CS541 class notes

Raphael Finkel

August 30, 2018

1 Intro

• Class 1, 8/22/2018
• Handout 1 — My names
  • Mr. / Dr. / Professor / —
  • Raphael / Rafi / Refoyl
  • Finkel / Goldstein

• Extra 5-10 minutes on every class to correct for missed sessions?
• Plagiarism — read aloud from handout 1
• Assignments on web. The first is very easy, the rest not, so start immediately.
• E-mail list: cs541001@cs.uky.edu; instructor uses to reach students.
• All students have MultiLab accounts, although you may use any computer you like to do assignments.
• Textbook — It is essential that you read ahead.

2 Overview of compilers: Chapter 1

• A compiler language is an example of a software tool.
• The compiler’s job.

[Diagram]

• Compiler outputs
  • Pure machine code: specific to a given architecture, no runtime linking. Example: Linux kernel.
  • Augmented machine code: specific to a given architecture and operating system. Example: C programs written for Linux, which may make OS calls.
  • Virtual machine code, interpreted or compiled on the fly during execution. Examples: Java (JVM), C# (.NET). Advantages: portability, code size. **Our assignments use this output type.**

• Output representations
  • Assembler: good for cross-compilation; avoids having the compiler resolve all references. Modular compilation. **Our assignments use this output format.**
  • Relocatable binary: defers resolving external references. Modular compilation. Very common; used by Java and C.
  • Absolute binary: all references resolved.

• The organization of a compiler [Figure 1.4 page 15]

• **Class 2, 8/24/2018**

  • Scanner: reads the source program and constructs a stream of tokens, removing comments, and processing directives such as listing.
• 

Example: `if (a < 39) {` is an input string of characters. The associated output tokens are `if:reserved, (symbol, a:identifier, <:operator, 39:integer, ):symbol, {:symbol`. 

• The scanner can discover and report errors, such as `39f`. 

• We describe tokens by regular expressions. 

• We recognize tokens by using a deterministic finite automaton (DFA). That automaton is built for us by a scanner generator tool such as `lex`, `flex`, or `jflex`. Our assignments use `jflex`. 

• Parser: reads the token stream and creates an abstract syntax tree (AST), verifying syntax and possibly repairing syntax errors. 

• Example: given the tokens above, the tree fragment would be:

```
    Statement
    /-----
   /     /
  if    
   |     |
   |     |
  <     
  |     |
  |     |
 Identifier  Integer 
     |       |
    a      39
```

• The parser can discover and report errors, such as `]` instead of `)` in the example. 

• We describe the syntax by a context-free grammar (CFG). 

• The table that drives the scanner is built for us by a parser generator tool such as `yacc`, `bison`, or `javaCUP`. Our assignments use `javaCUP`. 

• Type checker: navigates through the AST and verifies that variables are declared and that types are used consistently. 

• For instance, if `a` in the example is not of a numeric type, the type checker can report an error. 

• It can also modify the AST, for instance, introducing type-conversion nodes, if, for instance, `a` is a short integer, in which case it might be converted to a regular integer. 

• Translator: navigates through the AST and generates either an intermediate representation (IR) or some other representation.
of executable code. Our IR will be assembler for Java bytecode.

- Optimizer: Analyzes the IR to improve the code. There are many forms of optimization, such as simplifying expressions, moving code, re-using values, eliminating trivial arithmetic, replacing sequences of instructions. We will not cover optimization in this class.
- Code generator: Maps the IR to target machine code. Our assignments use Jasper to generate the target machine code: Java bytecode.

3 Programming language considerations

- Successful designers of programming languages often have strong backgrounds in constructing compilers. If it can’t be compiled, it’s not very useful.
- Many features of modern languages require special care.
  - passing by name (obsolete since Algol 60; requires thunks)
  - dynamic-sized arrays (requires runtime type descriptors)
  - nested name scopes (require static chains)
  - anonymous functions, first-class functions (as in Python, requiring closures)
  - multiple-yield iterators (as in Python, require special stack manipulation)
  - automatic reclamation of object store (requires garbage collection).

4 Computer architecture considerations

- How many registers? What operations use them? How many register classes?
- Some operations can be very expensive: virtual method dispatch, dynamic heap access, reflective programming, exceptions, threads
- The effect of memory architecture, such as paging and caches, is difficult to present to programmers but is significant.
5 Specialty compilers

- Debugging support, including participation in an integrated development environment (IDE).
- Highly optimizing compilers.
- Retargetable compilers.

6 The ac (adding calculator) language: Chapter 2

- Class 3, 8/27/2018
- This is a very simple language that lets us explore the components of a compiler.
- Components
  - Types: integer and float
  - Keywords: f, i, p
  - Variables: lowercase Roman single letters, excluding keywords
- Context-free grammar (CFG), expressed in Backus-Naur Form (BNF) [Figure 2.1 page 33]
- Parse tree for f b i a a = 5 b = a + 3.2 p b $
  [Figure 2.4 page 37]

7 The scanner

- Translates a stream of characters (as above) into a stream of tokens.
- A token has a type (such as operator or reserved) and a semantic value (such as plus or print).
- It’s a matter of choice whether each operator has its own type, in which case there is no need for semantic values.
- Likewise, one can choose that reserved words each have their own type, or that they are of type reserved with a semantic value (their spelling), or that they are of type id with a semantic value.
- Hard-coded example [Figure 2.5 page 40] uses peek() and advance()
• Production-quality scanners are constructed automatically from regular expressions. We will discuss them in the next chapter.

• This parse requires that we specify the syntax of tokens.

8 Formal language hierarchy

<table>
<thead>
<tr>
<th>Language type</th>
<th>Formalism</th>
<th>Automaton</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>Regular expressions</td>
<td>Finite-state automaton (FSA)</td>
</tr>
<tr>
<td>Context-free</td>
<td>CFG (like BNF)</td>
<td>Push-down automaton (PDA)</td>
</tr>
<tr>
<td>Context-sensitive</td>
<td>CSG</td>
<td>Linear-bounded automaton (LBA)</td>
</tr>
<tr>
<td>Type 0</td>
<td>various</td>
<td>Turing machine</td>
</tr>
</tbody>
</table>

9 The parser

• Translates a stream of tokens into an abstract syntax tree (AST)

• Simplest method is recursive descent. Each nonterminal has its own procedure. By looking ahead (using peek()), each procedure can decide which other procedures to call.

• Parsing statements in ac: Figure 2.7 page 42

• One needs to discover the predict sets for each alternative production that has the same left-hand side. For Stmt, the predict set for assignment is {id}.

• One needs to discover the follow sets for some productions that can derive λ in order to compute the predict set for their parent productions.

• Given the grammar in Figure 2.1 page 33, trace the parse of

\[ f \ b \ i \ a \ a = 5 \ b = a + 3.2 \ p \ b \]

10 Abstract syntax trees

• Syntax analysis (scanning and parsing) cannot:
  • enforce strong typing constraints
• disambiguate the meaning of some constructs, like $x.y.z$ in Java, which might be package-class-field or variable-field-field or many other possibilities.
• determine the meaning of an overloaded operator.

• Instead of using the parse tree, we prefer a simpler abstraction of the parse tree: the abstract syntax tree.
  • It omits punctuation.
  • Declarations store the identifier and its type in a single node.
  • It represents the order of executable statements and expressions.
  • Assignment nodes have two children: the identifier (the left-hand side) and the expression (the right-hand side).
  • Binary operations have two children.
  • The print statement is a single node that includes the name of the identifier to be printed.
  • Compare Figure 2.9 on page 44 with Figure 2.4 on page 37.

11 Semantic analysis

• Construct a symbol table for declarations and name scopes. In our case, it can be very simple, see Figure 2.11 on page 48.
• Enforce type consistency.
  • Walk the tree bottom-up, using visitor methods as shown in Figure 2.12 on page 49.
  • Insert to and query the symbol table as necessary.
  • Add a type field to nodes on the fly.
  • Modify the tree to introduce type conversion (in our case, widening) nodes.

12 Generating code

• In our case, the code is calculator buttons.
  • The calculator has registers; each is a single letter, such as $a$.
  • One can load or store a register with the $1$ and $s$ buttons.
• One sets the precision with the $k$ button.
• One prints with the $p$ button.

• We visit the AST with a visitor pattern to generate code, invoking `visit()` at each node. [Figure 2.14 on page 52]