COL100 : Lec 3

Recap

- Algorithm : A finite process with definite input and output, and is unambiguous!
- Computing models : Functional & Imperative

Functional
- Program is a mathematical Expression
- focus of the class

Imperative
- Program is really a sequence of Commands to be executed

- functional model of computation
  - Primitive Expressions
    - $\mathbb{N}$, $\mathbb{R}$, $\mathbb{B}$, $+$, $-$, $/$, $\wedge$, $\vee$, $\neg$, ...
    - Naming mechanism
  - Methods of Combination
    - Composition of functions
    - Inductive definitions

- Methods of Abstraction
  - functions, data structures, modules, classes etc.

Eg:

$$n!_0 = \begin{cases} 1 & \text{if } n<1 \\ n \times (n-1)!_0 & \text{otherwise} \end{cases}$$

- Uses mathematical induction
- Method of evaluation with termination condition

→ Algorithm

$$= \begin{cases} 1 & \text{if } n<1 \\ \dfrac{(n+1)!_0}{(n+1)} & \text{otherwise} \end{cases}$$

→ NOT an algorithm!

Eg. (Continued)

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

NOT an algorithm

Why?

A: No method of evaluation provided!

In short

Algorithmic Description Constitutes :

1. Directly specified in terms of pre-defined function which is either primitive or there Exists an algorithm to compute it.

2. Specified in terms of the Evaluation of a Condition

3. Inductively defined and validity Established by PMI Variants [all variants of PMI are equivalent]

4. Obtained through finite number of combinations of the above 3 steps.

Eg

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

Correctness

Case 1: if $a = b$, $gcd(a, b)$ is a

Case 2: if $a > b$.

Claim: Every common divisor of a & b is also a divisor of $a-b$ and b.

Proof: Let $d \geq 1$ be a common divisor of $a \Delta b$

Then $a = da'$ and ——— (1)

$b = db'$ ——— (2)

from (1) and (2) we have

that $a - b = d(a' - b')$

$\Rightarrow$ d is a common divisor of $a - b$ and b

Claim : $\gcd(a,b) = \gcd(a-b,b)$

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

Let $d > 0$ be any common divisor of a, b

Clearly $h \geq d$ → well founded ordering relation

By previous claim h is divisor of $a-b, b$ and

h is greatest of them

Well founded ordering relation

if the relation contains no

"Countable infinite descending chains", i.e. no $x_0, x_1, \ldots \in X$ s.t $x_{n+1} R x_n \forall n$.

Eg: Totally ordered relations are well founded

Termination

Case 1 : Trivial

Case 2 : If $a > b$

Easy to see : $a > a - b > 0$

$\Rightarrow$ • process is bounded above by a and below by 0

• the larger of the two arguments in the process traverses down in a strictly decreasing finite sequence $a-1$ $a-2$ .... 0 and hence must terminate.

Careful in applying PMI

Eg: Given any collection of n horses, if atleast one horse is brown then all n horses are brown

Proof:

- $P(n=1)$ is true
- $P(k) \Rightarrow P(k+1)$ can be shown by going from $n=3$ to $n=4$
- Assume $P(n=3)$ is true

  Let H_1 H_2 H_3 H_4 be the horses

- Take a group

H_1 H_2 H_3 (Let H_1 be brown)

by I.H. ($P(n=3)$ is true)

H_2 & H_3 are also brown

- Now take any group with H_4

it will have H_1, or H_2 or H_3

$\Rightarrow$ apply I.H. again

$\Rightarrow$ H_4 is brown too

$\Rightarrow$ $P(n=4)$ true

i.e. we $P(k) \Rightarrow P(k+1)$

have shown

$P(n=1) \Longrightarrow P(n=2)$ fails

- [Can't apply IoHo uniformly to all sets]

- The earlier logic was going from $n = k$ to $n = k+1$

  divide $k+1$ horses in to sets

  Set 1 ⟶ new horse is absent

  Set 2 ⟶ new horse is present → faulty

  Crux of logic: • Set 1 & Set 2 overlap

  • apply IH and say $P(k+1)$ to be true!

Several scientists were asked to prove that all odd integers higher than 2 are prime.

Mathematician: 3 is a prime, 5 is a prime, 7 is a prime, and
by induction - every odd integer higher than 2 is a prime.

Physicist: 3 is a prime, 5 is a prime, 7 is a prime, 9 is an experimental error, 11 is a prime.
Just to be sure, try several randomly chosen numbers: 17 is a prime, 23 is a prime...

Engineer: 3 is a prime, 5 is a prime, 7 is a prime, 9 is an approximation to a prime, 11 is a prime,...

Programmer (reading the output on the screen): 3 is a prime, 3 is a prime, 3 a is prime, 3 is a prime....

Biologist: 3 is a prime, 5 is a prime, 7 is a prime, 9 -- results have not arrived yet,...

Psychologist: 3 is a prime, 5 is a prime, 7 is a prime, 9 is a prime but tries to suppress it,...

Chemist (or Dan Quayle): What's a prime?

Politician: "Some numbers are prime.. but the goal is to create a kinder, gentler society where all numbers are prime... "

Programmer: "Wait a minute, I think I have an algorithm from Knuth on finding prime numbers...
just a little bit longer, I've found the last bug... no, that's not it... “

functions can be algorithmically defined in terms of two main processes

- Recursive
- Iterative

Eg: $$fact(n) = \begin{cases} 1 & \text{if } n=0 \\ n \times fact(n-1) & \text{otherwise} \end{cases}$$

fact (4) = (5 * fact 4)

(5 * (4 * fact(3))
...
(5 * 4 (3 * 2 (1 * fact 0)
(5 * 4 (3 * (2 * (1 * 1))))

Deferred Computation

More Examples on recursive procedures

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

fast_power (x, n)

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

Correctness?

apply induction

Every new variant must be checked for correctness!

Iterative Computational Process

- Via tail recursion

- motivation ?
  - to remove the inefficiencies of recursive processes via deferred Computation

- Key idea here:

iterative algorithms work with the state of computation at each stage (using additional (auxiliary variables) to obtain the final result.

Can be thought as a collection of instantaneous values of certain Entities
Eg: Particle in physics, mass, velocity, pos.

Eg: factorial

at $n=0$ value of fact $(n) = 1$

auxiliary variables

- C s.t. $c_0 \leq c \leq m$

where $c_0 = 0$

$m = n$

captures the counter upto n with one step increments

- f = O/p of factorial function at a particular counter value

$f_0 = 1$ when $c_0 = 0$

At each state a condition on the state of computation holds [called as an invariant]

what is that?

[Some relationship b/w state variables that does not change throughout the computation]

In other words

Invariant must hold

- Initially : when the computation is first invoked
- Continues : holds in every successive invocation of the Computation process

What is that invariant condition for factorial function:

- we know

$$0 \leq c \leq n$$

- for any step c

$$f = f_0 * \prod_{i=c_0+1}^{c} i$$

- finally $f_0 * \prod_{i=c_0+1}^{n} i = f * \prod_{i=c+1}^{n} i$

The resulting algorithm

$$factorial(n) = fact\_iter(n, 1, 0)$$

$$: \mathbb{N} \times \mathbb{P} \times \mathbb{N} \rightarrow \mathbb{P}$$

where

$$fact\_iter(n, f, c) = \begin{cases} f & \text{if } c = n \\ fact\_iter(n, f*(c+1), c+1) & \text{otherwise} \end{cases}$$

Why are they tail recursive?

'otherwise' clause here is recursive with just the function call

[In recursive factorial 'otherwise' clause had n * fact (....)]

This involves a multiplication + recursive call!

factorial 5)

= fact_iter (5, 1, 0)

= fact_iter (5, 1, 1)

= fact_iter (5, 2, 2)

= fact_iter (5, 6, 3)

= fact_iter (5, 24, 4)

= fact_iter (5, 120, 5)

= 120

Proof of Correctness

Basis : when $m = c$

$$\text{fact\_iter}(m, f, c) = f * \prod_{i=c+1}^{m} i$$

$$= f * 1 = f$$

I.H. : for some $k = m - c \geq 0$

$$\text{fact\_iter}(m, f, c) = f * \prod_{i=c+1}^{m} i$$

I.S : Let $(m - c) = k + 1 > 0$

then $\text{fact\_iter}(m, f, c) = \text{fact\_iter}(m, f * (c+1), c+1)$

$$= f * (c+1) * \prod_{c+2}^{m} i$$

$$= f * \prod_{c+1}^{m} i$$