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
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

\[ \in \quad \text{or} \quad \subseteq ? \]
The confusion

∈ or ⊆ ?

Agnieszka Kozubek
Less Naive Type Theory
What I want to do
How do I do this?
Main results

A simple example

\[ X \in P(P(\mathbb{N})) \]
\[ X_i \in X \]
\[ X_i \in P(\mathbb{N}) \]
\[ n \in X_i \]
\[ n \in \mathbb{N} \]

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

You need to build a type system in your head.
A simple example

You need to build a type system in your head.

\[
X \in P(P(\mathbb{N})) \\
X_i \in X \\
n \in X_i \\
n \in \mathbb{N}
\]

Agnieszka Kozubek

Less Naive Type Theory
A simple example

You need to build a type system in your head.

\[ X \in P(P(\mathbb{N})) \]

\[ X_i \in X \quad X_i \in P(\mathbb{N}) \]

\[ n \in X_i \quad n \in \mathbb{N} \]
A simple example

You need to build a type system in your head.

\[ X \in P(P(\mathbb{N})) \]

\[ X_i \in X \]

\[ n \in X_i \]

\[ n \in \mathbb{N} \]
A simple example

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

$X_i \in X$

$n \in X_i$

$n \in \mathbb{N}$

$A \in \bigcup X$?

You need to build a type system in your head.
A simple example

You need to build a type system in your head.
A simple example

You need to build a type system in your head.
A simple example

You need to build a type system in your head.
A simple example

You need to build a type system in your head.
A simple example

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
Features of the system

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

\[ \Gamma \vdash M : \tau \]

- Environment: the assumptions
- Object (term)
- Type: classifies the object

Typing rules
Rules telling how to create typing assignments.
Type assignment systems

\[ \Gamma \vdash M : \tau \]

- **Environment**: the assumptions
- **Object (term)**
- **Type**: classifies the object

**Typing rules**

Rules telling how to create typing assignments.
What I want to do
How do I do this?
Main results

Type assignment systems

\[ \Gamma \vdash M : T \]

- Environment
  - the assumptions
- Object (term)
- Type
  - classifies the object

Typing rules
Rules telling how to create typing assignments.
An example rule

function from $A$ to $B$

argument of type $A$

application of a function

The application rule
An example rule

function from $A$ to $B$

\[ \Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A \]

\[ \Gamma \vdash f(a) : B \]

argument of type $A$

application of a function

The application rule
An example rule

A function from $A$ to $B$

\[ \Gamma \vdash f : A \to B \]

\[ \Gamma \vdash a : A \]

\[ \Gamma \vdash f(a) : B \]

Argument of type $A$

Application of a function

The application rule
An example rule

- **function** from \( A \) to \( B \)
- argument of type \( A \)
- application of a function

\[
\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A \\
\Gamma \vdash f(a) : B
\]
An example rule

function from $A$ to $B$

argument of type $A$

application of a function

The application rule

$$\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A$$

$$\Gamma \vdash f(a) : B$$
An example rule

function from $A$ to $B$

argument of type $A$

application of a function

The application rule

$\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A$

$\Gamma \vdash f(a) : B$
An example rule

\[ \Gamma \vdash A \rightarrow B \quad \Gamma \vdash A \]

\[ \Gamma \vdash B \]

Does it look familiar?
An example rule

\[
\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A
\]

\[
\Gamma \vdash B
\]

The modus ponens rule
An example rule

The modus ponens rule

\[ \Gamma \vdash p : A \rightarrow B \quad \Gamma \vdash q : A \]
\[ \Rightarrow \quad \Gamma \vdash p(q) : B \]
The Curry-Howard isomorphism!
The computations

\[ f : \mathbb{N} \to \mathbb{N}, \quad f = \lambda x : \mathbb{N}. x + 7 \]

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

term which does not reduce

term which can be reduced infinitely

non-normalizing term

reduction
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
- Normal term which does not reduce
- Non-normalizing 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

\[ f : \mathbb{N} \to \mathbb{N}, \quad f = \lambda x : \mathbb{N}. x + 7 \]

\[ f(5) = (\lambda x : \mathbb{N}. x + 7)(5) \to 5 + 7 \to 12 \]
The computations

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

\[ f(5) = (\lambda x : \mathbb{N}.x + 7)(5) \rightarrow 5 + 7 \rightarrow 12 \]
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
The computations

\[ \text{Lemma}_1 : A \rightarrow B, \text{ Lemma}_2 : A \]

\[(\lambda \text{lem}_1 : A \rightarrow B \lambda \text{lem}_2 : A. \text{lem}_1(\text{lem}_2))\text{Lemma}_1 \text{Lemma}_2 : B \]

Suppose we have \text{lem}_1 which proves \( A \rightarrow B \) and \text{lem}_2 which proves \( A \). Then we can take \text{lem}_1 and apply it to \text{lem}_2 to obtain a proof of \( B \). Now, note that \text{Lemma}_1 is a proof of \( A \rightarrow B \) and \text{Lemma}_2 is a proof of \( A \). Thus we have a proof of \( B \).

\[ \text{Lemma}_1(\text{Lemma}_2) : B \]

To prove \( B \), apply \text{Lemma}_1 to \text{Lemma}_2.
The computations

\[ \text{Lemma}_1 : A \rightarrow B, \text{ Lemma}_2 : A \]

\[ (\lambda \text{lem}_1 : A \rightarrow B \lambda \text{lem}_2 : A.\text{lem}_1(\text{lem}_2))\text{Lemma}_1 \text{Lemma}_2 : B \]

Suppose we have \( \text{lem}_1 \) which proves \( A \rightarrow B \) and \( \text{lem}_2 \) which proves \( A \). Then we can take \( \text{lem}_1 \) and apply it to \( \text{lem}_2 \) to obtain a proof of \( B \). Now, note that \( \text{Lemma}_1 \) is a proof of \( A \rightarrow B \) and \( \text{Lemma}_2 \) is a proof of \( A \). Thus we have a proof of \( B \).

\[ \text{Lemma}_1(\text{Lemma}_2) : B \]

To prove \( B \), apply \( \text{Lemma}_1 \) to \( \text{Lemma}_2 \).
**The computations**

\[ \text{Lemma}_1 : A \rightarrow B, \text{ Lemma}_2 : A \]

\[ (\lambda \text{lem}_1 : A \rightarrow B \lambda \text{lem}_2 : A. \text{lem}_1(\text{lem}_2))\text{Lemma}_1\text{Lemma}_2 : B \]

Suppose we have \( \text{lem}_1 \) which proves \( A \rightarrow B \) and \( \text{lem}_2 \) which proves \( A \). Then we can take \( \text{lem}_1 \) and apply it to \( \text{lem}_2 \) to obtain a proof of \( B \). Now, note that \( \text{Lemma}_1 \) is a proof of \( A \rightarrow B \) and \( \text{Lemma}_2 \) is a proof of \( A \). Thus we have a proof of \( B \).
Suppose we have \( \text{lem}_1 \) which proves \( A \rightarrow B \) and \( \text{lem}_2 \) which proves \( A \). Then we can take \( \text{lem}_1 \) and apply it to \( \text{lem}_2 \) to obtain a proof of \( B \). Now, note that \( \text{Lemma}_1 \) is a proof of \( A \rightarrow B \) and \( \text{Lemma}_2 \) is a proof of \( A \). Thus we have a proof of \( B \).
The computations

\[ \text{Lemma}_1 : A \rightarrow B, \quad \text{Lemma}_2 : A \]

\[ (\lambda \text{lem}_1 : A \rightarrow B \lambda \text{lem}_2 : A. \text{lem}_1(\text{lem}_2)) \text{Lemma}_1 \text{Lemma}_2 : B \]

Suppose we have \( \text{lem}_1 \) which proves \( A \rightarrow B \) and \( \text{lem}_2 \) which proves \( A \). Then we can take \( \text{lem}_1 \) and apply it to \( \text{lem}_2 \) to obtain a proof of \( B \). Now, note that \( \text{Lemma}_1 \) is a proof of \( A \rightarrow B \) and \( \text{Lemma}_2 \) is a proof of \( A \). Thus we have a proof of \( B \).

\[ \text{Lemma}_1(\text{Lemma}_2) : B \]

To prove \( B \), apply \( \text{Lemma}_1 \) to \( \text{Lemma}_2 \).
Lemma_1 : A \rightarrow B, \ Lemma_2 : A

(\lambda \text{lem}_1 : A \rightarrow B \lambda \text{lem}_2 : A.\text{lem}_1(\text{lem}_2))\ Lemma_1\ Lemma_2 : B

Suppose we have \text{lem}_1 which proves A \rightarrow B and \text{lem}_2 which proves A. Then we can take \text{lem}_1 and apply it to \text{lem}_2 to obtain a proof of B. Now, note that \text{Lemma}_1 is a proof of A \rightarrow B and \text{Lemma}_2 is a proof of A. Thus we have a proof of B.

Lemma_1(\text{Lemma}_2) : B

To prove B, apply Lemma_1 to Lemma_2.
The computations

\[ \text{Lemma}_1 : A \rightarrow B, \; \text{Lemma}_2 : A \]

\[ (\lambda \text{lem}_1 : A \rightarrow B \lambda \text{lem}_2 : A. \text{lem}_1(\text{lem}_2)) \text{Lemma}_1 \text{Lemma}_2 : B \]

Suppose we have \( \text{lem}_1 \) which proves \( A \rightarrow B \) and \( \text{lem}_2 \) which proves \( A \). Then we can take \( \text{lem}_1 \) and apply it to \( \text{lem}_2 \) to obtain a proof of \( B \). Now, note that \( \text{Lemma}_1 \) is a proof of \( A \rightarrow B \) and \( \text{Lemma}_2 \) is a proof of \( A \). Thus we have a proof of \( B \).

\[ \text{Lemma}_1(\text{Lemma}_2) : B \]

To prove \( B \), apply \( \text{Lemma}_1 \) to \( \text{Lemma}_2 \).
Lemma_1 : A → B, Lemma_2 : A

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

Suppose we have lem_1 which proves A → 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 → 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 → B, Lemma_2 : A

(λlem_1 : A → Bλlem_2 : A.lem_1(lem_2))Lemma_1Lemma_2 : B

Suppose we have lem_1 which proves A → 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 → 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 → B, Lemma_2 : A

(λlem₁ : A → B) ≤ lem₂ : A.lem₁(lem₂))Lemma₁ Lemma₂ : B

Suppose we have lem₁ which proves A → B and lem₂ which proves A. Then we can take lem₁ and apply it to lem₂ to obtain a proof of B. Now, note that Lemma₁ is a proof of A → B and Lemma₂ is a proof of A. Thus we have a proof of B.

Lemma₁(Lemma₂) : B

To prove B, apply Lemma₁ to Lemma₂.
Lemma 1: \( A \rightarrow B \), Lemma 2: \( A \)

\[
(\lambda \text{lem}_1 : A \rightarrow B \lambda \text{lem}_2 : A.\text{lem}_1(\text{lem}_2))\text{Lemma}_1\text{Lemma}_2 : B
\]

Suppose we have \( \text{lem}_1 \) which proves \( A \rightarrow B \) and \( \text{lem}_2 \) which proves \( A \). Then we can take \( \text{lem}_1 \) and apply it to \( \text{lem}_2 \) to obtain a proof of \( B \). Now, note that \( \text{Lemma}_1 \) is a proof of \( A \rightarrow B \) and \( \text{Lemma}_2 \) is a proof of \( A \). Thus we have a proof of \( B \).

\[
\text{Lemma}_1(\text{Lemma}_2) : B
\]

To prove \( B \), apply \( \text{Lemma}_1 \) to \( \text{Lemma}_2 \).
What I want to do

How do I do this?

Main results

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.*
What I want to do
How do I do this?
Main results

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.
Theorem

*Less Naive Type Theory is consistent.*

Proof

Strong normalization via translation to the Calculus of Constructions.
### Inductive types

Let's 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
Inductive types

Lets you define objects by induction and prove their properties.

Example

Natural numbers, lists, trees, graphs, ...
Inductive types
Let's 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
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/