Polymorphism

I. What we have seen so far

Arithmetic can be expressed in the $\lambda\Pi$-calculus modulo
A variant of arithmetic (without N) led to inductive types

Simple type theory can be expressed in the $\lambda\Pi$-calculus modulo
A variant of Simple type theory (without ε) leads to polymorphism

Propositional contents

Perfect correspondence between propositions and propositional contents

Define sequents of propositional contents $t_1, ..., t_n \vdash u$ and drop ε

$t_1, ..., t_n \vdash u$ provable iff $\varepsilon(t_1), ..., \varepsilon(t_n) \vdash \varepsilon(u)$ is

Deduction rules directly on such sequents, for instance

$$\frac{\Gamma \vdash (\dot{\wedge}\ t\ u)}{\Gamma \vdash t}$$

II. The Calculus of constructions

Dropping reduction rules

$$\varepsilon(\dot{\Rightarrow}\ x\ y) \longrightarrow \varepsilon(x) \to \varepsilon(y)$$

becomes

$$(\dot{\Rightarrow}\ x\ y) \longrightarrow (x \to y)$$

i.e. replace $\dot{\Rightarrow}$ by $\to$

Same for

$$\varepsilon(\dot{\forall}_A\ x) \longrightarrow \Pi y : |A|\ \varepsilon(x\ y)$$

But... not all propositions are typable

$o : \mathit{Type}$

$c : \iota \to o : \mathit{Type}$

$\Pi c : \iota \to o\ \varepsilon(...)$ possible

x

$c : \iota \to \mathit{Type} : \mathit{Kind}$

not $\Pi c : \iota \to \mathit{Type}$...

A tension

Simple type theory $o : \mathit{Type}$
Curry-de Bruijn-Howard $o = \mathit{Type}$

Simple type theory + Curry-de Bruijn-Howard $\mathit{Type} : \mathit{Type}$

One way out: $o \neq \mathit{Type}$ ... ε (embedding)
Another $o \not: \mathit{Type}$ ... $o : \mathit{Kind}$

New typing rules

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x : A \vdash B : \mathit{Type}}{\Gamma \vdash \Pi x : A\ B : \mathit{Type}}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x : A \vdash B : \mathit{Kind}}{\Gamma \vdash \Pi x : A\ B : \mathit{Kind}}$$

$$\frac{\Gamma \vdash A : \mathit{Kind} \quad \Gamma, x : A \vdash B : \mathit{Type}}{\Gamma \vdash \Pi x : A\ B : \mathit{Type}}$$

$$\frac{\Gamma \vdash A : \mathit{Kind} \quad \Gamma, x : A \vdash B : \mathit{Kind}}{\Gamma \vdash \Pi x : A\ B : \mathit{Kind}}$$

The Calculus of constructions

Second new rule (type constructors): build the kind $\mathit{Type} \to \mathit{Type}$
A variable *array* of type $\mathit{Type} \to \mathit{Type}$ and types (*array nat*), (*array bool*), (*array* (*array nat*)), etc.

First new rule (polymorphic types): build the type
$\Pi x : \mathit{Type}\ (x \to (\mathit{array}\ x) \to (\mathit{array}\ x))$
A variable *cons* of this type

The Calculus of constructions

The empty context is well-formed:

$$\frac{}{[\,] \text{ well-formed}}$$

Declaration of a variable:

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \text{ well-formed}}$$

Type is a kind:

$$\frac{\Gamma \text{ well-formed}}{\Gamma \vdash \mathit{Type} : \mathit{Kind}}$$

Variables are terms:

$$\frac{\Gamma \text{ well-formed} \quad x : A \in \Gamma}{\Gamma \vdash x : A}$$

Product:

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A\ B : s_2}$$

Abstraction:

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}$$

Application:

$$\frac{\Gamma \vdash t : \Pi x : A\ B \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : (t'/x)B}$$

Conversion:

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash A : s \quad \Gamma \vdash B : s \quad A \equiv B}{\Gamma \vdash t : B}$$

Termination

Similar to $\lambda\Pi$-modulo the rules of Simple type theory

- $\mathcal{M}_{Type} = \mathcal{M}_{Kind} = \mathcal{C}$
- $\mathcal{M}_x = \{e\}$, an arbitrary singleton
- $\mathcal{M}_{\lambda x:A\ t} = \mathcal{M}_t$
- $\mathcal{M}_{(t\ u)} = \mathcal{M}_t$
- $\mathcal{M}_{\Pi x:A\ B}$ is the set of functions f from $\mathcal{M}_A$ to $\mathcal{M}_B$ except if $\mathcal{M}_B = \{e\}$, in which case $\mathcal{M}_{\Pi x:A\ B} = \{e\}$ or if $\mathcal{M}_A = \{e\}$ in which case $\mathcal{M}_{\Pi x:A\ B} = \mathcal{M}_B$

Like $\lambda\Pi$-calculus and $\lambda\Pi$-calculus modulo, if t object, then $\mathcal{M}_t = \{e\}$

Like $\lambda\Pi$-calculus and unlike in the $\lambda\Pi$-calculus modulo, if t is a type, we have $\mathcal{M}_t = \{e\}$ (no o at the Type level)

Unlike $\lambda\Pi$-calculus and like in the $\lambda\Pi$-calculus modulo, if t is a kind we do not have $\mathcal{M}_t = \mathcal{C}$ ($\mathcal{M}_{\mathit{Type} \to \mathit{Type}} = \mathcal{C} \to \mathcal{C}$)

- $[\![Type]\!]_\phi = [\![Kind]\!]_\phi$ set of strongly terminating terms
- $[\![x]\!]_\phi = \phi(x)$
- $[\![\lambda x : C\ t]\!]_\phi$ function mapping a in $\mathcal{M}_C$ to $[\![t]\!]_{\phi,x=a}$, except if $[\![t]\!]_{\phi,x=a} = e$ for all a, in which case $[\![\lambda x : C\ t]\!]_\phi = e$, or if $\mathcal{M}_C = \{e\}$, in which case $[\![\lambda x : C\ t]\!]_\phi = [\![t]\!]_{\phi,x=e}$
- $[\![(t\ u)]\!]_\phi = [\![t]\!]_\phi([\![u]\!]_\phi)$, except if $[\![t]\!]_\phi = e$, in which case $[\![(t\ u)]\!]_\phi = e$, or if $[\![u]\!]_\phi = e$, in which case $[\![(t\ u)]\!]_\phi = [\![t]\!]_\phi$
- $[\![\Pi x : C\ D]\!]_\phi = \tilde{\Pi}([\![C]\!]_\phi, \{[\![D]\!]_{\phi,x=c} \mid c \in \mathcal{M}_C\})$

If t a term of type B in Γ, then $t \in [\![B]\!]_\phi$, hence t strongly terminates

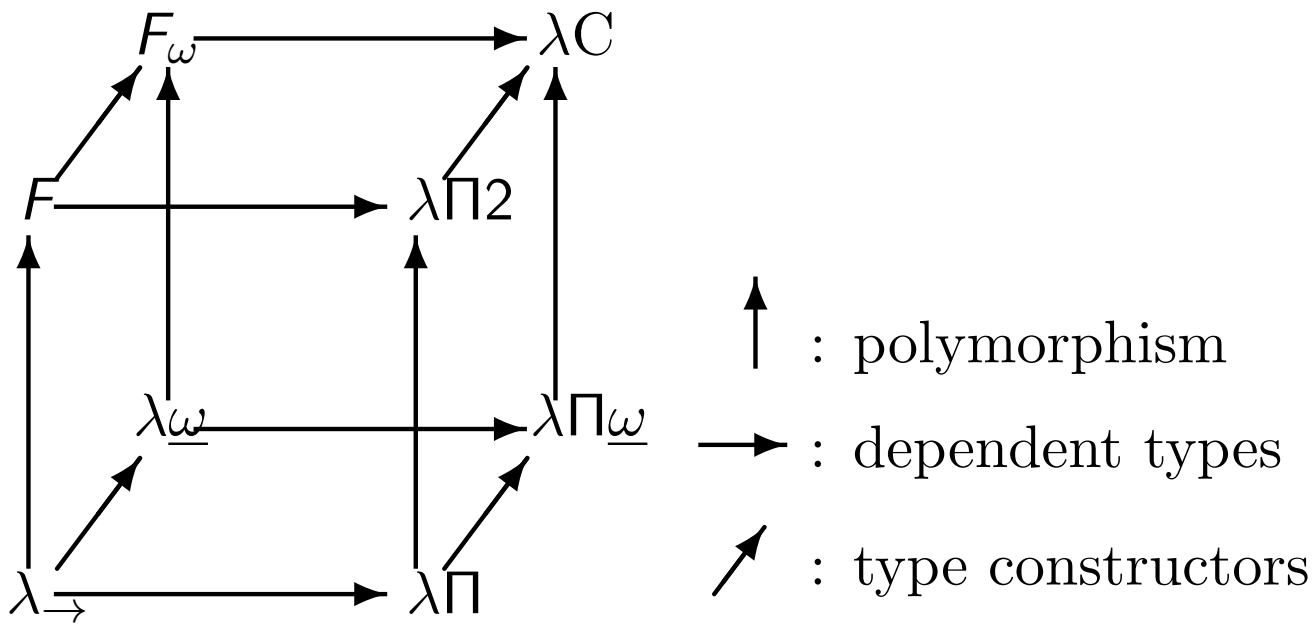
III. Pure Type Systems

Simply typed λ-calculus, $\lambda\Pi$, the Calculus of constructions
Only difference: Product and Abstraction rule
All the Product rules

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A\ B : s_3}$$

$\langle s_1, s_2, s_3 \rangle = \langle \mathit{Type}, \mathit{Type}, \mathit{Type} \rangle$ in all systems
$\langle \mathit{Type}, \mathit{Kind}, \mathit{Kind} \rangle$: dependent types
$\langle \mathit{Kind}, \mathit{Type}, \mathit{Type} \rangle$: polymorphic types
$\langle \mathit{Kind}, \mathit{Kind}, \mathit{Kind} \rangle$: type constructors

Barendregt's cube



More generally

An arbitrary set S of *sorts*

an arbitrary set Ax of pairs of sorts (*axioms*)

an arbitrary set of triples R of sorts (*rules*)

The empty context is well-formed:

$$\frac{}{[\,] \text{ well-formed}}$$

Declaration of a variable:

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \text{ well-formed}}\, s \in S$$

Typing the sorts:

$$\frac{\Gamma \text{ well-formed}}{\Gamma \vdash s_1 : s_2}\, \langle s_1, s_2 \rangle \in Ax$$

Variables are terms:

$$\frac{\Gamma \text{ well-formed} \quad x : A \in \Gamma}{\Gamma \vdash x : A}$$

Product:

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A\ B : s_3}\ \langle s_1, s_2, s_3 \rangle \in R$$

Abstraction:

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}\ \langle s_1, s_2, s_3 \rangle \in R$$

Application:

$$\frac{\Gamma \vdash t : \Pi x : A\ B \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : (t'/x)B}$$

Conversion:

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash A : s \quad \Gamma \vdash B : s \quad A \equiv B}{\Gamma \vdash t : B}\ s \in S$$

IV. The System F

The empty context is well-formed:

$$\frac{}{[\,] \text{ well-formed}}$$

Declaration of a variable:

$$\frac{\Gamma \vdash A : \mathit{Type}}{\Gamma, x : A \text{ well-formed}}$$

$$\frac{\Gamma \vdash A : \mathit{Kind}}{\Gamma, x : A \text{ well-formed}}$$

Type is a kind:

$$\frac{\Gamma \text{ well-formed}}{\Gamma \vdash \mathit{Type} : \mathit{Kind}}$$

Variables are terms:

$$\frac{\Gamma \text{ well-formed} \quad x : A \in \Gamma}{\Gamma \vdash x : A}$$

Product:

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x : A \vdash B : \mathit{Type}}{\Gamma \vdash \Pi x : A\ B : \mathit{Type}}$$

$$\frac{\Gamma \vdash A : \mathit{Kind} \quad \Gamma, x : A \vdash B : \mathit{Type}}{\Gamma \vdash \Pi x : A\ B : \mathit{Type}}$$

Abstraction:

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x : A \vdash B : \mathit{Type} \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}$$

$$\frac{\Gamma \vdash A : \mathit{Kind} \quad \Gamma, x : A \vdash B : \mathit{Type} \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A\ t : \Pi x : A\ B}$$

Application:

$$\frac{\Gamma \vdash t : \Pi x : A\ B \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : (t'/x)B}$$

Conversion:

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash A : \mathit{Type} \quad \Gamma \vdash B : \mathit{Type} \quad A \equiv B}{\Gamma \vdash t : B}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash A : \mathit{Kind} \quad \Gamma \vdash B : \mathit{Kind} \quad A \equiv B}{\Gamma \vdash t : B}$$

A much simpler definition

- A product always has type *Type*: *Type* is the only term of type *Kind*, drop the symbol *Kind*
- No conversion in types: drop conversion rule
- If A and B have type *Type*, no variable of type A can occur in B and the type $\Pi x : A\ B$ can be written $A \to B$

The empty context is well-formed:

$$\frac{}{[\,] \text{ well-formed}}$$

Declaration of a variable:

$$\frac{\Gamma \vdash A : \mathit{Type}}{\Gamma, x : A \text{ well-formed}}$$

$$\frac{\Gamma \text{ well-formed}}{\Gamma, x : \mathit{Type} \text{ well-formed}}$$

Variables are terms:

$$\frac{\Gamma \text{ well-formed} \quad x : A \in \Gamma}{\Gamma \vdash x : A}$$

Product:

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x : A \vdash B : \mathit{Type}}{\Gamma \vdash A \to B : \mathit{Type}}$$

$$\frac{\Gamma, x : \mathit{Type} \vdash B : \mathit{Type}}{\Gamma \vdash \Pi x : \mathit{Type}\ B : \mathit{Type}}$$

Abstraction:

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x : A \vdash B : \mathit{Type} \quad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A\ t : A \to B}$$

$$\frac{\Gamma, x : \mathit{Type} \vdash B : \mathit{Type} \quad \Gamma, x : \mathit{Type} \vdash t : B}{\Gamma \vdash \lambda x : \mathit{Type}\ t : \Pi x : \mathit{Type}\ B}$$

Application:

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash t' : A}{\Gamma \vdash (t\ t') : B}$$

$$\frac{\Gamma \vdash t : \Pi x : \mathit{Type}\ B \quad \Gamma \vdash t' : \mathit{Type}}{\Gamma \vdash (t\ t') : (t'/x)B}$$

With respect to Simply typed λ-calculus: three more rules

- A type formation rule, allowing to build types of the form $\Pi x : \mathit{Type}\ A\ (\Pi x : \mathit{Type}\ (x \to x))$
- An abstraction rule allowing to build terms of such types, (polymorphic identity $\lambda x : \mathit{Type}\ \lambda y : x\ y$)
- An application rule allowing to apply such polymorphic functions to a particular type $((\lambda x : \mathit{Type}\ \lambda y : x\ y)\ \mathit{nat})$

Foundations of proof systems

Constructive (Coq, Adga, Dedukti) or not (HOL, Isabelle, PVS)
Simple type theory (HOL, Isabelle, PVS) or proofs as objects (Coq, Adga, Dedukti)

But more common points than differences

Theories in proof systems

More often Simple type theory and extensions than set theory

that is: more often typed set theory than untyped set theory

that is: more often set theory in a many sorted logic than in a single sorted one

What is a theory?

A set of axioms

More often: deduction rules or computation rules

Deduction rules or computation rules?

This new vision on theories

Useful in proof systems, but also in automated theorem proving

Next time

Master Class + Exercises