

Undergraduate Compilers Review and Intro to MJC

Announcements

- Mailing list is in full swing

Today

- Some thoughts on grad school
- Finish parsing
- Semantic analysis
- Visitor pattern for abstract syntax trees

Some Thoughts on Grad School

Goal

- learn how to learn a subject in depth
- learn how to organize a project, execute it, and write about it

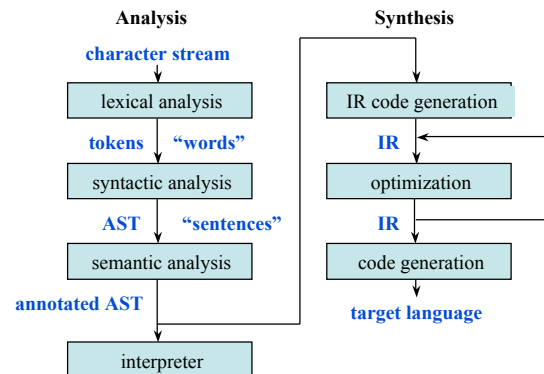
Iterate through the following:

- read the background material
- try some examples
- ask lots of questions
- repeat

You will have too much to do!

- learn to prioritize
- it is not possible to read ALL of the background material
- spend 2+ hours of dedicated time EACH day on each class/project
- what grade you get is not the point
- have fun and learn a ton!

Structure of a Typical Compiler



Lexing and Parsing

Lexing

- theoretical tool: regular expressions
- recognizing substrings instead of strings so need longest match and rule priority
- implementation tools: flex, lex, SableCC, etc. generate code that implements a deterministic finite automata that recognizes the specified tokens

Parsing

- theoretical tool: context free grammars
- recognizing a whole program of tokens
- implementation tools: bison, yacc, SableCC, etc. generate a LALR(1) or bottom-up parser that uses shift-reduce parsing to recognize the program and uses syntax-directed translation to generate an AST

Compiler Data Structures

Symbol Tables

- Compile-time data structure
- Holds names, type information, and *scope* information for variables

Scopes

- A name space
 - e.g.*, In Pascal, each procedure creates a new scope
 - e.g.*, In C, each set of curly braces defines a new scope
- Can create a separate symbol table for each scope

Using Symbol Tables

- For each variable declaration:
 - Check for symbol table entry
 - Add new entry (parsing); add type info (semantic analysis)
- For each variable use:
 - Check symbol table entry (semantic analysis)

CS553 Lecture

Undergraduate Compilers Review

14

Using the Visitor Pattern for semantic analysis

```
public class DepthFirstAdapter extends AnalysisAdapter {
    ...
    public void inAPlusExp(APlusExp node) {
        defaultIn(node);
    }
    public void outAPlusExp(APlusExp node) {
        defaultOut(node);
    }
    public void caseAPlusExp(APlusExp node) {
        inAPlusExp(node);
        if(node.getLExp() != null) {
            node.getLExp().apply(this);
        }
        if(node.getRExp() != null) {
            node.getRExp().apply(this);
        }
        outAPlusExp(node);
    }
}
--

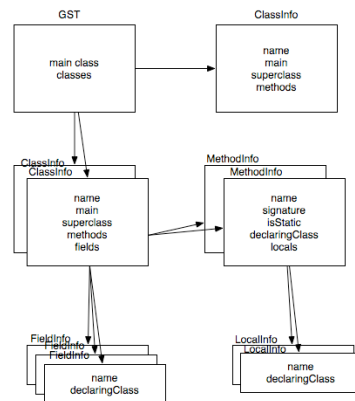
public final class APlusExp extends PExp {
    ...
    public void apply(Switch sw) {
        ((Analysis) sw).caseAPlusExp(this);
    }
    ...
}
```

CS553 Lecture

Undergraduate Compilers Review

15

Symbol Table in the MiniJava Compiler



CS553 Lecture

Undergraduate Compilers Review

16

Concepts

Compilation stages

- Scanning, parsing, semantic analysis, intermediate code generation, optimization, code generation

Parsing

- generating an AST
- shift-reduce parsing

Semantic Analysis

- symbol tables
- using visitors over the AST

CS553 Lecture

Undergraduate Compilers Review

17

Next Time

Reading

- skim Ch 2-6 in Appel
 - focus on 2.1, 2.2, 3.1, 3.3 except parser generation, Ch 4, 5.2, Ch 6
 - skip 3.2 except for FOLLOW description, 3.5, 5.1
- skim Ch 7-9, 12
 - focus on 7.1, 7.3, 8.1, 8.2, 9.3, 12
 - skip 9.2

Lecture

- Finish Undergrad Compilers Review

Parsing Terms (Definitely know these terms)

Lexical Analysis

- longest match and rule priority
- regular expressions
- tokens

CFG (Context-free Grammar)

- production rule
- terminal
- non-terminal
- FOLLOW(X): “the set of terminals that can immediately follow X”

Syntax-directed translation

- inherited attributes
- synthesized attributes

Parsing Terms cont ...

Top-down parsing

- **LL(1)**: left-to-right reading of tokens, leftmost derivation, 1 symbol look-ahead
- **Predictive parser**: an efficient non-backtracking top-down parser that can handle LL(1)
- More generally **recursive descent** parsing may involve backtracking

Bottom-up Parsing

- **LR(1)**: left-to-right reading of tokens, rightmost derivation in reverse, 1 symbol lookahead
- **Shift-reduce parsers**: for example, bison, yacc, and SableCC generated parsers
- Methods for producing an LR parsing table
 - SLR, simple LR
 - Canonical LR, most powerful
 - **LALR(1)**

BNF (Backus-Naur Form) and EBNF (Extended BNF): equivalent to CFGs