

Proofs as Programs Summer School  
Eugene Oregon June - July 2002

Type Systems

Herman Geuvers

Nijmegen University, NL

Lecture 2: Polynomial  $\lambda$ -calculus

## Why Polymorphic $\lambda$ -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:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \text{Va}.\sigma} \quad \text{for } \tau \text{ a } \lambda \rightarrow \text{-type}$$

2. Full (system F-style) polymorphism:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash \lambda\alpha.M : \text{Va}.\sigma}{\Gamma \vdash M : \sigma} \quad \text{for } \tau \text{ any type}$$

**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:

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall \alpha. \sigma} \quad \alpha \notin \text{FV}(\Gamma) \quad \text{for } \tau \text{ a } \lambda \rightarrow \text{-type}$$

2. Full (system F-style) polymorphism:

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \forall \alpha. \sigma}{\Gamma \vdash M : \sigma} \quad \text{for } \tau \text{ any type}$$

**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:  $\lambda x. \lambda y. x : \forall \alpha. \forall \beta. \alpha \rightarrow \beta \rightarrow \alpha$ .
- $\lambda_2$  à la Church:  $\lambda \alpha. \lambda \beta. \lambda x : \alpha. \lambda y : \beta. x : \forall \alpha. \forall \beta. \alpha \rightarrow \beta \rightarrow \alpha$ .

Derivation rules of  $\lambda_2$ , two styles:

1. Weak (ML-style) polymorphism:

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

2. Full (system F-style) polymorphism:

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

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

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

## Properties of $\lambda_2$

- TIP is **undecidable**, TCP and TSP are equivalent.

TCP	$\lambda$ à la Church à la Curry	decidable	decidable
• ML-style		decidable	<b>undecidable</b>
System F-style		decidable	<b>undecidable</b>

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:**

$$\frac{\Gamma \vdash \sigma \quad \Gamma \vdash \text{Va}.\sigma}{\Gamma \vdash \text{Va}.\sigma} \quad \text{a} \notin \text{FV}(\Gamma) \quad \frac{\Gamma \vdash \sigma[\tau/\text{a}]}{\Gamma \vdash \sigma}$$

**NB** This is **constructive** second order proposition logic:

$\text{Va}.\text{V}\beta.((\text{a} \rightarrow \beta) \rightarrow \text{a}) \rightarrow \text{a}$  Peirce's law

is not derivable.

$$\frac{\frac{\frac{\sigma \leftarrow \tau \leftarrow \sigma}{\sigma \leftarrow \tau} \quad \tau \leftarrow \sigma}{\sigma \leftarrow \tau} \quad \tau \leftarrow \sigma}{\sigma \leftarrow \tau} \quad \frac{\sigma}{\sigma \leftarrow \tau} \quad (\sigma \leftarrow \tau \leftarrow \sigma)}{\forall \alpha. (\sigma \leftarrow \tau \leftarrow \alpha) \leftarrow \alpha}$$

Example ( $\forall$ -elimination):

and all the standard constructive derivation rules are derivable.

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

Definability of the other connectives:

Definability of connectives and derivation rules:

$$\top := \forall \alpha. \alpha$$

$$\sigma \vee \tau := \forall \alpha. (\sigma \rightarrow \tau \rightarrow \alpha) \rightarrow \alpha$$

$$\sigma \wedge \tau := \forall \alpha. (\sigma \rightarrow \alpha) \rightarrow (\tau \rightarrow \alpha) \rightarrow \alpha$$

$$\exists \alpha. \sigma := \forall \beta. (\forall \alpha. \sigma \rightarrow \beta) \rightarrow \beta$$

Example ( $\vee$ -elimination) with  $\lambda$ -terms:

$$\frac{\frac{\frac{M : \forall \alpha. (\sigma \rightarrow \tau \rightarrow \alpha) \rightarrow \alpha}{M\sigma : (\sigma \rightarrow \tau \rightarrow \sigma) \rightarrow \sigma} \quad \frac{\frac{\lambda y : \tau. x : \tau \rightarrow \sigma}{\lambda x : \sigma. \lambda y : \tau. x : \sigma \rightarrow \tau \rightarrow \sigma}}{[x : \sigma]_1}}{M\sigma(\lambda x : \sigma. \lambda y : \tau. x) : \sigma}}{1}$$

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

$$\lambda z : \sigma \vee \tau. z \sigma (\lambda x : \sigma. \lambda y : \tau. x) : (\sigma \vee \tau) \rightarrow \sigma$$

## Data types in $\lambda 2$

$$\text{Nat} := \forall \alpha. \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$$

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

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

$$\bullet \text{ } 0 := \lambda \alpha. \lambda x: \alpha. \lambda f: \alpha \rightarrow \alpha. x$$

$$\bullet \text{ } S := \lambda n: \text{Nat}. \lambda \alpha. \lambda x: \alpha. \lambda f: \alpha \rightarrow \alpha. f(n \alpha x)$$

• **Iteration**: if  $c : \sigma$  and  $g : \sigma \rightarrow \sigma$ , then **It c g** :  $\text{Nat} \rightarrow \sigma$  is

defined as

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

Then

$$\text{It c g } n = g(\dots(g c) \dots)$$

Examples:

- Addition

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

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

- 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:

$$\text{Pred}(n + 1) \dashv\vdash \mathcal{E} \ n$$

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

Data types in  $\lambda 2$  ctd.

$$\text{List } A := \forall a. a \rightarrow (A \rightarrow a \rightarrow a) \rightarrow a$$

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

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

$$\bullet \text{Nil} := \lambda a. \lambda x. a. \lambda f. A \rightarrow a \rightarrow a. x$$

$$\bullet \text{Cons} := \lambda a: A. \lambda l: \text{List } A. \lambda \alpha. \lambda x: \alpha. \lambda f: A \rightarrow a \rightarrow \alpha. f a (l \alpha x f)$$

• **Iteration**: if  $c : \sigma$  and  $g : A \rightarrow \sigma \rightarrow \sigma$ , then **It c g** :  $\text{List } A \rightarrow \sigma$  is

defined as

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

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

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

Example:

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

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

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

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

## 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  $M \rightarrow^{\beta\eta} N$ , then  $\Gamma \vdash N : \sigma$ .
- **Strong Normalization**  
If  $\Gamma \vdash M : \sigma$ , then all  $\beta\eta$ -reductions from  $M$  terminate.

**Strong Normalization** of  $\beta$  for  $\lambda_2$ .

**Note:**

- There are two kinds of  $\beta$ -reductions

$$\begin{aligned} & - (\lambda x:\sigma.M)P \rightarrow_{\beta} M[P/x] \\ & - (\lambda\alpha.M)\tau \rightarrow_{\beta} M[\tau/\alpha] \end{aligned}$$

- The second doesn't do any harm, so we can just look at  $\lambda_2$

**à la Curry**

Recall the proof for  $\lambda \rightarrow$ :

$$\bullet \llbracket \alpha \rrbracket := \text{Term}(\alpha) \cup \text{SN}$$

$$\bullet \llbracket \sigma \rightarrow \tau \rrbracket := \{M : \sigma \rightarrow \tau \mid \forall N \in \llbracket \sigma \rrbracket (MN \in \llbracket \tau \rrbracket)\}.$$

**Question:**

How to define  $\llbracket \forall\alpha.\sigma \rrbracket$  ??

$$\text{?? } X^{=: \alpha} \llbracket \sigma \rrbracket \cup X^{\exists} \llbracket \sigma \rrbracket := \llbracket \forall\alpha.\sigma \rrbracket$$

Strong Normalization of  $\beta$  for  $\lambda 2$ .

Question:

How to define  $\llbracket \forall a.\sigma \rrbracket$  ??

$$\llbracket \forall a.\sigma \rrbracket := \prod_{X \in U} \llbracket \sigma \rrbracket_{\alpha := X} \quad ??$$

• What is  $U$ ?

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

- $\prod_{X \in U} \llbracket \sigma \rrbracket_{\alpha := X}$  may get very (too?) big.

Girard:

- $\llbracket \forall a.\sigma \rrbracket$  should be **small**

$$\bigcup_{X \in U} \llbracket \sigma \rrbracket_{\alpha := X}$$

- Characterization of  $U$ .

$U := \text{SAT}$ , the collection of **saturated sets** of (untyped)  $\lambda$ -terms.  
 $X \subseteq \Lambda$  is **saturated** if

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

Let  $\rho : \text{TVar} \rightarrow \text{SAT}$  be a **valuation** of type variables.  
**Define** the interpretation of types  $[\sigma]^\rho$  as follows.

- $[\alpha]^\rho := \rho(\alpha)$
- $[\sigma \rightarrow \tau]^\rho := \{M \mid \text{AN} \in [\sigma]^\rho, (MN)^\rho \in [\tau]^\rho\}$
- $[\forall \alpha. \sigma]^\rho := \bigcup_{X \in \text{SAT}} [\sigma]^\rho, \alpha := X$

## Proposition

$x_1 : T_1, \dots, x_n : T_n \vdash M : \sigma \Rightarrow M[P_1/x_1, \dots, P_n/x_n] \in \llbracket \sigma \rrbracket_\rho$   
for all valuations  $\rho$  and  $P_1 \in \llbracket T_1 \rrbracket_\rho, \dots, P_n \in \llbracket T_n \rrbracket_\rho$

## Proposition

$x_1 : T_1, \dots, x_n : T_n \vdash M : \sigma \Rightarrow M[P_1/x_1, \dots, P_n/x_n] \in \llbracket \sigma \rrbracket_\rho$   
for all valuations  $\rho$  and  $P_1 \in \llbracket T_1 \rrbracket_\rho, \dots, P_n \in \llbracket T_n \rrbracket_\rho$

Corollary  $\lambda 2$  is SN

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