

# Less Naive Type Theory

Agnieszka Kozubek

University of Warsaw

YRF@MFCSL, 21 August 2010

# Plan

- 1 What I want to do
- 2 How do I do this? Problems and challenges
- 3 Main results

# How do we teach foundations of mathematics?

## Set theory

- provides basic notions;
- unifies the language;
- builds on first principles;
- everyone does it.

# Is set theory easy?

It SEEMS easy.

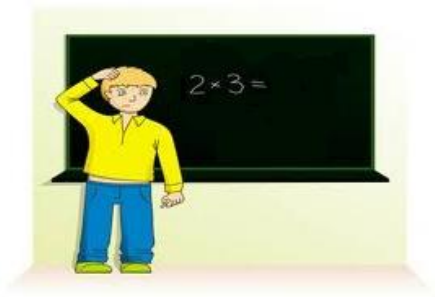
BUT it is not easy to learn.

# Is set theory easy?

It SEEMS easy.

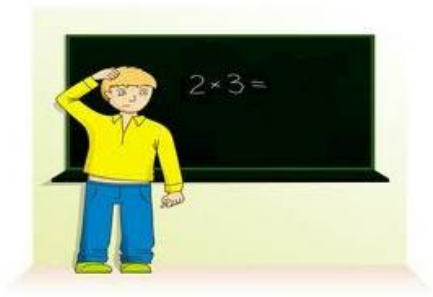
BUT it is not easy to learn.

# The confusion



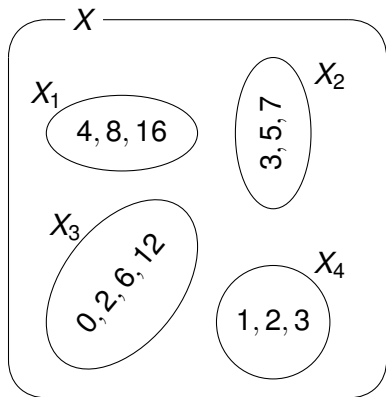
$\in$  or  $\subseteq$  ?

# The confusion



$\in$  or  $\subseteq$  ?

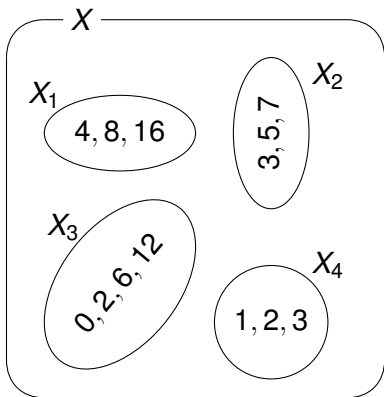
## A simple example



$$\begin{array}{ll} X \in P(P(\mathbb{N})) & \\ X_i \in X & X_i \in P(\mathbb{N}) \\ n \in X_i & n \in \mathbb{N} \end{array}$$

You need to build a **type system** in your head.

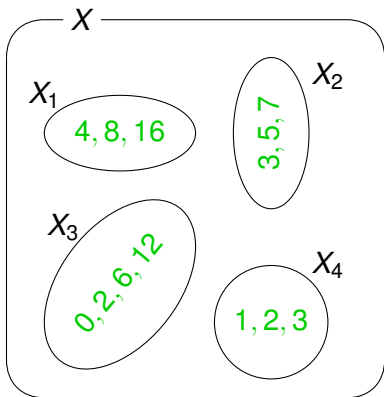
## A simple example



$$\begin{array}{l} X_i \in X \\ n \in X_i \end{array} \quad \begin{array}{l} X \in P(P(\mathbb{N})) \\ X_i \in P(\mathbb{N}) \\ n \in \mathbb{N} \end{array}$$

You need to build a **type system** in your head.

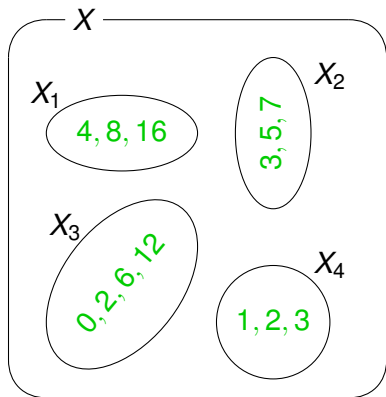
## A simple example



$$\begin{array}{ll} X_i \in X & X \in P(P(\mathbb{N})) \\ n \in X_i & X_i \in P(\mathbb{N}) \\ & n \in \mathbb{N} \end{array}$$

You need to build a **type system** in your head.

## A simple example



$$X_i \in X$$
$$n \in X_i$$

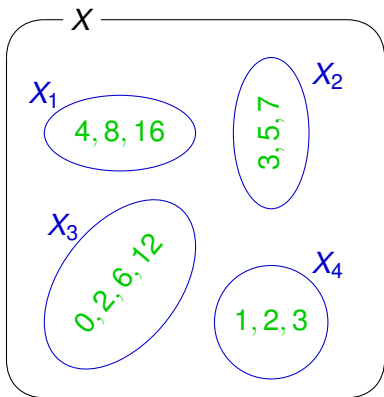
$$X \in P(P(\mathbb{N}))$$

$$X_i \in P(\mathbb{N})$$

$$n \in \mathbb{N}$$

You need to build a **type system** in your head.

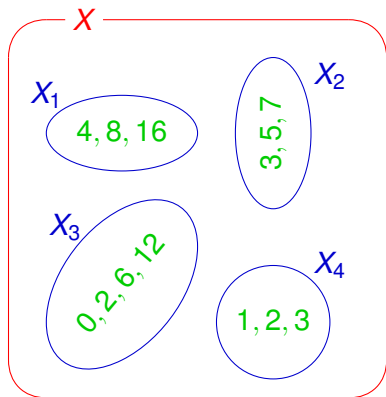
## A simple example



$$\begin{array}{ll} X_i \in X & X \in P(P(\mathbb{N})) \\ n \in X_i & X_i \in P(\mathbb{N}) \\ & n \in \mathbb{N} \end{array}$$

You need to build a **type system** in your head.

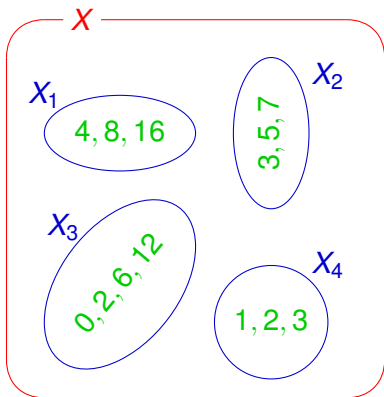
## A simple example



$$\begin{array}{ll} X_i \in X & X \in P(P(\mathbb{N})) \\ n \in X_i & X_i \in P(\mathbb{N}) \\ & n \in \mathbb{N} \end{array}$$

You need to build a **type system** in your head.

## A simple example



$$\begin{array}{ll} X \in P(P(\mathbb{N})) & \\ X_i \in X & X_i \in P(\mathbb{N}) \\ n \in X_i & n \in \mathbb{N} \end{array}$$

$$A \in \cup X?$$

You need to build a **type system** in your head.

# A simple example

$$\begin{array}{ll} X \in P(P(\mathbb{N})) & \\ X_i \in X & X_i \in P(\mathbb{N}) \\ n \in X_i & n \in \mathbb{N} \end{array}$$

$$A \in \bigcup X?$$

You need to build a **type system** in your head.

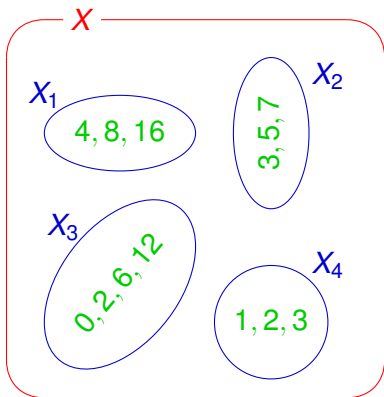
## A simple example

$$\begin{array}{ll} X \in P(P(\mathbb{N})) & \\ X_i \in X & X_i \in P(\mathbb{N}) \\ n \in X_i & n \in \mathbb{N} \end{array}$$

$$A \in \bigcup X?$$

You need to build a **type system** in your head.

## A simple example

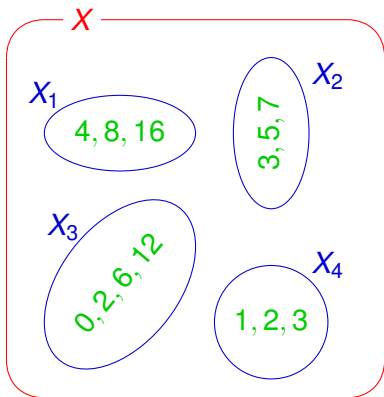


$$\begin{array}{ll} X \in P(P(\mathbb{N})) & \\ X_i \in X & X_i \in P(\mathbb{N}) \\ n \in X_i & n \in \mathbb{N} \end{array}$$

$$A \in \cup X?$$

You need to build a **type system** in your head.

## A simple example

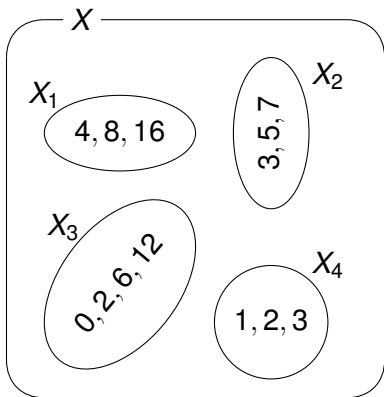


$$\begin{array}{ll} X \in P(P(\mathbb{N})) & \\ X_i \in X & X_i \in P(\mathbb{N}) \\ n \in X_i & n \in \mathbb{N} \end{array}$$

$$A \in \cup X?$$

You need to build a **type system** in your head.

## A simple example



$$\begin{array}{ll} X \in P(P(\mathbb{N})) & \text{red} \\ X_i \in X & \text{blue} \\ n \in X_i & \text{green} \\ X_i \in P(\mathbb{N}) & \text{blue} \\ n \in \mathbb{N} & \text{green} \end{array}$$

$$A \in \bigcup X?$$

You need to build a **type system** in your head.

# What I want to do

## The goal

Build a type theory which captures this informal type system.

# Pure Type Systems (PTSs)

- formalism to talk about type systems
- commonly used
- parametric representation
- useful for comparing type systems

# Less Naive Type Theory

## Features of the system

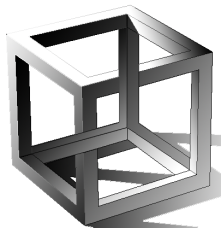
- function space,
- dependent types,
- first order logic,
- higher order logic,
- subsets as objects.

# Less Naive Type Theory

## Features of the system

- function space,
- dependent types,
- first order logic,
- higher order logic,
- **subsets as objects.**

# The main problem



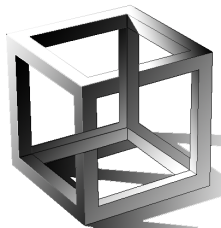
Is the system consistent?

## Paradoxes

- Russell's paradox,
- Naive Type Theory.

How do you prove consistency of a type system?

# The main problem



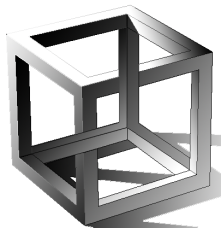
Is the system consistent?

## Paradoxes

- Russell's paradox,
- Naive Type Theory.

How do you prove consistency of a type system?

# The main problem



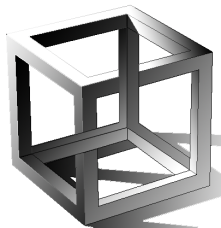
Is the system consistent?

## Paradoxes

- Russell's paradox,
- Naive Type Theory.

How do you prove consistency of a type system?

# The main problem



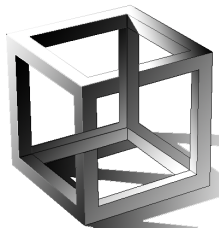
Is the system consistent?

## Paradoxes

- Russell's paradox,
- Naive Type Theory.

How do you prove consistency of a type system?

# The main problem



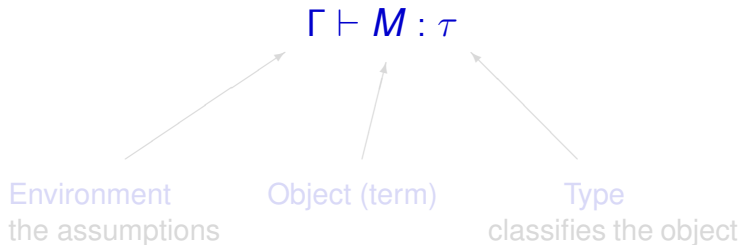
Is the system consistent?

## Paradoxes

- Russell's paradox,
- Naive Type Theory.

How do you prove consistency of a type system?

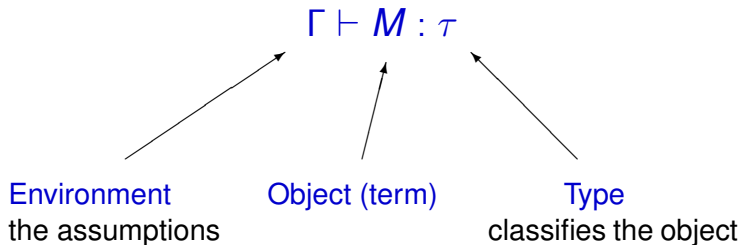
# Type assignment systems



## Typing rules

Rules telling how to create typing assignments.

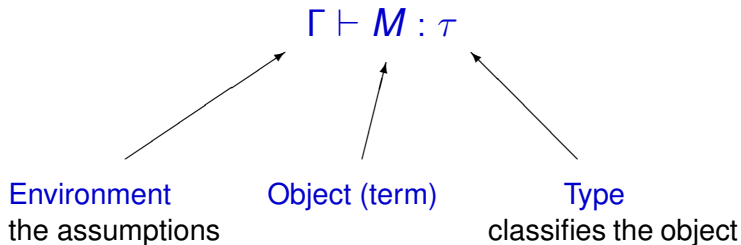
# Type assignment systems



## Typing rules

Rules telling how to create typing assignments.

# Type assignment systems



## Typing rules

Rules telling how to create typing assignments.

# An example rule

function  
from  $A$  to  $B$

argument  
of type  $A$

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

application of a function

The application rule

## An example rule

function  
from  $A$  to  $B$

argument  
of type  $A$

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

application of a function

The application rule

## An example rule

function  
from  $A$  to  $B$

argument  
of type  $A$

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

application of a function

The application rule

# An example rule

function  
from  $A$  to  $B$

argument  
of type  $A$

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

application of a function

The application rule

# An example rule

function  
from  $A$  to  $B$

argument  
of type  $A$

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

application of a function

The application rule

# An example rule

function  
from  $A$  to  $B$

argument  
of type  $A$

$$\frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

application of a function

The application rule

## An example rule

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

Does it look familiar?

## An example rule

$$\frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

The modus ponens rule

# An example rule

proof of  $A \rightarrow B$

proof of  $A$

$$\frac{\Gamma \vdash p : A \rightarrow B \quad \Gamma \vdash q : A}{\Gamma \vdash p(q) : B}$$

proof of  $B$

The modus ponens rule

# Eureka!

## The Curry-Howard isomorphism!

# The computations

$$f : \mathbb{N} \rightarrow \mathbb{N}, \quad f = \lambda x : \mathbb{N}. x + 7$$

$$f(5) = (\lambda x : \mathbb{N}. x + 7)(5) \rightarrow 5 + 7 \rightarrow 12$$

reduction

normal term

term which  
does not reduce

non-normalizing term term which can be reduced infinitely

# The computations

$$f : \mathbb{N} \rightarrow \mathbb{N}, \quad f = \lambda x : \mathbb{N}. x + 7$$

$$f(5) = (\lambda x : \mathbb{N}. x + 7)(5) \rightarrow 5 + 7 \rightarrow 12$$

reduction

normal term

term which  
does not reduce

non-normalizing term term which can be reduced infinitely

# The computations

$$f : \mathbb{N} \rightarrow \mathbb{N}, \quad f = \lambda x : \mathbb{N}. x + 7$$

$$f(5) = (\lambda x : \mathbb{N}. x + 7)(5) \rightarrow 5 + 7 \rightarrow 12$$

reduction

normal term

term which  
does not reduce

non-normalizing term term which can be reduced infinitely

# The computations

$$f : \mathbb{N} \rightarrow \mathbb{N}, \quad f = \lambda x : \mathbb{N}. x + 7$$

$$f(5) = (\lambda x : \mathbb{N}. x + 7)(5) \rightarrow 5 + 7 \rightarrow 12$$

reduction

normal term

term which  
does not reduce

non-normalizing term term which can be reduced infinitely

# The computations

$$f : \mathbb{N} \rightarrow \mathbb{N}, \quad f = \lambda x : \mathbb{N}. x + 7$$

$$f(5) = (\lambda x : \mathbb{N}. x + 7)(5) \rightarrow 5 + 7 \rightarrow 12$$

reduction

normal term

term which  
does not reduce

non-normalizing term term which can be reduced infinitely

# The computations

$$f : \mathbb{N} \rightarrow \mathbb{N}, \quad f = \lambda x : \mathbb{N}. x + 7$$

$$f(5) = (\lambda x : \mathbb{N}. x + 7)(5) \rightarrow 5 + 7 \rightarrow 12$$

reduction

normal term

term which  
does not reduce

non-normalizing term term which can be reduced infinitely

# The computations

*Lemma*<sub>1</sub> :  $A \rightarrow B$ , *Lemma*<sub>2</sub> :  $A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1 (lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have *lem*<sub>1</sub> with type  $A \rightarrow B$  and *lem*<sub>2</sub> with type  $A$ . Then we can apply *lem*<sub>1</sub> to *lem*<sub>2</sub> to obtain a proof of  $B$ . Now, note that *Lemma*<sub>1</sub> is a proof of  $A \rightarrow B$  and *Lemma*<sub>2</sub> is a proof of  $A$ . Thus we have a proof of  $B$ .

*Lemma*<sub>1</sub> (*Lemma*<sub>2</sub>) :  $B$

To prove  $B$ , apply *Lemma*<sub>1</sub> to *Lemma*<sub>2</sub>.

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1(lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1(lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1(lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1(lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can **take  $lem_1$**  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1(lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and **apply it to  $lem_2$**  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1(lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1(lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1 (lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1 (Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1(lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .

$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# The computations

$Lemma_1 : A \rightarrow B, Lemma_2 : A$

$(\lambda lem_1 : A \rightarrow B \lambda lem_2 : A. lem_1 (lem_2)) Lemma_1 Lemma_2 : B$

Suppose we have  $lem_1$  which proves  $A \rightarrow B$  and  $lem_2$  which proves  $A$ . Then we can take  $lem_1$  and apply it to  $lem_2$  to obtain a proof of  $B$ . Now, note that  $Lemma_1$  is a proof of  $A \rightarrow B$  and  $Lemma_2$  is a proof of  $A$ . Thus we have a proof of  $B$ .



$Lemma_1(Lemma_2) : B$

To prove  $B$ , apply  $Lemma_1$  to  $Lemma_2$ .

# Strong normalization

## Theorem

*Every proof of false is non-normalizing.*

## Corollary

*To prove consistency of a type system it is enough to prove that there are no non-normalizing terms.*

## Strong normalization property (Termination)

Every reduction sequence terminates = Every term is (strongly) normalizing.

# Strong normalization

## Theorem

*Every proof of false is non-normalizing.*

## Corollary

*To prove consistency of a type system it is enough to prove that there are no non-normalizing terms.*

## Strong normalization property (Termination)

Every reduction sequence terminates = Every term is (strongly) normalizing.

# Strong normalization

## Theorem

*Every proof of false is non-normalizing.*

## Corollary

*To prove consistency of a type system it is enough to prove that there are no non-normalizing terms.*

## Strong normalization property (Termination)

Every reduction sequence terminates = Every term is (strongly) normalizing.

# Main result

## Theorem

*Less Naive Type Theory is consistent.*

## Proof

Strong normalization via translation to the Calculus of Constructions.

# Main result

## Theorem

*Less Naive Type Theory is consistent.*

## Proof

Strong normalization via translation to the Calculus of Constructions.

# Work in progress

## Inductive types

Lets you define objects by induction and prove their properties.

## Example

Natural numbers, lists, trees, graphs, . . .

## Bad news

Much more complicated than pure type systems.

## Technique of the proof

Girard's candidates

# Work in progress

## Inductive types

Lets you define objects by induction and prove their properties.

## Example

Natural numbers, lists, trees, graphs, . . .

## Bad news

Much more complicated than pure type systems.

## Technique of the proof

Girard's candidates

# Work in progress

## Inductive types

Lets you define objects by induction and prove their properties.

## Example

Natural numbers, lists, trees, graphs, . . .

## Bad news

Much more complicated than pure type systems.

## Technique of the proof

Girard's candidates

# Work in progress

## Inductive types

Lets you define objects by induction and prove their properties.

## Example

Natural numbers, lists, trees, graphs, . . .

## Bad news

Much more complicated than pure type systems.

## Technique of the proof

Girard's candidates

Thank you.

# TYPES 2010

Warsaw, October 13-16, 2010

<http://types10.mimuw.edu.pl/>