Predicate logic over a type setup

Peter Aczel
petera@cs.man.ac.uk

Manchester University and SCAS

Philosophy and Foundations of Mathematics:
Epistemological and Ontological Aspects-
a conference dedicated to Per Martin-Löf
on the occasion of his retirement May 5-8, 2009
Swedish Collegium for Advanced Study (SCAS), Uppsala, Sweden
Type setups and logic enriched type theories

- Dependent type theories often use a fixed interpretation of the logical notions; e.g. props-as-types or some variant.
- Logic enriched type theories leave logic uninterpreted.

Plan of Talk

- Some dependent type theories and their logics
- Some category notions for type dependency
- Type setups
- Logic over a type setup
- The disjunction and existence properties
- Propositions as types
Some dependent type theories and their logics, 1

- **Basic Martin-Löf type theory**, with the forms of type

\[(\Pi x : A)B(x), (\Sigma x : A)B(x), A_1 + A_2, N_k(k = 0, 1, \ldots), I(A, a_1, a_2)\]

with \(A_1 \to A_2 =_{def} (\Pi_\_ : A_1)A_2, \quad A_1 \times A_2 =_{def} (\Sigma_\_ : A_1)A_2.\)

**Propositions-as-Ty pes (à la Curry-Howard )**

**Proposition = Type**

<table>
<thead>
<tr>
<th>Prop</th>
<th>(\bot)</th>
<th>(\top)</th>
<th>(A_1 \supset A_2)</th>
<th>(A_1 \land A_2)</th>
<th>(A_1 \lor A_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>(N_0)</td>
<td>(N_1)</td>
<td>(A_1 \rightarrow A_2)</td>
<td>(A_1 \times A_2)</td>
<td>(A_1 + A_2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Prop</th>
<th>((\forall x : A)B(x))</th>
<th>((\exists x : A)B(x))</th>
<th>((a_1 =_A a_2))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>((\Pi x : A)B(x))</td>
<td>((\Sigma x : A)B(x))</td>
<td>(I(A, a_1, a_2))</td>
</tr>
</tbody>
</table>
Some dependent type theories and their logics, 2

- **Martin-Lof type theory** in a logical framework
  This has a predicative type universe of sets/propositions:
  \[
  \text{Proposition} = \text{Set} = \text{Datatype}
  \]
  \[(\forall x : A)B(x) : Prop\] can only be formed if \(A : Set\).
- **Coq type theory** (a calculus of inductive constructions)
  This has a predicative type universe \(Set\) of datatypes and an impredicative type \(Prop\) where \((\Pi x : A)B(x) : Prop\) can be formed even when we do not have \(A : Set\).
- The impredicative Russell-Prawitz representation of logic in \(Prop\) is used; This representation can be given in terms of the Russell-Prawitz modality, \(J\), where \(J\) assigns to each type \(A\) the type \(JA : Prop\), where

\[
JA \equiv (\Pi p : Prop)((A \to p) \to p).
\]
Some dependent type theories and their logics, 3

Propositions-as-Types (à la Russell-Prawitz)
Proposition = type in \( Prop \)

<table>
<thead>
<tr>
<th>( Prop )</th>
<th>( \bot )</th>
<th>( \top )</th>
<th>( A_1 \supset A_2 )</th>
<th>( A_1 \land A_2 )</th>
<th>( A_1 \lor A_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Type )</td>
<td>( JN_0 )</td>
<td>( JN_1 )</td>
<td>( A_1 \rightarrow A_2 )</td>
<td>( J(A_1 \times A_2) )</td>
<td>( J(A_1 + A_2) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( Prop )</th>
<th>( (\forall x : A) B(x) )</th>
<th>( (\exists x : A) B(x) )</th>
<th>( (a_1 =_A a_2) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Type )</td>
<td>( (\prod x : A) B(x) )</td>
<td>( J(\Sigma x : A) B(x) )</td>
<td>( JI(A, a_1, a_2) )</td>
</tr>
</tbody>
</table>

\[ J A \equiv (\prod p : Prop)((A \rightarrow p) \rightarrow p). \]

\[ J A : Prop \]
Some dependent type theories and their logics, 4

- So the propositions-as-types-à-la-Russell-Prawitz representation of intuitionistic logic is the result of applying the propositions-as-types-à-la-Curry-Howard representation of intuitionistic logic followed by the $j$-translation of intuitionistic logic into itself.
- The $j$-translation generalises the $\neg\neg$-translation for any unary connective $j$ satisfying the laws

\[ \phi \vdash j\phi \quad \text{and} \quad (\phi \vdash j\psi) \vdash (j\phi \vdash j\psi). \]

- Note: For types $A, B$,

\[ j1 : A \rightarrow JA \quad \text{and} \quad j2 : (A \rightarrow JB) \rightarrow (JA \rightarrow JB) \]

where

\[ j1 \equiv (\lambda x : A, p : Prop, y : A \rightarrow p) \ y(x) \]
\[ j2 \equiv (\lambda x : A \rightarrow JB, y : JA) \ y(JB)(x). \]
Logic enriched type theories

These are obtained from type theories by simply adding a logic ‘on top’, using the types of a type theory as the possible ranges of the free and bound variables.

- **Dependently Sorted Logic** is obtained as a logic enrichment of an elementary type theory whose types and typed terms are just the sorts and sorted terms built up using sort and term constructors that may be dependent.
  - Each sort has the form $F(t_1, \ldots, t_n)$, where $F$ is a sort constructor and $t_1, \ldots, t_n$ are terms whose types match the argument types of $F$.

- Makkai’s FOLDS is dependently sorted logic without function symbols.
Category notions for the semantics of type dependency

- Category with attributes Cartmell 1978, Moggi 1991, Type category Pitts 1997
- Contextual category Cartmell 1978, Streicher 1991
- Category with families Dybjer 1996, Hoffman 1997
- Category with display maps (less general) Taylor 1986, Lamarche 1987, Hyland and Pitts 1989
- Comprehension category (more general) Jacobs 1991
- other relevant notions: locally cartesian closed categories, fibrations, indexed categories
- Type setups (for syntax) new notion
Category with families (CwF)

- a category $Ctxt$ of contexts $\Gamma$ and substitutions $\sigma : \Delta \rightarrow \Gamma$, with a distinguished terminal object $()$,

- a functor $T : Ctxt^{op} \rightarrow Fam$ mapping
  $$\Gamma \mapsto \{Term(\Gamma, A)\}_{A \in Type(\Gamma)}$$

and, if $\sigma : \Delta \rightarrow \Gamma$ then
  $$A \in Type(\Gamma) \quad \mapsto \quad A\sigma \in Type(\Delta)$$
  $$a \in Term(\Gamma, A) \quad \mapsto \quad a\sigma \in Term(\Delta, A\sigma)$$

- an assignment, to each context $\Gamma$ and each $A \in Type(\Gamma)$, of a comprehension $(\Gamma.A, p_A, v_A)$ such that
  $$p_A : \Gamma.A \rightarrow \Gamma$$ and $v_A \in Term(\Gamma.A, A p_A)$;

i.e. a terminal object in the category of $(\Gamma', \theta, a)$ such that
  $$\theta : \Gamma' \rightarrow \Gamma$$ and $a \in Term(\Gamma', A\theta)$. 
The metamathematical notion of a type setup is an abstraction of the syntactic notion of a dependent type theory, as is the notion of a $CwF$. The notion keeps

- variables, $x$, types $A$ and terms $a$,
- contexts $\Gamma$ as finite sequences of variable declarations, $x : A$,
- substitutions, $\sigma : \Delta \rightarrow \Gamma$, as finite sequences of variable assignments $x := a$,
- forms of judgement

\[
\begin{align*}
(\Gamma) & \ A \ \text{type} & & A \in Type(\Gamma) \\
(\Gamma) & \ A = B & & A \sim_\Gamma B \\
(\Gamma) & \ a : A & & a \in Term(\Gamma, A) \\
(\Gamma) & \ a = b : A & & a \sim_{\Gamma,A} b
\end{align*}
\]
Type Setups, 2

But it does not require judgements to be generated using rules of inference or types and terms to be generated using rules of expression formation. Like a $CwF$, contexts and substitutions form a category $Ctx$ and there is a functor $T : Ctx^{op} \to Fam$ such that

- for each context $\Gamma$

$$T(\Gamma) = \{Term(\Gamma, A)\}_{A \in Type(\Gamma)}$$

- for each substitution $\sigma : \Delta \to \Gamma$, $T(\sigma) : T(\Gamma) \to T(\Delta)$ maps

$$A \in Type(\Gamma) \mapsto A\sigma \in Type(\Delta),$$
$$a \in Term(\Gamma, A) \mapsto a\sigma \in Term(\Delta, A\sigma).$$
The relations $\sim_{\Gamma}$ and $\sim_{\Gamma,A}$ are equivalence relations on $Type(\Gamma)$ and $Term(\Gamma, A)$ respectively, that are invariant under substitutions.

In extensional set-theoretical mathematics they can be taken to be identity relations on sets, while in Martin-Löf’s type theory they can be taken to be definitional equalities on sets.

If $\Gamma$ and $\Delta$ are contexts such that $\Gamma \subseteq \Delta$; i.e. every variable declaration of $\Gamma$ is a variable declaration of $\Delta$, then

\[
(\Gamma) \ldots \Rightarrow (\Delta) \ldots
\]

and there is an inclusion substitution map $\iota_{\Delta \rightarrow \Gamma} : \Delta \rightarrow \Gamma$ such that

\[
(\Gamma) A \text{ type} \Rightarrow (\Delta) A_{\iota_{\Delta \rightarrow \Gamma}} = A
\]

\[
(\Gamma) a : A \text{ type} \Rightarrow (\Delta) a_{\iota_{\Delta \rightarrow \Gamma}} = a : A
\]
Type Setups, 4

• A finite sequence of variable declarations

\[ \Gamma \equiv x_1 : A_1, \ldots, x_n : A_n \]

is a context iff, for \( i = 1, \ldots, n \),

1. \( \Gamma_{<i} \equiv x_1 : A_1, \ldots, x_{i-1} : A_{i-1} \) is a context,
2. \( A_i \in \text{Type}(\Gamma_{<i}) \), and
3. \( x_i \) is \( \Gamma_{<i} \)-free.

and then \( x_i \in \text{Term}(\Gamma, A_i) \) for \( i = 1, \ldots, n \).

• Also a finite sequence of variable declarations

\[ \sigma \equiv [x_1 := a_1, \ldots, x_n := a_n]_{\Delta \rightarrow \Gamma} \]

is a substitution, \( \Delta \rightarrow \Gamma \), iff, for \( i = 1, \ldots, n \),

1. \( \sigma_{<i} \equiv [x_1 := a_1, \ldots, x_{i-1} := a_{i-1}]_{\Delta \rightarrow \Gamma_{<i}} \), and
2. \( a_i \in \text{Term}(\Delta, A_i \sigma_{<i}) \).
• Suppose that $\Gamma$ and $\Delta$ are contexts, with

$$\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n.$$  

If $\sigma \equiv [x_1 := a_1, \ldots, x_n := a_n]_{\Delta \rightarrow \Gamma}$ is a substitution $\Delta \rightarrow \Gamma$, then for $i = 1, \ldots, n$,

$$(\Delta) \ x_i \sigma = a_i : A_i \sigma.$$  

• If also $\sigma' : \Delta \rightarrow \Gamma$ such that, for $i = 1, \ldots, n$,

$$(\Delta) \ x_i \sigma' = a_i : A_i \sigma'$$  

then, for each $A \in \text{Type}(\Gamma)$, $$(\Delta) \ A \sigma' = A \sigma$$  

and, for each $a \in \text{Term}(\Gamma, A)$, $$(\Delta) \ a \sigma' = a \sigma : A \sigma.$$
• If $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ is a context, $A \in Type(\Gamma)$ and $x$ is $\Gamma$-free then we write

$$(\Gamma, x : A)$$

for the context $x_1 : A_1, \ldots, x_n : A_n, x : A$.

• If $\triangle$ is a context such that $(\cdots (\triangle, x_1 : A_1), \cdots, x_n : A_n)$ is also a context then we write this context

$$(\triangle, \Gamma)$$

where $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$. 
Type Setups, 7: Some notation

• If $\sigma \equiv [x_1 := a_1, \ldots, x_n := a_n]_{\Delta \rightarrow \Gamma}$ is a substitution $\Delta \rightarrow \Gamma$ and $a \in \text{Term}(\Delta, A\sigma)$ then we write

$$[\sigma, x := a]_{\Delta \rightarrow (\Gamma, x:A)}$$

for the substitution $[x_1 := a_1, \ldots, x_n := a_n, x := a]_{\Delta \rightarrow (\Gamma, x:A)}$.

• More generally, if $(\Gamma, \Lambda)$ is a context then we can define a substitution $[\sigma, \tau]_{\Delta \rightarrow (\Gamma, \Lambda)}$ for suitable sequences $\tau$ of variable assignments.

• If $(\Gamma) \ a : A$ then we write

$$[a/x]$$

for the substitution $[\iota_{\Gamma \rightarrow \Gamma}, x := a]_{\Gamma \rightarrow (\Gamma, x:A)}$. 
Logic over a type setup

- We assume given a type setup with a predicate signature consisting of a set of predicate symbols, each assigned a context as its arity. We define the formulae and inference rules of a formal system of dependently sorted intuitionistic predicate logic with equality, whose sorts are the types of the type setup and whose individual terms are the terms of the setup.
- We use the predicate signature to define the atomic $\Gamma$-formulae to have the form

$$P(b_1, \ldots, b_m)$$

where $P$ is a predicate symbol of arity

$$\Delta \equiv (y_1 : B_1, \ldots, y_m : B_m)$$

and $b_1, \ldots, b_m$ are the terms of a substitution

$$[y_1 := b_1, \ldots, y_m := b_m]_{\Gamma \rightarrow \Delta}.$$
The $\Gamma$-Formulae

- The **formulae** are inductively generated using the following rules.

  - Every atomic $\Gamma$-formula $P(b_1, \ldots, b_m)$ is a $\Gamma$-formula.
  - If $a_1, a_2 \in \text{Term}(\Gamma, A)$, where $A \in \text{Type}(\Gamma)$, then $(a_1 =_A a_2)$ is a $\Gamma$-formula.
  - $\bot$ and $\top$ are $\Gamma$-formulae.
  - If $\psi_1, \psi_2$ are $\Gamma$-formulae then so is $(\psi_1 \Box \psi_2)$, where $\Box \in \{\land, \lor, \supset\}$.
  - If $\psi_0$ is a $(\Gamma, x : A)$-formula then $(\nabla x : A)\psi_0$ is a $\Gamma$-formulae where $\nabla \in \{\forall, \exists\}$. 

Predicate logic over a type setup – p.18/33
Substitution

We can define substitution into formulae in more or less the usual way by structural recursion on the formula. So, for each \( \Gamma \)-formula \( \phi \), we associate with each substitution \( \tau : \Lambda \rightarrow \Gamma \) a \( \Lambda \)-formula \( \phi \tau \) using the following equations.

- If \( \phi \equiv P(b_1, \ldots, b_m) \) then \( \phi \tau \equiv P(b_1 \tau, \ldots, b_m \tau) \).
- If \( \phi \equiv (a_1 =_A a_2) \) then \( \phi \tau \equiv (a_1 \tau =_A \tau a_2 \tau) \).
- If \( \phi \equiv \bot \) or \( \top \) then \( \phi \tau \equiv \bot \) or \( \top \) respectively.
- If \( \phi \equiv (\psi_1 \Box \psi_2) \), where \( \Box \in \{ \land, \lor, \supset \} \), then \( \phi \tau \equiv (\psi_1 \tau \Box \psi_2 \tau) \).
- If \( \phi \equiv (\nabla x : A)\psi_0 \), where \( \nabla \in \{ \forall, \exists \} \), then \( \phi \tau \equiv (\nabla \tau x : A\tau)\psi_0 \tau' \), where \( \tau' \equiv [\tau, x := x](\Lambda, x : A\tau) \rightarrow (\Gamma, x : A) \).
The rules of inference, 1

- These are essentially the standard sequent formulation of the natural deduction rules for intuitionistic predicate logic with equality, using sequents of the form \((\Gamma) \Phi \rightarrow \phi\) where \(\Gamma\) is a context, \(\Phi\) is a finite sequence of \(\Gamma\)-formulae and \(\phi\) is a \(\Gamma\)-formula.
- e.g. here are the quantifier rules:

\[
\begin{align*}
(\forall I) & \quad (\Gamma, x : A) \Phi \Rightarrow \psi_0 \quad \quad (\forall E) & \quad (\Gamma) \Phi \Rightarrow (\forall x : A)\psi_0 \\
(\Gamma) \Phi \Rightarrow (\forall x : A)\psi_0 \\
(\exists I) & \quad (\Gamma) \Phi \Rightarrow \psi_0[a/x] \quad \quad (\exists E) & \quad (\Gamma) \Phi \Rightarrow (\exists x : A)\psi_0 \\
(\Gamma) \Phi \Rightarrow (\exists x : A)\psi_0 \\
(\exists E) & \quad (\Gamma, x : A) \Phi, \psi_0 \Rightarrow \phi \\
(\Gamma) \Phi \Rightarrow \phi
\end{align*}
\]

- Here \(a \in \text{Term}(\Gamma, A)\) and \([a/x] \equiv [\iota_{\Gamma \rightarrow \Gamma, x := a}]_{\Gamma \rightarrow (\Gamma, x: A)}\).
And here are the equality rules:

\[
\begin{align*}
\text{(} = \ I \text{)} & \quad \frac{(\Gamma)\Phi \Rightarrow (a =_A a)}{} \\
\text{(= E)} & \quad \frac{(\Gamma)\Phi \Rightarrow (a_1 =_A a_2)}{\Gamma)\Phi \Rightarrow \psi_0[a_1/x]} \\
& \quad \frac{\psi_0[a_1/x]}{\Gamma)\Phi \Rightarrow \psi_0[a_2/x]}
\end{align*}
\]

where \( A \in Type(\Gamma) \) and \( a, a_1, a_2 \in Term(\Gamma, A) \).
The disjunction and existence properties, 1

Let $\Phi$ be a finite sequence of $\Delta$-formulae.

- $(\Delta, \Phi)$ has the **disjunction property** if, for all $\Delta$-formulae $\psi_1, \psi_2$,

  $$
  \vdash (\Delta) \Phi \Rightarrow (\psi_1 \lor \psi_2) \text{ implies } \vdash (\Delta) \Phi \Rightarrow \psi_i \text{ for some } i \in \{1, 2\}.
  $$

- $(\Delta, \Phi)$ has the **existence property** if, for all $A \in Type(\Delta)$ and every $(\Delta, x : A)$-formula $\psi_0$,

  $$
  \vdash (\Delta) \Phi \Rightarrow (\exists x : A)\psi_0 \text{ implies } \vdash (\Delta) \Phi \Rightarrow \psi_0[a/x] \text{ for some } a \in Term(\Delta, A).
  $$

- $(\Delta, \Phi)$ is **saturated** if it has both properties.

- When is $(\Delta, \Phi)$ saturated?
Given a finite sequence $\Phi$ of $\Delta$-formulae let $\mathcal{X}_0 = \{ \psi \mid \vdash (\Delta) \Phi \Rightarrow \psi \}$. We define $(\Delta) \Phi \models \psi$ iff $\psi \in \mathcal{X}$, where $\psi \in \mathcal{X}$ is defined by the following structural recursion on the number of logical symbols in the $\Delta$-formula $\psi$. $\psi \in \mathcal{X}$ iff one of the following hold.

- $\psi$ is atomic, an equality or $\perp$ or $\top$ and $\psi \in \mathcal{X}_0$.
- $\psi \equiv (\psi_1 \land \psi_2)$ and $[\psi_1 \in \mathcal{X}$ and $\psi_2 \in \mathcal{X}]$.
- $\psi \equiv (\psi_1 \lor \psi_2)$ and $[\psi_1 \in \mathcal{X}$ or $\psi_2 \in \mathcal{X}]$.
- $\psi \equiv (\psi_1 \supset \psi_2) \in \mathcal{X}_0$ and $[\psi_1 \in \mathcal{X}$ implies $\psi_2 \in \mathcal{X}]$.
- $\psi \equiv (\forall x : A)\psi_0 \in \mathcal{X}_0$ and $[\psi_0[a/x] \in \mathcal{X}$ for all $a \in \text{Term}(\Delta, A)]$.
- $\psi \equiv (\exists x : A)\psi_0$ and $[\psi_0[a/x] \in \mathcal{X}$ for some $a \in \text{Term}(\Delta, A)]$. 
The saturation theorem, 1

Theorem: The following are equivalent:

1. \((\Delta, \Phi)\) is saturated.
2. \((\Delta) \Phi \vdash \phi\) for all \(\phi\) in \(\Phi\).
3. For every \(\Delta\)-formula \(\psi\)

\[\vdash (\Delta) \Phi \Rightarrow \psi \iff (\Delta) \Phi \vdash \psi.\]

Corollary: \((\Delta, \emptyset)\) is saturated

Proof of Theorem:

3 \(\Rightarrow\) 1&2 : Trivial.
1 \(\Rightarrow\) 3 : By Lemma 1.
2 \(\Rightarrow\) 3 : By Lemma 2.
Lemma 1:

1. \((\Delta) \Phi \vdash \psi\) implies \(\vdash (\Delta)\Phi \Rightarrow \psi\),

2. \(\vdash (\Delta)\Phi \Rightarrow \psi\) implies \((\Delta) \Phi \vdash \psi\), if \((\Delta, \Phi)\) is saturated.

Proof: By structural induction on \(\psi\).

- If \((\Delta, \Gamma)\) is a context let \(\tau \in Subst(\Delta; \Gamma)\) if \(\tau\) is a substitution \(\Delta \rightarrow (\Delta, \Gamma)\) of the form \([\iota\Delta \rightarrow \Delta, \rho]_{\Delta \rightarrow (\Delta, \Gamma)}\).

Lemma 2: If \(\vdash (\Delta, \Gamma) \Phi \Rightarrow \psi\) then, for all \(\tau \in Subst(\Delta; \Gamma)\),

\[(\Delta) \Phi \tau \vdash \phi \tau\] for all \(\phi\) in \(\Phi\) implies \((\Delta) \Phi \tau \vdash \psi \tau\).

Proof: By induction following the derivation of \((\Delta, \Gamma) \Phi \Rightarrow \psi\).
Types as propositions

- Think of a type $A$ as a proposition which is true if there is a term of type $A$.

- For each $A \in \text{Type}(\Delta)$, where $\Delta$ is a context, let $!A$ be the $\Delta$-formula $(\exists_\_ : A) \top$.

Theorem: If $A_1, \ldots, A_n, A \in \text{Type}(\Delta)$ and $x_1, \ldots, x_n$ are distinct variables, so that $(\Delta, x_1 : A_1, \ldots, x_n : A_n)$ is a context, then the following are equivalent:

1. $\vdash (\Delta) !A_1, \ldots, !A_n \Rightarrow !A$,

2. $\vdash (\Delta, x_1 : A_1, \ldots, x_n : A_n) \Rightarrow !A$,

3. there is a term in $\text{Term}((\Delta, x_1 : A_1, \ldots, x_n : A_n), A)$.

Proof: $3 \Rightarrow 2 \Leftrightarrow 1$ is trivial. $2 \Rightarrow 3$ uses Saturation.
We say that a type setup has \( \Pi \)-types if the standard formation, introduction, elimination and computation rules for \( \Pi \)-types are correct for the type setup; i.e. if \( \Gamma' \equiv (\Gamma, x : A) \) is a context then there are the following assignments:

\[
\begin{align*}
B \in Type(\Gamma') & \quad \mapsto (\Pi x : A) B \in Type(\Gamma), \\
b \in Term(\Gamma', B) & \quad \mapsto (\lambda x)b \in Term(\Gamma, (\Pi x : A) B), \\
f \in Term(\Gamma, (\Pi x : A) B) \quad & \quad \mapsto app(f, a) \in Term(\Gamma, B[a/x]), \\
a \in Term(\Gamma, A) \quad & \quad \mapsto \end{align*}
\]

such that if \( f \sim (\Pi x : A) B (\lambda x)b \) then \( app(f, a) \sim_{B[a/x]} b[a/x] \).
These must commute with substitution; i.e. for each
\( \sigma : \Delta \rightarrow \Gamma \),

\[
((\Pi x : A)B)\sigma \sim_{\Delta} (\Pi x : A\sigma)B\sigma',
\]
\[
((\lambda x)b)\sigma \sim_{\Delta} (\lambda x)b\sigma',
\]
\[
app(f, a)\sigma \sim_{\Delta} app(f\sigma, a\sigma),
\]

where \( \sigma' \equiv [\sigma, x := x]_{\Delta \rightarrow \Gamma'} : \Delta \rightarrow \Gamma' \).

Also, if \( y \) is \( \Gamma \)-free then

\[
(\Pi x : A)B \sim_{\Gamma} (\Pi y : A)B[y/x] \text{ and } (\lambda x)b \sim_{\Gamma} (\lambda y)b[y/x].
\]

The requirement that the type setup has other forms of type can be explained in a similar way.
Propositions as types, 1

- We assume given a type setup with predicate signature that has the forms of type \((\Pi x : A) B, (\Sigma x : A) B\), with the defined forms \(A \to B\) and \(A \times B\), the forms of type \(A_1 + A_2, N_k(k = 0, 1, \ldots), I(A, a_1, a_2)\) and also has associated with each predicate symbol \(P\), of arity the context \(\Delta\), a type \(P^\# \in Type(\Delta)\).
- Then the propositions-as-types interpretation recursively associates with each \(\Gamma\)-formula \(\phi\) a type \(Pr(\phi) \in Type(\Gamma)\) using the following rules.
  - If \(\phi\) is the atomic \(\Gamma\)-formula \(P(b_1, \ldots, b_m)\) then \(Pr(\phi)\) is the type \(P^#[y_1 := b_1, \ldots, y_m := b_m]_{\Gamma \to \Delta} \in Type(\Gamma)\).
  - If \(\phi\) is \((a_1 =_A a_2)\) then \(Pr(\phi)\) is the type \(I(A, a_1, a_2) \in Type(\Gamma)\).
  - If \(\phi\) is \(\bot\) or \(\top\) then \(Pr(\phi)\) is \(N_0\) or \(N_1\) respectively.
Propositions as types, 2

- If $\phi$ is $(\psi_1 \Box \psi_2)$, where $\Box$ is one of $\land, \lor, \supset$ then $Pr(\phi)$ is $(Pr(\psi_1) \Box' Pr(\psi_2))$ where $\Box'$ is the corresponding one of $\times, +, \to$.

- If $\phi$ is $(\nabla x : A)\psi_0$ where $\nabla$ is one of $\forall, \exists$ then $Pr(\phi)$ is $(\nabla' x : A)Pr(\psi_0)$ where $\nabla'$ is the corresponding one of $\Pi, \Sigma$.

**Proposition:** The interpretation is sound; i.e. if $\vdash (\Delta) \phi_1, \ldots, \phi_k \Rightarrow \phi$ then there is a term in

$$\text{Term}((\Delta, x_1 : Pr(\phi_1), \ldots, x_k : Pr(\phi_k)), Pr(\phi)),$$

where $x_1, \ldots, x_k$ are distinct $\Delta$-free variables.

- But the interpretation is not complete as the type theoretic axiom of choice holds; i.e.
If $\Gamma$ is a context, $x, y$ are distinct $\Gamma$-free variables, $A \in Type(\Gamma)$, $B \in Type((\Gamma, x : A))$ and $\theta$ is a $(\Gamma, x : A, y : B)$-formula then let $ac(\Gamma, x : A, y : B, \theta)$ be the sequent

$$(\Gamma)(\forall x : A)(\exists y : B)\theta \Rightarrow (\exists z : (\Pi x : A)B)(\forall x : A)\theta[app(z, x)/y],$$

and let $AC$ be the set of all such sequents.

**Proposition:** If $\vdash ac(\Gamma, x : A, y : B, \theta)$ then there is a term in

$$Term((\Gamma, _ : Pr(((\forall x : A)(\exists y : B)\theta)), Pr((\exists z : (\Pi x : A)B)(\forall x : A)\theta[app(z, x)/y])).$$
Propositions as types, 4

• If $\Sigma$ is a set of sequents we write $\Sigma \vdash (\Gamma) \Phi \Rightarrow \phi$ if the sequent $(\Gamma) \Phi \Rightarrow \phi$ can be derived using the rules of inference for intuitionistic predicate logic and the sequents in $\Sigma$ as additional axioms.

• Let $PaT$ be the set of all sequents having one of the forms

$$(\Gamma) \phi \Rightarrow !Pr(\phi) \text{ or } (\Gamma) !Pr(\phi) \Rightarrow \phi.$$

• Let $PaT_{atomic}$ be the set of all those sequents in $PaT$ where $\phi$ is an atomic formula $P(b_1, \ldots, b_m)$. 
Theorem: The following are equivalent

1. There is a term in

\[ \text{Term}((\Delta, x_1 : Pr(\phi_1), \ldots, x_k : Pr(\phi_k)), Pr(\phi)), \]

2. \( PaT \vdash (\Delta) \phi_1, \ldots, \phi_k \Rightarrow \phi, \)

3. \( AC \cup PaT_{atomic} \cup \Sigma \vdash (\Delta) \phi_1, \ldots, \phi_k \Rightarrow \phi, \)

where \( \Sigma \) is the set of sequents having one of the forms:

- \( (\Gamma) \Rightarrow (\forall _{-} : N_0) \bot, \)
- \( (\Gamma) \Rightarrow (\forall _{-} : A + B) (!A \lor !B), \)
- \( (\Gamma) \Rightarrow (\forall _{-} : I(A, a_1, a_2)) (a_1 =_A a_2). \)

Here \( A, B \in Type(\Gamma) \) and \( a_1, a_2 \in Term(\Gamma, A). \)