Principles of (Functional) Programming

(4190.306)

Chung-Kil Hur (허충길)

Department of Computer Science and Engineering
Seoul National University

Syllabus

- Lecture
  - Mon & Tue, 9:00 ~ 10:50 (302-208)
  - https://github.com/snu-sf-class/pp201602
- Instructor
  - Chung-Kil Hur
  - http://sf.snu.ac.kr/gil.hur/
- Teaching Assistant
  - Youngju Song
  - http://sf.snu.ac.kr/youngju.song/
- Grading
  - Attendance: 5%
  - Assignments: 25%
  - Midterm exam: 30%
  - Final exam: 40%

Schedule

- Functional Programming Principles in Scala
  - https://www.coursera.org/learn/progfun1/
  - Week 1: Functions and Evaluations
  - Week 2: Higher-order Functions
  - Week 3: Data and Abstraction
  - Week 4: Types and Pattern Matching
  - Week 5: Lists
  - Week 6: Collections
- Functional Program Design in Scala
  - https://www.coursera.org/learn/progfun2/
  - Week 7: Expressions and Monads
  - Week 8: Lazy Evaluation
  - Week 9: Functions and State
  - Week 10: Timely Effects

Introduction

Imperative vs. Functional Programming

- Imperative Programming
  - Computation by memory reads/writes
  - Sequence of read/write operations
  - Repetition by loop
  - More procedural
  - Easier to write efficient code

```
sum = 0;
i = n;
while (i > 0) {
  sum = sum + i;
  i = i - 1;
}
```

- Functional Programming
  - Computation by function application
  - Composition of function applications
  - Repetition by recursion
  - More declarative
  - Easier to write safe code

```
def sum(n) =
  if (n <= 0)
     0
  else
     n + sum(n-1)
```

Both Imperative & Functional Style Supported

- Many languages support both imperative & functional style
  - More imperative: Java, Javascript, C++, Python, …
  - More functional: OCaml, SML, Lisp, Scheme, …
  - Middle: Scala
  - Purely functional: Haskell

- Why Scala?
  - Equally well support both imperative & functional style
  - A lot of advanced features
  - Compatible with Java

Functions and Evaluations

Values, Expressions, Names

- Types and Values
  - A type is a set of values
  - Int: {-2147483648,…,-1,0,1, …,2147483647} //32-bit integers
  - Double: 64-bit floating point numbers // real numbers in practice
  - Boolean: {true, false}
  - …
- Expressions
  - Composition of
    values, names, primitive operations
- Name Binding (= Programming)
  - Binding expressions to names
- Examples

```
def a = 1 + (2 + 3)
def b = 3 + a * 4
```

Evaluation

- Evaluation
  - Reducing an expression into a value
  - Strategy
  1. Take a name or an operator (outer to inner)
  2. (name) Replace the name with its associated expression
  3. (name) Evaluate the expression
  4. (operator) Evaluate its operands (left to right)
  5. (operator) Apply the operator to its operands

- Examples

```
5+b ~ 5+(3+a*4) ~ … ~ 32
```

Functions and Substitution

- Functions
  - Expressions with Parameters
  - Binding functions to names

```
def f(x: Int): Int = x + a
```

- Evaluation by substitution
  - …
  - (function) Evaluate its operands (left to right)
  - (function)
    Replace the function application by the expression of the function
    Replace its parameters with the operands

```
5+f(f(3)+1) ~ 5+f((3+a)+1) ~ … ~ 5+f(10) ~
5+(10+a) ~ … ~ 21
```