1 System F ($\lambda 2$)

1.1 Syntax

$$
\begin{array}{rcll}
\kappa & ::= & \star & \text{kinds} \\
\tau & ::= & X \mid \tau \to \tau \mid \forall X.\tau & \text{types} \\
e & ::= & x \mid \lambda\, x : \tau\, .\, e \mid e_1\, e_2 \mid \Lambda\, X :\ \kappa\, .\, e \mid e@\tau & \text{terms} \\
 & & \lambda & \\
\Gamma & ::= & \emptyset \mid \Gamma, x : \tau \mid \Gamma, X\!: \kappa & \text{contexts}
\end{array}
$$

1.2 Typing Rules

STLC $\dfrac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ [Var]$ $\qquad \dfrac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2}\ [App]$ $\qquad \dfrac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.e : \tau_1 \to \tau_2}\ [Abs]$

$\dfrac{\begin{array}{c} FV(\tau) \subseteq \mathfrak{Dom}(\Gamma) \\ \tau \text{ is well-formed} \end{array}}{\Gamma \vdash \tau : \star}\ [Form]$ $\qquad \dfrac{\Gamma \vdash e : \forall X\!: \star.\sigma \quad \Gamma \vdash \tau : \star}{\Gamma \vdash e@\tau : \sigma[X/\tau]}\ [TApp]$ $\qquad \dfrac{\Gamma,\ X\!: \star \vdash e : \tau}{\Gamma \vdash \Lambda X.e : \forall X.\tau}\ [TAbs]$

1.3 Examples

$3 \equiv \Lambda\alpha.\lambda x^{\alpha}.\lambda f^{\alpha\to\alpha}.f(f(f(x)))\ :\ \forall\alpha.\alpha \to (\alpha \to \alpha) \to \alpha \qquad id \equiv \Lambda\alpha.\lambda x^{\alpha}.x\ :\ \forall\alpha.\alpha \to \alpha$

$$id(\forall\alpha.\alpha \to \alpha)\ :\ (\forall\alpha.\alpha \to \alpha) \to (\forall\alpha.\alpha \to \alpha)$$

1.4 β-reduction

Basis (beta):

- $(\lambda x : \alpha.M)N \to_\beta M[x/N]$
- $(\Lambda\alpha : \star.M)@\sigma \to_\beta M[\alpha/\sigma]$

Compatibility:
If $M \to_\beta N$ then Application

- $ML \to_\beta NL$
- $LM \to_\beta LN$
- $M@\sigma \to_\beta N@\sigma$

Abstraction (if strong):

- $\lambda x : \alpha.M \to_\beta \lambda x : \alpha.N$
- $\Lambda\alpha : \star.M \to_\beta \Lambda\alpha : \star.N$

1.5 Example: Polymorphic Lists

Suppose we have *List* type constructor (can't be expressed in $\lambda 2$ but may be encoded)

$$\begin{aligned} nil &: \forall X.\ List\ X \\ cons &: \forall X.\ X \to List\ X \to List\ X \end{aligned}$$

$$\begin{aligned} isnil &: \forall X.\ List\ X \to Bool \\ head &: \forall X.\ List\ X \to X \\ tail &: \forall X.\ List\ X \to List\ X \end{aligned}$$

Example: polymorphic *map*

$$\begin{aligned} map = \lambda\, X. \quad & \lambda\, Y. \lambda\, f : X \to Y. (fix\ (\lambda m : (List\ X) \to (List\ Y). \lambda\, l : List\ X. \\ & \text{if } isnil\ X\ l \text{ then } nil\ Y \text{ else } cons\ Y\ (f\ (head\ X\ l))\ (m\ (tail\ [X]\ l)))); \end{aligned}$$

$>\ map\ :\ \forall X. \forall Y. (X \to Y) \to List\ X \to List\ Y$
$l\ =\ cons\ Nat\ 4\ (cons\ Nat\ 3\ (cons\ Nat\ 2\ (nil\ Nat)));$
$>\ l\ :\ List\ Nat$
$head\ Nat\ (map\ Nat\ Nat\ (\lambda\, x : Nat.\ succ\ x)\ l);$
$>\ 5\ :\ Nat$

1.6 Exercises:

1. Prove that *map* really has the type shown (by constructing type derivation)
2. Write a polymorphic list-reversing function: $reverse\ :\ \forall X.\ List\ X \to List\ X$
3. Write a simple polymorphic sorting function: $sort\ :\ \forall X.\ (X \to X \to Bool) \to (List\ X) \to List\ X$

1.7 Expressive Power: $\lambda 2 \sim$ Primitive Recursion

Primitive Recursion:
Smallest class of functions over $\mathbb{N}$ containing:

- constant function 0
- successor
- projection

closed under (final number of) composition and primitive recursion corresponds to for-loops
Church numerals:
$\Lambda\alpha : \star.\lambda f : \alpha \to \alpha.\lambda x : \alpha.f^n x : \quad C \equiv \forall\alpha : \star.(\alpha \to \alpha) \to \alpha \to \alpha$
Pairs:
$\langle M, N\rangle \equiv \lambda z : C \to C \to C.\ z\ M\ N\ :\ pair \equiv (C \to C \to C) \to C$
$fst \equiv \lambda p : pair.(\lambda x : C.\lambda y : C.x)$
$snd \equiv \lambda p : pair.(\lambda x : C.\lambda y : C.y)$

$$
\begin{array}{lll}
fst\langle M,N\rangle & \equiv (\lambda p:\ pair.\ p\ (\lambda x:C.\ \lambda y:C.\ x))\ \langle M,N\rangle & =_\beta \langle M,N\rangle\ (\lambda x:C.\ \lambda y:C.\ x) \\
 & \equiv (\lambda z:C\to C\to C.\ z\ M\ N)(\lambda x:C.\ \lambda y:C.\ x) & =_\beta (\lambda x:C.\ \lambda y:C.\ x)\ M\ N =_\beta M \\
snd\langle M,N\rangle & \equiv (\lambda p:\ pair.\ p\ (\lambda x:C.\ \lambda y:C.\ y))\ \langle M,N\rangle & =_\beta \langle M,N\rangle\ (\lambda x:C.\ \lambda y:C.\ y) \\
 & \equiv (\lambda z:C\to C\to C.\ z\ M\ N)(\lambda x:C.\ \lambda y:C.\ y) & =_\beta (\lambda x:C.\ \lambda y:C.\ y)\ M\ N =_\beta N
\end{array}
$$

Example: Factorial by Primitive Recursion

Recursive Definiton:

$$fact(n) = \text{if } (n = 0) \text{ then } 1 \text{ else } n * fact(n-1)$$

Primitive Recursive Definiton

$$
\begin{array}{lcl}
fact(0) & = & 1 \\
fact(n+1) & = & mult(n+1, fact(n))
\end{array}
$$

Simulating Primitive Recursion via Pairs:

$$
\begin{array}{lcl}
next\langle n, fact(n)\rangle & \equiv & \langle n+1, fact(n+1)\rangle \\
fact\ c_0 & \equiv & c_1 \\
fact\ (succ\ c_n) & \equiv & snd(next\ c_n\ (fact\ c_n))
\end{array}
$$

How to define *next*?

$$next := \lambda p : pair.\langle succ(fst\ p),\ mult\ (succ(fst\ p))\ (snd\ p)\rangle : pair \to pair$$

$$
\begin{array}{lcl}
\langle 0,\ fact\ 0\rangle =_\beta \langle 0,1\rangle & \equiv & next^0\langle 0,1\rangle \\
\langle 1,\ fact\ 1\rangle =_\beta next\langle 0,1\rangle & \equiv & next^1\langle 0,1\rangle \\
\langle 2,\ fact\ 2\rangle =_\beta next\langle 1, fact 1\rangle =_\beta next(next\langle 0,1\rangle) & \equiv & next^2\langle 0,1\rangle \\
\dots & & \dots \\
\langle n,\ fact\ n\rangle =_\beta next^n\langle 0,1\rangle & \equiv & next^n\langle 0,1\rangle
\end{array}
$$

Now we have function *next*:

$$next := \lambda p : pair.\langle succ(fst\ p),\ mult\ (succ(fst\ p))\ (snd\ p)\rangle : pair \to pair$$

s.t. $next^n\langle 0,1\rangle =_\beta \langle n,\ fact(n)\rangle$
Q: How to define $fact$?

Remember Church numerals?

$$\lambda\alpha.\lambda f : \alpha \to \alpha.\lambda x : \alpha.\ f^n\ x\ :\ \forall\alpha.\ (\alpha \to \alpha) \to \alpha \to \alpha$$

$$fact \equiv \lambda n : C.\ snd\ (n\ pair\ next\ \langle 0,1\rangle)$$

Primitive Recursion

- $f(0) = a$
- $f(n+1) = h(n, f(n))$

$$next \equiv \lambda p : pair.\langle succ\ (fst\ p),\ h\ (fst\ p)\ (snd\ p)\rangle$$

$$f \equiv \lambda n : C.\ snd\ (n\ pair\ next\ \langle 0,\ a\rangle)$$

Why can't we do it in STLC?

$$\begin{array}{ll} & \lambda_{\rightarrow} \\ c_n & : (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \\ f & \equiv n\ next\ \langle 0, a\rangle \end{array}$$

$$\begin{array}{ll} & \lambda 2 \\ c_n & : \forall\alpha : \star.(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha \\ f & \equiv n\ pair\ next\ \langle 0, a\rangle \end{array}$$

$next : pair \rightarrow pair$; hence, $\alpha \equiv pair$

$pair = (C \rightarrow C \rightarrow C) \rightarrow C$ and $C \equiv (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$ Thus $\alpha \equiv \underbrace{((\underbrace{(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha}_{C}) \rightarrow (\underbrace{(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha}_{C}) \rightarrow (\underbrace{(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha}_{C})) \rightarrow (\underbrace{(\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha}_{C})}_{pair}$?!?

Qed.

2 $\lambda\underline{\omega}$: type constructors

2.1 Syntax

$$\begin{array}{rlrl} t ::= x \mid \lambda x : \tau.t \mid t\ t & \text{terms} & \kappa ::= \star \mid \star \Rightarrow \star & \text{kinds} \\ \tau ::= X \mid \lambda x :: \kappa.\tau \mid \tau\ \tau \mid \tau \rightarrow \tau & \text{types} & \Gamma ::= \emptyset \mid \Gamma,\ x : \tau \mid \Gamma,\ X :: \kappa & \text{contexts} \end{array}$$

2.2 Kinding

$$\frac{X :: \kappa \in \Gamma}{\Gamma \vdash X :: \kappa}\ \text{K-TVar} \qquad \frac{\Gamma,\ X :: \kappa_1 \vdash \tau :: \kappa_2}{\Gamma \vdash \lambda X :: \kappa_1.\tau\ ::\ \kappa_1 \Rightarrow \kappa_2}\ \text{K-Abs} \qquad \frac{\Gamma \vdash \tau_1 :: \kappa_1 \Rightarrow \kappa_2 \quad \Gamma \vdash \tau_2 :: \kappa_1}{\Gamma \vdash \tau_1\ \tau_2\ ::\ \kappa_2}\ \text{K-App} \qquad \frac{\Gamma \vdash \tau_1 :: \star \quad \Gamma \vdash \tau_2 :: \star}{\Gamma \vdash \tau_1 \rightarrow \tau_2 :: \star}\ \text{K-Arrow}$$

2.3 Typing

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ \text{T-Var} \qquad \frac{\Gamma \vdash \tau_1 :: \star \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \lambda x : \tau_1.e : \tau_1 \rightarrow \tau_2}\ \text{T-Abs} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ \text{T-App} \qquad \frac{\Gamma \vdash e : \tau_2 \quad \tau_1 \equiv \tau_2 \quad \Gamma \vdash \tau_1 :: \star}{\Gamma \vdash e : \tau_1}\ \text{T-Eq}$$

2.4 Type Equivalence $\equiv$

$$\frac{}{\tau \equiv \tau}\ \text{Q-Refl} \qquad \frac{\tau_2 \equiv \tau_1}{\tau_1 \equiv \tau_2}\ \text{Q-Symm} \qquad \frac{\tau_1 \equiv \tau_2 \quad \tau_2 \equiv \tau_3}{\tau_1 \equiv \tau_3}\ \text{Q-Trans} \qquad \frac{\tau_1 \equiv \sigma_1 \quad \tau_2 \equiv \sigma_2}{\tau_1 \rightarrow \tau_2 \equiv \sigma_1 \rightarrow \sigma_2}\ \text{Q-Arrow}$$

$$\frac{\tau_1 \equiv \tau_2}{\lambda X :: \kappa_1.\tau_1 \equiv \lambda X :: \kappa_2.\tau_2}\ \text{Q-Abs} \qquad \frac{\tau_1 \equiv \sigma_1 \quad \tau_2 \equiv \sigma_2}{\tau_1\ \tau_2 \equiv \sigma_1 \sigma_2}\ \text{Q-App} \qquad \frac{}{(\lambda X :: \kappa_1.\tau_1)\ \tau_2 \equiv \tau_1[X/\tau_2]}\ \text{Q-AppAbs}$$

2.5 Alternative syntax and rules

Alternative Syntax

$t ::= x \mid \lambda x : \tau.\, t \mid t\ t$	terms	$\kappa ::= \star \mid \cancel{\star \Rightarrow \star}$ $\star \rightarrow \star$	kinds
$\tau ::= x \mid \lambda x : \kappa.\, \tau \mid \tau\ \tau \mid \tau \rightarrow \tau$	types	$\Gamma ::= \emptyset \mid \cancel{\Gamma,\ x : \tau \mid \Gamma,\ X :: \kappa}$ $\Gamma,\ x : s$	contexts
$s ::= \star \mid \square$	sorts		

Example:

$$\underbrace{\lambda x : \alpha . x}_{\text{term}} : \underbrace{\alpha \rightarrow \alpha}_{\text{type/constructor}} : \underbrace{\star}_{\text{kind}} : \underbrace{\square}_{\textit{sort}}$$

$$✗ \quad \underbrace{\lambda \alpha : \star . \alpha \rightarrow \alpha}_{\text{proper constructor}} : \underbrace{\star \rightarrow \star}_{\text{kind}} : \underbrace{\square}_{\textit{sort}}$$

2.6 Alternative Typing and Kinding Rules

$$\frac{\Gamma \vdash A : s \quad x \notin \Gamma}{\Gamma, x : A \vdash x : A}\ \text{Var} \qquad \text{instead of two rules} \qquad \frac{\Gamma \vdash A : \star \quad x \notin \Gamma}{\Gamma, x : A \vdash x : A}\ [Var^{\star}] \quad \frac{\Gamma \vdash A : \square \quad x \notin \Gamma}{\Gamma, x : A \vdash x : A}\ [Var^{\square}]$$

$$\frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash A \rightarrow B : s}{\Gamma \vdash \lambda x : A.M : A \rightarrow B}\ \text{Abs} \qquad \text{instead of two rules}$$

$$\frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash A \rightarrow B : \star}{\Gamma \vdash \lambda x : A.M : A \rightarrow B}\ Abs^{\star}$$

$$\frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash A \rightarrow B : \square}{\Gamma \vdash \lambda x : A.M : A \rightarrow B}\ Abs^{\square}$$

$$\frac{\Gamma \vdash F : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : B}\ \text{App}$$

$$\frac{\Gamma \vdash A : s \quad \Gamma \vdash B : s}{\Gamma \vdash A \rightarrow B : s}\ \text{Form} \qquad \text{instead of two rules}$$

$$\frac{\Gamma \vdash A : \star \quad \Gamma \vdash B : \star}{\Gamma \vdash A \rightarrow B : \star}\ \text{Form}^{\star} \qquad \text{ex:}\ \frac{\Gamma \vdash \alpha : \star \quad \Gamma \vdash \beta \rightarrow \gamma : \star}{\Gamma \vdash \alpha \rightarrow (\beta \rightarrow \gamma) : \star}\ \text{Form}^{\star}$$

$$\frac{\Gamma \vdash A : \square \quad \Gamma \vdash B : \square}{\Gamma \vdash A \rightarrow B : \square}\ \text{Form}^{\square} \qquad \text{ex:}\ \frac{\Gamma \vdash \star : \square \quad \Gamma \vdash \star \rightarrow \star : \square}{\Gamma \vdash \star \rightarrow (\star \rightarrow \star) : \square}\ \text{Form}^{\square}$$

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_{\beta} B'}{\Gamma \vdash A : B'}\ [Conv]$$

$$\frac{}{\vdash \star : \square}\ \text{Sort (Ax)}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B : s \quad x \notin \Gamma}{\Gamma, x : B \vdash a : A}\ \text{Weak}$$

[Weak] motivation:

$$\dfrac{\dfrac{\dfrac{}{\vdash \star : \square}\ Ax}{\alpha : \star \vdash \alpha : \alpha}\ Var}{\alpha : \star, x : \alpha \vdash x : \alpha}\ Var$$

$$\dfrac{\dfrac{\dfrac{}{\alpha : \star, y : \alpha \vdash \star : \square}\ \text{✗}}{\alpha : \star, y : \alpha \vdash \alpha : \alpha}\ Var}{\alpha : \star, x : \alpha \vdash x : \alpha, y : \alpha}\ Var$$

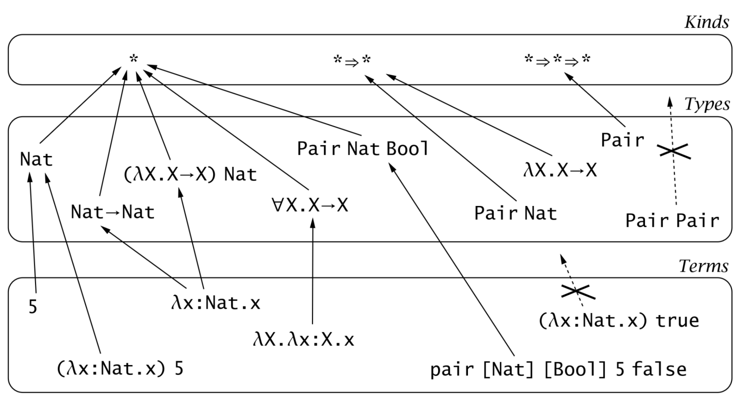
2.7 Expressive power, System F omega

$\lambda\underline{\omega}$ and $\lambda_{\rightarrow}$ have THE SAME computational power!

$\lambda\omega$ (**System** F_ω) ::= $\lambda\underline{\omega} + \lambda 2$

F_1	$\mathcal{K}_1$	$= \emptyset$ or $\{\star\}$	$\lambda_{\rightarrow}$
F_2	$\mathcal{K}_2$	$= \{\star\}$	System F
F_{i+1}	$\mathcal{K}_{i+1}$	$= \{\star\} \cup \{\kappa \Rightarrow \iota \mid \kappa \in \mathcal{K}_i$ and $\iota \in \mathcal{K}_{i+1}\}$	
F_ω	$\mathcal{K}_\omega$	$= \bigcup_{1 \leq i} \mathcal{K}_i$	F_ω

Example (from Pierce)



3 Assignments:

1. see section 1.6
2. Write a function that computes the sum of a list of natural numbers in System $F\omega$ and provide type derivation for it
3. Implement System F: evaluation + typechecking + Church numerals (with functions inc, add, mult, dec, minus)