Strict syntax of type theory via alpha-normalisation

Viktor Bense Ambrus Kaposi Szumi Xie

Eötvös Loránd University

11 June 2024
TYPES, Copenhagen

Thanks to EuroProofNet COST Action CA20111 for funding my participation.

Extrinsic formalisation of type theory

Extrinsic formalisation of type theory

- Untyped terms (AST)

Extrinsic formalisation of type theory

- Untyped terms (AST)
- Instantiation of terms by a substitution is a recursively defined function

Extrinsic formalisation of type theory

- Untyped terms (AST)
- Instantiation of terms by a substitution is a recursively defined function
- Contexts, typing relation

Extrinsic formalisation of type theory

- Untyped terms (AST)
- Instantiation of terms by a substitution is a recursively defined function
- Contexts, typing relation
- Conversion relation

Extrinsic formalisation of type theory

- Untyped terms (AST)
- Instantiation of terms by a substitution is a recursively defined function
- Contexts, typing relation
- Conversion relation
- Consequences:
  - Substitution laws are definitional

Extrinsic formalisation of type theory

- Untyped terms (AST)
- Instantiation of terms by a substitution is a recursively defined function
- Contexts, typing relation
- Conversion relation
- Consequences:
  - Substitution laws are definitional
  - Universe à la Russell is easy

Extrinsic formalisation of type theory

- Untyped terms (AST)
- Instantiation of terms by a substitution is a recursively defined function
- Contexts, typing relation
- Conversion relation
- Consequences:
  - Substitution laws are definitional
  - Universe à la Russell is easy
  - Low level, ad-hoc choices to make about the implementation

Extrinsic formalisation of type theory

- Untyped terms (AST)
- Instantiation of terms by a substitution is a recursively defined function
- Contexts, typing relation
- Conversion relation
- Consequences:
  - Substitution laws are definitional
  - Universe à la Russell is easy
  - Low level, ad-hoc choices to make about the implementation
  - Too long and tedious to define

Intrinsic formalisation

Intrinsic formalisation

- Contexts, types

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation
- Instantiation is a constructor instead of a function

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation
- Instantiation is a constructor instead of a function
  - ```
    data T : Set
    f : T → A
    ```
    ```

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation
- Instantiation is a constructor instead of a function
  - ```
    data T : Set
    f : T → A
    ```
    ✓
    ```

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation
- Instantiation is a constructor instead of a function
  - ```
    data T : Set
    f : T → A
    ```
    ✓
  - ```
    data T : Set
    f : T → T
    ```

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation
- Instantiation is a constructor instead of a function
  - ```
    data T : Set
    f : T → A
    ```
    ✓
  - ```
    data T : Set
    f : T → T
    ```
    ?

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation
- Instantiation is a constructor instead of a function
  - ```
    data T : Set
    f : T → A
    ```
    ✓
  - ```
    data T : Set
    f : T → T
    ```
    ?
- Equality constructors for conversion rules e.g. β

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation
- Instantiation is a constructor instead of a function
  - ```
    data T : Set
    f : T → A
    ```
    ✓
  - ```
    data T : Set
    f : T → T
    ```
    ?
- Equality constructors for conversion rules e.g. β
  - No need for conversion relation

Intrinsic formalisation

- Contexts, types
- Types indexed by contexts
- Terms indexed by contexts and types
  - No need for typing relation
- Instantiation is a constructor instead of a function
  - ```
    data T : Set
    f : T → A
    ```
    ✓
  - ```
    data T : Set
    f : T → T
    ```
    ?
- Equality constructors for conversion rules e.g. β
  - No need for conversion relation
- Implementation: QIIT, initial GAT, initial CwF with extra structure

Transport hell in the syntax

Transport hell in the syntax

$$
\begin{aligned}
\Pi &: (A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ (\Gamma \triangleright A) \to \mathsf{Ty}\ \Gamma \\
\Pi[] &: \Pi\ A\ B\ [\gamma]^T \equiv \Pi\ (A[\gamma]^T)\ (B[\gamma^+]^T) \\
\\
\mathsf{lam} &: \mathsf{Tm}\ (\Gamma \triangleright A)\ B \to \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B) \\
\mathsf{lam}[] &: \mathsf{lam}\ b\ [\gamma]^t \equiv \mathsf{lam}\ (b[\gamma^+]^t)
\end{aligned}
$$

Transport hell in the syntax

$$
\begin{aligned}
\Pi \quad &: (A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ (\Gamma \triangleright A) \to \mathsf{Ty}\ \Gamma \\
\Pi[] \quad &: \Pi\ A\ B\ [\gamma]^T \equiv \Pi\ (A[\gamma]^T)\ (B[\gamma^+]^T)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{lam} \quad &: \mathsf{Tm}\ (\Gamma \triangleright A)\ B \to \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B) \\
\mathsf{lam}[] &: \underbrace{\mathsf{lam}\ b\ [\gamma]^t}_{:\ \mathsf{Tm}\ \Delta\ (\Pi\ A\ B\ [\gamma]^T)} \equiv \underbrace{\mathsf{lam}\ (b[\gamma^+]^t)}_{:\ \mathsf{Tm}\ \Delta\ (\Pi\ (A[\gamma]^T)\ (B[\gamma^+]^T))}
\end{aligned}
$$

Transport hell in the syntax

$$
\begin{aligned}
\Pi \quad &: (A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ (\Gamma \triangleright A) \to \mathsf{Ty}\ \Gamma \\
\Pi[] \quad &: \Pi\ A\ B\ [\gamma]^T \equiv \Pi\ (A[\gamma]^T)\ (B[\gamma^+]^T) \\
\\
\mathsf{lam} \quad &: \mathsf{Tm}\ (\Gamma \triangleright A)\ B \to \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B) \\
\mathsf{lam}[] \quad &: \mathsf{transport}\ (\mathsf{Tm}\ \Delta)\ \Pi[]\ (\mathsf{lam}\ b\ [\gamma]^t) \equiv \mathsf{lam}\ (b[\gamma^+]^t)
\end{aligned}
$$

Transport hell in the syntax

$$\begin{aligned}
\Pi \quad &: (A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ (\Gamma \rhd A) \to \mathsf{Ty}\ \Gamma \\
\Pi[] \quad &: \Pi\ A\ B\ [\gamma]^T \equiv \Pi\ (A[\gamma]^T)\ (B[\gamma^+]^T)
\end{aligned}$$

$$\begin{aligned}
\mathsf{lam} \quad &: \mathsf{Tm}\ (\Gamma \rhd A)\ B \to \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B) \\
\mathsf{lam}[] &: \mathsf{transport}\ (\mathsf{Tm}\ \Delta)\ \Pi[]\ (\mathsf{lam}\ b\ [\gamma]^t) \equiv \mathsf{lam}\ (b[\gamma^+]^t)
\end{aligned}$$

We want definitional substitution laws.

Transport hell in the syntax

$$
\begin{aligned}
\Pi \quad &: (A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ (\Gamma \rhd A) \to \mathsf{Ty}\ \Gamma \\
\Pi[] \quad &: \Pi\ A\ B\ [\gamma]^T \equiv \Pi\ (A[\gamma]^T)\ (B[\gamma^+]^T) \\
\\
\mathsf{lam} \quad &: \mathsf{Tm}\ (\Gamma \rhd A)\ B \to \mathsf{Tm}\ \Gamma\ (\Pi\ A\ B) \\
\mathsf{lam}[] \quad &: \mathsf{transport}\ (\mathsf{Tm}\ \Delta)\ \Pi[]\ (\mathsf{lam}\ b\ [\gamma]^t) \equiv \mathsf{lam}\ (b[\gamma^+]^t)
\end{aligned}
$$

We want definitional substitution laws.

Note: Extrinsic formalisation doesn't suffer from transport hell.

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

$$Vec\ A : \mathbb{N} \to \mathsf{Set} \ \longleftrightarrow (List\ A : \mathsf{Set}) \times (length : List\ A \to \mathbb{N})$$

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

$$\begin{aligned} Vec\ A &: \mathbb{N} \to \mathsf{Set} \longleftrightarrow (List\ A : \mathsf{Set}) \times (length : List\ A \to \mathbb{N}) \\ \mathsf{Tm} &: \mathsf{Ty} \to \mathsf{Set} \longleftrightarrow (\mathsf{Tm} : \mathsf{Set}) \times (ty : \mathsf{Tm} \to \mathsf{Ty}) \end{aligned}$$

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

$$\begin{aligned} Vec\ A &: \mathbb{N} \to \mathsf{Set} \longleftrightarrow (List\ A : \mathsf{Set}) \times (length : List\ A \to \mathbb{N}) \\ \mathsf{Tm} &: \mathsf{Ty} \to \mathsf{Set} \longleftrightarrow (\mathsf{Tm} : \mathsf{Set}) \times (ty : \mathsf{Tm} \to \mathsf{Ty}) \end{aligned}$$

(iii) Hacks like rewrite rules (Cockx, 2019), shallow embedding (Kaposi–Kovács–Kraus, 2019)

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

$$\begin{aligned} Vec\ A : \mathbb{N} \to \mathsf{Set} &\longleftrightarrow (List\ A : \mathsf{Set}) \times (length : List\ A \to \mathbb{N}) \\ \mathsf{Tm} \quad : \mathsf{Ty} \to \mathsf{Set} &\longleftrightarrow (\mathsf{Tm} \ : \mathsf{Set}) \times (ty \ : \mathsf{Tm} \to \mathsf{Ty}) \end{aligned}$$

(iii) Hacks like rewrite rules (Cockx, 2019), shallow embedding (Kaposi–Kovács–Kraus, 2019)

(iv) Higher-order abstract syntax (HOAS, LF, SOGAT) (Harper, 2021) (Bocquet–Kaposi–Sattler, 2023)

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

$$\begin{aligned} Vec\ A &: \mathbb{N} \to \mathsf{Set} \longleftrightarrow (List\ A : \mathsf{Set}) \times (length : List\ A \to \mathbb{N}) \\ \mathsf{Tm} &: \mathsf{Ty} \to \mathsf{Set} \longleftrightarrow (\mathsf{Tm}\ : \mathsf{Set}) \times (ty\ : \mathsf{Tm} \to \mathsf{Ty}) \end{aligned}$$

(iii) Hacks like rewrite rules (Cockx, 2019), shallow embedding (Kaposi–Kovács–Kraus, 2019)

(iv) Higher-order abstract syntax (HOAS, LF, SOGAT) (Harper, 2021) (Bocquet–Kaposi–Sattler, 2023)

- Proofs are internal

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

$$
\begin{aligned}
Vec\ A &: \mathbb{N} \to \mathsf{Set} \ \longleftrightarrow (List\ A : \mathsf{Set}) \times (length : List\ A \to \mathbb{N}) \\
\mathsf{Tm} &: \mathsf{Ty} \to \mathsf{Set} \longleftrightarrow (\mathsf{Tm} \ : \mathsf{Set}) \times (ty \ : \mathsf{Tm} \to \mathsf{Ty})
\end{aligned}
$$

(iii) Hacks like rewrite rules (Cockx, 2019), shallow embedding (Kaposi–Kovács–Kraus, 2019)

(iv) Higher-order abstract syntax (HOAS, LF, SOGAT) (Harper, 2021) (Bocquet–Kaposi–Sattler, 2023)

- Proofs are internal
- Metatheoretic step needed to transfer proofs to the real syntax

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

$$Vec\ A : \mathbb{N} \to \mathsf{Set} \ \longleftrightarrow (List\ A : \mathsf{Set}) \times (length : List\ A \to \mathbb{N})$$
$$\mathsf{Tm} \quad : \mathsf{Ty} \to \mathsf{Set} \longleftrightarrow (\mathsf{Tm} \ : \mathsf{Set}) \times (ty \ : \mathsf{Tm} \to \mathsf{Ty})$$

(iii) Hacks like rewrite rules (Cockx, 2019), shallow embedding (Kaposi–Kovács–Kraus, 2019)

(iv) Higher-order abstract syntax (HOAS, LF, SOGAT) (Harper, 2021) (Bocquet–Kaposi–Sattler, 2023)

- Proofs are internal
- Metatheoretic step needed to transfer proofs to the real syntax

(v) Fight through transport hell like a real man (Altenkirch–Kaposi, 2016)

Solutions to transport hell

(i) Go back to extrinsic (Abel–Öhman–Vezzosi, 2018)

(ii) Natural models: no indexing of terms by types (Awodey, 2018)(Brunerie–Boer, 2020)

$$\begin{aligned} &Vec\ A : \mathbb{N} \to \mathsf{Set} \ \longleftrightarrow (List\ A : \mathsf{Set}) \times (length : List\ A \to \mathbb{N}) \\ &\mathsf{Tm} \quad\ \ : \mathsf{Ty} \to \mathsf{Set} \longleftrightarrow (\mathsf{Tm}\ \ : \mathsf{Set}) \times (ty\ \ : \mathsf{Tm} \to \mathsf{Ty}) \end{aligned}$$

(iii) Hacks like rewrite rules (Cockx, 2019), shallow embedding (Kaposi–Kovács–Kraus, 2019)

(iv) Higher-order abstract syntax (HOAS, LF, SOGAT) (Harper, 2021) (Bocquet–Kaposi–Sattler, 2023)

- Proofs are internal
- Metatheoretic step needed to transfer proofs to the real syntax

(v) Fight through transport hell like a real man (Altenkirch–Kaposi, 2016)

We try to do (iii) without hacking.

This talk: strictification of syntax using α-normal forms

This talk: strictification of syntax using α-normal forms

$$\mathsf{data}\ \mathsf{isNf}\ :\ (\Gamma\ :\ \mathsf{Con})(A\ :\ \mathsf{Ty}\ \Gamma) \to \mathsf{Tm}\ \Gamma\ A \to \mathsf{Set}\ \mathsf{where}$$

This talk: strictification of syntax using α-normal forms

$$
\begin{aligned}
&\mathsf{data}\ \mathsf{isNf} : (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Tm}\ \Gamma\ A \to \mathsf{Set}\ \mathsf{where}\\
&\quad \mathsf{varNf} : (x : \mathsf{Var}\ \Gamma\ A) \to \mathsf{isNf}\ \Gamma\ A\ (\mathsf{var}\ x)\\
&\quad \mathsf{lamNf} : \mathsf{isNf}\ (\Gamma \triangleright A)\ B\ b \to \mathsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ (\mathsf{lam}\ b)\\
&\quad \mathsf{appNf} : \mathsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ f \to \mathsf{isNf}\ \Gamma\ A\ a \to \mathsf{isNf}\ \Gamma\ (B[\langle a \rangle])\ (app\ f\ a)
\end{aligned}
$$

This talk: strictification of syntax using α-normal forms

$$
\begin{array}{lll}
\mathsf{data\ isNf} & : & (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Tm}\ \Gamma\ A \to \mathsf{Set\ where} \\
\quad \mathsf{varNf} & : & (x : \mathsf{Var}\ \Gamma\ A) \to \mathsf{isNf}\ \Gamma\ A\ (\mathsf{var}\ x) \\
\quad \mathsf{lamNf} & : & \mathsf{isNf}\ (\Gamma \rhd A)\ B\ b \to \mathsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ (\mathsf{lam}\ b) \\
\quad \mathsf{appNf} & : & \mathsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ f \to \mathsf{isNf}\ \Gamma\ A\ a \to \mathsf{isNf}\ \Gamma\ (B[\langle a \rangle])\ (app\ f\ a) \\
\quad \mathsf{eq} & : & (u^N : \mathsf{isNf}\ \Gamma\ A\ u)(v^N : \mathsf{isNf}\ \Gamma\ A\ v) \to u \equiv v \to u^N \equiv v^N
\end{array}
$$

This talk: strictification of syntax using α-normal forms

$$
\begin{array}{lll}
\textsf{data isNf} & : & (\Gamma\ :\ \textsf{Con})(A\ :\ \textsf{Ty}\ \Gamma) \to \textsf{Tm}\ \Gamma\ A \to \textsf{Set where} \\
\quad \textsf{varNf} & : & (x\ :\ \textsf{Var}\ \Gamma\ A) \to \textsf{isNf}\ \Gamma\ A\ (\textsf{var}\ x) \\
\quad \textsf{lamNf} & : & \textsf{isNf}\ (\Gamma \rhd A)\ B\ b \to \textsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ (\textsf{lam}\ b) \\
\quad \textsf{appNf} & : & \textsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ f \to \textsf{isNf}\ \Gamma\ A\ a \to \textsf{isNf}\ \Gamma\ (B[\langle a \rangle])\ (app\ f\ a) \\
\quad \textsf{eq} & : & (u^N\ v^N : \textsf{isNf}\ \Gamma\ A\ u) \to u^N \equiv v^N
\end{array}
$$

This talk: strictification of syntax using α-normal forms

$$
\begin{aligned}
&\textsf{data isNf} : (\Gamma : \textsf{Con})(A : \textsf{Ty}\ \Gamma) \to \textsf{Tm}\ \Gamma\ A \to \textsf{Set where} \\
&\quad \textsf{varNf} : (x : \textsf{Var}\ \Gamma\ A) \to \textsf{isNf}\ \Gamma\ A\ (\textsf{var}\ x) \\
&\quad \textsf{lamNf} : \textsf{isNf}\ (\Gamma \rhd A)\ B\ b \to \textsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ (\textsf{lam}\ b) \\
&\quad \textsf{appNf} : \textsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ f \to \textsf{isNf}\ \Gamma\ A\ a \to \textsf{isNf}\ \Gamma\ (B[\langle a \rangle])\ (app\ f\ a) \\
&\quad \textsf{eq} : (u^N\ v^N : \textsf{isNf}\ \Gamma\ A\ u) \to u^N \equiv v^N
\end{aligned}
$$

$$
\textsf{norm} : (u : \textsf{Tm}\ \Gamma\ A) \to \textsf{isNf}\ \Gamma\ A\ u
$$

This talk: strictification of syntax using α-normal forms

$$
\begin{aligned}
&\text{data isNf} : (\Gamma : \text{Con})(A : \text{Ty}\ \Gamma) \to \text{Tm}\ \Gamma\ A \to \text{Set where} \\
&\quad \text{varNf} : (x : \text{Var}\ \Gamma\ A) \to \text{isNf}\ \Gamma\ A\ (\text{var}\ x) \\
&\quad \text{lamNf} : \text{isNf}\ (\Gamma \rhd A)\ B\ b \to \text{isNf}\ \Gamma\ (\Pi\ A\ B)\ (\text{lam}\ b) \\
&\quad \text{appNf} : \text{isNf}\ \Gamma\ (\Pi\ A\ B)\ f \to \text{isNf}\ \Gamma\ A\ a \to \text{isNf}\ \Gamma\ (B[\langle a \rangle])\ (app\ f\ a) \\
&\quad \text{eq} : (u^N\ v^N : \text{isNf}\ \Gamma\ A\ u) \to u^N \equiv v^N
\end{aligned}
$$

$$
\text{norm} : (u : \text{Tm}\ \Gamma\ A) \to \text{isNf}\ \Gamma\ A\ u
$$

$\_[\_]^N$ on terms using α-normalisation:
$\_[\_]^N : \text{isNf}\ \Gamma\ A\ a \to \text{isNfs}\ \Delta\ \Gamma\ \gamma \to (a' : \text{Tm}\ \Delta\ (A[\gamma]^T)) \times (a' \equiv a[\gamma]^t)$

This talk: strictification of syntax using α-normal forms

$$
\begin{aligned}
&\mathsf{data\ isNf} : (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Tm}\ \Gamma\ A \to \mathsf{Set\ where} \\
&\quad \mathsf{varNf} : (x : \mathsf{Var}\ \Gamma\ A) \to \mathsf{isNf}\ \Gamma\ A\ (\mathsf{var}\ x) \\
&\quad \mathsf{lamNf} : \mathsf{isNf}\ (\Gamma \rhd A)\ B\ b \to \mathsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ (\mathsf{lam}\ b) \\
&\quad \mathsf{appNf} : \mathsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ f \to \mathsf{isNf}\ \Gamma\ A\ a \to \mathsf{isNf}\ \Gamma\ (B[\langle a \rangle])\ (app\ f\ a) \\
&\quad \mathsf{eq} : (u^N\ v^N : \mathsf{isNf}\ \Gamma\ A\ u) \to u^N \equiv v^N
\end{aligned}
$$

$$
\mathsf{norm} : (u : \mathsf{Tm}\ \Gamma\ A) \to \mathsf{isNf}\ \Gamma\ A\ u
$$

$\_[\_]^N$ on terms using α-normalisation:

$$
\_[\_]^N : \mathsf{isNf}\ \Gamma\ A\ a \to \mathsf{isNfs}\ \Delta\ \Gamma\ \gamma \to (a' : \mathsf{Tm}\ \Delta\ (A[\gamma]^T)) \times (a' \equiv a[\gamma]^t)
$$

$$
\begin{aligned}
\_[\_] &: \mathsf{Tm}\ \Gamma\ A \to (\gamma : \mathsf{Sub}\ \Delta\ \Gamma) \to \mathsf{Tm}\ \Delta\ (A[\gamma]^T) \\
t\ [\ \gamma\ ] &:= \mathsf{fst}\ (\mathsf{norm}\ t\ [\mathsf{norm}^S\ \gamma]^N)
\end{aligned}
$$

This talk: strictification of syntax using α-normal forms

$$
\begin{aligned}
&\textsf{data isNf} &&: (\Gamma : \textsf{Con})(A : \textsf{Ty}\ \Gamma) \to \textsf{Tm}\ \Gamma\ A \to \textsf{Set where} \\
&\quad \textsf{varNf} &&: (x : \textsf{Var}\ \Gamma\ A) \to \textsf{isNf}\ \Gamma\ A\ (\textsf{var}\ x) \\
&\quad \textsf{lamNf} &&: \textsf{isNf}\ (\Gamma \rhd A)\ B\ b \to \textsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ (\textsf{lam}\ b) \\
&\quad \textsf{appNf} &&: \textsf{isNf}\ \Gamma\ (\Pi\ A\ B)\ f \to \textsf{isNf}\ \Gamma\ A\ a \to \textsf{isNf}\ \Gamma\ (B[\langle a \rangle])\ (app\ f\ a) \\
&\quad \textsf{eq} &&: (u^N\ v^N : \textsf{isNf}\ \Gamma\ A\ u) \to u^N \equiv v^N
\end{aligned}
$$

$$
\textsf{norm} : (u : \textsf{Tm}\ \Gamma\ A) \to \textsf{isNf}\ \Gamma\ A\ u
$$

$\_[\_]^N$ on terms using α-normalisation:

$$
\_[\_]^N : \textsf{isNf}\ \Gamma\ A\ a \to \textsf{isNfs}\ \Delta\ \Gamma\ \gamma \to (a' : \textsf{Tm}\ \Delta\ (A[\gamma]^T)) \times (a' \equiv a[\gamma]^t)
$$

$$
\begin{aligned}
\_[\_] &: \textsf{Tm}\ \Gamma\ A \to (\gamma : \textsf{Sub}\ \Delta\ \Gamma) \to \textsf{Tm}\ \Delta\ (A[\gamma]^T) \\
t\ [\ \gamma\ ] &:= \textsf{fst}\ (\textsf{norm}\ t\ [\textsf{norm}^S\ \gamma]^N)
\end{aligned}
$$

This talk: strictification of syntax using α-normal forms

$$
\begin{aligned}
&\text{data isNf} : (\Gamma : \text{Con})(A : \text{Ty}\ \Gamma) \to \text{Tm}\ \Gamma\ A \to \text{Set where} \\
&\quad \text{varNf} : (x : \text{Var}\ \Gamma\ A) \to \text{isNf}\ \Gamma\ A\ (\text{var}\ x) \\
&\quad \text{lamNf} : \text{isNf}\ (\Gamma \rhd A)\ B\ b \to \text{isNf}\ \Gamma\ (\Pi\ A\ B)\ (\text{lam}\ b) \\
&\quad \text{appNf} : \text{isNf}\ \Gamma\ (\Pi\ A\ B)\ f \to \text{isNf}\ \Gamma\ A\ a \to \text{isNf}\ \Gamma\ (B[\langle a \rangle])\ (app\ f\ a) \\
&\quad \text{eq} : (u^N\ v^N : \text{isNf}\ \Gamma\ A\ u) \to u^N \equiv v^N
\end{aligned}
$$

$$
\text{norm} : (u : \text{Tm}\ \Gamma\ A) \to \text{isNf}\ \Gamma\ A\ u
$$

$\_[\_]^N$ on terms using α-normalisation:
$\_[\_]^N : \text{isNf}\ \Gamma\ A\ a \to \text{isNfs}\ \Delta\ \Gamma\ \gamma \to (a' : \text{Tm}\ \Delta\ (A[\gamma]^T)) \times (a' \equiv a[\gamma]^t)$

$$
\begin{aligned}
&\_[\_] : \text{Tm}\ \Gamma\ A \to (\gamma : \text{Sub}\ \Delta\ \Gamma) \to \text{Tm}\ \Delta\ (A[\gamma]^T) \\
&t\ [\ \gamma\ ] := \text{fst}\ (\text{norm}\ t\ [\text{norm}^S\ \gamma]^N)
\end{aligned}
$$

Summary

Summary

- Implemented in Cubical Agda for STT

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax
  - two different substitution operations

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax
  - two different substitution operations
- All the substitution laws are definitional

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax
  - two different substitution operations
- All the substitution laws are definitional
  - we don't know how to achieve this without separating

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax
  - two different substitution operations
- All the substitution laws are definitional
  - we don't know how to achieve this without separating
- Defined strict syntax, proved its induction principle (dependent models have a section)

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax
  - two different substitution operations
- All the substitution laws are definitional
  - we don't know how to achieve this without separating
- Defined strict syntax, proved its induction principle (dependent models have a section)
- Derived parallel substitutions (their laws are not strict anymore)

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax
  - two different substitution operations
- All the substitution laws are definitional
  - we don't know how to achieve this without separating
- Defined strict syntax, proved its induction principle (dependent models have a section)
- Derived parallel substitutions (their laws are not strict anymore)

Future work:

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax
  - two different substitution operations
- All the substitution laws are definitional
  - we don't know how to achieve this without separating
- Defined strict syntax, proved its induction principle (dependent models have a section)
- Derived parallel substitutions (their laws are not strict anymore)

Future work:

- We need parallel substitutions for applications, and it is hard to make them all strict.

Summary

- Implemented in Cubical Agda for STT
  - using single substitution calculus
  - weakenings and single substitutions are separated in the syntax
  - two different substitution operations
- All the substitution laws are definitional
  - we don't know how to achieve this without separating
- Defined strict syntax, proved its induction principle (dependent models have a section)
- Derived parallel substitutions (their laws are not strict anymore)

Future work:

- We need parallel substitutions for applications, and it is hard to make them all strict.
- Extension to dependent types.