COL 100 - Lec 12

1. Currying

- Eg: Definition
  add x y = x+y

  fn: $\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$

  - Note partial evaluation possible

    i.e. add 3 will evaluate to a function that adds 3 to its argument

    add 3 = fn: $\mathbb{Z} \rightarrow \mathbb{Z}$

    $y \rightarrow y+3$

  - where is currying critical?
    - where you expect the output to be a function too!

Eg: function derivatives, ...

Thus

Higher Order function we saw in the last class

→ HOSumProd Op termFnc nextfnc iden l u

= if l > u then iden

else Op (termFnc l) (HOSumProd
Op termFnc nextfnc
iden (nextfnc l) u)

Could also be specified as the following:

** Show fact Computation!
** Show sum Computation!

→ HOSumProd (Op, termFnc, nextfnc, iden, l, u)

= if (l > u) then iden

else op (termfnc (l), HoSumProd (. . --- .))

↳ in the non-curried form

What have we looked so far

- Procedures as arguments (or functions)

Let us look now at constructing procedures as λ Expressions. (lambda)

Motivation: In the example to compute sum_series_pi we needed

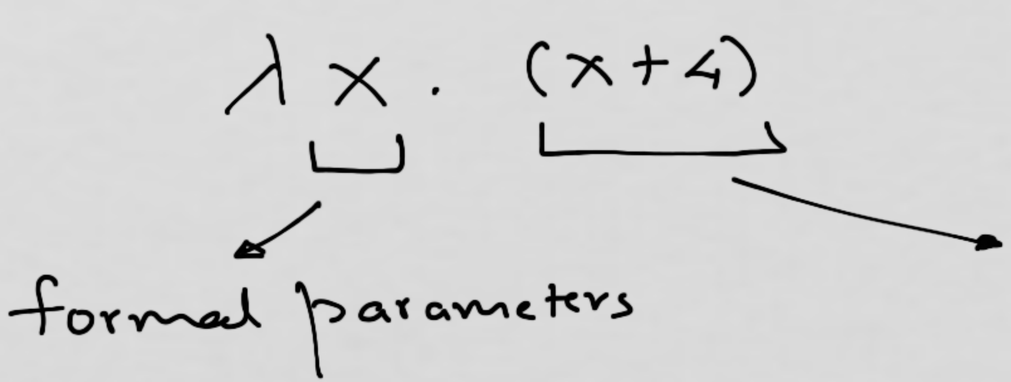
- pi_term to be `(1.0/x*(x+2))`

pi_next to be `(x+4)`

It is awkward to define trivial procedures pi_term & pi_next just so we can define our higher order function.

A more convenient way is to use lambda Expressions

Mathematically,



SML

```
fn x => (x+4)
```

The resulting procedure is the same as what one would have obtained using keyword fun with a procedure name, Except that lambda procedures are no-name (or nameless) procedures

Procedures as general methods

Elaborate Example

- finding roots of equations by the half-interval method

$f(x) = 0$ [f is continuous]

idea is if we are given pts a, b

s.t. $f(a) < 0 < f(b)$

then f must have one root (or zero) betⁿ a & b.

take

$$x = \frac{a+b}{2}$$

- if $f(x) > 0$ then the root is betⁿ a & x
- if $f(x) < 0$ then the root is betⁿ x & b
- Continuing this way we reduce the interval by half at each step until some tolerance level is reached (let us call that T)

- Then, the process stops in $O(\log(L/T))$ steps.

  [L is the length of original interval & T is the tolerance]

- function (or procedure) to implement our strategy

```
fun search f neg-pt pos-pt =
  let
    val midpt = average neg-pt pos-pt
  in
    if (close-enough? neg-pt pos-pt)
    then midpt
    Else
      let
        val test_value = f midpt
      in
        if test-value > 0.0 then
                        search f neg.pt midpt
        Else if test_value < 0.0 then
                        search f midpt pos-pt
        Else midpt
```

```
fun close-enough? x y
= (abs (x-y)<0.001)
```

```
fun half-interval f a b =
    let
        val a-v = f a
        val b-v = f b
    in
        if a-v < 0.0 andalso b-v >= 0.0 then
            search f a b
        Else if a-v >= 0.0 andalso b-v < 0.0 then
            search f b a

        Else
            raise NotOppositeSigns
```

half-interval-method sin 2.0 4.0

for $x^3 - 2x - 3 = 0$, bet^n 1 & 2

half-interval-method (fn x ⇒ x*x*x - 2*x - 3) 1.0 2.0