Inductive types in Less Naive Type Theory

Agnieszka Kozubek

University of Warsaw, Institute of Informatics

Types 2010 — 15 October 2010

Plan

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 Systems

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

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

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

The powerset rule

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

The subset is an object

$$M : \tau \longrightarrow *$$

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

The powerset rule

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

The subset is an object

$$M : \tau \longrightarrow *$$

$$(*, \Box, ?)$$

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

The powerset rule

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

The subset is an object

$$M : \tau \longrightarrow *$$

$$(*, \Box, ?)$$

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

The powerset rule

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

The subset is an object

$$M : \tau \longrightarrow *$$

$$(*, \Box, ?)$$

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

The powerset rule

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

The subset is an object

$$M : \tau \longrightarrow * : * \qquad\qquad (*, \Box, ?)$$

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

The powerset rule

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

The subset is an object

$$M : \tau \longrightarrow * : * \qquad\qquad (*, \Box, *)$$

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

The powerset rule

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

The subset is an object

$$M : \tau \longrightarrow * : * \qquad\qquad (*, \Box, *)$$

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

The powerset rule

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

The subset is an object

$$M : \tau \longrightarrow * : * \qquad\qquad (*, \Box, *)$$

$$s_2 \neq s_3$$

Other examples

1. van Benthem Jutting, AUTOMATH
2. Berardi, $\lambda PRED$

Naive Type Theory

$$\mathcal{S} = \{*, \Box\}$$
$$\mathcal{A} = \{* : \Box\}$$
$$\mathcal{R} = \{(*, *, *), (*, \Box, *), (\Box, *, *)\}$$

Naive Type Theory

$$\mathcal{S} = \{*, \Box\}$$
$$\mathcal{A} = \{* : \Box\}$$
$$\mathcal{R} = \{(*, *, *), (*, \Box, *), (\Box, *, *)\}$$

This is inconsistent!

Less Naive Type Theory

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

$$\mathcal{R} = \{$$

$(*^t, *^t, *^t)$	simple types
$(*^p, *^p, *^p)$	implication
$(*^t, *^p, *^p)$	universal quantification
$(\Box^p, *^p, *^p)$	higher order logic
$(*^t, \Box^t, \Box^t)$	dependent types
$(*^t, \Box^p, *^t)$	subsets are objects

$$\}$$

The two towers

	object world		logical world	
	$\Box^t$	$\Box^t$		
	..	..		
kinds	B	$*^t$	$\Box^p$	
	..	..	..	
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	φ	formulas
		$\lambda x : \tau.\varphi$	..	
			D	proofs

The two towers

	object world			logical world	
	$\Box^t$	$\Box^t$			
	..	..			
kinds	B	$*^t$		$\Box^p$	
	..	..		..	
constructors, types	κ	τ	$\Pi x : \tau.*^p$	$*^p$	
		..		..	
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

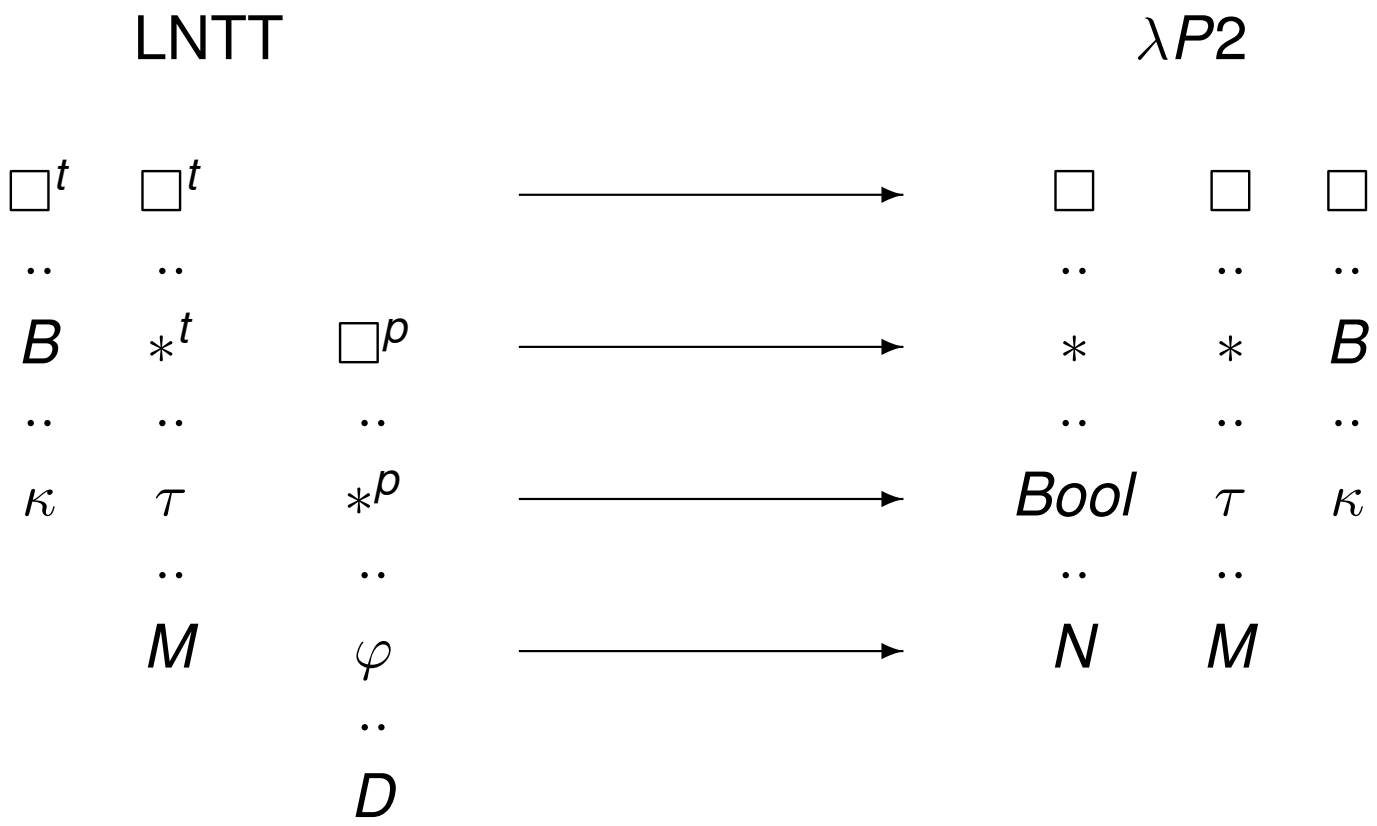
Consistency of LNTT

- 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

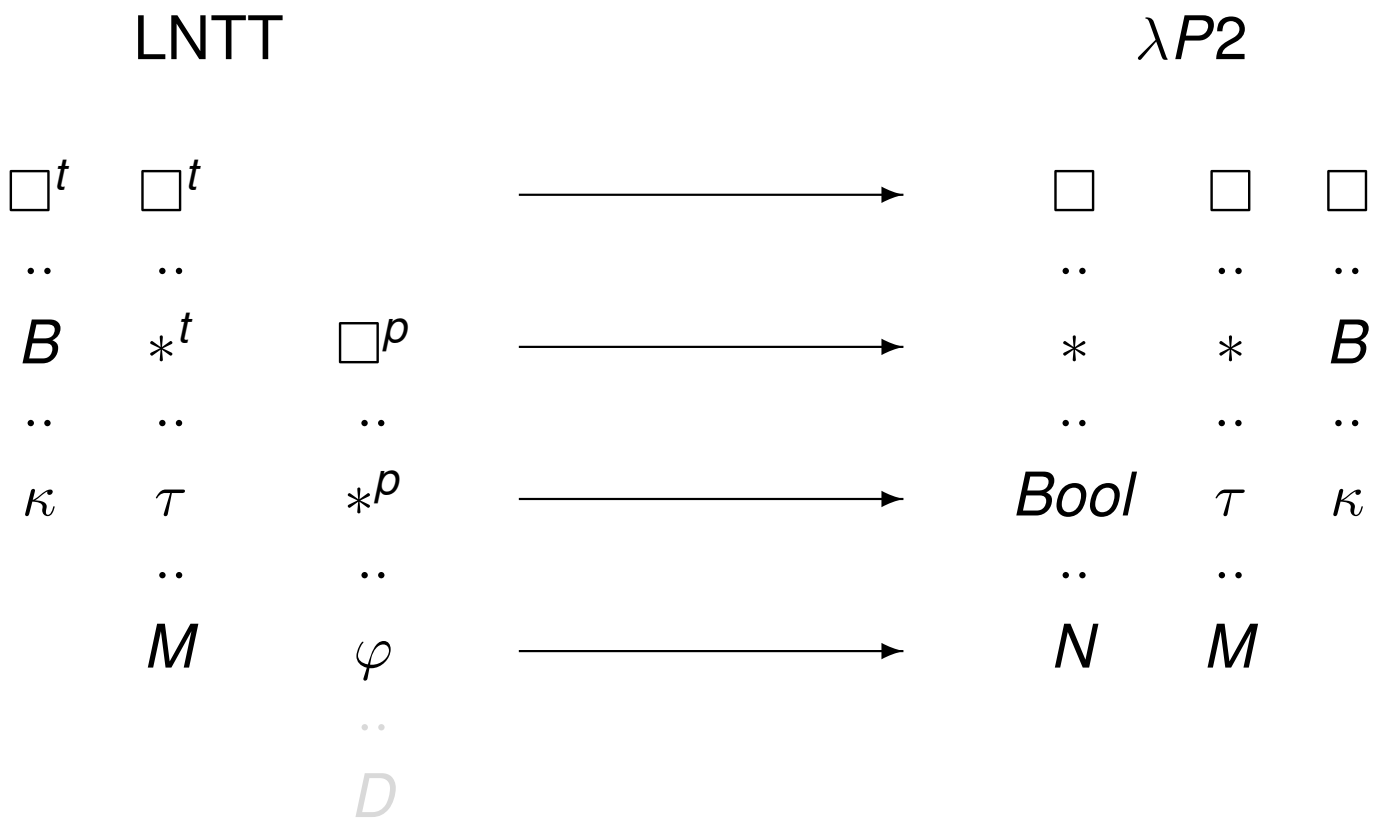
Consistency of LNTT

- 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

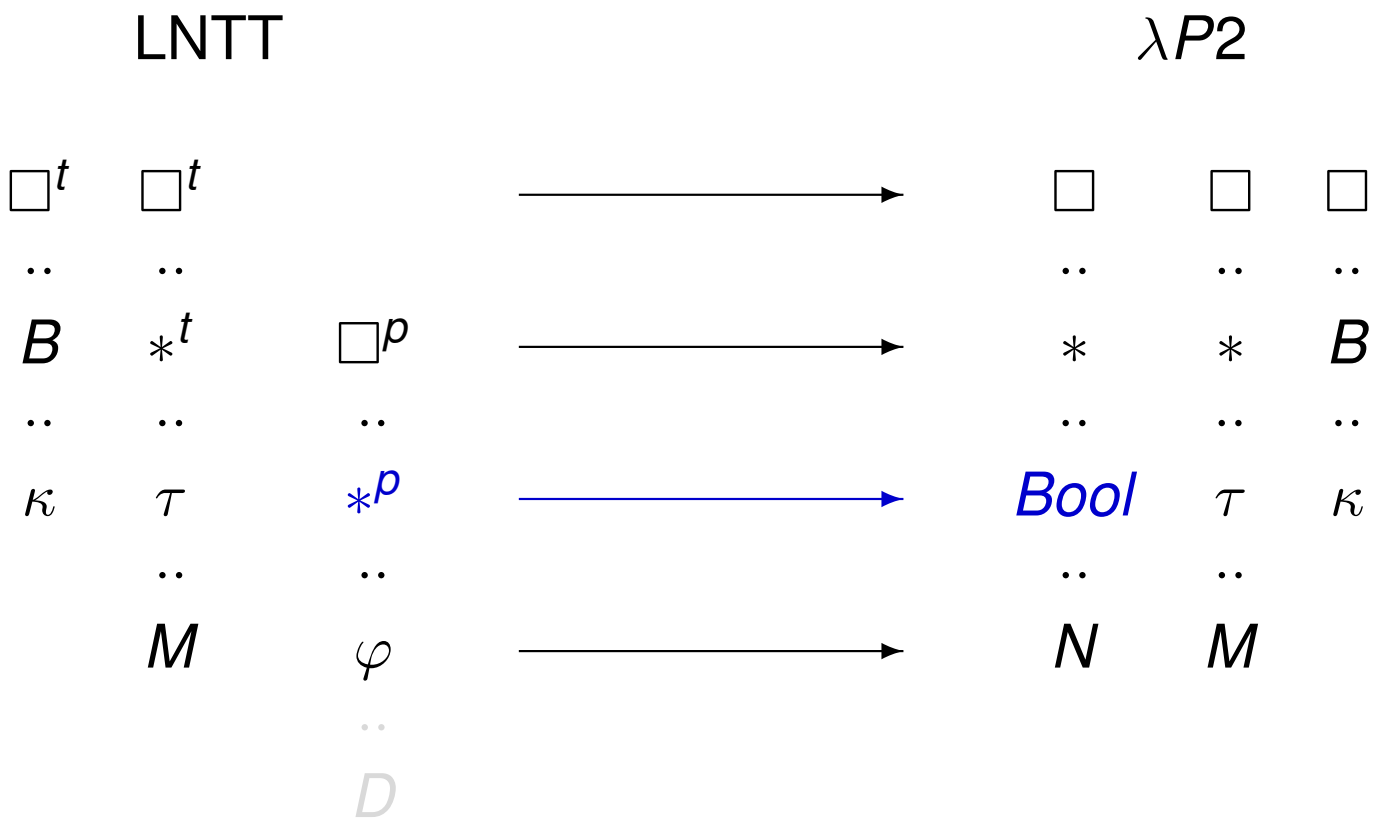
Part 1: non-proofs



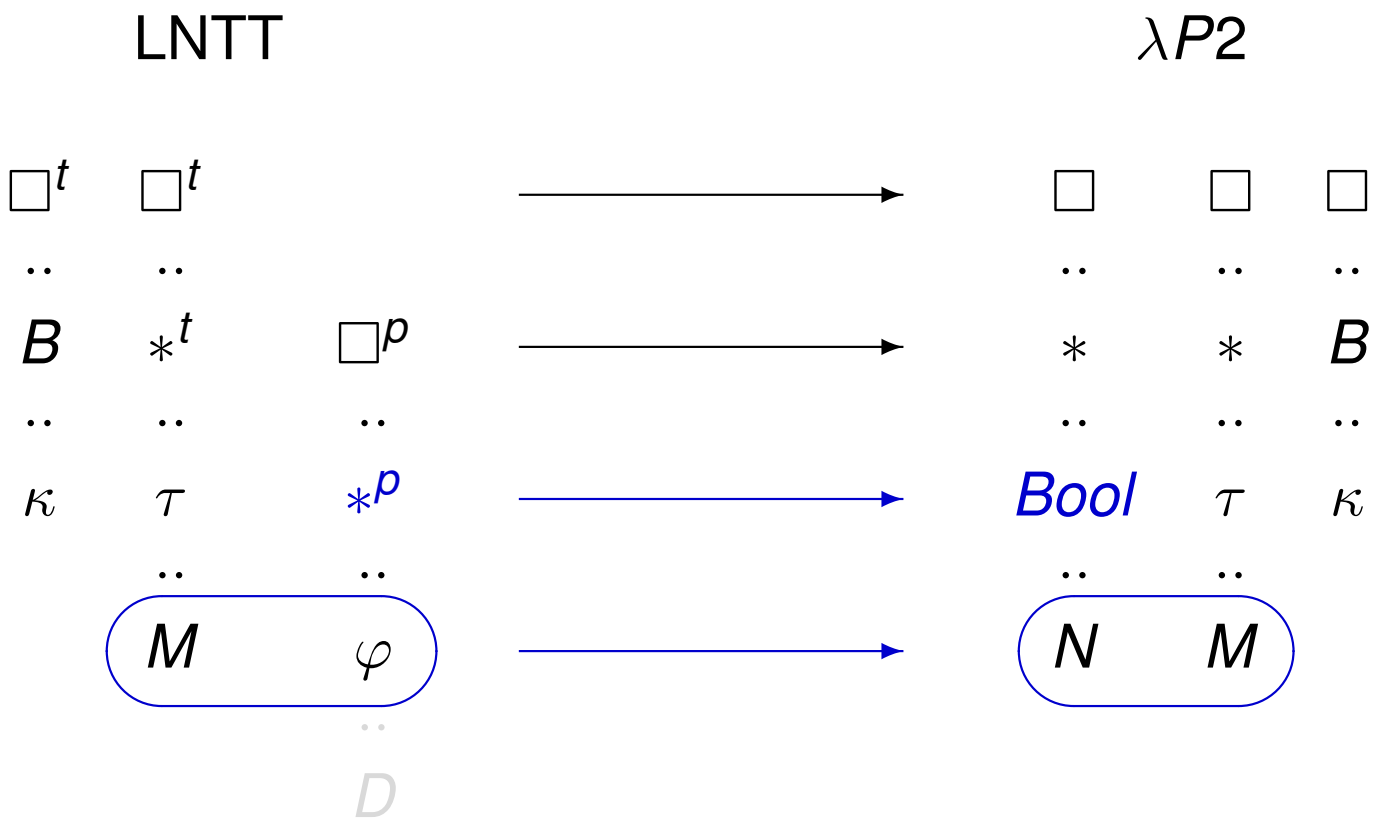
Part 1: non-proofs



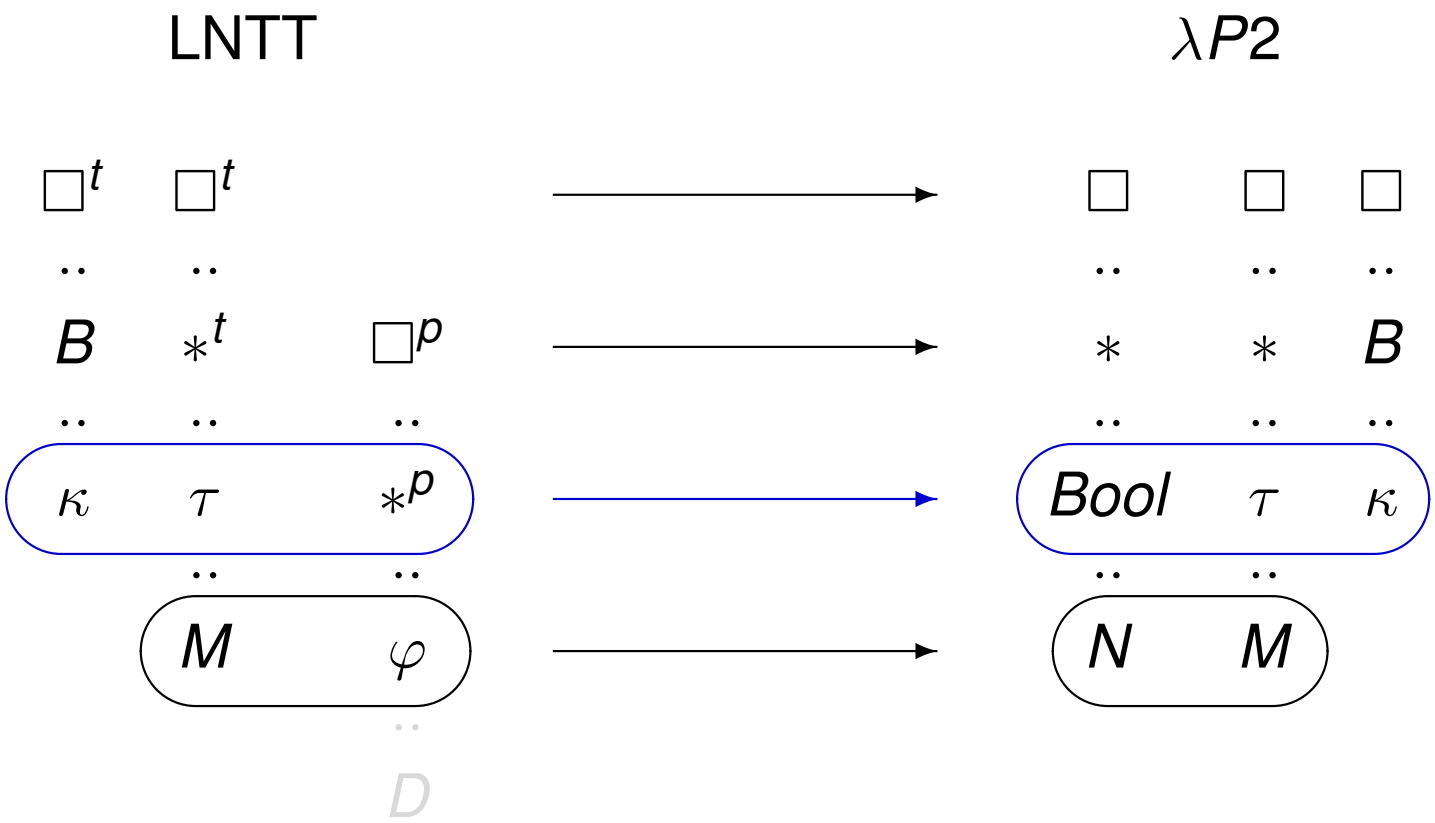
Part 1: non-proofs



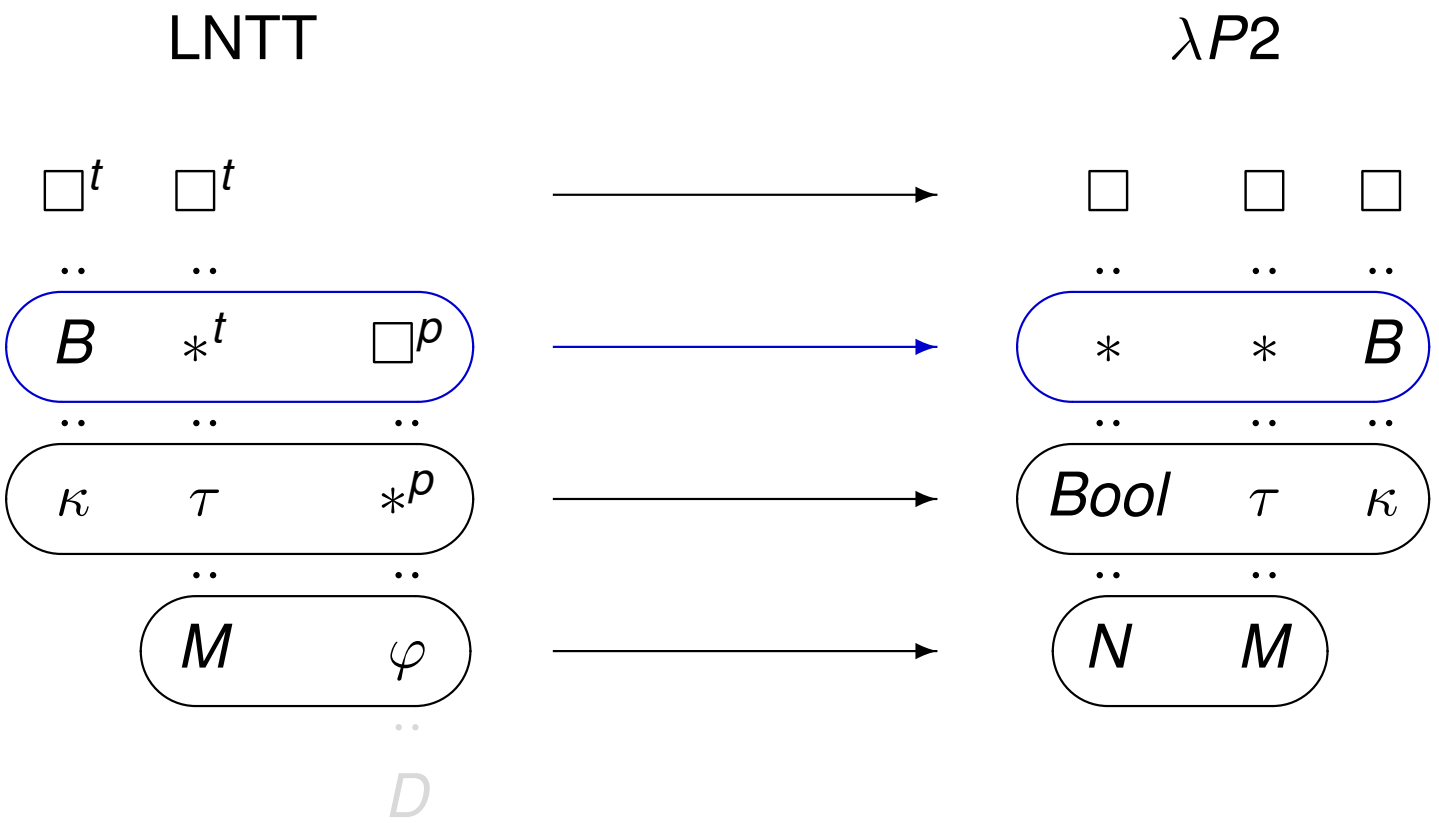
Part 1: non-proofs



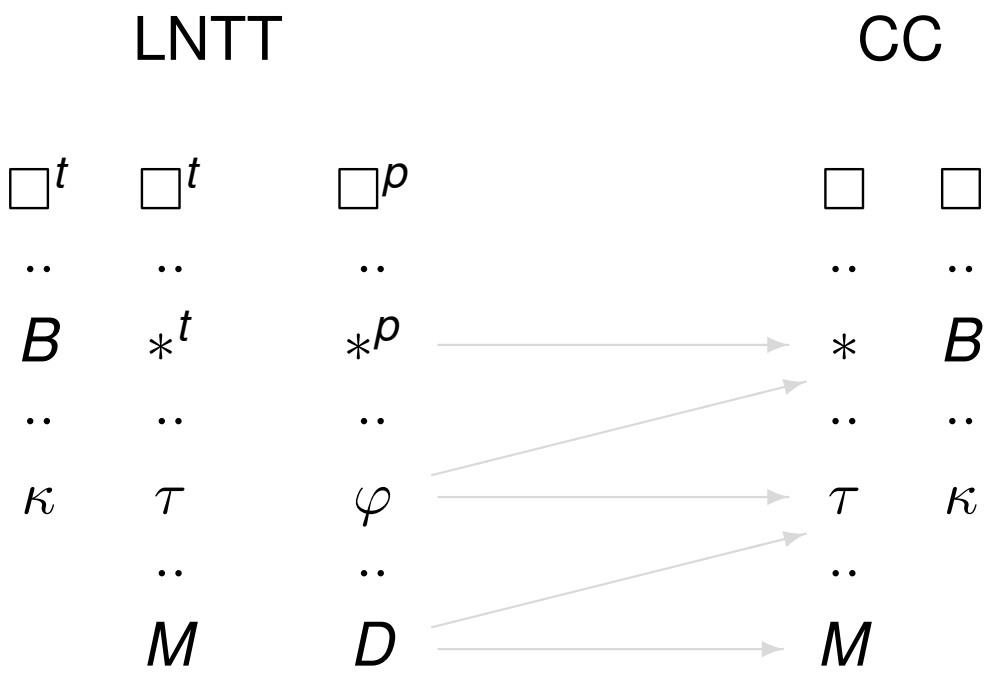
Part 1: non-proofs



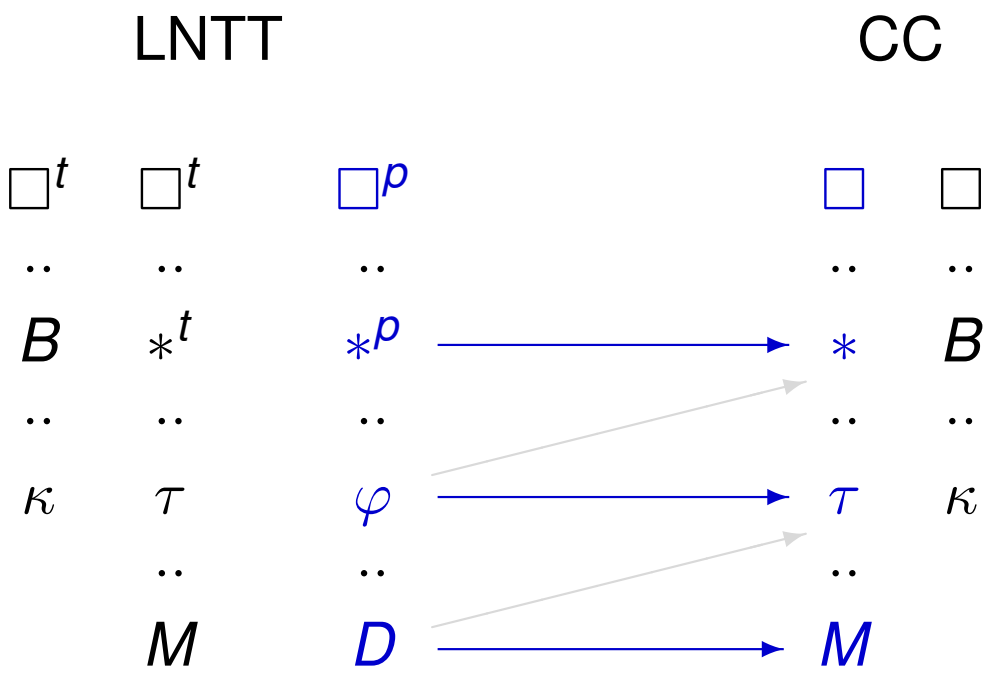
Part 1: non-proofs



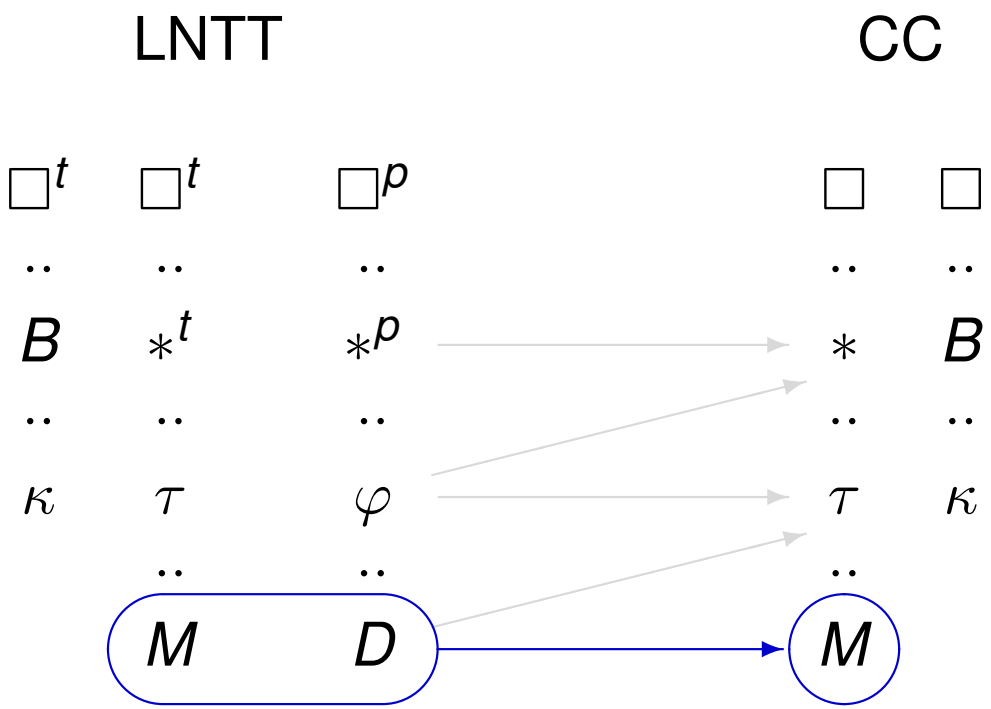
Part 2: proofs



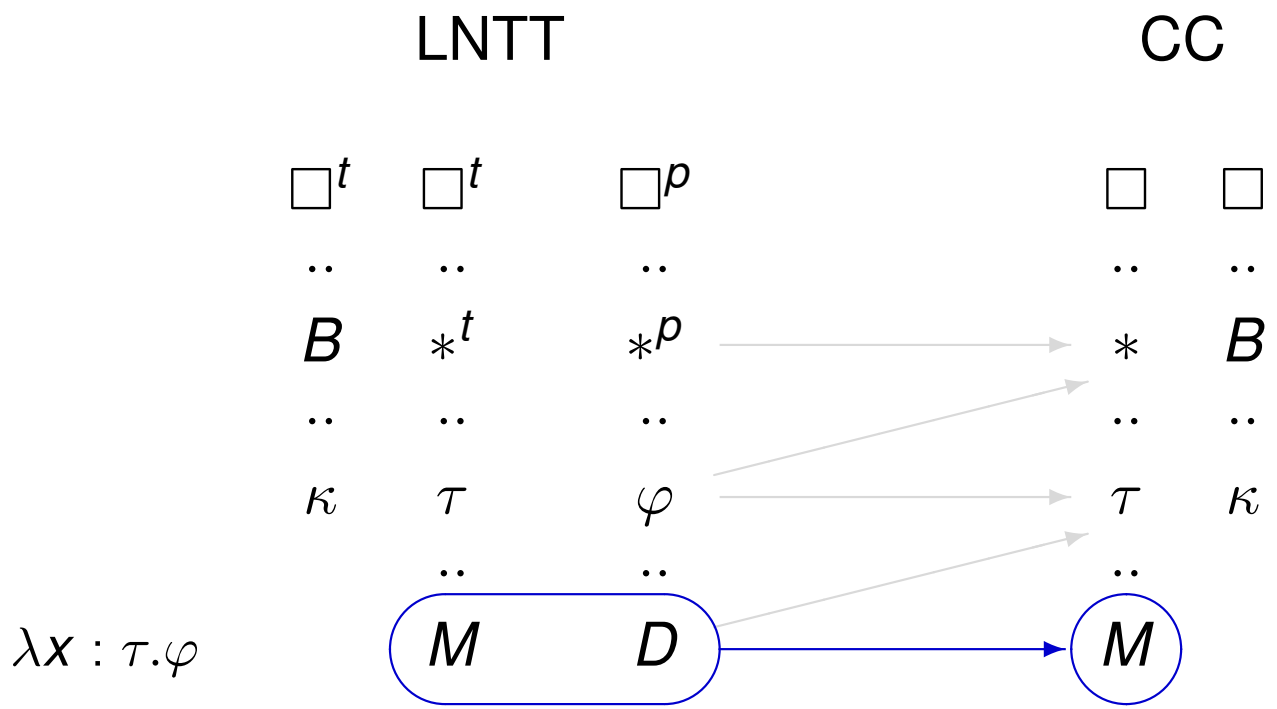
Part 2: proofs



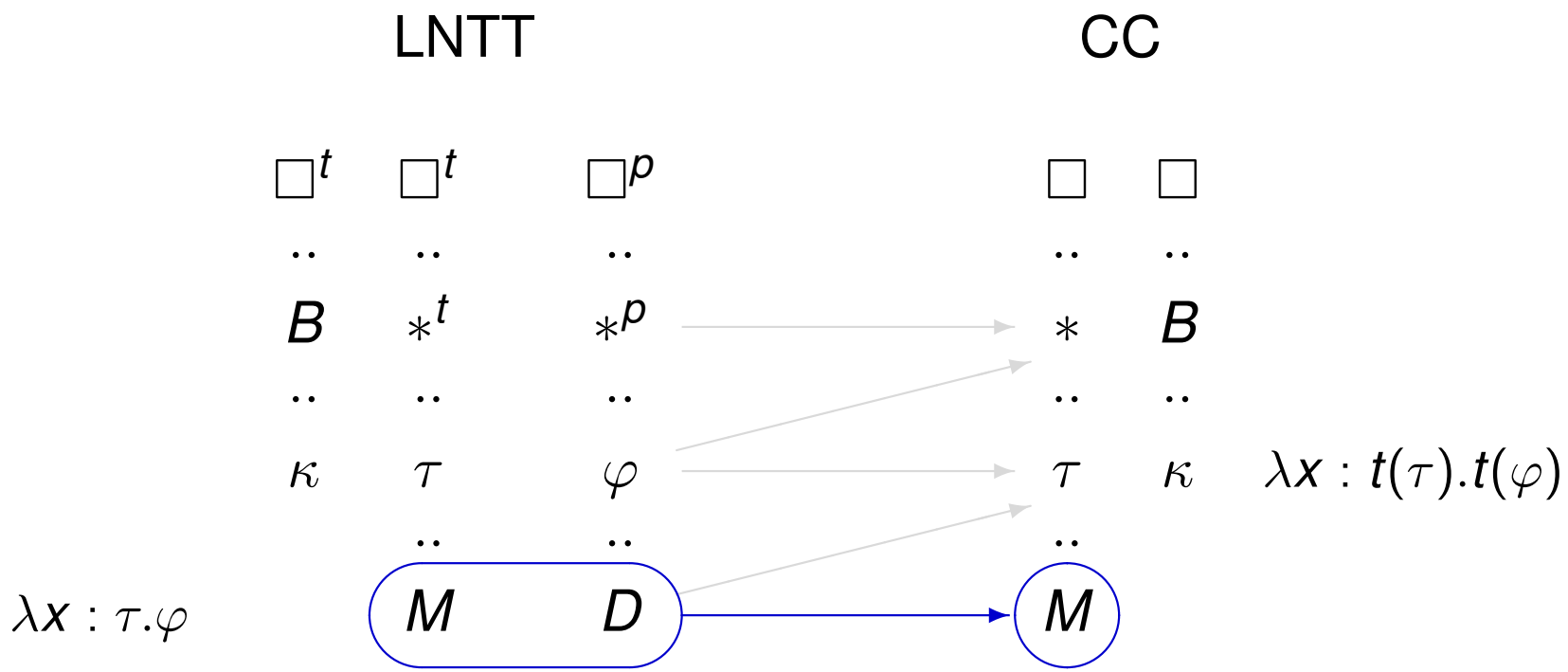
Part 2: proofs



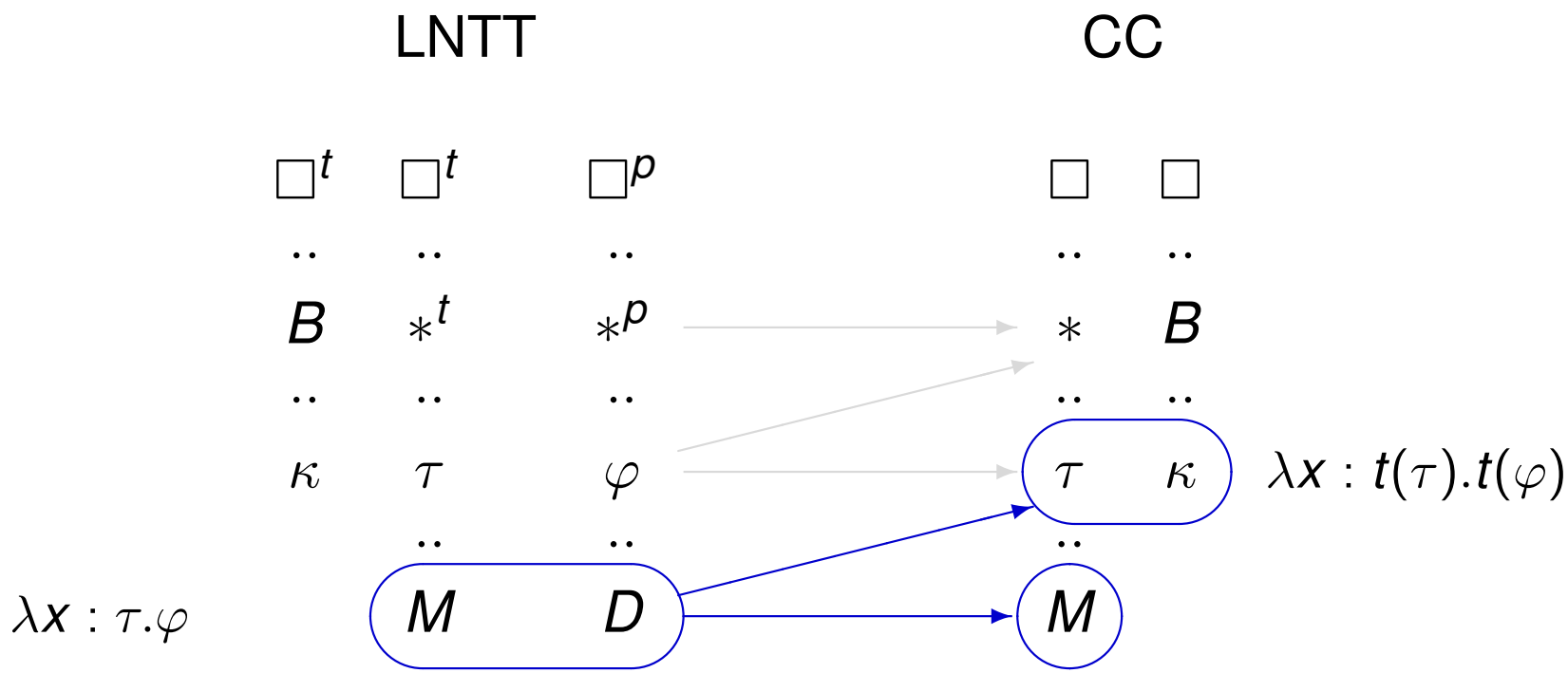
Part 2: proofs



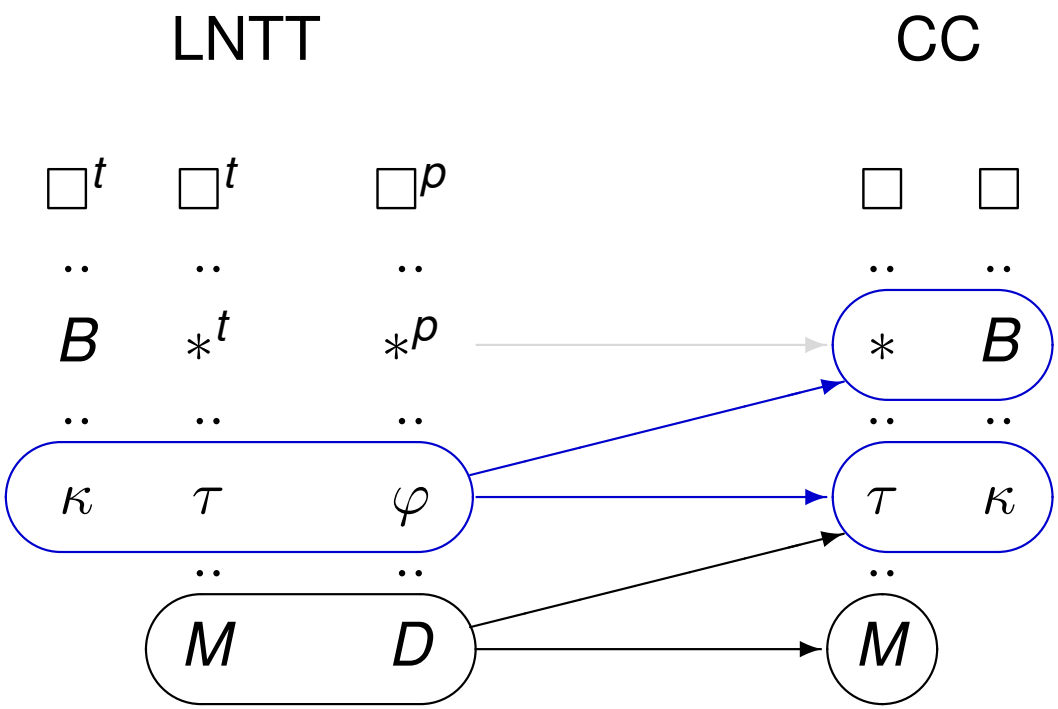
Part 2: proofs



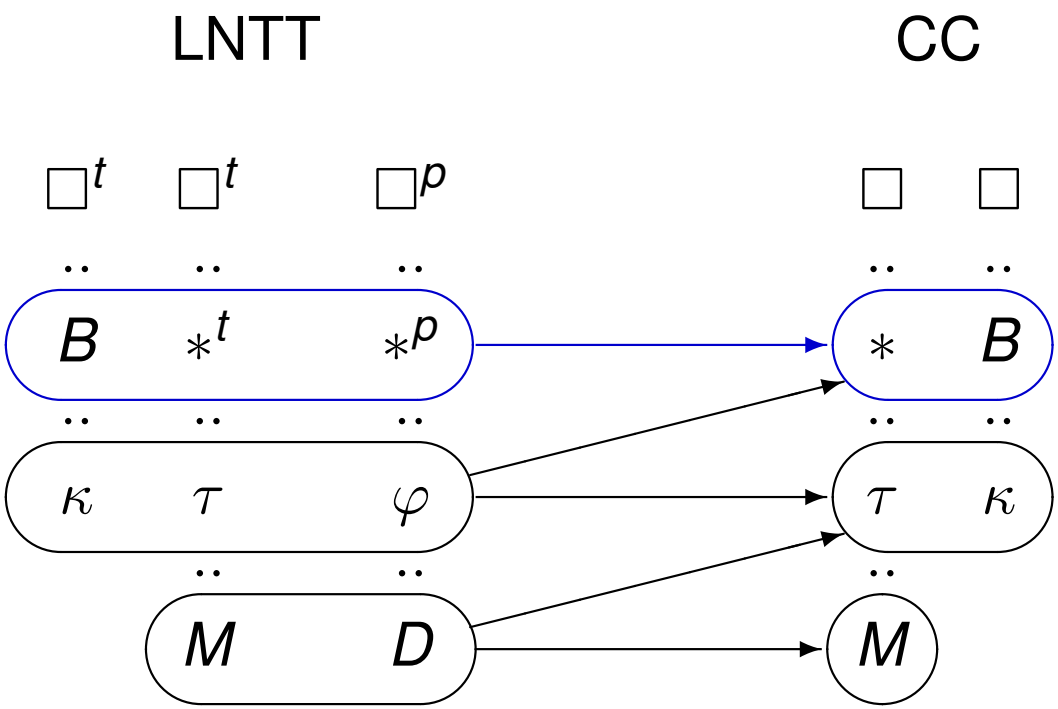
Part 2: proofs



Part 2: proofs



Part 2: proofs



Inductive types

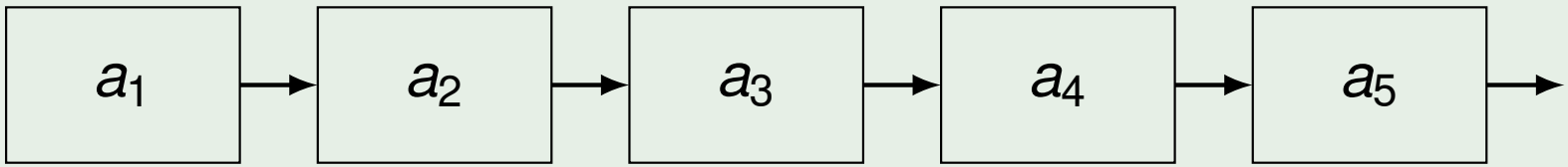
- adding datatypes (natural numbers, lists, graphs, ...)
- inductive types and inductive predicates
- usual positivity condition
- small dependent elimination on inductive types
- small non-dependent elimination on inductive predicates

Example

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

Example (Lists)

$$\begin{aligned} &\mathit{Inductive\ List} : *^{t} := \\ &\qquad \mathit{nil} : \mathit{List} \mid \\ &\qquad \mathit{cons} : A \to \mathit{List} \to \mathit{List}. \end{aligned}$$

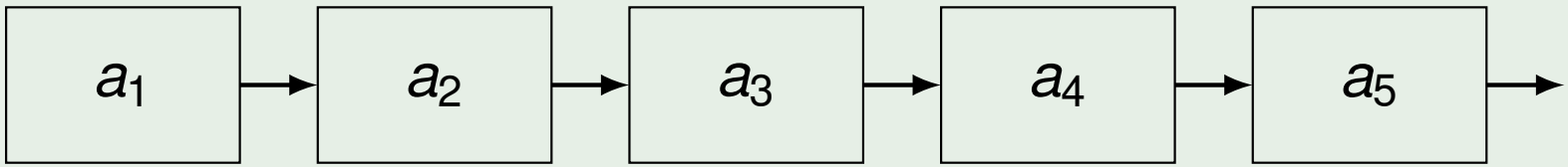


Example

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

Example (Lists)

$$\begin{aligned} &\mathit{Inductive\ List} : *^t := \\ &\qquad \mathit{nil} : \mathit{List} \mid \\ &\qquad \mathit{cons} : A \to \mathit{List} \to \mathit{List}. \end{aligned}$$



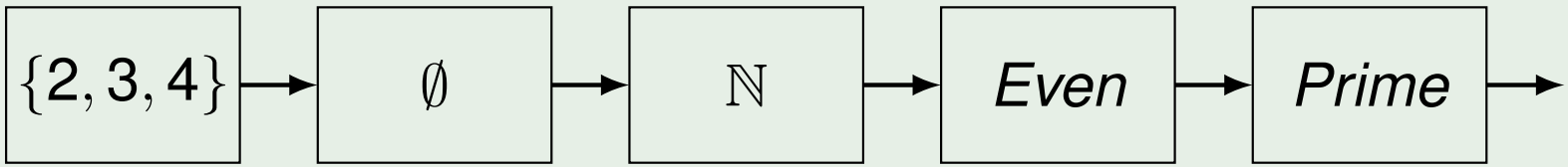
This also works if $A = \tau \to *^p$!

Example

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

Example (Lists)

$$\begin{aligned} &\mathit{Inductive\ List} : *^t := \\ &\qquad \mathit{nil} : \mathit{List} \mid \\ &\qquad \mathit{cons} : A \to \mathit{List} \to \mathit{List}. \end{aligned}$$

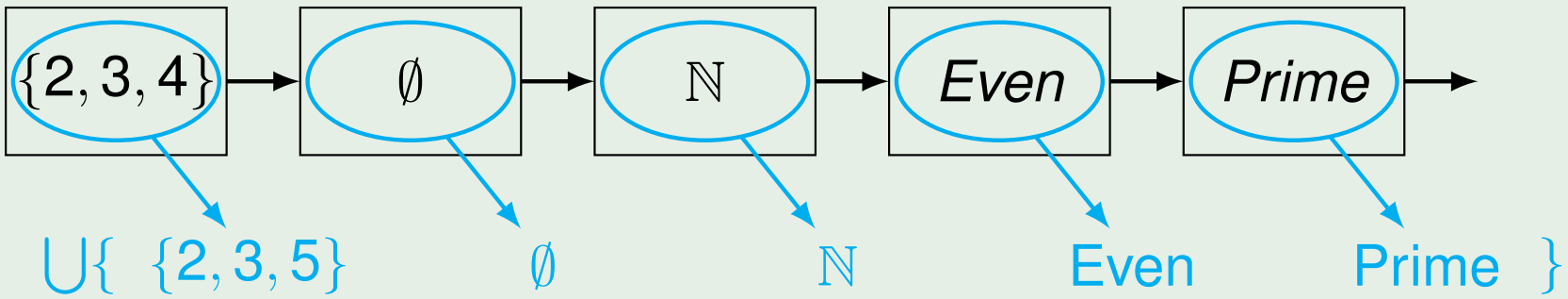


This also works if $A = \tau \to *^p$!

Example

Example (Lists)

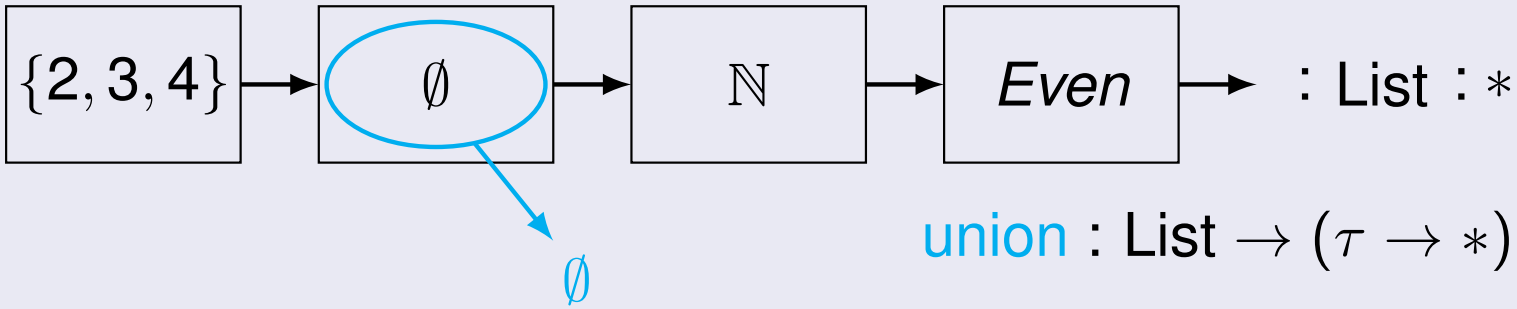
$$\begin{aligned}&\text{fun } \mathit{union} : \mathit{List} \to (\tau \to *^p) := \\ &\qquad \lambda l : \mathit{List}.\text{match } l \text{ with} \\ &\qquad\qquad \mathit{nil} : \mathit{Empty} \mid \\ &\qquad\qquad \mathit{cons}\ X\ t \Rightarrow X \cup (\mathit{union}\ t).\end{aligned}$$



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

Comparison with Coq – we can do more!

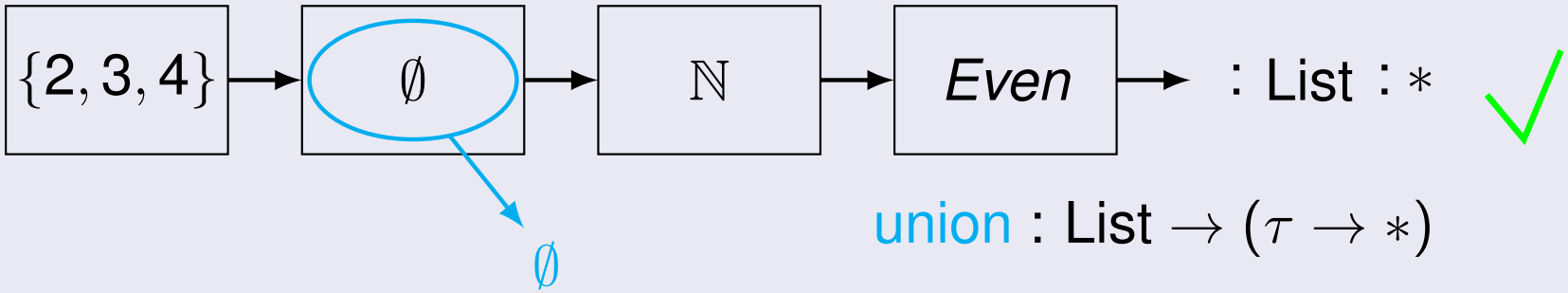
Coq with impredicative *Set*



Coq with predicative *Set*

Comparison with Coq – we can do more!

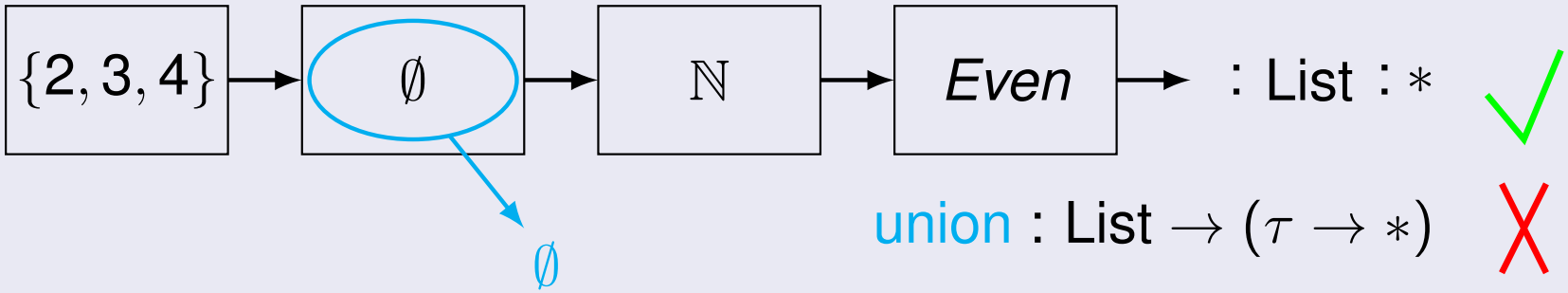
Coq with impredicative *Set*



Coq with predicative *Set*

Comparison with Coq – we can do more!

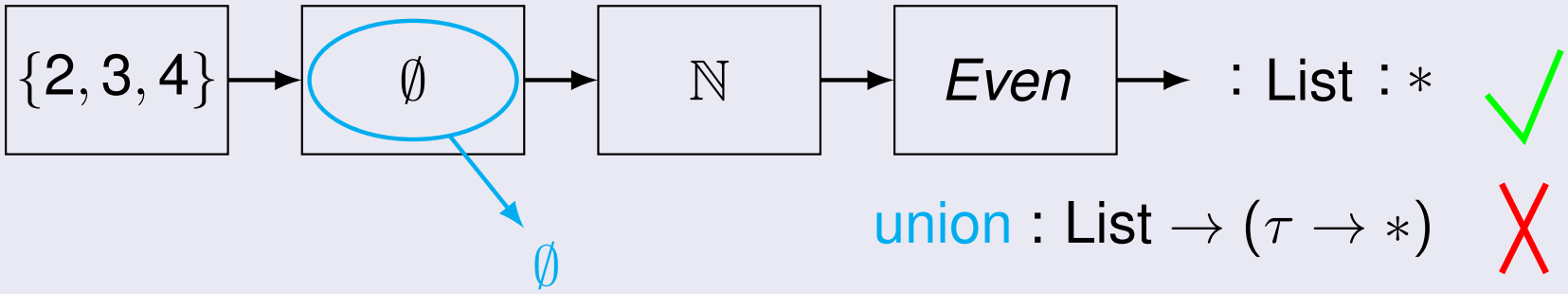
Coq with impredicative *Set*



Coq with predicative *Set*

Comparison with Coq – we can do more!

Coq with impredicative *Set*

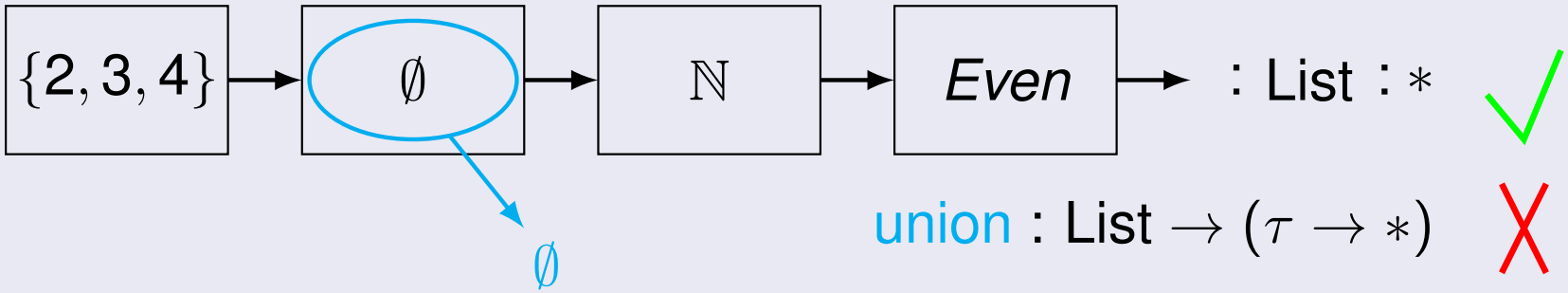


Large elimination on large inductive type

Coq with predicative *Set*

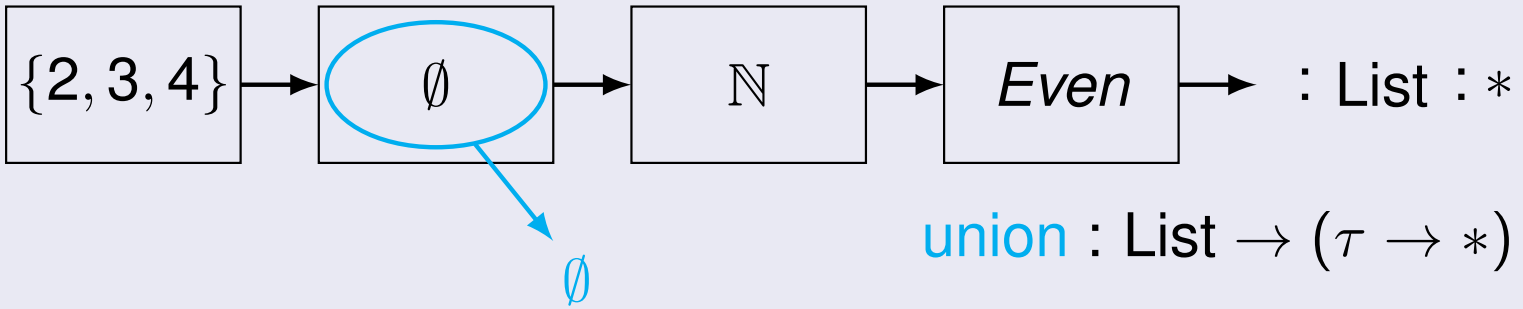
Comparison with Coq – we can do more!

Coq with impredicative *Set*



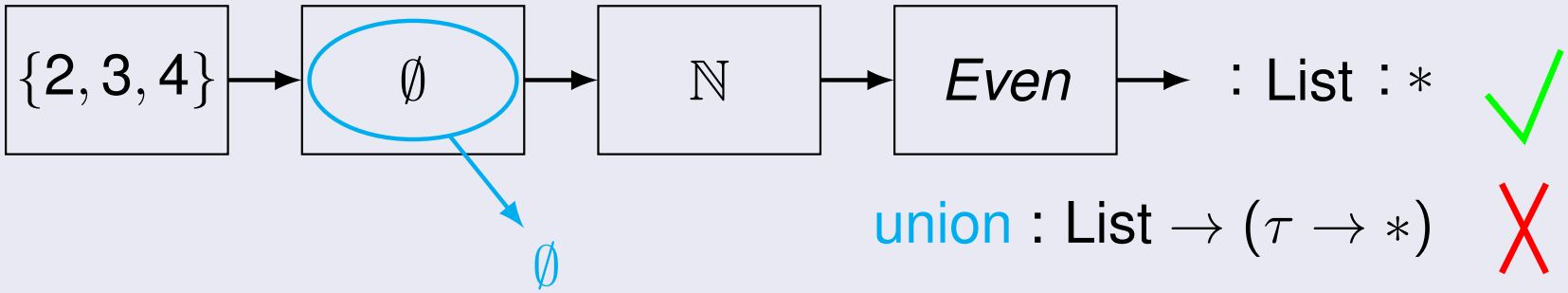
Large elimination on large inductive type

Coq with predicative *Set*



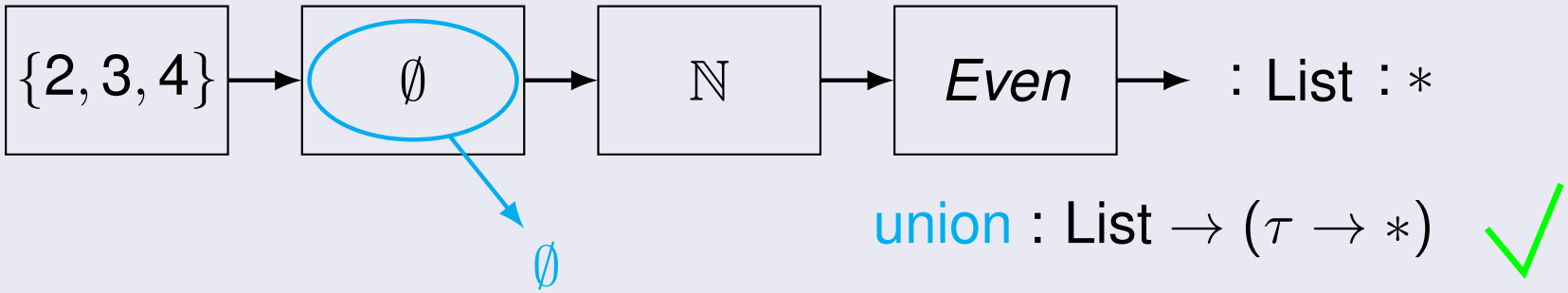
Comparison with Coq – we can do more!

Coq with impredicative *Set*



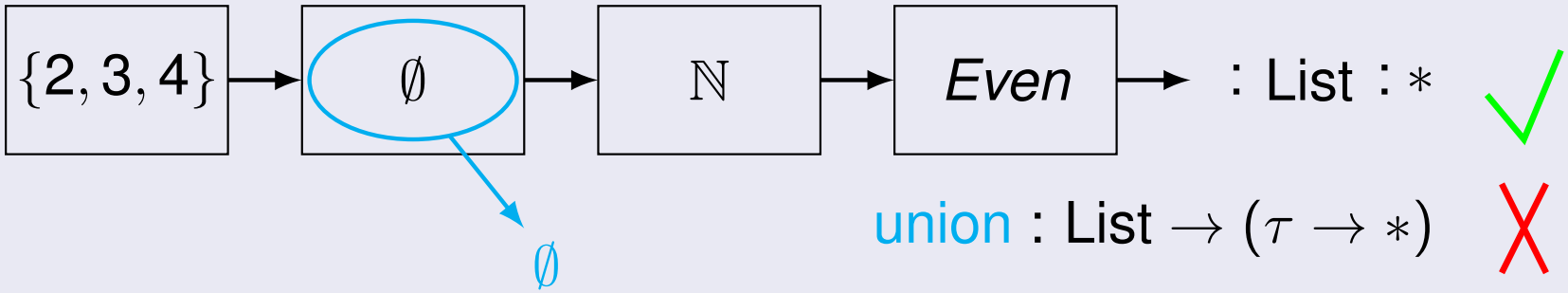
Large elimination on large inductive type

Coq with predicative *Set*



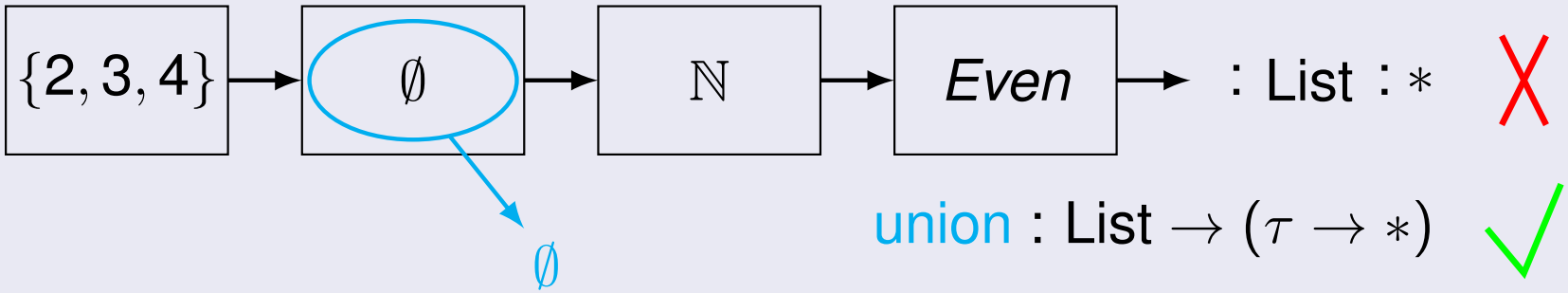
Comparison with Coq – we can do more!

Coq with impredicative *Set*



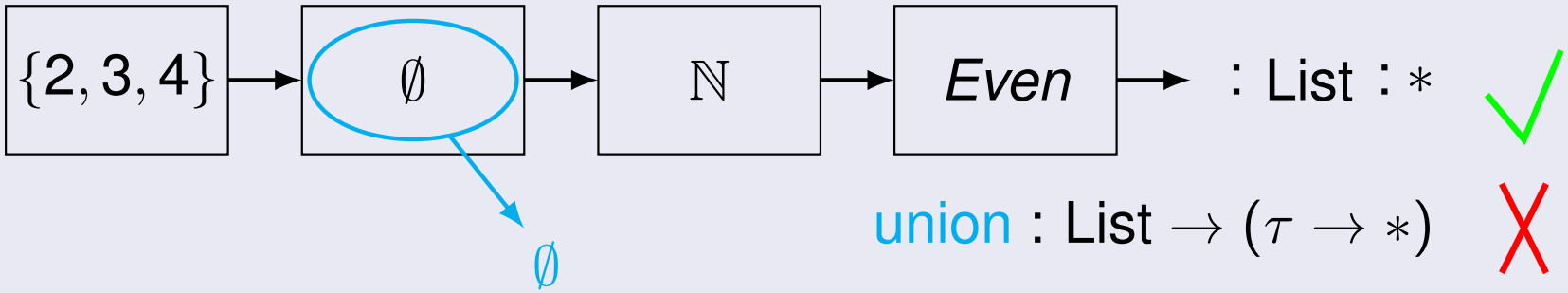
Large elimination on large inductive type

Coq with predicative *Set*



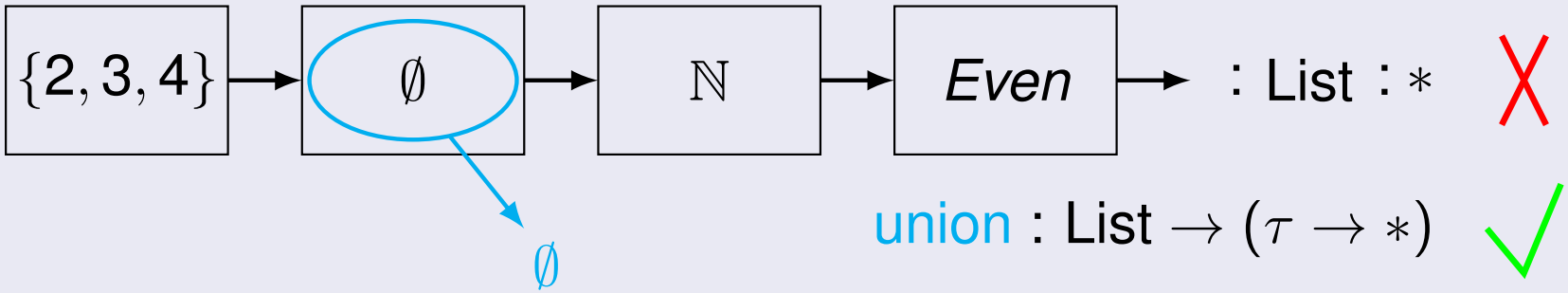
Comparison with Coq – we can do more!

Coq with impredicative *Set*



Large elimination on large inductive type

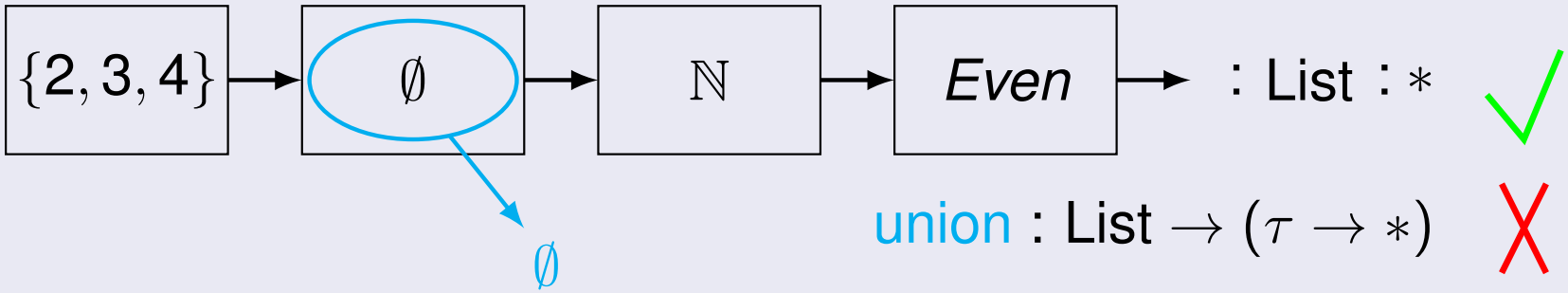
Coq with predicative *Set*



Predicativity of *Set*

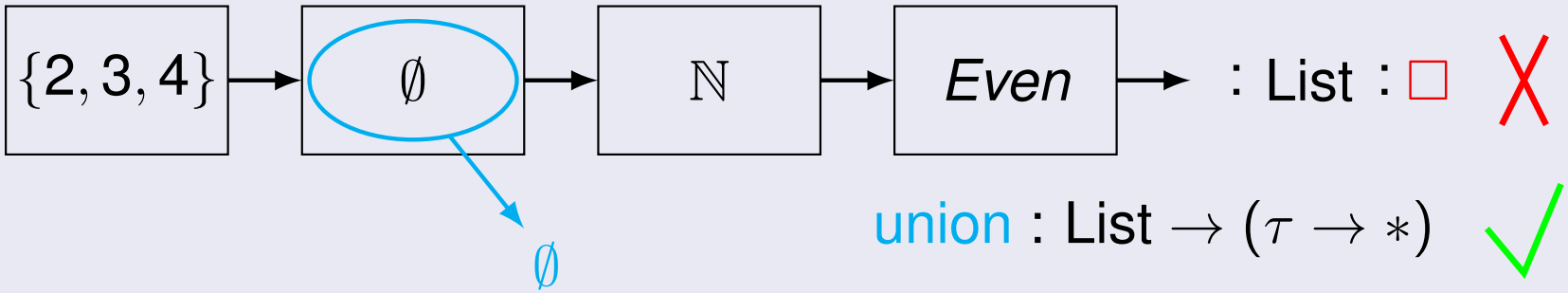
Comparison with Coq – we can do more!

Coq with impredicative *Set*



Large elimination on large inductive type

Coq with predicative *Set*

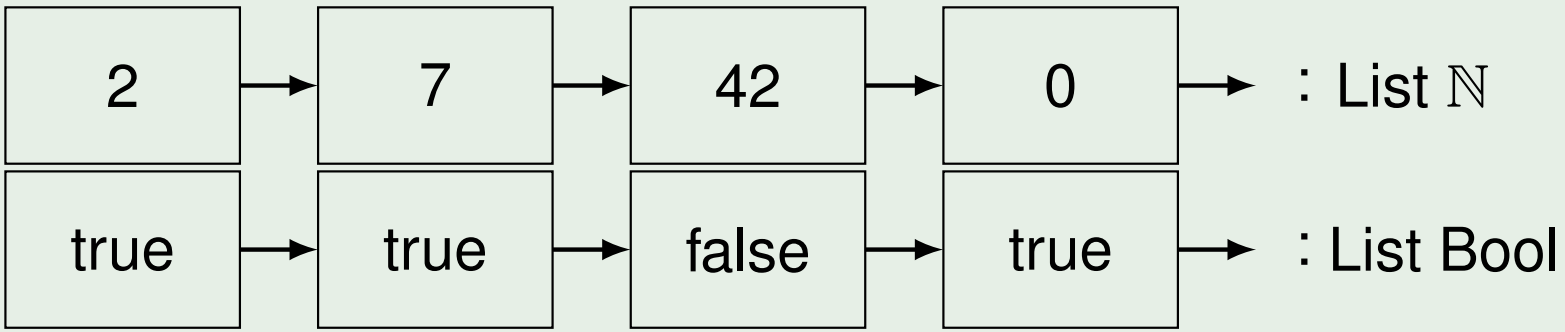


Predicativity of *Set*

Comparison with Coq – we can do less :(

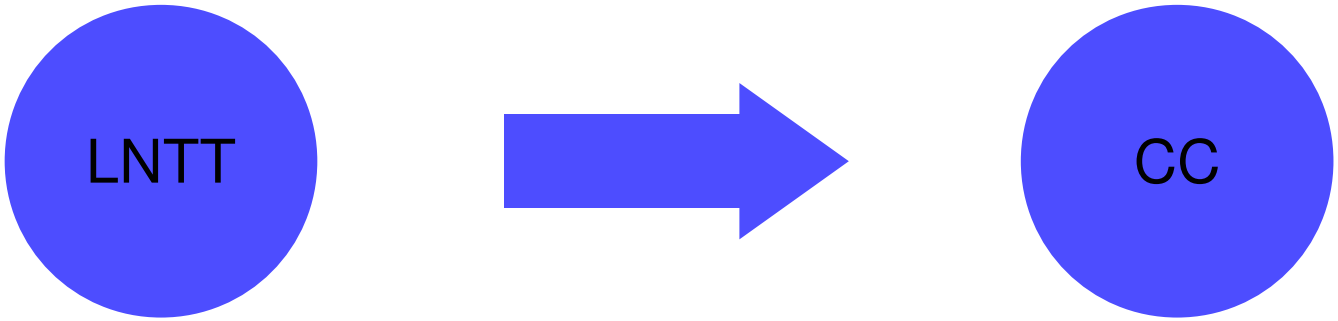
Example (Polymorphic lists)

$$\mathit{Inductive}\ \mathit{List}(A : *^t) : *^t :=$$
$$\mathit{nil} : \mathit{List}\ A \mid$$
$$\mathit{cons} : A \to \mathit{List}\ A \to \mathit{List}\ A.$$

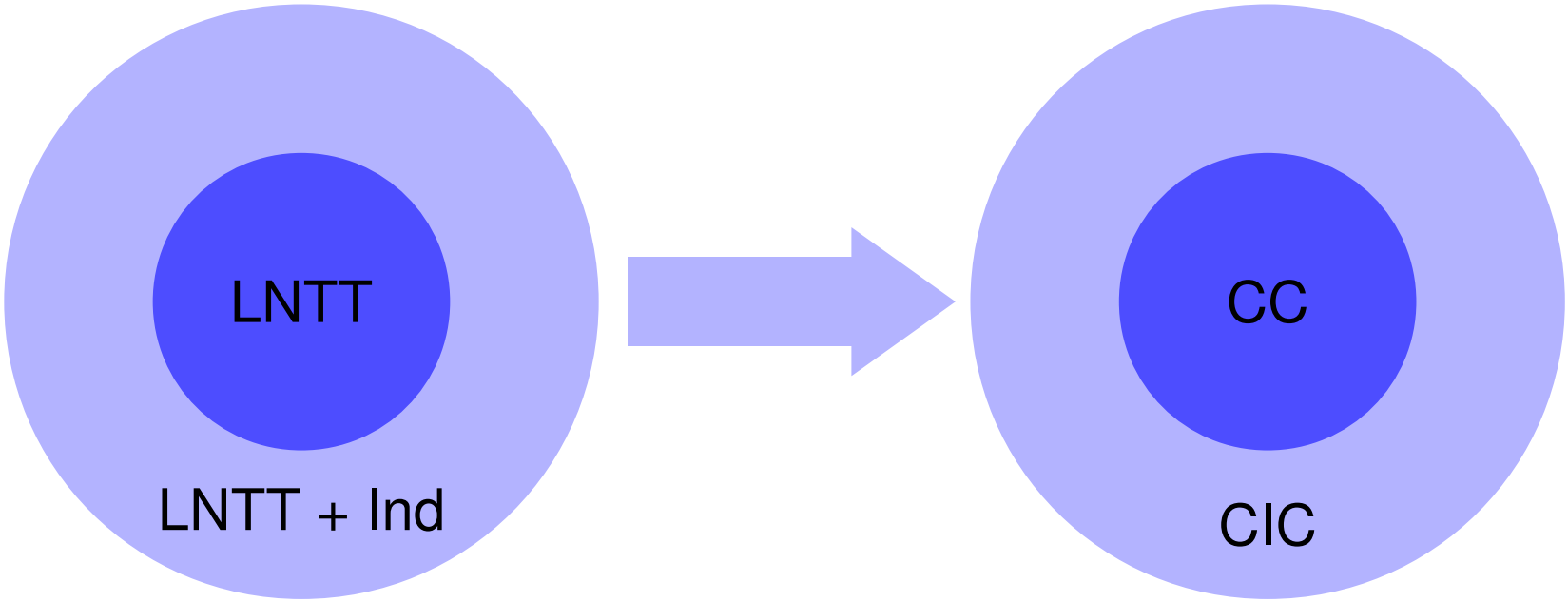


This requires type polymorphism.

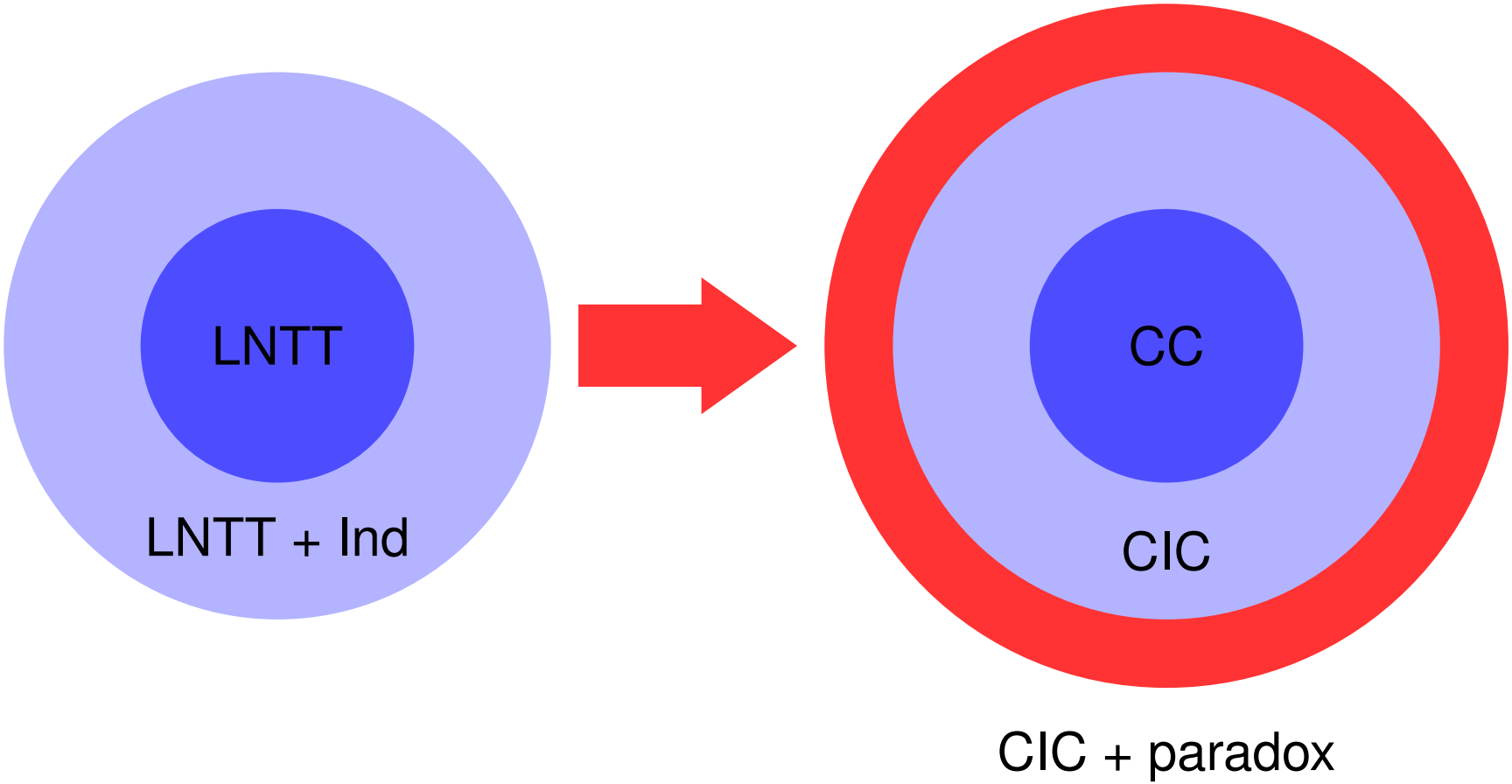
Consistency of LNTT + Ind – first attempt



Consistency of LNTT + Ind – first attempt



Consistency of LNTT + Ind – first attempt

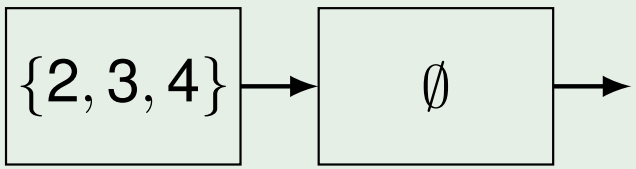


The translation does not work

LNTT + Ind

Lists

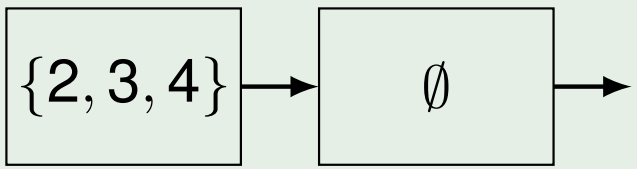
$$\mathit{Inductive\ List} : *^{t} :=$$
$$\mathit{nil} : \mathit{List} \mid$$
$$\mathit{cons} : (\tau \to *^{p}) \to \mathit{List} \to \mathit{List}.$$



CIC

Lists

$$\mathit{Inductive\ List} : * :=$$
$$\mathit{nil} : \mathit{List} \mid$$
$$\mathit{cons} : (\tau \to *) \to \mathit{List} \to \mathit{List}.$$

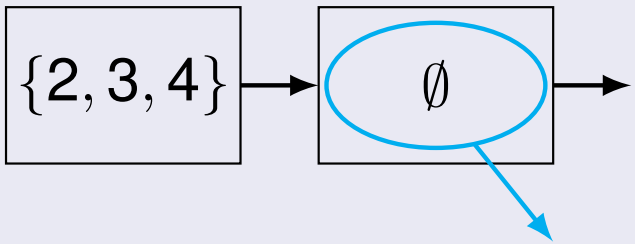


The translation does not work

LNTT + Ind

Lists

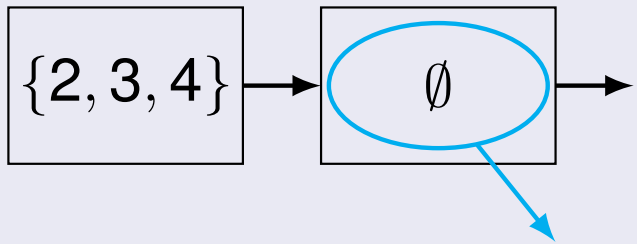
$$\text{fun } \mathit{union} : \mathit{List} \to (\tau \to *^{p}) :=$$
$$\lambda l : \mathit{List}.\text{match } l \text{ with}$$
$$\mathit{nil} : \mathit{Empty} \mid$$
$$\mathit{cons}\ X\ t \Rightarrow X \cup (\mathit{union}\ t).$$



CIC

Lists

$$\text{fun } \mathit{union} : \mathit{List} \to (\tau \to *) :=$$
$$\lambda l : \mathit{List}.\text{match } l \text{ with}$$
$$\mathit{nil} : \mathit{Empty} \mid$$
$$\mathit{cons}\ X\ t \Rightarrow X \cup (\mathit{union}\ t).$$

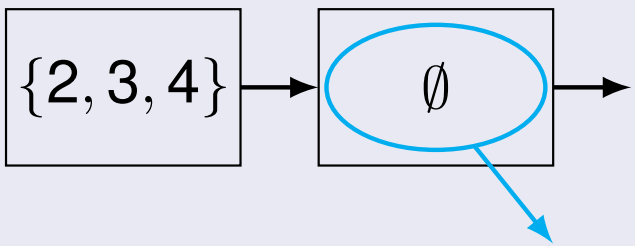


The translation does not work

LNTT + Ind

Lists

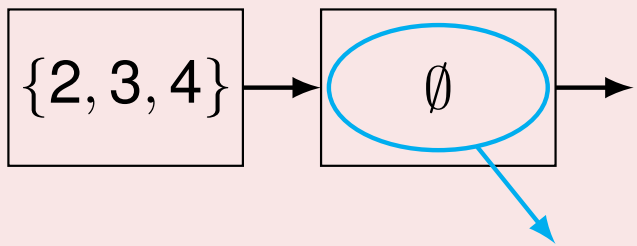
$$\text{fun } \mathit{union} : \mathit{List} \to (\tau \to *^{p}) :=$$
$$\lambda l : \mathit{List}.\text{match } l \text{ with}$$
$$\mathit{nil} : \mathit{Empty} \mid$$
$$\mathit{cons}\ X\ t \Rightarrow X \cup (\mathit{union}\ t).$$



CIC

Lists

$$\text{fun } \mathit{union} : \mathit{List} \to (\tau \to *) :=$$
$$\lambda l : \mathit{List}.\text{match } l \text{ with}$$
$$\mathit{nil} : \mathit{Empty} \mid$$
$$\mathit{cons}\ X\ t \Rightarrow X \cup (\mathit{union}\ t).$$



The strong normalization property

- proof by the method of candidates
- typed candidates

$$\{M \mid \ldots\}$$

- 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

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

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

The strong normalization property

- interpretation for inductive type

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

- interpretation for inductive objects

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

  the interpretation is used for conversion rule
- interpretation for elimination terms

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

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

Main result

Theorem

Less Naive Type Theory with inductive types is strongly normalizing.

Corollary

Less Naive Type Theory with inductive types is consistent.

Thank you!