What are we thinking when we present a type theory?

Peter LeFanu Lumsdaine
(joint work with Bauer, Haselwarter)

Stockholm University

HoTTEST, June 2020

Video: https://youtu.be/kQe0knDuZqg

Some familiar rules

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma, x{:}A \vdash B \text{ type}}{\Gamma \vdash \Pi(x{:}A)B \text{ type}}\ \Pi$$

$$\frac{\begin{array}{c}\Gamma \vdash A \text{ type} \qquad \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma, x{:}A \vdash b : B\end{array}}{\Gamma \vdash \lambda x{:}A.\ b \text{ type}}\ \lambda$$

$$\frac{\begin{array}{c}\Gamma \vdash A \text{ type} \qquad \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \qquad \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{app}_{x:A,\ B}(f, a) : B[a/x]}\ \textsc{app}$$

$$\frac{\begin{array}{c}\Gamma \vdash A \text{ type} \qquad \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma, x{:}A \vdash b : B \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{app}_{x:A,\ B}((\lambda x{:}A.\ b), a) \equiv b[a/x] : B[a/x]}\ \beta$$

Some unfamiliar rules

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \ \mathsf{type}}{\Gamma \vdash \mathsf{T}(x.P) \ \mathsf{type}}$$

Some unfamiliar rules

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \ \mathsf{type}}{\Gamma \vdash \mathsf{T}(x.P) \ \mathsf{type}}$$

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \ \mathsf{type}}{\Gamma \vdash \mathsf{t}_{x.P} : P[\mathsf{refl}(0)/x]}$$

Some unfamiliar rules

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \ \mathsf{type}}{\Gamma \vdash \mathsf{T}(x.P) \ \mathsf{type}}$$

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \ \mathsf{type}}{\Gamma \vdash \mathsf{t}_{x.P} : P[\mathsf{refl}(0)/x]}$$

$$\frac{\Gamma \vdash a : Q[0/x, \mathsf{refl}(0)/y]}{\Gamma \vdash \mathsf{T}_{x.Q,a} \ \mathsf{type}}$$

Some unfamiliar rules

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \;\mathsf{type}}{\Gamma \vdash \mathsf{T}(x.P) \;\mathsf{type}}$$

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \;\mathsf{type}}{\Gamma \vdash \mathsf{t}_{x.P} : P[\mathsf{refl}(0)/x]}$$

$$\frac{\Gamma \vdash a : Q[0/x, \mathsf{refl}(0)/y]}{\Gamma \vdash \mathsf{T}_{x.Q,a} \;\mathsf{type}}$$

$$\frac{\Gamma \vdash a : \mathsf{Bool} \qquad \Gamma, x{:}\mathsf{N} \vdash a : \mathsf{Id}_{\mathsf{N}}(x, x)}{\Gamma \vdash T(a) \;\mathsf{type}}$$

Some unfamiliar rules

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \ \mathsf{type}}{\Gamma \vdash \mathsf{T}(x.P) \ \mathsf{type}}$$

$$\frac{\Gamma, x{:}\mathsf{N} \vdash P \ \mathsf{type}}{\Gamma \vdash \mathsf{t}_{x.P} : P[\mathsf{refl}(0)/x]}$$

$$\frac{\Gamma \vdash a : Q[0/x, \mathsf{refl}(0)/y]}{\Gamma \vdash \mathsf{T}_{x.Q,a} \ \mathsf{type}}$$

$$\frac{\Gamma \vdash a : \mathsf{Bool} \qquad \Gamma, x{:}\mathsf{N} \vdash a : \mathsf{Id}_{\mathsf{N}}(x, x)}{\Gamma \vdash T(a) \ \mathsf{type}}$$

Question

What criteria make us accept some of these, reject others?

Basic setup

- Background setup: raw syntax, raw rules, raw type theories, derivability of judgements …
- Desirable properties of rules
- Well-ordered presentations
- Semantics

Basic setup

- Background setup: raw syntax, raw rules, raw type theories, derivability of judgements …
- Desirable properties of rules
- Well-ordered presentations
- Semantics

Goals:

- articulate what we have implicitly in mind when writing/reading type theories;
- formalise the idea “A type theory is a well-ordered family of rules, each well-formed over the type theory given by the earlier rules.”
- show this suffices to give good behaiour, algebraic semantics.

Signatures

Definition

- Syntactic classes: Ty, Tm.
- Arity: a list of pairs of syntactic class and number.
- Signature:
  - a set Σ of symbols;
  - a function $a : \Sigma \longrightarrow \mathsf{Class} \times \mathsf{Arity}$, the (input) arity and (output) class of each symbol.

Tersely: signature is a family of pairs of a class and arity.

Idea: arity gives, for each argument of a symbol, the class and number of bindings. E.g. signature for Π-types:

$$\begin{array}{rcl} \Pi & \mathsf{Ty} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1)] \\ \lambda & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 1)] \\ \mathsf{app} & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 0), (\mathsf{Tm}, 0)] \end{array}$$

Signatures

Definition

- Syntactic classes: Ty, Tm.
- Arity: a list of pairs of syntactic class and number.
- Signature:
  - a set Σ of symbols;
  - a function $a : \Sigma \longrightarrow \mathrm{Class} \times \mathrm{Arity}$, the (input) arity and (output) class of each symbol.

  Tersely: signature is a family of pairs of a class and arity.

Idea: arity gives, for each argument of a symbol, the class and number of bindings. E.g. signature for Π-types:

$$\begin{array}{rll} \Pi & \mathsf{Ty} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1)] \\ \lambda & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 1)] \\ \mathsf{app} & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 0), (\mathsf{Tm}, 0)] \end{array}$$

Signatures

Definition

- Syntactic classes: Ty, Tm.
- Arity: a list of pairs of syntactic class and number.
- Signature:
  - a set Σ of symbols;
  - a function $a : \Sigma \longrightarrow \text{Class} \times \text{Arity}$, the (input) arity and (output) class of each symbol.

Tersely: signature is a family of pairs of a class and arity.

Idea: arity gives, for each argument of a symbol, the class and number of bindings. E.g. signature for Π-types:

$$\begin{array}{rll} \Pi & \mathsf{Ty} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1)] \\ \lambda & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 1)] \\ \mathsf{app} & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 0), (\mathsf{Tm}, 0)] \end{array}$$

Signatures

Definition

- Syntactic classes: Ty, Tm.
- Arity: a list of pairs of syntactic class and number.
- Signature:
  - a set Σ of symbols;
  - a function $a : \Sigma \longrightarrow \text{Class} \times \text{Arity}$, the (input) arity and (output) class of each symbol.

Tersely: signature is a family of pairs of a class and arity.

Idea: arity gives, for each argument of a symbol, the class and number of bindings. E.g. signature for Π-types:

$$
\begin{array}{rcl}
\Pi & \mathsf{Ty} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1)] \\
\lambda & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 1)] \\
\mathsf{app} & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 0), (\mathsf{Tm}, 0)]
\end{array}
$$

Signatures

Definition

- Syntactic classes: Ty, Tm.
- Arity: a list of pairs of syntactic class and number.
- Signature:
  - a set Σ of symbols;
  - a function $a : \Sigma \longrightarrow \mathsf{Class} \times \mathsf{Arity}$, the (input) arity and (output) class of each symbol.

  Tersely: signature is a family of pairs of a class and arity.

Idea: arity gives, for each argument of a symbol, the class and number of bindings. E.g. signature for Π-types:

$$\begin{array}{rcl} \Pi & \mathsf{Ty} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1)] \\ \lambda & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 1)] \\ \mathsf{app} & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 0), (\mathsf{Tm}, 0)] \end{array}$$

Signatures

Definition

- Syntactic classes: Ty, Tm.
- Arity: a list of pairs of syntactic class and number.
- Signature:
  - a set Σ of symbols;
  - a function $a : \Sigma \longrightarrow \mathsf{Class} \times \mathsf{Arity}$, the (input) arity and (output) class of each symbol.

  Tersely: signature is a family of pairs of a class and arity.

Idea: arity gives, for each argument of a symbol, the class and number of bindings. E.g. signature for Π-types:

$$\begin{array}{rcl} \Pi & \mathsf{Ty} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1)] \\ \lambda & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 1)] \\ \mathsf{app} & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 0), (\mathsf{Tm}, 0)] \end{array}$$

Signatures

Definition

- Syntactic classes: Ty, Tm.
- Arity: a list of pairs of syntactic class and number.
- Signature:
  - a set Σ of symbols;
  - a function $a : \Sigma \longrightarrow \mathsf{Class} \times \mathsf{Arity}$, the (input) arity and (output) class of each symbol.

  Tersely: signature is a family of pairs of a class and arity.

Idea: arity gives, for each argument of a symbol, the class and number of bindings. E.g. signature for Π-types:

$$\begin{array}{rll} \Pi & \mathsf{Ty} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1)] \\ \lambda & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 1)] \\ \mathsf{app} & \mathsf{Tm} & [(\mathsf{Ty}, 0), (\mathsf{Ty}, 1), (\mathsf{Tm}, 0), (\mathsf{Tm}, 0)] \end{array}$$

Expressions, judgements

Definition

Over a signature Σ, define:

- Raw (scoped) expressions $\mathrm{Expr}^{\mathsf{Ty}}_{\Sigma}(n)$, $\mathrm{Expr}^{\mathsf{Tm}}_{\Sigma}(n)$: sets of raw type/term expressions in n variables
- Raw contexts Γ: suitable lists of raw type epxressions
- Judgement forms, judgements: suitable lists/tuples of expressions

$$\begin{array}{rl} \mathsf{Ty} & \Gamma \vdash A \ \mathsf{type} \\ \mathsf{Tm} & \Gamma \vdash a : A \\ \mathsf{TyEq} & \Gamma \vdash A \equiv B \\ \mathsf{TmEq} & \Gamma \vdash a \equiv b : A \end{array}$$

Expressions, judgements

Definition

Over a signature Σ, define:

- Raw (scoped) expressions $\mathrm{Expr}_\Sigma^{\mathsf{Ty}}(n)$, $\mathrm{Expr}_\Sigma^{\mathsf{Tm}}(n)$: sets of raw type/term expressions in n variables
- Raw contexts Γ: suitable lists of raw type epxressions
- Judgement forms, judgements: suitable lists/tuples of expressions

$$\left.\begin{array}{ll} \mathsf{Ty} & \Gamma \vdash A \ \mathsf{type} \\ \mathsf{Tm} & \Gamma \vdash a : A \end{array}\right\}\text{object judgements}$$

$$\left.\begin{array}{ll} \mathsf{TyEq} & \Gamma \vdash A \equiv B \\ \mathsf{TmEq} & \Gamma \vdash a \equiv b : A \end{array}\right\}\text{equality judgements}$$

Raw rules

What do we mean when we write down a rule?

- A list of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability

$$\frac{\begin{array}{c}\Gamma \vdash A \text{ type} \\ \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathrm{app}(A, B, f, a) : B[a/x]} \quad \rightsquigarrow \quad \begin{array}{c}\text{for all raw } \Gamma, A, B, f, a, \text{ if} \\ \Gamma \vdash A \text{ type} \\ \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A \\ \text{are all derivable, then} \\ \Gamma \vdash \mathrm{app}(A, B, f, a) : B[a/x] \\ \text{is derivable}\end{array}$$

Raw rules

What do we mean when we write down a rule?

- A ~~list~~ family of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability

$$\frac{\begin{array}{c}\Gamma \vdash A \text{ type} \\ \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x]} \quad \rightsquigarrow \quad \begin{array}{c}\text{for all raw } \Gamma, A, B, f, a, \text{ if} \\ \Gamma \vdash A \text{ type} \\ \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A \\ \text{are all derivable, then} \\ \Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x] \\ \text{is derivable}\end{array}$$

Raw rules

What do we mean when we write down a rule?

- A ~~list~~ family of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability

$$\frac{\begin{array}{c}\Gamma \vdash A \text{ type} \\ \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x]} \quad \rightsquigarrow \quad \begin{array}{c}\text{for all raw } \Gamma, A, B, f, a, \text{ if} \\ \Gamma \vdash A \text{ type} \\ \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A \\ \text{are all derivable, then} \\ \Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x] \\ \text{is derivable}\end{array}$$

Raw rules

What do we mean when we write down a rule?

- A ~~list~~ family of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability
- All rules hold over arbitrary ambient contexts. So: context not needed in rule specification!

$$\frac{\begin{array}{c}\Gamma \vdash A \text{ type} \\ \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathrm{app}(A, B, f, a) : B[a/x]} \quad \rightsquigarrow \quad \begin{array}{c}\text{for all raw } \Gamma, A, B, f, a, \text{ if} \\ \Gamma \vdash A \text{ type} \\ \Gamma, x{:}A \vdash B \text{ type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A \\ \text{are all derivable, then} \\ \Gamma \vdash \mathrm{app}(A, B, f, a) : B[a/x] \\ \text{is derivable}\end{array}$$

Raw rules

What do we mean when we write down a rule?

- A ~~list~~ family of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability
- All rules hold over arbitrary ambient contexts. So: context not needed in rule specification!

$$\frac{\vdash A \text{ type} \qquad x{:}A \vdash B \text{ type} \qquad \vdash f : \Pi(x{:}A)B \qquad \vdash a : A}{\vdash \mathsf{app}(A, B, f, a) : B[a/x]} \quad \rightsquigarrow$$

for all raw Γ, A, B, f, a, if

$\Gamma \vdash A$ type

$\Gamma, x{:}A \vdash B$ type

$\Gamma \vdash f : \Pi(x{:}A)B$

$\Gamma \vdash a : A$

are all derivable, then

$\Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x]$

is derivable

Raw rules

What do we mean when we write down a rule?

- A ~~list~~ family of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability
- All rules hold over arbitrary ambient contexts. So: context not needed in rule specification!
- Treatment of metavariables? Add symbols to signature.

$$\frac{\vdash A\ \mathsf{type} \quad x{:}A \vdash B\ \mathsf{type} \quad \vdash f : \Pi(x{:}A)B \quad \vdash a : A}{\vdash \mathsf{app}(A, B, f, a) : B[a/x]} \quad \rightsquigarrow$$

for all raw Γ, A, B, f, a, if

$\Gamma \vdash A\ \mathsf{type}$

$\Gamma, x{:}A \vdash B\ \mathsf{type}$

$\Gamma \vdash f : \Pi(x{:}A)B$

$\Gamma \vdash a : A$

are all derivable, then

$\Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x]$

is derivable

Raw rules

What do we mean when we write down a rule?

- A ~~list~~ family of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability
- All rules hold over arbitrary ambient contexts. So: context not needed in rule specification!
- Treatment of metavariables? Add symbols to signature.

$$\frac{\begin{array}{c}\vdash \mathsf{A}\ \mathsf{type} \\ x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \\ \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(x)) \\ \vdash \mathsf{a} : \mathsf{A}\end{array}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})} \quad \rightsquigarrow \quad \begin{array}{c}\text{for all raw } \Gamma, A, B, f, a, \text{ if} \\ \Gamma \vdash A\ \mathsf{type} \\ \Gamma, x{:}A \vdash B\ \mathsf{type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A \\ \text{are all derivable, then} \\ \Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x] \\ \text{is derivable}\end{array}$$

Raw rules

What do we mean when we write down a rule?

- A ~~list~~ family of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability
- All rules hold over arbitrary ambient contexts. So: context not needed in rule specification!
- Treatment of metavariables? Add symbols to signature.
- Substitution in metavariables? Arguments to symbols; instantiate as actual substitution.

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \qquad \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(x)) \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})} \quad \rightsquigarrow$$

for all raw Γ, A, B, f, a, if

$$\Gamma \vdash A\ \mathsf{type}$$
$$\Gamma, x{:}A \vdash B\ \mathsf{type}$$
$$\Gamma \vdash f : \Pi(x{:}A)B$$
$$\Gamma \vdash a : A$$

are all derivable, then

$$\Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x]$$

is derivable

Raw rules

What do we mean when we write down a rule?

- A ~~list~~ family of judgements (premises) and a judgement (conclusion), interpreted as closure condition on derivability
- All rules hold over arbitrary ambient contexts. So: context not needed in rule specification!
- Treatment of metavariables? Add symbols to signature.
- Substitution in metavariables? Arguments to symbols; instantiate as actual substitution.

$$
\dfrac{\begin{array}{c} \vdash \mathsf{A}\ \mathsf{type} \\ x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \\ \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(x)) \\ \vdash \mathsf{a} : \mathsf{A} \end{array}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}
\quad \rightsquigarrow \quad
\begin{array}{c} \text{for all raw } \Gamma, A, B, f, a, \text{ if} \\ \Gamma \vdash A\ \mathsf{type} \\ \Gamma, x{:}A \vdash B\ \mathsf{type} \\ \Gamma \vdash f : \Pi(x{:}A)B \\ \Gamma \vdash a : A \\ \text{are all derivable, then} \\ \Gamma \vdash \mathsf{app}(A, B, f, a) : B[a/x] \\ \text{is derivable} \end{array}
$$

Raw rules

Definition

- Metavariable extension $\Sigma + a$ (a an arity): signature extending Σ by symbols for the arguments of a.
- Raw rule over Σ of arity a: family of judgements (premises) and one more judgement (conclusion), all over $\Sigma + a$.
- Instantiation of a over Σ: a raw context Γ and suitable expressions according to a, specfying mapping from syntax of $\Sigma + a$ to syntax of Σ.

Raw rules

Definition

- Metavariable extension $\Sigma + a$ (a an arity): signature extending Σ by symbols for the arguments of a.
- Raw rule over Σ of arity a: family of judgements (premises) and one more judgement (conclusion), all over $\Sigma + a$.
- Instantiation of a over Σ: a raw context Γ and suitable expressions according to a, specfying mapping from syntax of $\Sigma + a$ to syntax of Σ.
- Raw type theory over Σ: family of raw rules over Σ.
- Derivability over raw type theory T: relation on judgements, inductively defined by closure conditions for
  - standard structural rules;
  - all instantiations of all raw rules in T.

Summary so far

Have defined:

- signatures, raw syntax, judgements;
- raw rules;
- raw type theories, derivability.

A satisfactory account of what these are usually understood to mean.

However: too general. Need to add more requirements to ensure:

- well-behavedness as a formal system (metatheorems);
- intuitive comprehensibility;
- can assign good semantics.

Presuppositions, boundaries

Definition

Any judgement has a family of presuppositions:

- $\Gamma \vdash A$ type has no presuppositions;
- only presuppositions of $\Gamma \vdash a : A$ is $\Gamma \vdash A$ type;
- presuppositions of $\Gamma \vdash A \equiv A'$ are $\Gamma \vdash A$ type, $\Gamma \vdash A'$ type;
- presuppositions of $\Gamma \vdash a \equiv a' : A$ are $\Gamma \vdash A$ type, $\Gamma \vdash a : A$, $\Gamma \vdash a' : A$.

A judgement boundary is like a judgement, but missing the head expression (if any):

$$\Gamma \vdash \_ \text{ type} \qquad \Gamma \vdash \_ : A \qquad \Gamma \vdash A \stackrel{?}{\equiv} A' \qquad \Gamma \vdash a \stackrel{?}{\equiv} a' : A$$

Boundary holds same data as presuppositions, but seen as a single configuration, not just a family of judgements.

Compare: the faces and boundary of a simplex.

Presupposivity

Definition

- A raw rule is (derivably) presuppositive over T if all presuppositions of its premises and conclusion are derivable from its premises, over T .
- A raw rule is admissibly presuppositive over T if whenever its premises are derivable, so are all presuppositions of its premises and conclusion.

- Admissible presuppositivity: never(?) violated in practice.
- Derivable presuppositivity: sometimes violated. May need to close premises under presuppositions, inversion principles, etc.
- WLONG[1] all rules can be assumed (derivably) presuppositive.

Proposition

If all rules of T *are presuppositive, then whenever a judgement is derivable over* T*, so are all its presuppositions.*

[1] without loss of natural generality

Tightness

Definition

A raw rule is tight if its metavariables correspond bijectively to its object-judgement premises,
each premise introducing the corresponding metavariable in general form.

Tightness

Definition

A raw rule is tight if its metavariables correspond bijectively to its object-judgement premises,
each premise introducing the corresponding metavariable in general form.

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}}{\vdash \Pi(\mathsf{A}, \mathsf{B}(x))\ \mathsf{type}} \qquad \frac{x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}}{\vdash \Pi(\mathsf{A}, \mathsf{B}(x))\ \mathsf{type}}$$

$$\frac{\vdash \mathsf{a} : \mathsf{Bool} \qquad x{:}\mathsf{N} \vdash \mathsf{a} : \mathsf{Id}_{\mathsf{N}}(x, x)}{\vdash \mathsf{T}(\mathsf{a})\ \mathsf{type}}$$

Tightness

Definition

A raw rule is tight if its metavariables correspond bijectively to its object-judgement premises,
each premise introducing the corresponding metavariable in general form.

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}}{\vdash \Pi(\mathsf{A}, \mathsf{B}(x))\ \mathsf{type}} \qquad\qquad \frac{x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}}{\vdash \Pi(\mathsf{A}, \mathsf{B}(x))\ \mathsf{type}}$$

$$\frac{\vdash \mathsf{a} : \mathsf{Bool} \qquad x{:}\mathsf{N} \vdash \mathsf{a} : \mathsf{Id}_{\mathsf{N}}(x, x)}{\vdash \mathsf{T}(\mathsf{a})\ \mathsf{type}}$$

Violated frequently, but within strict limits: “missing premises” can always(?) be inferred via presuppositions, inversion principles, etc.

WLONG, all(?) natural examples are equivalent to tight rules.

Tightness of theories

Definition

A raw type theory is tight if all its rules are tight, and its object-judgement rules correspond precisely to symbols of its signature.

Proposition

Any tight, congruent, presuppositive type theory satisfies uniqueness of typing:

if $\Gamma \vdash a : A$ *and* $\Gamma \vdash a : A'$, *then* $\Gamma \vdash A \equiv A'$.

Substitutivity, congruity

Two more properties, a bit more negotiable depending on choice of structural rules: *substitutivity*, *congruity*.

Definition

- A rule is substitutive if the context of its conclusion is empty.
- A type theory is substitutive if all its rules are.

Cf. *universal* vs *hypothetical* forms of rules.

Substitutivity, congruity

Two more properties, a bit more negotiable depending on choice of structural rules: *substitutivity*, *congruity*.

Definition

- A rule is substitutive if the context of its conclusion is empty.
- A type theory is substitutive if all its rules are.

Cf. *universal* vs *hypothetical* forms of rules.

Congruity: Every object-judgement rule has an associated congruence rule. Can include these as structural rules, or ask they be included in the raw type theory.

Substitutivity, congruity

Two more properties, a bit more negotiable depending on choice of structural rules: *substitutivity*, *congruity*.

Definition

- A rule is substitutive if the context of its conclusion is empty.
- A type theory is substitutive if all its rules are.

Cf. *universal* vs *hypothetical* forms of rules.

Congruity: Every object-judgement rule has an associated congruence rule. Can include these as structural rules, or ask they be included in the raw type theory.

Proposition

- *Over a substitutive type theory, the substitution structural rule can be eliminated.*
- *Given the substitution structural rule, every rule is equivalent to a substitutive one.*

Orderedness

Major missing ingredient so far: order of presentation.

Shows up at various levels:

- Types of a context
- Premises of a rule
- Rules of a theory

Raw expressions of each type/premise/rule use only earlier variables/metavariables/constructors.

Typechecking of each component use only earlier variable-typing/premises/rules.

Orderedness

Major missing ingredient so far: order of presentation.

Shows up at various levels:

- Types of a context
- Premises of a rule
- Rules of a theory

Raw expressions of each type/premise/rule use only earlier variables/metavariables/constructors.

Typechecking of each component use only earlier variable-typing/premises/rules.

Definition

Well-formed (sequential) contexts: inductively defined.

- [] is a well-formed context of length 0;
- for Γ a well-formed context of length n,
  and A a type expression in scope n,
  the extension $(\Gamma; A)$ is a well-formed context of length $n + 1$.

Ordered rules

Definition

Sequentially-presented premise family over signature Σ, raw type theory T:

1. $\emptyset$ is a sequentially-presented premise family, of arity $\emptyset$;
2. for P a seq.-pres. prem. fam. of arity a,
   and B a judgement boundary in $\Sigma + a$,
   of form j and context length n, well-formed over $\mathsf{T} + P$,
   the extension $(P; B)$ is a seq.-pres. prem. fam. of arity $(a; (j, n))$.

Sequentially-presented headless rule over Σ, T:
a seq.-pres. premise family P of arity P,
together with a boundary C over $\Sigma + a$, well-formed over $\mathsf{T} + P$.

Ordered rules

Definition

Sequentially-presented premise family over signature Σ, raw type theory T:

1. $\emptyset$ is a sequentially-presented premise family, of arity $\emptyset$;
2. for P a seq.-pres. prem. fam. of arity a,
   and B a judgement boundary in $\Sigma + a$,
   of form j and context length n, well-formed over $\mathsf{T} + P$,
   the extension $(P; B)$ is a seq.-pres. prem. fam. of arity $(a; (j, n))$.

Sequentially-presented headless rule over Σ, T:
a seq.-pres. premise family P of arity P,
together with a boundary C over $\Sigma + a$, well-formed over $\mathsf{T} + P$.

Why are premises and conclusion given just as boundaries?

To ensure tightness.

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\vdash \_ \ \mathsf{type}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\vdash A \text{ type}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\begin{array}{c} \vdash \mathsf{A} \ \mathsf{type} \\ x{:}\mathsf{A} \vdash \underline{\qquad} \ \mathsf{type} \end{array}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\vdash \mathsf{A}\ \mathsf{type}$$
$$x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule “headless”): its head (if any) will later be filled in as the constructor it introduces.

$$\vdash A \text{ type}$$
$$x{:}A \vdash B(x) \text{ type}$$
$$\vdash \_ : \Pi(x{:}A)B(x)$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\vdash \mathsf{A}\ \mathsf{type}$$
$$x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}$$
$$\vdash \mathsf{f} : \Pi(x{:}\mathsf{A})\mathsf{B}(x)$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\vdash A \text{ type}$$
$$x{:}A \vdash B(x) \text{ type}$$
$$\vdash f : \Pi(x{:}A)B(x)$$
$$\vdash \_ : A$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\vdash \mathsf{A}\ \mathsf{type}$$
$$x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}$$
$$\vdash \mathsf{f} : \Pi(x{:}\mathsf{A})\mathsf{B}(x)$$
$$\vdash \mathsf{a} : \mathsf{A}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule “headless”): its head (if any) will later be filled in as the constructor it introduces.

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \qquad \vdash \mathsf{f} : \Pi(x{:}\mathsf{A})\mathsf{B}(x) \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \underline{\qquad\qquad} : \mathsf{B}(\mathsf{a})}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\frac{\begin{array}{c}\vdash A \text{ type} \\ x{:}A \vdash B(x) \text{ type} \\ \vdash f : \Pi(x{:}A)B(x) \\ \vdash a : A\end{array}}{\vdash \underline{\qquad\qquad\qquad} : B(a)}\ \textsc{app}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule “headless”): its head (if any) will later be filled in as the constructor it introduces.

$$\frac{\begin{array}{c} \vdash \mathsf{A}\ \mathsf{type} \\ x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \\ \vdash \mathsf{f} : \Pi(x{:}\mathsf{A})\mathsf{B}(x) \\ \vdash \mathsf{a} : \mathsf{A} \end{array}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}\ \textsc{app}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\frac{\begin{array}{c}\vdash \mathsf{A}\ \mathsf{type} \\ x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \\ \vdash \mathsf{f} : \Pi(x{:}\mathsf{A})\mathsf{B}(x) \\ \vdash \mathsf{a} : \mathsf{A}\end{array}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}\ \textsc{app}$$

Ordered rules

Premises just boundaries: their heads will be filled in with the corresponding metavariables

Similarly, conclusion just a boundary (rule "headless"): its head (if any) will later be filled in as the constructor it introduces.

$$\frac{\begin{array}{c} \vdash \mathsf{A}\ \mathsf{type} \\ x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \\ \vdash \mathsf{f} : \Pi(x{:}\mathsf{A})\mathsf{B}(x) \\ \vdash \mathsf{a} : \mathsf{A} \end{array}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})}\ \textsc{app}$$

Ordered type theories

Definition

Linearly well-presented type theories: defined inductively.

1. $\emptyset$ is a linearly well-presented type theory.
2. For T linearly well-presented, and R a sequentially-presented headless rule over T, the extension $(\mathsf{T}; R)$ is linearly well-presented.
3. For α a limit ordinal, and $\langle T_i \rangle_{i \in \alpha}$ an increasing sequence of linearly well-presented type-theories, the union $\bigcup_{i<\alpha} \mathsf{T}_i$ is linearly well-presented.

Ordered type theories

Definition

Linearly well-presented type theories: defined inductively.

1. ~~$\emptyset$ is a linearly well-presented type theory.~~ (Follows from 3.)
2. For T linearly well-presented, and R a sequentially-presented headless rule over T, the extension $(\mathsf{T}; R)$ is linearly well-presented.
3. For α a limit ordinal, and $\langle T_i \rangle_{i \in \alpha}$ an increasing sequence of linearly well-presented type-theories, the union $\bigcup_{i<\alpha} \mathsf{T}_i$ is linearly well-presented.

Ordered type theories

Definition

Linearly well-presented type theories: defined inductively.

1. ~~$\emptyset$ is a linearly well-presented type theory.~~ (Follows from 3.)
2. For T linearly well-presented, and R a sequentially-presented headless rule over T, the extension $(\mathsf{T}; R)$ is linearly well-presented.
3. For α a limit ordinal, and $\langle T_i \rangle_{i \in \alpha}$ an increasing sequence of linearly well-presented type-theories, the union $\bigcup_{i<\alpha} \mathsf{T}_i$ is linearly well-presented.

Two equivalent ways to read this: an inductive-recursive type; or an inductive predicate on raw type theories.

Ordered type theories

Definition

Linearly well-presented type theories: defined inductively.

1. ~~$\emptyset$ is a linearly well-presented type theory.~~ (Follows from 3.)
2. For T linearly well-presented, and R a sequentially-presented headless rule over T, the extension $(\mathsf{T}; R)$ is linearly well-presented.
3. For α a limit ordinal, and $\langle T_i \rangle_{i \in \alpha}$ an increasing sequence of linearly well-presented type-theories, the union $\bigcup_{i<\alpha} \mathsf{T}_i$ is linearly well-presented.

Two equivalent ways to read this: an inductive-recursive type; or an inductive predicate on raw type theories.

(Cf. Uemura signatures.)

Shortcomings:

- In many examples, order not naturally total.
- Constructively, assuming order total is not WLOG!

Ordered type theories

Definition

Well-presented type theory:

- A well-ordering $(\mathcal{I}, \prec)$, and family $\langle (a_i, R_i, D_i) \rangle_{i \in \mathcal{I}}$, where
- each a_i is a finite rule-arity;
- each R_i is a seq.-pres. headless raw rule of arity a_i, over the signature derived from $\langle a_j \rangle_{j \prec i}$;
- each D_i is a tuple of derivations witnessing that R_i is well-formed over the raw type theory $\langle R_j \rangle_{j \prec i}$.

Concisely: A well-ordered family of rules, each well-formed over the type theory formed by the earlier rules.

Ordered type theories

Definition

Well-presented type theory:

- A well-ordering $(\mathcal{I}, \prec)$, and family $\langle (a_i, R_i, D_i) \rangle_{i \in \mathcal{I}}$, where
- each a_i is a finite rule-arity;
- each R_i is a seq.-pres. headless raw rule of arity a_i, over the signature derived from $\langle a_j \rangle_{j \prec i}$;
- each D_i is a tuple of derivations witnessing that R_i is well-formed over the raw type theory $\langle R_j \rangle_{j \prec i}$.

Concisely: A well-ordered family of rules, each well-formed over the type theory formed by the earlier rules.

(Formally: 3 separate families $\langle a_i \rangle_{i \in \mathcal{I}}$, $\langle R_i \rangle_{i \in \mathcal{I}}$, $\langle D_i \rangle_{i \in \mathcal{I}}$?)

Ordered type theories

Definition

Well-presented type theory:

- A well-ordering $(\mathcal{I}, \prec)$, and family $\langle (a_i, R_i, D_i) \rangle_{i \in \mathcal{I}}$, where
- each a_i is a finite rule-arity;
- each R_i is a seq.-pres. headless raw rule of arity a_i, over the signature derived from $\langle a_j \rangle_{j \prec i}$;
- each D_i is a tuple of derivations witnessing that R_i is well-formed over the raw type theory $\langle R_j \rangle_{j \prec i}$.

Concisely: A well-ordered family of rules, each well-formed over the type theory formed by the earlier rules.

(Formally: 3 separate families $\langle a_i \rangle_{i \in \mathcal{I}}$, $\langle R_i \rangle_{i \in \mathcal{I}}$, $\langle D_i \rangle_{i \in \mathcal{I}}$?)

Proposition

A well-presented type theory is congruous, substitutive, tight, & presuppositive. □

Notions of type theory

- raw type theory
  - reasonably elementary
  - certainly part of traditional reading of type theories
  - very general: “niceness” not assumed/implied
  - used as an auxiliary notion in nicer definitions

Notions of type theory

- raw type theory
  - reasonably elementary
  - certainly part of traditional reading of type theories
  - very general: “niceness” not assumed/implied
  - used as an auxiliary notion in nicer definitions

- raw type theory + niceness properties
  - reasonably elementary
  - arguably reflects traditional intentions
  - semantics unclear

Notions of type theory

- raw type theory
  - reasonably elementary
  - certainly part of traditional reading of type theories
  - very general: “niceness” not assumed/implied
  - used as an auxiliary notion in nicer definitions

- raw type theory + niceness properties
  - reasonably elementary
  - arguably reflects traditional intentions
  - semantics unclear

- linearly well-presented type theory
  - reasonably clear definition
  - enjoys strong niceness properties, good semantics
  - linearity not part of traditional intention?

Notions of type theory

- raw type theory
  - reasonably elementary
  - certainly part of traditional reading of type theories
  - very general: "niceness" not assumed/implied
  - used as an auxiliary notion in nicer definitions
- raw type theory + niceness properties
  - reasonably elementary
  - arguably reflects traditional intentions
  - semantics unclear
- linearly well-presented type theory
  - reasonably clear definition
  - enjoys strong niceness properties, good semantics
  - linearity not part of traditional intention?
- (general) well-presented type theory
  - definition hard to formulate clearly
  - enjoys strong niceness properties, good semantics
  - reflects traditional intentions well?

Categorical analysis

Raw type theories form category **RTT**.

Rule-extension: inclusion maps $\mathsf{T} \longrightarrow (\mathsf{T}; R)$ in **RTT**.

Categorical analysis

Raw type theories form category **RTT**.

Rule-extension: inclusion maps $\mathsf{T} \longrightarrow (\mathsf{T}; R)$ in **RTT**.

Linearly well-presented type theories: cell complexes of rule-extensions,
i.e. transfinite composite

$$0 = T_0 \longrightarrow T_1 \longrightarrow \cdots \longrightarrow T_\alpha \longrightarrow \cdots \quad \text{for all } \alpha < \kappa$$

rule-extension at successor stages, colimit at limit stages.

Categorical analysis

Raw type theories form category **RTT**.

Rule-extension: inclusion maps $\mathsf{T} \longrightarrow (\mathsf{T}; R)$ in **RTT**.

Linearly well-presented type theories: cell complexes of rule-extensions,
i.e. transfinite composite

$$0 = T_0 \longrightarrow T_1 \longrightarrow \cdots \longrightarrow T_\alpha \longrightarrow \cdots \quad \text{for all } \alpha < \kappa$$

rule-extension at successor stages, colimit at limit stages.

Well-presented type theories: good colimits (Lurie) / fat cell complexes (cf. Makkai, Rosický, Vokřínek) of rule extensions.

Semantics

Have category **STT** of semantic type theories.

Roughly: a STT is an ess. alg. theory extending CwF's by adding operations strictly stable under reindexing. (Cf. Isaev 2016.)

Semantics

Have category **STT** of semantic type theories.

Roughly: a STT is an ess. alg. theory extending CwF's by adding operations strictly stable under reindexing. (Cf. Isaev 2016.)

A correspondence E between a raw type theory T and semantic type theory $\mathbf{S}$: an equivalence $\mathbf{AlgExt}(\mathsf{T}) \simeq \mathbf{Mod}(\mathbf{S})$, acting "the obvious way" on underlying CwF's.

Correspondences extend by rules: for $E : \mathsf{T} \simeq \mathbf{S}$ and R a rule over T, get $(E; R) : (T; R) \simeq (\mathbf{S}; E[R])$.

Correspondences respect suitable colimits.

Semantics

Have category **STT** of semantic type theories.

Roughly: a STT is an ess. alg. theory extending CwF's by adding operations strictly stable under reindexing. (Cf. Isaev 2016.)

A correspondence E between a raw type theory T and semantic type theory $\mathbf{S}$: an equivalence $\mathbf{AlgExt}(\mathsf{T}) \simeq \mathbf{Mod}(\mathbf{S})$, acting "the obvious way" on underlying CwF's.

Correspondences extend by rules: for $E : \mathsf{T} \simeq \mathbf{S}$ and R a rule over T, get $(E; R) : (T; R) \simeq (\mathbf{S}; E[R])$.

Correspondences respect suitable colimits.

Theorem

- *Any (linearly or generally) well-presented type theory* T *has a corresponding semantic type theory* $\mathbf{S}_\mathsf{T}$.
- *The syntactic CwF of* T *underlies the initial model of* $\mathbf{S}_\mathsf{T}$.
- *For familiar* T, $\mathbf{S}_\mathsf{T}$ *is exactly the standard CwF-based semantics.*

A closing curiosity

Very dependent function types (Hickey 1996):

- type of functions over a well-founded domain,
- type of each value can depend on earlier values.

$$\frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma,\ x, y{:}A \vdash x \prec y\ \mathsf{type} \qquad \Gamma \vdash H : \mathsf{IsWellFounded}[A, \prec] \qquad \Gamma,\ x{:}A,\ f : \{g \mid p{:}\Sigma(y{:}A)y \prec x \longrightarrow B(p, g)\} \vdash B(x, f)\ \mathsf{type}}{\Gamma \vdash \{f \mid x{:}A \longrightarrow B(x, f)\}\ \mathsf{type}}$$

(Several details swept under rug here.)

A closing curiosity

Very dependent function types (Hickey 1996):

- type of functions over a well-founded domain,
- type of each value can depend on earlier values.

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma,\ x, y{:}A \vdash x \prec y \text{ type} \qquad \Gamma \vdash H : \mathsf{IsWellFounded}[A, \prec] \qquad \Gamma,\ x{:}A,\ f : \{g \mid p{:}\Sigma(y{:}A)y \prec x \longrightarrow B(p, g)\} \vdash B(x, f) \text{ type}}{\Gamma \vdash \{f \mid x{:}A \longrightarrow B(x, f)\} \text{ type}}$$

(Several details swept under rug here.)

- Allows clean definition of well-presented type theories.
- Natural example of non-well-presented type theory!

Summary

- Various principles in mind when presenting type theories.
- Usually followed; can always be followed WLONG.
- Congruity; substitutivity; tightness; presuppositivity…
- Well-ordered presentations!
- Categorical analysis: (fat) cell complexes of rule-extensions.
- Well-presented type theory: sufficient to assign a good CwF-based semantics.

Appendix: related work

Note: here have focused on concrete details of our approach.

For comparison with related work — in particular, LF-based approaches — see PLL's Edinburgh LFCS seminar talk, *General definitions of dependent type theories*, 21 April 2020, https://youtu.be/FTyQ5EFOtbQ.