Proofs as Programs Summer School
Eugene Oregon June - July 2002

Type Systems

Herman Geuvers
Nijmegen University, NL

Lecture 2: 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 of $\lambda 2$, two styles:

1. Weak (ML-style) polymorphism:

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

2. Full (system F-style) polymorphism:

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

NB: (1) is presented à la Curry, and (2) is presented à la Church (but that could be done otherwise).

Derivation rules of $\lambda 2$, two styles:

1. Weak (ML-style) polymorphism:

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

2. Full (system F-style) polymorphism:

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

NB: (1) is presented à la Curry, and (2) is presented à la Church (but that could be done otherwise).

Examples valid in both (1) and (2):

- $\lambda 2$ à la Curry: $\lambda x.\lambda y.x : \forall \alpha.\forall \beta.\alpha{\to}\beta{\to}\alpha$.
- $\lambda 2$ à la Church: $\lambda \alpha.\lambda \beta.\lambda x{:}\alpha.\lambda y{:}\beta.x : \forall \alpha.\forall \beta.\alpha{\to}\beta{\to}\alpha$.

Derivation rules of $\lambda 2$, two styles:

1. Weak (ML-style) polymorphism:

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

2. Full (system F-style) polymorphism:

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

Examples valid only in (2):

- $\lambda 2$ 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

$$\Gamma \vdash M : \sigma? \quad \text{TCP}$$
$$\Gamma \vdash M : ? \quad \text{TSP}$$
$$\Gamma \vdash ? : \sigma \quad \text{TIP}$$

Properties of $\lambda 2$

- TIP is undecidable, TCP and TSP are equivalent.

-

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.x

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

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.

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

- $0 := \lambda\alpha.\lambda x{:}\alpha.\lambda f{:}\alpha\to\alpha.x$
- $S := \lambda n{:}\mathsf{Nat}.\lambda\alpha.\lambda x{:}\alpha.\lambda f{:}\alpha\to\alpha.f(n\alpha x f)$
- 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

  $$\mathsf{It}\,c\,g\,n = g(\ldots(g c)) \;\; (n \text{ times } g)$$

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.

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

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

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

Then, for $l = [a_1, a_2, \ldots, a_n]$,

$$\text{It}\,c\,g\,l = g\,a_1(g\,a_2(\ldots(g\,a_n\,c))) \;\; (n \text{ times } g)$$

Example:

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

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

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

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

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 $\lambda{\rightarrow}$:

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

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 is U?
  The collection of all 'possible' interpretations of types (?)
- $\Pi_{X\in U}[\![\sigma]\!]_{\alpha:=X}$ may get very (too?) big.

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

- Var $\subseteq X$
- $X \subseteq$ SN
- If $M[N/x] \in X$ and $N \in$ SN, then $(\lambda x.M)N \in X$.

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 $P_1 \in [\![\tau_1]\!]_\rho, \ldots, P_n \in [\![\tau_n]\!]_\rho$

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 $P_1 \in [\![\tau_1]\!]_\rho, \ldots, P_n \in [\![\tau_n]\!]_\rho$

Corollary $\lambda 2$ is SN

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