City University London

IN2009
Language Processors

Epilogue (Week 11)

Revision

Igor Siveroni

What's in store this week

- Revision of weeks 1-5, 7-10
- The Exam
- Any other questions

Guide to textbook

- Chapter 1
- Chapter 2 (2.1, 2.2, 2.5)
- Chapter 3 (3.1, 3.2 [but not after page 47 FIRST and FOLLOW until page 51 Eliminating Left Recursion, 3.4)
- Chapter 4 (4.1, 4.2, 4.3)
- Chapter 5 (5.1 [but not functional symbol tables], 5.2)
- Chapter 6 (not higher-order functions, 6.1, sketch of 6.2),\\
- Chapter 7 (7.1, 7.2, 7.3 [as covered by Session 7 foils])

Session 1 recap

Session 1
- Module details, aims, resources
- What is language processing and implementation?
- Syntax definition
- Straight-line programming language
- Abstract syntax trees
- Some Java reminders...

Session 2 recap

Session 2
- Language processing
- Lexical analysis
- Syntax analysis
- Lexical syntax (token) examples
- Lexical syntax (token) definition
- Regular expressions
- Implementation
- Tools

Session 3 recap

Session 3: Parsing (syntax analysis)
- syntax definition
  - context free grammars (BNF)
- parsing
- ambiguous grammars
- removal of left recursion
- top down recursive descent parsing
- extended BNF (EBNF)
- parsing using JavaCC

Session 4 recap

Session 4: Parsing (abstract syntax)

Covered in weeks 4 and 5.

- MiniJava introduction and parsing
- Lookahead
- JavaCC grammars and semantic actions and values
- Simple expression evaluator
- Abstract syntax trees

Session 5 recap

Session 5: Semantic analysis

- Symbol tables
  - Environments
  - Hash tables
  - Symbol table for MiniJava
- Typechecking

Session 6 recap

Session 6: Activation Records (Stack Frames)

- Memory model
- Local variables
- Stack frames
  - layout
  - frame pointer and stack pointer
  - parameter passing
  - calling conventions
- Static links

Session 7 recap

Session 7: Translation to intermediate representation

- Intermediate representation
- Why use IR
- Definition of an IR using trees
- Example translations
- See book for while-loops, for-loops, functions, declarations

The Exam

- 1.5 hours in length
- Answer TWO questions from a choice of three.

What will be on the exam?

Q. What topics might be on the exam?

A. Everything.

Next…

- Keep an eye on Cityspace in the next few weeks
  - Sample answers for courseworks.
  - This week: Marks for courseworks 2 and 3.
- Some sample questions…

Q1

- Consider the following grammar for strings of balanced parentheses:

  S → S S
  S → (S)
  S → a
- Explain what it means for a context-free grammar to be ambiguous. Using your explanation show that the balanced-parenthesis grammar is ambiguous using the shortest string that will illustrate the ambiguity.

Q2

- Supose the reference manual for a MiniJava-like programming language contains the following grammar for a repeat-until statement:

  Statement → ***repeat*** *Statement* ***until*** *(Exp)*
  - Sketch a possible abstract syntax for the *repeat-until* statement.
  - Show how semantic actions in a grammar for a parser-generator such as JavaCC can be used to produce abstract syntax trees for the *repeat-until* statement.

Q2…cont

- Informally describe an appropriate typecheck for the *repeat-until* statement.
- Outline the intermediate code that might be generated in translation of the *repeat-until* statement to IR trees. You may wish to use a simple example to explain your translation:

```
repeat
  { sum = sum + x; prod prod * x; }
until (x < 20)
```


Q3

- The following regular expression recognises certain strings consisting of the letters a, b and c:

  (a|c)((bc)|c)* c*
  - For the following 5 strings, indicate whether or not they are recognised by the above regular expression: cbc, abbc,c,abcbcbccc,ccbbcbcc
  - Show three more strings that are recognised by the above expression
  - Show two more strings consisting of the letters a, b and c that are ***not*** recognised by the above regular expression.

Q4

- Explain why left-recursion must be eliminated from grammar productions which which are to be used in the construction of a recursive-descent parser.
- Write down a general rule for rewriting left-recursive grammar productions to equivalent right-recursive grammar productions.
- Use the rule above to rewrite the following productions

E → E + T   E → T   T → T * F
T → F   F → (E)   F → integer

Q5

- Consider the following piece of code. Given an initial environment σ_0 derive the type binding environments for the method at each use of an identifier and indicate where type lookups will occur:

```
class Console {
  String name; int float;
  public void add(String game, String platf) {
    System.out.println(name);
    int score = 10;
    int a = score + 5;
    System.out.println(platf); System.out.println(a);
   System.out.println(game); System.out.println(score)
  }
}
```