Introduction to Barendregt's Lambda Cube

Silvia Ghilezan

University of Novi Sad
Mathematical Institute SASA, Serbia

Lecture 2

Oregon Programming Language Summer School
Eugene, June 2023

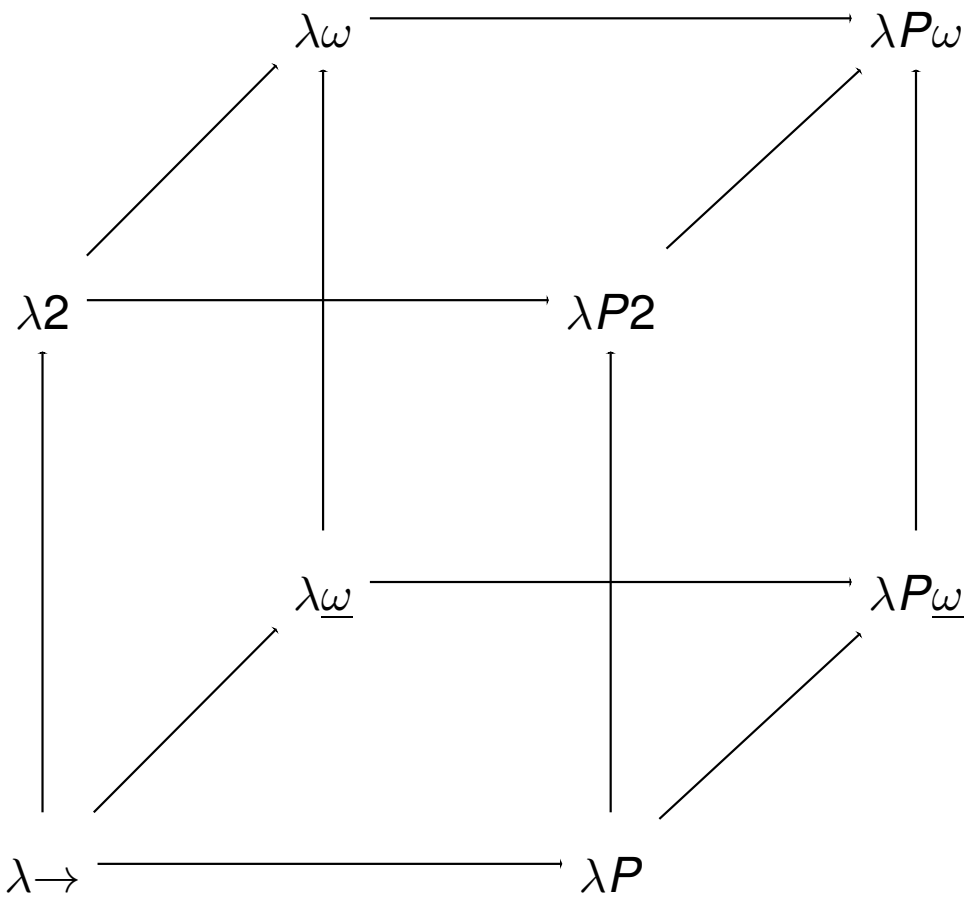

Figure: Lambda cube

- λ Untyped Lambda calculus
- $\lambda\!\rightarrow$ Simple types
- $\lambda 2$ Polymorphic types
- $\lambda\underline{\omega}$
- λP Dependent types
- Lambda cube
- Logic cube

Roadmap

$\lambda 2$ Polymorphic types

Dependency

Weak $\lambda\omega$

Roadmap

$\lambda 2$ Polymorphic types

Dependency

Weak $\lambda\omega$

$\lambda 2$ Polymorphic types - Why?

Problem:

- expressiveness of $\lambda \rightarrow$ is very limited
- in $\lambda \rightarrow$ a function is bound to its type and it cannot be reused
- $\lambda x : \alpha . x : \alpha \rightarrow \alpha$ and $\lambda x : \beta . x : \beta \rightarrow \beta$

Solution:

- define functions that treat types polymorphically
- add types $\forall \alpha . \sigma$
- $M : \forall \alpha . (\alpha \rightarrow \alpha)$, then M can map any type to itself.

$\lambda 2$ ($\mathcal{F}$) language

Terms and types

1970s

- Girard system $\mathcal{F}$
- Reynolds language FORSYTHE

Language

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{C} \mid (\mathcal{T}\mathcal{T}) \mid \lambda x : \text{Type}.\mathcal{T} \mid \lambda\alpha.\mathcal{T} \mid (\mathcal{T}\,\text{Type})$$

$$\text{Type} := \text{TVar} \mid (\text{Type} \to \text{Type}) \mid \forall\alpha.\text{Type}$$

Reductions

There are two kinds of β-reductions

- $(\lambda x : \sigma.M)P \longrightarrow_\beta M[P/x]$
- $(\lambda\alpha.M)\tau \longrightarrow_\beta M[\tau/\alpha]$

$\lambda 2$ type system

$\lambda 2$ type system = $\lambda \rightarrow$ extended with

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

Example

- K = $\lambda x.\lambda y.x$
  $\lambda\alpha.\lambda\beta.\lambda x:\alpha.\lambda y:\beta.x : \forall\alpha.\forall\beta.\alpha \rightarrow \beta \rightarrow \alpha.$
- K = $\lambda x.\lambda y.x$
  $\lambda x:(\forall\alpha.\alpha).\lambda y:\sigma.x\tau : (\forall\alpha.\alpha) \rightarrow \sigma \rightarrow \tau.$
- $\lambda x.xx$
  $\lambda x:(\forall\alpha.\alpha).x(\sigma \rightarrow \tau)(x\,\sigma) : (\forall\alpha.\alpha) \rightarrow \tau$

I. $\lambda 2$ Fundamental properties

Properties of $\lambda 2$.

- Uniqueness of types
  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma = \tau$

- Church-Rosser property holds in $\lambda 2$

- Subject Reduction
  If $\Gamma \vdash M : \sigma$ and $M \longrightarrow_{\beta\eta} N$, then $\Gamma \vdash N : \sigma$

- Strong Normalisation
  If $\Gamma \vdash M : \sigma$, then $M \in \mathrm{SN}$ i.e. all $\beta\eta$-reductions from M terminate

II. $\lambda 2$ Strong Normalisation

Strong Normalisation $\lambda 2$[Girard, Reynolds, 1970]
If $\Gamma \vdash M : \sigma$, then $M \in \mathrm{SN}$

Gödel's first incompleteness theorem (1931)

- by Gödel's first incompleteness theorem there are true statement that could be stated in the language of arithmetic, but not proved in Peano arithmetic
- Gödel sentence: "true but not provable"
- SN $\lambda 2$ (1970)
  - this is a result expressible in Peano arithmetic but not provable in this system. It is however provable in slightly stronger systems
  - this result is a "natural" example of a Gödel sentence (a true statement about termination that could be stated in the language of arithmetic, but not proved in Peano arithmetic)
- Paris-Harington theorem (1977)
  - states that a certain combinatorial principal in Ramsey theory (the strengthened finite Ramsey theorem)
  - this result is also a "natural" example of a Gödel sentence

III. $\lambda 2$ expressiveness

Data types in $\lambda 2$

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

Church numerals

$$\underline{n} \equiv \lambda x.\lambda f.f(\cdots(fx)) \;\; n\text{-times } f$$

- $\underline{0} \equiv \lambda\alpha.\lambda x:\alpha.\lambda f:\alpha \to \alpha.x$: **Nat**
- $\underline{n} \equiv \lambda\alpha.\lambda x:\alpha.\lambda f:\alpha \to \alpha.f^n x$: **Nat**
- $Succ \equiv \lambda n:\mathsf{Nat}.\lambda\alpha.\lambda x:\alpha.\lambda f:\alpha \to \alpha.f(n\alpha xf)$
- It*er*ation: if $c:\sigma$ and $g : \sigma \to \sigma$, then define It $c\ g$: Nat $\to \sigma$ as

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

Then

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

Exercise. Define $+, \times, \ldots$ using iteration.

III. $\lambda 2$ expressiveness

Theorem

$\lambda 2$ types exactly all primitive recursive functions

Almost all (meaningful) programs are typable in $\lambda 2$

Ackerman-Péter function

$$\begin{aligned} \mathrm{A}(0, n) &= n + 1 \\ \mathrm{A}(m+1, 0) &= \mathrm{A}(m, 1) \\ \mathrm{A}(m+1, n+1) &= \mathrm{A}(m, \mathrm{A}(m+1, n)) \end{aligned}$$

- recursive but not primitive recursive
- not typable in $\lambda 2$

IV. $\lambda 2$ logic PROP2

Curry-Howard: formulas-as-types, proofs-as-terms

PROP2 = PROP extended with

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

Constructive second order proposition logic

$$\not\vdash M : \forall\alpha.\forall\beta.((\alpha \to \beta) \to \alpha) \to \alpha \text{ Peirce's law}$$

NB Peirce's law is not inhabited.

Theorem (Girard, Reynolds, Curry-Howard)

$$\vdash \sigma\ (\mathit{PROP2}) \quad \Leftrightarrow \quad \vdash M : \sigma\ (\lambda 2)$$

IV. $\lambda 2$ logic PROP2

Definability of the other connectives in PROP2:

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

NB

- In constructive propositional logic connectives are independent
- PROP is minimal logic - implicational fragment of constructive propositional logic

$\lambda 2$: 3 Type?

Theorem

In $\lambda 2$

- *Type checking ($(M : \sigma)?$) is undecidable*
- *Type inference ($M :?$) is undecidable*
- *Type inhabitation ($? : \sigma$) is undecidable*

Proof.

- Type inhabitation is equivalent to provability in *PROP*2, by the Curry-Howard correspondence
- Type checking and types inference were open questions till 1990, Wells

□

A restriction of $\lambda 2$ has an easy type inference algorithm (Hindley-Milner) and is used for many statically typed functional programming languages such as Haskell and the ML family

Roadmap

$\lambda 2$ Polymorphic types

Dependency

Weak $\lambda\omega$

Dependency

Terms and types are mutually dependent

terms depending on terms	$\lambda \rightarrow$
terms depending on types	$\lambda 2$
types depending on types	$\lambda \underline{\omega}$
types depending on terms	λP

1. terms depending on terms - $\lambda \rightarrow$

$$M : \sigma \rightarrow \tau \quad N : \sigma \quad \Rightarrow \quad MN : \tau \qquad (app)$$

$$\lambda x : \sigma . Mx : \sigma \rightarrow \tau \qquad (abstr)$$

Dependency - $\star$

2. terms depending on types - $\lambda 2$

$$P : \forall\alpha.\alpha \to \alpha \quad \sigma \text{ a type} \quad \Rightarrow \quad P\sigma : \sigma \to \sigma \qquad (app)$$

$$\lambda\alpha.P\alpha : \forall\alpha.\alpha \to \alpha \qquad (abstr)$$

$\sigma \in$ Type (informal notation) becomes $\sigma : \star$ (formal)

$$P : \forall\alpha.\alpha \to \alpha \quad \sigma : \star \quad \Rightarrow \quad P\sigma : \sigma \to \sigma$$

$$\lambda\alpha : \star.P\alpha : \forall\alpha.\alpha \to \alpha$$

Dependency - type operators

3. types depending on types - $\lambda\underline{\omega}$

Informal statement

$$\alpha, \beta \in \text{Type} \quad \Rightarrow \quad \alpha \to \beta \in \text{Type}$$

becomes formal operation on types

$$\alpha : \star, \beta : \star \vdash \alpha \to \beta : \star$$

- $\sigma \to \sigma$ depends on σ
- define $f \equiv \lambda\alpha : \star.\alpha \to \alpha$ s.t. $f(\sigma) = \sigma \to \sigma$
- where does f live?

$$f : \star \qquad \textit{NO}$$

$$f : \star \to \star$$

Dependency - type operators

3. types depending on types - $\lambda\underline{\omega}$

$\mathcal{K} = \{\star, \star \to \star, \ldots\}$ kinds

$k \in \mathcal{K}$ (informal notation) becomes $k : \Box$ (formal)

- $\sigma \in \mathcal{T}$ then $\sigma : \star$
- if $k : \Box$ and $f : k$ then f is the constructor of kind k
- types and terms of $\lambda\underline{\omega}$ can be kept separate

Roadmap

$\lambda 2$ Polymorphic types

Dependency

Weak $\lambda\omega$

$\lambda\underline{\omega}$ - Weak $\lambda\omega$

Terms and types

- Pseudo-expressions: terms and types

$$\mathcal{T} ::= \mathcal{V} \mid \mathcal{C} \mid \mathcal{T}\mathcal{T} \mid \lambda\mathcal{V} : \mathcal{T}.\mathcal{T} \mid \mathcal{T} \to \mathcal{T}$$

- $\{\star, \Box\} = \mathcal{S} \subseteq \mathcal{C}$ sorts
- modification of statements $M : A$, where both $M, A \in \mathcal{T}$
- bases linearly ordered $\Gamma =< x_1 : A_1, \ldots, x_n : A_n >$

$$\begin{array}{rcll} \alpha : \star, x : \alpha & \vdash & x : \alpha & \\ \alpha : \star & \vdash & \lambda x : \alpha.x : \alpha \to \alpha & \text{YES} \end{array}$$

$$\begin{array}{rcll} x : \alpha, \alpha : \star & \vdash & x : \alpha & \\ x : \alpha & \vdash & \lambda\alpha : \star.x : \star \to \alpha & \text{NO} \end{array}$$

$\lambda\underline{\omega}$ typing rules

(ax/sort) $\quad \vdash \star : \Box$

(var) $\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$ if $x \notin \Gamma$

(weak) $\dfrac{\Gamma \vdash M : B \quad \Gamma \vdash A : s}{\Gamma, x : A \vdash M : B}$ if $x \notin \Gamma$

(λ) $\dfrac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash A \to B : s}{\Gamma \vdash \lambda x : A.M : A \to B}$

(app) $\dfrac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$

(conv$_\beta$) $\dfrac{\Gamma \vdash M : A \quad \Gamma \vdash B : s}{\Gamma \vdash M : B}\ A =_\beta B$

(type/kind) $\dfrac{\Gamma \vdash A : s \quad \Gamma \vdash B : s}{\Gamma \vdash A \to B : s}$

$\lambda\underline{\omega}$ examples

Example (1)

$$\dfrac{\dfrac{\alpha : \star, \beta : \star \quad \vdash \quad (\alpha \to \beta) : \star}{\alpha : \star, \beta : \star, x : (\alpha \to \beta) \quad \vdash \quad x : (\alpha \to \beta)}}{\alpha : \star, \beta : \star \quad \vdash \quad \lambda x : (\alpha \to \beta).x : ((\alpha \to \beta) \to (\alpha \to \beta))}$$

$$\not\forall\ \lambda\alpha : \star.\lambda\beta : \star.RHS : \star \to \star \to ((\alpha \to \beta) \to (\alpha \to \beta)) \qquad \text{why?}$$

Example (2)

$$x : (\alpha \to \beta) \quad \not\forall \quad x : \alpha \to \beta \qquad \text{why?}$$

Example (Exercise)

$D \equiv \lambda\beta : \star.(\beta \to \beta)$

$$\dfrac{\vdash \quad D : \star \to \star}{\alpha : \star \quad \vdash \quad \lambda D\alpha.x : D(D\alpha)}$$

I. $\lambda\underline{\omega}$ Fundamental properties

Properties of $\lambda\underline{\omega}$.

- Uniqueness of types
  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma = \tau$
- Church-Rosser property holds in $\lambda\underline{\omega}$
- Subject Reduction
  If $\Gamma \vdash M : \sigma$ and $M \longrightarrow_{\beta\eta} N$, then $\Gamma \vdash N : \sigma$
- Strong Normalisation
  If $\Gamma \vdash M : \sigma$, then $M \in \mathrm{SN}$ i.e. all $\beta\eta$-reductions from M terminate

II. $\lambda\underline{\omega}$ expressiveness

Theorem

1. $\lambda\underline{\omega}$ *is of same expressive power as* $\lambda \rightarrow$
2. $\lambda\underline{\omega}$ *types exactly all extended polynomials*