A Framework for Computational Theories with Minimal Syntax and Bidirectional Typing

Thiago Felicissimo

Europroofnet WG6 Meeting

April 25, 2023

Logical frameworks Frameworks for defining theories

Unify study and implementation of type theories

Logical frameworks Frameworks for defining theories

Unify study and implementation of type theories

Theoretical interest

- One unified notion of theory, of model, etc
- Theorems proven once and for all

Logical frameworks Frameworks for defining theories

Unify study and implementation of type theories

Theoretical interest

- One unified notion of theory, of model, etc
- Theorems proven once and for all

Practical interest

- One unified implementation
- Prototyping new systems (like with rewrite rules in Agda)
- Rechecking proofs (as in Dedukti)

Following Harper, LFs can be classified in two groups

Syntactic LFs

Semantic LFs

Following Harper, LFs can be classified in two groups

Syntactic LFs

- Fixed definitional equality, hence decidable (main example: ELF)

Semantic LFs

Following Harper, LFs can be classified in two groups

Syntactic LFs

- Fixed definitional equality, hence decidable (main example: ELF)
- ~~Terms~~ derivations encoded as terms (judgments as types)

Semantic LFs

Following Harper, LFs can be classified in two groups

Syntactic LFs

- Fixed definitional equality, hence decidable (main example: ELF)
- ~~Terms~~ derivations encoded as terms (judgments as types)

✓ Good for formalizing metatheory (Twelf, Beluga)

Semantic LFs

Following Harper, LFs can be classified in two groups

Syntactic LFs

- Fixed definitional equality, hence decidable (main example: ELF)
- ~~Terms~~ derivations encoded as terms (judgments as types)

✓ Good for formalizing metatheory (Twelf, Beluga)

✗ Typechecker for ~~the theory~~ its derivations

Semantic LFs

Following Harper, LFs can be classified in two groups

Syntactic LFs

- Fixed definitional equality, hence decidable (main example: ELF)
- ~~Terms~~ derivations encoded as terms (judgments as types)

✓ Good for formalizing metatheory (Twelf, Beluga)

✗ Typechecker for ~~the theory~~ its derivations

Semantic LFs

✓ Customizable definitional equality, allows defining theories directly

Following Harper, LFs can be classified in two groups

Syntactic LFs

- Fixed definitional equality, hence decidable (main example: ELF)
- ~~Terms~~ derivations encoded as terms (judgments as types)

✓ Good for formalizing metatheory (Twelf, Beluga)

✗ Typechecker for ~~the theory~~ its derivations

Semantic LFs

✓ Customizable definitional equality, allows defining theories directly

- Growing in interest for semantic methods (e.g. Uemura's LF)

Following Harper, LFs can be classified in two groups

Syntactic LFs

- Fixed definitional equality, hence decidable (main example: ELF)
- ~~Terms~~ derivations encoded as terms (judgments as types)

✓ Good for formalizing metatheory (Twelf, Beluga)

✗ Typechecker for ~~the theory~~ its derivations

Semantic LFs

✓ Customizable definitional equality, allows defining theories directly

- Growing in interest for semantic methods (e.g. Uemura's LF)

✗ Few proposals are "implementable"

Semantic LFs which are implemented:

Andromeda (officially not a LF)

Dedukti

Semantic LFs which are implemented:

Andromeda (officially not a LF)

- Very general definition of type theories

Dedukti

Semantic LFs which are implemented:

Andromeda (officially not a LF)

- Very general definition of type theories
- ✓ Equality checker for computational and extensionality equality rules

Dedukti

Semantic LFs which are implemented:

Andromeda (officially not a LF)

- Very general definition of type theories
- ✓ Equality checker for computational and extensionality equality rules
- ✗ Checker not very fast and not known to be complete (actually, it can't be)

Dedukti

Semantic LFs which are implemented:

Andromeda (officially not a LF)

- Very general definition of type theories
- ✓ Equality checker for computational and extensionality equality rules
- ✗ Checker not very fast and not known to be complete (actually, it can't be)
- ✗ Implements fully annotated syntax: $\lambda x.t \implies \lambda\ A\ (x.B)\ (x.t)$

Dedukti

Semantic LFs which are implemented:

Andromeda (officially not a LF)

- Very general definition of type theories
- ✓ Equality checker for computational and extensionality equality rules
- ✗ Checker not very fast and not known to be complete (actually, it can't be)
- ✗ Implements fully annotated syntax: $\lambda x.t \implies \lambda\ A\ (x.B)\ (x.t)$

Dedukti

- Only computational rules, no support for extensionality rules

Semantic LFs which are implemented:

Andromeda (officially not a LF)

- Very general definition of type theories
- ✓ Equality checker for computational and extensionality equality rules
- ✗ Checker not very fast and not known to be complete (actually, it can't be)
- ✗ Implements fully annotated syntax: $\lambda x.t \implies \lambda\ A\ (x.B)\ (x.t)$

Dedukti

- Only computational rules, no support for extensionality rules
- ✓ Rewriting allows (fast!) theory-agnostic equality checking (experience rechecking big proof libraries confirms this)

Semantic LFs which are implemented:

Andromeda (officially not a LF)

- Very general definition of type theories
- ✓ Equality checker for computational and extensionality equality rules
- ✗ Checker not very fast and not known to be complete (actually, it can't be)
- ✗ Implements fully annotated syntax: $\lambda x.t \implies \lambda\ A\ (x.B)\ (x.t)$

Dedukti

- Only computational rules, no support for extensionality rules
- ✓ Rewriting allows (fast!) theory-agnostic equality checking (experience rechecking big proof libraries confirms this)
- ✗ Also fully annotated syntax: $\lambda x.t \implies \lambda\ A\ (x.B)\ (x.t)$

Semantic LFs which are implemented:

Andromeda (officially not a LF)

- ▪ Very general definition of type theories
- ✓ Equality checker for computational and extensionality equality rules
- ✗ Checker not very fast and not known to be complete (actually, it can't be)
- ✗ Implements fully annotated syntax: $\lambda x.t \implies \lambda\ A\ (x.B)\ (x.t)$

Dedukti

- ▪ Only computational rules, no support for extensionality rules
- ✓ Rewriting allows (fast!) theory-agnostic equality checking (experience rechecking big proof libraries confirms this)
- ✗ Also fully annotated syntax: $\lambda x.t \implies \lambda\ A\ (x.B)\ (x.t)$
- ✗ "Bureaucratic" meaningless terms, not in the image of translation function: $\lambda\ (x.\ @\ t\ x) \quad = \quad \lambda\ (@\ t) \quad = \quad \lambda\ ((z.z)\ @\ t)$

1st Contribution | propose CompLF

1st Contribution I propose CompLF

- A logical framework for computational type theories

1st Contribution I propose CompLF

- A logical framework for computational type theories

✓ Like in Dedukti, fast theory-agnostic equality checking with rewriting

1st Contribution I propose CompLF

- A logical framework for computational type theories

✓ Like in Dedukti, fast theory-agnostic equality checking with rewriting

✓ No bureaucratic terms, only meaningful ones

~~$\lambda((z.z)(@, t))$~~ ~~$\lambda(@(t))$~~ $\lambda(x.@(t, x))$

1st Contribution I propose CompLF

- A logical framework for computational type theories
- ✓ Like in Dedukti, fast theory-agnostic equality checking with rewriting
- ✓ No bureaucratic terms, only meaningful ones

  ~~$\lambda((z.z)(@, t))$~~ ~~$\lambda(@(t))$~~ $\lambda(x.@(t, x))$
- ✓ Supports minimal syntaxes, with *erased arguments*

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

1st Contribution I propose CompLF

- A logical framework for computational type theories

✓ Like in Dedukti, fast theory-agnostic equality checking with rewriting

✓ No bureaucratic terms, only meaningful ones

~~$\lambda((z.z)(@, t))$~~ ~~$\lambda(@(t))$~~ $\lambda(x.@(t, x))$

✓ Supports minimal syntaxes, with *erased arguments* ($\neq$ implicit arguments)

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

1st Contribution I propose CompLF

- A logical framework for computational type theories

✓ Like in Dedukti, fast theory-agnostic equality checking with rewriting

✓ No bureaucratic terms, only meaningful ones

~~$\lambda((z.z)(@, t))$~~ ~~$\lambda(@(t))$~~ $\lambda(x.@(t, x))$

✓ Supports minimal syntaxes, with *erased arguments* ($\neq$ implicit arguments)

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

Problem Minimal syntax jeopardizes decidability of typing

Typing algorithm might need to guess erased arguments

Bidirectional typing algorithms Alternate between two modes

$\boxed{\Gamma \vdash t \Leftarrow T}$ Check (input: Γ, t, T)

$\boxed{\Gamma \vdash t \Rightarrow T}$ Infer (input: Γ, t) (output: T)

Bidirectional typing algorithms Alternate between two modes

$\boxed{\Gamma \vdash t \Leftarrow T}$ Check (input: Γ, t, T)

$\boxed{\Gamma \vdash t \Rightarrow T}$ Infer (input: Γ, t) (output: T)

Allow specify flow of type information in typing rules

$$\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathrm{Tm}\ A \vdash t \Leftarrow \mathrm{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathrm{Tm}\ C}$$

Bidirectional typing algorithms Alternate between two modes

$\boxed{\Gamma \vdash t \Leftarrow T}$ Check (input: Γ, t, T)

$\boxed{\Gamma \vdash t \Rightarrow T}$ Infer (input: Γ, t) (output: T)

Allow specify flow of type information in typing rules

$$\frac{C \longrightarrow^{*} \Pi(A, x.B) \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Leftarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathsf{Tm}\ C}$$

Complement erased arguments very well, explains why they are redundant

Bidirectional typing algorithms Alternate between two modes

$\boxed{\Gamma \vdash t \Leftarrow T}$ Check (input: Γ, t, T)

$\boxed{\Gamma \vdash t \Rightarrow T}$ Infer (input: Γ, t) (output: T)

Allow specify flow of type information in typing rules

$$\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Leftarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathsf{Tm}\ C}$$

Complement erased arguments very well, explains why they are redundant

Previous work Principles of (dependent) bidirectional typing well-known (sometimes called McBride's discipline)

However, no formal account of general case (as far as I know)

Bidirectional typing algorithms Alternate between two modes

$\boxed{\Gamma \vdash t \Leftarrow T}$ Check (input: Γ, t, T)

$\boxed{\Gamma \vdash t \Rightarrow T}$ Infer (input: Γ, t) (output: T)

Allow specify flow of type information in typing rules

$$\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Leftarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathsf{Tm}\ C}$$

Complement erased arguments very well, explains why they are redundant

Previous work Principles of (dependent) bidirectional typing well-known (sometimes called McBride's discipline)

However, no formal account of general case (as far as I know)

LFs can be used for this!

2nd Contribution Theory-agnostic bidirectional typing algorithm

2nd Contribution Theory-agnostic bidirectional typing algorithm

Can be instantiated by given modes to signature in a *mode-correct* way

2nd Contribution Theory-agnostic bidirectional typing algorithm

Can be instantiated by given modes to signature in a *mode-correct* way

✓ Implemented

2nd Contribution Theory-agnostic bidirectional typing algorithm

Can be instantiated by given modes to signature in a *mode-correct* way

- ✓ Implemented
- ✓ Sound (assuming confluence and subject reduction)

2nd Contribution Theory-agnostic bidirectional typing algorithm

Can be instantiated by given modes to signature in a *mode-correct* way

- ✓ Implemented
- ✓ Sound (assuming confluence and subject reduction)
- ✓ Complete for *well-moded terms* (assuming also strong normalisation)

2nd Contribution Theory-agnostic bidirectional typing algorithm

Can be instantiated by given modes to signature in a *mode-correct* way

- ✓ Implemented
- ✓ Sound (assuming confluence and subject reduction)
- ✓ Complete for *well-moded terms* (assuming also strong normalisation)

You, the theory designer, chooses amount of annotations and completeness

2nd Contribution Theory-agnostic bidirectional typing algorithm

Can be instantiated by given modes to signature in a *mode-correct* way

- ✓ Implemented
- ✓ Sound (assuming confluence and subject reduction)
- ✓ Complete for *well-moded terms* (assuming also strong normalisation)

You, the theory designer, chooses amount of annotations and completeness

$$\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Leftarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathsf{Tm}\ C}$$

Well-moded $= \beta$-normal forms

2nd Contribution Theory-agnostic bidirectional typing algorithm

Can be instantiated by given modes to signature in a *mode-correct* way

- ✓ Implemented
- ✓ Sound (assuming confluence and subject reduction)
- ✓ Complete for *well-moded terms* (assuming also strong normalisation)

You, the theory designer, chooses amount of annotations and completeness

$$\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Leftarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathsf{Tm}\ C}$$

Well-moded $= \beta$-normal forms

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

Well-moded $=$ all terms

2nd Contribution Theory-agnostic bidirectional typing algorithm

Can be instantiated by given modes to signature in a *mode-correct* way

- ✓ Implemented
- ✓ Sound (assuming confluence and subject reduction)
- ✓ Complete for *well-moded terms* (assuming also strong normalisation)

You, the theory designer, chooses amount of annotations and completeness

$$\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathrm{Tm}\ A \vdash t \Leftarrow \mathrm{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathrm{Tm}\ C}$$

Well-moded $=$ β-normal forms

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

Well-moded $=$ all terms

For the first time (as far as I know), modular proof of correctness!

CompLF

An excerpt of its definition

Well-scoped & sorted raw syntax

$\boxed{\mathsf{Term}\ \gamma\ s}$ Terms of sort s in scope γ

Well-scoped & sorted raw syntax

$\boxed{\mathsf{Term}\ \gamma\ s}$ Terms of sort s in scope γ

sort = syntactic class

scope $\gamma, \delta ::= () \mid \gamma, x :: \delta \to s$

Well-scoped & sorted raw syntax

$\boxed{\textsf{Term}\ \gamma\ s}$ Terms of sort s in scope γ

sort = syntactic class   scope $\gamma, \delta ::= () \mid \gamma, x :: \delta \to s$

$\boxed{\textsf{Spine}\ \gamma\ \delta}$ Spines of "output" scope δ in scope γ (substitutions)

Well-scoped & sorted raw syntax

$\boxed{\text{Term } \gamma\ s}$ Terms of sort s in scope γ

sort = syntactic class   scope $\gamma, \delta ::= () \mid \gamma, x :: \delta \to s$

$\boxed{\text{Spine } \gamma\ \delta}$ Spines of "output" scope δ in scope γ (substitutions)

$$\frac{\begin{array}{c} h :: \delta \to s \in \varsigma \text{ or } \gamma \\ h = x \text{ or } c \end{array} \quad \mathbf{t} \in \text{Spine } \gamma\ \delta}{h(\mathbf{t}) \in \text{Term } \gamma\ s} \qquad \frac{}{\varepsilon \in \text{Spine } \gamma\ ()} \qquad \frac{\mathbf{t} \in \text{Spine } \gamma\ \delta \qquad t \in \text{Term } \gamma.\gamma'\ s}{\mathbf{t}, \vec{x}_{\gamma'}.t \in \text{Spine } \gamma\ (\delta, x :: \gamma' \to s)}$$

Well-scoped & sorted raw syntax

$\boxed{\mathsf{Term}\ \gamma\ s}$ Terms of sort s in scope γ

sort = syntactic class scope $\gamma, \delta ::= () \mid \gamma, x :: \delta \to s$

$\boxed{\mathsf{Spine}\ \gamma\ \delta}$ Spines of "output" scope δ in scope γ (substitutions)

$$\frac{\begin{array}{c} h :: \delta \to s \in \varsigma \text{ or } \gamma \\ h = x \text{ or } c \end{array} \quad \mathbf{t} \in \mathsf{Spine}\ \gamma\ \delta}{h(\mathbf{t}) \in \mathsf{Term}\ \gamma\ s} \qquad \frac{}{\varepsilon \in \mathsf{Spine}\ \gamma\ ()} \qquad \frac{\mathbf{t} \in \mathsf{Spine}\ \gamma\ \delta \qquad t \in \mathsf{Term}\ \gamma.\gamma'\ s}{\mathbf{t}, \vec{x}_{\gamma'}.t \in \mathsf{Spine}\ \gamma\ (\delta, x :: \gamma' \to s)}$$

Example By taking pre-signature

$$\varsigma_\lambda = \quad \lambda :: (\mathtt{t} :: (\mathtt{x} :: \mathsf{tm}) \to \mathsf{tm}) \to \mathsf{tm} \qquad \mathtt{@} :: (\mathtt{t} :: \mathsf{tm})(\mathtt{u} :: \mathsf{tm}) \to \mathsf{tm}$$

$\mathsf{Term}\ (\vec{x} :: \vec{\mathsf{tm}})\ \mathsf{tm}$ = λ-terms with free variables in $\vec{x}$

Well-scoped & sorted raw syntax

$\boxed{\mathsf{Term}\ \gamma\ s}$ Terms of sort s in scope γ

sort = syntactic class $\qquad$ scope $\gamma, \delta ::= () \mid \gamma, x :: \delta \to s$

$\boxed{\mathsf{Spine}\ \gamma\ \delta}$ Spines of "output" scope δ in scope γ (substitutions)

$$\begin{array}{c} h :: \delta \to s \in \varsigma \text{ or } \gamma \\ h = x \text{ or } c \end{array} \quad \frac{\mathbf{t} \in \mathsf{Spine}\ \gamma\ \delta}{h(\mathbf{t}) \in \mathsf{Term}\ \gamma\ s} \qquad \frac{}{\varepsilon \in \mathsf{Spine}\ \gamma\ ()} \qquad \frac{\mathbf{t} \in \mathsf{Spine}\ \gamma\ \delta \qquad t \in \mathsf{Term}\ \gamma.\gamma'\ s}{\mathbf{t}, \vec{x}_{\gamma'}.t \in \mathsf{Spine}\ \gamma\ (\delta, x :: \gamma' \to s)}$$

Example By taking pre-signature

$$\varsigma_\lambda = \quad \lambda :: (\mathtt{t} :: (\mathtt{x} :: \mathsf{tm}) \to \mathsf{tm}) \to \mathsf{tm} \qquad \mathtt{@} :: (\mathtt{t} :: \mathsf{tm})(\mathtt{u} :: \mathsf{tm}) \to \mathsf{tm}$$

$\mathsf{Term}\ (\vec{x} :: \vec{\mathsf{tm}})\ \mathsf{tm}$ = λ-terms with free variables in $\vec{x}$

✓ Accurate account of raw syntax

Signatures Description of typing rules

Signatures Description of typing rules

Example Dependently-typed λ-calculus:

$$
\begin{aligned}
\mathsf{Ty} :\ & \Box \\
\mathsf{Tm} :\ & (\mathtt{A} : \mathsf{Ty}) \to \Box \\
\Pi :\ & (\mathtt{A} : \mathsf{Ty})(\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}) \to \mathsf{Ty} \\
\lambda :\ & \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
& (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x)) \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) \\
@ :\ & \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
& (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))(\mathtt{u} : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u})
\end{aligned}
$$

Dependency erasure map as a design principle to glue the layers

Type layer	types	contexts	signatures
	$\downarrow \lvert - \rvert$	$\downarrow \lvert - \rvert$	$\downarrow \lvert - \rvert$
Syntax layer	sorts	scopes	pre-signatures

Signatures Description of typing rules

Example Dependently-typed λ-calculus:

$$\begin{aligned}
\mathsf{Ty} &: \square & &\xrightarrow{|-|} & \mathsf{Ty} &:: \square \\
\mathsf{Tm} &: (\mathtt{A} : \mathsf{Ty}) \to \square & &\xrightarrow{|-|} & \mathsf{Tm} &:: (\mathtt{A} :: \mathsf{ty}) \to \square \\
\Pi &: (\mathtt{A} : \mathsf{Ty})(\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}) \to \mathsf{Ty} & &\xrightarrow{|-|} & \Pi &:: (\mathtt{A} :: \mathsf{ty})(\mathtt{B} :: (x :: \mathsf{tm}) \to \mathsf{ty}) \to \mathsf{ty} \\
\lambda &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} & &\xrightarrow{|-|} & \lambda &:: (\mathtt{t} :: (x :: \mathsf{tm}) \to \mathsf{tm}) \to \mathsf{tm} \\
&\quad (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x)) \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) & & & & \\
@ &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} & &\xrightarrow{|-|} & @ &:: (\mathtt{t} :: \mathsf{tm})(\mathtt{u} :: \mathsf{tm}) \to \mathsf{tm} \\
&\quad (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))(\mathtt{u} : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u}) & & & &
\end{aligned}$$

Signatures Description of typing rules

Example Dependently-typed λ-calculus:

$$
\begin{array}{rlcrl}
\mathsf{Ty} : & \square & \xrightarrow{|-|} & \mathsf{Ty} :: & \square \\
\mathsf{Tm} : & (\mathtt{A} : \mathsf{Ty}) \to \square & \xrightarrow{|-|} & \mathsf{Tm} :: & (\mathtt{A} :: \mathsf{ty}) \to \square \\
\Pi : & (\mathtt{A} : \mathsf{Ty})(\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}) \to \mathsf{Ty} & \xrightarrow{|-|} & \Pi :: & (\mathtt{A} :: \mathsf{ty})(\mathtt{B} :: (x :: \mathsf{tm}) \to \mathsf{ty}) \to \mathsf{ty} \\
\lambda : & \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} & \xrightarrow{|-|} & \lambda :: & (\mathtt{t} :: (x :: \mathsf{tm}) \to \mathsf{tm}) \to \mathsf{tm} \\
 & (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x)) \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) & & & \\
@ : & \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} & \xrightarrow{|-|} & @ :: & (\mathtt{t} :: \mathsf{tm})(\mathtt{u} :: \mathsf{tm}) \to \mathsf{tm} \\
 & (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))(\mathtt{u} : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u}) & & &
\end{array}
$$

Erased arguments Marked with $\{-\}$, removed from the syntax

Typing rules With

$$\lambda : \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\}(\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x)) \to \mathsf{Tm}\ \Pi(\mathtt{A}, \mathtt{B}(x)) \in \Sigma$$

we have

Typing rules With

$$\lambda : \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\}(\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x)) \to \mathsf{Tm}\ \Pi(\mathtt{A}, \mathtt{B}(x)) \in \Sigma$$

we have

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

Typing rules With

$$\lambda : \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\}(\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x)) \to \mathsf{Tm}\ \Pi(\mathtt{A}, \mathtt{B}(x)) \in \Sigma$$

we have

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

The leaves give the expected typing rule for λ

Typing rules With

$$\lambda : \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\}(\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x)) \to \mathsf{Tm}\ \Pi(\mathtt{A}, \mathtt{B}(x)) \in \Sigma$$

we have

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

The leaves give the expected typing rule for λ

✓ Accurate account of typing rules

Moded signatures Refine signatures with modes

Moded signatures Refine signatures with modes

$+ = \text{infer} \qquad - = \text{check}$

$$
\begin{aligned}
\mathsf{Ty} &: \Box \\
\mathsf{Tm} &: (\mathtt{A} : \mathsf{Ty})^- \to \Box \\
\Pi^+ &: (\mathtt{A} : \mathsf{Ty})^-(\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty})^- \to \mathsf{Ty} \\
\lambda^- &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x))^- \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) \\
@^+ &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))^+(\mathtt{u} : \mathsf{Tm}\ \mathtt{A})^- \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u})
\end{aligned}
$$

Moded signatures Refine signatures with modes

$+ =$ infer $\qquad - =$ check

$$
\begin{aligned}
\mathsf{Ty} &: \square \\
\mathsf{Tm} &: (\mathtt{A} : \mathsf{Ty})^- \to \square \\
\Pi^+ &: (\mathtt{A} : \mathsf{Ty})^- (\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty})^- \to \mathsf{Ty} \\
\lambda^- &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x))^- \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) \\
@^+ &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))^+ (\mathtt{u} : \mathsf{Tm}\ \mathtt{A})^- \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u})
\end{aligned}
$$

$$
\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Leftarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathsf{Tm}\ C}
$$

Moded signatures Refine signatures with modes

$+ =$ infer $\qquad - =$ check

$$
\begin{aligned}
\mathsf{Ty} &: \square \\
\mathsf{Tm} &: (\mathtt{A} : \mathsf{Ty})^- \to \square \\
\Pi^+ &: (\mathtt{A} : \mathsf{Ty})^- (\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty})^- \to \mathsf{Ty} \\
\lambda^- &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x))^- \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) \\
@^+ &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))^+ (\mathtt{u} : \mathsf{Tm}\ \mathtt{A})^- \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Ty} &: \square \\
\mathsf{Tm} &: (\mathtt{A} : \mathsf{Ty})^- \to \square \\
\Pi^+ &: (\mathtt{A} : \mathsf{Ty})^- (\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty})^- \to \mathsf{Ty} \\
\lambda^+ &: (\mathtt{A} : \mathsf{Ty})^- \{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x))^+ \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) \\
@^+ &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))^+ (\mathtt{u} : \mathsf{Tm}\ \mathtt{A})^- \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u})
\end{aligned}
$$

$$
\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Leftarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathsf{Tm}\ C}
$$

Moded signatures Refine signatures with modes

$+ =$ infer $\qquad - =$ check

$$
\begin{aligned}
\mathsf{Ty} &: \Box \\
\mathsf{Tm} &: (\mathtt{A} : \mathsf{Ty})^- \to \Box \\
\Pi^+ &: (\mathtt{A} : \mathsf{Ty})^- (\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty})^- \to \mathsf{Ty} \\
\lambda^- &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x))^- \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) \\
@^+ &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))^+ (\mathtt{u} : \mathsf{Tm}\ \mathtt{A})^- \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u})
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Ty} &: \Box \\
\mathsf{Tm} &: (\mathtt{A} : \mathsf{Ty})^- \to \Box \\
\Pi^+ &: (\mathtt{A} : \mathsf{Ty})^- (\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty})^- \to \mathsf{Ty} \\
\lambda^+ &: (\mathtt{A} : \mathsf{Ty})^- \{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Tm}\ \mathtt{B}(x))^+ \to \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)) \\
@^+ &: \{\mathtt{A} : \mathsf{Ty}\}\{\mathtt{B} : (x : \mathsf{Tm}\ \mathtt{A}) \to \mathsf{Ty}\} \\
&\quad (\mathtt{t} : \mathsf{Tm}\ \Pi(\mathtt{A}, x.\mathtt{B}(x)))^+ (\mathtt{u} : \mathsf{Tm}\ \mathtt{A})^- \to \mathsf{Tm}\ \mathtt{B}(\mathtt{u})
\end{aligned}
$$

$$
\frac{C \longrightarrow^* \Pi(A, x.B) \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Leftarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(x.t) \Leftarrow \mathsf{Tm}\ C}
\qquad
\frac{\Gamma \vdash A \Leftarrow \mathsf{Ty} \qquad \Gamma, x : \mathsf{Tm}\ A \vdash t \Rightarrow \mathsf{Tm}\ B}{\Gamma \vdash \lambda(A, x.t) \Rightarrow \mathsf{Tm}\ \Pi(A, x.B)}
$$

```
(* Judgment forms *)
symbol Ty : *

symbol Tm (A : Ty)- : *


(* Dependent products (lambda not annotated) *)
symbol+ Π (A : Ty)- (B : (x : Tm A) Ty)- : Ty

symbol- λ {A : Ty} {B : (_ : Tm A) Ty} (t : (x : Tm A) Tm B(x))- : Tm Π(A, x. B(x))

symbol+ @ {A : Ty} {B : (_ : Tm A) Ty} (t : Tm Π(A, x. B(x)))+ (u : Tm A)- : Tm B(u)

rew @(λ(x. $t(x)), $u) --> $t($u)


symbol+ T : Ty (* Auxiliary base type *)

(* Example *)
let church1 : Tm Π(Π(T, _. T), _. Π(T, _. T)) := λ(f. λ(x. @(f, x)))
```

```
(* Gives error *)
(* let redex : Tm Π(T, _. T) := λ(x. @(λ(y.y), x)) *)

(* Dependent products (lambda annotated) *)
symbol+ Π' (A : Ty)- (B : (x : Tm A) Ty)- : Ty

symbol+ @' {A : Ty} {B : (_ : Tm A) Ty} (t : Tm Π'(A, x. B(x)))+ (u : Tm A)- : Tm B(u)

symbol+ λ' (A : Ty)- {B : (_ : Tm A) Ty} (t : (x : Tm A) Tm B(x))+ : Tm Π'(A, x. B(x))

rew @'(λ'($T, x. $t(x)), $u) --> $t($u)

(* Now it works! *)
type λ'(T, x. @'(λ'(T, y.y), x))
```

```
1.7k  complf/test/wg6.complf  15:0 21%                                    Fundamental (+4)
[type] λ'(T, x0. @'(λ'(T, x1. x1), x0)) : Tm(Π'(T, x0. T))
thiago@thiago-work:~/git/complf$
```

Beyond dependent products

```
(* Universe *)
symbol+ U : Ty
symbol+ El (A : Tm U)- : Ty

(* Equality type *)
symbol+ Eq (A : Ty)- (t : Tm A)- (u : Tm A)- : Ty

symbol- refl {A : Ty} {t : Tm A} : Tm Eq(A, t, t)

symbol+ J {A : Ty} {a : Tm A} {b : Tm A} (t : Tm Eq(A, a, b))+
       (P : (x : Tm A, y : Tm Eq(A, a, x)) Ty)- (prefl : Tm P(a, refl))- : Tm P(b, t)

rew J(refl, x y. $P(x, y), $prefl) --> $prefl

(* Code in U for Eq *)
symbol+ eq (a : Tm U)- (x : Tm El(a))- (y : Tm El(a))- : Tm U

rew El(eq($a, $x, $y)) --> Eq(El($a), $x, $y)

(* 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 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)))))
```

Beyond dependent products

```
(* Universe *)
symbol+ U : Ty
symbol+ El (A : Tm U)- : Ty

(* Equality type *)
symbol+ Eq (A : Ty)- (t : Tm A)- (u : Tm A)- : Ty

symbol- refl {A : Ty} {t : Tm A} : Tm Eq(A, t, t)

symbol+ J {A : Ty} {a : Tm A} {b : Tm A} (t : Tm Eq(A, a, b))+
       (P : (x : Tm A, y : Tm Eq(A, a, x)) Ty)- (prefl : Tm P(a, refl))- : Tm P(b, t)

rew J(refl, x y. $P(x, y), $prefl) --> $prefl

(* Code in U for Eq *)
symbol+ eq (a : Tm U)- (x : Tm El(a))- (y : Tm El(a))- : Tm U

rew El(eq($a, $x, $y)) --> Eq(El($a), $x, $y)

(* 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 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)))))
```

But also other types (Σ, List, Nat,...), Coquand-style universes, universe polymorphism, pure type systems, higher-order logic, etc

Conclusion

CompLF Logical framework for computational type theories

Faithful presentation of syntax, erased arguments

Conclusion

CompLF Logical framework for computational type theories

Faithful presentation of syntax, erased arguments

Customisable bidirectional typing algorithm

Try it at `https://github.com/thiagofelicissimo/complf`

Conclusion

CompLF Logical framework for computational type theories

Faithful presentation of syntax, erased arguments

Customisable bidirectional typing algorithm

Try it at `https://github.com/thiagofelicissimo/complf`

Thank you for your attention!