Introduction to Type Theory
August 2007
Types Summer School
Bertinoro, It

Herman Geuvers
Nijmegen NL

Lecture 3: Polymorphic λ-calculus

Why Polymorphic λ-calculus?

- Simple type theory $\lambda{\rightarrow}$ is not very expressive
- In simple type theory, we can not 'reuse' a function.
  E.g. $\lambda x{:}\alpha.x : \alpha{\rightarrow}\alpha$ and $\lambda x{:}\beta.x : \beta{\rightarrow}\beta$.

We want to define functions that can treat types polymorphically: add types $\forall\alpha.\sigma$:

Examples

- $\forall\alpha.\alpha{\rightarrow}\alpha$
  If $M : \forall\alpha.\alpha{\rightarrow}\alpha$, then M can map any type to itself.
- $\forall\alpha.\forall\beta.\alpha{\rightarrow}\beta{\rightarrow}\alpha$
  If $M : \forall\alpha.\forall\beta.\alpha{\rightarrow}\beta{\rightarrow}\alpha$, then M can take two inputs (of arbitrary types) and return a value of the first input type.

Derivation rules for Weak (ML-style) polymorphism,

Typ : add $\forall \alpha_1 . \ldots . \forall \alpha_n . \sigma$ for σ a $\lambda {\to}$-type.

1. Curry style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall \alpha . \sigma} \; \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall \alpha . \sigma}{\Gamma \vdash M : \sigma[\tau/\alpha]} \text{ for } \tau \text{ a } \lambda{\to}\text{-type}$$

2. Church style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda \alpha . M : \forall \alpha . \sigma} \; \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall \alpha . \sigma}{\Gamma \vdash M \tau : \sigma[\tau/\alpha]} \text{ for } \tau \text{ a } \lambda{\to}\text{-type}$$

- $\forall$ only occurs on the outside and is therefore usually left out: "all type variables are implicitly universally quantified"
- With weak polymorphism, type checking is still decidable: the principal types algorithm still works.

Derivation rules for Weak (ML-style) polymorphism,

Also the abstraction rule is restricted to $\lambda{\rightarrow}$-types:

1. Curry style: $$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau{\rightarrow}\sigma} \ \tau \text{ a } \lambda{\rightarrow}\text{-type}$$

2. Church style: $$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\tau.M : \tau{\rightarrow}\sigma} \ \tau \text{ a } \lambda{\rightarrow}\text{-type}$$

Examples:

- $\lambda 2$ à la Curry: $\lambda x.\lambda y.x : \forall\alpha.\forall\beta.\alpha{\rightarrow}\beta{\rightarrow}\alpha$.
- $\lambda 2$ à la Church: $\lambda\alpha.\lambda\beta.\lambda x{:}\alpha.\lambda y{:}\beta.x : \forall\alpha.\forall\beta.\alpha{\rightarrow}\beta{\rightarrow}\alpha$.
- $\lambda 2$ à la Curry: $z : \forall\alpha.\alpha{\rightarrow}\alpha \vdash z\,z : \forall\alpha.\alpha{\rightarrow}\alpha$.
- $\lambda 2$ à la Church: $z : \forall\alpha.\alpha{\rightarrow}\alpha \vdash \lambda\alpha.z\,(\alpha{\rightarrow}\alpha)\,(z\,\alpha) : \forall\alpha.\alpha{\rightarrow}\alpha$.
- But NOT $\vdash \lambda z.z\,z : \ldots$

Making the slogan "all type variables are implicitly universally quantified" precise.

Suppressing $\forall$ in Curry-style $\lambda 2$ with ML-style polymorphism:

Derivation rules and types are the same as for $\lambda\!\rightarrow$ but add a type substitution rule. (We denote derivability in this system with $\vdash_i$)

$$\frac{\Gamma \vdash_i M : \sigma}{\Gamma \vdash_i M : \sigma[\tau/\alpha]}\ \tau \text{ a } \lambda\!\rightarrow \text{ type}$$

Example: $z : \alpha\rightarrow\alpha \vdash z\,z : \alpha\rightarrow\alpha$.

Theorem For Curry-style $\lambda 2$ with ML-style polymorphism:

$$\Gamma \vdash_i M : \sigma \quad \Longrightarrow \quad \Gamma^{+\forall} \vdash M : \forall\vec{\alpha}.\sigma$$

$$|\Gamma|^{-\forall} \vdash_i M : \sigma \quad \Longleftarrow \quad \Gamma \vdash M : \forall\vec{\alpha}.\sigma$$

Where $\Gamma^{+\forall}$ adds $\forall$ and $|\Gamma|^{-\forall}$ removes $\forall$.

Derivation rules of $\lambda 2$ with full (system F-style) polymorphism:

$$\text{Typ} := \text{TVar} \mid (\text{Typ} \rightarrow \text{Typ}) \mid \forall \alpha . \text{Typ}$$

1. Curry style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall \alpha . \sigma} \; \alpha \notin \text{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall \alpha . \sigma}{\Gamma \vdash M : \sigma[\tau / \alpha]} \; \text{for } \tau \text{ any } \lambda 2\text{-type}$$

2. Church style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda \alpha . M : \forall \alpha . \sigma} \; \alpha \notin \text{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall \alpha . \sigma}{\Gamma \vdash M \tau : \sigma[\tau / \alpha]} \; \text{for } \tau \text{ any } \lambda 2\text{-type}$$

- $\forall$ can also occur deeper in a type.
- With full polymorphism, type checking becomes undecidable! [Wells 1993]

Derivation rules of $\lambda 2$ with full (system F-style) polymorphism:

$$\mathsf{Typ} := \mathsf{TVar} \mid (\mathsf{Typ} \to \mathsf{Typ}) \mid \forall \alpha . \mathsf{Typ}$$

NB: In the abstraction rule all types are $\lambda 2$-types:

1. Curry style: $\dfrac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x . M : \tau \to \sigma} \; \sigma, \tau \;\; \lambda 2\text{-types}$
2. Church style: $\dfrac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\tau . M : \tau \to \sigma} \; \sigma, \tau \;\; \lambda 2\text{-types}$

From $\lambda 2$ à la Church to $\lambda 2$ à la Curry: erasure map:

$$
\begin{array}{lcllcl}
|x| & := & x & & & \\
|\lambda x{:}\sigma.M| & := & |\lambda x.M| & |\lambda \alpha.M| & := & |M| \\
|MN| & := & |M|\,|N| & |M\sigma| & := & |M|
\end{array}
$$

Theorem If $\Gamma \vdash M : \sigma$ in $\lambda 2$ à la Church, then $\Gamma \vdash |M| : \sigma$ in $\lambda 2$ à la Curry.

Theorem If $\Gamma \vdash P : \sigma$ in $\lambda 2$ à la Curry, then there is an M such that $|M| \equiv P$ and $\Gamma \vdash M : \sigma$ in $\lambda 2$ à la Church.

Derivation rules of $\lambda 2$ with full (system F-style) polymorphism:

$$\mathsf{Typ} := \mathsf{TVar} \mid (\mathsf{Typ}{\rightarrow}\mathsf{Typ}) \mid \forall\alpha.\mathsf{Typ}$$

1. Curry style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall\alpha.\sigma}\ \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M : \sigma[\tau/\alpha]}\ \text{for } \tau \text{ any } \lambda 2\text{-type}$$

2. Church style:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \lambda\alpha.M : \forall\alpha.\sigma}\ \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash M : \forall\alpha.\sigma}{\Gamma \vdash M\tau : \sigma[\tau/\alpha]}\ \text{for } \tau \text{ any } \lambda 2\text{-type}$$

Examples valid only with full polymorphism:

- $\lambda 2$ à la Curry: $\lambda x.\lambda y.x : (\forall\alpha.\alpha){\rightarrow}\sigma{\rightarrow}\tau$.
- $\lambda 2$ à la Church: $\lambda x{:}(\forall\alpha.\alpha).\lambda y{:}\sigma.x\tau : (\forall\alpha.\alpha){\rightarrow}\sigma{\rightarrow}\tau$.

Recall: Important Properties

$\Gamma \vdash M : \sigma?$	TCP
$\Gamma \vdash M : ?$	TSP
$\vdash ? : \sigma$	TIP

Properties of polymorphic λ-calculus

- TIP is undecidable, TCP and TSP are equivalent & decidable.
-

TCP	à la Church	à la Curry
ML-style	decidable	decidable
System F-style	decidable	undecidable

  With full polymorphism (system F), untyped terms contain too little information to compute the type.

NB: we mainly consider full (system F-style) $\lambda 2$ (mainly à la Church).

Some examples of typing in $\lambda 2$: Abbreviate $\bot := \forall\alpha.\alpha$, $\top := \forall\alpha.\alpha{\to}\alpha$.

- Curry $\lambda 2$: $\lambda x.xx$: $\bot{\to}\bot$
- Church $\lambda 2$: $\lambda x{:}\bot.x(\bot{\to}\bot)x$: $\bot{\to}\bot$.
- Church $\lambda 2$: $\lambda x{:}\bot.\lambda\alpha.x(\alpha{\to}\alpha)(x\alpha)$: $\bot{\to}\bot$.

Exercises:

- Verify that in Church $\lambda 2$: $\lambda x{:}\top.x\top x$: $\top{\to}\top$.
- Verify that in Curry $\lambda 2$: $\lambda x.xx$: $\top{\to}\top$
- Find a type in Curry $\lambda 2$ for $\lambda x.x\,x\,x$
- Find a type in Curry $\lambda 2$ for $\lambda x.(x\,x)(x\,x)$

Formulas-as-types for $\lambda 2$:

There is a formulas-as-types isomorphism between $\lambda 2$ and second order proposition logic, PROP2

Derivation rules of PROP2:

$$\frac{\Gamma \vdash \sigma}{\Gamma \vdash \forall \alpha . \sigma} \alpha \notin \mathsf{FV}(\Gamma) \qquad \frac{\Gamma \vdash \forall \alpha . \sigma}{\Gamma \vdash \sigma[\tau/\alpha]}$$

NB This is constructive second order proposition logic:

$$\forall \alpha . \forall \beta . ((\alpha \rightarrow \beta) \rightarrow \alpha) \rightarrow \alpha \text{ Peirce's law}$$

is not derivable.

Definability of the other connectives:

$$
\begin{aligned}
\bot &:= \forall\alpha.\alpha \\
\sigma\wedge\tau &:= \forall\alpha.(\sigma\to\tau\to\alpha)\to\alpha \\
\sigma\vee\tau &:= \forall\alpha.(\sigma\to\alpha)\to(\tau\to\alpha)\to\alpha \\
\exists\alpha.\sigma &:= \forall\beta.(\forall\alpha.\sigma\to\beta)\to\beta
\end{aligned}
$$

and all the standard constructive derivation rules are derivable.

Example ($\wedge$-elimination):

$$
\dfrac{\dfrac{\forall\alpha.(\sigma\to\tau\to\alpha)\to\alpha}{(\sigma\to\tau\to\sigma)\to\sigma} \qquad \dfrac{\dfrac{[\sigma]^1}{\tau\to\sigma}}{\sigma\to\tau\to\sigma}\,1}{\sigma}
$$

Definability of connectives and derivation rules:

$$
\begin{aligned}
\bot &:= \forall\alpha.\alpha \\
\sigma\wedge\tau &:= \forall\alpha.(\sigma\to\tau\to\alpha)\to\alpha \\
\sigma\vee\tau &:= \forall\alpha.(\sigma\to\alpha)\to(\tau\to\alpha)\to\alpha \\
\exists\alpha.\sigma &:= \forall\beta.(\forall\alpha.\sigma\to\beta)\to\beta
\end{aligned}
$$

Example ($\wedge$-elimination) with λ-terms:

$$
\dfrac{\dfrac{M : \forall\alpha.(\sigma\to\tau\to\alpha)\to\alpha}{M\sigma : (\sigma\to\tau\to\sigma)\to\sigma} \qquad \dfrac{\dfrac{[x : \sigma]^1}{\lambda y{:}\tau.x : \tau\to\sigma}}{\lambda x{:}\sigma.\lambda y{:}\tau.x : \sigma\to\tau\to\sigma}\,1}{M\sigma(\lambda x{:}\sigma.\lambda y{:}\tau.x) : \sigma}
$$

So the following term is a 'witness' for the $\wedge$-elimination.

$$\lambda z{:}\sigma\wedge\tau . z\,\sigma\,(\lambda x{:}\sigma.\lambda y{:}\tau.x) : (\sigma \wedge \tau)\to\sigma$$

Data types in $\lambda 2$

$$\mathsf{Nat} := \forall\alpha.\alpha\to(\alpha\to\alpha)\to\alpha$$

This type can be used as the type of natural numbers, using the encoding of $\mathbb{N}$ as Church numerals in the λ-calculus.

$$n \mapsto c_n := \lambda x.\lambda f.f(\ldots(f x)) \quad n\text{-times } f$$

- $0 := \lambda\alpha.\lambda x{:}\alpha.\lambda f{:}\alpha\to\alpha.x$
- $S := \lambda n{:}\mathsf{Nat}.\lambda\alpha.\lambda x{:}\alpha.\lambda f{:}\alpha\to\alpha.f(n\alpha x f)$
- Iteration: if $c : \sigma$ and $g : \sigma\to\sigma$, then $\mathsf{It}\, c\, g : \mathsf{Nat}\to\sigma$ is defined as

$$\lambda n{:}\mathsf{Nat}.n\,\sigma\, c\, g$$

  Then $\mathsf{It}\, c\, g\, n = g(\ldots(g\, c))$ (n times g), i.e.

$$\mathsf{It}\, c\, g\, 0 = c \quad \text{and} \quad \mathsf{It}\, c\, g\, (S\, x) = g(\mathsf{It}\, c\, g\, x)$$

Why is this a good/useful type for the natural numbers?

- It's the straightforward type for the Church numerals.
- It represents the type of proofs that a number is inductive in second order predicate logic:
  $0 : D, S : D \to D$

  $$N(x) := \forall P. P\, 0 \to (\forall y. P\, y \to P\, (S\, y)) \to P\, x$$

  $N(x)$ iff x is in the smallest 'set' containing 0 and closed under S.
  E.g. $N(0)$, $(N(S\, 0)$, ..., $N(S^p(0))$.
  Stripping all first order information (moving from $\mathrm{PRED2}$ to PROP):

  $$N := \forall P. P \to (P \to P) \to P$$

  The normal proof of $N(S^p(0))$ is the Church numeral c_n under a suitable Curry-Howard embedding.

Examples:

- Addition

$$\text{Plus} := \lambda n{:}\text{Nat}.\lambda m{:}\text{Nat}.\text{It}\, m\, S\, n$$

  or Plus $:= \lambda n{:}\text{Nat}.\lambda m{:}\text{Nat}.n\,\text{Nat}\, m\, S$

- Multiplication

$$\text{Mult} := \lambda n{:}\text{Nat}.\lambda m{:}\text{Nat}.\text{It}\, 0\, (\lambda x{:}\text{Nat}.\text{Plus}\, m\, x)\, n$$

- Predecessor is difficult!

  This requires defining primitive recursion in terms of iteration.

  As a consequence:

$$\text{Pred}(n+1) \twoheadrightarrow_\beta n$$

  in a number of steps of $O(n)$.

Data types in $\lambda 2$ ctd.

$$\text{List}_A := \forall\alpha.\alpha\to(A\to\alpha\to\alpha)\to\alpha$$

represents the type of lists over the type A, using the following encoding of lists in the untyped λ-calculus.

$$[a_1, a_2, \ldots, a_n] \mapsto \lambda x.\lambda f.f a_1 (f a_2 (\ldots (f a_n x))) \quad n\text{-times } f$$

- Nil $:= \lambda\alpha.\lambda x{:}\alpha.\lambda f{:}A\to\alpha\to\alpha.x$
- Cons $:= \lambda a{:}A.\lambda l{:}\text{List}_A.\lambda\alpha.\lambda x{:}\alpha.\lambda f{:}A\to\alpha\to\alpha.f\, a(l\,\alpha\, x\, f)$
- Iteration: if $c : \sigma$ and $g : A\to\sigma\to\sigma$, then $\text{It}\, c\, g : \text{List}_A\to\sigma$ is def. as

$$\lambda l{:}\text{List}_A.l\,\sigma\, c\, g$$

Then, for $l = [a_1, \ldots, a_n]$, $\text{It}\, c\, g\, l = g\, a_1(\ldots(g\, a_n\, c))$ (n times g) i.e.

$$\text{It}\, c\, g\, \text{Nil} = c \quad \text{and} \quad \text{It}\, c\, g\, (\text{Cons}\, a\, l) = g\, a\, (\text{It}\, c\, g\, l)$$

Example:

- Map, given $f : \sigma \rightarrow \tau$, $\mathsf{Map}\, f : \mathsf{List}_\sigma \rightarrow \mathsf{List}_\tau$ applies f to all elements in a list.

$$\mathsf{Map} := \lambda f{:}\sigma\rightarrow\tau.\mathsf{It}\,\mathsf{Nil}(\lambda x{:}\sigma.\lambda l{:}\mathsf{List}_\tau.\mathsf{Cons}(f\ x)l).$$

Then

$$\begin{aligned}
\mathsf{Map}\, f\, \mathsf{Nil} &= \mathsf{Nil} \\
\mathsf{Map}\, f\, (\mathsf{Cons}\, a\, k) &= \mathsf{It}\,\mathsf{Nil}(\lambda x{:}\sigma.\lambda l{:}\mathsf{List}_\tau.\mathsf{Cons}(f\ x)l)\, (\mathsf{Cons}\, a\, k) \\
&= (\lambda x{:}\sigma.\lambda l{:}\mathsf{List}_\tau.\mathsf{Cons}(f\ x)l)a(\mathsf{Map}\, f\ k) \\
&= \mathsf{Cons}(f\ a)(\mathsf{Map}\, f\ k)
\end{aligned}$$

Many data-types can be defined in $\lambda 2$:

- Product of two data-types: $\sigma \times \tau := \forall \alpha.(\sigma \to \tau \to \alpha) \to \alpha$
- Sum of two data-types: $\sigma + \tau := \forall \alpha.(\sigma \to \alpha) \to (\tau \to \alpha) \to \alpha$
- Unit type: Unit $:= \forall \alpha.\alpha \to \alpha$
- Binary trees with nodes in A and leaves in B:
  $\mathsf{Tree}_{A,B} := \forall \alpha.(B \to \alpha) \to (A \to \alpha \to \alpha) \to \alpha$

Exercise:

- Define inl : $\sigma \to \sigma + \tau$
- Define the first projection: $\pi_1 : \sigma \times \tau \to \sigma$
- Define join : $\mathsf{Tree}_{A,B} \to \mathsf{Tree}_{A,B} \to A \to \mathsf{Tree}_{A,B}$

Properties of $\lambda 2$.

- Uniqueness of types
  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma = \tau$.
- Subject Reduction
  If $\Gamma \vdash M : \sigma$ and $M \longrightarrow_{\beta\eta} N$, then $\Gamma \vdash N : \sigma$.
- Strong Normalization
  If $\Gamma \vdash M : \sigma$, then all $\beta\eta$-reductions from M terminate.

Strong Normalization of β for $\lambda 2$.

Note:

- There are two kinds of β-reductions
  - $(\lambda x{:}\sigma.M)P \longrightarrow_\beta M[P/x]$
  - $(\lambda \alpha.M)\tau \longrightarrow_\beta M[\tau/\alpha]$
- The second doesn't do any harm, so we can just look at $\lambda 2$ à la Curry

Recall the proof for $\lambda{\rightarrow}$:

- $[\![\alpha]\!] :=$ SN.
- $[\![\sigma{\rightarrow}\tau]\!] := \{M \mid \forall N \in [\![\sigma]\!](MN \in [\![\tau]\!])\}$.

Question:

How to define $[\![\forall\alpha.\sigma]\!]$??

$$[\![\forall\alpha.\sigma]\!] := \Pi_{X \in U} [\![\sigma]\!]_{\alpha:=X} ??$$

Strong Normalization of β for $\lambda 2$.

Question:

How to define $[\![\forall\alpha.\sigma]\!]$??

$$[\![\forall\alpha.\sigma]\!] := \Pi_{X \in U} [\![\sigma]\!]_{\alpha:=X} ??$$

- What should be U?

  The collection of "all possible interpretations" of types (?)

- $\Pi_{X \in U} [\![\sigma]\!]_{\alpha:=X}$ gets too big: $\mathsf{card}(\Pi_{X \in U} [\![\sigma]\!]_{\alpha:=X}) > \mathsf{card}(U)$

Girard:

- $[\![\forall\alpha.\sigma]\!]$ should be small

$$\bigcap_{X \in U} [\![\sigma]\!]_{\alpha:=X}$$

- Characterization of U.

U := SAT, the collection of saturated sets of (untyped) λ-terms.
$X \subset \Lambda$ is saturated if

- $xP_1 \ldots P_n \in X$ (for all $x \in$ Var, $P_1, \ldots, P_n \in$ SN)
- $X \subseteq$ SN
- If $M[N/x]\vec{P} \in X$ and $N \in$ SN, then $(\lambda x.M)N\vec{P} \in X$.

Let ρ : TVar $\rightarrow$ SAT be a valuation of type variables.
Define the interpretation of types $[\![\sigma]\!]_\rho$ as follows.

- $[\![\alpha]\!]_\rho := \rho(\alpha)$
- $[\![\sigma{\rightarrow}\tau]\!]_\rho := \{M | \forall N \in [\![\sigma]\!]_\rho (MN \in [\![\tau]\!]_\rho)\}$
- $[\![\forall\alpha.\sigma]\!]_\rho := \cap_{X \in \mathsf{SAT}} [\![\sigma]\!]_{\rho,\alpha:=X}$

Proposition

$$x_1 : \tau_1, \ldots, x_n : \tau_n \vdash M : \sigma \Rightarrow M[P_1/x_1, \ldots, P_n/x_n] \in [\![\sigma]\!]_\rho$$

for all valuations ρ and $P_1 \in [\![\tau_1]\!]_\rho, \ldots, P_n \in [\![\tau_n]\!]_\rho$

Proof

By induction on the derivation of $\Gamma \vdash M : \sigma$.

Corollary $\lambda 2$ is SN

(Proof: take P_1 to be x_1, ..., P_n to be x_n.)

A little bit on semantics

$\lambda 2$ does not have a set-theoretic model! [Reynolds]

Theorem: If

$$[\![\sigma \to \tau]\!] := [\![\tau]\!]^{[\![\sigma]\!]} \quad \text{(set theoretic function space)}$$

then $[\![\sigma]\!]$ is a singleton set for every σ.

So: in a $\lambda 2$-model, $[\![\sigma \to \tau]\!]$ must be ‘small’.