L11: Continuations and Currying

Rachata Ausavarungnirun

(rachata.a@tggs.kmutnb.ac.th)

February 11th, 2020

Architecture Research Group

Software System Engineering

Thai-German Graduate School, KMUTNB

Quiz 1 Next Week

- Please bring in your laptop

- More choices for you:
  - In-class exam (2 hours)
  - Take-home exam (24 hours)

Continuations

- So far, all our functions return something
- You can also call a new function at the end
  - This is called the continuation passing style (CPS)

    ↳ Once finish the function, continue another function

- This allows you to make every function a tail call
- Let's use a sum of all integer as an example

  def sum(L: List[Int]): Int = L match {

  case Nil => 0 case x::xs => sum(xs) + x

Continuations

- Tail call version:

- sum_tc(L: List[Int]): Int = {   [Accumulator]

```
  def sumHelper(L: List[Int], a: Int): Int = L match {
    case Nil => a
    case x::xs => sumHelper(xs, a + x) }
  sumHelper(L, 0) }
```

Initial number = 0

- Continuation version

- def sum_cont(L: List[Int]): Int = {   [Take Int return Int (function)]

```
  def sumHelper(L: List[Int], K: Int => Int): Int = L match {
    case Nil => K(0)
    case x::xs => sumHelper(xs, (r: Int) => K(r + x)) }
  sumHelper(L, (x: Int) => x) }
```

- Base Case

[List(h1, h2)]

$h_1 + h_2 + 0$

Create Base Case (0) => h_1 + func(x=>x) => $h_1 + h_2$ + func(x=>x)

Example 2: Binary Tree

- Fundamentally, a binary tree can be defined recursively
  - A node can be
    - Empty
    - A node with two sub-tree
- Let's make the trait Tree
- sealed trait Tree - Tree Object
  case object Empty extends Tree - Empty
  case class Node(**left: Tree, key: Int, right: Tree**) extends Tree

Example 2: Binary Tree

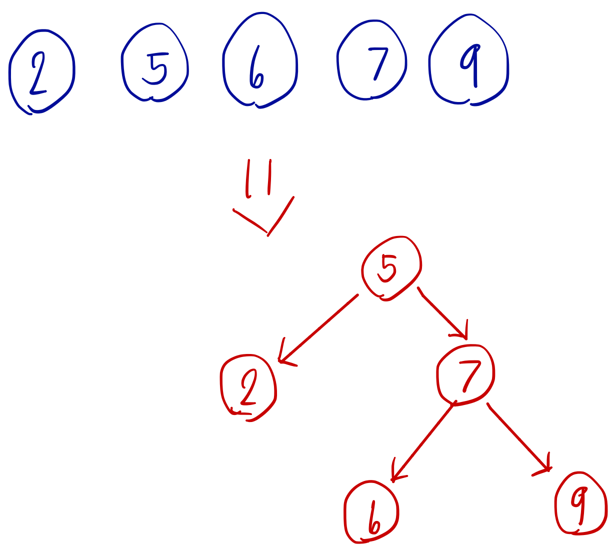
- Making a tree can be done by

[Left] [Root]

- val t = Node(Node(Empty, 2, Empty), 5, Node(Node(Empty, 6, Empty), 7, Node(Empty, 9, Empty)))

[Right]

- What does this tree looks like?



Example 2: Binary Tree Traversal

[Tree as input]

- def walkInorder(t: Tree): List[Int] = t match {
  case Empty => Nil - Tree is nothing
  case Node(l, k, r) => walkInorder(l):::(k::walkInorder(r))
  }
  ↳ Walk left subtree
  Append list to another list

- What if we want to use continuation?
  - We need to pass done the functions to call at the end
  - What should that function do?

Example 2: Binary Tree Traversal

- def walkInorder(t: Tree): List[Int] = {
  def contWalk(t: Tree, K: List[Int] => List[Int]): List[Int] = t
  match {

  ↳ Take List, produce List

  case Empty => K(Nil) - K is empty, put in nothing

  Build left subtree

  case Node(l, k, r) => contWalk(l, leftList => {
  contWalk(r, rightList => K(leftList:::(k::rightList)))
  })
  }

  Build right subtree

  root

  Appen root to right

  Combine left with right

  contWalk(t, (r: List[Int]) => r)
  }

Currying

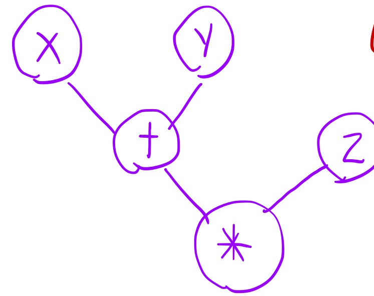
- Instead of accepting parameters normally, accept through a sequence of functions
- def sortedUncurry(x: Int, y: Int, z: Int) = x <= y && y <= z
- val sortedTriple = { (x: Int) => (y: Int) => (z: Int) => x <= y && y <= z }

- Currying version:
  - def sortedTriple **(x: Int) (y: Int) (z: Int)** = x <= y && y <= z

↳ Individual 3 items

↳ Once x, y are defined, do the computation

Currying



Once you have x, y, start adding

- Benefits:
  - You can stage the function
    - Parts of the execution can run as soon as the values are ready
  - Maps well with dataflow model
  - This can allow the compiler and the hardware to be faster
    - Eliminate data dependency as soon as possible

- Actual efficiency: It depends
  - Compiler is very smart nowadays
  - Run a profiler to test the two formats

States and Mutable Variables

- We assumed variables are immutable
  - This is annoying in some cases
  - What if we need to store a state

[Ex. temp = A, A = B, B = temp]

- **State:** the intermediate steps that need to be stored
  - Real hardware also needs the concept of state

- So, many functional languages have mutable variables

- Benefit of mutable variables
  - Allow programmer to keep the state

Declaring Mutable Variables

- var x = value

- Example: implementing a while loop
  - Using currying and mutable variables

- def my_while (condition: => Boolean) (block: => Unit): Unit = {

```
def my_while (condition: => Boolean) (block: => Unit):
Unit = {
    if (condition) {
        block
        my_while (condition) (block)
    } else ()
}
```

Before We Leave Today

In-class Exercise 11

- Implement fibonacci recursively in the continuation-passing style

- Preorder traversal: visit root first, then left, then right. Write preorderWalk in CPS style