

Inductive types in Less Naive Type Theory

Agnieszka Kozubek

University of Warsaw, Institute of Informatics

Types 2010 — 15 October 2010

- 1 What I want to do?
- 2 Less Naive Type Theory
- 3 Inductive types in LNTT

Motivation

- to build a type system;
- subsets (predicates) are objects;
- natural idea;
- does not occur usually in type systems.

The (short-term) goal

- system which may serve as foundation of mathematics;
- capture first-year mathematics;
- an informal type system is already used;
- to be used in teaching;

Pure Type System: triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$

- \mathcal{S} – the set of sorts,
- \mathcal{A} – the set of axioms,
- \mathcal{R} – the set of rules,
rules are triples in $\mathcal{S} \times \mathcal{S} \times \mathcal{S}$

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} (s_1, s_2, s_3)$$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, *AUTOMATH*
- 2 Berardi, *λ PRED*

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} (s_1, s_2, s_3)$$

The subset is an object

$$M : \tau \rightarrow *$$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, *AUTOMATH*
- 2 Berardi, *λ PRED*

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} \quad (s_1, s_2, s_3)$$

The subset is an object

$$M : \tau \rightarrow * \quad (*, \square, ?)$$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, *AUTOMATH*
- 2 Berardi, *λ PRED*

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} \quad (s_1, s_2, s_3)$$

The subset is an object

$$M : \tau \rightarrow * \quad (*, \square, ?)$$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, AUTOMATH
- 2 Berardi, λ PRED

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} \quad (s_1, s_2, s_3)$$

The subset is an object

$$M : \tau \rightarrow * \quad (*, \square, ?)$$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, AUTOMATH
- 2 Berardi, λ PRED

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} \quad (s_1, s_2, s_3)$$

The subset is an object

$$M : \tau \rightarrow * : *$$

$(*, \square, ?)$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, AUTOMATH
- 2 Berardi, λ PRED

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} \quad (s_1, s_2, s_3)$$

The subset is an object

$$M : \tau \rightarrow * : *$$

$$(*, \square, *)$$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, AUTOMATH
- 2 Berardi, λ PRED

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} \quad (s_1, s_2, s_3)$$

The subset is an object

$$M : \mathcal{T} \rightarrow * : *$$

$$(*, \square, *)$$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, *AUTOMATH*
- 2 Berardi, *λ PRED*

The powerset rule

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A. B : s_3} (s_1, s_2, s_3)$$

The subset is an object

$$M : \tau \rightarrow * : *$$

$$(*, \square, *)$$

$$s_2 \neq s_3$$

Other examples

- 1 van Benthem Jutting, AUTOMATH
- 2 Berardi, λ PRED

$$\mathcal{S} = \{*, \square\}$$

$$\mathcal{A} = \{* : \square\}$$

$$\mathcal{R} = \{(*, *, *), (*, \square, *), (\square, *, *)\}$$

$$\mathcal{S} = \{*, \square\}$$

$$\mathcal{A} = \{* : \square\}$$

$$\mathcal{R} = \{(*, *, *), (*, \square, *), (\square, *, *)\}$$

This is inconsistent!

Less Naive Type Theory

$$\mathcal{S} = \{ *^t, *^p, \Box^t, \Box^p \} \quad \mathcal{A} = \{ *^t : \Box^t, *^p : \Box^p \}$$

$\mathcal{R} = \{$

$(*^t, *^t, *^t)$

$(*^p, *^p, *^p)$

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

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

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

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

$\}$

simple types

implication

universal quantification

higher order logic

dependent types

subsets are objects

The two towers

	object world		logical world	
	\square^t	\square^t		
		
kinds	B	$*^t$	\square^p	
	
constructors, types	κ	τ	$*^p$	
		
objects		M	φ	formulas
			..	
			D	proofs

The two towers

	object world		logical world	
	\square^t	\square^t		
		
kinds	B	$*^t$	\square^p	
	
constructors, types	κ	τ	$*^p$	
		
objects	M	$\lambda x : \tau. \varphi$	φ	formulas
			..	
			D	proofs

The two towers

	object world			logical world	
	\square^t	\square^t			
			
kinds	B	$*^t$		\square^p	
	
constructors, types	κ	τ	$\prod x : \tau. *^p$	$*^p$	
		
objects		M	$\lambda x : \tau. \varphi$	φ	formulas
				..	
				D	proofs

Consistency of LNTT

- we prove the strong normalization property
- 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

Consistency of LNTT

- we prove the strong normalization property
- 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

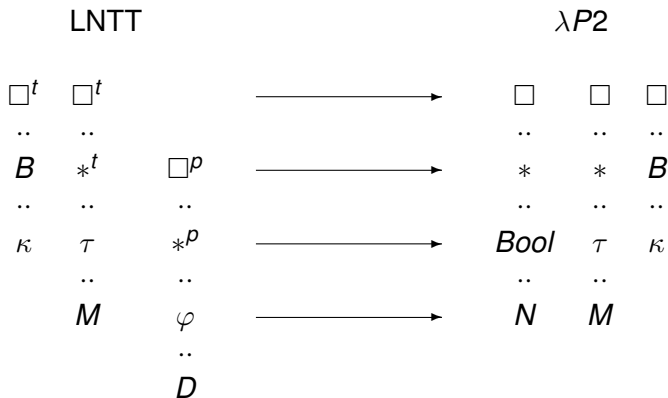
Consistency of LNTT

- we prove the strong normalization property
- 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

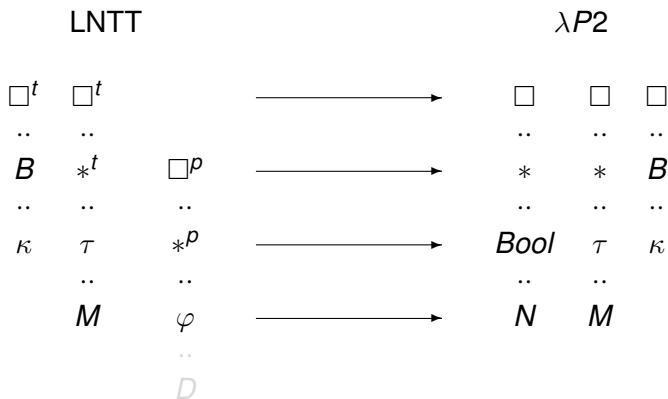
Consistency of LNTT

- we prove the strong normalization property
- 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

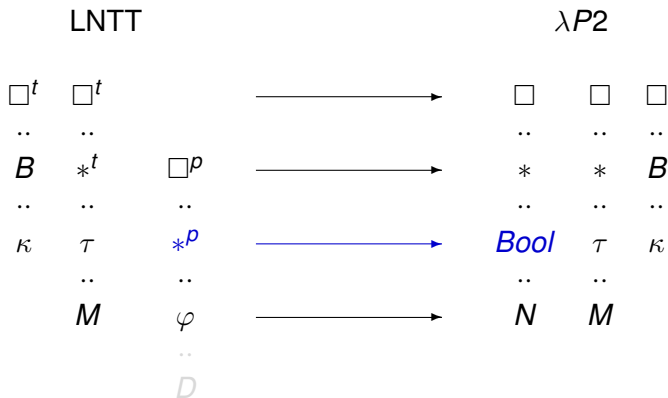
Part 1: non-proofs



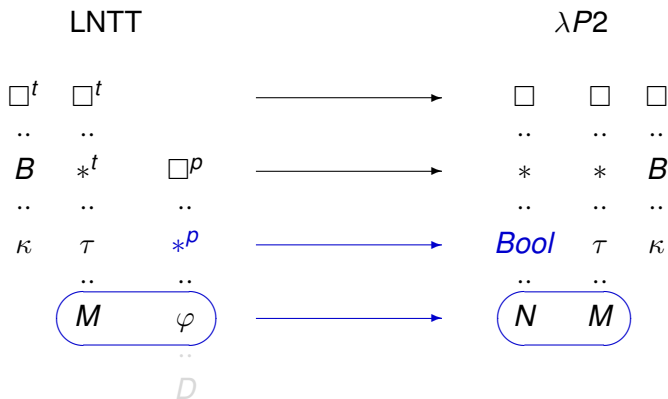
Part 1: non-proofs



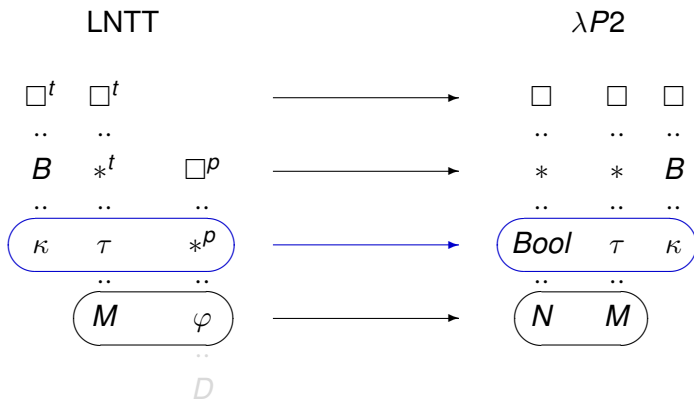
Part 1: non-proofs



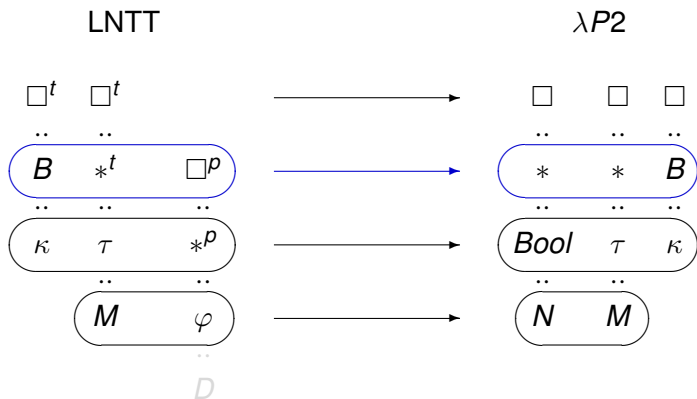
Part 1: non-proofs



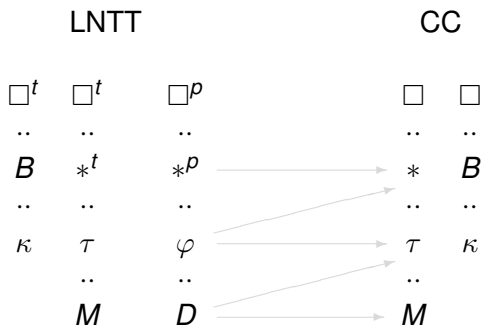
Part 1: non-proofs



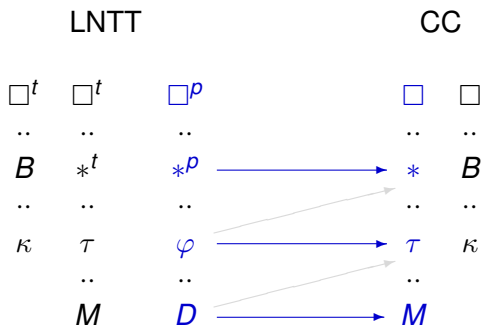
Part 1: non-proofs



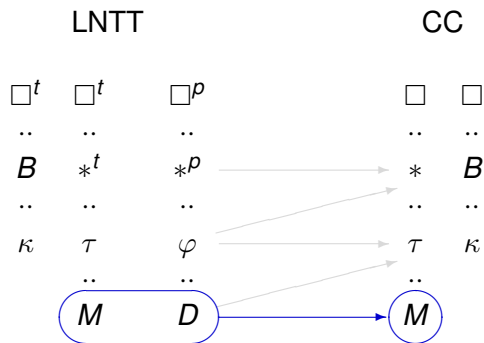
Part 2: proofs



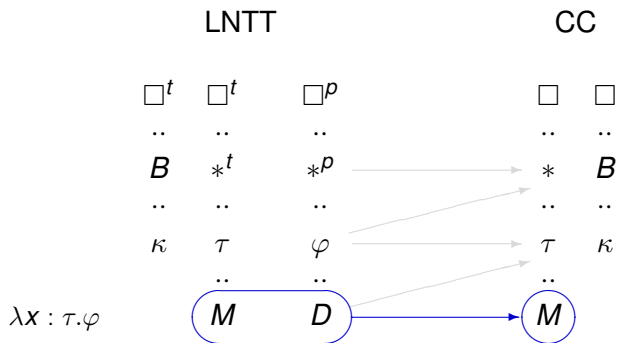
Part 2: proofs



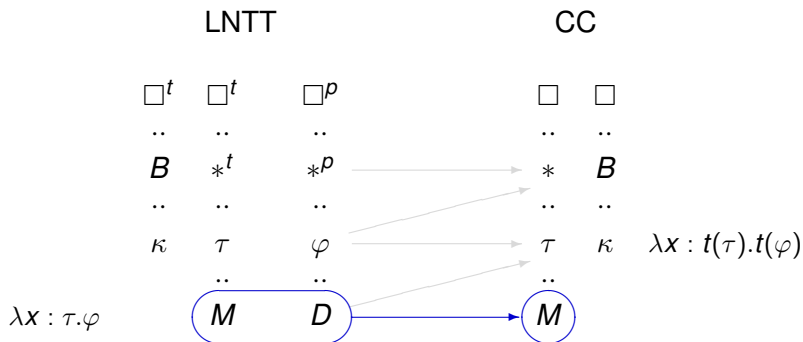
Part 2: proofs



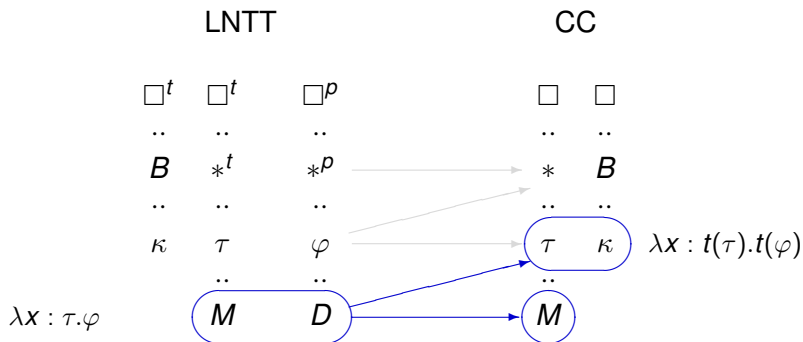
Part 2: proofs



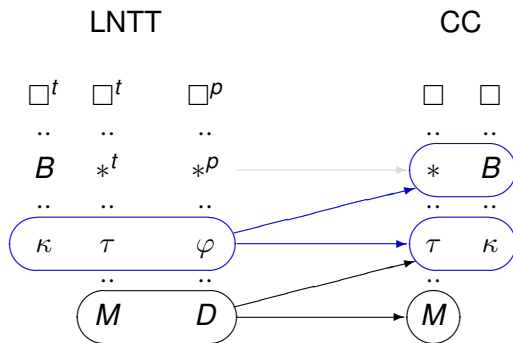
Part 2: proofs



Part 2: proofs



Part 2: proofs



- adding datatypes (natural numbers, lists, graphs, ...)
- inductive types and inductive predicates
- usual positivity condition
- small dependent elimination on inductive types
- small non-dependent elimination on inductive predicates

Example

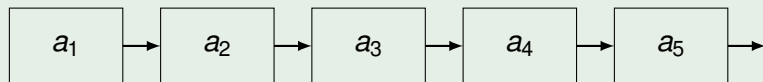
We have natural numbers, disjoint sum, empty type, ...

Example (Lists)

Inductive List : $*^t :=$

nil : List |

cons : $A \rightarrow \text{List} \rightarrow \text{List}$.



Example

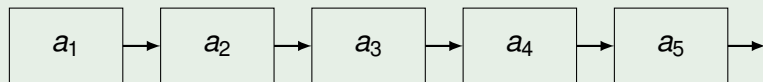
We have natural numbers, disjoint sum, empty type, ...

Example (Lists)

Inductive List : $*^t :=$

nil : List |

cons : $A \rightarrow \text{List} \rightarrow \text{List}$.



This also works if $A = \tau \rightarrow *^p!$

Example

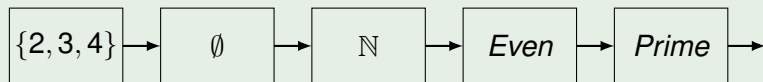
We have natural numbers, disjoint sum, empty type, ...

Example (Lists)

Inductive List : $*^t :=$

nil : *List* |

cons : $A \rightarrow \textit{List} \rightarrow \textit{List}$.

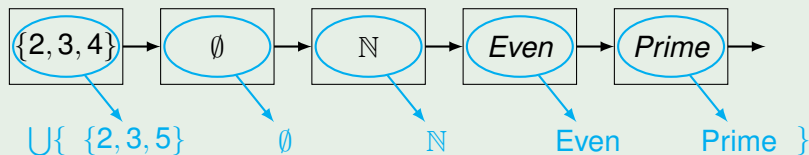


This also works if $A = \tau \rightarrow *^p!$

Example

Example (Lists)

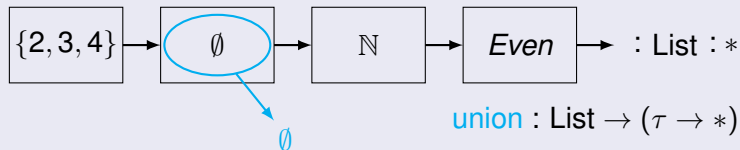
```
fun union : List → (τ → *P) :=  
  λl : List.match l with  
    nil : Empty |  
    cons X t ⇒ X ∪ (union t).
```



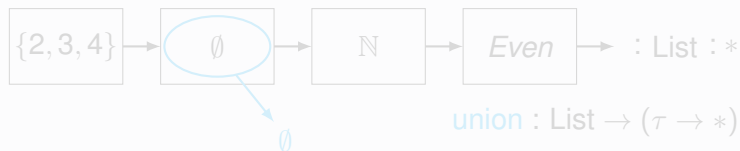
Small elimination but we may extract something (almost) large.

Comparison with Coq – we can do more!

Coq with impredicative *Set*

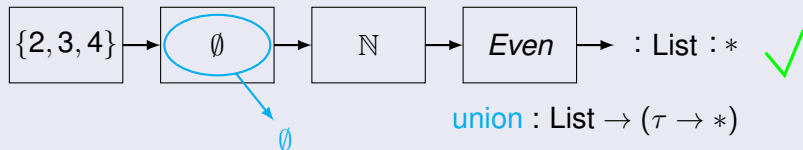


Coq with predicative *Set*

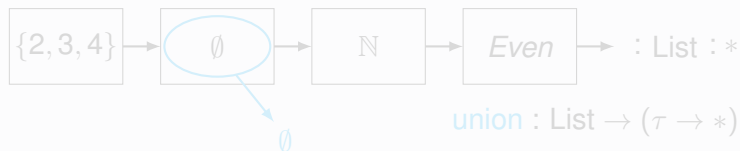


Comparison with Coq – we can do more!

Coq with impredicative *Set*

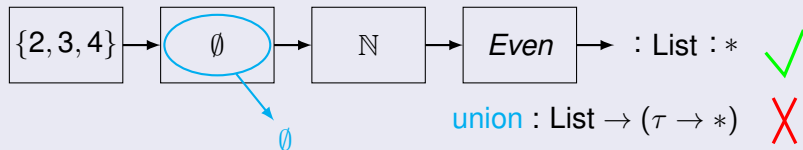


Coq with predicative *Set*



Comparison with Coq – we can do more!

Coq with impredicative *Set*

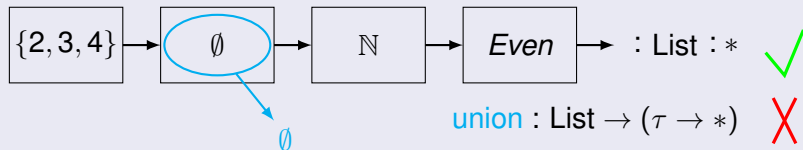


Coq with predicative *Set*



Comparison with Coq – we can do more!

Coq with impredicative *Set*



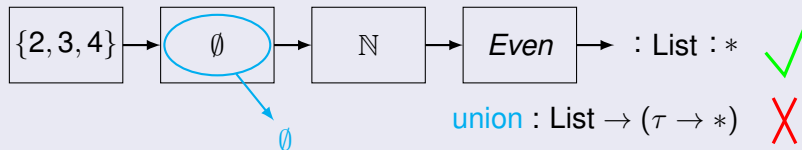
Large elimination on large inductive type

Coq with predicative *Set*



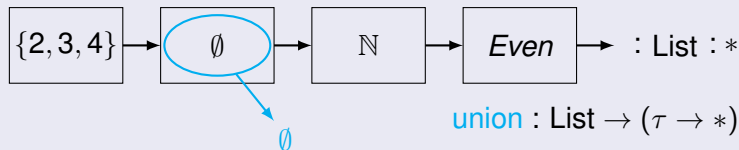
Comparison with Coq – we can do more!

Coq with impredicative *Set*



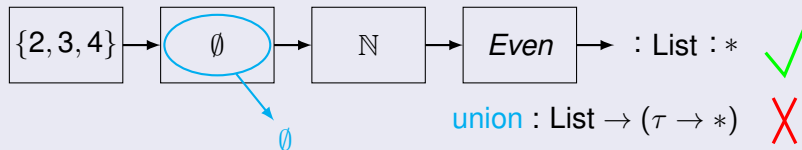
Large elimination on large inductive type

Coq with predicative *Set*



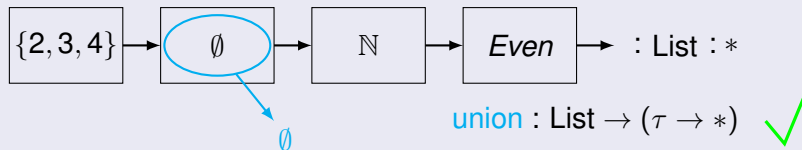
Comparison with Coq – we can do more!

Coq with impredicative *Set*



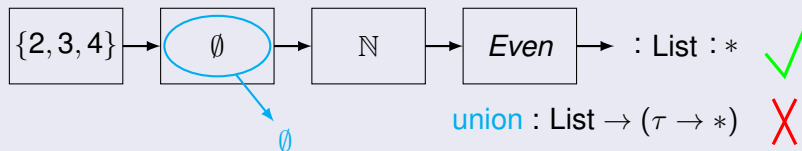
Large elimination on large inductive type

Coq with predicative *Set*



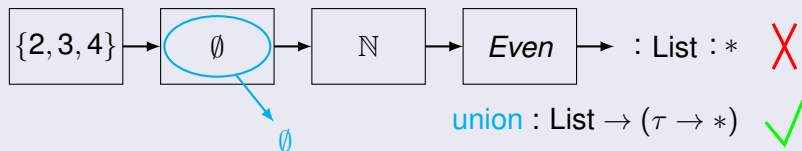
Comparison with Coq – we can do more!

Coq with impredicative *Set*



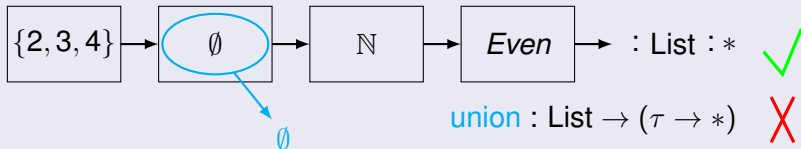
Large elimination on large inductive type

Coq with predicative *Set*



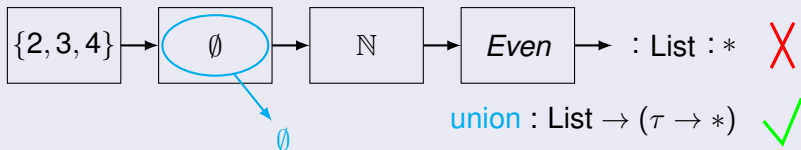
Comparison with Coq – we can do more!

Coq with impredicative *Set*



Large elimination on large inductive type

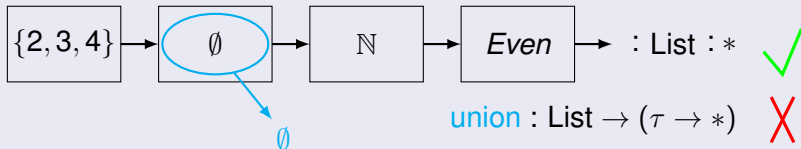
Coq with predicative *Set*



Predicativity of *Set*

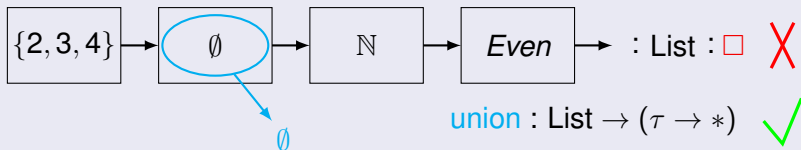
Comparison with Coq – we can do more!

Coq with impredicative *Set*



Large elimination on large inductive type

Coq with predicative *Set*



Predicativity of *Set*

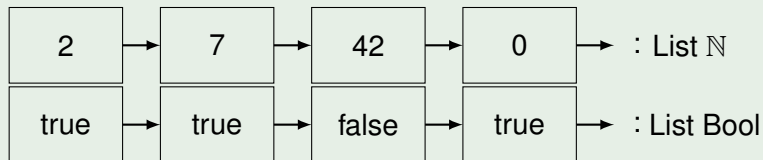
Comparison with Coq – we can do less :(

Example (Polymorphic lists)

Inductive List($A : *^t$) : $*^t :=$

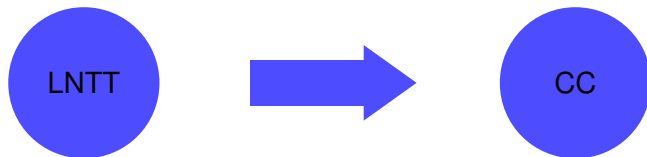
nil : *List* A |

cons : $A \rightarrow \textit{List } A \rightarrow \textit{List } A.$

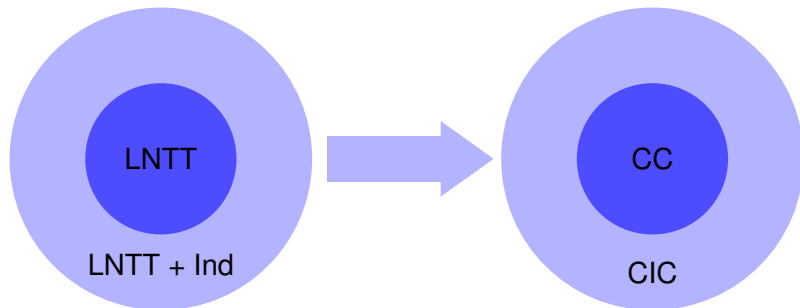


This requires type polymorphism.

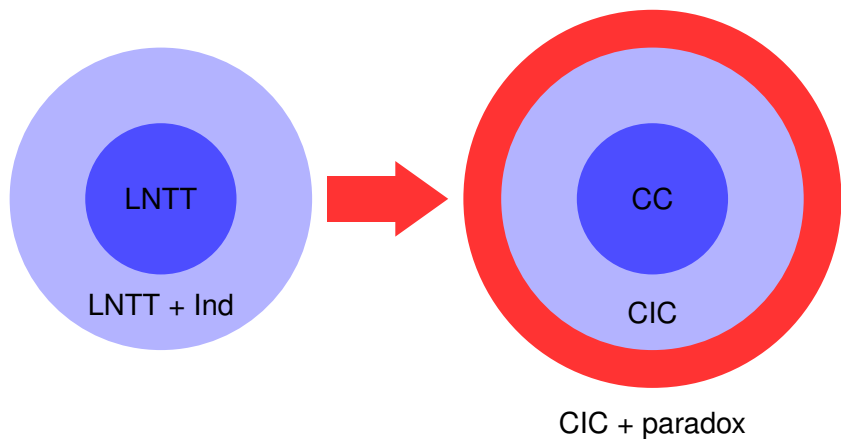
Consistency of LNTT + Ind – first attempt



Consistency of LNTT + Ind – first attempt



Consistency of LNTT + Ind – first attempt

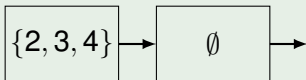


The translation does not work

LNTT + Ind

Lists

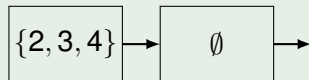
Inductive List : $*^t$:=
nil : List |
cons : $(\tau \rightarrow *^p) \rightarrow \text{List} \rightarrow \text{List}$.



CIC

Lists

Inductive List : * :=
nil : List |
cons : $(\tau \rightarrow *) \rightarrow \text{List} \rightarrow \text{List}$.

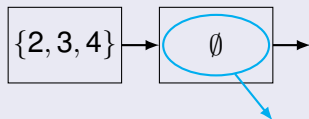


The translation does not work

LNTT + Ind

Lists

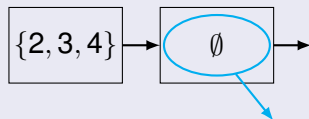
```
fun union : List → (τ → *P) :=  
λl : List.match l with  
nil : Empty |  
cons X t ⇒ X ∪ (union t).
```



CIC

Lists

```
fun union : List → (τ → *) :=  
λl : List.match l with  
nil : Empty |  
cons X t ⇒ X ∪ (union t).
```

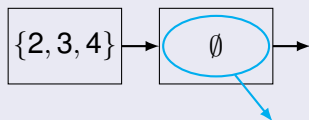


The translation does not work

LNTT + Ind

Lists

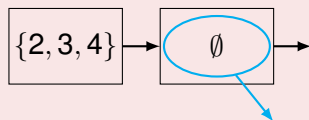
```
fun union : List → (τ → *P) :=  
λl : List.match l with  
nil : Empty |  
cons X t ⇒ X ∪ (union t).
```



CIC

Lists

```
fun union : List → (τ → *) :=  
λl : List.match l with  
nil : Empty |  
cons X t ⇒ X ∪ (union t).
```



The strong normalization property

- proof by the method of candidates
- typed candidates

$$\{M \mid \dots\}$$

- we provide interpretation for
 - type-like terms (types, formulas, kinds)
 - type constructors and large inductive objects

The strong normalization property

- proof by the method of candidates
- typed candidates

$$\{\Delta \vdash M \mid \dots\}$$

- we provide interpretation for
 - type-like terms (types, formulas, kinds)
 - type constructors and large inductive objects

The strong normalization property

- interpretation for inductive type

$$[List] = lfp(F)$$

$$F(S) = Base \cup \{ \Delta \vdash M \mid M \in SN, M \rightarrow^* nil \vee \\ M \rightarrow^* cons X I \wedge X \in [\tau] \wedge I \in S \}$$

- interpretation for inductive objects

$$[cons X I] = \langle [X], [I] \rangle$$

the interpretation is used for conversion rule

- interpretation for elimination terms

$$[match (cons X I) with f_1 \mid f_2] = [f_2 X I]$$

- different interpretation for inductive predicates (due to polymorphism for formulas)

Theorem

Less Naive Type Theory with inductive types is strongly normalizing.

Corollary

Less Naive Type Theory with inductive types is consistent.

Thank you!