Generic Bidirectional Typing for Dependent Type Theories

Thiago Felicissimo

WG6 meeting in Leuven

April 4, 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

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

What one gets when seeing type theory as an algebraic 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

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

What one gets when seeing type theory as an algebraic 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 seeing type theory as an algebraic 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

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

What one gets when seeing type theory as an algebraic theory

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

Non-annotated syntax restores usability by eliding parameter annotations

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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?

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

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

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

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

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

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

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

This work *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 work *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 work *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

BiTTs: A theory-independent bidirectional type-checker

Our framework not only of theoretic interest, can also have practical applications

BiTTs: A theory-independent bidirectional type-checker

Our framework not only of theoretic interest, can also have practical applications

Implemented in the theory-independent bidirectional type-checker BiTTs

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

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 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)))))
```

BiTTs: A theory-independent bidirectional type-checker

Our framework not only of theoretic interest, can also have practical applications

Implemented in the theory-independent bidirectional type-checker BiTTs

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

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 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)))))
```

Many theories supported: flavours of MLTT, OTT, HOL (see the implementation)

BiTTs: A theory-independent bidirectional type-checker

Our framework not only of theoretic interest, can also have practical applications

Implemented in the theory-independent bidirectional type-checker BiTTs

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

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 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)))))
```

Many theories supported: flavours of MLTT, OTT, HOL (see the implementation)

Compared with other theory-independent type-checkers (Dedukti, Andromeda) non-annotated syntax should allow for better performances

The theories

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 A *sort*[1] is a term T that can appear in the right of typing judgment $t : T$

Used to represent the judgment forms of the theory (as in GATs, SOGATs, …)

[1] I avoid calling them "types" to prevent a name clash with the types of the object theories

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 A *sort*[1] is a term T that can appear in the right of typing judgment $t : T$

Used to represent the judgment forms of the theory (as in GATs, SOGATs, ...)

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

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

[1] I avoid calling them "types" to prevent a name clash with the types of the object theories

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 A *sort*[1] is a term T that can appear in the right of typing judgment $t : T$

Used to represent the judgment forms of the theory (as in GATs, SOGATs, ...)

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

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

Formally, of the form $c(\Theta)$ sort, with Θ metavariable context representing premises.

Example in formal notation: $\mathsf{Ty}(\cdot)$ sort and $\mathsf{Tm}(\mathsf{A} : \mathsf{Ty})$ sort

[1] I avoid calling them "types" to prevent a name clash with the types of the object theories

The theories

Constructor rules In bidirectional typing, constructors support *type-checking*, so missing annotations recovered from the sort given as input.

The theories

Constructor rules In bidirectional typing, constructors support *type-checking*, so missing annotations recovered from the sort given as input.

Two groups of premises: Θ_1 erased and Θ_2 kept in the syntax.

Sort of the rule should be a linear pattern containing metavariables of Θ_1.

The theories

Constructor rules In bidirectional typing, constructors support *type-checking*, so missing annotations recovered from the sort given as input.

Two groups of premises: Θ_1 erased and Θ_2 kept in the syntax.

Sort of the rule should be a linear pattern containing metavariables of Θ_1.

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

The theories

Constructor rules In bidirectional typing, constructors support *type-checking*, so missing annotations recovered from the sort given as input.

Two groups of premises: Θ_1 erased and Θ_2 kept in the syntax.

Sort of the rule should be a linear pattern containing metavariables of Θ_1.

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

Formally, constructor rules of the form $c(\Theta_1; \Theta_2) : U^{\mathsf{P}}$, with U^{P} pattern on Θ_1

Example in formal notation: $\Pi(\cdot;\ \mathsf{A} : \mathrm{Ty},\ \mathsf{B}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Ty}) : \mathrm{Ty}$ and $\lambda(\mathsf{A} : \mathrm{Ty},\ \mathsf{B}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Ty};\ \mathtt{t}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Tm}(\mathsf{B}\{x\})) : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}))$.

The theories

Destructor rules In bidirectional typing, destructors support *type-inference*, so missing arguments are recovered by inferring a *principal argument*.

The theories

Destructor rules In bidirectional typing, destructors support *type-inference*, so missing arguments are recovered by inferring a *principal argument*.

Two groups of premises: Θ_1 erased and Θ_2 kept in the syntax.

And a principal argument $\mathsf{x} : T^{\mathsf{P}}$, with T a pattern on Θ_1.

The theories

Destructor rules In bidirectional typing, destructors support *type-inference*, so missing arguments are recovered by inferring a *principal argument*.

Two groups of premises: Θ_1 erased and Θ_2 kept in the syntax.

And a principal argument $\mathsf{x} : T^{\mathsf{P}}$, with T a pattern on Θ_1.

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

The theories

Destructor rules In bidirectional typing, destructors support *type-inference*, so missing arguments are recovered by inferring a *principal argument*.

Two groups of premises: Θ_1 erased and Θ_2 kept in the syntax.

And a principal argument $\mathsf{x} : T^{\mathsf{P}}$, with T a pattern on Θ_1.

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

Formally, of the form $d(\Theta_1;\ \mathsf{x} : T^{\mathsf{P}};\ \Theta_2) : U$, with T a pattern on Θ_1

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

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 $d(c(\mathbf{t}_1^{\mathsf{P}}), \mathbf{t}_2^{\mathsf{P}}) \longmapsto r$ with $(\mathrm{metas}(\mathbf{t}_1^{\mathsf{P}}) \cap \mathrm{metas}(\mathbf{t}_2^{\mathsf{P}}) = \emptyset)$.

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 $d(c(\mathbf{t}_1^{\mathsf{P}}), \mathbf{t}_2^{\mathsf{P}}) \longmapsto r$ with $(\mathsf{metas}(\mathbf{t}_1^{\mathsf{P}}) \cap \mathsf{metas}(\mathbf{t}_2^{\mathsf{P}}) = \emptyset)$.

Condition: no two left-hand sides unify.

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

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 $d(c(\mathbf{t}_1^{\mathsf{P}}), \mathbf{t}_2^{\mathsf{P}}) \longmapsto r$ with $(\mathrm{metas}(\mathbf{t}_1^{\mathsf{P}}) \cap \mathrm{metas}(\mathbf{t}_2^{\mathsf{P}}) = \emptyset)$.

Condition: no two left-hand sides unify.

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

Full example Theory $\mathbb{T}_{\lambda\Pi}$.

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

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

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

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

Declarative type system

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

Declarative type system

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

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

$$d(\Xi_1; \mathsf{x} : T; \Xi_2) : U \in \mathbb{T} \; \frac{\Theta; \Gamma \vdash \mathbf{t}, t, \mathbf{u} : \Xi_1.(\mathsf{x} : T).\Xi_2}{\Theta; \Gamma \vdash d(t, \mathbf{u}) : U[\mathbf{t}, t, \mathbf{u}]}\;\textsc{Dest}$$

Declarative type system

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

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

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

(for $@(\mathtt{A} : \mathbf{Ty},\ \mathtt{B}\{x : \mathbf{Tm}(\mathtt{A})\} : \mathbf{Ty};\ \mathtt{t} : \mathbf{Tm}(\Pi(\mathtt{A}, x.\mathtt{B}\{x\}));\ \mathtt{u} : \mathbf{Tm}(\mathtt{A})) : \mathbf{Tm}(\mathtt{B}\{\mathtt{u}\}) \in \mathbb{T}_{\lambda\Pi}$)

Declarative type system

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

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

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

(for $@(\mathsf{A} : \mathrm{Ty},\ \mathsf{B}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Ty};\ \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}));\ \mathsf{u} : \mathrm{Tm}(\mathsf{A})) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\}) \in \mathbb{T}_{\lambda\Pi}$)

Reading bottom-up, requires guessing A and B

Declarative type system

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

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

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

(for $@(\mathtt{A} : \mathrm{Ty},\ \mathtt{B}\{x : \mathrm{Tm}(\mathtt{A})\} : \mathrm{Ty};\ \mathtt{t} : \mathrm{Tm}(\Pi(\mathtt{A}, x.\mathtt{B}\{x\}));\ \mathtt{u} : \mathrm{Tm}(\mathtt{A})) : \mathrm{Tm}(\mathtt{B}\{\mathtt{u}\}) \in \mathbb{T}_{\lambda\Pi}$)

Reading bottom-up, requires guessing A and B

Properties of the declarative system Weakening, substitution property, sorts are well-typed, subject reduction, etc (see the paper)

Bidirectional typing system

Matching modulo rewriting

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

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

2

Matching modulo rewriting

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?

2

Matching modulo rewriting

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?

Solution We define an algorithmic[2] matching judgment $T^{\mathsf{P}} \prec U \rightsquigarrow \mathbf{v}$

We have $T^{\mathsf{P}} \prec U \rightsquigarrow \mathbf{v}$ iff $T^{\mathsf{P}}[\mathbf{v}] \equiv U$

[2]Decidable when U is normalizing

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

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

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

$$\boxed{\mathrm{Tm}^{\mathsf{i}}} \ni \qquad t^{\mathsf{i}}, u^{\mathsf{i}} ::= x \mid d(t^{\mathsf{i}}, \mathbf{t}^{\mathsf{c}}) \mid t^{\mathsf{c}} :: T^{\mathsf{c}}$$

$$\boxed{\mathrm{Tm}^{\mathsf{c}}} \ni \qquad t^{\mathsf{c}}, u^{\mathsf{c}} ::= c(\mathbf{t}^{\mathsf{c}}) \mid \underline{t}^{\mathsf{i}}$$

$$\boxed{\mathrm{MSub}^{\mathsf{c}}} \ni \qquad \mathbf{t}^{\mathsf{c}}, \mathbf{u}^{\mathsf{c}} ::= \epsilon \mid \mathbf{t}^{\mathsf{c}}, \vec{x}.t^{\mathsf{c}}$$

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 system defined over *inferrable* and *checkable* terms

$$\boxed{\mathsf{Tm}^{\mathsf{i}}} \ni \qquad t^{\mathsf{i}}, u^{\mathsf{i}} ::= x \mid d(t^{\mathsf{i}}, \mathbf{t}^{\mathsf{c}}) \mid t^{\mathsf{c}} :: T^{\mathsf{c}}$$

$$\boxed{\mathsf{Tm}^{\mathsf{c}}} \ni \qquad t^{\mathsf{c}}, u^{\mathsf{c}} ::= c(\mathbf{t}^{\mathsf{c}}) \mid \underline{t}^{\mathsf{i}}$$

$$\boxed{\mathsf{MSub}^{\mathsf{c}}} \ni \qquad \mathbf{t}^{\mathsf{c}}, \mathbf{u}^{\mathsf{c}} ::= \epsilon \mid \mathbf{t}^{\mathsf{c}}, \vec{x}.t^{\mathsf{c}}$$

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

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

Bidirectional type system

Each $\mathbb{T}$ defines a bidirectional system. Main judgments: $\Gamma \vdash t^{\mathsf{c}} \Leftarrow T$ and $\Gamma \vdash t^{\mathsf{i}} \Rightarrow T$

Bidirectional type system

Each $\mathbb{T}$ defines a bidirectional system. Main 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 $\mathbb{T}$:[3]

$$d(\Xi_1; \mathsf{t} : T; \Xi_2) : U \in \mathbb{T} \quad \frac{\begin{array}{c}\textsc{Dest} \\ \Gamma \vdash t^{\mathsf{i}} \Rightarrow T' \qquad T \prec T' \rightsquigarrow \mathbf{v} \\ \Gamma \mid (\mathbf{v}, \ulcorner t^{\mathsf{i}} \urcorner) : (\Xi_1, \mathsf{x} : T) \vdash \mathbf{u}^{\mathsf{c}} \Leftarrow \Xi_2\end{array}}{\Gamma \vdash d(t^{\mathsf{i}}, \mathbf{u}^{\mathsf{c}}) \Rightarrow U[\mathbf{v}, \ulcorner t^{\mathsf{i}} \urcorner, \ulcorner \mathbf{u}^{\mathsf{c}} \urcorner]}$$

[3]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 type system

Each $\mathbb{T}$ defines a bidirectional system. Main 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 $\mathbb{T}$:[3]

$$\frac{\Gamma \vdash t^{\mathsf{i}} \Rightarrow T' \qquad \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) < T' \rightsquigarrow A/\mathsf{A},\ x.B/\mathsf{B} \qquad \Gamma \vdash u^{\mathsf{c}} \Leftarrow \mathrm{Tm}(A)}{\Gamma \vdash @(t^{\mathsf{i}}, u^{\mathsf{c}}) \Rightarrow \mathrm{Tm}(B[\ulcorner u^{\mathsf{c}} \urcorner / x])}$$

(for $@(\mathsf{A} : \mathrm{Ty},\ \mathsf{B}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Ty};\ \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}));\ \mathsf{u} : \mathrm{Tm}(\mathsf{A})) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\}) \in \mathbb{T}_{\lambda\Pi}$)

[3]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 type system

Each $\mathbb{T}$ defines a bidirectional system. Main 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 $\mathbb{T}$:[3]

$$\frac{\Gamma \vdash t^{\mathsf{i}} \Rightarrow T' \qquad \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\})) < T' \rightsquigarrow A/\mathsf{A},\ x.B/\mathsf{B} \qquad \Gamma \vdash u^{\mathsf{c}} \Leftarrow \mathrm{Tm}(A)}{\Gamma \vdash @(t^{\mathsf{i}}, u^{\mathsf{c}}) \Rightarrow \mathrm{Tm}(B[\ulcorner u^{\mathsf{c}} \urcorner / x])}$$

(for $@(\mathsf{A} : \mathrm{Ty},\ \mathsf{B}\{x : \mathrm{Tm}(\mathsf{A})\} : \mathrm{Ty};\ \mathsf{t} : \mathrm{Tm}(\Pi(\mathsf{A}, x.\mathsf{B}\{x\}));\ \mathsf{u} : \mathrm{Tm}(\mathsf{A})) : \mathrm{Tm}(\mathsf{B}\{\mathsf{u}\}) \in \mathbb{T}_{\lambda\Pi}$)

Reading bottom-up, no more need to guess A and B!

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

Correctness with respect to declarative typing

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

Correctness with respect to declarative typing

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

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

Correctness with respect to declarative typing

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

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

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

Correctness with respect to declarative typing

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

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

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.

More examples

Dependent sums

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

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

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

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

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

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

Lists

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

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

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

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

Definition of constructor rules needs to be modified to account for indexed types (see the paper)

Equality

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

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

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

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

Definition of constructor rules needs to be modified to account for indexed types (see the paper)

Vectors

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

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

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

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

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

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

Other examples

In the implementation, you can also find:

- Higher-order logic
- Tarksi-style universes, with cumulativity (lifts $\uparrow$)
- (Weak) Coquand-style universes, with cumulativity and universe polymorphism
- Flavous of Observational Type Theory

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

Bidirectional system implemented in a prototype, available at

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

Conclusion

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

Bidirectional system implemented in a prototype, available at

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

Future work

Conclusion

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

Bidirectional system implemented in a prototype, available at

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

Future work

1. Test implementation with real proof libraries, compare with Dedukti

Conclusion

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

Bidirectional system implemented in a prototype, available at

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

Future work

1. Test implementation with real proof libraries, compare with Dedukti
2. Type-directed equalities (η-rules, proof irrelevance), generically?
   Alternatively, treat conversion with a black-box approach

Conclusion

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

Bidirectional system implemented in a prototype, available at

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

Future work

1. Test implementation with real proof libraries, compare with Dedukti
2. Type-directed equalities (η-rules, proof irrelevance), generically?
   Alternatively, treat conversion with a black-box approach
3. More abstract declarative type system (fully-annotated syntax, typed equality, fully-quotiented terms)?
   Generic bidirectional elaboration for a class of SOGATs?

Thank you for your attention!