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

\[
\begin{align*}
\Gamma \vdash A \text{ type} & \quad \Gamma, x:A \vdash B \text{ type} \quad \Pi \\
\hline
\Gamma \vdash \Pi(x:A)B \text{ type} \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash A \text{ type} & \quad \Gamma, x:A \vdash B \text{ type} \\
\Gamma, x:A \vdash b : B \\
\hline
\Gamma \vdash \lambda x:A. \, b \text{ type} \quad \lambda \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash A \text{ type} & \quad \Gamma, x:A \vdash B \text{ type} \\
\Gamma \vdash f : \Pi(x:A)B & \quad \Gamma \vdash a : A \\
\hline
\Gamma \vdash \text{app}_{x:A,B} (f, a) : B[a/x] \quad \text{APP} \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash A \text{ type} & \quad \Gamma, x:A \vdash B \text{ type} \\
\Gamma, x:A \vdash b : B & \quad \Gamma \vdash a : A \\
\hline
\Gamma \vdash \text{app}_{x:A,B} (((\lambda x:A. \, b), a)) \equiv b[a/x] : B[a/x] \quad \beta \\
\end{align*}
\]
Some unfamiliar rules

\[
\begin{align*}
\Gamma, x: \mathbb{N} &\vdash P \text{ type} \\
\Gamma &\vdash T(x.P) \text{ type}
\end{align*}
\]
Some unfamiliar rules

\[
\Gamma, x:N \vdash P \text{ type} \\
\Gamma \vdash T(x.P) \text{ type}
\]

\[
\Gamma, x:N \vdash P \text{ type} \\
\Gamma \vdash t_{x.P} : P[\text{refl}(0)/x]
\]
Some unfamiliar rules

\[
\begin{align*}
\Gamma, x : \text{N} & \vdash P \text{ type} \\
\Gamma & \vdash T(x.P) \text{ type} \\
\Gamma, x : \text{N} & \vdash P \text{ type} \\
\Gamma & \vdash t_{x.P} : P[\text{refl}(0)/x] \\
\Gamma & \vdash a : Q[0/x, \text{refl}(0)/y] \\
\Gamma & \vdash T_{x.Q,a} \text{ type}
\end{align*}
\]
Some unfamiliar rules

\[
\begin{align*}
\Gamma, x: N &\vdash P \text{ type} \\
\hline
\Gamma &\vdash T(x.P) \text{ type} \\
\end{align*}
\]

\[
\begin{align*}
\Gamma, x: N &\vdash P \text{ type} \\
\hline
\Gamma &\vdash t_{x.P} : P[\text{refl}(0)/x] \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &\vdash a : Q[0/x, \text{refl}(0)/y] \\
\hline
\Gamma &\vdash T_{x.Q,a} \text{ type} \\
\end{align*}
\]

\[
\begin{align*}
\Gamma &\vdash a : \text{Bool} \\
\Gamma, x: N &\vdash a : \text{Id}_N(x, x) \\
\hline
\Gamma &\vdash T(a) \text{ type} \\
\end{align*}
\]
Some unfamiliar rules

\[
\begin{align*}
\Gamma, x: \mathbb{N} \vdash P \text{ type} & \quad \Rightarrow \quad \Gamma \vdash T(x.P) \text{ type} \\
\Gamma, x: \mathbb{N} \vdash P \text{ type} & \quad \Rightarrow \quad \Gamma \vdash t_{x.P} : P[\text{refl}(0)/x] \\
\Gamma \vdash a : Q[0/x, \text{refl}(0)/y] & \quad \Rightarrow \quad \Gamma \vdash T_{x.Q,a} \text{ type} \\
\Gamma \vdash a : \text{Bool} & \quad \Rightarrow \quad \Gamma, x: \mathbb{N} \vdash a : \text{Id}_{\mathbb{N}}(x, x) \\
& \quad \Rightarrow \quad \Gamma \vdash T(a) \text{ type}
\end{align*}
\]

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 behaviour, algebraic semantics.
Signatures

Definition

- **Syntactic classes**: Ty, Tm.
- **Arity**: a list of pairs of syntactic class and number.
- **Signature**:
  - a set $\Sigma$ of symbols;
  - a function $a : \Sigma \rightarrow \text{Class} \times \text{Arity}$, 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 $\Pi$-types:

\[
\begin{align*}
\Pi & \quad \text{Ty} & \quad [(\text{Ty}, 0), (\text{Ty}, 1)] \\
\lambda & \quad \text{Tm} & \quad [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 1)] \\
\text{app} & \quad \text{Tm} & \quad [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 0), (\text{Tm}, 0)]
\end{align*}
\]
Signatures

Definition

- **Syntactic classes**: Ty, Tm.
- **Arity**: a list of pairs of syntactic class and number.
- **Signature**:
  - a set $\Sigma$ of symbols;
  - a function $a : \Sigma \to \text{Class} \times \text{Arity}$, 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 $\Pi$-types:

$$
\begin{align*}
\Pi & \quad \text{Ty} & [(\text{Ty}, 0), (\text{Ty}, 1)] \\
\lambda & \quad \text{Tm} & [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 1)] \\
\text{app} & \quad \text{Tm} & [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 0), (\text{Tm}, 0)]
\end{align*}
$$
Signatures

Definition

- **Syntactic classes**: Ty, Tm.
- **Arity**: a list of pairs of syntactic class and number.
- **Signature**: 
  - a set $\Sigma$ of symbols;
  - a function $a : \Sigma \rightarrow \text{Class} \times \text{Arity}$, 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 $\Pi$-types:

$$
\begin{align*}
\Pi & \quad \text{Ty} & \quad [(\text{Ty}, 0), (\text{Ty}, 1)] \\
\lambda & \quad \text{Tm} & \quad [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 1)] \\
\text{app} & \quad \text{Tm} & \quad [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 0), (\text{Tm}, 0)]
\end{align*}
$$
Signatures

Definition

- **Syntactic classes**: Ty, Tm.
- **Arity**: a list of pairs of syntactic class and number.
- **Signature**:
  - a set \( \Sigma \) of symbols;
  - a function \( a : \Sigma \to \text{Class} \times \text{Arity} \), 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 \( \Pi \)-types:

\[
\begin{align*}
\Pi & \quad \text{Ty} & \quad [(\text{Ty}, 0), (\text{Ty}, 1)] \\
\lambda & \quad \text{Tm} & \quad [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 1)] \\
\text{app} & \quad \text{Tm} & \quad [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 0), (\text{Tm}, 0)]
\end{align*}
\]
Signatures

Definition

- **Syntactic classes**: Ty, Tm.
- **Arity**: a list of pairs of syntactic class and number.
- **Signature**:
  - a set $\Sigma$ of symbols;
  - a function $a : \Sigma \rightarrow \text{Class} \times \text{Arity}$, 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 $\Pi$-types:

- $\Pi$ Ty $[(\text{Ty, 0)}, (\text{Ty, 1})]$  
- $\lambda$ Tm $[(\text{Ty, 0)}, (\text{Ty, 1}), (\text{Tm, 1})]$  
- app Tm $[(\text{Ty, 0)}, (\text{Ty, 1}), (\text{Tm, 0}), (\text{Tm, 0})]$
Signatures

Definition

▷ **Syntactic classes**: Ty, Tm.
▷ **Arity**: a list of pairs of syntactic class and number.
▷ **Signature**:
  ▶ a set Σ of symbols;
  ▶ a function $a : \Sigma \rightarrow \text{Class} \times \text{Arity}$, 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:

\[
\begin{align*}
\Pi & \quad \text{Ty} & [(Ty, 0), (Ty, 1)] \\
\lambda & \quad \text{Tm} & [(Ty, 0), (Ty, 1), (Tm, 1)] \\
\text{app} & \quad \text{Tm} & [(Ty, 0), (Ty, 1), (Tm, 0), (Tm, 0)]
\end{align*}
\]
Signatures

Definition

- **Syntactic classes**: Ty, Tm.
- **Arity**: a list of pairs of syntactic class and number.
- **Signature**:
  - a set $\Sigma$ of symbols;
  - a function $a : \Sigma \rightarrow \text{Class} \times \text{Arity}$, 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 $\Pi$-types:

$$
\begin{align*}
\Pi & \quad \text{Ty} & & [(\text{Ty}, 0), (\text{Ty}, 1)] \\
\lambda & \quad \text{Tm} & & [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 1)] \\
\text{app} & \quad \text{Tm} & & [(\text{Ty}, 0), (\text{Ty}, 1), (\text{Tm}, 0), (\text{Tm}, 0)]
\end{align*}
$$
Expressions, judgements

Definition

Over a signature $\Sigma$, define:

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

\[
\begin{align*}
\text{Ty} & \quad \Gamma \vdash A \text{ type} \\
\text{Tm} & \quad \Gamma \vdash a : A \\
\text{TyEq} & \quad \Gamma \vdash A \equiv B \\
\text{TmEq} & \quad \Gamma \vdash a \equiv b : A
\end{align*}
\]
Expressions, judgements

Definition

Over a signature \( \Sigma \), define:

- **Raw (scoped) expressions** \( \text{Expr}_\Sigma^\text{Ty}(n), \text{Expr}_\Sigma^\text{Tm}(n) \): sets of raw type/term expressions in \( n \) variables

- **Raw contexts** \( \Gamma \): suitable lists of raw type expressions

- **Judgement forms, judgements**: suitable lists/tuples of expressions

\[
\begin{align*}
\text{Ty} & \quad \Gamma \vdash A \text{ type} \\
\text{Tm} & \quad \Gamma \vdash a : A \\
\text{TyEq} & \quad \Gamma \vdash A \equiv B \\
\text{TmEq} & \quad \Gamma \vdash a \equiv b : A
\end{align*}
\]

\begin{align*}
\begin{cases}
\{ \text{object judgements} \} \\
\{ \text{equality judgements} \}
\end{cases}
\end{align*}
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

\[
\begin{align*}
\Gamma \vdash A \text{ type} \\
\Gamma, x:A \vdash B \text{ type} \\
\Gamma \vdash f : \Pi(x:A)B \\
\Gamma \vdash a : A \\
\Gamma \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

for all raw \( \Gamma, A, B, f, a, \) if

\[
\begin{align*}
\Gamma \vdash A \text{ type} \\
\Gamma, x:A \vdash B \text{ type} \\
\Gamma \vdash f : \Pi(x:A)B \\
\Gamma \vdash a : A \\
\Gamma \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

are all derivable, then

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

\[
\begin{align*}
\Gamma &\vdash A \text{ type} \\
\Gamma, x:A &\vdash B \text{ type} \\
\Gamma &\vdash f : \Pi(x:A)B \\
\Gamma &\vdash a : A \\
\hline \\
\Gamma &\vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

for all raw \(\Gamma, A, B, f, a\), if

\[
\begin{align*}
\Gamma &\vdash A \text{ type} \\
\Gamma, x:A &\vdash B \text{ type} \\
\Gamma &\vdash f : \Pi(x:A)B \\
\Gamma &\vdash a : A \\
\hline \\
\Gamma &\vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

are all derivable, then

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

\[
\begin{align*}
\Gamma \vdash A \text{ type} \\
\Gamma, x:A \vdash B \text{ type} \\
\Gamma \vdash f : \Pi(x:A)B \\
\Gamma \vdash a : A \\
\hline
\Gamma \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

for all \( \Gamma, A, B, f, a, \) if

\[
\begin{align*}
\Gamma \vdash A \text{ type} \\
\Gamma, x:A \vdash B \text{ type} \\
\Gamma \vdash f : \Pi(x:A)B \\
\Gamma \vdash a : A \\
\hline
\Gamma \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

are all derivable, then

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

\[
\begin{align*}
\Gamma & \vdash A \text{ type} \\
\Gamma, x:A & \vdash B \text{ type} \\
\Gamma & \vdash f : \Pi(x:A)B \\
\Gamma & \vdash a : A \\
\hline \\
\Gamma & \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

for all raw \( \Gamma, A, B, f, a \), if

\[
\begin{align*}
\Gamma & \vdash A \text{ type} \\
\Gamma, x:A & \vdash B \text{ type} \\
\Gamma & \vdash f : \Pi(x:A)B \\
\Gamma & \vdash a : A \\
\hline \\
\Gamma & \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

are all derivable, then

\[
\begin{align*}
\Gamma & \vdash a : A \\
\hline \\
\Gamma & \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

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!

\[
\begin{align*}
\Gamma \vdash A & \text{ type} \\
x : A & \vdash B \text{ type} \\
\Gamma \vdash f : \Pi(x : A) B \\
\Gamma \vdash a : A \\
\hline
\Gamma \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

for all raw \( \Gamma, A, B, f, a, \) if

\[
\begin{align*}
\Gamma, x : A & \vdash B \text{ type} \\
\Gamma & \vdash f : \Pi(x : A) B \\
\Gamma & \vdash a : A \\
\hline
\Gamma & \vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

are all derivable, then

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

\[
\begin{align*}
\Gamma &\vdash A \text{ type} \\
x : A &\vdash B \text{ type} \\
\Gamma &\vdash f : \Pi(x : A)B \\
\Gamma &\vdash a : A \\
\hline
\Gamma &\vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

for all raw $\Gamma, A, B, f, a$, if
\[
\begin{align*}
\Gamma &\vdash A \text{ type} \\
\Gamma, x : A &\vdash B \text{ type} \\
\Gamma &\vdash f : \Pi(x : A)B \\
\Gamma &\vdash a : A \\
\hline
\Gamma &\vdash \text{app}(A, B, f, a) : B[a/x]
\end{align*}
\]

are all derivable, then $\Gamma \vdash \text{app}(A, B, f, a) : B[a/x]$ 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.

\[
\begin{align*}
\Gamma \vdash A \text{ type} \\
\vdash x : A \vdash B(x) \text{ type} \\
\vdash f : \Pi(A, B(x)) \\
\vdash a : A \\
\hline
\vdash \text{app}(A, B(x), f, a) : B(a)
\end{align*}
\]

for all raw \( \Gamma, A, B, f, a, \) if

\[
\begin{align*}
\Gamma \vdash A \text{ type} \\
\Gamma, x : A \vdash B \text{ type} \\
\Gamma \vdash f : \Pi(x : A)B \\
\Gamma \vdash a : A
\end{align*}
\]

are all derivable, then

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

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.

\[
\begin{align*}
\Gamma &\vdash A \text{ type} \\
\Gamma &\vdash x : A \vdash B(x) \text{ type} \\
\Gamma &\vdash f : \Pi(A, B(x)) \\
\Gamma &\vdash a : A \\
\hline
\Gamma &\vdash \text{app}(\Gamma, B(x), f, a) : B(a)
\end{align*}
\]

for all raw \( \Gamma, A, B, f, a \), if

\[
\begin{align*}
\Gamma &\vdash A \text{ type} \\
\Gamma, x : A &\vdash B \text{ type} \\
\Gamma &\vdash f : \Pi(x : A) B \\
\Gamma &\vdash a : A \\
\hline
\Gamma &\vdash \text{app}(\Gamma, B, f, a) : B[a/x]
\end{align*}
\]

are all derivable, then 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.

for all raw $\Gamma, A, B, f, a$, if

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

are all derivable, then

$$
\Gamma \vdash \text{app}(A, B, f, a) : B[a/x] \text{ is derivable}
$$
Raw rules

Definition

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

Definition

- **M**etavariable extension $\Sigma + a$ ($a$ an arity): signature extending $\Sigma$ by symbols for the arguments of $a$.
- **R**aw rule over $\Sigma$ of arity $a$: family of judgements (premises) and one more judgement (conclusion), all over $\Sigma + a$.
- **I**nterpretation of $a$ over $\Sigma$: a raw context $\Gamma$ and suitable expressions according to $a$, specifying mapping from syntax of $\Sigma + a$ to syntax of $\Sigma$.
- **R**aw type theory over $\Sigma$: family of raw rules over $\Sigma$.
- **D**erivability over raw type theory $\mathcal{T}$: relation on judgements, inductively defined by closure conditions for
  - standard structural rules;
  - all instantiations of all raw rules in $\mathcal{T}$.
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):

\[
\begin{align*}
\Gamma & \vdash \_ \text{ type} \\
\Gamma & \vdash \_ : A \\
\Gamma & \vdash A \equiv A' \\
\Gamma & \vdash a \equiv a' : A
\end{align*}
\]

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.

\[
\begin{align*}
\vdash A & \text{ type} \\
x:A \vdash B(x) & \text{ type} \\
\vdash \Pi(A, B(x)) & \text{ type} \\
\vdash \Pi(A, B(x)) & \text{ type} \\
\vdash a : \text{Bool} & \text{ type} \\
x:N \vdash a : \text{Id}_N(x, x) & \text{ type} \\
\vdash T(a) & \text{ type}
\end{align*}
\]

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

\[ \vdash A \text{ type} \]
\[ x:A \vdash B(x) \text{ type} \]
\[ x:A \vdash B(x) \text{ type} \]
\[ \vdash \Pi(A, B(x)) \text{ type} \]
\[ \vdash \Pi(A, B(x)) \text{ type} \]

\[ \vdash a : \text{Bool} \]
\[ x:N \vdash a : \text{Id}_N(x, x) \]
\[ \vdash \text{T}(a) \text{ type} \]

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:*

\[ \text{if } \Gamma \vdash a : A \text{ and } \Gamma \vdash a : A', \text{ then } \Gamma \vdash A \equiv A'. \]
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/metavariabes/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/metavariabes/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 $\Sigma$, raw type theory $\mathcal{T}$:

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 $\mathcal{T} + P$, the extension $(P; B)$ is a seq.-pres. prem. fam. of arity $(a; (j, n))$.

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

Why are premises and conclusion given just as boundaries?
To ensure tightness.
Ordered rules

**Definition**

**Sequentially-presented premise family** over signature \( \Sigma \), 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 \( T + P \), the extension \( (P; B) \) is a seq.-pres. prem. fam. of arity \( (a; (j, n)) \).

**Sequentially-presented headless rule** over \( \Sigma \), \( T \):
a seq.-pres. premise family \( P \) of arity \( P \), together with a boundary \( C \) over \( \Sigma + a \), well-formed over \( T + P \).

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.

⊢ _ 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.

⊢ A 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.

\[ \vdash A \text{ type} \]
\[ x : A \vdash \text{___ type} \]
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.

\[ \vdash A \text{ type} \]
\[ x:A \vdash B(x) \text{ type} \]
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.

\[ \vdash A \text{ type} \]
\[ x:A \vdash B(x) \text{ type} \]
\[ \vdash _ : \Pi(x:A)B(x) \]
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.

\[ \vdash A \text{ type} \]
\[ x : A \vdash B(x) \text{ type} \]
\[ \vdash f : \Pi(x : A) B(x) \]
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.

\[ \vdash A \text{ type} \]
\[ x:A \vdash B(x) \text{ type} \]
\[ \vdash f : \Pi(x:A)B(x) \]
\[ \vdash _ : A \]
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.

\[ \vdash A \text{ type} \]
\[ x : A \vdash B(x) \text{ type} \]
\[ \vdash f : \Pi(x : A)B(x) \]
\[ \vdash a : A \]
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.

\[ \vdash A \text{ type} \]
\[ x:A \vdash B(x) \text{ type} \]
\[ \vdash f : \Pi(x:A)B(x) \]
\[ \vdash a : A \]

\[ \hline \]
\[ \vdash \underline{\text{ }} : B(a) \]
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.

\[ \vdash A \text{ type} \]
\[ x : A \vdash B (x) \text{ type} \]
\[ \vdash f : \Pi (x : A) B (x) \]
\[ \vdash a : A \]

\[ \underline{\vdash} \quad \underline{\vdash} \quad \underline{\vdash} \]

\[ \vdash \_ \_ \_ : B (a) \]
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.

\[
\begin{align*}
\vdash & \quad \text{A type} \\
\vdash & \quad x : A \vdash B(x) \quad \text{type} \\
\vdash & \quad f : \Pi(x : A)B(x) \\
\vdash & \quad a : A \\
\hline
\vdash & \quad \text{app}(A, B(x), f, a) : B(a)
\end{align*}
\]
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.

\[
\begin{align*}
\vdash & \text{A type} \\
\vdash & x: A \vdash B(x) \text{ type} \\
\vdash & f : \Pi(x:A)B(x) \\
\vdash & a : A \\
\hline
\vdash & \text{app}(A, B(x), f, a) : B(a)
\end{align*}
\]
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.

\[
\begin{align*}
\vdash A \text{ type} \\
x : A &\vdash B(x) \text{ type} \\
\vdash f : \Pi(x:A)B(x) \\
\vdash a : A \\
\hline
\vdash \text{app}(A, B(x), f, a) : B(a)
\end{align*}
\]
Ordered type theories

**Definition**

**Linearly well-presented type theories:** defined inductively.

1. $\varnothing$ is a linearly well-presented type theory.

2. For $T$ linearly well-presented, and $R$ a sequentially-presented headless rule over $T$, the extension $(T; R)$ is linearly well-presented.

3. For $\alpha$ a limit ordinal, and $\langle T_i \rangle_{i \in \alpha}$ an increasing sequence of linearly well-presented type-theories, the union $\bigcup_{i < \alpha} T_i$ 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 $(T; R)$ is linearly well-presented.

3. For $\alpha$ a limit ordinal, and $\langle T_i \rangle_{i \in \alpha}$ an increasing sequence of linearly well-presented type-theories, the union $\bigcup_{i < \alpha} T_i$ 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 $(T; R)$ is linearly well-presented.

3. For $\alpha$ a limit ordinal, and $\langle T_i \rangle_{i \in \alpha}$ an increasing sequence of linearly well-presented type-theories, the union $\bigcup_{i < \alpha} T_i$ 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 \((T; R)\) is linearly well-presented.

3. For \( \alpha \) a limit ordinal, and \( \langle T_i \rangle_{i \in \alpha} \) an increasing sequence of linearly well-presented type-theories, the union \( \bigcup_{i \prec \alpha} T_i \) 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 \((I, <)\), and family \(\langle (a_i, R_i, D_i) \rangle_{i \in I}\), 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 \(\langle a_j \rangle_{j < i}\);
  - each \(D_i\) is a tuple of derivations witnessing that \(R_i\) is
    well-formed over the raw type theory \(\langle R_j \rangle_{j < i}\).

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 \((I, <)\), and family \(\langle (a_i, R_i, D_i) \rangle_{i \in I}\), 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 \(\langle a_j \rangle_{j < i}\);
- each \(D_i\) is a tuple of derivations witnessing that \(R_i\) is well-formed over the raw type theory \(\langle R_j \rangle_{j < i}\).

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

(Formally: 3 separate families \(\langle a_i \rangle_{i \in I}, \langle R_i \rangle_{i \in I}, \langle D_i \rangle_{i \in I}\)?)
## Ordered type theories

### Definition

**Well-presented type theory:**

- A well-ordering \((I, \prec)\), and family \(\langle (a_i, R_i, D_i) \rangle_{i \in I}\), 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 \(\langle a_i \rangle_{i \in I}, \langle R_i \rangle_{i \in I}, \langle D_i \rangle_{i \in I}\)?)

### Proposition

*A well-presented type theory is congruous, substitutive, tight, & presuppositional.*
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 $T \rightarrow (T; R)$ in **RTT**.
Categorical analysis

Raw type theories form category $\text{RTT}$.  

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

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

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

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

Raw type theories form category \textbf{RTT}.

\textbf{Rule-extension}: inclusion maps \( T \rightarrow (T; R) \) in \textbf{RTT}.

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

\[ 0 = T_0 \rightarrow T_1 \rightarrow \cdots \rightarrow T_\alpha \rightarrow \cdots \]  

for all \( \alpha < \kappa \)

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

Well-presented type theories: \textbf{good colimits} (Lurie) / \textbf{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 \textbf{STT} of \textit{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 \textbf{correspondence} \(E\) between a raw type theory \(T\) and semantic type theory \(S\): an equivalence \(\text{AlgExt}(T) \simeq \text{Mod}(S)\), acting “the obvious way” on underlying CwF’s.

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

Correspondences respect suitable colimits.
Semantics

Have category $\text{STT}$ 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 $S$: an equivalence $\text{AlgExt}(T) \simeq \text{Mod}(S)$, acting “the obvious way” on underlying CwF’s.

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

Correspondences respect suitable colimits.

Theorem

- Any (linearly or generally) well-presented type theory $T$ has a corresponding semantic type theory $S_T$.
- The syntactic CwF of $T$ underlies the initial model of $S_T$.
- For familiar $T$, $S_T$ 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.

\[
\begin{align*}
\Gamma & \vdash A \text{ type} \\
\Gamma, \ x, y : A & \vdash x < y \text{ type} \\
\Gamma & \vdash H : \text{IsWellFounded}[A, <] \\
\Gamma, \ x : A, \ f : \{g \mid p : \Sigma(y : A) y < x \rightarrow B(p, g)\} & \vdash B(x, f) \text{ type} \\
\Gamma & \vdash \{f \mid x : A \rightarrow B(x, f)\} \text{ type}
\end{align*}
\]

(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.

\[
\begin{align*}
\Gamma &\vdash A \text{ type} \\
\Gamma, x, y:A &\vdash x < y \text{ type} \\
\Gamma &\vdash H : \text{IsWellFounded}[A, <] \\
\Gamma, x:A, f : \{g \mid p: \Sigma(y:A) y < x \rightarrow B(p, g)\} &\vdash B(x, f) \text{ type} \\
\Gamma &\vdash \{f \mid x:A \rightarrow B(x, f)\} \text{ type}
\end{align*}
\]

(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.