2/22

CSCI 7000 - 11

Simply typed Lambda Calculus

* Why types?

* Types: $\underline{A}, \underline{B} ::= \text{unit}$
  $\mid A \to B$
  $\mid A \times B$
  $\mid \ldots$

* Terms grammar

* Typing judgment $\left(\boxed{\Gamma \vdash e : A}\right)$
  * Rules: var, unit, $\to$ elim, $\to$ intro, $\times$ elim 1, $\times$ elim 2, $\times$ intro

  * Eg 1: $\lambda(x : \text{unit}) . x \; : \text{unit} \to \text{unit}$

  * Eg 2: $(\lambda x : A \to A . \lambda y : A . x (x y))$
    $: (A \to A) \to A \to A$

* Typability: Not all terms have types.
  * Eg: $\text{fst}\,(\lambda x . e)$
    $\langle e_1, e_2 \rangle \; e_3$ } what's common?

* No polymorphism: "Simply typed"

* What about Y combinator?

* Fixpoint operator

$\text{pair} = \lambda f.\ \lambda s.\ b\ \ b\ f\ s$

pair

$\text{fst} = \lambda p.\ p\ \text{true}$

$\text{snd} = \lambda p.\ p\ \text{false}$

$\text{fst}\ (\underbrace{\text{pair}\ x\ y}) = (\text{pair}\ x\ y)\ \text{true}$

$((\lambda f. \lambda s. \lambda b\ b\ f\ s)\ x\ y)\ \text{true}$

$= (\text{true}\ x\ y)$

$= x$

$\text{fst}\ (\lambda f.\ \lambda s.\ \lambda b.\ b\ f\ s)$

$\to (\lambda f.\ \lambda s.\ \lambda b.\ b\ f\ s)\ \text{true}$

$(\lambda s.\ \lambda b.\ b\ \text{true}\ s)$

* Types $A, B ::= \text{Unit}$

$\mid A \to B$

$\mid A \times B$

* Terms $M, N ::= M\ N$ (e_1 e_2)

$\mid \langle M, N \rangle$

$\mid \text{fst } M$

$\mid \text{snd } M$

$\mid ()$

Simple extension

$\mid \boxed{\lambda(x : A)}\ M$

$\mid x$

$\mid \text{fix } M$

* Type checking:

Math $\boxed{e \to v}$

Code

"Dynamic Semantics"

operational

$$\frac{e_1 \to \lambda x.e \qquad e_2 \to v}{e_1\ e_2 \to e[v/x]}$$

Interpreter

Runtime

"Static Semantics"

Typing judgement

$\boxed{\Gamma \vdash e : A}$

"Typing Context"

Static time

Type checker

$$\boxed{\lambda x.\ \lambda y.\ \ x\ y}$$

$$\underbrace{\left(\lambda (x : A \to B).\ \lambda (y : A).\ \overbrace{(x\ y)}^{B}\right)}_{e} : \underbrace{(A \to B) \to A \to B}_{T}$$

furnish 'x'

lvdam

$$\begin{pmatrix} x : A \to B, \\ y : A \end{pmatrix} \vdash x\ y : B$$

$\longrightarrow$ Type contexts:

$$\Gamma ::= \phi$$

$$\mid \Gamma, x : A$$

$$\boxed{\Gamma \vdash e : A}$$

$$\frac{}{\Gamma \vdash () : \text{unit}}\ (\text{Unit})$$

$$\boxed{\frac{\Gamma(x) = A}{\Gamma \vdash x : A}}\ [\text{Var}]$$

$$\frac{\Gamma \vdash e : A \times B}{\Gamma \vdash (\text{fst } e) : A}\ [\times\text{-Elim-L}]$$

$$\frac{\Gamma \vdash e : A \times B}{\Gamma \vdash (\text{snd } e) : B}\ [\times\text{-Elim-R}]$$

$$\frac{\Gamma \vdash e_1 : A \qquad \Gamma \vdash e_2 : B}{\Gamma \vdash \langle e_1, e_2 \rangle : A \times B}\ [\times\text{-intro}]$$

$$\frac{\Gamma \vdash e_1 : A \to B \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1\ e_2 : B}\ [\to\text{-Elim}]$$

$$\frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash (\lambda x : A).\ e : A \to B}\ [\to\text{ intro}]$$

Examples of Typing derivations

$\lambda(x : \text{unit}).\ x : \text{unit} \to \text{unit}$

$(\lambda x : A \to A.\ \lambda y : A.\ x\ (x\ y))$

$: (A \to A) \to A \to A$

$\Gamma' = (\Gamma, x : A)$

$\Gamma'(x) = A$

$[x : \text{int}] \vdash (\lambda x : \text{unit}.\ x)$

$\downarrow$ α-renam

$(\lambda w : \text{unit}).\ w$

$$(\Gamma, x : \mathrm{unit})(x) = \mathrm{unit}$$

$$\dfrac{\dfrac{\Gamma'(x) = \mathrm{unit}}{\Gamma' \; (\Gamma, x : \mathrm{unit}) \vdash x : \mathrm{unit}}\;[\mathrm{Var}]}{\Gamma \vdash \underbrace{\lambda(x : \mathrm{unit}).\, x}_{\mathrm{Lam}\,(\text{"}x\text{"},\ \mathrm{Var}\ \text{"}x\text{"})} : \mathrm{unit} \Rightarrow \mathrm{unit}}\;(\to \mathrm{intro})$$

"Typing derivation"

type derivation tree

$$\Gamma_1(x) = A \to A$$

$$\dfrac{\dfrac{\cdots}{\Gamma_1 \vdash x : A \to A}\;\mathrm{Var} \qquad \dfrac{\cdots}{\Gamma_1 \vdash y : A}\;\mathrm{Var}}{\quad}\;[\to \mathrm{elim}]$$

$$\frac{[x : A \to A;\ y : A] \vdash x : A \to A \qquad \Gamma_1 \vdash x\ y : A}{[x : A \to A;\ y : A] \vdash x\ (x\ y) : A}\ [\to \text{Elim}]$$

$$\frac{[x : A \to A;\ y : A] \vdash x\ (x\ y) : A}{[x : A \to A] \vdash \lambda(y : A).\ x\ (x\ y) : A \to A}\ [\to \text{intro}]$$

$$\frac{[x : A \to A] \vdash \lambda(y : A).\ x\ (x\ y) : A \to A}{\phi \vdash (\lambda x : A \to A.\ \lambda y : A.\ x\ (x\ y)) : (A \to A) \to A \to A}\ [\to \text{intro}]$$

Curry - Howard Isomorphism

Computation	Logic	1.	Coq
Types	Proposition	2.	Isabelle HoL
Programs	proofs	3.	ELF

→ Ω combinator

$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$

$(\lambda\ (x : \_).\ x\ x)\ (\lambda x : \_ .\ x\ x)$

$x : A \to A \qquad A = A \to A$

Y combinator

$$Y = \lambda f.\ (\lambda x.\ f(x\ x))\ (\lambda x.\ f(x\ x))$$

$$\frac{\Gamma \vdash f : (\underbrace{A \to A}_{T}) \to \underbrace{A \to A}_{T}}{\Gamma \vdash (\text{fix } f) : \underbrace{A \to A}_{T}}$$

$$(\text{fix-eval})\ \frac{e_2 \to v \qquad e[v/x]\left[\frac{\text{fix } g}{f}\right] \to v'}{\underbrace{\text{fix } (\lambda f.\ \lambda x.\ e)}_{g}\ e_2 \to v'}$$

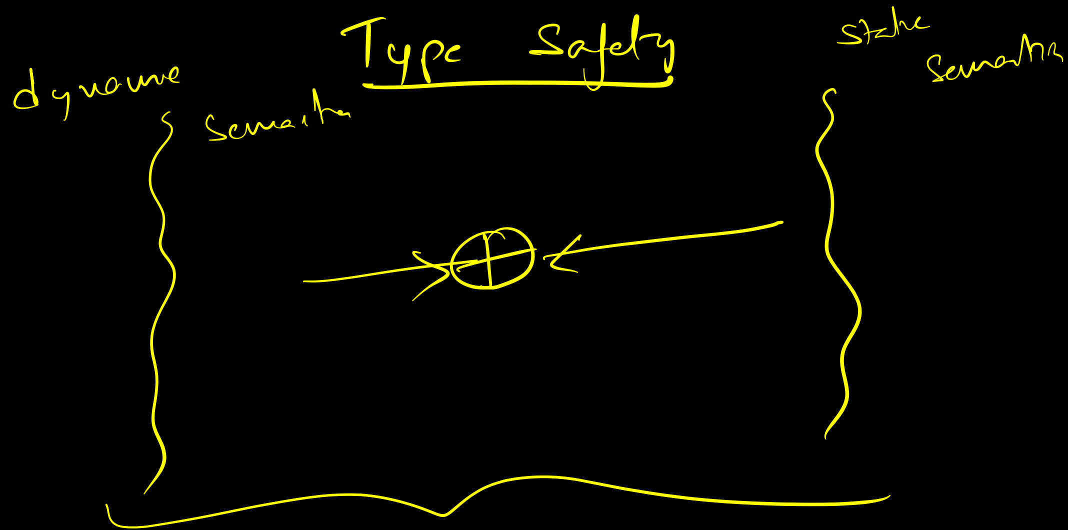
$$\text{fix } \underbrace{(\lambda f^{(N \to N)}.\ \lambda n^{N}.\ \overbrace{\text{if } n = 0 \text{ then } 1 \text{ else } n * f(n-1)}^{e\ \ N})}_{\text{fact}}$$

$$(\text{fix fact})\ 2$$

$$e[2/n]\left[\frac{\text{fix fact}}{f}\right]$$

if 2 = 0 then 1 else 2 * (fix fact' 1)

$$\mathrm{fix}\quad \underbrace{\mathrm{fact} : \underbrace{(N \to N)}_{A} \to \underbrace{N \to N}_{A}}_{\underline{N} \to N}$$



Well-typed programs do not get "stuck".

$e ::= \ldots$
$\quad \mid \mathrm{fst}\ e$
$\quad \mid \mathrm{snd}\ e$
$\quad \mid e_1\ e_2$
$\quad \mid \ldots$

$$\frac{e \to \langle v_1, v_2 \rangle}{\mathrm{fst}\ e \to v_1}\ [\mathrm{Fst\text{-}Eval}]$$

$$\frac{e_1 \to n_1 \qquad e_2 \to n_2}{e_1 + e_2 \to v}\quad v = n_1 + n_2$$

$1 + \text{"Hello"}$

Type Safety : $\forall e. \forall T.\ e : T \Rightarrow \exists v.\ e \longrightarrow v$

(Imprecise)

Angelic Language

$$\left\{ \frac{e_1 \longrightarrow v_1 \qquad e_2 \longrightarrow v_2 \qquad v_2 \neq 0}{e_1 / e_2 \longrightarrow (v_1 / v_2)} \right.$$

$\underset{\sim}{T}$

int

float

$$\boxed{\frac{\ldots\ldots\ldots \quad e_1 \longrightarrow v_1 \qquad v_1 \neq \text{NULL}}{e_1.n \longrightarrow v}}$$

Int : Set of all integers

$\underbrace{\{ v : \text{Int} \mid v > 0 \}}_{\text{Type refinement}}$: Set of all +ve integers

$$\frac{e_1 : \text{Int} \qquad e_2 : \{ v : \text{Int} \mid v > 0 \}}{e_1 / e : \text{Int}}$$

$$\frac{T_1 <: T_2}{\text{Array}[T_1] <: \text{Array}[T_2]}$$

$$\frac{e_1 \longrightarrow v_1 \qquad v_1 = \text{Null}}{e_1 . n \longrightarrow \underbrace{\text{Null Pointer Exception}}}$$

→ Liquid Haskell   Haskell + refinement types

ML + refinement types : Catalyst

array . get : a → i { i < Len(a) } → v