Lecture 3: Higher-order logic

Łukasz Czajka

Higher-order logic

- In first-order logic there are function and predicate symbols, but no quantification over them.

Higher-order logic

- In first-order logic there are function and predicate symbols, but no quantification over them.
- $\forall x R(f(x))$ is a first-order formula.

Higher-order logic

- In first-order logic there are function and predicate symbols, but no quantification over them.
- $\forall x R(f(x))$ is a first-order formula.
- $\exists R \forall f \forall x R(f(x))$ is not.

Higher-order logic

- In first-order logic there are function and predicate symbols, but no quantification over them.
- $\forall x R(f(x))$ is a first-order formula.
- $\exists R \forall f \forall x R(f(x))$ is not.
- Second-order logic: quantification over first-order predicates is allowed.

Higher-order logic

- In first-order logic there are function and predicate symbols, but no quantification over them.
- $\forall x R(f(x))$ is a first-order formula.
- $\exists R \forall f \forall x R(f(x))$ is not.
- Second-order logic: quantification over first-order predicates is allowed.
- Second-order predicates: e.g. $Q(R) := \forall x Rx$.

Higher-order logic

- In first-order logic there are function and predicate symbols, but no quantification over them.
- $\forall x R(f(x))$ is a first-order formula.
- $\exists R \forall f \forall x R(f(x))$ is not.
- Second-order logic: quantification over first-order predicates is allowed.
- Second-order predicates: e.g. $Q(R) := \forall x Rx$.
- Third-order logic: quantification over first- and second-order predicates allowed.

Higher-order logic

- In first-order logic there are function and predicate symbols, but no quantification over them.
- $\forall x R(f(x))$ is a first-order formula.
- $\exists R \forall f \forall x R(f(x))$ is not.
- Second-order logic: quantification over first-order predicates is allowed.
- Second-order predicates: e.g. $Q(R) := \forall x R x$.
- Third-order logic: quantification over first- and second-order predicates allowed.
- Higher-order logic: why not go all the way up?

Higher-order logic: object types

Definition

The object types (or domains) A, B, C are given by

$$\mathcal{D} \quad ::= \quad \mathcal{B} \mid \texttt{Prop} \mid \mathcal{D} \to \mathcal{D}$$

where $\mathcal{B}$ is a fixed set of basic domains.

Higher-order logic: object types

Definition

The object types (or domains) A, B, C are given by

$$\mathcal{D} \quad ::= \quad \mathcal{B} \mid \texttt{Prop} \mid \mathcal{D} \to \mathcal{D}$$

where $\mathcal{B}$ is a fixed set of basic domains.

Examples (assuming $\texttt{nat}, \texttt{bool} \in \mathcal{B}$):

- first-order predicates: $\texttt{nat} \to \texttt{Prop}$, $\texttt{bool} \to \texttt{nat} \to \texttt{Prop}$;

Higher-order logic: object types

Definition

The object types (or domains) A, B, C are given by

$$\mathcal{D} \quad ::= \quad \mathcal{B} \mid \texttt{Prop} \mid \mathcal{D} \to \mathcal{D}$$

where $\mathcal{B}$ is a fixed set of basic domains.

Examples (assuming $\texttt{nat}, \texttt{bool} \in \mathcal{B}$):

- first-order predicates: $\texttt{nat} \to \texttt{Prop}$, $\texttt{bool} \to \texttt{nat} \to \texttt{Prop}$;
- first-order functions: $\texttt{bool} \to \texttt{bool}$, $\texttt{nat} \to \texttt{bool} \to \texttt{nat}$;

Higher-order logic: object types

Definition

The object types (or domains) A, B, C are given by

$$\mathcal{D} \quad ::= \quad \mathcal{B} \mid \texttt{Prop} \mid \mathcal{D} \to \mathcal{D}$$

where $\mathcal{B}$ is a fixed set of basic domains.

Examples (assuming $\texttt{nat}, \texttt{bool} \in \mathcal{B}$):

- first-order predicates: $\texttt{nat} \to \texttt{Prop}$, $\texttt{bool} \to \texttt{nat} \to \texttt{Prop}$;
- first-order functions: $\texttt{bool} \to \texttt{bool}$, $\texttt{nat} \to \texttt{bool} \to \texttt{nat}$;
- higher-order predicates: $(\texttt{nat} \to \texttt{Prop}) \to \texttt{Prop}$;
  $(\texttt{Prop} \to \texttt{Prop}) \to \texttt{Prop}$;

Higher-order logic: object types

Definition

The object types (or domains) A, B, C are given by

$$\mathcal{D} \quad ::= \quad \mathcal{B} \mid \texttt{Prop} \mid \mathcal{D} \to \mathcal{D}$$

where $\mathcal{B}$ is a fixed set of basic domains.

Examples (assuming $\texttt{nat}, \texttt{bool} \in \mathcal{B}$):

- first-order predicates: $\texttt{nat} \to \texttt{Prop}$, $\texttt{bool} \to \texttt{nat} \to \texttt{Prop}$;
- first-order functions: $\texttt{bool} \to \texttt{bool}$, $\texttt{nat} \to \texttt{bool} \to \texttt{nat}$;
- higher-order predicates: $(\texttt{nat} \to \texttt{Prop}) \to \texttt{Prop}$; $(\texttt{Prop} \to \texttt{Prop}) \to \texttt{Prop}$;
- higher-order functions: $(\texttt{nat} \to \texttt{bool}) \to \texttt{nat}$; $((\texttt{bool} \to \texttt{bool}) \to \texttt{bool}) \to \texttt{nat}$;

Higher-order logic: object types

Definition

The object types (or domains) A, B, C are given by

$$\mathcal{D} \quad ::= \quad \mathcal{B} \mid \texttt{Prop} \mid \mathcal{D} \to \mathcal{D}$$

where $\mathcal{B}$ is a fixed set of basic domains.

Examples (assuming $\texttt{nat}, \texttt{bool} \in \mathcal{B}$):

- first-order predicates: $\texttt{nat} \to \texttt{Prop}$, $\texttt{bool} \to \texttt{nat} \to \texttt{Prop}$;
- first-order functions: $\texttt{bool} \to \texttt{bool}$, $\texttt{nat} \to \texttt{bool} \to \texttt{nat}$;
- higher-order predicates: $(\texttt{nat} \to \texttt{Prop}) \to \texttt{Prop}$; $(\texttt{Prop} \to \texttt{Prop}) \to \texttt{Prop}$;
- higher-order functions: $(\texttt{nat} \to \texttt{bool}) \to \texttt{nat}$; $((\texttt{bool} \to \texttt{bool}) \to \texttt{bool}) \to \texttt{nat}$;
- functions with predicate arguments: $(\texttt{nat} \to \texttt{Prop}) \to \texttt{nat}$; $\texttt{Prop} \to \texttt{bool}$.

Higher-order logic: object terms

- An object term t is an object variable x, y, z, an application $t_1 t_2$, an abstraction $\lambda x : A.t'$, an implication $t_1 \Rightarrow t_2$, or a universal quantification $\forall x : A.t$.

Higher-order logic: object terms

- An object term t is an object variable x, y, z, an application $t_1 t_2$, an abstraction $\lambda x : A.t'$, an implication $t_1 \Rightarrow t_2$, or a universal quantification $\forall x : A.t$.
- An object context Γ is a finite set of unique declarations of the form $x : A$.

Higher-order logic: object terms

- An object term t is an object variable x, y, z, an application $t_1 t_2$, an abstraction $\lambda x : A.t'$, an implication $t_1 \Rightarrow t_2$, or a universal quantification $\forall x : A.t$.
- An object context Γ is a finite set of unique declarations of the form $x : A$. We write $\Gamma, x : A$ for $\Gamma \cup \{x : A\}$.

Higher-order logic: object terms

- An <u>object term</u> t is an object variable x, y, z, an application $t_1 t_2$, an abstraction $\lambda x : A.t'$, an implication $t_1 \Rightarrow t_2$, or a universal quantification $\forall x : A.t$.
- An <u>object context</u> Γ is a finite set of unique declarations of the form $x : A$. We write $\Gamma, x : A$ for $\Gamma \cup \{x : A\}$.
- An object term t has type $A \in \mathcal{D}$ in Γ if $\Gamma \vdash t : A$ can be derived using the following rules.

$$\frac{}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B} \qquad \frac{\Gamma \vdash t_1 : A \to B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 t_2 : B}$$

$$\frac{\Gamma \vdash \varphi : \texttt{Prop} \quad \Gamma \vdash \psi : \texttt{Prop}}{\Gamma \vdash \varphi \Rightarrow \psi : \texttt{Prop}} \qquad \frac{\Gamma, x : A \vdash \varphi : \texttt{Prop}}{\Gamma \vdash \forall x : A.\varphi : \texttt{Prop}}$$

Higher-order logic: object terms

- An <u>object term</u> t is an object variable x, y, z, an application $t_1 t_2$, an abstraction $\lambda x : A.t'$, an implication $t_1 \Rightarrow t_2$, or a universal quantification $\forall x : A.t$.
- An <u>object context</u> Γ is a finite set of unique declarations of the form $x : A$. We write $\Gamma, x : A$ for $\Gamma \cup \{x : A\}$.
- An object term t has type $A \in \mathcal{D}$ in Γ if $\Gamma \vdash t : A$ can be derived using the following rules.

$$\frac{}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B} \qquad \frac{\Gamma \vdash t_1 : A \to B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 t_2 : B}$$

$$\frac{\Gamma \vdash \varphi : \mathtt{Prop} \quad \Gamma \vdash \psi : \mathtt{Prop}}{\Gamma \vdash \varphi \Rightarrow \psi : \mathtt{Prop}} \qquad \frac{\Gamma, x : A \vdash \varphi : \mathtt{Prop}}{\Gamma \vdash \forall x : A.\varphi : \mathtt{Prop}}$$

We consider only well-typed object terms.

Higher-order logic: object terms

- An <u>object term</u> t is an object variable x, y, z, an application $t_1 t_2$, an abstraction $\lambda x : A.t'$, an implication $t_1 \Rightarrow t_2$, or a universal quantification $\forall x : A.t$.
- An <u>object context</u> Γ is a finite set of unique declarations of the form $x : A$. We write $\Gamma, x : A$ for $\Gamma \cup \{x : A\}$.
- An object term t has type $A \in \mathcal{D}$ in Γ if $\Gamma \vdash t : A$ can be derived using the following rules.

$$\frac{}{\Gamma, x : A \vdash x : A}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \to B} \qquad \frac{\Gamma \vdash t_1 : A \to B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 t_2 : B}$$

$$\frac{\Gamma \vdash \varphi : \texttt{Prop} \quad \Gamma \vdash \psi : \texttt{Prop}}{\Gamma \vdash \varphi \Rightarrow \psi : \texttt{Prop}} \qquad \frac{\Gamma, x : A \vdash \varphi : \texttt{Prop}}{\Gamma \vdash \forall x : A.\varphi : \texttt{Prop}}$$

  We consider only well-typed object terms.
- A <u>formula</u> φ, ψ is an object term of type `Prop`.

Higher-order logic: object terms

Examples:

- $f : A \to A \vdash \lambda x : A \lambda y : B.f(fx) : A \to B \to A$;

Higher-order logic: object terms

Examples:

- $f : A \to A \vdash \lambda x : A \lambda y : B.f(fx) : A \to B \to A$;
- $f : A \to A \vdash \lambda x : A.(\lambda g : A \to A.g(fx))f : A \to A$;

Higher-order logic: object terms

Examples:

- $f : A \to A \vdash \lambda x : A \lambda y : B.f(fx) : A \to B \to A$;
- $f : A \to A \vdash \lambda x : A.(\lambda g : A \to A.g(fx))f : A \to A$;
- $f : A \to A, R : A \to \mathtt{Prop} \vdash \forall x : A.R(fx) \Rightarrow R(f(fx)) : \mathtt{Prop}$;

Higher-order logic: object terms

Examples:

- $f : A \to A \vdash \lambda x : A \lambda y : B.f(fx) : A \to B \to A$;
- $f : A \to A \vdash \lambda x : A.(\lambda g : A \to A.g(fx))f : A \to A$;
- $f : A \to A, R : A \to \texttt{Prop} \vdash \forall x : A.R(fx) \Rightarrow R(f(fx)) : \texttt{Prop}$;
- $R : (A \to B) \to \texttt{Prop} \vdash \forall f : A \to B.Rf \Rightarrow R(\lambda x : A.fx) : \texttt{Prop}$;

Higher-order logic: object terms

Examples:

- $f : A \to A \vdash \lambda x : A\lambda y : B.f(fx) : A \to B \to A$;
- $f : A \to A \vdash \lambda x : A.(\lambda g : A \to A.g(fx))f : A \to A$;
- $f : A \to A, R : A \to \texttt{Prop} \vdash \forall x : A.R(fx) \Rightarrow R(f(fx)) : \texttt{Prop}$;
- $R : (A \to B) \to \texttt{Prop} \vdash \forall f : A \to B.Rf \Rightarrow R(\lambda x : A.fx) : \texttt{Prop}$;
- $x : A, y : A \vdash \forall R : A \to \texttt{Prop}.Rx \Rightarrow Ry : \texttt{Prop}$.

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \to_\beta \lambda x : A.x$.

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \rightarrow_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \rightarrow_\beta \lambda x : A.x$.
- The relation $=_\beta$ of β-equality is the least equivalence relation including β-reduction.

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \to_\beta \lambda x : A.x$.
- The relation $=_\beta$ of β-equality is the least equivalence relation including β-reduction.
- η-reduction "implements" syntactic extensionality of functions:

$$(\lambda x : A.tx) \to_\eta t \qquad \text{if } x \notin \mathrm{FV}(t)$$

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \to_\beta \lambda x : A.x$.
- The relation $=_\beta$ of β-equality is the least equivalence relation including β-reduction.
- η-reduction "implements" syntactic extensionality of functions:

$$(\lambda x : A.tx) \to_\eta t \qquad \text{if } x \notin \mathrm{FV}(t)$$

- The relation $=_{\beta\eta}$ of $\beta\eta$-equality is the least equivalence relation including β- and η-reduction.

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \to_\beta \lambda x : A.x$.
- The relation $=_\beta$ of β-equality is the least equivalence relation including β-reduction.
- η-reduction "implements" <u>syntactic</u> extensionality of functions:

$$(\lambda x : A.tx) \to_\eta t \qquad\qquad \text{if } x \notin \mathrm{FV}(t)$$

- The relation $=_{\beta\eta}$ of $\beta\eta$-equality is the least equivalence relation including β- and η-reduction.
- The relation $\equiv$ of <u>definitional equality</u> (also called <u>computational equality</u>) is defined to be $\beta\eta$-equality.

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \to_\beta \lambda x : A.x$.
- The relation $=_\beta$ of β-equality is the least equivalence relation including β-reduction.
- η-reduction "implements" <u>syntactic</u> extensionality of functions:

$$(\lambda x : A.tx) \to_\eta t \qquad \text{if } x \notin \mathrm{FV}(t)$$

- The relation $=_{\beta\eta}$ of $\beta\eta$-equality is the least equivalence relation including β- and η-reduction.
- The relation $\equiv$ of <u>definitional equality</u> (also called <u>computational equality</u>) is defined to be $\beta\eta$-equality.
  - Definitional equality is different for different systems.

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \to_\beta \lambda x : A.x$.
- The relation $=_\beta$ of β-equality is the least equivalence relation including β-reduction.
- η-reduction "implements" <u>syntactic</u> extensionality of functions:

$$(\lambda x : A.tx) \to_\eta t \qquad\qquad \text{if } x \notin \mathrm{FV}(t)$$

- The relation $=_{\beta\eta}$ of $\beta\eta$-equality is the least equivalence relation including β- and η-reduction.
- The relation $\equiv$ of <u>definitional equality</u> (also called <u>computational equality</u>) is defined to be $\beta\eta$-equality.
  - Definitional equality is different for different systems.
  - Definitional equality is an equivalence relation compatible with the structure of terms.

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \rightarrow_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \rightarrow_\beta \lambda x : A.x$.
- The relation $=_\beta$ of β-equality is the least equivalence relation including β-reduction.
- η-reduction "implements" syntactic extensionality of functions:

$$(\lambda x : A.tx) \rightarrow_\eta t \qquad\qquad \text{if } x \notin \mathrm{FV}(t)$$

- The relation $=_{\beta\eta}$ of $\beta\eta$-equality is the least equivalence relation including β- and η-reduction.
- The relation $\equiv$ of definitional equality (also called computational equality) is defined to be $\beta\eta$-equality.
  - Definitional equality is different for different systems.
  - Definitional equality is an equivalence relation compatible with the structure of terms.
    - E.g. if $t \equiv t'$ then $\lambda x : A.ftx \equiv \lambda x : A.ft'x$.

Higher-order logic: computation

- β-reduction "implements" applying a function to an argument:

$$(\lambda x : A.t)t' \to_\beta t[t'/x]$$

- Example: $\lambda x : A.(\lambda y : A.y)x \to_\beta \lambda x : A.x$.
- The relation $=_\beta$ of β-equality is the least equivalence relation including β-reduction.
- η-reduction "implements" syntactic extensionality of functions:

$$(\lambda x : A.tx) \to_\eta t \qquad \text{if } x \notin \mathrm{FV}(t)$$

- The relation $=_{\beta\eta}$ of $\beta\eta$-equality is the least equivalence relation including β- and η-reduction.
- The relation $\equiv$ of definitional equality (also called computational equality) is defined to be $\beta\eta$-equality.
  - Definitional equality is different for different systems.
  - Definitional equality is an equivalence relation compatible with the structure of terms.
    - E.g. if $t \equiv t'$ then $\lambda x : A.ftx \equiv \lambda x : A.ft'x$.
  - Definitional equality is decidable.

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Proof.

Let $f, g : A \to B$ in Γ. Assume $ft \equiv gt$ for all t such that $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$.

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Proof.

Let $f, g : A \to B$ in Γ. Assume $ft \equiv gt$ for all t such that $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$. Take a fresh variable $x \notin \mathrm{FV}(f, g, \Gamma)$ and let $\Gamma' = \Gamma, x : A$.

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Proof.

Let $f, g : A \to B$ in Γ. Assume $ft \equiv gt$ for all t such that $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$. Take a fresh variable $x \notin \mathrm{FV}(f, g, \Gamma)$ and let $\Gamma' = \Gamma, x : A$. Then $\Gamma' \vdash x : A$, so $fx \equiv gx$.

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Proof.

Let $f, g : A \to B$ in Γ. Assume $ft \equiv gt$ for all t such that $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$. Take a fresh variable $x \notin \mathrm{FV}(f, g, \Gamma)$ and let $\Gamma' = \Gamma, x : A$. Then $\Gamma' \vdash x : A$, so $fx \equiv gx$. Hence also $\lambda x : A.fx \equiv \lambda x : A.gx$.

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Proof.

Let $f, g : A \to B$ in Γ. Assume $ft \equiv gt$ for all t such that $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$. Take a fresh variable $x \notin \mathrm{FV}(f, g, \Gamma)$ and let $\Gamma' = \Gamma, x : A$. Then $\Gamma' \vdash x : A$, so $fx \equiv gx$. Hence also $\lambda x : A.fx \equiv \lambda x : A.gx$. But $\lambda x : A.fx \to_\eta f$ and $\lambda x : A.gx \to_\eta g$ (recall $x \notin \mathrm{FV}(f, g)$).

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Proof.

Let $f, g : A \to B$ in Γ. Assume $ft \equiv gt$ for all t such that $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$. Take a fresh variable $x \notin \mathrm{FV}(f, g, \Gamma)$ and let $\Gamma' = \Gamma, x : A$. Then $\Gamma' \vdash x : A$, so $fx \equiv gx$. Hence also $\lambda x : A.fx \equiv \lambda x : A.gx$. But $\lambda x : A.fx \to_\eta f$ and $\lambda x : A.gx \to_\eta g$ (recall $x \notin \mathrm{FV}(f, g)$). Then $\lambda x : A.fx \equiv f$ and $\lambda x : A.gx \equiv g$ because $\equiv$ includes η-reduction.

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Proof.

Let $f, g : A \to B$ in Γ. Assume $ft \equiv gt$ for all t such that $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$. Take a fresh variable $x \notin \mathrm{FV}(f, g, \Gamma)$ and let $\Gamma' = \Gamma, x : A$. Then $\Gamma' \vdash x : A$, so $fx \equiv gx$. Hence also $\lambda x : A.fx \equiv \lambda x : A.gx$. But $\lambda x : A.fx \to_\eta f$ and $\lambda x : A.gx \to_\eta g$ (recall $x \notin \mathrm{FV}(f, g)$). Then $\lambda x : A.fx \equiv f$ and $\lambda x : A.gx \equiv g$ because $\equiv$ includes η-reduction. This implies $f \equiv g$. $\square$

Syntactic functional extensionality and η-reduction

Definition

Syntactic functional extensionality for Γ, A, B is the following (meta) statement:

- for any f, g with $\Gamma \vdash f : A \to B$ and $\Gamma \vdash g : A \to B$, if $ft \equiv gt$ for every t with $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$, then $f \equiv g$.

Fact

If definitional equality includes η-reduction then syntactic functional extensionality holds.

Proof.

Let $f, g : A \to B$ in Γ. Assume $ft \equiv gt$ for all t such that $\Gamma' \vdash t : A$ for some $\Gamma' \supseteq \Gamma$. Take a fresh variable $x \notin \mathrm{FV}(f, g, \Gamma)$ and let $\Gamma' = \Gamma, x : A$. Then $\Gamma' \vdash x : A$, so $fx \equiv gx$. Hence also $\lambda x : A.fx \equiv \lambda x : A.gx$. But $\lambda x : A.fx \to_\eta f$ and $\lambda x : A.gx \to_\eta g$ (recall $x \notin \mathrm{FV}(f, g)$). Then $\lambda x : A.fx \equiv f$ and $\lambda x : A.gx \equiv g$ because $\equiv$ includes η-reduction. This implies $f \equiv g$. $\square$

Trivially, if syntactic functional extensionality holds and definitional equality includes β-reduction, then it also includes η-reduction (exercise).

Intermission: the simply-typed lambda-calculus

Simple types: $\mathcal{T} ::= \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$ where $\mathcal{B}$ is a fixed finite set of type constants.

Intermission: the simply-typed lambda-calculus

Simple types: $\mathcal{T} ::= \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$ where $\mathcal{B}$ is a fixed finite set of type constants.

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \tau}$$

$$\frac{\Gamma \cup \{x : \tau\} \vdash t : \tau}{\Gamma \vdash \lambda x : \tau.t : \tau \to \sigma} \qquad \frac{\Gamma \vdash t_1 : \tau \to \sigma \quad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 t_2 : \sigma}$$

Intermission: the simply-typed lambda-calculus

Simple types: $\mathcal{T} ::= \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$ where $\mathcal{B}$ is a fixed finite set of type constants.

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \tau}$$

$$\frac{\Gamma \cup \{x : \tau\} \vdash t : \tau}{\Gamma \vdash \lambda x : \tau . t : \tau \to \sigma} \qquad \frac{\Gamma \vdash t_1 : \tau \to \sigma \quad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 t_2 : \sigma}$$

- β-reduction: $(\lambda x : \tau . t)t' \to_\beta t[t'/x]$.

Intermission: the simply-typed lambda-calculus

Simple types: $\mathcal{T} ::= \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$ where $\mathcal{B}$ is a fixed finite set of type constants.

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \tau}$$

$$\frac{\Gamma \cup \{x : \tau\} \vdash t : \tau}{\Gamma \vdash \lambda x : \tau . t : \tau \to \sigma} \qquad \frac{\Gamma \vdash t_1 : \tau \to \sigma \quad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 t_2 : \sigma}$$

- β-reduction: $(\lambda x : \tau . t)t' \to_\beta t[t'/x]$.
- Subject reduction theorem: if $\Gamma \vdash t : \tau$ and $t \to_\beta^* t'$ then $\Gamma \vdash t' : \tau$.

Intermission: the simply-typed lambda-calculus

Simple types: $\mathcal{T} ::= \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$ where $\mathcal{B}$ is a fixed finite set of type constants.

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \tau}$$

$$\frac{\Gamma \cup \{x : \tau\} \vdash t : \tau}{\Gamma \vdash \lambda x : \tau . t : \tau \to \sigma} \qquad \frac{\Gamma \vdash t_1 : \tau \to \sigma \quad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 t_2 : \sigma}$$

- β-reduction: $(\lambda x : \tau . t)t' \to_\beta t[t'/x]$.
- Subject reduction theorem: if $\Gamma \vdash t : \tau$ and $t \to_\beta^* t'$ then $\Gamma \vdash t' : \tau$.
- Strong normalisation theorem: if $\Gamma \vdash t : \tau$ then every reduction sequence starting from t ends in a β-normal form (i.e., in a term with no β-redexes).

Intermission: the simply-typed lambda-calculus

Simple types: $\mathcal{T} ::= \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$ where $\mathcal{B}$ is a fixed finite set of type constants.

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \tau}$$

$$\frac{\Gamma \cup \{x : \tau\} \vdash t : \tau}{\Gamma \vdash \lambda x : \tau . t : \tau \to \sigma} \qquad \frac{\Gamma \vdash t_1 : \tau \to \sigma \quad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 t_2 : \sigma}$$

- β-reduction: $(\lambda x : \tau . t)t' \to_\beta t[t'/x]$.
- Subject reduction theorem: if $\Gamma \vdash t : \tau$ and $t \to_\beta^* t'$ then $\Gamma \vdash t' : \tau$.
- Strong normalisation theorem: if $\Gamma \vdash t : \tau$ then every reduction sequence starting from t ends in a β-normal form (i.e., in a term with no β-redexes).
- Uniqueness of normal forms: if t_1, t_2 are in β-normal form and $t_1 =_\beta t_2$, then $t_1 = t_2$.

Intermission: the simply-typed lambda-calculus

Simple types: $\mathcal{T} ::= \mathcal{B} \mid \mathcal{T} \to \mathcal{T}$ where $\mathcal{B}$ is a fixed finite set of type constants.

$$\frac{}{\Gamma \cup \{x : \tau\} \vdash x : \tau}$$

$$\frac{\Gamma \cup \{x : \tau\} \vdash t : \tau}{\Gamma \vdash \lambda x : \tau . t : \tau \to \sigma} \qquad \frac{\Gamma \vdash t_1 : \tau \to \sigma \quad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash t_1 t_2 : \sigma}$$

- β-reduction: $(\lambda x : \tau . t) t' \to_\beta t[t'/x]$.
- Subject reduction theorem: if $\Gamma \vdash t : \tau$ and $t \to_\beta^* t'$ then $\Gamma \vdash t' : \tau$.
- Strong normalisation theorem: if $\Gamma \vdash t : \tau$ then every reduction sequence starting from t ends in a β-normal form (i.e., in a term with no β-redexes).
- Uniqueness of normal forms: if t_1, t_2 are in β-normal form and $t_1 =_\beta t_2$, then $t_1 = t_2$.
- Exercise: β-equality on simply-typed terms is decidable.

Higher-order logic: proof terms

- A proof term M, N is a proof variable X, Y, Z, a lambda abstraction $\lambda X : \varphi . M$ or $\lambda x : A.M$, or an application $M_1 M_2$ or Mt.

Higher-order logic: proof terms

- A proof term M, N is a proof variable X, Y, Z, a lambda abstraction $\lambda X : \varphi.M$ or $\lambda x : A.M$, or an application $M_1 M_2$ or Mt.
- A proof context Δ is a finite set of unique declarations of the form $X : \varphi$.

Higher-order logic: proof terms

- A proof term M, N is a proof variable X, Y, Z, a lambda abstraction $\lambda X : \varphi.M$ or $\lambda x : A.M$, or an application $M_1 M_2$ or Mt.
- A proof context Δ is a finite set of unique declarations of the form $X : \varphi$.
- A judgement has the form $\Gamma; \Delta \vdash M : \varphi$.

Intermission: derivation rules

$$\frac{J_1 \quad \ldots \quad J_n}{J}\ S$$

- If we have derived the judgements $J_1, \ldots, J_n$ and the side condition S holds, then we can derive the judgement J.

Intermission: derivation rules

$$\frac{J_1 \quad \dots \quad J_n}{J} \; S$$

- If we have derived the judgements $J_1, \dots, J_n$ and the side condition S holds, then we can derive the judgement J.
- Sometimes we write the side condition(s) above the line together with the judgements $J_1, \dots, J_n$.

Intermission: derivation trees

$$\dfrac{\dfrac{\dfrac{}{J_3} \quad \dfrac{\dfrac{}{J_5}}{J_4}}{J_1} \qquad \dfrac{}{J_2}}{J}$$

- To derive a judgement J we build a derivation tree using the derivation rules: each node is a valid application of a derivation rule.

Intermission: derivation trees

$$\dfrac{\dfrac{\dfrac{}{J_3} \qquad \dfrac{\dfrac{}{J_5}}{J_4}}{J_1} \qquad \dfrac{}{J_2}}{J}$$

- To derive a judgement J we build a derivation tree using the derivation rules: each node is a valid application of a derivation rule.
- At the leaves of the tree we need rules with no judgements above the line.

Intuitionistic higher-order logic: rules

$$\frac{}{\Gamma; \Delta, X : \varphi \vdash X : \varphi} \text{ (Ax)}$$

$$\frac{\Gamma; \Delta, X : \varphi \vdash M : \psi}{\Gamma; \Delta \vdash \lambda X : \varphi . M : \varphi \Rightarrow \psi} \text{ (}{\Rightarrow}\text{I)} \qquad \frac{\Gamma; \Delta \vdash M : \varphi \Rightarrow \psi \quad \Gamma; \Delta \vdash N : \varphi}{\Gamma; \Delta \vdash MN : \psi} \text{ (}{\Rightarrow}\text{E)}$$

$$\frac{\Gamma, x : A; \Delta \vdash M : \varphi \quad x \notin \mathrm{FV}(\Delta)}{\Gamma; \Delta \vdash \lambda x : A.M : \forall x : A.\varphi} \text{ (}\forall\text{I)} \qquad \frac{\Gamma; \Delta \vdash M : \forall x : A.\varphi \quad \Gamma \vdash t : A}{\Gamma; \Delta \vdash Mt : \varphi[t/x]} \text{ (}\forall\text{E)}$$

$$\frac{\Gamma; \Delta \vdash M : \varphi \quad \Gamma \vdash \psi : \texttt{Prop} \quad \varphi \equiv \psi}{\Gamma; \Delta \vdash M : \psi} \text{ (conv)}$$

Intuitionistic higher-order logic: example derivation

$$\dfrac{\dfrac{\dfrac{}{\Gamma;\Delta \vdash X_1 : \forall x : A.Px \Rightarrow Q} \quad \dfrac{}{\Gamma \vdash x : A}}{\Gamma;\Delta \vdash X_1 x : Px \Rightarrow Q} \qquad \dfrac{\dfrac{}{\Gamma;\Delta \vdash X_2 : \forall x : A.Px} \quad \dfrac{}{\Gamma \vdash x : A}}{\Gamma;\Delta \vdash X_2 x : Px}}{\Gamma;\Delta \vdash X_1 x (X_2 x) : Q}$$

- $\Gamma \quad = \quad P : A \to \texttt{Prop}, \quad Q : \texttt{Prop}, \quad x : A.$
- $\Delta \quad = \quad X_1 : \forall x : A.Px \Rightarrow Q, \quad X_2 : \forall x : A.Px.$

Higher-order logic: expressiveness

- A second-order predicate expressing the transitivity of a binary relation:

$$\texttt{Trans} := \lambda R : A \to A \to \texttt{Prop}. \forall xyz : A. Rxy \Rightarrow Ryz \Rightarrow Rxz$$

Higher-order logic: expressiveness

- A second-order predicate expressing the transitivity of a binary relation:

$$\texttt{Trans} := \lambda R : A \to A \to \texttt{Prop}.\forall xyz : A.Rxy \Rightarrow Ryz \Rightarrow Rxz$$

- A binary relation R is included in S if for all x, y, Rxy implies Sxy:

$$\texttt{Subrel} := \lambda RS : A \to A \to \texttt{Prop}.\forall xy : A.Rxy \Rightarrow Sxy$$

Higher-order logic: expressiveness

- A second-order predicate expressing the transitivity of a binary relation:

$$\texttt{Trans} := \lambda R : A \to A \to \texttt{Prop}.\forall xyz : A.Rxy \Rightarrow Ryz \Rightarrow Rxz$$

- A binary relation R is included in S if for all x, y, Rxy implies Sxy:

$$\texttt{Subrel} := \lambda RS : A \to A \to \texttt{Prop}.\forall xy : A.Rxy \Rightarrow Sxy$$

- The transitive closure of a binary relation R is the least transitive relation including R.

Higher-order logic: expressiveness

- A second-order predicate expressing the transitivity of a binary relation:

$$\texttt{Trans} := \lambda R : A \to A \to \texttt{Prop}.\forall xyz : A.Rxy \Rightarrow Ryz \Rightarrow Rxz$$

- A binary relation R is included in S if for all x, y, Rxy implies Sxy:

$$\texttt{Subrel} := \lambda RS : A \to A \to \texttt{Prop}.\forall xy : A.Rxy \Rightarrow Sxy$$

- The transitive closure of a binary relation R is the least transitive relation including R. This can be defined as the intersection of all transitive relations including R:

$$\begin{aligned}\texttt{TC} := \lambda R : A \to A \to \texttt{Prop}.\lambda xy : A.\forall S : A \to A \to \texttt{Prop}.\\ \texttt{Trans}(S) \Rightarrow \texttt{Subrel}\, R\, S \Rightarrow Sxy\end{aligned}$$

Higher-order logic: expressiveness

$$\begin{aligned}\texttt{TC} := \lambda R : A \to A \to \texttt{Prop}.\lambda xy : A.\forall S : A \to A \to \texttt{Prop}.\\ \texttt{Trans}(S) \Rightarrow \texttt{Subrel}\, R\, S \Rightarrow Sxy\end{aligned}$$

Exercise: for arbitrary $R : A \to A \to \texttt{Prop}$ prove that $\texttt{TC}(R)$ is indeed the least transitive relation including R, i.e.,

- it is transitive:

$$\texttt{Trans}(\texttt{TC}(R))$$

- it includes R:

$$\texttt{Subrel}\, R\, (\texttt{TC}(R))$$

- every other transitive relation which includes R also includes $\texttt{TC}(R)$:

$$\forall S : A \to A \to \texttt{Prop}.\texttt{Trans}(S) \Rightarrow \texttt{Subrel}\, R\, S \Rightarrow \texttt{Subrel}\, (\texttt{TC}(R))\, S$$

Higher-order logic: expressiveness

Induction principle for natural numbers:

$$\forall P : \texttt{nat} \to \texttt{Prop}. P 0 \Rightarrow (\forall n : \texttt{nat}. P n \Rightarrow P(S n)) \Rightarrow \forall n : \texttt{nat}. P n$$

Higher-order encodings of logical connectives

In higher-order logic all other connectives and equality may be defined using $\forall$ and $\Rightarrow$.

Higher-order encodings of logical connectives

In higher-order logic all other connectives and equality may be defined using $\forall$ and $\Rightarrow$.

- Truth: $\top := \forall P : \texttt{Prop}. P \Rightarrow P$.

Higher-order encodings of logical connectives

In higher-order logic all other connectives and equality may be defined using $\forall$ and $\Rightarrow$.

- Truth: $\top := \forall P : \texttt{Prop}. P \Rightarrow P$.
- Falsity: $\bot := \forall P : \texttt{Prop}. P$.

Higher-order encodings of logical connectives

In higher-order logic all other connectives and equality may be defined using $\forall$ and $\Rightarrow$.

- Truth: $\top := \forall P : \texttt{Prop}.P \Rightarrow P$.
- Falsity: $\bot := \forall P : \texttt{Prop}.P$.
- Conjunction: $\varphi \wedge \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow \psi \Rightarrow P) \Rightarrow P$.

Higher-order encodings of logical connectives

In higher-order logic all other connectives and equality may be defined using $\forall$ and $\Rightarrow$.

- Truth: $\top := \forall P : \texttt{Prop}.P \Rightarrow P$.
- Falsity: $\bot := \forall P : \texttt{Prop}.P$.
- Conjunction: $\varphi \wedge \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow \psi \Rightarrow P) \Rightarrow P$.
- Disjunction: $\varphi \vee \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow P) \Rightarrow (\psi \Rightarrow P) \Rightarrow P$.

Higher-order encodings of logical connectives

In higher-order logic all other connectives and equality may be defined using $\forall$ and $\Rightarrow$.

- Truth: $\top := \forall P : \texttt{Prop}.P \Rightarrow P$.
- Falsity: $\bot := \forall P : \texttt{Prop}.P$.
- Conjunction: $\varphi \wedge \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow \psi \Rightarrow P) \Rightarrow P$.
- Disjunction: $\varphi \vee \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow P) \Rightarrow (\psi \Rightarrow P) \Rightarrow P$.
- Existential quantification: $\exists x : A.\varphi(x) := \forall P : \texttt{Prop}.\forall x : A(\varphi(x) \Rightarrow P) \Rightarrow P$.

Higher-order encodings of logical connectives

In higher-order logic all other connectives and equality may be defined using $\forall$ and $\Rightarrow$.

- Truth: $\top := \forall P : \texttt{Prop}.P \Rightarrow P$.
- Falsity: $\bot := \forall P : \texttt{Prop}.P$.
- Conjunction: $\varphi \wedge \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow \psi \Rightarrow P) \Rightarrow P$.
- Disjunction: $\varphi \vee \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow P) \Rightarrow (\psi \Rightarrow P) \Rightarrow P$.
- Existential quantification:
  $\exists x : A.\varphi(x) := \forall P : \texttt{Prop}.\forall x : A(\varphi(x) \Rightarrow P) \Rightarrow P$.
- Leibniz equality on A: $x =_A y := \forall R : A \to \texttt{Prop}.Rx \Rightarrow Ry$.

Higher-order encodings of logical connectives

In higher-order logic all other connectives and equality may be defined using $\forall$ and $\Rightarrow$.

- Truth: $\top := \forall P : \texttt{Prop}.P \Rightarrow P$.
- Falsity: $\bot := \forall P : \texttt{Prop}.P$.
- Conjunction: $\varphi \wedge \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow \psi \Rightarrow P) \Rightarrow P$.
- Disjunction: $\varphi \vee \psi := \forall P : \texttt{Prop}.(\varphi \Rightarrow P) \Rightarrow (\psi \Rightarrow P) \Rightarrow P$.
- Existential quantification:
  $\exists x : A.\varphi(x) := \forall P : \texttt{Prop}.\forall x : A(\varphi(x) \Rightarrow P) \Rightarrow P$.
- Leibniz equality on A: $x =_A y := \forall R : A \to \texttt{Prop}.Rx \Rightarrow Ry$.

The corresponding introduction and elimination rules are derivable.

Classical higher-order logic

Excluded middle axiom:

$$\forall P : \texttt{Prop}. P \vee \neg P$$

Extensionality

- Functional extensionality axiom (scheme):

$$\forall f g : A \to B.(\forall x : A.f x = g x) \Rightarrow f = g.$$

Extensionality

- Functional extensionality axiom (scheme):

$$\forall f g : A \to B.(\forall x : A. f x = g x) \Rightarrow f = g.$$

NOTE:

- Syntactic functional extensionality does not imply functional extensionality!

Extensionality

- Functional extensionality axiom (scheme):

$$\forall fg : A \to B.(\forall x : A.fx = gx) \Rightarrow f = g.$$

NOTE:

- Syntactic functional extensionality does <u>not</u> imply functional extensionality!
  - More precisely: that a formal system of logic satisfies syntactic functional extensionality (a <u>meta-theoretic</u> property!) does not imply that the functional extensionality axiom is provable in the system.

Extensionality

- Functional extensionality axiom (scheme):

$$\forall f g : A \to B. (\forall x : A. f x = g x) \Rightarrow f = g.$$

NOTE:

- Syntactic functional extensionality does not imply functional extensionality!
  - More precisely: that a formal system of logic satisfies syntactic functional extensionality (a meta-theoretic property!) does not imply that the functional extensionality axiom is provable in the system.
- Functional extensionality does not imply syntactic functional extensionality either!

Extensionality

- Functional extensionality axiom (scheme):

$$\forall f g : A \to B.(\forall x : A. f x = g x) \Rightarrow f = g.$$

NOTE:

- Syntactic functional extensionality does not imply functional extensionality!
  - More precisely: that a formal system of logic satisfies syntactic functional extensionality (a meta-theoretic property!) does not imply that the functional extensionality axiom is provable in the system.
- Functional extensionality does not imply syntactic functional extensionality either!

- Propositional extensionality axiom:

$$\forall P_1 P_2 : \texttt{Prop}.(P_1 \Leftrightarrow P_2) \Rightarrow P_1 =_{\texttt{Prop}} P_2.$$

Extensionality

- Functional extensionality axiom (scheme):

$$\forall fg : A \to B.(\forall x : A.fx = gx) \Rightarrow f = g.$$

NOTE:

- Syntactic functional extensionality does not imply functional extensionality!
  - More precisely: that a formal system of logic satisfies syntactic functional extensionality (a meta-theoretic property!) does not imply that the functional extensionality axiom is provable in the system.
- Functional extensionality does not imply syntactic functional extensionality either!

- Propositional extensionality axiom:

$$\forall P_1 P_2 : \mathtt{Prop}.(P_1 \Leftrightarrow P_2) \Rightarrow P_1 =_{\mathtt{Prop}} P_2.$$

- Predicate extensionality axiom (scheme):

$$\forall R_1 R_2 : A \to \mathtt{Prop}.(\forall x : A.R_1 x \Leftrightarrow R_2 x) \Rightarrow R_1 = R_2.$$

Choice

Axiom of choice (scheme):

$$(\forall x : A. \exists y : B. Rxy) \Rightarrow \exists f : A \to B. \forall x : A. Rx(fx).$$

Church's Simple Type Theory

- Church's Simple Type Theory is essentially classical higher-order logic with extensionality and choice.

Church's Simple Type Theory

- Church's Simple Type Theory is essentially classical higher-order logic with extensionality and choice.
- Alonzo Church, "A formulation of the simple theory of types", JSL 1940.

Church's Simple Type Theory

- Church's Simple Type Theory is essentially classical higher-order logic with extensionality and choice.
- Alonzo Church, "A formulation of the simple theory of types", JSL 1940.
    - The simply-typed lambda-calculus originates from this paper, where it was used to define the object terms of Church's higher-order logic.

Relativised choice

Relativised axiom of choice:

$$(\forall x : A.Qx \Rightarrow \exists y : B.Rxy) \Rightarrow \exists f : A \to B.\forall x : A.Qx \Rightarrow Rx(fx).$$

Diaconescu's theorem

Theorem (Diaconescu)

In intuitionistic higher-order logic, the predicate extensionality axiom and the relativised axiom of choice together imply the excluded middle axiom.