dfrac{\dfrac{}{x : A \times B \vdash x : A \times B}}{x : A \times B \vdash \mathsf{snd}\, x : B} \qquad \dfrac{\dfrac{}{x : A \times B \vdash x : A \times B}}{x : A \times B \vdash \mathsf{fst}\, x : A}}{\dfrac{x : A \times B \vdash (\mathsf{snd}\, x, \mathsf{fst}\, x) : B \times A}{\vdash \lambda(x : A \times B).(\mathsf{snd}\, x, \mathsf{fst}\, x) : A \times B \to B \times A}}$$

# The empty type

Formation

$$\frac{}{\mathbf{0}\ \mathsf{type}}$$

Introduction (none)

Elimination

$$\frac{C\ \mathsf{type} \qquad p : \mathbf{0}}{\mathsf{elim}_{\mathbf{0}}(C, p) : C}$$

Computation (none)

**Exercise**

Prove a dependent elimination rule from the non-dependent one:

$$\frac{z : \mathbf{0} \vdash C\ \mathsf{type} \qquad p : \mathbf{0}}{\mathsf{elim}_{\mathbf{0}}(C, p) : C[z \mapsto p]}$$

# The unit type

Formation

$$\frac{}{\mathbf{1}\ \mathsf{type}}$$

Introduction $\star : \mathbf{1}$.

Elimination (none)

Computation (none)

Uniqueness

$$\frac{u : \mathbf{1}}{u \equiv \star : \mathbf{1}}$$

**Exercise**

Formulate and prove elimination and computation rules.

# Disjoint union type

Formation

$$\frac{A \text{ type} \quad B \text{ type}}{A + B \text{ type}}$$

Introduction

$$\frac{a : A}{\mathsf{inl}\, a : A + B} \qquad \frac{b : B}{\mathsf{inr}\, b : A + B}$$

Elimination

$$\frac{z : A + B \vdash C \text{ type} \quad \begin{array}{c} x : A \vdash c : C[z \mapsto \mathsf{inl}\, x] \\ y : B \vdash d : C[z \mapsto \mathsf{inr}\, y] \end{array} \quad s : A + B}{\mathsf{elim}_+(C, c, d, s) : C[z \mapsto s]}$$

Computation

$$\mathsf{elim}_+(C, c, d, \mathsf{inl}\, a) \equiv c[x \mapsto a] : C[z \mapsto \mathsf{inl}\, a]$$
$$\mathsf{elim}_+(C, c, d, \mathsf{inr}\, b) \equiv d[y \mapsto b] : C[z \mapsto \mathsf{inr}\, b]$$

Exercise

Define $\mathrm{Bool} = \mathbf{1} + \mathbf{1}$, and formulate and prove its rules.

# Dependent function types

Formation

$$\frac{A \text{ type} \qquad x : A \vdash B \text{ type}}{(x : A) \to B \text{ type}}$$

Introduction

$$\frac{x : A \vdash t : B}{\lambda(x : A).t : (x : A) \to B}$$

Elimination

$$\frac{f : (x : A) \to B \qquad a : A}{f\, a : B[x \mapsto a]}$$

Computation $(\lambda(x : A).t)\, a \equiv t[x \mapsto a] : B[x \mapsto a]$

Uniqueness

$$\frac{f : (x : A) \to B}{f \equiv (\lambda(x : A).f\, x) : (x : A) \to B}$$

$A \to B$ is the special case when $B$ does not depend on $x : A$.

# Dependent pair types

Formation

$$\frac{A \text{ type} \qquad x : A \vdash B \text{ type}}{(x : A) \times B \text{ type}}$$

Introduction

$$\frac{a : A \qquad b : B[x \mapsto a]}{(a, b) : (x : A) \times B}$$

Elimination

$$\frac{p : (x : A) \times B}{\mathsf{fst}\, p : A} \qquad \frac{p : (x : A) \times B}{\mathsf{snd}\, p : B[x \mapsto \mathsf{fst}\, p]}$$

Computation $\mathsf{fst}\,(a, b) \equiv a : A$ and $\mathsf{snd}\,(a, b) \equiv b : B[x \mapsto a]$.

Uniqueness

$$\frac{p : (x : A) \times B}{p \equiv (\mathsf{fst}(p), \mathsf{snd}(p)) : (x : A) \times B}$$

$A \times B$ is the special case when $B$ does not depend on $x : A$.

# Example: The type-theoretic "Theorem of Choice"

Assume $A$ type, $B$ type and $x : A, y : B \vdash R$ type.

$$\mathsf{ac} : \Big((x : A) \to \big((y : B) \times R[x, y]\big)\Big) \to \Big((f : A \to B) \times \big((x : A) \to R[x, f\,x]\big)\Big)$$

$$\mathsf{ac}\,g = \boxed{?_0 : (f : A \to B) \times \big((x : A) \to R[x, f\,x]\big)}$$

# Example: The type-theoretic "Theorem of Choice"

Assume $A$ type, $B$ type and $x : A, y : B \vdash R$ type.

$$\mathsf{ac} : \Big((x : A) \to \big((y : B) \times R[x, y]\big)\Big) \to \Big((f : A \to B) \times \big((x : A) \to R[x, f\,x]\big)\Big)$$

$$\mathsf{ac}\, g = (\ \boxed{?_1 : A \to B}\ ,\ \boxed{?_2 : (x : A) \to R[x, ?_1\, x]}\ )$$

# Example: The type-theoretic "Theorem of Choice"

Assume $A$ type, $B$ type and $x : A, y : B \vdash R$ type.

$$\mathsf{ac} : \Big((x : A) \to \big((y : B) \times R[x, y]\big)\Big) \to \Big((f : A \to B) \times \big((x : A) \to R[x, f\,x]\big)\Big)$$

$$\mathsf{ac}\,g = (\lambda(x : A).\ ?_3 : B\ ,\ ?_2 : (x : A) \to R[x, ?_1\,x]\ )$$

# Example: The type-theoretic "Theorem of Choice"

Assume $A$ type, $B$ type and $x : A, y : B \vdash R$ type.

$$\mathsf{ac} : \Big((x : A) \to \big((y : B) \times R[x, y]\big)\Big) \to \Big((f : A \to B) \times \big((x : A) \to R[x, f\,x]\big)\Big)$$

$$\mathsf{ac}\,g = (\lambda(x : A).\mathsf{fst}\,(g\,x),\ \boxed{?_2 : (x : A) \to R[x, \mathsf{fst}\,(g\,x)]}\,)$$

# Example: The type-theoretic "Theorem of Choice"

Assume $A$ type, $B$ type and $x : A, y : B \vdash R$ type.

$$\mathsf{ac} : \Big((x : A) \to \big((y : B) \times R[x, y]\big)\Big) \to \Big((f : A \to B) \times \big((x : A) \to R[x, f\,x]\big)\Big)$$

$$\mathsf{ac}\,g = (\lambda(x : A).\mathsf{fst}\,(g\,x), \lambda(x : A).\, ?_4 : R[x, \mathsf{fst}\,(g\,x)]\ )$$

# Example: The type-theoretic "Theorem of Choice"

Assume $A$ type, $B$ type and $x : A, y : B \vdash R$ type.

$$\mathsf{ac} : \Big((x : A) \to ((y : B) \times R[x, y])\Big) \to \Big((f : A \to B) \times ((x : A) \to R[x, f\,x])\Big)$$

$$\mathsf{ac}\,g = (\lambda(x : A).\mathsf{fst}\,(g\,x), \lambda(x : A).\mathsf{snd}\,(g\,x))$$

# Example: The type-theoretic "Theorem of Choice"

Assume $A$ type, $B$ type and $x : A, y : B \vdash R$ type.

$$\mathsf{ac} : \Big((x : A) \to ((y : B) \times R[x, y])\Big) \to \Big((f : A \to B) \times ((x : A) \to R[x, f\,x])\Big)$$

$$\mathsf{ac}\, g = (\lambda(x : A).\mathsf{fst}\,(g\,x), \lambda(x : A).\mathsf{snd}\,(g\,x))$$

*A choice function exists in constructive mathematics, because a choice is implied by the very meaning of existence.*
*— Bishop 1967*

# Example: The type-theoretic "Theorem of Choice"

Assume $A$ type, $B$ type and $x : A, y : B \vdash R$ type.

$$\mathsf{ac} : \Big((x : A) \to ((y : B) \times R[x, y])\Big) \to \Big((f : A \to B) \times ((x : A) \to R[x, f\,x])\Big)$$
$$\mathsf{ac}\,g = (\lambda(x : A).\mathsf{fst}\,(g\,x), \lambda(x : A).\mathsf{snd}\,(g\,x))$$

> *A choice function exists in constructive mathematics, because a choice is implied by the very meaning of existence.*
> *— Bishop 1967*

However: does not work for "truncated pair types", or setoids.

# Universes

A type $U$ of types.

**"À la Russel":**

$$\frac{A : U}{A \text{ type}}$$

**"À la Tarski":**

$$\frac{A : U}{T(A) \text{ type}}$$

$U$ contains "codes" for the types we are interested in. Allows computing types from data ("large elimination"), by computing a code in the universe instead.

A universe Type also allows abuse of notation "$P : A \to \text{Type}$" for "$x : A \vdash P \text{ type}$".

# Natural numbers

Formation

$$\frac{}{\mathbb{N}\ \mathsf{type}}$$

Introduction

$$\frac{}{0 : \mathbb{N}} \qquad \frac{n : \mathbb{N}}{\mathsf{suc}\, n : \mathbb{N}}$$

Elimination

$$\frac{z : \mathbb{N} \vdash C\ \mathsf{type} \quad \begin{array}{c} c : C[z \mapsto 0] \\ x : \mathbb{N}, \bar{x} : C[z \mapsto x] \vdash d : C[z \mapsto \mathsf{suc}\, x] \end{array} \quad n : \mathbb{N}}{\mathsf{elim}_{\mathbb{N}}(C, c, d, n) : C[z \mapsto n]}$$

Computation

$$\mathsf{elim}_{\mathbb{N}}(C, c, d, 0) \equiv c : C[z \mapsto 0]$$
$$\mathsf{elim}_{\mathbb{N}}(C, c, d, \mathsf{suc}\, n) \equiv d[x \mapsto n, \bar{x} \mapsto \mathsf{elim}_{\mathbb{N}}(C, c, d, n)] : C[z \mapsto \mathsf{suc}\, n]$$

# Lists

## Formation

$$\frac{A \text{ type}}{\text{List } A \text{ type}}$$

## Introduction

$$\frac{}{[] : \text{List } A} \qquad \frac{a : A \qquad as : \text{List } A}{(a :: as) : \text{List } A}$$

## Elimination

$$\frac{\begin{array}{c} as : \text{List } A \\ z : \text{List } A \vdash C \text{ type} \end{array} \qquad \begin{array}{c} c : C[z \mapsto []] \\ xs : \text{List } A, \bar{xs} : C[z \mapsto xs] \vdash d : C[z \mapsto x :: xs] \end{array}}{\text{elim}_{\text{List}}(C, c, d, as) : C[z \mapsto as]}$$

## Computation

$$\text{elim}_{\text{List}}(C, c, d, []) \equiv c : C[z \mapsto []]$$

$$\text{elim}_{\text{List}}(C, c, d, a :: as) \equiv d[xs \mapsto as, \bar{xs} \mapsto \text{elim}_{\text{List}}(C, c, d, as)] : C[a :: as]$$

# Propositions as types

| **A proof of** | **. . . is, according to BHK. . .** |
|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ |
| $A \to B$ | a way to prove $A$ given a proof of $B$ |
| $\top$ | always has a proof |
| $\bot$ | never has a proof |
| $\forall(x : A).B[x]$ | a way to prove $B[a]$ for any $a : A$ |
| $\exists(x : A).B[x]$ | a choice of $a : A$ and a proof of $B[a]$ |

# Propositions as types

| **A proof of** | **. . . is, according to BHK. . .** | |
|---|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ | $A \times B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ | |
| $A \to B$ | a way to prove $A$ given a proof of $B$ | |
| $\top$ | always has a proof | |
| $\bot$ | never has a proof | |
| $\forall(x : A).B[x]$ | a way to prove $B[a]$ for any $a : A$ | |
| $\exists(x : A).B[x]$ | a choice of $a : A$ and a proof of $B[a]$ | |

# Propositions as types

| **A proof of** | **...is, according to BHK...** | |
|---|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ | $A \times B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ | $A + B$ |
| $A \to B$ | a way to prove $A$ given a proof of $B$ | |
| $\top$ | always has a proof | |
| $\bot$ | never has a proof | |
| $\forall (x : A). B[x]$ | a way to prove $B[a]$ for any $a : A$ | |
| $\exists (x : A). B[x]$ | a choice of $a : A$ and a proof of $B[a]$ | |

# Propositions as types

| **A proof of** | **. . . is, according to BHK. . .** | |
|---|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ | $A \times B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ | $A + B$ |
| $A \to B$ | a way to prove $A$ given a proof of $B$ | $A \to B$ |
| $\top$ | always has a proof | |
| $\bot$ | never has a proof | |
| $\forall(x : A).B[x]$ | a way to prove $B[a]$ for any $a : A$ | |
| $\exists(x : A).B[x]$ | a choice of $a : A$ and a proof of $B[a]$ | |

# Propositions as types

| **A proof of** | **. . . is, according to BHK. . .** | |
|---|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ | $A \times B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ | $A + B$ |
| $A \to B$ | a way to prove $A$ given a proof of $B$ | $A \to B$ |
| $\top$ | always has a proof | $\mathbf{1}$ |
| $\bot$ | never has a proof | |
| $\forall (x : A). B[x]$ | a way to prove $B[a]$ for any $a : A$ | |
| $\exists (x : A). B[x]$ | a choice of $a : A$ and a proof of $B[a]$ | |

# Propositions as types

| **A proof of** | **. . . is, according to BHK. . .** | |
|---|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ | $A \times B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ | $A + B$ |
| $A \rightarrow B$ | a way to prove $A$ given a proof of $B$ | $A \rightarrow B$ |
| $\top$ | always has a proof | $\mathbf{1}$ |
| $\bot$ | never has a proof | $\mathbf{0}$ |
| $\forall(x : A).B[x]$ | a way to prove $B[a]$ for any $a : A$ | |
| $\exists(x : A).B[x]$ | a choice of $a : A$ and a proof of $B[a]$ | |

# Propositions as types

| **A proof of** | **. . . is, according to BHK. . .** | |
|---|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ | $A \times B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ | $A + B$ |
| $A \to B$ | a way to prove $A$ given a proof of $B$ | $A \to B$ |
| $\top$ | always has a proof | $\mathbf{1}$ |
| $\bot$ | never has a proof | $\mathbf{0}$ |
| $\forall(x : A).B[x]$ | a way to prove $B[a]$ for any $a : A$ | $(x : A) \to B$ |
| $\exists(x : A).B[x]$ | a choice of $a : A$ and a proof of $B[a]$ | |

# Propositions as types

| **A proof of** | **. . . is, according to BHK. . .** | |
|---|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ | $A \times B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ | $A + B$ |
| $A \to B$ | a way to prove $A$ given a proof of $B$ | $A \to B$ |
| $\top$ | always has a proof | $\mathbf{1}$ |
| $\bot$ | never has a proof | $\mathbf{0}$ |
| $\forall (x : A). B[x]$ | a way to prove $B[a]$ for any $a : A$ | $(x : A) \to B$ |
| $\exists (x : A). B[x]$ | a choice of $a : A$ and a proof of $B[a]$ | $(x : A) \times B$ |

# Propositions as types

| **A proof of** | **. . . is, according to BHK. . .** | |
|---|---|---|
| $A \wedge B$ | a proof of $A$ and a proof of $B$ | $A \times B$ |
| $A \vee B$ | a proof of $A$ or a proof of $B$ | $A + B$ |
| $A \to B$ | a way to prove $A$ given a proof of $B$ | $A \to B$ |
| $\top$ | always has a proof | $\mathbf{1}$ |
| $\bot$ | never has a proof | $\mathbf{0}$ |
| $\forall(x : A).B[x]$ | a way to prove $B[a]$ for any $a : A$ | $(x : A) \to B$ |
| $\exists(x : A).B[x]$ | a choice of $a : A$ and a proof of $B[a]$ | $(x : A) \times B$ |
| $s = t$ | | ? |

# The identity type

### Formation

$$\frac{A \text{ type} \qquad a : A \qquad a' : A}{a =_A a' \text{ type}}$$

### Introduction

$$\frac{a : A}{\mathsf{refl}_a : a =_A a}$$

### Elimination

$$\frac{\begin{array}{c} x : A, y : A, z : x =_A y \vdash C \text{ type} \\ x : A \vdash d : C[x \mapsto x, y \mapsto x, z \mapsto \mathsf{refl}_x] \quad p : a =_A a' \end{array}}{\mathsf{elim}_=(C, d, p) : C[x \mapsto a, y \mapsto a', z \mapsto p]}$$

### Computation

$$\mathsf{elim}_=(C, c, \mathsf{refl}_a) \equiv d[x \mapsto a] : C[x \mapsto a, y \mapsto a, z \mapsto \mathsf{refl}_a].$$

**Exercise**

Use $\mathsf{elim}_=$ to show $=$ is symmetric and transitive, and to define $\mathsf{subst} : x =_A y \to P[x] \to P[y]$.

# Contentious axioms

Many extensions of type theory relates to the identity type.

**Function extensionality:**

$$\frac{(x : A) \to f\,x =_B g\,x}{f =_{(x:A)\to B} g}$$

**Extensional Type Theory:** Adds the equality reflection rule

$$\frac{p : a =_A b}{a \equiv b : A}$$

**Uniqueness of Identity Proofs:**

$$\frac{p : a =_A b \qquad q : a =_A b}{p =_{a =_A b} q}$$

**Univalence:** "$(A =_{\mathsf{Type}} B) \cong (A \cong B)$"

# Contentious axioms

Many extensions of type theory relates to the identity type.

**Function extensionality:**

$$\frac{(x : A) \to f\,x =_B g\,x}{f =_{(x:A)\to B} g}$$

**Extensional Type Theory:** Adds the equality reflection rule

$$\frac{p : a =_A b}{a \equiv b : A}$$

**Uniqueness of Identity Proofs:**

$$\frac{p : a =_A b \qquad q : a =_A b}{p =_{a =_A b} q}$$

**Univalence:** "$(A =_{\mathsf{Type}} B) \cong (A \cong B)$"

We will try to avoid all of them.

# Summary/Outlook

Martin-Löf Type Theory a foundation for constructive mathematics.

Judgement $t : A$ means simultaneously:

- $t$ is an object of type $A$
- $t$ is a proof of the proposition $A$

Systematic way to add a type to the theory: formation, introduction, elimination, computation rules.

Can we turn the systematic into a system?