Generic Bidirectional Typing for Dependent Type Theories

Thiago Felicissimo

Journées d'hiver du GT SCALP

November 27, 2023

Type annotations in dependent type theory

Dependent type theory suffers from verbosity of type annotations

$$\text{Application:} \quad t @_{A,x.B} u$$

$$\text{Dependent pair:} \quad \langle t, u \rangle_{A,x.B}$$

$$\text{Cons:} \quad t ::_A l$$

Not only one application, but one for each pair A, B.

Type annotations in dependent type theory

Dependent type theory suffers from verbosity of type annotations

$$\text{Application:} \quad t @_{A,x.B} u$$

$$\text{Dependent pair:} \quad \langle t, u \rangle_{A,x.B}$$

$$\text{Cons:} \quad t ::_A l$$

Not only one application, but one for each pair A, B. Unusable in practice...

Type annotations in dependent type theory

Dependent type theory suffers from verbosity of type annotations

Application: $t@_{A,x.B}u$

Dependent pair: $\langle t, u\rangle_{A,x.B}$

Cons: $t ::_A l$

Not only one application, but one for each pair A, B. Unusable in practice...

Most presentation restore usability by eliding type annotations from syntax

Application: $t\ u$

Dependent pair: $\langle t, u\rangle$

Cons: $t :: l$

Type annotations in dependent type theory

Dependent type theory suffers from verbosity of type annotations

Application: $t @_{A,x.B} u$

Dependent pair: $\langle t, u \rangle_{A,x.B}$

Cons: $t ::_A l$

Not only one application, but one for each pair A, B. Unusable in practice...

Most presentation restore usability by eliding type annotations from syntax

Application: $t\ u$

Dependent pair: $\langle t, u \rangle$

Cons: $t :: l$

Syntax so common that many don't realize that an omission is being made

Typechecking without annotations

Omission has a cost Knowing annotations is needed for typing

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma, x : A \vdash B \text{ type} \qquad \Gamma \vdash t : \Pi x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u/x]}$$

Typechecking without annotations

Omission has a cost Knowing annotations is needed for typing

$$\frac{\Gamma \vdash ?\ \mathsf{type} \qquad \Gamma, x : ? \vdash ?\ \mathsf{type} \qquad \Gamma \vdash t : ? \qquad \Gamma \vdash u : ?}{\Gamma \vdash t\ u : ?}$$

Typechecking without annotations

Omission has a cost Knowing annotations is needed for typing

$$\frac{\Gamma \vdash ?\ \mathsf{type} \qquad \Gamma, x : ? \vdash ?\ \mathsf{type} \qquad \Gamma \vdash t : ? \qquad \Gamma \vdash u : ?}{\Gamma \vdash t\ u : ?}$$

How to find A and B if they're not stored in syntax?

Typechecking without annotations

Omission has a cost Knowing annotations is needed for typing

$$\frac{\Gamma \vdash ?\ \mathsf{type} \qquad \Gamma, x : ? \vdash ?\ \mathsf{type} \qquad \Gamma \vdash t : ? \qquad \Gamma \vdash u : ?}{\Gamma \vdash t\ u : ?}$$

How to find A and B if they're not stored in syntax?

Bidirectional typing Decompose $t : A$ in modes check $t \Leftarrow A$ and infer $t \Rightarrow A$

Typechecking without annotations

Omission has a cost Knowing annotations is needed for typing

$$\frac{\Gamma \vdash ?\ \text{type} \qquad \Gamma, x : ? \vdash ?\ \text{type} \qquad \Gamma \vdash t : ? \qquad \Gamma \vdash u : ?}{\Gamma \vdash t\ u : ?}$$

How to find A and B if they're not stored in syntax?

Bidirectional typing Decompose $t : A$ in modes check $t \Leftarrow A$ and infer $t \Rightarrow A$

Allow specify flow of type information in typing rules, explain how to use them

$$\frac{\Gamma \vdash t \Rightarrow C \qquad C \longrightarrow^{*} \Pi x : A.B \qquad \Gamma \vdash u \Leftarrow A}{\Gamma \vdash t\ u \Rightarrow B[u/x]}$$

Typechecking without annotations

Omission has a cost Knowing annotations is needed for typing

$$\frac{\Gamma \vdash ?\ \text{type} \qquad \Gamma, x : ? \vdash ?\ \text{type} \qquad \Gamma \vdash t : ? \qquad \Gamma \vdash u : ?}{\Gamma \vdash t\ u : ?}$$

How to find A and B if they're not stored in syntax?

Bidirectional typing Decompose $t : A$ in modes check $t \Leftarrow A$ and infer $t \Rightarrow A$

Allow specify flow of type information in typing rules, explain how to use them

$$\frac{\Gamma \vdash t \Rightarrow C \qquad C \longrightarrow^{*} \Pi x : A.B \qquad \Gamma \vdash u \Leftarrow A}{\Gamma \vdash t\ u \Rightarrow B[u/x]}$$

Complements unannotated syntax very well, explains how to recover annotations

Contribution

Bidirectional type systems have been studied and proposed for many theories

However, general guidelines have remained informal, no unified framework

Contribution

Bidirectional type systems have been studied and proposed for many theories

However, general guidelines have remained informal, no unified framework

This talk Generic account of bidirectional typing for class of type theories

Contribution

Bidirectional type systems have been studied and proposed for many theories

However, general guidelines have remained informal, no unified framework

This talk Generic account of bidirectional typing for class of type theories

Roadmap

Contribution

Bidirectional type systems have been studied and proposed for many theories

However, general guidelines have remained informal, no unified framework

This talk Generic account of bidirectional typing for class of type theories

Roadmap

1. We give a general definition of type theories (or equivalently, a logical framework) supporting non-annotated syntaxes

Contribution

Bidirectional type systems have been studied and proposed for many theories

However, general guidelines have remained informal, no unified framework

This talk Generic account of bidirectional typing for class of type theories

Roadmap

1. We give a general definition of type theories (or equivalently, a logical framework) supporting non-annotated syntaxes
2. For each theory, we define declarative and bidirectional type systems

Contribution

Bidirectional type systems have been studied and proposed for many theories

However, general guidelines have remained informal, no unified framework

This talk Generic account of bidirectional typing for class of type theories

Roadmap

1. We give a general definition of type theories (or equivalently, a logical framework) supporting non-annotated syntaxes
2. For each theory, we define declarative and bidirectional type systems
3. We show, in a theory-independent fashion, their equivalence

The theories

One syntax for all!

$$
\begin{aligned}
t, u, T, U \ ::= \ & \mid x && \text{(variables)} \\
& \mid c(\vec{x}_1.u_1, \ldots, \vec{x}_k.u_k) && \text{(constructor application)} \\
& \mid d(t;\ \vec{x}_1.u_1, \ldots, \vec{x}_k.u_k) && \text{(destructor application)} \\
& \mid \mathsf{x}\{u_1, \ldots, u_k\} && \text{(metavariables)}
\end{aligned}
$$

In $d(t; \ldots)$, we call t the *principal argument*.

One syntax for all!

$$
\begin{aligned}
t, u, T, U \ ::= &\mid x && \text{(variables)} \\
&\mid c(\vec{x}_1.u_1, \ldots, \vec{x}_k.u_k) && \text{(constructor application)} \\
&\mid d(t;\ \vec{x}_1.u_1, \ldots, \vec{x}_k.u_k) && \text{(destructor application)} \\
&\mid \mathsf{x}\{u_1, \ldots, u_k\} && \text{(metavariables)}
\end{aligned}
$$

In $d(t; \ldots)$, we call t the *principal argument*.

Example

$$
\begin{aligned}
\Sigma_{\lambda\Pi} = \quad & \Pi(\mathsf{A}, \mathsf{B}\{x\}),\ \lambda(\mathsf{t}\{x\}),\ \mathrm{Ty},\ \mathrm{Tm}(\mathsf{A}), && \text{(constructors)} \\
& @(\mathsf{u}) && \text{(destructors)}
\end{aligned}
$$

$$
t, u, A, B ::= x \mid \mathsf{x}\{\vec{t}\} \mid \mathrm{Ty} \mid \mathrm{Tm}(A) \mid @(t; u) \mid \lambda(x.t) \mid \Pi(A, x.B)
$$

The theories

A *theory* $\mathbb{T}$ is made of *schematic typing rules* and *rewrite rules*.

3 schematic typing rules: *sort rules*, *constructor rules* and *destructor rules*

1

The theories

A *theory* $\mathbb{T}$ is made of *schematic typing rules* and *rewrite rules*.

3 schematic typing rules: *sort rules*, *constructor rules* and *destructor rules*

Sort rules Sorts are terms that can type other terms[1].

Used to define the *judgment forms* of the theory.

[1] We use the name "sort" instead of "type" to avoid a name clash with the types of the theory

The theories

A *theory* $\mathbb{T}$ is made of *schematic typing rules* and *rewrite rules*.

3 schematic typing rules: *sort rules*, *constructor rules* and *destructor rules*

Sort rules Sorts are terms that can type other terms[1].

Used to define the *judgment forms* of the theory.

Example: In MLTT, 2 judgment forms: $\square$ type and $\square : A$ for a type A.

$$\frac{}{\vdash \mathsf{Ty}\ \mathsf{sort}} \qquad \frac{\vdash \mathsf{A} : \mathsf{Ty}}{\vdash \mathsf{Tm}(\mathsf{A})\ \mathsf{sort}}$$

We can then write $A : \mathsf{Ty}$ for A type, and $t : \mathsf{Tm}(A)$ for $t : A$

[1] We use the name "sort" instead of "type" to avoid a name clash with the types of the theory

The theories

Constructor rules are bidirectionally typed in mode check

The sort of the rule is a pattern allowing to recover the omitted arguments

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathtt{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathtt{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))}$$

The theories

Constructor rules are bidirectionally typed in mode check

The sort of the rule is a pattern allowing to recover the omitted arguments

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathtt{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathtt{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))}$$

Destructor rules are bidirectionally typed in mode infer

The sort of the *principal argument* $\mathtt{t} : T^{\mathsf{P}}$ should be a pattern allowing to recover the omitted arguments

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad \vdash \mathtt{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \qquad \vdash \mathtt{u} : \mathrm{Tm}(\mathsf{A})}{\vdash @(\mathtt{t}; \mathtt{u}) : \mathrm{Tm}(\mathsf{B}\{\mathtt{u}\})}$$

The theories

Rewrite rules Define the definitional equality (aka conversion) $\equiv$ of the theory.

$$@(\lambda(x.\mathtt{t}\{x\});\mathtt{u}) \longmapsto \mathtt{t}\{\mathtt{u}\}$$

In general, of the form $d(t^{\mathsf{P}}; \vec{x}_1.t_1^{\mathsf{P}}, \ldots, \vec{x}_k.t_k^{\mathsf{P}}) \longmapsto r$, with left-hand-side linear.

The theories

Rewrite rules Define the definitional equality (aka conversion) $\equiv$ of the theory.

$$@(\lambda(x.\mathtt{t}\{x\});\mathtt{u}) \longmapsto \mathtt{t}\{\mathtt{u}\}$$

In general, of the form $d(t^{\mathsf{P}}; \vec{x}_1.t_1^{\mathsf{P}}, \ldots, \vec{x}_k.t_k^{\mathsf{P}}) \longmapsto r$, with left-hand-side linear.

Condition: no two left-hand sides unify.

Therefore, rewrite systems are orthogonal, hence confluent by construction!

Full example

Theory $\mathbb{T}_{\lambda\Pi}$, defining minimalistic Martin-Lof Type Theory.

$$\mathrm{Ty}(\cdot)\ \mathsf{sort}$$

$$\mathrm{Tm}(\mathsf{A} : \mathrm{Ty})\ \mathsf{sort}$$

$$\Pi(\cdot;\ \ \mathsf{A} : \mathrm{Ty},\ \mathsf{B}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Ty}) : \mathrm{Ty}$$

$$\lambda(\mathsf{A} : \mathrm{Ty},\ \mathsf{B}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Ty};\ \ \mathsf{t}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Tm}(\mathsf{B}\{x\})) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))$$

$$@(\mathsf{A} : \mathrm{Ty},\ \mathsf{B}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Ty};\ \ \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}));\ \ \mathsf{u} : \mathrm{Tm}(\mathsf{A})) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\})$$

$$@(\lambda(x.\mathsf{t}\{x\}); \mathsf{u}) \longmapsto \mathsf{t}\{\mathsf{u}\}$$

Declarative typing

Declarative typing rules

Each theory $\mathbb{T}$ defines a declarative type system.

Declarative typing rules

Each theory $\mathbb{T}$ defines a declarative type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$\rightsquigarrow$

$\rightsquigarrow$

Declarative typing rules

Each theory $\mathbb{T}$ defines a declarative type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathsf{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))} \quad \rightsquigarrow$$

$$\rightsquigarrow$$

Declarative typing rules

Each theory $\mathbb{T}$ defines a declarative type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathsf{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))} \quad \rightsquigarrow \quad \frac{\Gamma \vdash A : \mathrm{Ty} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash B : \mathrm{Ty} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash t : \mathrm{Tm}(B)}{\Gamma \vdash \lambda(x.t) : \mathrm{Tm}(\Pi(A, x.B))}$$

$$\rightsquigarrow$$

Declarative typing rules

Each theory $\mathbb{T}$ defines a declarative type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathsf{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))} \quad \rightsquigarrow \quad \frac{\Gamma \vdash A : \mathrm{Ty} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash B : \mathrm{Ty} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash t : \mathrm{Tm}(B)}{\Gamma \vdash \lambda(x.t) : \mathrm{Tm}(\Pi(A, x.B))}$$

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad \vdash \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \qquad \vdash \mathsf{u} : \mathrm{Tm}(\mathsf{A})}{\vdash @(\mathsf{t}; \mathsf{u}) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\})} \quad \rightsquigarrow$$

Declarative typing rules

Each theory $\mathbb{T}$ defines a declarative type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathsf{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))} \quad \rightsquigarrow \quad \frac{\Gamma \vdash A : \mathrm{Ty} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash B : \mathrm{Ty} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash t : \mathrm{Tm}(B)}{\Gamma \vdash \lambda(x.t) : \mathrm{Tm}(\Pi(A, x.B))}$$

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad \vdash \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \qquad \vdash \mathsf{u} : \mathrm{Tm}(\mathsf{A})}{\vdash @(\mathsf{t}; \mathsf{u}) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\})} \quad \rightsquigarrow \quad \frac{\Gamma \vdash A : \mathrm{Ty} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash B : \mathrm{Ty} \qquad \Gamma \vdash t : \mathrm{Tm}(\Pi(A, x.B)) \qquad \Gamma \vdash u : \mathrm{Tm}(A)}{\Gamma \vdash @(t; u) : \mathrm{Tm}(B[u/x])}$$

Bidirectional typing

Matching modulo rewriting

In bidirectional typing, we need matching modulo rewriting to recover missing arguments.

$$\frac{\Gamma \vdash t \Rightarrow U \qquad \ldots}{\Gamma \vdash @(t; u) \Rightarrow}$$

Matching modulo rewriting

In bidirectional typing, we need matching modulo rewriting to recover missing arguments.

$$\frac{\Gamma \vdash t \Rightarrow U \qquad \ldots}{\Gamma \vdash @(t; u) \Rightarrow}$$

We know

$$U \equiv \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))[A/\mathsf{A},\ x.B/\mathsf{B}]$$

but how to recover A and B from U?

Matching modulo rewriting

In bidirectional typing, we need matching modulo rewriting to recover missing arguments.

$$\frac{\Gamma \vdash t \Rightarrow U \qquad \dots}{\Gamma \vdash @(t; u) \Rightarrow}$$

We know

$$U \equiv \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))[A/\mathsf{A},\ x.B/\mathsf{B}]$$

but how to recover A and B from U?

Given t^{P} and u, we define a matching judgment

$$t^{\mathsf{P}} \prec u \rightsquigarrow \vec{x}_1.t_1/\mathsf{x}_1, \dots, \vec{x}_k.t_k/\mathsf{x}_k$$

that tries to compute a metavariable substitution s.t. $t^{\mathsf{P}}[\vec{x}_1.t_1/\mathsf{x}_1, \dots, \vec{x}_k.t_k/\mathsf{x}_k] \equiv u$.

Inferable and checkable terms

Not all unannotated terms can be algorithmically typed

$$\dfrac{\dfrac{?}{\Gamma \vdash \lambda(x.t) \Rightarrow ?} \qquad \cdots}{\Gamma \vdash @(\lambda(x.t); u) \Rightarrow ?}$$

Inferable and checkable terms

Not all unannotated terms can be algorithmically typed

$$\dfrac{\dfrac{?}{\Gamma \vdash \lambda(x.t) \Rightarrow ?} \qquad \cdots}{\Gamma \vdash @(\lambda(x.t); u) \Rightarrow ?}$$

Avoided by defining bidirectional typing only for *inferrable* and *checkable* terms.

$$t^{\mathrm{i}}, u^{\mathrm{i}} ::= x \mid d(t^{\mathrm{i}};\ \vec{x}_1.u_1^{\mathrm{c}}, \ldots, \vec{x}_k.u_k^{\mathrm{c}})$$
$$t^{\mathrm{c}}, u^{\mathrm{c}} ::= c(\vec{x}_1.u_1^{\mathrm{c}}, \ldots, \vec{x}_k.u_k^{\mathrm{c}}) \mid \underline{t}^{\mathrm{i}}$$

Inferable and checkable terms

Not all unannotated terms can be algorithmically typed

$$\dfrac{\dfrac{?}{\Gamma \vdash \lambda(x.t) \Rightarrow ?} \qquad \dots}{\Gamma \vdash @(\lambda(x.t); u) \Rightarrow ?}$$

Avoided by defining bidirectional typing only for *inferrable* and *checkable* terms.

$$\begin{aligned} t^{\mathsf{i}}, u^{\mathsf{i}} &::= x \mid d(t^{\mathsf{i}};\ \vec{x}_1.u_1^{\mathsf{c}}, \dots, \vec{x}_k.u_k^{\mathsf{c}}) \\ t^{\mathsf{c}}, u^{\mathsf{c}} &::= c(\vec{x}_1.u_1^{\mathsf{c}}, \dots, \vec{x}_k.u_k^{\mathsf{c}}) \mid \underline{t}^{\mathsf{i}} \end{aligned}$$

Principal argument of a destructor can only be variable or another destructor.

For most theories: $t^{\mathsf{c}}, u^{\mathsf{c}}, \dots$ = normal forms, and $t^{\mathsf{i}}, u^{\mathsf{i}}, \dots$ = neutrals

Bidirectional typing rules

Each theory $\mathbb{T}$ defines a bidirectional type system.

Bidirectional typing rules

Each theory $\mathbb{T}$ defines a bidirectional type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$\rightsquigarrow$

$\rightsquigarrow$

Bidirectional typing rules

Each theory $\mathbb{T}$ defines a bidirectional type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathsf{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))} \quad \rightsquigarrow$$

$$\rightsquigarrow$$

Bidirectional typing rules

Each theory $\mathbb{T}$ defines a bidirectional type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathtt{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathtt{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))} \quad \rightsquigarrow \quad \frac{\mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \prec T \rightsquigarrow A/\mathsf{A},\; x.B/\mathsf{B} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash t^{\mathrm{c}} \Leftarrow \mathrm{Tm}(B)}{\Gamma \vdash \lambda(x.t^{\mathrm{c}}) \Leftarrow T}$$

$$\rightsquigarrow$$

Bidirectional typing rules

Each theory $\mathbb{T}$ defines a bidirectional type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathsf{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))} \quad \rightsquigarrow \quad \frac{\mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \prec T \rightsquigarrow A/\mathsf{A},\ x.B/\mathsf{B} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash t^{\mathsf{c}} \Leftarrow \mathrm{Tm}(B)}{\Gamma \vdash \lambda(x.t^{\mathsf{c}}) \Leftarrow T}$$

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad \vdash \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \qquad \vdash \mathsf{u} : \mathrm{Tm}(\mathsf{A})}{\vdash @(\mathsf{t}; \mathsf{u}) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\})} \quad \rightsquigarrow$$

Bidirectional typing rules

Each theory $\mathbb{T}$ defines a bidirectional type system.

Main typing rules instantiate the schematic rules of $\mathbb{T}$:

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{t} : \mathrm{Tm}(\mathsf{B}\{x\})}{\vdash \lambda(\mathsf{t}) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))} \quad \rightsquigarrow \quad \frac{\mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \prec T \rightsquigarrow A/\mathsf{A},\ x.B/\mathsf{B} \qquad \Gamma, x : \mathrm{Tm}(A) \vdash t^{\mathsf{c}} \Leftarrow \mathrm{Tm}(B)}{\Gamma \vdash \lambda(x.t^{\mathsf{c}}) \Leftarrow T}$$

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad \vdash \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \qquad \vdash \mathsf{u} : \mathrm{Tm}(\mathsf{A})}{\vdash @(\mathsf{t}; \mathsf{u}) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\})} \quad \rightsquigarrow \quad \frac{\Gamma \vdash t^{\mathsf{i}} \Rightarrow T \qquad \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \prec T \rightsquigarrow A/\mathsf{A},\ x.B/\mathsf{B} \qquad \Gamma \vdash u^{\mathsf{c}} \Leftarrow \mathrm{Tm}(A)}{\Gamma \vdash @(t^{\mathsf{i}}; u^{\mathsf{c}}) \Rightarrow \mathrm{Tm}(B[u/x])}$$

Equivalence with declarative typing

Suppose underlying theory $\mathbb{T}$ is valid.

Equivalence with declarative typing

Suppose underlying theory $\mathbb{T}$ is valid.

Soundness If $\Gamma \vdash$ and $\Gamma \vdash t^{\mathrm{i}} \Rightarrow T$ then $\Gamma \vdash t : T$.
If $\Gamma \vdash T$ sort and $\Gamma \vdash t^{\mathrm{c}} \Leftarrow T$ then $\Gamma \vdash t : T$.

Equivalence with declarative typing

Suppose underlying theory $\mathbb{T}$ is valid.

Soundness If $\Gamma \vdash$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$ then $\Gamma \vdash t : T$.
If $\Gamma \vdash T$ sort and $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ then $\Gamma \vdash t : T$.

Completeness For t^{i} inferable, if $\Gamma \vdash t : T$ then $\Gamma \vdash t^{\mathsf{i}} \Rightarrow U$ with $T \equiv U$.
For t^{c} checkable, if $\Gamma \vdash t : T$ then $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$.

Equivalence with declarative typing

Suppose underlying theory $\mathbb{T}$ is valid.

Soundness If $\Gamma \vdash$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$ then $\Gamma \vdash t : T$.
If $\Gamma \vdash T$ sort and $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ then $\Gamma \vdash t : T$.

Completeness For t^{i} inferable, if $\Gamma \vdash t : T$ then $\Gamma \vdash t^{\mathsf{i}} \Rightarrow U$ with $T \equiv U$.
For t^{c} checkable, if $\Gamma \vdash t : T$ then $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$.

Decidability If $\mathbb{T}$ weak normalizing, then inference is decidable for inferable terms, and checking is decidable for checkable terms.

More examples

Dependent sums

Extends $\mathbb{T}_{\lambda\Pi}$ with

$$\frac{\vdash \mathsf{A} : \mathbf{Ty} \qquad x : \mathbf{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathbf{Ty}}{\vdash \Sigma(\mathsf{A}, \mathsf{B}) \ : \mathbf{Ty}}$$

$$\frac{\vdash \mathsf{A} : \mathbf{Ty} \qquad x : \mathbf{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathbf{Ty} \qquad \vdash \mathsf{t} : \mathbf{Tm}(\mathsf{A}) \qquad \vdash \mathsf{u} : \mathbf{Tm}(\mathsf{B}\{\mathsf{t}\})}{\vdash \mathrm{pair}(\mathsf{t}, \mathsf{u}) \ : \mathbf{Tm}(\Sigma(\mathsf{A}, x.\mathsf{B}\{x\}))}$$

$$\frac{\vdash \mathsf{A} : \mathbf{Ty} \qquad x : \mathbf{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathbf{Ty} \qquad \vdash \mathsf{t} : \mathbf{Tm}(\Sigma(\mathsf{A}, x.\mathsf{B}\{x\}))}{\vdash \mathrm{proj}_1(\mathsf{t}; \cdot) \ : \mathbf{Tm}(\mathsf{A})}$$

$$\frac{\vdash \mathsf{A} : \mathbf{Ty} \qquad x : \mathbf{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathbf{Ty} \qquad \vdash \mathsf{t} : \mathbf{Tm}(\Sigma(\mathsf{A}, x.\mathsf{B}\{x\}))}{\vdash \mathrm{proj}_2(\mathsf{t}; \cdot) \ : \mathbf{Tm}(\mathsf{B}\{\mathrm{proj}_1(\mathsf{t})\})}$$

$$\mathrm{proj}_1(\mathrm{pair}(\mathsf{t}, \mathsf{u}); \varepsilon) \longmapsto \mathsf{t} \qquad \mathrm{proj}_2(\mathrm{pair}(\mathsf{t}, \mathsf{u}); \varepsilon) \longmapsto \mathsf{u}$$

Lists

Extends $\mathbb{T}_{\lambda\Pi}$ with

$$\frac{\vdash \mathsf{A} : \mathbf{Ty}}{\vdash \mathbf{List}(\mathsf{A}) : \mathbf{Ty}} \qquad \frac{\vdash \mathsf{A} : \mathbf{Ty}}{\vdash \mathbf{nil} : \mathbf{Tm}(\mathbf{List}(\mathsf{A}))} \qquad \frac{\vdash \mathsf{A} : \mathbf{Ty} \qquad \vdash \mathsf{x} : \mathbf{Tm}(\mathsf{A}) \qquad \vdash \mathsf{l} : \mathbf{Tm}(\mathbf{List}(\mathsf{A}))}{\vdash \mathbf{cons}(\mathsf{x}, \mathsf{l}) : \mathbf{Tm}(\mathbf{List}(\mathsf{A}))}$$

$$\frac{\vdash \mathsf{A} : \mathbf{Ty} \qquad \vdash \mathsf{l} : \mathbf{Tm}(\mathbf{List}(\mathsf{A})) \qquad x : \mathbf{Tm}(\mathbf{List}(\mathsf{A})) \vdash \mathsf{P} : \mathbf{Ty} \qquad \vdash \mathsf{pnil} : \mathbf{Tm}(\mathsf{P}\{\mathbf{nil}\}) \qquad x : \mathbf{Tm}(\mathsf{A}), y : \mathbf{Tm}(\mathbf{List}(\mathsf{A})), z : \mathbf{Tm}(\mathsf{P}\{y\}) \vdash \mathsf{pcons} : \mathbf{Tm}(\mathsf{P}\{\mathbf{cons}(x, y)\})}{\vdash \mathbf{ListRec}(\mathsf{l}; \mathsf{P}, \mathsf{pnil}, \mathsf{pcons}) : \mathbf{Tm}(\mathsf{P}\{\mathsf{l}\})}$$

$$\mathbf{ListRec}(\mathbf{nil}; x.\mathsf{P}\{x\}, \mathsf{pnil}, xyz.\mathsf{pcons}\{x, y, z\}) \longmapsto \mathsf{pnil}$$

$$\mathbf{ListRec}(\mathbf{cons}(\mathsf{x}, \mathsf{l}); x.\mathsf{P}\{x\}, \mathsf{pnil}, xyz.\mathsf{pcons}\{x, y, z\}) \longmapsto \mathsf{pcons}\{\mathsf{x}, \mathsf{l}, \mathbf{ListRec}(\mathsf{l}; x.\mathsf{P}\{x\}, \mathsf{pnil}, xyz.\mathsf{pcons}\{x, y, z\})\}$$

W types

Extends $\mathbb{T}_{\lambda\Pi}$ with

$$\frac{\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty}}{\vdash \mathrm{W}(\mathsf{A}, \mathsf{B}) : \mathrm{Ty}}$$

$$\frac{\begin{array}{c}\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \\ \vdash \mathsf{a} : \mathrm{Tm}(\mathsf{A}) \qquad \vdash \mathsf{f} : \mathrm{Tm}(\Pi(\mathsf{B}\{\mathsf{a}\}, x'.\mathrm{W}(\mathsf{A}, x.\mathsf{B}\{x\})))\end{array}}{\vdash \mathrm{sup}(\mathsf{a}, \mathsf{f}) : \mathrm{Tm}(\mathrm{W}(\mathsf{A}, x.\mathsf{B}\{x\}))}$$

$$\frac{\begin{array}{c}\vdash \mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad \vdash \mathsf{t} : \mathrm{Tm}(\mathrm{W}(\mathsf{A}, x.\mathsf{B}\{x\})) \qquad x : \mathrm{Tm}(\mathrm{W}(\mathsf{A}, x.\mathsf{B}\{x\})) \vdash \mathsf{P} : \mathrm{Ty} \\ x : \mathrm{Tm}(\mathsf{A}), y : \mathrm{Tm}(\Pi(\mathsf{B}\{x\}, x'.\mathrm{W}(\mathsf{A}, x.\mathsf{B}\{x\}))), z : \mathrm{Tm}(\Pi(\mathsf{B}\{x\}, x'.\mathsf{P}\{@(y, x')\})) \vdash \mathsf{p} : \mathrm{Tm}(\mathsf{P}\{\mathrm{sup}(x, y)\})\end{array}}{\vdash \mathrm{WRec}(\mathsf{t}; \mathsf{P}, \mathsf{p}) : \mathrm{Tm}(\mathsf{P}\{\mathsf{t}\})}$$

$$\mathrm{WRec}(\mathrm{sup}(\mathsf{a}, \mathsf{f}); x.\mathsf{P}\{x\}, xyz.\mathsf{p}\{x, y, z\}) \longmapsto \mathsf{p}\{\mathsf{a}, \mathsf{f}, \lambda(x.\mathrm{WRec}(@(\mathsf{f}, x); x.\mathsf{P}\{x\}, xyz.\mathsf{p}\{x, y, z\}))\}$$

Universes

Extends $\mathbb{T}_{\lambda\Pi}$ with

$$\frac{}{\vdash \mathrm{U}(\cdot) : \mathrm{Ty}} \qquad \frac{\vdash \mathsf{a} : \mathrm{Tm}(\mathrm{U})}{\vdash \mathrm{El}(\mathsf{a}; \cdot) : \mathrm{Ty}}$$

Tarski-style Adds codes for all types

$$\frac{}{\vdash \mathrm{u}(\cdot) : \mathrm{Tm}(\mathrm{U})} \qquad \mathrm{El}(\mathrm{u}; \varepsilon) \longmapsto \mathrm{U}$$

$$\frac{\vdash \mathsf{a} : \mathrm{Tm}(\mathrm{U}) \qquad x : \mathrm{Tm}(\mathrm{El}(\mathsf{a})) \vdash \mathsf{b} : \mathrm{Tm}(\mathrm{U})}{\vdash \pi(\mathsf{a}, \mathsf{b}) : \mathrm{Tm}(\mathrm{U})}$$

$$\mathrm{El}(\pi(\mathsf{a}, x.\mathsf{b}\{x\}); \varepsilon) \longmapsto \Pi(\mathrm{El}(\mathsf{a}; \varepsilon), x.\mathrm{El}(\mathsf{b}\{x\}; \varepsilon))$$

(Weak) Coquand-style

Adds a code constructor c

$$\frac{\vdash \mathsf{A} : \mathrm{Ty}}{\vdash \mathrm{c}(\mathsf{A}) : \mathrm{Tm}(\mathrm{U})}$$

$$\mathrm{El}(\mathrm{c}(\mathsf{A}); \varepsilon) \longmapsto \mathsf{A}$$

Conclusion

Conclusion

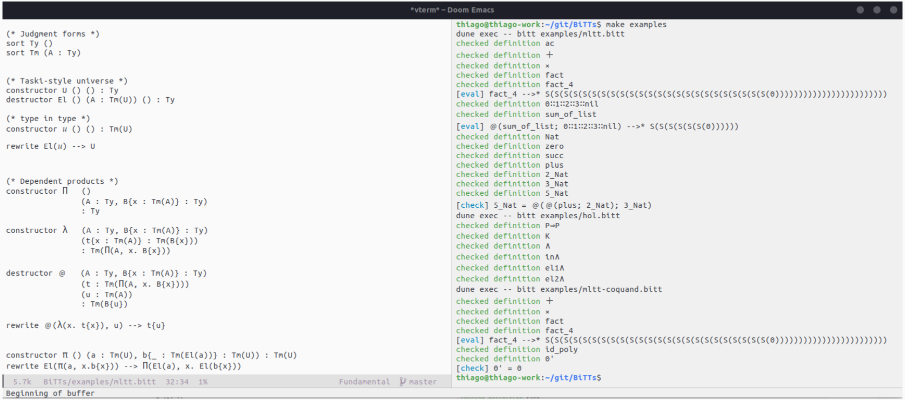
Generic account of bidirectional typing for class of dependent type theories

Conclusion

Generic account of bidirectional typing for class of dependent type theories

Bidirectional system implemented in a prototype, available at

https://github.com/thiagofelicissimo/BiTTs



Thank you for your attention!