Midterm Review

COMP 524: Programming Languages
Srinivas Krishnan
March 14, 2011

Based in part on slides and notes by Bjoern Brandenburg, S. Olivier and A. Block.
Lay of the Land

Question Types

- Simple True/False, Fill in the Blanks
- Problem Solving: Read a scenario and apply what you have learnt so far
- Theoretical Problems: Similar to quiz questions on Lexical and Syntactic Analysis
- Code: Write Snippets of Code, Read and Evaluate Code
- Extra Credit: Problem Solving based on class lectures
Lay of the Land

Nuts n Bolts

- 75 Minutes
- Closed Book
- Total Points: 75-100
- Coverage: Introduction till Scope, Topic List on website
Recap Intro

- What’s the relationship between assembly and machine code?
  - A direct mapping; one-to-one correspondence.
  - Assembly is converted into machine code.

- Name three limitations of programming on the assembly level.
  - Hard to read and write for humans.
  - Not portable.
  - Error-prone due to lack of checking

- Which major language is the ancestor of Pascal, C, Java
  - Algol
Recap Intro

➡ What are the two main language design paradigms
   ‣ Declarative and Imperative

➡ List the three major sub-categories of declarative languages and examples of each
   ‣ Functional programming. E.g., Haskell.
   ‣ Logic/constrained-based programming. E.g., Prolog.
   ‣ Dataflow. E.g., Id, Val.

➡ What is the design metaphor of object-orientation
   ‣ Inspired by human organization.
   ‣ Communicating experts.
   ‣ Problem decomposition by delegation to “domain specialists”.
   ‣ Objects send and respond to messages.
   ‣ Problem is solved by “team” of “collaborating” “workers.”
Recap Compilation and Interpretation

Name the translation strategy for each of the following languages: Fortran, Lisp, Python, Java, Shell, C, C++, C#

- Fortran: separate compilation + linking
- Lisp: either interpretation or compilation
- Python: implicit compilation + interpretation
- Java: explicit compilation + interpretation
- Shell: interpretation
- C: separate compilation + linking
- C++: separate compilation + linking
- C#: explicit compilation + interpretation
Recap Compilation and Interpretation

➡ “missing symbol:” What kind of error is this? (compile-time, link-time, run-time)
  ‣ Link-Time

➡ What are the two requirements for a language to be a programming language
  ‣ It must be universal and implementable

➡ What are the three main design goals in programming language design
  ‣ Developer productivity.
  ‣ Program safety.
  ‣ Program efficiency
Recap Compilation and Interpretation

What’s conditional compilation? Why would you use it?

- A pre-processor technique. Some code is only compiled if specific flags are defined.

- This can be used to configure a complex code base for a specific use. For example, debugging checks can be disabled in release builds, and architecture-dependent features can be enabled or disabled depending on the choice of target architecture (e.g., Linux).
Recap Compilation and Interpretation

- How are compilers built from scratch?
  - Written in the language they implement
  - Self-Hosting
  - Process called Bootstrapping

- What changes if you already have a compiler for some platform?
  - Can use cross-compilation to derive other compilers
Example: Cross-Compilation

Going from Intel x86 to Sun’s SPARC V9.

x86 ➜ x86
V9 ➜ V9

Source

Host Compiler

Input

Cross Compiler

Target Compiler

Input ➜ gcc: x86 ➜ x86

Cross Compiler ➜ cv9: x86 ➜ V9

Target Compiler ➜ cv9: V9 ➜ V9
Recap Lexical Analysis

➡ What is a token?
  ‣ Program fragment with atomic meaning
➡ How can we specify tokens?
  ‣ With regular expressions
➡ How can we recognize tokens efficiently
  ‣ Using DFAs
➡ Why is there a separate lexical analysis phase?
  ‣ To simplify the following syntactical analysis phase.
➡ Does an assembler need a lexical analysis phase
  ‣ Yes, it needs to recognize instructions, operands, and macros.
➡ How do we get from a regular expression to a DFA
  ‣ Via an NFA
Recap Lexical Analysis

- Draw the NFA corresponding to the regular expression for a|01
Recap Lexical Analysis

Draw the NFA corresponding to the regular expression: $x(yx)^*0$
Recap Lexical Analysis

Convert the previous NFA to DFA
Recap Lexical Analysis

▶ Minimize the DFA
Recap Syntax Analysis

- Why can’t we specify all programming language syntax with regular grammars?
  - Because regular grammars cannot express most recursive structures (such as balanced parenthesis)
- What is a derivation? (in the context of grammars)
  - A sequence of string-to-string mappings that is obtained by substituting a non-terminal symbol with the body of one of its productions in each step.
- Which are the two most-commonly used grammar classes to implement O(n) parsers
  - LL and LR.
Recap Syntax Analysis

What does LL and LR mean
  ‣ LL: left-to-right, left-most derivation
  ‣ LR: left-to-right, right-most derivation
Recap Syntax Analysis

- Describe what this CFG does in minimal words
  - The language consists of all strings of balanced parentheses and brackets

```plaintext
<P> -> <E> $$
<E> -> <A>
<E> -> <B>
<E> -> λ
<A> -> (E)E
<B> -> [E]E
```

Question-6

Postfix Notation also known as reverse Polish is a form of arithmetic expression in which the operator is written after operands. e.g., XY will be rewritten in Polish notation as YX. The grammar below describes this notation:

```plaintext
Exp → Number
Exp → Exp Unary Operator
Exp → Exp Exp Binary Operator
Unary Operator → ∼ |
S
Binary Operator → + − ∗ /
```

You may think of ∼ as denoting additive inverse (what is usually marked as ± on a calculator). You may think of S as denoting factorial. Rest of the question uses this grammar.

lh
- Give a parse tree for the string o o l e d
mh
- Perform a leftmost derivation of the same string
nh
- Give reasons as to why this is not LL(1) and write the equivalent LL(1) grammar

Question-7

Recall that we used two clauses for mother and father to describe a royal family tree in class. For this problem, assume that a couple has kids if and only if they are married. This is after all a very honorable and royal family.
## Predict Sets

<table>
<thead>
<tr>
<th>Rule</th>
<th>First</th>
<th>Follow</th>
<th>Predict</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P \rightarrow E )</td>
<td>([\lambda])</td>
<td>()</td>
<td>([\ ])</td>
</tr>
<tr>
<td>( E \rightarrow A )</td>
<td>()</td>
<td>(\ ))</td>
<td>()</td>
</tr>
<tr>
<td>( E \rightarrow B )</td>
<td>([\ ])</td>
<td>()</td>
<td>()</td>
</tr>
<tr>
<td>( E \rightarrow \lambda )</td>
<td>(\lambda)</td>
<td>()</td>
<td>()</td>
</tr>
<tr>
<td>( A \rightarrow (E) E )</td>
<td>()</td>
<td>()</td>
<td>()</td>
</tr>
<tr>
<td>( B \rightarrow [E] E )</td>
<td>([\ ])</td>
<td>()</td>
<td>()</td>
</tr>
</tbody>
</table>
Using Predict Sets

Is this LL(1)?

- E is the only nonterminal on the left-hand side of more than one production, and the predictsets of those productions are disjoint, so yes, the grammar is LL(1).

<table>
<thead>
<tr>
<th>Production</th>
<th>First</th>
<th>Follow</th>
<th>Predict</th>
</tr>
</thead>
<tbody>
<tr>
<td>P → E</td>
<td>([λ]</td>
<td>$$</td>
<td>([λ $$</td>
</tr>
<tr>
<td>E → A</td>
<td>(</td>
<td>$$ ) ]</td>
<td>(</td>
</tr>
<tr>
<td>E → B</td>
<td>[</td>
<td>) ] $$</td>
<td>[</td>
</tr>
<tr>
<td>E → λ</td>
<td>λ</td>
<td>) ] $$</td>
<td>) ] $$</td>
</tr>
<tr>
<td>A → (E) E</td>
<td>(</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td>B → [E] E</td>
<td>[</td>
<td></td>
<td>[</td>
</tr>
</tbody>
</table>
Recap Syntax Analysis

→ Draw Parse Tree[()]() $$
Give a left most derivation for \[(())(])\] $$
- P 
- E $$
- B $$
- [E]E $$
- [A]E $$
- [(E)E]E $$
- [(]E)E $$
- [()]E $$
- [()]E $$
- [(]A $$
- [(]E $$
- [(]E $$
- [(]() $$
- [()] $$

$$ 
< P \rightarrow < E > $$
< E \rightarrow < A > $$
< E \rightarrow < B > $$
< E \rightarrow \lambda $$
< A \rightarrow (E)E $$
< B \rightarrow [E]E $$
Give a right most derivation for \([()]()\):

- \(P\)
- \(E\)
- \(B\)
- \([E]E\)
- \([E]A\)
- \([E](E)E\)
- \([E](E)\)
- \([A]()\)
- \([(E)E]()\)
- \([(E)E]()\)
- \([()]()\)


\(<P> \rightarrow <E> \rightarrow <A> \rightarrow <B> \rightarrow \lambda\)

\(<E> \rightarrow (E)E\)

\(<B> \rightarrow [E]E\)
Scope and Name Binding

- What are the two principle sources of memory waste in heap allocation
  - Internal and external fragmentation
- What is the referencing environment
  - The set of all active bindings; all valid names
- What is a Stack Frame
  - Holds all the variables, arguments and addresses for the currently executing procedure
- How does the stack grow in an x86 machine?
  - Downwards
- What are the common causes of memory leaks in C?
  - Dangling Pointers
  - Manual deallocation
public class Scope {
    static int x = 10;

    static void printX(String from) {
        System.out.println("from "+from+": x = " + x);
    }

    static void test0() {
        int x = 0;
        printX("test0");
    }

    static void test1() {
        int x = 1;
        test0();
        printX("test1");
    }

    public static void main(String... args) {
        test1();
        printX("main");
    }
}

Output:

from test0: x = 10
from test1: x = 10
from main: x = 10

from test0: x = 0
from test1: x = 1
from main: x = 10