CSN312 POPL **Lecture 1** **Spring 2025**

Course credit: 3

Class hours: M Th F M, Th: 4-5pm F: 5-6pm **Tutorial**: M 11-12, 12-1

Components: CWS 35 MTE 25 ETE 40

Lecture notes and Related materials would be provided from time to time.

Instructor: Rajdeep Niyogi rajdeep.niyogi@cs.iitr.ac.in

Introduction

What do we mean by programming languages (PLs)?

When we say PL we invariably mean High Level PLs.

Low level PLs: machine language, assembly language

Machine language: best language for the m/c but the most difficult language for a human

Early days of programming: programs were written in machine language and the input was fed using punched cards.

Assembly language: understanding by a human becomes a lot easier but it is tied with machine architecture.

A high level PL should be independent of the machine architecture.

- PL classification: imperative (e.g,, Fortran, Pascal, C)
  object oriented (e.g., C++, java, python)
  functional (e.g., scheme, Haskell)

examples of functional languages include Lisp, Haskell, Scheme, ML, etc.

- Paradigms of different PLs: how-to-do (e.g, Fortran, Pascal, C, C++, java, python)
  what-to-do (e.g., scheme, Haskell)

  How-to-do: machine will execute the instructions as given in the code. m/c will not do anything else but this.

  What-to-do: tell the machine what you want to be done. The m/c will do it in some manner which the programmer need not bother about.

  Eg, quicksort program: sketch the code in your favorite PL. this is how-to-do.

  There are two parts:

  Input (A, p, r)

  1. Obtain two partitions of A: (A, p, q) and (A, q+1, r) on which apply quicksort

2. Partition function

Input (A,p,r)

Obtain an index j that creates two partitions of A wrt a pivot x s.t

All elements in (A,p,j) are <= x and all elements in (A,j+1,r) are > x

50 70 10 55 65 30 40 20

20 40 10 30 65 55 70 50 x=50 j=4

What-to-do: this means we need to provide the definition of a function as a code

Can we write a quicksort program in say within 2 lines of code? This is what-to-do

Let us revisit the idea of partition:

Now input is in the form of a list e.g., [1,5,7] same as 1:[5,7]

Let ++ be an operator that concatenates two lists, e.g. [1,4] ++ [3,5] will give [1,4,3,5]

For a pivot x, apply quicksort on two sublists left and right s.t. ...

All elements in left are <= x and all elements in right are > x

and then combine them as

Qsort(x:xs) = Qsort(left) ++ [x] ++ Qsort(right)

left = [20,40,10,30] right = [70,55,65]

left = [30,10,40,20] right = [65,70,55]

Suppose you want to implement QSort in a C-like language. The heart of the code is the partition function. What is the guarantee that your implementation of the function is correct? It is not easy to look at any such implementation and say that it is correct. You need a piece of paper and a pencil to work it out.

See the above code—it is the definition of partition function and thus the meaning of Qsort. So it is easy to see that the code is correct.

Moral of the story: code in a functional PL gives the definition of the function that it implements.

In general, the size of such codes is small but designing the code is nontrivial.

We will study the design of a generic Functional PL called Lambda Calculus.

Motivation:

LC is the basis for other functional PLs.

Java, C#, and some other non-functional PLs have now included LC as a component.

Historical background: λ-calculus was developed by Alonzo Church in the 1930s.

Church (1903-1995)

Turing (1912-1954) (Church's student)

Church's idea: to understand computation via functions. (LC)
Turing's idea: to understand computation via states. (TM)--1936
Church-Turing hypothesis: LC and TM are equivalent.

Any program written in any PL can be encoded in LC.

`Turing award` (Noble prize in computing) and `Turing test` are named after Alan Turing, who is also one of the founding fathers of AI.

Our **MOTTO** from now on would be:

- Programming is a mechanical activity
- Establishing that a program is correct

Course objectives:

This course attempts to provide training on the following:

We shall learn how to compose programs using a mathematical language, which is the source of many modern functional programming languages. We wish to learn the foundations of functional programming (lambda calculus).

We shall design a tool by which we can mechanically claim that a program is (in)correct.

We shall study the mathematical objects corresponding to the OO programming constructs.

=