# Towards a translation of Weak Type Theory into Dependent Type Theory

Georgi Jojgov

G.I.Jojgov@tue.nl

(work in progress)

TYPES 2004

Jouy-en-Josas, 18 December 2004

# Motivation

Study the process of formalizing of informal math texts

- (Input) Informal math text in CML (e.g. English)
- (Output) Formalization in a proof assistant

We want as much as possible:

- Reliability
- Usability by Humans
- Computer Assistance

# Motivation

Phased formalization:

Start with very weak correctness criterion
(To reduce the amount of work on the informal level)

Define one or more intermediate steps that require increasingly stronger correctness to be demonstrated.

The last stage is a completely formalized proof in a proof assistant.

# Overview

- Phased Formalization using Weak Type Theory
- Translating Weak Type Theory into Type Theory with open terms.
- Examples and Problems

# Phased Formalization using Weak Type Theory

# What is Weak Type Theory

Formal language in the Automath tradition

- Identify structure: definitions, statements, etc.
- Rich in math primitives

$$\lambda x : A.M;\ \sum_{x:A} f(x);\ \iota x : A.\phi(x);\ \{x : A | \phi(x)\} \ldots$$

- No built-in logic
- Enforces only linguistic correctness via weak types:

```
adjective, noun, term, set, statement
```

# Examples

`adjective`

$Adj_{x:\mathsf{nat}}.(0 < x) \rightarrow$ "positive"

$Adj_{x:\mathsf{nat}}.(\exists_{n:\mathsf{nat}}.2n = x) \rightarrow$ "even"

`statement`

$\forall k : Positive\ \mathsf{nat}.((2^k - 1)\,\mathsf{is}\,prime) \Rightarrow (2^{k-1}(2^k - 1)\,\mathsf{is}\,perfect)$

`term` $\sqrt{b^2 - 4ac}$, $\lambda x : \mathsf{nat}.x + x$

`noun`

$Noun_{x:\mathsf{nat}}\ prime(n) \rightarrow$ "a prime number"

$positive\ even\ \mathsf{nat} \rightarrow$ "a positive even number"

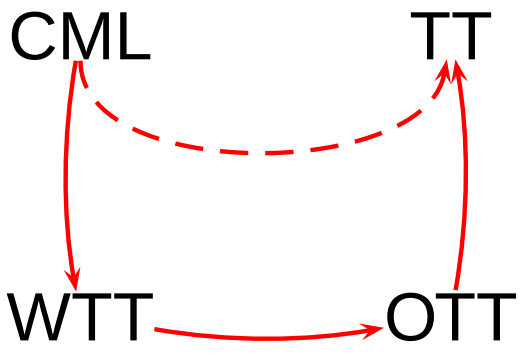
# Weak Type Theory

## The Language

| | | | |
|---|---|---|---|
| term | $\mathcal{T}$ | $::=$ | $x \mid c(\vec{\mathcal{P}}) \mid \lambda_{\mathcal{Z}}\mathcal{T} \mid \ldots$ |
| adjective | $\mathcal{A}$ | $::=$ | $c(\vec{\mathcal{P}}) \mid Adj_{\mathcal{Z}}(\mathcal{S}_p) \mid \ldots$ |
| noun | $\mathcal{N}$ | $::=$ | $c(\vec{\mathcal{P}}) \mid Noun_{\mathcal{Z}}(\mathcal{S}_p) \mid Abst_{\mathcal{Z}}(\mathcal{T}/\mathcal{N}/\mathcal{S}_s) \mid \mathcal{A}\mathcal{N} \mid S_s\downarrow \mid \ldots$ |
| set | $\mathcal{S}_s$ | $::=$ | $x \mid c(\vec{\mathcal{P}}) \mid Set_{\mathcal{Z}}(\mathcal{S}_p) \mid \mathcal{N}\uparrow \mid \ldots$ |
| statement | $\mathcal{S}_p$ | $::=$ | $x \mid c(\vec{\mathcal{P}}) \mid \mathcal{S}_p \to \mathcal{S}_p \mid \forall_{\mathcal{Z}}(\mathcal{S}_p) \mid \exists_{\mathcal{Z}}(\mathcal{S}_p) \mid \mathcal{T}\ \mathsf{is}\ \mathcal{A} \mid \mathcal{T}\ \mathsf{is}\ \mathcal{N} \mid \ldots$ |
| binder | $\mathcal{B}$ | $::=$ | $B_{\mathcal{Z}}.(\mathcal{T}/\mathcal{A}/\mathcal{N}/\mathcal{S}_s/\mathcal{S}_p)$ |
| argument | $\mathcal{P}$ | $::=$ | $\mathcal{S}_s \mid \mathcal{S}_p \mid \mathcal{T}$ |
| declaration | $\mathcal{Z}$ | $::=$ | $x{:}\mathsf{SET} \mid x{:}\mathsf{Prop} \mid x{:}\mathcal{N} \mid x{:}S_s$ |
| context | $\Gamma$ | $::=$ | $\emptyset \mid \Gamma, \mathcal{Z} \mid \Gamma, \mathcal{S}_p$ |
| definition | $\mathcal{D}$ | $::=$ | $c(\vec{x}) := (\mathcal{T}/\mathcal{S}_p/\mathcal{S}_s/\mathcal{A}/\mathcal{N})$ |
| line | $l$ | $::=$ | $\Gamma \rhd \mathcal{D} \mid \Gamma \rhd \mathcal{S}_p$ |
| book | $\mathcal{B}$ | $::=$ | $\emptyset \mid \mathcal{B} \circ l$ |

# The Formalization Path

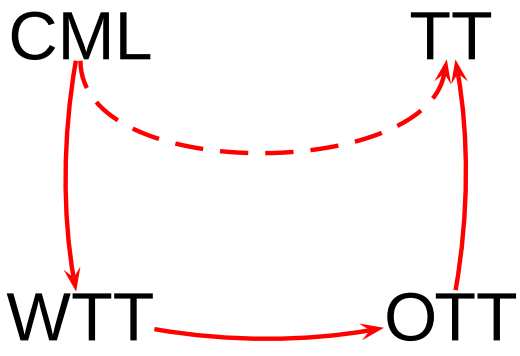
# The Formalization Path




TT: Type Theory

OTT: TT with Open terms

# The Formalization Path

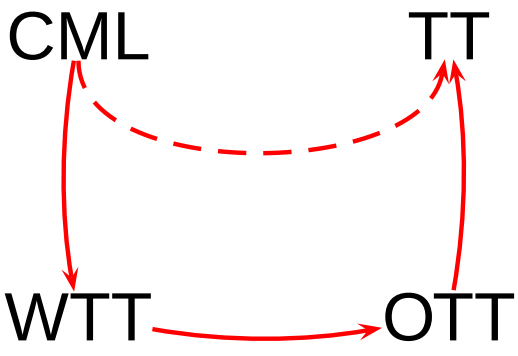



TT: Type Theory

OTT: TT with Open terms

- Write down the informal text into the formal language, considering only very weak (linguistic) correctness criteria

# The Formalization Path

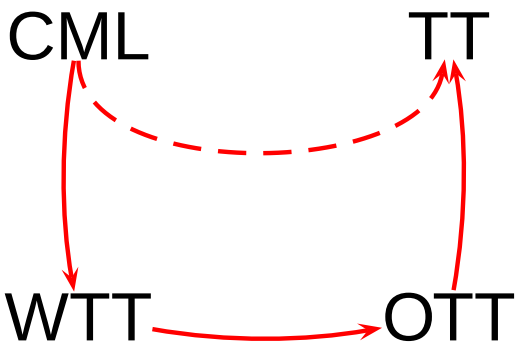



TT: Type Theory

OTT: TT with Open terms

- Write down the informal text into the formal language, considering only very weak (linguistic) correctness criteria
- Give "semantics" of the formal text. Here we make most of the "formalization" choices.

# The Formalization Path

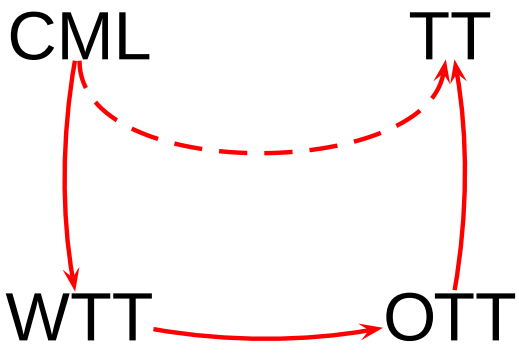



TT: Type Theory

OTT: TT with Open terms

- Write down the informal text into the formal language, considering only very weak (linguistic) correctness criteria
- Give "semantics" of the formal text. Here we make most of the "formalization" choices.
- Fill in implicit/missing details

# Benefits




Reliability

Added value to the user

- Remain close to the original
- Getting faster "to the point"
- Opportunities for automation/checks/feedback
- Complements existing technologies
- Library of "models" of WTT

# Translating WTT into Type Theory with Open Terms

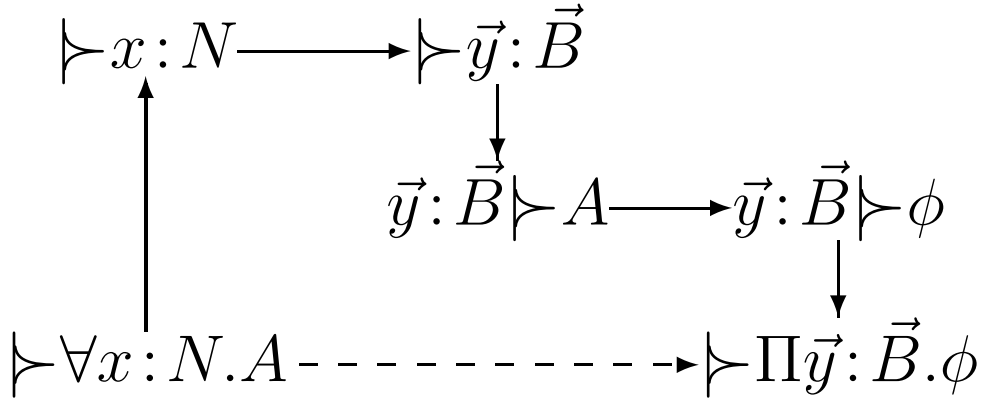
# Translating Books and Lines

Books, lines:

$$\mathcal{B} := \emptyset \;\mid\; \mathcal{B}; \Delta \rhd c(\vec{x}) := M \;\mid\; \mathcal{B}; \Delta \rhd \psi$$

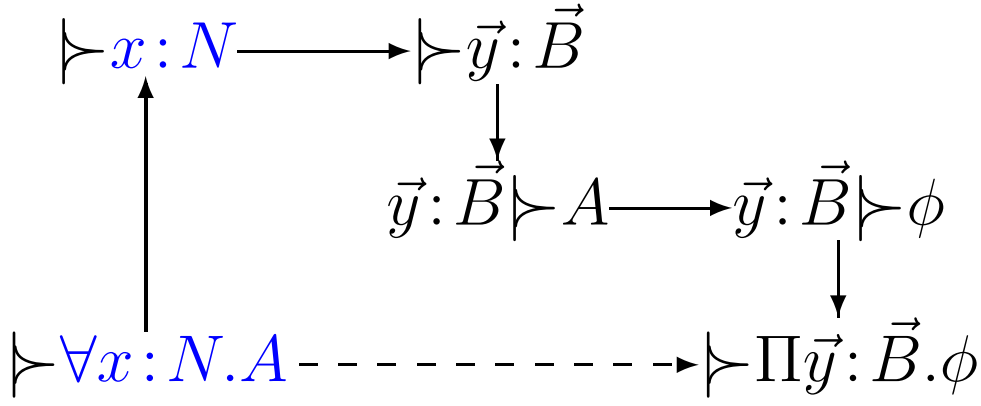
Books are mapped into TT contexts containing constants, meta-variables and definitions

# Rule Diagrams



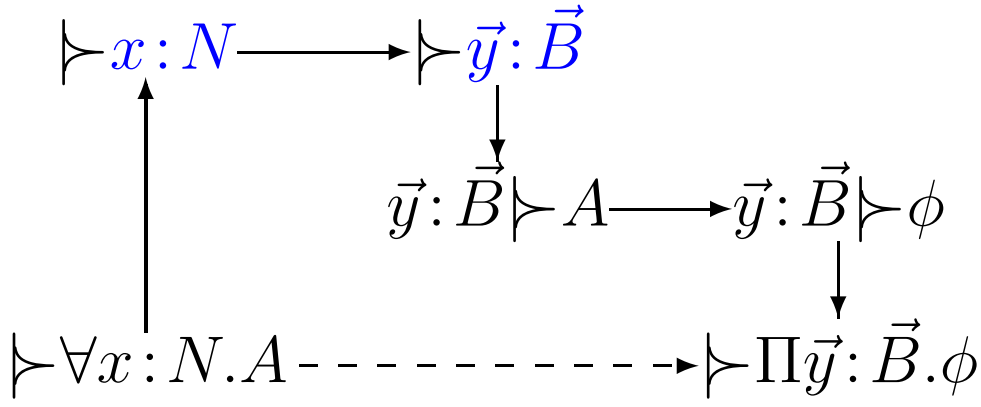

$$\frac{\Gamma_1 \vdash x : N \longrightarrow \Gamma_2 \vdash \vec{y} : \vec{B} \qquad \Gamma_2, \vec{y} : \vec{B} \vdash A \longrightarrow \Gamma_3, \vec{y} : \vec{B} \vdash \phi}{\Gamma_1 \vdash \forall x : N.A \longrightarrow \Gamma_3 \vdash \Pi \vec{y} : \vec{B}.\phi}$$

# Rule Diagrams



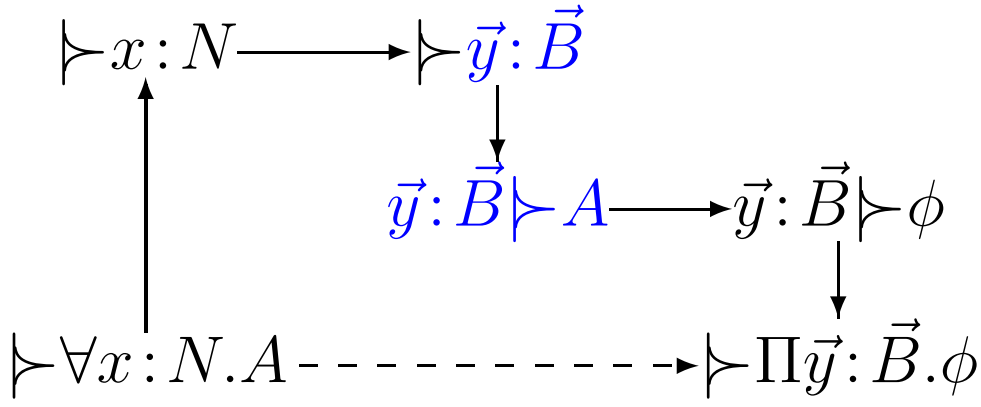

$$\frac{\Gamma_1 \vdash x : N \longrightarrow \Gamma_2 \vdash \vec{y} : \vec{B} \qquad \Gamma_2, \vec{y} : \vec{B} \vdash A \longrightarrow \Gamma_3, \vec{y} : \vec{B} \vdash \phi}{\Gamma_1 \vdash \forall x : N.A \longrightarrow \Gamma_3 \vdash \Pi \vec{y} : \vec{B}.\phi}$$

# Rule Diagrams



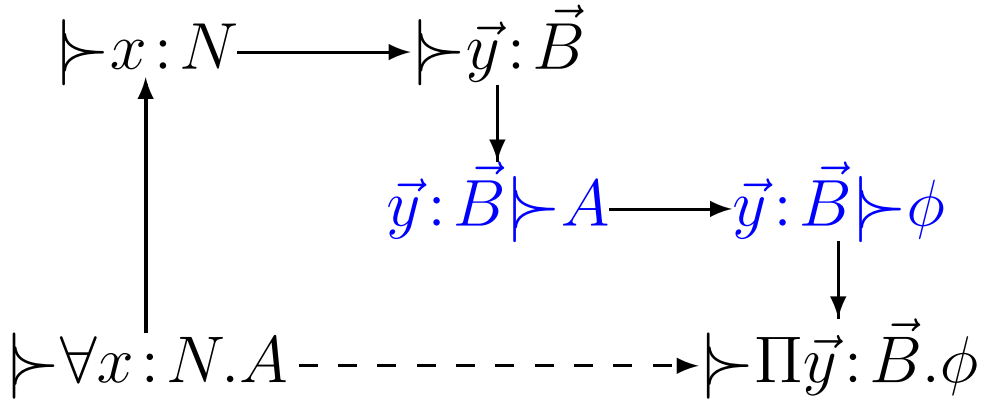

$$\frac{\Gamma_1 \vdash x : N \longrightarrow \Gamma_2 \vdash \vec{y} : \vec{B} \qquad \Gamma_2, \vec{y} : \vec{B} \vdash A \longrightarrow \Gamma_3, \vec{y} : \vec{B} \vdash \phi}{\Gamma_1 \vdash \forall x : N.A \longrightarrow \Gamma_3 \vdash \Pi \vec{y} : \vec{B}.\phi}$$

# Rule Diagrams



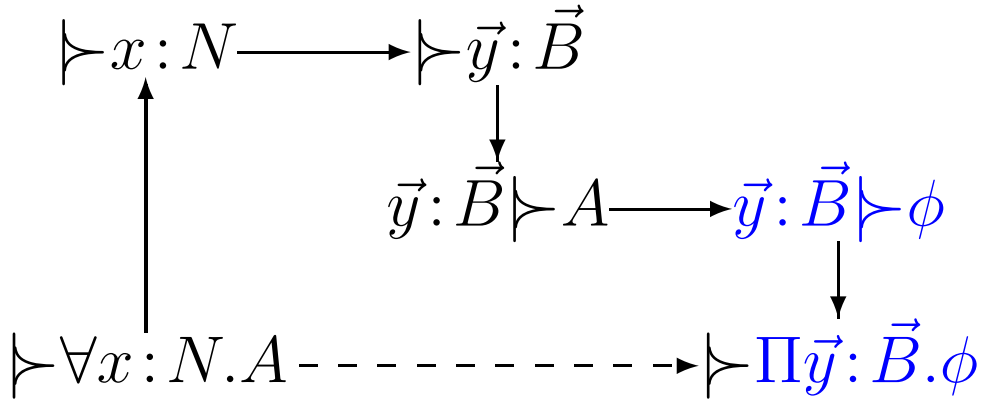

$$\frac{\Gamma_1 \vdash x : N \longrightarrow \Gamma_2 \vdash \vec{y} : \vec{B} \qquad \Gamma_2, \vec{y} : \vec{B} \vdash A \longrightarrow \Gamma_3, \vec{y} : \vec{B} \vdash \phi}{\Gamma_1 \vdash \forall x : N.A \longrightarrow \Gamma_3 \vdash \Pi \vec{y} : \vec{B}.\phi}$$

# Rule Diagrams




$$\frac{\Gamma_1 \vdash x : N \longrightarrow \Gamma_2 \vdash \vec{y} : \vec{B} \qquad \Gamma_2, \vec{y} : \vec{B} \vdash A \longrightarrow \Gamma_3, \vec{y} : \vec{B} \vdash \phi}{\Gamma_1 \vdash \forall x : N.A \longrightarrow \Gamma_3 \vdash \Pi \vec{y} : \vec{B}.\phi}$$

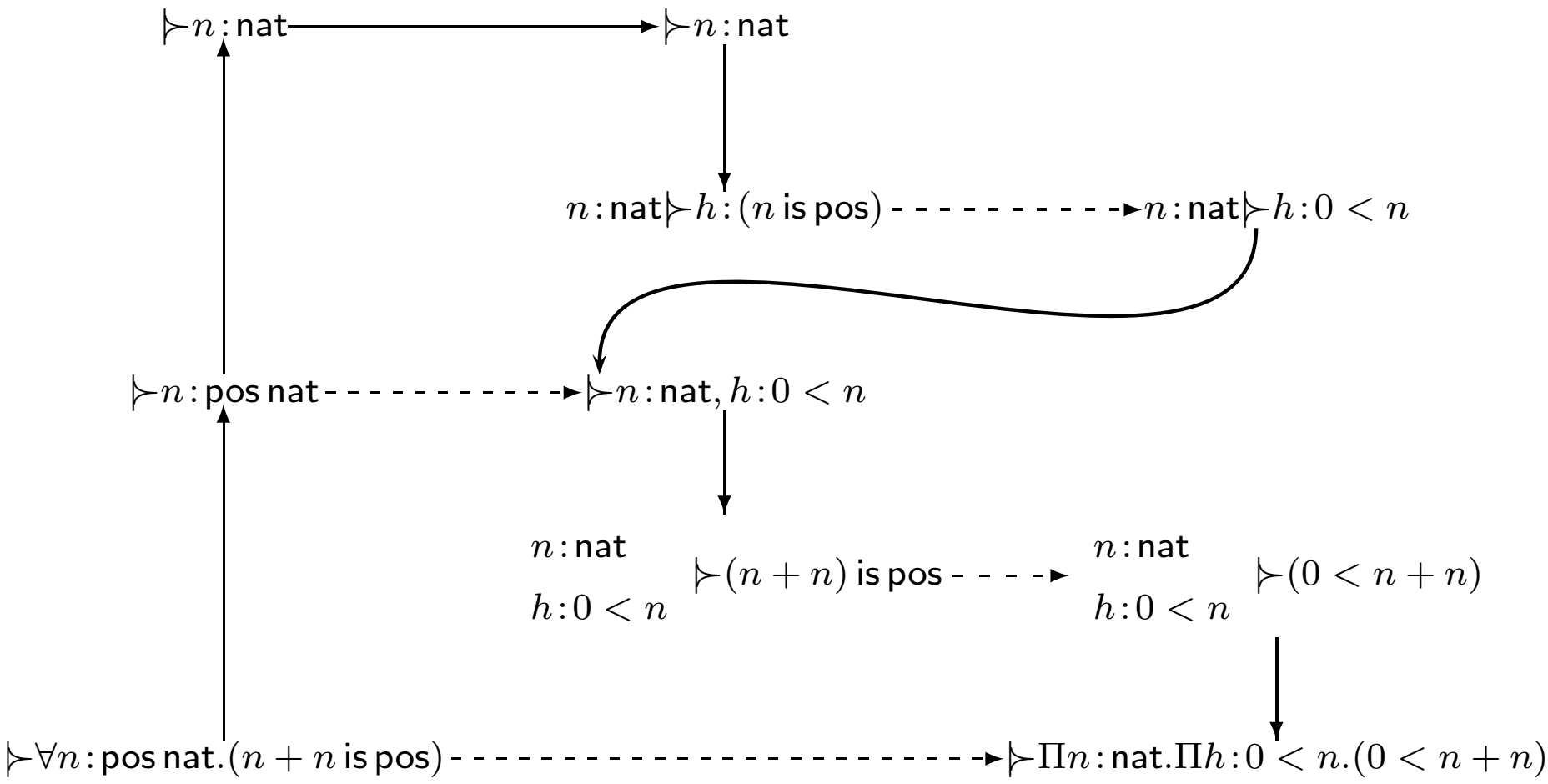
# Rule Diagrams




$$\frac{\Gamma_1 \vdash x : N \longrightarrow \Gamma_2 \vdash \vec{y} : \vec{B} \qquad \Gamma_2, \vec{y} : \vec{B} \vdash A \longrightarrow \Gamma_3, \vec{y} : \vec{B} \vdash \phi}{\Gamma_1 \vdash \forall x : N.A \longrightarrow \Gamma_3 \vdash \Pi \vec{y} : \vec{B}.\phi}$$

# Example

$\mathsf{pos} := Adj\, n : \mathsf{nat}.0 < n$



# "Demo": Export to Coq

```
|=> positive := Adj[n:nat] (lt [zero, n]).
x:positive[] nat |=> double := plus[x,x].
|:> Forall[n:positive[]~nat](double[n] is positive[]).

Definition positive := (fun n:nat=>(lt 0 n)).
Definition double:=
     fun x:nat=> fun H:(positive x)=> (plus x x).
Lemma PRF :                    Lemma STMLINE :
 forall n:nat,                  (forall n:nat,
 forall H:(positive n),         (forall H:(positive n),
 (positive n).                  (positive (double n (PRF n H))
```

# "Demo": Export to Coq

```
|=> positive := Adj[n:nat] (lt [zero, n]).
x:positive[] nat |=> double := plus[x,x].
|:> Forall[n:positive[]~nat](double[n] is positive[]).

Definition positive := (fun n:nat=>(lt 0 n)).
Definition double:=
     fun x:nat=> fun H:(positive x)=> (plus x x).
Lemma PRF :                    Lemma STMLINE :
 forall n:nat,                  (forall n:nat,
 forall H:(positive n),         (forall H:(positive n),
 (positive n).                  (positive (double n (PRF n H))
```

# "Demo": Export to Coq

```
|=> positive := Adj[n:nat] (lt [zero, n]).
x:positive[] nat |=> double := plus[x,x].
|:> Forall[n:positive[]~nat](double[n] is positive[]).

Definition positive := (fun n:nat=>(lt 0 n)).
Definition double:=
    fun x:nat=> fun H:(positive x)=> (plus x x).
Lemma PRF :                  Lemma STMLINE :
 forall n:nat,                (forall n:nat,
 forall H:(positive n),       (forall H:(positive n),
 (positive n).                (positive (double n (PRF n H))).
```

# "Demo": Export to Coq

```
x:nat |=> quad := plus[
   double[x], app[Lambda[y:nat](double[x]),x]].

Lemma PRF : forall x:nat,(positive x).
Admitted.

Lemma PRF1 :  forall x:nat, forall y:nat,(positive x).
Admitted.

Definition quad := fun x:nat=> (plus
     (double x (PRF x))
     ((fun y:nat=>(double x (PRF1 x y))) x)).
```

# Problems and Future Work

- How to define correctness criteria in general?
- One expects equal objects to be mapped to equal objects. The proof terms may break this - hence the need for unification of proof terms.
- How strong unification/matching do we need?
- Other models
- Debugging and testing the implementation with larger case study
- Modelling Meta-variables by Lemma’s and Admitted is not the best solution.

# Questions?