Polymorphism

Motivation

Code reuse : use same code in many contexts, allow variable to contain values of many types "Polymorphism" = "Many shapes".
There are a lot o way to do it, in a dynamic language you can have some object that contain indifferently some different type and the language before every operation check the type of variable.
We are going to talk about programming language.

Type

- **Subtype polymorphism:** $T_1 <: T_2$ if a value of type T_1 can be used whenever we expected T_2, T_1 must support all operation of T_2.

- **Parametric polymorphism**(**HASKELL**)**:** The main ideas is introduce some parameters or type variable and a variable can store any type.

  EX. Haskell
  Data List a= Nil| Cons a (List a)

  xs :: List Int
  xs = Cons 1(Cons 2 Nil) *//[1,2]*

  - Can instantiate a type parameter with a type
  - Can variables of type parameter type

  My length :: List a → Int *// ∀a.List a→Int* (**Universal Type**)
  Map:: (a→ b) → [a] → [b] *// ∀a, ∀b.(a→b)→[a]→[b]* ∀

Polymorphic Lambda calculus (System F)- John Reynolds *1974*

E::= n|x|e_1 e_2|λx:T.e|∧ α.e | e[T]
id [Int]
id [Bool]
id [Int→Bool]
id=∧ α.λx
T::= Int |T_1--.T_2|∀ α T| α

Type role

Γ⊢n: Int
Γ⊢x: Γ(x)

$\frac{\Gamma \vdash e_1 : T_1 \to T_2 \quad \Gamma \vdash e_1 : T_1}{\Gamma \vdash e_1\, e_2 : T_2}$

$\frac{\Gamma \vdash x : T_1 \vdash e_1 : T_2}{\Gamma \vdash \lambda x : T_1 \vdash e_1 : T \to_1 T_2}$

$\Gamma ::= \varnothing \mid \Gamma, x:T \mid \Gamma, \alpha$

$$\frac{\Gamma, \alpha \vdash e : T}{\Gamma \vdash \Lambda \alpha . e : \Lambda \alpha . T}$$

$id = \Lambda\, \alpha . \lambda x : \forall \alpha . \alpha \rightarrow \alpha$

Evalation Rules

$$\frac{\Gamma \vdash e : : \forall \alpha . T' \quad \Gamma \vdash T}{\Gamma \vdash e[T] : T'[\alpha \rightarrow T]}$$

$Id\ [Int] : (\alpha \rightarrow \alpha)\ [\alpha \rightarrow Int\} + Int \rightarrow Int$

$$\frac{\Gamma, \alpha \vdash \alpha \qquad \Gamma, \alpha \vdash T}{\Gamma \vdash \forall \alpha . T}$$

List all values of this type:
f:: $\forall \alpha . \alpha \rightarrow \alpha$
if f = $\lambda x . 2 * x : Int \rightarrow Int$ not equal to $\forall \alpha . \alpha \rightarrow \alpha$