

# In the search of a naive type theory

Agnieszka Kozubek and Paweł Urzyczyn

Warsaw University, Institute of Informatics

Types 2007 — 5 May 2007

# Why do we teach set theory?

- Provides basic notions;
- Unifies the language;
- Builds on first principles;
- Everyone does it.

# The confusion

Set theory identifies two notions:

- Set as a domain (universe);
- Set as a (materialized) predicate.

With unrestricted comprehension this leads to paradoxes.

# Avoiding paradoxes

- Pretending there is no problem;
- Axiomatic set theory.

# The price of consistency


## Restricted comprehension (set formation)

Instead of  $\{x \mid W(x)\}$  use  $\{x \in A \mid W(x)\}$

## Low level implementation of simple ideas

- $\langle a, b \rangle = \{\{a\}, \{a, b\}\}$ ;
- $[a]_r = \{b \in A \mid \langle a, b \rangle \in r\}$ ;
- $7 = 5 \cup \{5, 6\}$ .

# Absolute extensionality

$$\{x \in \mathbb{Q} \mid x^2 = 2\} = \left\{ \text{ in this room \right\}$$

# Slogans

- Set (predicate)  $\neq$  Universe (type);
- Abstract idea  $\neq$  Implementation;
- Mathematics  $\neq$  Foundations of mathematics.

# A naive PTS

- Set of objects of type  $A$  is of type  $A \rightarrow *$ ;
- Therefore  $P(A) = A \rightarrow *$  is a type;
- Since  $A \rightarrow * = \prod x : A. *$  we need rule  $(*, \square, *)$ .

Bad news

This is inconsistent (Geuvers, Hurkens)

# A naive PTS

- Set of objects of type  $A$  is of type  $A \rightarrow *$ ;
- Therefore  $P(A) = A \rightarrow *$  is a type;
- Since  $A \rightarrow * = \prod x : A. *$  we need rule  $(*, \square, *)$ .

## Bad news

This is inconsistent (Geuvers, Hurkens)

# Girard's paradox

What makes a PTS inconsistent?

- A powerset construction  $P(x)$  on any object  $x$  of a sort  $s$  lives in the same sort  $s$ .
- There is enough polymorphism available in  $s$  to implement a construction of an inductive object  $\mu x:s.P(s)$ .

$$\mu t : s. P(t) = \forall t:s(\forall u:s((u \rightarrow t) \rightarrow P(t) \rightarrow t) \rightarrow t)$$

# Girard's paradox in naive type theory

In the presence of polymorphism one can define

$$\mu t : *. P(t) = \forall t: * (\forall u: * ((u \rightarrow t) \rightarrow P(t) \rightarrow t) \rightarrow t)$$

Without polymorphism one can still take:

$$\mu t : *. P(t) = \forall x: T \rightarrow * (\forall y: T \rightarrow * ((ya \rightarrow xa) \rightarrow P(ya) \rightarrow xa) \rightarrow xa).$$

# Solution: Less Naive Type Theory (LNTT)

Distinguish between types and propositions:

## Axioms

$$*^p : \Box^p \qquad *^t : \Box^t$$

## Rules

$$(*^t, *^t, *^t), (*^p, *^p, *^p), (*^t, \Box^t, \Box^t), (*^t, *^p, *^p), (*^t, \Box^p, *^t)$$

# Rules

- $(*^t, *^t, *^t)$  — simple types;
- $(*^\rho, *^\rho, *^\rho)$  — implication;
- $(*^t, \square^t, \square^t)$  — dependent types;
- $(*^t, *^\rho, *^\rho)$  — higher-order logic;
- $(*^t, \square^\rho, *^t)$  — powerset type.

# The two towers

	object world	logical world	
	$\square^t$ $\square^t$	$\square^p$	
	..   ..	..	
kinds	$B$ $*^t$	$*^p$	
	..   ..	..	
constructors, types	$\kappa$ $\tau$	$\varphi$	formulas
	..	..	
objects	$M$	$D$	proofs

It's not that simple!

# The two towers

	object world	logical world	
	$\Box^t$ $\Box^t$	$\Box^p$	
	..   ..	..	
kinds	$B$ $*^t$	$*^p$	
	..   ..	..	
constructors, types	$\kappa$ $\tau$	$\varphi$	formulas
	..	..	
objects	$M$	$D$	proofs

It's not that simple!

# The two towers

	object world		logical world	
	$\Box^t$	$\Box^t$	$\Box^p$	
	..	..	..	
kinds	$B$	$*^t$	$*^p$	
	..	..	..	
constructors, types	$\kappa$	$\mathcal{T}$	$\varphi$	formulas
		..	..	
objects		$M$	$D$	proofs

It's not that simple!

# The two towers

	object world		logical world	
	$\Box^t$	$\Box^t$	$\Box^p$	
	..	..	..	
kinds	$B$	$*^t$	$*^p$	
	..	..	..	
constructors, types	$\kappa$	$\mathcal{T}$	$\varphi$	formulas
		..	..	
objects		$M$	$D$	proofs

It's not that simple!

# The two towers

	object world		logical world	
	$\Box^t$	$\Box^t$	$\Box^p$	
	..	..	..	
kinds	$B$	$*^t$	$*^p$	
	..	..	..	
constructors, types	$\kappa$	$\mathcal{T}$	$\varphi$	formulas
		..	..	
objects		$M$	$D$	proofs

It's not that simple!

# The two towers

	object world	logical world
	$\square^t$ $\square^t$	
	..   ..	
kinds	$B$ $*^t$	$\square^p$
	..   ..	..
constructors, types	$\kappa$ $\tau$	$*^p$
	..	..
objects	$M$	$\varphi$ formulas
		..
		$D$ proofs

# The two towers

	object world	logical world
	$\square^t$ $\square^t$	
	..   ..	
kinds	$B$ $*^t$	$\square^p$
	..   ..	..
constructors, types	$\kappa$ $\tau$	$*^p$
	..	..
objects	$M$	$\varphi$ formulas
<hr/>		
		$D$ proofs

# Overview of the proof

- proof via translation to the Calculus of Constructions
- 5 level-hierarchy in LNTT vs 4 level-hierarchy in CC
- two parts
  - proof for non-proofs
  - proof for proofs

# Overview of the proof

- proof via translation to the Calculus of Constructions
- 5 level-hierarchy in LNTT vs 4 level-hierarchy in CC
- two parts
  - proof for non-proofs
  - proof for proofs

# Overview of the proof

- proof via translation to the Calculus of Constructions
- 5 level-hierarchy in LNTT vs 4 level-hierarchy in CC
- two parts
  - proof for non-proofs
  - proof for proofs

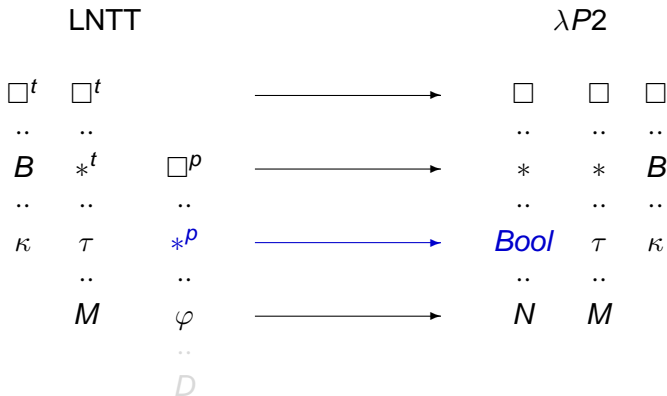
# Part 1: overview of the translation $T$

LN $TT$				$\lambda P2$		
$\Box^t$	$\Box^t$		$\longrightarrow$	$\Box$	$\Box$	$\Box$
$\dots$	$\dots$			$\dots$	$\dots$	$\dots$
$B$	$*^t$	$\Box^p$	$\longrightarrow$	$*$	$*$	$B$
$\dots$	$\dots$	$\dots$		$\dots$	$\dots$	$\dots$
$\kappa$	$\tau$	$*^p$	$\longrightarrow$	$Bool$	$\tau$	$\kappa$
	$\dots$	$\dots$		$\dots$	$\dots$	
	$M$	$\varphi$	$\longrightarrow$	$N$	$M$	
		$\dots$				
		$D$				

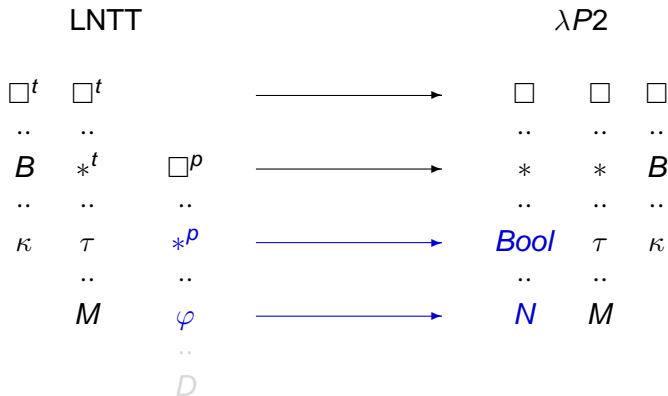
# Part 1: overview of the translation $T$

LN $TT$				$\lambda P2$		
$\square^t$	$\square^t$		$\longrightarrow$	$\square$	$\square$	$\square$
$\dots$	$\dots$			$\dots$	$\dots$	$\dots$
$B$	$*^t$	$\square^p$	$\longrightarrow$	$*$	$*$	$B$
$\dots$	$\dots$	$\dots$		$\dots$	$\dots$	$\dots$
$\kappa$	$\tau$	$*^p$	$\longrightarrow$	$Bool$	$\tau$	$\kappa$
	$\dots$	$\dots$		$\dots$	$\dots$	
	$M$	$\varphi$	$\longrightarrow$	$N$	$M$	
		$\dots$				
		$D$				

# Part 1: overview of the translation $T$

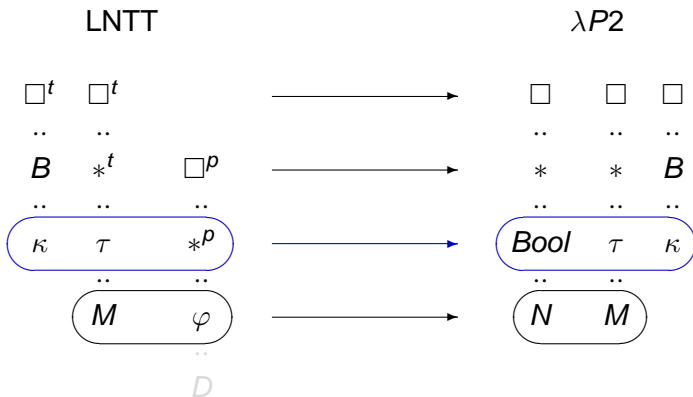


# Part 1: overview of the translation $T$

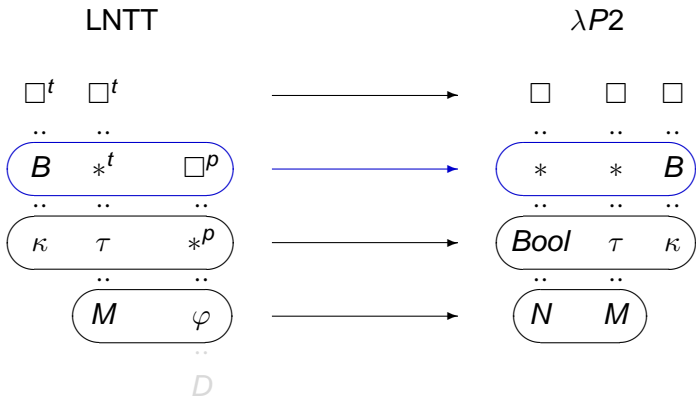




# Part 1: overview of the translation $T$



# Part 1: overview of the translation $T$



# Part 1: Soundness of the translation

## Lemma (Preserving beta-reduction)

*If*

$$M \rightarrow_{\beta} M'$$

*then*

$$T(M) \rightarrow_{\beta}^{+} T(M').$$

## Lemma (Soundness for non-proofs)

*If  $A$  is a non-proof and*

$$\Gamma \vdash A : B$$

*then*

$$T(\Gamma) \vdash T(A) : T(B).$$

## Part 1: Soundness of the translation

### Lemma (Preserving beta-reduction)

*If*

$$M \rightarrow_{\beta} M'$$

*then*

$$T(M) \rightarrow_{\beta}^{+} T(M').$$

### Lemma (Soundness for non-proofs)

*If A is a non-proof and*

$$\Gamma \vdash A : B$$

*then*

$$T(\Gamma) \vdash T(A) : T(B).$$

# Part 1: Conclusion

## Lemma

*Non-proofs are strongly normalizing.*

## Part 2: overview

- strong normalization for proofs
- we use strong normalization for non-proofs
- more complicated part
- really five levels of the hierarchy
- translation to the Calculus of Constructions

## Part 2: overview

- strong normalization for proofs
- we use strong normalization for non-proofs
- more complicated part
- really five levels of the hierarchy
- translation to the Calculus of Constructions

## Part 2: overview

- strong normalization for proofs
- we use strong normalization for non-proofs
- more complicated part
- really five levels of the hierarchy
- translation to the Calculus of Constructions

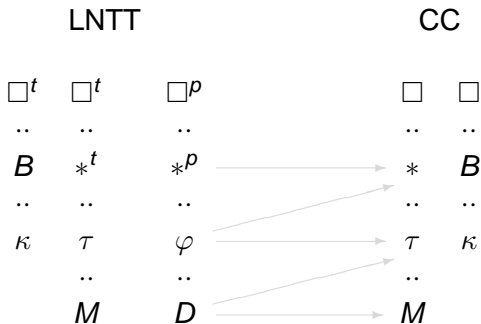
## Part 2: overview

- strong normalization for proofs
- we use strong normalization for non-proofs
- more complicated part
- really five levels of the hierarchy
- translation to the Calculus of Constructions

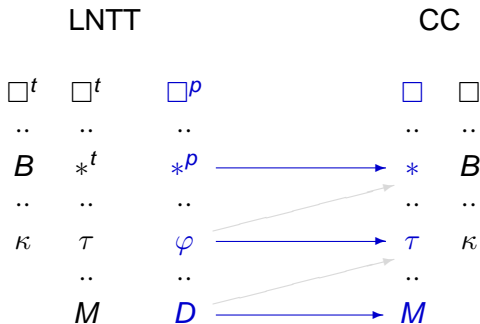
## Part 2: overview

- strong normalization for proofs
- we use strong normalization for non-proofs
- more complicated part
- really five levels of the hierarchy
- translation to the Calculus of Constructions

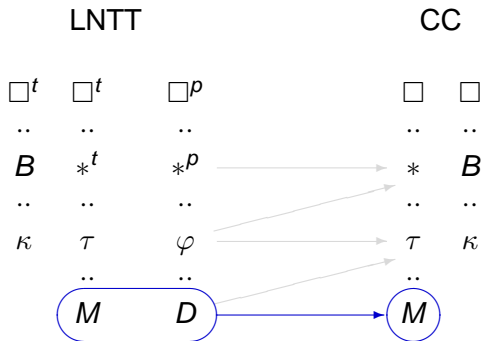
# Part 2: overview of the translation $t$



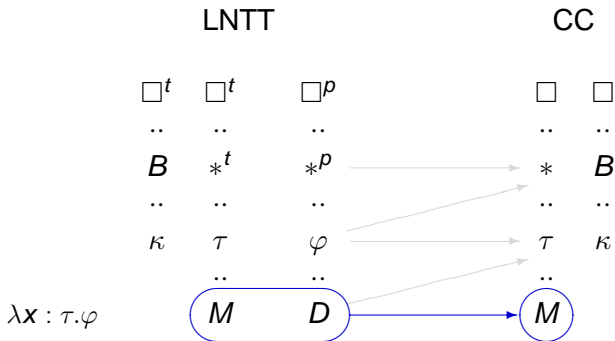
# Part 2: overview of the translation $t$



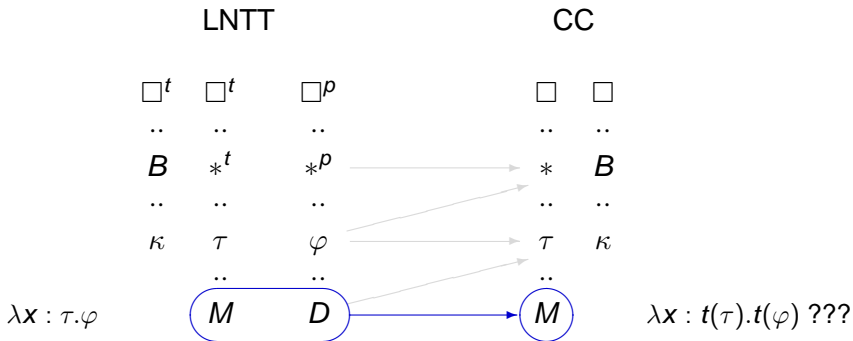
## Part 2: overview of the translation $t$



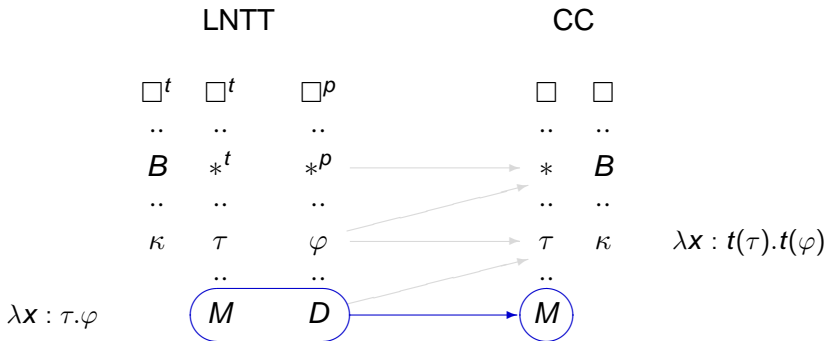
# Part 2: overview of the translation $t$



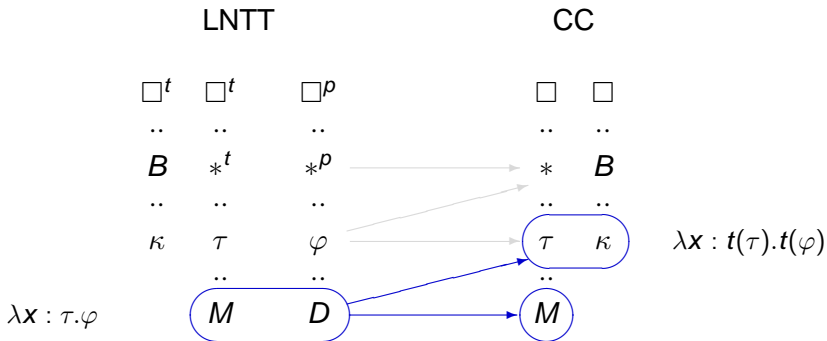
# Part 2: overview of the translation $t$



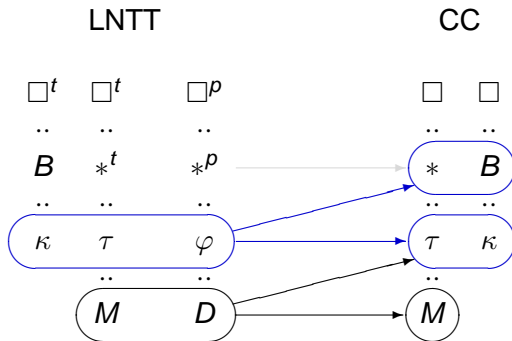
# Part 2: overview of the translation $t$



## Part 2: overview of the translation $t$

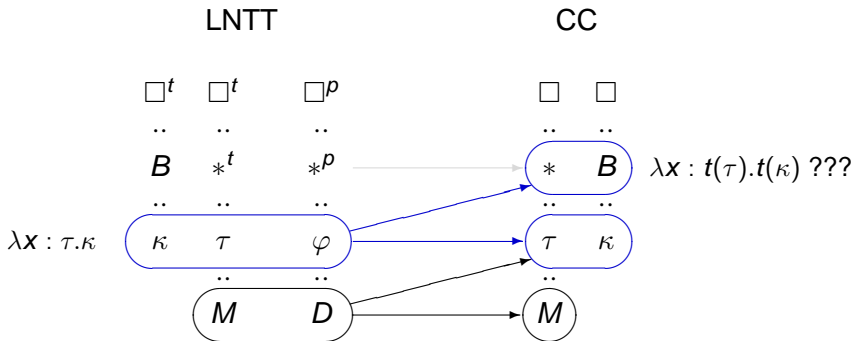


## Part 2: overview of the translation $t$

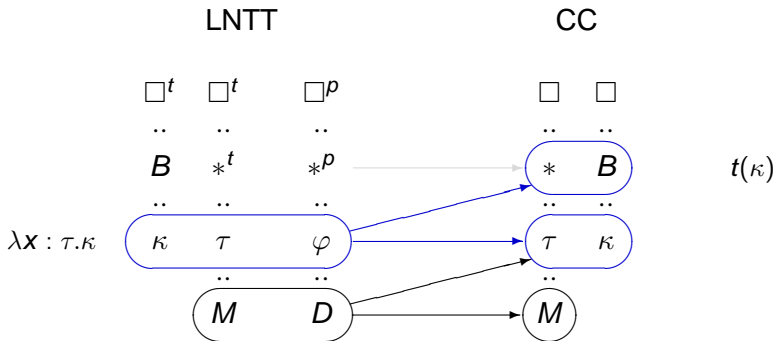




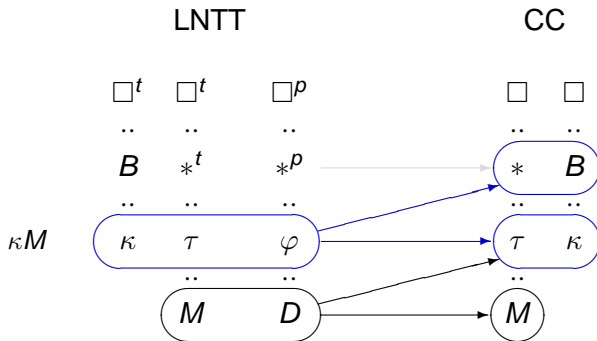
# Part 2: overview of the translation $t$



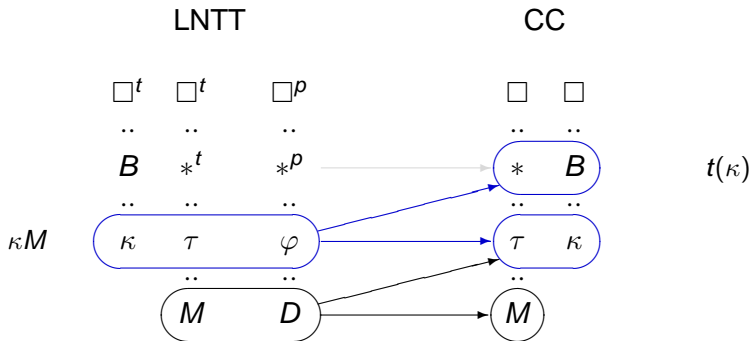
# Part 2: overview of the translation $t$



# Part 2: overview of the translation $t$



# Part 2: overview of the translation $t$





## Fact

Some reductions are erased by the translation.

Lemma (Soundness for proofs)

*If  $D$  is a proof and*

$$\Gamma \vdash D : T$$

*then*

$$t(\Gamma) \vdash t(D) : t(T).$$

## Fact

Some reductions are erased by the translation.

## Lemma (Soundness for proofs)

*If  $D$  is a proof and*

$$\Gamma \vdash D : T$$

*then*

$$t(\Gamma) \vdash t(D) : t(T).$$

## Part 2: silent reductions

Two kinds of reductions:

- reductions which are preserved by the translation  $t$
- **silent** reductions, which are erased by the translation  $t$

### Lemma

*Silent reductions in proofs occur only in types for bound variables.*

If  $D \rightarrow_{\beta} D'$  is silent then it is of the form

$$C[\lambda x : \tau.M] \rightarrow C[\lambda x : \tau'.M].$$

Reductions in  $\tau$  create no new redexes outside  $\tau$ .

## Part 2: silent reductions

Two kinds of reductions:

- reductions which are preserved by the translation  $t$
- **silent** reductions, which are erased by the translation  $t$

### Lemma

*Silent reductions in proofs occur only in types for bound variables.*

If  $D \rightarrow_{\beta} D'$  is silent then it is of the form

$$C[\lambda x : \tau.M] \rightarrow C[\lambda x : \tau'.M].$$

Reductions in  $\tau$  create no new redexes outside  $\tau$ .

## Part 2: silent reductions

Two kinds of reductions:

- reductions which are preserved by the translation  $t$
- **silent** reductions, which are erased by the translation  $t$

### Lemma

*Silent reductions in proofs occur only in types for bound variables.*

If  $D \rightarrow_{\beta} D'$  is silent then it is of the form

$$C[\lambda x : \tau.M] \rightarrow C[\lambda x : \tau'.M].$$

Reductions in  $\tau$  create no new redexes outside  $\tau$ .

## Part 2: silent reductions

Two kinds of reductions:

- reductions which are preserved by the translation  $t$
- **silent** reductions, which are erased by the translation  $t$

### Lemma

*Silent reductions in proofs occur only in types for bound variables.*

If  $D \rightarrow_{\beta} D'$  is silent then it is of the form

$$C[\lambda x : \tau.M] \rightarrow C[\lambda x : \tau'.M].$$

Reductions in  $\tau$  create no new redexes outside  $\tau$ .

## Part 2: conclusion

### Lemma

*Proofs are strongly normalizing.*

### Proof.

Let's assume there is an infinite reduction  $D \rightarrow_{\beta} D_1 \rightarrow_{\beta} \dots$

- 1 At some point there are only silent reductions.
- 2 The reductions occur in types for bound variables.
- 3 No new redexes outside  $\tau$ .
- 4 There are only finitely many bound variables in  $D$ .
- 5 By part 1, types for bound variables are strongly normalizing.



## Part 2: conclusion

### Lemma

*Proofs are strongly normalizing.*

### Proof.

Let's assume there is an infinite reduction  $D \rightarrow_{\beta} D_1 \rightarrow_{\beta} \dots$

- 1 At some point there are only silent reductions.
- 2 The reductions occur in types for bound variables.
- 3 No new redexes outside  $\tau$ .
- 4 There are only finitely many bound variables in  $D$ .
- 5 By part 1, types for bound variables are strongly normalizing.



## Part 2: conclusion

### Lemma

*Proofs are strongly normalizing.*

### Proof.

Let's assume there is an infinite reduction  $D \rightarrow_{\beta} D_1 \rightarrow_{\beta} \dots$

- 1 At some point there are only silent reductions.
- 2 The reductions occur in types for bound variables.
- 3 No new redexes outside  $\tau$ .
- 4 There are only finitely many bound variables in  $D$ .
- 5 By part 1, types for bound variables are strongly normalizing.



## Part 2: conclusion

### Lemma

*Proofs are strongly normalizing.*

### Proof.

Let's assume there is an infinite reduction  $D \rightarrow_{\beta} D_1 \rightarrow_{\beta} \dots$

- 1 At some point there are only silent reductions.
- 2 The reductions occur in types for bound variables.
- 3 No new redexes outside  $\tau$ .
- 4 There are only finitely many bound variables in  $D$ .
- 5 By part 1, types for bound variables are strongly normalizing.



## Part 2: conclusion

### Lemma

*Proofs are strongly normalizing.*

### Proof.

Let's assume there is an infinite reduction  $D \rightarrow_{\beta} D_1 \rightarrow_{\beta} \dots$

- 1 At some point there are only silent reductions.
- 2 The reductions occur in types for bound variables.
- 3 No new redexes outside  $\tau$ .
- 4 There are only finitely many bound variables in  $D$ .
- 5 By part 1, types for bound variables are strongly normalizing.



## Part 2: conclusion

### Lemma

*Proofs are strongly normalizing.*

### Proof.

Let's assume there is an infinite reduction  $D \rightarrow_{\beta} D_1 \rightarrow_{\beta} \dots$

- 1 At some point there are only silent reductions.
- 2 The reductions occur in types for bound variables.
- 3 No new redexes outside  $\tau$ .
- 4 There are only finitely many bound variables in  $D$ .
- 5 By part 1, types for bound variables are strongly normalizing.



# Conclusion

## Theorem

*LNTT has strong normalization property.*