CENG 2010 - Programming Language Concepts

Week 12: Polymorphically Typed λ-Calculus (System-F)

Burak Ekici

May 29, 2023

Outline

Need for Polymorphic Types

- In $\lambda^{\rightarrow}$ each function works exactly for one type

Need for Polymorphic Types

- In $\lambda^{\rightarrow}$ each function works exactly for one type
- E.g., the monomorphic identity function, $\mathtt{id} = \lambda x\colon \mathtt{int}.\, x\colon \mathtt{int} \rightarrow \mathtt{int}$

Need for Polymorphic Types

- In $\lambda^{\rightarrow}$ each function works exactly for one type
- E.g., the monomorphic identity function, $\texttt{id} = \lambda x\colon \texttt{int}.\, x\colon \texttt{int} \rightarrow \texttt{int}$
- Use a more flexible type system that lets us write only one identity that works for every type function

Need for Polymorphic Types

- In $\lambda^{\rightarrow}$ each function works exactly for one type
- E.g., the monomorphic identity function, $\texttt{id} = \lambda x{:}\ \texttt{int}.\ x{:}\ \texttt{int} \rightarrow \texttt{int}$
- Use a more flexible type system that lets us write only one identity that works for every type function
- E.g., the polymorphic identity function, $\texttt{id} = \lambda x{:}\ \tau.\ x{:}\ \tau \rightarrow \tau$

Definition (Polymorphism)

- A function is polymorphic if it can be applied to “many” types of arguments

Definition (Polymorphism)

- A function is polymorphic if it can be applied to "many" types of arguments
- Various kinds of polymorphism depending on the definition of "many"

Definition (Polymorphism)

- A function is polymorphic if it can be applied to “many” types of arguments
- Various kinds of polymorphism depending on the definition of “many”

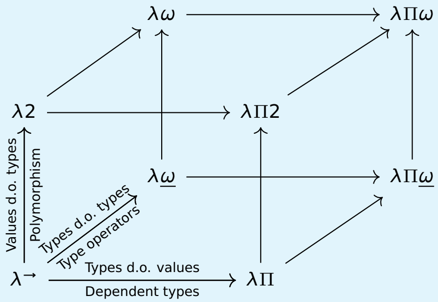
ad-hoc polymorphism “many” = function behavior chosen at runtime

Definition (Polymorphism)

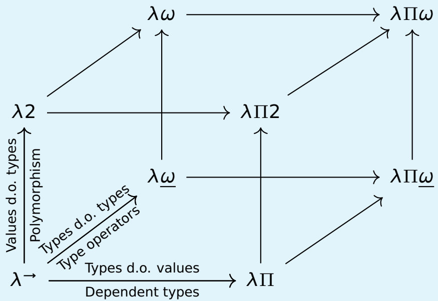
- A function is polymorphic if it can be applied to “many” types of arguments
- Various kinds of polymorphism depending on the definition of “many”

ad-hoc polymorphism	“many”	=	function behavior	chosen at runtime
parametric polymorphism	“many”	=	types	type variables decorate function signatures

Lambda (Barendregt) Cube



Lambda (Barendregt) Cube



Lambda (Barendregt) Cube

$$
\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda\Pi\omega \\
\lambda 2 \arrow[ur] \arrow[rr] & & \lambda\Pi 2 \arrow[ur] & \\
 & \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda\Pi\underline{\omega} \arrow[uu] \\
\lambda^{\to} \arrow[uu, "\text{Values d.o. types}", "\text{Polymorphism}"'] \arrow[ur, "\text{Types d.o. types}", "\text{Type operators}"'] \arrow[rr, "\text{Types d.o. values}", "\text{Dependent types}"'] & & \lambda\Pi \arrow[uu] \arrow[ur] &
\end{tikzcd}
$$

Outline

Definition (Polymorphically Typed λ-Calculus (System-F))

Extends simply-typed lambda-calculus with the ability to

- abstract over a type variable (type generalization)
- and to apply such an abstraction to a type (type specialization)

Types $A, B, C, \ldots$:=

|	$G, G', G'', \ldots$	"ground" types
|	unit	unit type
|	$A \times B$	product type
|	$A \to B$	function type
|	τ	variable (countably many)
|	$\forall\tau. A$	type quantifier

Terms s, t, r :=

|	c^A	constants (of given type A)
|	x	variable (countable many)
|	$()$	unit value
|	(s, t)	pair
|	$\mathsf{fst}\ t$	first pair projection
|	$\mathsf{snd}\ t$	second pair projection
|	$\lambda x : A. t$	function abstraction
|	$s\ t$	function application
|	$\Lambda x. s$	type generalization
|	$s A$	type specialization

Definition (System-F typing relation: $\Gamma \vdash t : A$)

Γ ranges over typing environments (or typing contexts)

$$
\begin{array}{lll}
\Gamma := & & \\
\quad \mid & [\,] & \text{“empty” environment} \\
\quad \mid & \Gamma, x : A & \text{“non-empty” environment}
\end{array}
$$

Definition (System-F typing relation: $\Gamma \vdash t : A$)

Γ ranges over typing environments (or typing contexts)

$$
\begin{array}{llll}
\Gamma := & & & \\
& \mid & [\,] & \text{“empty” environment} \\
& \mid & \Gamma, x : A & \text{“non-empty” environment}
\end{array}
$$

typing environments are comma-separated snoc-lists of (variable,type)-pairs – in fact only the lists whose variables are mutually distinct get used

Definition (System-F typing relation: $\Gamma \vdash t : A$)

Γ ranges over typing environments (or typing contexts)

$$
\begin{array}{lll}
\Gamma := & & \\
\mid & [\,] & \text{“empty” environment} \\
\mid & \Gamma, x : A & \text{“non-empty” environment}
\end{array}
$$

typing environments are comma-separated snoc-lists of (variable,type)-pairs – in fact only the lists whose variables are mutually distinct get used

Notation

- Γ ok means that no variable occurs more than once in Γ

Definition (System-F typing relation: $\Gamma \vdash t : A$)

Γ ranges over typing environments (or typing contexts)

$$
\begin{array}{lll}
\Gamma := & & \\
\mid & [\,] & \text{“empty” environment} \\
\mid & \Gamma, x : A & \text{“non-empty” environment}
\end{array}
$$

typing environments are comma-separated snoc-lists of (variable,type)-pairs – in fact only the lists whose variables are mutually distinct get used

Notation

- Γ ok means that no variable occurs more than once in Γ
- dom Γ denotes the finite set of variables occurring in Γ

Definition (System-F typing relation: $\Gamma \vdash t : A$ (cont'd))

$$\frac{\Gamma\ \mathrm{ok} \qquad x \notin \mathrm{dom}\,\Gamma}{\Gamma, x : A \vdash x : A}\ (\mathrm{var}) \qquad \frac{\Gamma \vdash x : A \qquad x' \notin \mathrm{dom}\,\Gamma}{\Gamma, x' : A \vdash x : A}\ (\mathrm{var'}) \qquad \frac{\Gamma\ \mathrm{ok}}{\Gamma \vdash c^A : A}\ (\mathrm{const})$$

$$\frac{\Gamma\ \mathrm{ok}}{\Gamma \vdash () : \mathrm{unit}}\ (\mathrm{unit}) \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash (s, t) : A \times B}\ (\mathrm{pair}) \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \mathrm{fst}\ t : A}\ (\mathrm{fstT})$$

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \mathrm{snd}\ t : B}\ (\mathrm{sndT}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.\, t : A \to B}\ (\mathrm{fun}) \qquad \frac{\Gamma \vdash s : A \to B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\ t : B}\ (\mathrm{app})$$

$$\frac{\Gamma, x : \tau \vdash s : A}{\Gamma \vdash \Lambda x.\, s : \forall \tau.\, A}\ (\mathrm{gen}) \qquad \frac{\Gamma \vdash s : \forall \tau.\, A}{\Gamma \vdash s\, B : A[B/\tau]}\ (\mathrm{spec})$$

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f : A \to B.\, \lambda x : A.\, f\ x$ should have the same meaning as $\lambda x : A \to B.\, \lambda y : A.\, x\ y$

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f : A \to B.\, \lambda x : A.\, f\ x$ should have the same meaning as $\lambda x : A \to B.\, \lambda y : A.\, x\ y$
- this issue is best dealt with at the level of syntax rather than semantics

Definition (α-equivalence)

- names of λ-bound variables should not affect meaning
- e.g., $\lambda f : A \to B.\, \lambda x : A.\, f\ x$ should have the same meaning as $\lambda x : A \to B.\, \lambda y : A.\, x\ y$
- this issue is best dealt with at the level of syntax rather than semantics
- from now on we re-define $\lambda^{\to}$ term to mean not an abstract syntax tree but rather an equivalence class of such trees with respect to α-equivalence $s =_\alpha t$:

$$\frac{}{c^A =_\alpha c^A} \qquad \frac{}{x =_\alpha x} \qquad \frac{}{() =_\alpha ()}$$

$$\frac{s =_\alpha s' \qquad t =_\alpha t'}{(s, t) =_\alpha (s', t')} \qquad \frac{t =_\alpha t'}{\mathsf{fst}\ t =_\alpha \mathsf{fst}\ t'} \qquad \frac{t =_\alpha t'}{\mathsf{snd}\ t =_\alpha \mathsf{snd}\ t'}$$

$$\frac{s =_\alpha s' \qquad t =_\alpha t'}{s\ t =_\alpha s'\ t'} \qquad \frac{t \cdot (y\ x) =_\alpha t' \cdot (y\ x') \qquad y \text{ does not occur in } \{x,x',t,t'\}}{\lambda x : A.\, t =_\alpha \lambda x' : A.\, t'}$$

$$\frac{s \cdot (x'\ x) =_\alpha s' \cdot (x\ x') \qquad x' \text{ does not occur in } \{s\}}{\Lambda x.\, s =_\alpha \Lambda x'.\, s'} \qquad \frac{s =_\alpha s'}{s A =_\alpha s' A}$$

where $t \cdot (y\ x)$ denotes the result of replacing all occurrences of x with y in t

Definition (α-equivalence – types)

- types contain variables which could be renamed

$$\frac{A \cdot (\tau' \ \tau) =_\alpha \tau' \cdot (\tau \ \tau') \qquad \tau' \text{ does not occur in } \{A\}}{\forall \tau. A =_\alpha \forall \tau'. A'} \qquad \frac{A =_\alpha A' \qquad B =_\alpha B'}{A \rightarrow B =_\alpha A' \rightarrow B'}$$

$$\frac{A =_\alpha A' \qquad B =_\alpha B'}{A \times B =_\alpha A' \times B'}$$

where $t \cdot (y \ x)$ denotes the result of replacing all occurrences of x with y in t

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x : A.\ \_$ binder) by the term s

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x : A.\ \_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them "capturing" any free variables of t

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x : A.\ \_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them “capturing” any free variables of t
- e.g., $(\lambda y : A.\ (y, x))[y/x]$ is $\lambda z : A.\ (z, y)$ and is not $\lambda y : A.\ (y, y)$

Definition (substitution)

- substitution $t[s/x]$ denotes the result of replacing all free occurrences of variable x in term t (i.e. those not occurring within the scope of a $\lambda x\colon A.\ \_$ binder) by the term s
- alpha-converting λ-bound variables in t to avoid them “capturing” any free variables of t
- e.g., $(\lambda y\colon A.\ (y, x))[y/x]$ is $\lambda z\colon A.\ (z, y)$ and is not $\lambda y\colon A.\ (y, y)$
- the relation $t[s/x] = t'$ can be inductively defined by the following rules:

$$\frac{}{c^A[s/x] = c^A} \qquad \frac{}{x[s/x] = s} \qquad \frac{y \neq x}{y[s/x] = y} \qquad \frac{}{()[s/x] = ()}$$

$$\frac{t_1[s/x] = t_1' \qquad t_2[s/x] = t_2'}{(t_1, t_2)[s/x] = (t_1', t_2')} \qquad \frac{t[s/x] = t'}{(\mathrm{fst}\ t)[s/x] = \mathrm{fst}\ t'} \qquad \frac{t[s/x] = t'}{(\mathrm{snd}\ t)[s/x] = \mathrm{snd}\ t'} \qquad \frac{t_1[s/x] = t_1' \qquad t_2[s/x] = t_2'}{(t_1\ t_2)[s/x] = t_1'\ t_2'}$$

$$\frac{t[s/x] = t' \qquad y \neq x \text{ and } y \text{ does not freely occur in } s}{(\lambda y\colon A.\ t)[s/x] = \lambda y\colon A.\ t'}$$

$$\frac{t[s/x] = t' \qquad y \neq x \text{ and } y \text{ does not freely occur in } s}{(\Lambda y.\ t)[s/x] = \Lambda y.\ t'} \qquad \frac{t[s/x] = t'}{(t\ A)[s/x] = t'\ A}$$

Definition (substitution – types)

- types contain variables which could be substituted

$$\frac{A[s/x] = A' \qquad \tau \neq x \text{ and } y \text{ does not freely occur in } s}{(\forall \tau.\, A)[s/x] = \forall \tau.\, A'} \qquad \frac{A[s/x] = A' \qquad B[s/x] = B'}{(A \to B)[s/x] = A' \to B'}$$

$$\frac{A[s/x] = A' \qquad B[s/x] = B'}{(A \times B)[s/x] = A' \times B'}$$

Definition ($\beta\eta$-equality)

the relation $\Gamma \vdash s =_{\beta\eta} t : A$ (where Γ ranges over typing environments, s and t over terms and A over types) is inductively defined by the following rules:

- β-conversion

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x : A.\, t)\ s =_{\beta\eta} t[s/x] : B} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{fst}(s, t) =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{snd}(s, t) =_{\beta\eta} t : B}$$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\Lambda x.\, t)\ s =_{\beta\eta} t[s/x] : B}$$

Definition ($\beta\eta$-equality)

the relation $\Gamma \vdash s =_{\beta\eta} t : A$ (where Γ ranges over typing environments, s and t over terms and A over types) is inductively defined by the following rules:

- β-conversion

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x : A.\, t)\ s =_{\beta\eta} t[s/x] : B} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathsf{fst}(s, t) =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathsf{snd}(s, t) =_{\beta\eta} t : B}$$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\Lambda x.\, t)\ s =_{\beta\eta} t[s/x] : B}$$

- η-conversion

$$\frac{\Gamma \vdash t : A \to B \qquad x \text{ does not occur in } t}{\Gamma \vdash t =_{\beta\eta} (\lambda x : A.\, t\ x) : A \to B} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t =_{\beta\eta} (\mathsf{fst}\ t, \mathsf{snd}\ t) : A \times B} \qquad \frac{\Gamma \vdash t : \mathsf{unit}}{\Gamma \vdash t =_{\beta\eta} () : \mathsf{unit}}$$

Definition ($\beta\eta$-equality)

the relation $\Gamma \vdash s =_{\beta\eta} t : A$ (where Γ ranges over typing environments, s and t over terms and A over types) is inductively defined by the following rules:

- β-conversion

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x : A.\, t)\ s =_{\beta\eta} t[s/x] : B} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathsf{fst}(s, t) =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathsf{snd}(s, t) =_{\beta\eta} t : B}$$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\Lambda x.\, t)\ s =_{\beta\eta} t[s/x] : B}$$

- η-conversion

$$\frac{\Gamma \vdash t : A \to B \qquad x \text{ does not occur in } t}{\Gamma \vdash t =_{\beta\eta} (\lambda x : A.\, t\ x) : A \to B} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t =_{\beta\eta} (\mathsf{fst}\ t, \mathsf{snd}\ t) : A \times B} \qquad \frac{\Gamma \vdash t : \mathsf{unit}}{\Gamma \vdash t =_{\beta\eta} () : \mathsf{unit}}$$

- congruence rules

$$\frac{\Gamma, x : A \vdash t =_{\beta\eta} t' : B}{\Gamma \vdash \lambda x : A.\, t =_{\beta\eta} \lambda x : A.\, t' : A \to B} \qquad \frac{\Gamma \vdash s =_{\beta\eta} s' : A \to B \qquad \Gamma \vdash t =_{\beta\eta} t' : A}{\Gamma \vdash s\ t =_{\beta\eta} s'\ t' : B}$$

Definition ($\beta\eta$-equality)

the relation $\Gamma \vdash s =_{\beta\eta} t : A$ (where Γ ranges over typing environments, s and t over terms and A over types) is inductively defined by the following rules:

- β-conversion

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\lambda x : A.\, t)\ s =_{\beta\eta} t[s/x] : B} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{fst}(s, t) =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash s : A \qquad \Gamma \vdash t : B}{\Gamma \vdash \mathrm{snd}(s, t) =_{\beta\eta} t : B}$$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash s : A}{\Gamma \vdash (\Lambda x.\, t)\ s =_{\beta\eta} t[s/x] : B}$$

- η-conversion

$$\frac{\Gamma \vdash t : A \to B \qquad x \text{ does not occur in } t}{\Gamma \vdash t =_{\beta\eta} (\lambda x : A.\, t\ x) : A \to B} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash t =_{\beta\eta} (\mathrm{fst}\ t, \mathrm{snd}\ t) : A \times B} \qquad \frac{\Gamma \vdash t : \mathrm{unit}}{\Gamma \vdash t =_{\beta\eta} () : \mathrm{unit}}$$

- congruence rules

$$\frac{\Gamma, x : A \vdash t =_{\beta\eta} t' : B}{\Gamma \vdash \lambda x : A.\, t =_{\beta\eta} \lambda x : A.\, t' : A \to B} \qquad \frac{\Gamma \vdash s =_{\beta\eta} s' : A \to B \qquad \Gamma \vdash t =_{\beta\eta} t' : A}{\Gamma \vdash s\ t =_{\beta\eta} s'\ t' : B}$$

- $=_{\beta\eta}$ is reflexive, symmetric and transitive

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t =_{\beta\eta} t : A} \qquad \frac{\Gamma \vdash s =_{\beta\eta} t : A}{\Gamma \vdash t =_{\beta\eta} s : A} \qquad \frac{\Gamma \vdash r =_{\beta\eta} s : A \qquad \Gamma \vdash s =_{\beta\eta} t : A}{\Gamma \vdash r =_{\beta\eta} t : A}$$

Theorem (Progress)

$\forall e : \mathtt{term}, \vdash e : \tau \implies \mathtt{value}\ e \lor \exists e', e \to_\beta e'$

Theorem (Progress)

$\forall e : \texttt{term},\ \vdash e : \tau \implies \texttt{value}\ e \lor \exists e', e \rightarrow_\beta e'$

Theorem (Preservation)

$\forall e, e' : \texttt{term},\ \vdash e : \tau \land e \rightarrow_\beta e' \implies \vdash e' : \tau$

Thanks! & Questions?