Generic Bidirectional Typing for Dependent Type Theories

Thiago Felicissimo

Proofs and algorithms seminar

16 December 2024

The syntax of type theory

When defining syntax of programming languages and type theories, many choices:

The syntax of type theory

When defining syntax of programming languages and type theories, many choices:

Fully-annotated syntax keeps track of all annotations

$$t \mathbin{@}_{A,x.B} u \qquad \langle t, u \rangle_{A,x.B} \qquad t ::_A l \qquad \ldots$$

The syntax of type theory

When defining syntax of programming languages and type theories, many choices:

Fully-annotated syntax keeps track of all annotations

$$t \mathbin{@_{A,x.B}} u \qquad \langle t, u \rangle_{A,x.B} \qquad t ::_A l \qquad \dots$$

What one gets when looking at semantics of type theory

Arguably the most canonical choice

The syntax of type theory

When defining syntax of programming languages and type theories, many choices:

Fully-annotated syntax keeps track of all annotations

$$t \mathbin{@_{A,x.B}} u \qquad \langle t, u \rangle_{A,x.B} \qquad t ::_A l \qquad \ldots$$

What one gets when looking at semantics of type theory

Arguably the most canonical choice, but the syntax is unusable in practice...

The syntax of type theory

When defining syntax of programming languages and type theories, many choices:

Fully-annotated syntax keeps track of all annotations

$$t \mathbin{@_{A,x.B}} u \qquad \langle t, u\rangle_{A,x.B} \qquad t ::_A l \qquad \dots$$

What one gets when looking at semantics of type theory

Arguably the most canonical choice, but the syntax is unusable in practice...

Non-annotated syntax restores usability by eliding parameter annotations

$$t\ u \qquad \langle t, u\rangle \qquad t :: l \qquad \dots$$

The syntax of type theory

When defining syntax of programming languages and type theories, many choices:

Fully-annotated syntax keeps track of all annotations

$$t \mathbin{@_{A,x.B}} u \qquad \langle t, u \rangle_{A,x.B} \qquad t ::_A l \qquad \ldots$$

What one gets when looking at semantics of type theory

Arguably the most canonical choice, but the syntax is unusable in practice...

Non-annotated syntax restores usability by eliding parameter annotations

$$t\ u \qquad \langle t, u \rangle \qquad t :: l \qquad \ldots$$

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

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

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

$$\frac{\Gamma \vdash t \Rightarrow ?}{\Gamma \vdash t\ u \Rightarrow ?}$$

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 $t \Rightarrow A$

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

$$\frac{\Gamma \vdash t \Rightarrow C}{\Gamma \vdash t\ u \Rightarrow ?}$$

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

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

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

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

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, *locally* 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 work *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 work *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

$\boxed{\text{\textbf{This work} \textit{Generic} account of bidirectional typing for class of type theories}}$

Roadmap

1. We give a general definition of bidirectional type theory

Contribution

Bidirectional type systems have been studied and proposed for many theories

However, general guidelines have remained informal, no unified framework

This work *Generic* account of bidirectional typing for class of type theories

Roadmap

1. We give a general definition of bidirectional type theory
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 work *Generic* account of bidirectional typing for class of type theories

Roadmap

1. We give a general definition of bidirectional type theory
2. For each theory, we define declarative and bidirectional type systems
3. We show, in a theory-independent fashion, their equivalence

BiTTs: A theory-independent bidirectional type-checker

Our framework not only of theoretic interest, also has practical applications

BiTTs: A theory-independent bidirectional type-checker

Our framework not only of theoretic interest, also has practical applications

Implemented in the theory-independent bidirectional type-checker BiTTs

```
(* Equality *)
constructor Eq () (A : Ty, x : Tm(A), y : Tm(A)) : Ty

constructor refl (A : Ty, x : Tm(A), y : Tm(A)) () (x = y : Tm(A)) : Tm(Eq(A, x, y))

destructor J (A : Ty, x : Tm(A), y : Tm(A))
             [t : Tm(Eq(A, x, y))]
             (P{y : Tm(A), e : Tm(Eq(A, x, y))} : Ty, p : Tm(P{x, refl}))
             : Tm(P{y, t})

equation J(refl, y e. P{y, e}, p) --> p

(* some basic properties of equality *)
let sym : Tm(Π(U, a. Π(El(a), x. Π(El(a), y. Π(Eq(El(a), x, y), _. Eq(El(a), y, x))))))
    :=  λ(a. λ(x. λ(y. λ(p. J(p, z q. Eq(El(a), z, x), refl)))))

let trans : Tm(Π(U, a. Π(El(a), x. Π(El(a), y. Π(El(a), z. Π(Eq(El(a), x, y), _. Π(Eq(El(a), y, z), _. Eq(El(a), x, z))))))))
    := λ(a. λ(x. λ(y. λ(z. λ(p. λ(q. J(q, k r. Eq(El(a), x, k), p)))))))

let transp : Tm(Π(U, a. Π(U, b. Π(Eq(U, a, b), _. Π(El(a), _. El(b))))))
    := λ(a. λ(b. λ(p. λ(x. J(p, z q. El(z), x)))))
```

The theories

The theories

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

The theories

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

Three kinds of schematic typing rules: *sort rules*, *constructor rules*, *destructor rules*

The theories

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

Three kinds of schematic typing rules: *sort rules*, *constructor rules*, *destructor rules*

Sort rules[1] Sorts are terms T that can appear to the right in judgment $t : T$
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 *bidirectional theory* $\mathbb{T}^\flat$ is made of *schematic typing rules* and *rewrite rules*

Three kinds of schematic typing rules: *sort rules*, *constructor rules*, *destructor rules*

Sort rules[1] Sorts are terms T that can appear to the right in judgment $t : T$
Used to define the *judgment forms* of the theory

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

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

The theories

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

Three kinds of schematic typing rules: *sort rules*, *constructor rules*, *destructor rules*

Sort rules[1] Sorts are terms T that can appear to the right in judgment $t : T$
Used to define the *judgment forms* of the theory

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

$$\frac{}{\mathrm{Ty}\ \mathsf{sort}} \qquad A\ \text{type} \quad \rightsquigarrow \quad A : \mathrm{Ty}$$

$$\frac{\mathsf{A} : \mathrm{Ty}}{\mathrm{Tm}(\mathsf{A})\ \mathsf{sort}} \qquad t : A \quad \rightsquigarrow \quad t : \mathrm{Tm}(A)$$

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

Constructor rules

Term-level rules split between *constructors* and *destructors*

Constructor rules

Term-level rules split between *constructors* and *destructors*

Constructor rules In bidirectional typing, constructors support *checking*

Missing information recovered from the sort given as input

The sort of the rule should be a linear (Miller) *pattern* T^{P} on the missing arguments

Constructor rules

Term-level rules split between *constructors* and *destructors*

Constructor rules In bidirectional typing, constructors support *checking*

Missing information recovered from the sort given as input

The sort of the rule should be a linear (Miller) *pattern* T^{P} on the missing arguments

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

Constructor rules

Term-level rules split between *constructors* and *destructors*

Constructor rules In bidirectional typing, constructors support *checking*

Missing information recovered from the sort given as input

The sort of the rule should be a linear (Miller) *pattern* T^{P} on the missing arguments

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

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

Destructor rules & Rewrite rules

Destructor rules In bidirectional typing, destructors support *inference*

Missing arguments are recovered by inferring the *principal argument* $\mathtt{t} : T^{\mathsf{P}}$

Sort of the principal argument $\mathtt{t} : T^{\mathsf{P}}$ should be a pattern on the missing arguments

Destructor rules & Rewrite rules

Destructor rules In bidirectional typing, destructors support *inference*

Missing arguments are recovered by inferring the *principal argument* $\mathsf{t} : T^{\mathsf{P}}$

Sort of the principal argument $\mathsf{t} : T^{\mathsf{P}}$ should be a pattern on the missing arguments

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

Destructor rules & Rewrite rules

Destructor rules In bidirectional typing, destructors support *inference*

Missing arguments are recovered by inferring the *principal argument* $\mathtt{t} : T^{\mathsf{P}}$

Sort of the principal argument $\mathtt{t} : T^{\mathsf{P}}$ should be a pattern on the missing arguments

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

Destructors written in orange, constructors and sorts written in blue

Destructor rules & Rewrite rules

Destructor rules In bidirectional typing, destructors support *inference*

Missing arguments are recovered by inferring the *principal argument* $\mathtt{t} : T^{\mathsf{P}}$

Sort of the principal argument $\mathtt{t} : T^{\mathsf{P}}$ should be a pattern on the missing arguments

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

Destructors written in orange, constructors and sorts written in blue

Rewrite rules Specify the conversion of the theory

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

In general, of the form $l^{\mathsf{P}} \longmapsto r$

Full example, in the *formal* notation

Previous inference-rule notation can be desugared into a formal notation

Full example, in the *formal* notation

Previous inference-rule notation can be desugared into a formal notation

Theory $\mathbb{T}^{\flat}_{\lambda\Pi}$ A minimalistic type theory with only dependent functions

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

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

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

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

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

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

Full example, in the *formal* notation

Previous inference-rule notation can be desugared into a formal notation

Theory $\mathbb{T}^{\flat}_{\lambda\Pi}$ A minimalistic type theory with only dependent functions

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

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

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

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

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

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

In the rest of the talk, we use the inference-rule notation for readability ☺

Declarative type system

Declarative type system

Each $\mathbb{T}^\flat$ defines a declarative type system, with main judgment $\Gamma \vdash t : T$

Declarative type system

Each $\mathbb{T}^\flat$ defines a declarative type system, with main judgment $\Gamma \vdash t : T$

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

Declarative type system

Each $\mathbb{T}^\flat$ defines a declarative type system, with main judgment $\Gamma \vdash t : T$

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

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

Declarative type system

Each $\mathbb{T}^\flat$ defines a declarative type system, with main judgment $\Gamma \vdash t : T$

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

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

Declarative type system

Each $\mathbb{T}^\flat$ defines a declarative type system, with main judgment $\Gamma \vdash t : T$

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

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

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

Declarative type system

Each $\mathbb{T}^\flat$ defines a declarative type system, with main judgment $\Gamma \vdash t : T$

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

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

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

Declarative type system

Each $\mathbb{T}^\flat$ defines a declarative type system, with main judgment $\Gamma \vdash t : T$

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

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

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

Note that typing terms bottom-up requires guessing A and B

Meta-theory of the declarative system

We establish basic metaproperties of the declarative system, such as:

Meta-theory of the declarative system

We establish basic metaproperties of the declarative system, such as:

Weakening The following rule is admissible

$$\frac{\Delta \vdash t : T \qquad \Gamma \vdash}{\Gamma \vdash t : T}\ \Delta \sqsubseteq \Gamma$$

Meta-theory of the declarative system

We establish basic metaproperties of the declarative system, such as:

Weakening The following rule is admissible

$$\frac{\Delta \vdash t : T \qquad \Gamma \vdash}{\Gamma \vdash t : T}\ \Delta \sqsubseteq \Gamma$$

Substitution property The following rule is admissible

$$\frac{\Gamma \vdash \vec{u} : \Delta \qquad \Delta \vdash t : T}{\Gamma \vdash t[\vec{u}/\vec{x}_\Delta] : T[\vec{u}/\vec{x}_\Delta]}$$

Meta-theory of the declarative system

We establish basic metaproperties of the declarative system, such as:

Weakening The following rule is admissible

$$\frac{\Delta \vdash t : T \qquad \Gamma \vdash}{\Gamma \vdash t : T}\ \Delta \sqsubseteq \Gamma$$

Substitution property The following rule is admissible

$$\frac{\Gamma \vdash \vec{u} : \Delta \qquad \Delta \vdash t : T}{\Gamma \vdash t[\vec{u}/\vec{x}_\Delta] : T[\vec{u}/\vec{x}_\Delta]}$$

Not shown for one theory, but generically for all bidirectional theories $\mathbb{T}^\flat$

Bidirectional type system

Matching modulo for recovering arguments

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

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

Matching modulo for recovering arguments

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

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

If $@(t, u)$ is well-typed (in the declarative system), for some A, B we have

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

but how to recover A and B from U?

Matching modulo for recovering arguments

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

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

If $@(t, u)$ is well-typed (in the declarative system), for some A, B we have

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

but how to recover A and B from U?

Solution Given T^{P} and U, we have an algorithmic matching judgment

$$T^{\mathsf{P}} \prec U \rightsquigarrow \vec{x}_1.t_1/\mathsf{x}_1, \ldots, \vec{x}_k.t_k/\mathsf{x}_k$$

that tries to compute $t_1, \ldots, t_k$ s.t. $T^{\mathsf{P}}[\vec{x}_1.t_1/\mathsf{x}_1, \ldots, \vec{x}_k.t_k/\mathsf{x}_k] \equiv U$

Bidirectional syntax

Not all unannotated terms can be algorithmically typed

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

Bidirectional syntax

Not all unannotated terms can be algorithmically typed

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

Limitation not specific to bidirectional typing, undecidable in general!

Bidirectional syntax

Not all unannotated terms can be algorithmically typed

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

Limitation not specific to bidirectional typing, undecidable in general!

Bidirectional system defined over *inferrable* and *checkable* terms

$$\boxed{\mathsf{Tm}^{\mathsf{i}}} \ni \qquad t^{\mathsf{i}}, u^{\mathsf{i}} ::= x \mid d(t^{\mathsf{i}}, \vec{x}_1.u_1^{\mathsf{c}}, \dots, \vec{x}_k.u_k^{\mathsf{c}}) \mid t^{\mathsf{c}} :: T^{\mathsf{c}}$$

$$\boxed{\mathsf{Tm}^{\mathsf{c}}} \ni \qquad t^{\mathsf{c}}, u^{\mathsf{c}} ::= c(\vec{x}_1.u_1^{\mathsf{c}}, \dots, \vec{x}_k.u_k^{\mathsf{c}}) \mid \underline{t}^{\mathsf{i}}$$

Bidirectional syntax

Not all unannotated terms can be algorithmically typed

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

Limitation not specific to bidirectional typing, undecidable in general!

Bidirectional system defined over *inferrable* and *checkable* terms

$$\boxed{\mathrm{Tm}^{\mathrm{i}}} \ni \qquad t^{\mathrm{i}}, u^{\mathrm{i}} ::= x \mid d(t^{\mathrm{i}}, \vec{x}_1.u^{\mathrm{c}}_1, \ldots, \vec{x}_k.u^{\mathrm{c}}_k) \mid t^{\mathrm{c}} :: T^{\mathrm{c}}$$

$$\boxed{\mathrm{Tm}^{\mathrm{c}}} \ni \qquad t^{\mathrm{c}}, u^{\mathrm{c}} ::= c(\vec{x}_1.u^{\mathrm{c}}_1, \ldots, \vec{x}_k.u^{\mathrm{c}}_k) \mid \underline{t}^{\mathrm{i}}$$

When destructor meets a constructor, we need an *ascription*, in the style of McBride:

$$@(\lambda(x.t^{\mathrm{c}}) :: T^{\mathrm{c}}, u^{\mathrm{c}})$$

Bidirectional typing rules

Each $\mathbb{T}^\flat$ defines a bid. type system with judgments $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$

Bidirectional typing rules

Each $\mathbb{T}^\flat$ defines a bid. type system with judgments $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$

The main typing rules instantiate the schematic rules of $\mathbb{T}^\flat$:

Bidirectional typing rules

Each $\mathbb{T}^\flat$ defines a bid. type system with judgments $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$

The main typing rules instantiate the schematic rules of $\mathbb{T}^\flat$:

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

Bidirectional typing rules

Each $\mathbb{T}^\flat$ defines a bid. type system with judgments $\Gamma \vdash t^{c} \Leftarrow T$ and $\Gamma \vdash t^{i} \Rightarrow T$

The main typing rules instantiate the schematic rules of $\mathbb{T}^\flat$:

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

Bidirectional typing rules

Each $\mathbb{T}^\flat$ defines a bid. type system with judgments $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$

The main typing rules instantiate the schematic rules of $\mathbb{T}^\flat$:

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

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

Bidirectional typing rules

Each $\mathbb{T}^\flat$ defines a bid. type system with judgments $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$

The main typing rules instantiate the schematic rules of $\mathbb{T}^\flat$:

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

$$\frac{\mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \qquad \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \qquad \mathsf{u} : \mathrm{Tm}(\mathsf{A})}{@(\mathsf{t}, \mathsf{u}) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\})} \quad \rightsquigarrow \quad \frac{\Gamma \vdash t^{\mathsf{i}} \Rightarrow T \qquad \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) < T \rightsquigarrow A/\mathsf{A},\ x.B/\mathsf{B} \qquad \Gamma \vdash u^{\mathsf{c}} \Leftarrow \mathrm{Tm}(A)}{\Gamma \vdash @(t^{\mathsf{i}}, u^{\mathsf{c}}) \Rightarrow \mathrm{Tm}(B[\ulcorner u^{\mathsf{c}} \urcorner / x])^2}$$

[2] Given t^{i} or u^{c}, I write $\ulcorner t^{\mathsf{i}} \urcorner$ or $\ulcorner u^{\mathsf{c}} \urcorner$ for the underlying regular term.

Bidirectional typing rules

Each $\mathbb{T}^\flat$ defines a bid. type system with judgments $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$

The main typing rules instantiate the schematic rules of $\mathbb{T}^\flat$:

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

$$\frac{\mathsf{A} : \mathrm{Ty} \qquad x : \mathrm{Tm}(\mathsf{A}) \vdash \mathsf{B} : \mathrm{Ty} \quad\quad \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) \qquad \mathsf{u} : \mathrm{Tm}(\mathsf{A})}{@(\mathsf{t}, \mathsf{u}) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\})} \quad \rightsquigarrow \quad \frac{\Gamma \vdash t^{\mathsf{i}} \Rightarrow T \quad\quad \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) < T \rightsquigarrow A/\mathsf{A},\ x.B/\mathsf{B} \quad\quad \Gamma \vdash u^{\mathsf{c}} \Leftarrow \mathrm{Tm}(A)}{\Gamma \vdash @(t^{\mathsf{i}}, u^{\mathsf{c}}) \Rightarrow \mathrm{Tm}(B[\ulcorner u^{\mathsf{c}} \urcorner / x])^2}$$

Note that no more need to guess A and B!

[2] Given t^{i} or u^{c}, I write $\ulcorner t^{\mathsf{i}} \urcorner$ or $\ulcorner u^{\mathsf{c}} \urcorner$ for the underlying regular term.

Equivalence with declarative typing

(Suppose underlying $\mathbb{T}^\flat$ is *valid*)

Equivalence with declarative typing

(Suppose underlying $\mathbb{T}^\flat$ is *valid*)

Soundness If $\Gamma \vdash$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$ then $\Gamma \vdash \ulcorner t^{\mathsf{i}} \urcorner : T$

If $\Gamma \vdash T$ sort and $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ then $\Gamma \vdash \ulcorner t^{\mathsf{c}} \urcorner : T$

Equivalence with declarative typing

(Suppose underlying $\mathbb{T}^\flat$ is *valid*)

Soundness If $\Gamma \vdash$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$ then $\Gamma \vdash \ulcorner t^{\mathsf{i}} \urcorner : T$

If $\Gamma \vdash T$ sort and $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ then $\Gamma \vdash \ulcorner t^{\mathsf{c}} \urcorner : T$

Annotability If $\Gamma \vdash t : T$ then for some u^{c} with $\ulcorner u^{\mathsf{c}} \urcorner = t$ we have $\Gamma \vdash u^{\mathsf{c}} \Leftarrow T$

Equivalence with declarative typing

(Suppose underlying $\mathbb{T}^\flat$ is *valid*)

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

Annotability If $\Gamma \vdash t : T$ then for some u^{c} with $\ulcorner u^{\mathsf{c}} \urcorner = t$ we have $\Gamma \vdash u^{\mathsf{c}} \Leftarrow T$

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

Equivalence with declarative typing

(Suppose underlying $\mathbb{T}^\flat$ is *valid*)

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

Annotability If $\Gamma \vdash t : T$ then for some u^{c} with $\ulcorner u^{\mathsf{c}} \urcorner = t$ we have $\Gamma \vdash u^{\mathsf{c}} \Leftarrow T$

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

Not shown for one particular theory, but for *all* instances of our framework

More examples

Dependent sums

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

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

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

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

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

$$\mathrm{proj}_1(\mathrm{pair}(\mathsf{t}, \mathsf{u})) \longmapsto \mathsf{t}$$

$$\mathrm{proj}_2(\mathrm{pair}(\mathsf{t}, \mathsf{u})) \longmapsto \mathsf{u}$$

Lists

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

$$\frac{\mathtt{A} : \mathrm{Ty}}{\mathrm{List}(\mathtt{A}) : \mathrm{Ty}} \qquad \frac{\mathtt{A} : \mathrm{Ty}}{\mathrm{nil} : \mathrm{Tm}(\mathrm{List}(\mathtt{A}))} \qquad \frac{\mathtt{A} : \mathrm{Ty} \qquad \mathtt{x} : \mathrm{Tm}(\mathtt{A}) \qquad \mathtt{l} : \mathrm{Tm}(\mathrm{List}(\mathtt{A}))}{\mathrm{cons}(\mathtt{x}, \mathtt{l}) : \mathrm{Tm}(\mathrm{List}(\mathtt{A}))}$$

$$\frac{\begin{array}{c}\mathtt{A} : \mathrm{Ty} \qquad \mathtt{l} : \mathrm{Tm}(\mathrm{List}(\mathtt{A})) \qquad x : \mathrm{Tm}(\mathrm{List}(\mathtt{A})) \vdash \mathsf{P} : \mathrm{Ty} \qquad \mathtt{pnil} : \mathrm{Tm}(\mathsf{P}\{\mathrm{nil}\}) \\ x : \mathrm{Tm}(\mathtt{A}), y : \mathrm{Tm}(\mathrm{List}(\mathtt{A})), z : \mathrm{Tm}(\mathsf{P}\{y\}) \vdash \mathsf{pcons} : \mathrm{Tm}(\mathsf{P}\{\mathrm{cons}(x, y)\})\end{array}}{\mathrm{ListRec}(\mathtt{l}, x.\mathsf{P}\{x\}, \mathtt{pnil}, xyz.\mathsf{pcons}\{x, y, z\}) : \mathrm{Tm}(\mathsf{P}\{\mathtt{l}\})}$$

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

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

$$\mathsf{pcons}\{\mathtt{x}, \mathtt{l}, \mathrm{ListRec}(\mathtt{l}; x.\mathsf{P}\{x\}, \mathtt{pnil}, xyz.\mathsf{pcons}\{x, y, z\})\}$$

Equality

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

$$\frac{\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathbf{Tm}(\mathsf{A})}{\mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) : \mathbf{Ty}} \qquad \frac{\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A})}{\mathbf{refl} : \mathbf{Tm}(\mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{a}))}$$

$$\frac{\begin{array}{c}\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{t} : \mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) \\ x : \mathbf{Tm}(\mathsf{A}), y : \mathbf{Tm}(\mathbf{Eq}(\mathsf{A}, \mathsf{a}, x)) \vdash \mathsf{P} : \mathbf{Ty} \qquad \mathsf{p} : \mathbf{Tm}(\mathsf{P}\{\mathsf{a}, \mathbf{refl}\})\end{array}}{\mathsf{J}(\mathsf{t}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) : \mathbf{Tm}(\mathsf{P}\{\mathsf{b}, \mathsf{t}\})}$$

$$\mathsf{J}(\mathbf{refl}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) \longmapsto \mathsf{p}$$

Equality

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

$$\frac{\mathsf{A} : \mathrm{Ty} \qquad \mathsf{a} : \mathrm{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathrm{Tm}(\mathsf{A})}{\mathrm{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) : \mathrm{Ty}} \qquad \frac{\mathsf{A} : \mathrm{Ty} \qquad \mathsf{a} : \mathrm{Tm}(\mathsf{A})}{\mathrm{refl} : \mathrm{Tm}(\mathrm{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{a}))}$$

$$\frac{\begin{array}{c}\mathsf{A} : \mathrm{Ty} \qquad \mathsf{a} : \mathrm{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathrm{Tm}(\mathsf{A}) \qquad \mathsf{t} : \mathrm{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) \\ x : \mathrm{Tm}(\mathsf{A}), y : \mathrm{Tm}(\mathrm{Eq}(\mathsf{A}, \mathsf{a}, x)) \vdash \mathsf{P} : \mathrm{Ty} \qquad \mathsf{p} : \mathrm{Tm}(\mathsf{P}\{\mathsf{a}, \mathrm{refl}\})\end{array}}{\mathrm{J}(\mathsf{t}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) : \mathrm{Tm}(\mathsf{P}\{\mathsf{b}, \mathsf{t}\})}$$

$$\mathrm{J}(\mathrm{refl}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) \longmapsto \mathsf{p}$$

Equality

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

$$\frac{\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathbf{Tm}(\mathsf{A})}{\mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) : \mathbf{Ty}} \qquad \frac{\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A})}{\mathbf{refl} : \mathbf{Tm}(\mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{a}))}$$

$$\frac{\begin{array}{cccc}\mathsf{A} : \mathbf{Ty} & \mathsf{a} : \mathbf{Tm}(\mathsf{A}) & \mathsf{b} : \mathbf{Tm}(\mathsf{A}) & \mathsf{t} : \mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) \\ \multicolumn{3}{c}{x : \mathbf{Tm}(\mathsf{A}), y : \mathbf{Tm}(\mathbf{Eq}(\mathsf{A}, \mathsf{a}, x)) \vdash \mathsf{P} : \mathbf{Ty}} & \mathsf{p} : \mathbf{Tm}(\mathsf{P}\{\mathsf{a}, \mathbf{refl}\})\end{array}}{\mathbf{J}(\mathsf{t}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) : \mathbf{Tm}(\mathsf{P}\{\mathsf{b}, \mathsf{t}\})}$$

$$\mathbf{J}(\mathbf{refl}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) \longmapsto \mathsf{p}$$

Constructor rules need to be extended to account for *indexed* types:

Equality

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

$$\frac{\mathsf{A} : \mathrm{Ty} \qquad \mathsf{a} : \mathrm{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathrm{Tm}(\mathsf{A})}{\mathrm{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) : \mathrm{Ty}} \qquad \frac{\mathsf{A} : \mathrm{Ty} \qquad \mathsf{a} : \mathrm{Tm}(\mathsf{A})}{\mathrm{refl} : \mathrm{Tm}(\mathrm{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{a}))}$$

$$\frac{\begin{array}{c}\mathsf{A} : \mathrm{Ty} \qquad \mathsf{a} : \mathrm{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathrm{Tm}(\mathsf{A}) \qquad \mathsf{t} : \mathrm{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) \\ x : \mathrm{Tm}(\mathsf{A}), y : \mathrm{Tm}(\mathrm{Eq}(\mathsf{A}, \mathsf{a}, x)) \vdash \mathsf{P} : \mathrm{Ty} \qquad \mathsf{p} : \mathrm{Tm}(\mathsf{P}\{\mathsf{a}, \mathrm{refl}\})\end{array}}{\mathrm{J}(\mathsf{t}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) : \mathrm{Tm}(\mathsf{P}\{\mathsf{b}, \mathsf{t}\})}$$

$$\mathrm{J}(\mathrm{refl}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) \longmapsto \mathsf{p}$$

Constructor rules need to be extended to account for *indexed* types:

- Solution 1: Ad-hoc treatement of indices (see the TOPLAS paper)

Equality

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

$$\frac{\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathbf{Tm}(\mathsf{A})}{\mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) : \mathbf{Ty}} \qquad \frac{\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A})}{\mathbf{refl} : \mathbf{Tm}(\mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{a}))}$$

$$\frac{\begin{array}{c}\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{t} : \mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) \\ x : \mathbf{Tm}(\mathsf{A}), y : \mathbf{Tm}(\mathbf{Eq}(\mathsf{A}, \mathsf{a}, x)) \vdash \mathsf{P} : \mathbf{Ty} \qquad \mathsf{p} : \mathbf{Tm}(\mathsf{P}\{\mathsf{a}, \mathbf{refl}\})\end{array}}{\mathbf{J}(\mathsf{t}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) : \mathbf{Tm}(\mathsf{P}\{\mathsf{b}, \mathsf{t}\})}$$

$$\mathbf{J}(\mathbf{refl}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) \longmapsto \mathsf{p}$$

Constructor rules need to be extended to account for *indexed* types:

- Solution 1: Ad-hoc treatement of indices (see the TOPLAS paper)
- Solution 2: Equational premises (see my PhD thesis & implementation)

Equality

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

$$\frac{\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathbf{Tm}(\mathsf{A})}{\mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) : \mathbf{Ty}} \qquad \frac{\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{a} \equiv \mathsf{b}}{\mathbf{refl} : \mathbf{Tm}(\mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}))}$$

$$\frac{\begin{array}{c}\mathsf{A} : \mathbf{Ty} \qquad \mathsf{a} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{b} : \mathbf{Tm}(\mathsf{A}) \qquad \mathsf{t} : \mathbf{Eq}(\mathsf{A}, \mathsf{a}, \mathsf{b}) \\ x : \mathbf{Tm}(\mathsf{A}), y : \mathbf{Tm}(\mathbf{Eq}(\mathsf{A}, \mathsf{a}, x)) \vdash \mathsf{P} : \mathbf{Ty} \qquad \mathsf{p} : \mathbf{Tm}(\mathsf{P}\{\mathsf{a}, \mathbf{refl}\})\end{array}}{\mathbf{J}(\mathsf{t}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) : \mathbf{Tm}(\mathsf{P}\{\mathsf{b}, \mathsf{t}\})}$$

$$\mathbf{J}(\mathbf{refl}, xy.\mathsf{P}\{x, y\}, \mathsf{p}) \longmapsto \mathsf{p}$$

Constructor rules need to be extended to account for *indexed* types:

- Solution 1: Ad-hoc treatement of indices (see the TOPLAS paper)
- Solution 2: Equational premises (see my PhD thesis & implementation)

Vectors

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

$$\frac{\mathtt{A} : \mathrm{Ty} \qquad \mathtt{n} : \mathrm{Tm}(\mathrm{Nat})}{\mathrm{Vec}(\mathtt{A}, \mathtt{n}) : \mathrm{Ty}} \qquad \frac{\begin{array}{c}\mathtt{A} : \mathrm{Ty} \qquad \mathtt{n} : \mathrm{Tm}(\mathrm{Nat}) \\ \mathtt{n} \equiv 0 : \mathrm{Tm}(\mathrm{Nat})\end{array}}{\mathrm{nil} : \mathrm{Tm}(\mathrm{Vec}(\mathtt{A}, \mathtt{n}))} \qquad \frac{\begin{array}{c}\mathtt{A} : \mathrm{Ty} \qquad \mathtt{n}, \mathtt{m} : \mathrm{Tm}(\mathrm{Nat}) \qquad \mathtt{x} : \mathrm{Tm}(\mathtt{A}) \\ \mathtt{l} : \mathrm{Tm}(\mathrm{Vec}(\mathtt{A}, \mathtt{m})) \qquad \mathtt{n} \equiv \mathrm{S}(\mathtt{m}) : \mathrm{Tm}(\mathrm{Nat})\end{array}}{\mathrm{cons}(\mathtt{m}, \mathtt{x}, \mathtt{l}) : \mathrm{Tm}(\mathrm{Vec}(\mathtt{A}, \mathtt{n}))}$$

$$\frac{\begin{array}{c}\mathtt{A} : \mathrm{Ty} \qquad \mathtt{n} : \mathrm{Tm}(\mathrm{Nat}) \qquad \mathtt{l} : \mathrm{Tm}(\mathrm{Vec}(\mathtt{A}, \mathtt{n})) \\ x : \mathrm{Tm}(\mathrm{Nat}), y : \mathrm{Tm}(\mathrm{Vec}(\mathtt{A}, x)) \vdash \mathtt{P} : \mathrm{Ty} \qquad \mathtt{pnil} : \mathrm{Tm}(\mathtt{P}\{0, \mathrm{nil}\}) \\ x : \mathrm{Tm}(\mathrm{Nat}), y : \mathrm{Tm}(\mathtt{A}), z : \mathrm{Tm}(\mathrm{Vec}(\mathtt{A}, x)), w : \mathrm{Tm}(\mathtt{P}\{x, z\}) \vdash \mathtt{pcons} : \mathrm{Tm}(\mathtt{P}\{\mathrm{S}(x), \mathrm{cons}(x, y, z)\})\end{array}}{\mathrm{VecRec}(\mathtt{l}, xy.\mathtt{P}\{x, y\}, \mathtt{pnil}, xyzw.\mathtt{pcons}\{x, y, z, w\}) : \mathrm{Tm}(\mathtt{P}\{\mathtt{n}, \mathtt{l}\})}$$

$$\mathrm{VecRec}(\mathrm{nil}, x.\mathtt{P}\{x\}, \mathtt{pnil}, xyzw.\mathtt{pcons}\{x, y, z, w\}) \longmapsto \mathtt{pnil}$$

$$\mathrm{VecRec}(\mathrm{cons}(\mathtt{n}, \mathtt{x}, \mathtt{l}), x.\mathtt{P}\{x\}, \mathtt{pnil}, xyzw.\mathtt{pcons}\{x, y, z, w\}) \longmapsto$$

$$\mathtt{pcons}\{\mathtt{n}, \mathtt{x}, \mathtt{l}, \mathrm{VecRec}(\mathtt{l}, x.\mathtt{P}\{x\}, \mathtt{pnil}, xyzw.\mathtt{pcons}\{x, y, z, w\})\}$$

Other examples

In the implementation at

`https://github.com/thiagofelicissimo/BiTTs`

you can also find:

- Tarski-, Russell- and Coquand-style universes, with/without cumulativity and universe polymorphism
- Flavous of Observational Type Theory
- A variant of Exceptional Type Theory (type theory with exceptions)

Conclusion

Conclusion

We have given a generic account of bidirectional typing for a class of type theories

Conclusion

We have given a generic account of bidirectional typing for a class of type theories

Beyond theoretic interest of understanding bidirectional typing formally, implementation BiTTs can also be used in practice for rechecking proofs

Conclusion

We have given a generic account of bidirectional typing for a class of type theories

Beyond theoretic interest of understanding bidirectional typing formally, implementation BiTTs can also be used in practice for rechecking proofs

Future work

Conclusion

We have given a generic account of bidirectional typing for a class of type theories

Beyond theoretic interest of understanding bidirectional typing formally, implementation BiTTs can also be used in practice for rechecking proofs

Future work

1. Remove reliance on rewriting, to allow for η-rules and proof irrelevance
   Implementation would require a generic type-directed equality-checker

Conclusion

We have given a generic account of bidirectional typing for a class of type theories

Beyond theoretic interest of understanding bidirectional typing formally, implementation BiTTs can also be used in practice for rechecking proofs

Future work

1. Remove reliance on rewriting, to allow for η-rules and proof irrelevance
   Implementation would require a generic type-directed equality-checker
2. Develop a framework for formalizing decidability of typing proofs
   Derivation of syntax, substitution, typing rules, etc automatic from $\mathbb{T}^\flat$
   To prove decidability of typing, only need to show $\mathbb{T}^\flat$ is valid and s.n.

Conclusion

We have given a generic account of bidirectional typing for a class of type theories

Beyond theoretic interest of understanding bidirectional typing formally, implementation BiTTs can also be used in practice for rechecking proofs

Future work

1. Remove reliance on rewriting, to allow for η-rules and proof irrelevance
   Implementation would require a generic type-directed equality-checker
2. Develop a framework for formalizing decidability of typing proofs
   Derivation of syntax, substitution, typing rules, etc automatic from $\mathbb{T}^\flat$
   To prove decidability of typing, only need to show $\mathbb{T}^\flat$ is valid and s.n.

Thank you for your attention!