Principles of Programming Languages – midterm review

Wael Aboelsaadat

wael@cs.toronto.edu

http://www.cs.toronto.edu/~wael

Chapter 1: introduction

- **Abstraction levels of programming languages**
  - Machine language
  - Assembly language
  - High-level

- **Language translation:**
  - Compilation
  - Interpretation
  - Pseudo compilation

- **Language Paradigms:**
  - Imperative
  - Object-oriented
  - Functional
  - Logic-based

Chapter 1: introduction - cont'd

- **Characteristics of a "good" language**

	Criteria		
Characteristic	Readability	Writability	Reliability
Simplicity	✓	✓	✓
Control Structures	✓	✓	✓
Data types & structures	✓	✓	✓
Syntax design	✓	✓	✓
Support for abstraction		✓	✓
Expressivity		✓	✓
Type checking			✓
Exception handling			✓
Restricted aliasing			✓

Chapter 3&4: Syntax

- **Language Specification:**
  - Syntax-formal methods
  - Semantics-informal descriptions

- **Context-free grammars:**
  - How are CFG's descriptions of languages?
  - Derivations
  - Parse trees
  - Ambiguity
  - Fixing an ambiguous grammar:
    - Introducing delimiters
    - Imposing associativity/precedence
    - Inherently ambiguous grammar
    - Limitations of CFGs

Chapter 3&4: Syntax – cont'd

- **E.g.:**

  Using the following grammar:

  <assign> → <id> = <expr>
  <id> → A | B | C
  <expr> → <id> + <expr>
  | <id> * <expr>
  | (<expr>)
  | <id>

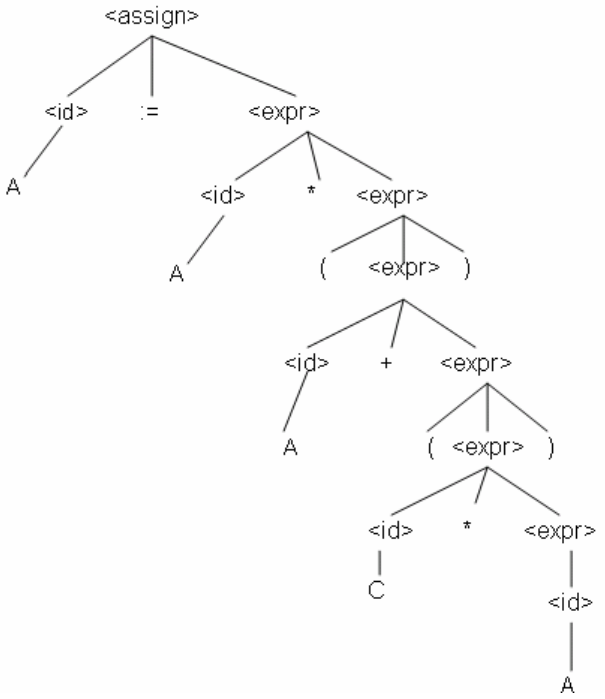
  Show a parse tree and a leftmost derivation for

  A = A * (B + (C * A))

Chapter 3&4: Syntax – cont'd

- **Answer:**

<assign> => <id> = <expr>
=> A = <expr>
=> A = <id> * <expr>
=> A = A * <expr>
=> A = A * (<expr>)
=> A = A * (<id> + <expr>)
=> A = A * (B + <expr>)
=> A = A * (B + (<expr>))
=> A = A * (B + (<id> * <expr>))
=> A = A * (B + (C * <expr>))
=> A = A * (B + (C * <id>))
=> A = A * (B + (C * A))



Chapter 5: Binding & Scope

- **What is the issue?**
- **Static vs. dynamic scope**
- **What are the scope-rules design choices?**

Chapter 5: Binding & Scope – cont'd

- **E.g.:**

Assume the following program was compiled and executed using static scoping rules. What value of **x** is printed in procedure **sub1**? Under dynamic scoping rules, what value of **x** is printed in procedure **sub1**?

```
program main;
  var x : integer;
  procedure sub1;
    begin  { sub1 }
    writeln('x =', x)
    end;  { sub1 }
  procedure sub2;
    var x : integer;
    begin  { sub2 }
    x := 10;
    sub1
    end;  { sub2 }
  begin  { main }
  x := 5;
  sub2
  end.  { main }
```

Chapter 6: Data types

- **Primitive vs. structured data types**
- **Pointer problems**
- **GC algorithms**
  - Reference counting
  - Mark and sweep GC
  - Copying GC
- **What are the design choices?**

Chapter 6: Data types

- **E.g.**

  Develop the access functions for three-dimensional arrays using row major order and using column major order?

Chapter 6: Data types – cont'd

- **Answer:**

Let the subscript ranges of the three dimensions be named min(1), min(2), min(3), max(1), max(2), and max(3).

Let the sizes of the subscript ranges be size(1), size(2), and size(3).

Assume the element size is 1.

Row Major Order:

location(a[i,j,k]) = (address of a[min(1),min(2),min(3)])
+((i-min(1))*size(3) + (j-min(2)))*size(2) + (k-min(3))

Column Major Order:

location(a[i,j,k]) = (address of a[min(1),min(2),min(3)])
+((k-min(3))*size(1) + (j-min(2)))*size(2) + (i-min(1))

Chapter 7: Expressions

- **E.g.**

  Let the function FUN be defined as

```
function FUN( var k : integer ) : integer;
begin
   K := K + 4;
   FUN := 3 * K – 1;
end;
```

  Suppose FUN is used in a program as follows:

```
   I := 10;
   SUM1 := (I/2) + FUN(I);
   J := 10;
   SUM2 := FUN(J) + (J/2);
```

  What are the values of SUM1 and SUM2

  a. if the operands in the expressions are evaluated left to right?

  b. if the operands in the expressions are evaluated right to left?

Chapter 7: Expressions – cont'd

- **Answer:**

  a. SUM1 = 46 and SUM2 = 48

  b. SUM1 = 48 and SUM2 = 46

Chapter 15: Functional PL

- **Characteristics**
- **Basic constructs**
- **Lists**
- **Unnamed functions**
- **Eval, map, apply,..**
- **Let, let***