# Lec 2 : ILFP

Recap from Lec 1

Brief intro to :
- Computation, Computability, Algorithm, program & programming, Languages etc.
- History of programming languages
- Functional programming & its differences with imperative programming.
- Program Specification [and its importance]

Today [and few more lectures in forthcoming weeks]

- Functional programming
  - Recursion and induction

We had discussed that

- Pure f.p. languages have no assignments
  - → i.e. have only "referentially transparent" expressions
    - → As a result easier compiler optimisations such as → CSE. [Common Subexpression Elimination]
- Instead of iteration recursion is used

Eg

```
x = a*b+5
y = a*b+x
```

# Long multiplication

$$a = a_m a_{m-1} \ldots a_0$$

$$b = b_n b_{n-1} \ldots b_0$$

$$a * b = a * \sum_{j=0}^{n} 10^j b_j = a b_0 + a b_1 \cdot 10 + \ldots + a \cdot b_n \cdot 10^n$$

Computation is finite terms terminating!

Alg:

$$LongMult(a,b) = \begin{cases} a b_0 & \text{if } n = 0 \\ a b_0 + LongMult(a, b') * 10 & \text{if } n > 0 \end{cases}$$

Base Case

Induction step

where

$b_0 = b \text{ `rem` } 10$

$b' = b \text{ `quot` } 10$

**Proof** Base Case: if $b = 0$ then clearly

$$ab_0 = 0 = \text{LongMult}(a, b)$$

IH: $\text{LongMult}(a, c) = ac$ where $c$ has $n$ digits

i.e. $0 \leq c < 10^n$

IS: $\text{LongMult}(a, b) = ab_0 + \text{LongMult}(a, b') \cdot 10$

$= ab_0 + \underbrace{ab' \cdot 10}_{\text{from IH}}$

$= a(b_0 + b' \cdot 10)$

$= ab$ [from $b = 10b' + b_0$]

* Key obsv: inducted on the length of the sequence of digits in $b$.

Curiously many mathematical forms but not all are algorithms.

Eg

$$n! = \begin{cases} 1 & \text{if } n < 1 \\ n * (n-1)! & \text{otherwise} \end{cases}$$

] An alg.

$$= \begin{cases} 1 & \text{if } n < 1 \\ \dfrac{(n+1)!}{(n+1)} & \text{otherwise} \end{cases}$$

] Is not an algorithm

Why?

A: The method of Evaluation is non-terminating

Fact (n) = if n=0 then 1 else n * fact(n-1)

fact(4) = (4 * fact(4-1))

= (4 * fact(3))

= (4 * 3 * fact(3-1))

= (4 * 3 * fact(2))

= (4 * 3 * 2 * fact(2-1))

⋮

= (4 * 3 * 2 * 1 * fact(0))

fact(0) — termination base case

Deferred Computation

Analysis

$$T(n) = \begin{cases} 0 \\ 1 + T(n-1) \end{cases}$$

$T(n) = n$

# More Examples of recursion

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x * x^{n-1} & \text{otherwise} \end{cases}$$

# Fast power

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ (x^2)^{n \text{ quot } 2} & \text{if } n \text{ is even} \\ x \cdot (x^2)^{n \text{ quot } 2} & \text{if } n \text{ is odd} \end{cases}$$

# Correctness of fastpower

Base: $n = 0 \Rightarrow$ fastPower $(x, n) = 1 = x^0$

I.H: fastPower $(x, m) = x^m$ for $m \leq (n-1)$

I.S.

- if $n$ is odd, i.e $n = 2k+1$, $k \geq 0$

  fastPower $(x, n) = x \cdot$ (fastPower $(x,$ n `quot` 2$))^2$

  $= x \cdot x^{n\text{`quot`}2} \cdot x^{n\text{`quot`}2}$ [By I.H.]

  $\Rightarrow$ n `quot` 2 $= k$

  $= x \cdot x^{n-1}$

  $= x^n$

- if $n$ is even, then $n = 2k$, n `quot` 2 $= k$

# Efficiency Analysis

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & n > 1 \end{cases}$$

Assume $n = 2^m \Rightarrow \log_2 n = m$

$$T(n) = T(2^{m-1}) + 1$$

$$= T(2^{m-2}) + 2$$

$$\vdots$$

$$= T(2^0) + m$$

$$= 1 + m = O(\log_2 m)$$

Similarly

$$Sqrt(n) = \begin{cases} m & \text{if } \exists m: \; m*m = n \\ 0 & \text{otherwise} \end{cases}$$

Not an algorithm → why?

A: No method of evaluation provided

Another recursion : gcd(a,b) : Largest ≠ that divides both a, b.

Eudid → Greek Mathamatician

↳ gave this efficient method of computation in his book Elements

Fun fact : GCD is one of the oldest algorithms in use → foundation of crypto primitives

Observation :

If $p \mid a$ & $p \mid b$

then $a = p * a'$ & $b = p * b'$

then $p \mid a-b$

Proof $a = pa'$, $b = pb'$

$a - b = p(a' - b')$

$\Rightarrow p \mid (a-b)$ && $p \mid b$

thus $d = GCD(a,b)$ can be represented as a linear combination of

$$\boxed{d = c_1 \cdot a + c_2 \cdot b}$$ Bezout's Identity

Alg :-

$\forall\ a,b>0,$ $gcd(a,b) =$ $\begin{cases} a & \text{if } a=b \\ gcd(a-b,\ b) & \text{if } a>b \\ gcd(\ a,\ b-a) & \text{if } b>a \end{cases}$

Correctness

Base case

- i) $a=b$ then $gcd(a,b)=a$
- i) $a>b$ Claim $gcd(a,b) = gcd(a-b,b)$

Let $d = \gcd(a,b)$

Let $h$ be any often common divisor of $a, b$

then $\boxed{d \geq h}$

Well founded ordering relation ] i.e. Every non-empty set has a minimal element w.r.t. the relation

$\Rightarrow d \mid (a-b)$ , $d \mid b$

$\&$ is greatest among such divisors

$\Rightarrow \gcd(a,b) = d = \gcd(a-b, b)$

# Haskell

Basic types : Int, Integer, Char, Bool,
float, Double

Op: *, +, `quot`, `rem`, ....
||, && ⟶ boolean OR, AND

Special object it : Stores the value of the last evaluated expr.

== (Equality check)

:t type
:b browse
:l load
:r reload

if cond then exp else exp

: $t_1$ : $t_2$

same type

## Standard Computation

- Left to right
- top to bottom
  - unless precedence of ( ) & operators is specified

# Compositions

$f(x) = x^2 + 1$

$g(x) = 3x + 2$

for $x = 4$ compute $f(g(x))$

**Innermost Comp.**

$f(3 \cdot 4 + 2)$

$= f(14)$

$= 14^2 + 1$

$= 197$

Q: . Which is more efficient

. Which is easier to implement?

. Which is more correct?

**Outermost Comp.**

$g(4)^2 + 1$

$= (3 * 4 + 2)^2 + 1$

$= 14^2 + 1$

$= 197$

## Exercise

Determine whether a +ive integer is a perfect number or not!

perfect number → Sum of its proper divisors add up to itself.

**Code :**

$$perfect(n) = \begin{cases} True & \text{if } n = add\_factors(n) \\ False & \text{otherwise} \end{cases}$$

$$add\_factors(n) = \{ sum(1, n\ `quot`\ 2) \}$$

why $n\ `quot`\ 2$ ?

$$sum(a, b) = \begin{cases} 0 & \text{if } a > b \\ \text{factor\_value} + sum(a, b-1) & \end{cases}$$

$$\text{factor\_value} = f(i)$$

$$f(i) = \begin{cases} i & \text{if } n \text{ \`remi } i = 0 \\ 0 & \text{otherwise} \end{cases}$$