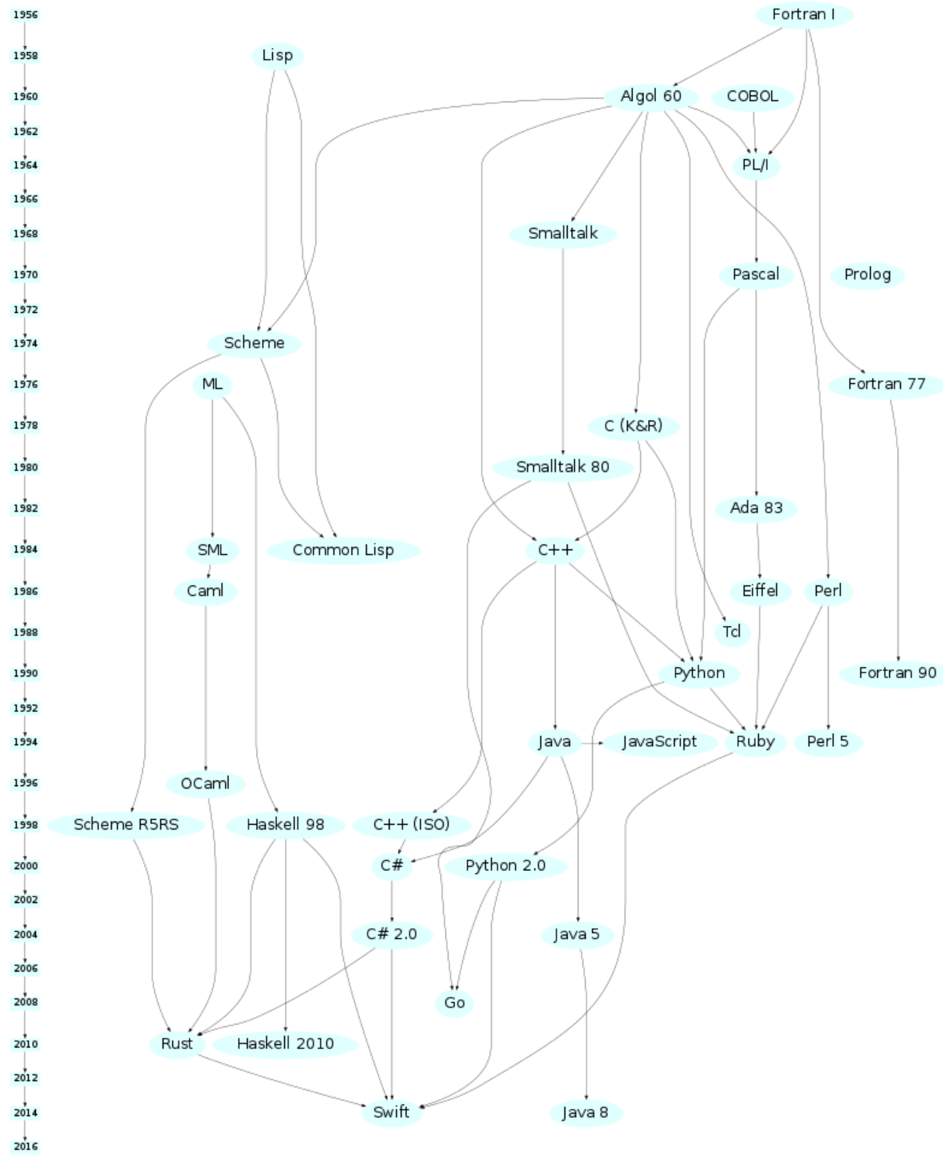
Lecture 12: Functional Programming

CS 51P

October 16, 2019

A History of Programming Languages



Programming Paradigms

- Imperative Languages
  - programs are sequences of instructions that tell the computer how to modify the program state (i.e., update values in variables)
  - Procedural Languages:
    - programs are organized sets as functions (aka. procedures) called modules
    - Examples: C, Python
  - Object-Oriented Languages:
    - programs are organized as values (called objects), each type of object has a set of component values (fields) and a set of functions (methods)
    - Examples: C++, Java, Swift, Python!

- Functional Languages
  - programs define what stuff is
  - Examples: Haskell, OCaml, F#, Coq

Pure Functional Programming Languages

- stateless
- use recursion instead of loops

- Today we'll use OCaml as an example. It's not actually purely functional, but for simplicity we'll pretend it is

OCaml Values and Types

- # 47;;
  - : int = 47

- # 47.;;
  - : float = 47.

- # true;;
  - : bool = true

- # ‘a’;;
  -: char=’a’

- # “CS51p”;;
  - : string = “CS51p”

Function Values

- # fun n ->n + 47 ;;
  - : int ->int = <fun>

- We just defined a (anonymous) function

- It takes one argument n
- The function returns the sum of the argument and 47
- The function is of type int -> int

OCaml Expressions

- Same general idea
- Slightly different syntax
- No shared operators, no automatic casting,

- # 33 + 14;;
  - : int = 47
- # 33. +. 14.;;
  - : float = 47.
- # 33. +. 14;;
  Error

Variables and let

- Use keyword let to define (ie., bind a value to) a variable

- # let v = 33 + 14 ;;
  val v: int=47

- Same syntax to define a function

- # let add47 n = n + 47 ;;
  val add47 : int -> int = <fun>

- So functions are values!

Applying Functions

- # add47 1 ;;
  -: int=48
- # add47 (add47 1) ;;
  -: int=95
- # add47 (add47 (add47 (add47 1)));;

  -: int=189

Multi-Parameter Functions

- # let xPlus2y x y = x + 2 * y;;
  val xPlus2y : int -> int -> int= <fun>
- We just defined a function called xPlus2y
  - It takes two arguments x and y
  - It computes x + 2y
  - The function is of type int -> int -> int

- xPlus2y 6 5;;
  -: int=16
- xPlus2y 6;;
  -: int -> int = <fun>
- (xPlus2y 6) 5;;
  -: int =16

Conditionals

if boolExp then exp1 else exp2

- Conditionals are expressions (they evaluate to values)

- # if true then 1 else 2;;

  -: int=1
- # if false then 1 else 2;;

  -: int=2

Recursion instead of Loops

- Recursion: solving a problem by solving smaller problem(s) of the same type.

$$\text{sum}(n) = 1 + 2 + 3 + \cdots + n$$
$$= \text{sum}(n - 1) + n$$

- # let rec sum n= if n<=0 then 0 else n + (sum (n-1));;

  val sum : int -> int = <fun>
- # sum 5;;
  -: int=15
- # sum 47 ;;

  -: int=1128

- Exercise: how would you define fib?

Recursion is very general

```python
def sum_squares(n):
  sum = 0
  for i in range(1,n+1):
    sum += i*i
  return sum
```

```ocaml
let rec sumsquare n =
  if n <= 0 then 0
  else n*n +
    (sumsquare (n-1));;
```