Introduction to Type Theory

February 2008

Alpha Lernet Summer School

Piriapolis, Uruguay

Herman Geuvers

Nijmegen & Eindhoven, NL

Lecture 1: Introduction, Overview, Simple Type Theory

Types and sets

Types are not sets.

- Types are a bit like sets, but: ...
- types give "syntactic information"

$$3 + (7 * 8)^5 : \mathsf{nat}$$

- sets give "semantic information"

$$3 \in \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}$$

Sets are about semantics

$$3 \in \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}$$

because there are no positive x, y, z such that $x^n + y^n = z^n$ [Wiles; "Fermat's last Theorem"]

- set theory talks about what things exist (semantics, ontology). A set X such that for all sets Y with $|Y| < |X|$, $|2^Y| < |X|$?
- sets are extensional:

$$\{n \in \mathbb{N} \mid \exists x, y, z \in \mathbb{N}^+ (x^n + y^n = z^n)\} = \{0, 1, 2\}$$

- sets are "collections of things".
- membership is undecidable

Types are about syntax

$$3 + (7 * 8)^5 : \mathsf{nat}$$

because 3, 7, 8 are of type nat and the operations take objects of type nat to nat.

$$\frac{1}{2}\Sigma_{n=0}^{\infty} 2^{-n} : \mathbb{N}$$

is not a typing judgment.

- type theory talks about how things can be constructed (syntax, formal language, expressions)
- types are intensional

  $$\{n \mid \exists x, y, z \in \mathsf{nat}^+ (x^n + y^n \neq z^n)\} \neq \{n \mid n = 0 \vee n = 1 \vee n = 2\}$$

- typing (and type checking) is decidable

Overview

What I will be doing in these lectures.

Problem: there are so many type systems and so many ways of defining them . . . .

Central theme: two readings of typing judments

$$M : A$$

- M is a term (program, expression) of the data type A
- M is a proof (derivation) of the formula A

Curry-Howard isomorphism of formulas-as-types
(and proofs-as-terms)

Overview of these lectures

1. Simply typed λ-calculus (Simple Type Theory) and Curry Howard isomorphism
2. Simple Type Theory: "Curry" type assignment, principle type algorithm and normalization
3. Polymorphic type theory: full polymorphism and ML style polymorphism
4. Dependent type theory: logical framework and type checking algorithm
5. Higher order logic, inductive types and proof assistants based on type theory.

Simplest system: $\lambda\!\!\rightarrow$ or STT

Just arrow types

$$\mathsf{Typ} := \mathsf{TVar} \mid (\mathsf{Typ}\!\rightarrow\!\mathsf{Typ})$$

- Examples: $(\alpha\!\rightarrow\!\beta)\!\rightarrow\!\alpha$, $(\alpha\!\rightarrow\!\beta)\!\rightarrow\!((\beta\!\rightarrow\!\gamma)\!\rightarrow\!(\alpha\!\rightarrow\!\gamma))$
- Brackets associate to the right and outside brackets are omitted: $(\alpha\!\rightarrow\!\beta)\!\rightarrow\!(\beta\!\rightarrow\!\gamma)\!\rightarrow\!\alpha\!\rightarrow\!\gamma$ denotes $(\alpha\!\rightarrow\!\beta)\!\rightarrow\!((\beta\!\rightarrow\!\gamma)\!\rightarrow\!(\alpha\!\rightarrow\!\gamma))$
- Types are denoted by $\sigma, \tau, \ldots$.

Terms:

- typed variables $x_1^\sigma, x_2^\sigma, \ldots$, countably many for every σ.
- application: if $M : \sigma\!\rightarrow\!\tau$ and $N : \sigma$, then $(MN) : \tau$
- abstraction: if $P : \tau$, then $(\lambda x^\sigma.P) : \sigma\!\rightarrow\!\tau$

Examples

$$
\begin{array}{rrcl}
\mathbf{I}_\sigma := & \lambda x^\sigma.x & : & \sigma\to\sigma \\
\mathbf{K}_{\sigma\tau} := & \lambda x^\sigma.\lambda y^\tau.x & : & \sigma\to\tau\to\sigma \\
& \lambda x^{\alpha\to\beta}.\lambda y^{\beta\to\gamma}.\lambda z^\alpha.y(xz) & : & (\alpha\to\beta)\to(\beta\to\gamma)\to\alpha\to\gamma \\
& \lambda x^\alpha.\lambda y^{(\beta\to\alpha)\to\alpha}.y(\lambda z^\beta.x) & : & \alpha\to((\beta\to\alpha)\to\alpha)\to\alpha
\end{array}
$$

For $\lambda x^{\alpha\to\beta}.\lambda y^{\beta\to\gamma}.\lambda z^\alpha.y(xz)$:

- If $x : \alpha\to\beta$, $y : \beta\to\gamma$ and $z : \alpha$, then $xz : \beta$,
- so $y(xz) : \gamma$,
- so $\lambda z^\alpha.y(xz) : \alpha\to\gamma$,
- so $\lambda y^{\beta\to\gamma}.\lambda z^\alpha.y(xz) : (\beta\to\gamma)\to\alpha\to\gamma$,
- so $\lambda x^{\alpha\to\beta}.\lambda y^{\beta\to\gamma}.\lambda z^\alpha.y(xz) : (\alpha\to\beta)\to(\beta\to\gamma)\to\alpha\to\gamma$

Notation

In λ-calculus (and type theory) we have some special notational conventions:

- λ-abstraction:
  - we write $\lambda x^\sigma.\lambda y^\tau.M$ or even $\lambda x^\sigma\, y^\tau.M$ for $\lambda x^\sigma.(\lambda y^\sigma.M)$
- Function application:
  - we write $(F\,M)$ (and not $F(M)$)!
  - in multiple applications, $F\,M\,N$ denotes $(F\,M)\,N$ (and not $F\,(M\,N)$).
  "Brackets associate to the left" in applications.
- Types:
  - we write $\sigma{\to}\tau{\to}\rho$ for $\sigma{\to}(\tau{\to}\rho)$.
  "Brackets associate to the right" in types.

Notation

Conventions about types and applications fit together nicely:
If $F : \sigma \to \tau \to \rho$, $M : \sigma$ and $P : \tau$, then

$$F\,M : \tau \to \rho \quad \text{and} \quad F\,M\,P : \rho$$

NB Every type of STT can be written as

$$\sigma_1 \to \sigma_2 \to \ldots \to \alpha$$

with α a type variable

NB Functions of multiple arguments are dealt with by Currying:
We don't have

$$F : \sigma \times \tau \to \rho$$

but instead we use

$$F : \sigma \to \tau \to \rho$$

Computation

A λ-term of the form $(\lambda x^\sigma.M)P$ is a β-redex (reducible expression).
A redex can be contracted:

$$(\lambda x^\sigma.M)P \longrightarrow_\beta M[x := P]$$

$M[x := P]$ denotes M with P substituted for x

Example $(\lambda x^\sigma.\lambda y^\tau.x)P \longrightarrow_\beta \lambda y^\tau.P$

But what if $P = y$? then

$$(\lambda x^\sigma.\lambda y^\tau.x)y \longrightarrow_\beta \lambda y^\tau.y \quad \dagger\dagger!!$$

This is clearly not what we want.

The λ is a binder and we have to make sure that free variables don't get bound by a substitution.

Solution: Remaning of bound variables before substitution

Free and bound variables and α-conversion

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{FV}(x) & = & \{x\} & \mathsf{BV}(x) & = & \emptyset \\
\mathsf{FV}(MN) & = & \mathsf{FV}(M) \cup \mathsf{FV}(N) & \mathsf{BV}(MN) & = & \mathsf{BV}(M) \cup \mathsf{BV}(N) \\
\mathsf{FV}(\lambda x^\sigma.M) & = & \mathsf{FV}(M) \setminus \{x\} & \mathsf{BV}(\lambda x^\sigma.M) & = & \mathsf{BV}(M) \cup \{x\}
\end{array}$$

Definition $M \equiv N$ or $M =_\alpha N$

if M is equal to N modulo renaming of bound variables

Examples:

- $\lambda x^\sigma.\lambda y^\tau.x \equiv \lambda x^\sigma.\lambda z^\tau.x$
- $\lambda x^\sigma.\lambda y^\tau.x \equiv \lambda y^\sigma.\lambda x^\tau.y$
- NB also: $\lambda x^\sigma.\lambda y^\tau.y \equiv \lambda x^\sigma.\lambda x^\tau.x$

We work "modulo α-conversion"

We consider terms modulo α-equality, so we don't distinguish between

$$\lambda x^{\sigma}.\lambda y^{\tau}.x \text{ and } \lambda x^{\sigma}.\lambda z^{\tau}.x.$$

In examples we always rename bound variables such that no clashes can arise.
This is known as the Barendregt convention

Before reduction or substitution, we rename (if necessary):

$$(\lambda x^{\sigma}.\lambda y^{\tau}.x)y \equiv (\lambda x^{\sigma}.\lambda z^{\tau}.x)y \longrightarrow_{\beta} \lambda z^{\tau}.y$$

Notation: $\twoheadrightarrow_{\beta}$ is the transitive reflexive closure of $\longrightarrow_{\beta}$,
$=_{\beta}$ is the transitive reflexive symmetric closure of $\longrightarrow_{\beta}$.

Examples of computation with simple typed terms

The type $(\sigma\to\sigma)\to\sigma\to\sigma$ is called the type of numerals over σ:

$$\bar{n} := \lambda f^{\sigma\to\sigma}.\lambda x^{\sigma}.f^{n}(x)$$

where

$$f^{n}(x) \text{ denotes } \underbrace{f(\ldots f(f}_{n \text{ times } f}\ x))$$

So $\bar{2} := \lambda f^{\sigma\to\sigma}.\lambda x^{\sigma}.f(f\,x)$

$$\begin{array}{rcl}
\lambda z^{\sigma}.\bar{2}\,I_{\sigma}\,z & \equiv & \lambda z^{\sigma}.(\lambda f^{\sigma\to\sigma}.\lambda x^{\sigma}.f(f\,x))I_{\sigma}\,z \\
& \longrightarrow_{\beta} & \lambda z^{\sigma}.(\lambda x^{\sigma}.\mathbf{I}_{\sigma}(\mathbf{I}_{\sigma}\,x))z \\
& \longrightarrow_{\beta} & \lambda z^{\sigma}.\mathbf{I}_{\sigma}(\mathbf{I}_{\sigma}\,z) \\
& \longrightarrow_{\beta} & \lambda z^{\sigma}.\mathbf{I}_{\sigma}\,z \\
& \longrightarrow_{\beta} & \lambda z^{\sigma}.z \equiv \mathbf{I}_{\sigma}
\end{array}$$

Examples of computation with simple typed terms

$$\mathbf{S} := \lambda x^{\sigma\to\sigma\to\sigma}.\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.x\, z(y\, z) \;\; : \;\; (\sigma\to\sigma\to\sigma)\to(\sigma\to\sigma)\to\sigma\to\sigma$$

Then $\mathbf{S}\,\mathbf{K}_{\sigma\sigma}\,\mathbf{I}_{\sigma} : \sigma\to\sigma$ and

$$\mathbf{S}\,\mathbf{K}_{\sigma\sigma}\,\mathbf{I}_{\sigma} \longrightarrow_{\beta} (\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.\mathbf{K}_{\sigma\sigma}\, z(y\, z))\mathbf{I}_{\sigma}$$

There are several ways of reducing this term further:

$$(\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.\mathbf{K}_{\sigma\sigma}\, z(y\, z))\mathbf{I}_{\sigma} \qquad \text{is a redex}$$

$$\mathbf{K}_{\sigma\sigma}\, z \qquad \text{is a redex}$$

Example ctd

$$
\begin{array}{rcl}
(\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.\mathbf{K}_{\sigma\sigma}\; z(y\,z))\mathbf{I}_{\sigma} & \longrightarrow_{\beta} & \lambda z^{\sigma}.\mathbf{K}_{\sigma\sigma}\; z(\mathbf{I}_{\sigma}\; z) \\
 & \equiv & \lambda z^{\sigma}.(\lambda p^{\sigma}, q^{\sigma}.p)\; z(\mathbf{I}_{\sigma}\; z) \\
 & \longrightarrow_{\beta} & \lambda z^{\sigma}.(\lambda q^{\sigma}.z)\; (\mathbf{I}_{\sigma}\; z) \\
\text{Call by Value} & \longrightarrow_{\beta} & \lambda z^{\sigma}.(\lambda q^{\sigma}.z)z \\
 & \longrightarrow_{\beta} & \lambda z^{\sigma}.z
\end{array}
$$

But also

$$
\begin{array}{rcl}
(\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.\mathbf{K}_{\sigma\sigma}\; z(y\,z))\mathbf{I}_{\sigma} & \equiv & (\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.(\lambda p^{\sigma}, q^{\sigma}.p)\; z(y\,z))\mathbf{I}_{\sigma} \\
 & \longrightarrow_{\beta} & (\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.(\lambda q^{\sigma}.z)(y\,z)\mathbf{I}_{\sigma} \\
\text{Call by Name} & \longrightarrow_{\beta} & (\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.z)\mathbf{I}_{\sigma} \\
 & \longrightarrow_{\beta} & \lambda z^{\sigma}.z
\end{array}
$$

Properties of reduction in STT

- Subject Reduction
  If $M : \sigma$ and $M \longrightarrow_\beta P$, then $P : \sigma$.

- Church-Rosser
  If M is well-typed and $M \twoheadrightarrow_\beta P$ and $M \twoheadrightarrow_\beta N$, then there is a Q such that $P \twoheadrightarrow_\beta Q$ and $N \twoheadrightarrow_\beta Q$.

- Strong Normalisation
  If M is well-typed, then there is no infinite β-reduction path starting from M.

Different presentations of STT

Inductive definition of the terms:

- typed variables $x_1^\sigma, x_2^\sigma, \ldots$, countably many for every σ.
- application: if $M : \sigma \to \tau$ and $N : \sigma$, then $(MN) : \tau$
- abstraction: if $P : \tau$, then $(\lambda x^\sigma . P) : \sigma \to \tau$

Alternative: Inductive definition of the terms in rule form:

$$\frac{}{x^\sigma : \sigma} \qquad \frac{M : \sigma \to \tau \;\; N : \sigma}{MN : \tau} \qquad \frac{P : \tau}{\lambda x^\sigma . P : \sigma \to \tau}$$

Advantage: We also have a derivation tree, a proof of the fact that the term has that type.

We can reason over derivations.

Simple type theory a la Church

Formulation with contexts to declare the free variables:

$$x_1 : \sigma_1, x_2 : \sigma_2, \ldots, x_n : \sigma_n$$

is a context, usually denoted by Γ.

Derivation rules of $\lambda\!\!\rightarrow$ (à la Church):

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma{\rightarrow}\tau \;\; \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \qquad \frac{\Gamma, x{:}\sigma \vdash P : \tau}{\Gamma \vdash \lambda x{:}\sigma.P : \sigma{\rightarrow}\tau}$$

$\Gamma \vdash_{\lambda\rightarrow} M : \sigma$ if there is a derivation using these rules with conclusion $\Gamma \vdash M : \sigma$

Formulas-as-Types (Curry, Howard)

There are two readings of a judgement $M : \sigma$

1. term as algorithm/program, type as specification:
   M is a function of type σ

2. type as a proposition, term as its proof:
   M is a proof of the proposition σ

- There is a one-to-one correspondence:

  typable terms in $\lambda{\to} \simeq$ derivations in minimal proposition logic

- The judgement $x_1 : \tau_1, x_2 : \tau_2, \ldots, x_n : \tau_n \vdash M : \sigma$ can be read as
  M is a proof of σ from the assumptions $\tau_1, \tau_2, \ldots, \tau_n$.

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{[\alpha\to\beta\to\gamma]^3 \quad [\alpha]^1}{\beta\to\gamma} \qquad \dfrac{[\alpha\to\beta]^2 \quad [\alpha]^1}{\beta}}{\gamma}\,1}{\alpha\to\gamma}\,2}{(\alpha\to\beta)\to\alpha\to\gamma}\,3}{(\alpha\to\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma} \quad \simeq \quad \begin{array}{l}\lambda x{:}\alpha\to\beta\to\gamma.\lambda y{:}\alpha\to\beta.\lambda z{:}\alpha.xz(yz)\\ : (\alpha\to\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma\end{array}$$

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{[x : \alpha{\to}\beta{\to}\gamma]^3 \;\; [z : \alpha]^1}{xz : \beta{\to}\gamma} \qquad \dfrac{[y : \alpha{\to}\beta]^2 \;\; [z : \alpha]^1}{yz : \beta}}{xz(yz) : \gamma}\,1}{\lambda z{:}\alpha.xz(yz) : \alpha{\to}\gamma}\,2}{\lambda y{:}\alpha{\to}\beta.\lambda z{:}\alpha.xz(yz) : (\alpha{\to}\beta){\to}\alpha{\to}\gamma}\,3$$

$$\lambda x{:}\alpha{\to}\beta{\to}\gamma.\lambda y{:}\alpha{\to}\beta.\lambda z{:}\alpha.xz(yz) : (\alpha{\to}\beta{\to}\gamma){\to}(\alpha{\to}\beta){\to}\alpha{\to}\gamma$$

Exercise: Add types to the λ-abstractions) and give the derivation that corresponds to

$$\lambda x.\lambda y.y(\lambda z.y\,x) : (\gamma{\to}\varepsilon){\to}((\gamma{\to}\varepsilon){\to}\varepsilon){\to}\varepsilon$$

Computation

Cut-elimination in minimal logic $=$ β-reduction in $\lambda\!\!\rightarrow$.

$$\dfrac{\dfrac{\begin{array}{c}[\sigma]^1\\ \mathcal{D}_1\\ \tau\end{array}}{\sigma\!\rightarrow\!\tau}\,1 \qquad \dfrac{\mathcal{D}_2}{\sigma}}{\tau} \quad\longrightarrow\quad \begin{array}{c}\mathcal{D}_2\\ \sigma\\ \mathcal{D}_1\\ \tau\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}[x:\sigma]^1\\ \mathcal{D}_1\\ M:\tau\end{array}}{\lambda x{:}\sigma.M:\sigma\!\rightarrow\!\tau}\,1 \qquad \dfrac{\mathcal{D}_2}{P:\sigma}}{(\lambda x{:}\sigma.M)P:\tau} \quad\longrightarrow_\beta\quad \begin{array}{c}\mathcal{D}_2\\ P:\sigma\\ \mathcal{D}_1\\ M[x:=P]:\tau\end{array}$$

Example: Proof of $A \to A \to B, (A \to B) \to A \vdash B$

$$\dfrac{\dfrac{[A]^1 \quad \dfrac{[A]^1 \quad A \to A \to B}{A \to B}}{B} \qquad \dfrac{[A]^1 \quad A \to B}{B}}{\dfrac{A \to B \qquad (A \to B) \to A}{A}}$$
$$\dfrac{\qquad\qquad}{B}$$

It contains a cut: a $\to$-i directly followed by an $\to$-e.

Example: Proof of $A \to A \to B, (A \to B) \to A \vdash B$ after reduction

$$\dfrac{\dfrac{(A\to B)\to A \qquad \dfrac{[A]^1 \qquad \dfrac{[A]^1 \qquad A\to A\to B}{A\to B}}{B}}{A} \qquad \dfrac{(A\to B)\to A \qquad \dfrac{[A]^1 \qquad \dfrac{[A]^1 \qquad A\to A\to B}{A\to B}}{B}}{A} \qquad A\to A\to B}{B}$$

Example: Proof of $A{\to}A{\to}B, (A{\to}B){\to}A \vdash B$ with term information

$$\dfrac{\dfrac{\dfrac{[x:A]^1 \quad \dfrac{[x:A]^1 \quad p : A{\to}A{\to}B}{p\,x : A{\to}B}}{p\,x\,x : B}}{\lambda x{:}A.p\,x\,x : A{\to}B} \qquad \dfrac{q : (A{\to}B){\to}A \qquad \dfrac{[x:A]^1 \quad p : A{\to}A{\to}B}{p\,x : A{\to}B}}{q(\lambda x{:}A.p\,x\,x) : A}}{(\lambda x{:}A.p\,x\,x)(q(\lambda x{:}A.p\,x\,x)) : B}$$

Term contains a β-redex: $(\lambda x{:}A.p\,x\,x)\,(q(\lambda x{:}A.p\,x\,x))$

Example: Reduced proof of $A \to A \to B, (A \to B) \to A \vdash B$ with term info

$$\dfrac{\dfrac{q : (A \to B) \to A \qquad \dfrac{\dfrac{[x : A]^1 \qquad \dfrac{p : A \to A \to B}{p\ x : A \to B}}{p\ x\ x : B}}{\lambda x{:}A.p\ x\ x : A \to B}}{q(\lambda x{:}A.p\ x\ x) : A} \qquad \dfrac{\dfrac{q : (A \to B) \to A \qquad \dfrac{\dfrac{[x : A]^1 \qquad \dfrac{p : A \to A \to B}{p\ x : A \to B}}{p\ x\ x : B}}{\lambda x{:}A.p\ x\ x : A \to B}}{q(\lambda x{:}A.p\ x\ x) : A} \qquad p : A \to A \to B}{p(q(\lambda x{:}A.p\ x\ x)) : A \to B}}{p(q(\lambda x{:}A.p\ x\ x))(q(\lambda x{:}A.p\ x\ x)) : B}$$