COMPILERS

PHASE 3

INTERMEDIATE CODE GENERATOR

Made by:

Mohamed Ramadan– 56

Abdulrahman Atef – 41

Mahmoud Saied – 60

OUTLINE

1. Problem Statement
2. Program Anatomy
3. Used data structures
4. Algorithms and Techniques
5. Functions of all phases
6. Assumptions

PROBLEM STATEMENT

OBJECTIVE

This phase of the assignment aims to practice techniques of constructing semantics rules to generate intermediate code.

DESCRIPTION:

Generated bytecode must follow Standard bytecode instructions defined in Java Virtual

Machine Specification

http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html

http://en.wikipedia.org/wiki/Java_bytecode

Proposed grammars are required to cover the following features:

- Primitive types (**int, float**) with operations on them **(+, - , *, /)**
- Boolean Expressions (Bonus marks)
- Arithmetic Expressions
- Assignment statements
- **If**-**else** statements
- **for** loops (Bonus marks)
- **while** loops

PROGRAM ANATOMY

This phase is divided into several parts which are:

Lexical Analyzer (lex.l): in this phase we tokenize the input and define the meaning of every set of characters in the input code.

Syntax Analyzer (syntax.y) : Rules are written in this file, then semantic actions is written in every rule. Here we write actions to generate Java Bytecode.

C++ code: both files eventually generate c++ code that can be compiled with g++, so in both files you can put c++ code to enhance code generation.

DATA STRUCTURES

%UNION% {SYNTAX.Y}

In Bison, union directive specify a union for every possible type of terminals and non-terminals. For example, for constants of type `int` we put there type in the union as int ival. Then we define a terminal as: "%token <ival> INT_CONST", then we have terminal INT_CONST that has a semantic value of int.

ALGORTHIMS

Conditional expressions: in Boolean expressions we first generate two labels for true and false then we send them to the expression. After that the two labels are printed ahead of the two statements responsible of true and false.

Boolean Expressions: in Boolean expressions we generate the appropriate branch code using the inherited true and false attributes.

FUNCTIONS OF ALL PHASES

PHASE 1

This phase represents the building of a lexical analyzer in a typical compiler. It takes regular expressions for different tokens is a specific language and produces a minimized NFA then this NFA is used in lexical analyzer to match different tokens.

PHASE2

This phase represents the building of a syntax analyzer in a typical compiler. It takes grammatical rules and produces LL(1) parser for this grammar. LL(1) parses the tokens from lexical analyzer and produces parse tree.

PHASE3

This phase represents the semantic analyzer in a typical compiler. It takes the parse tree then produces the intermediate code.

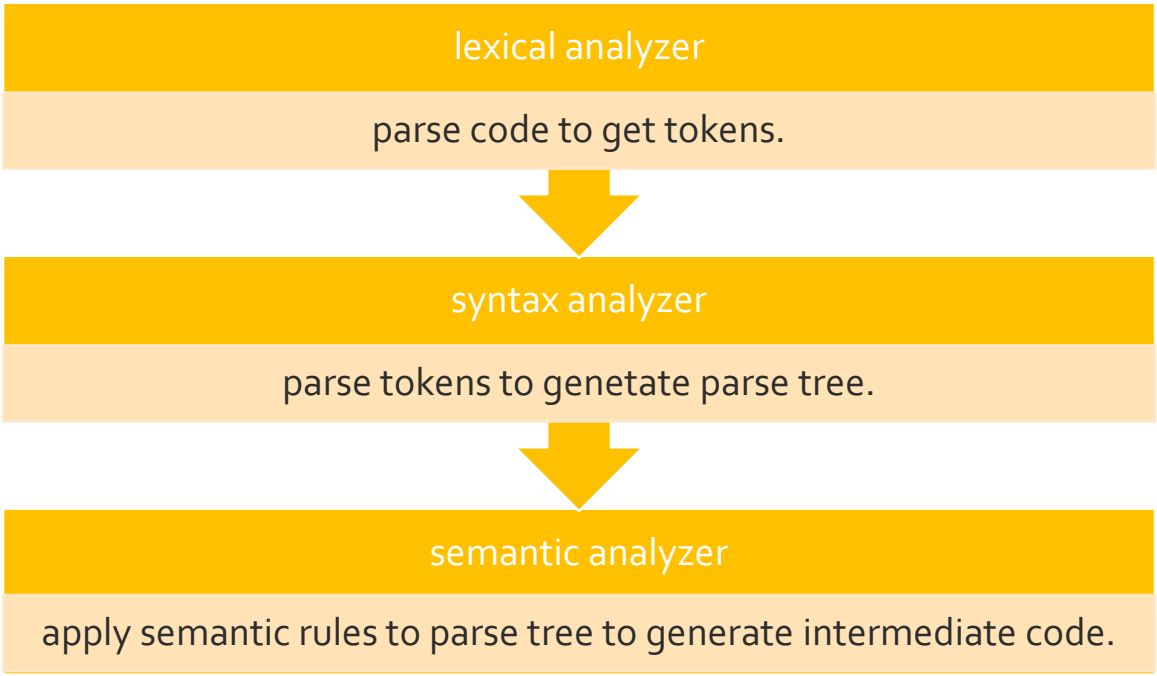
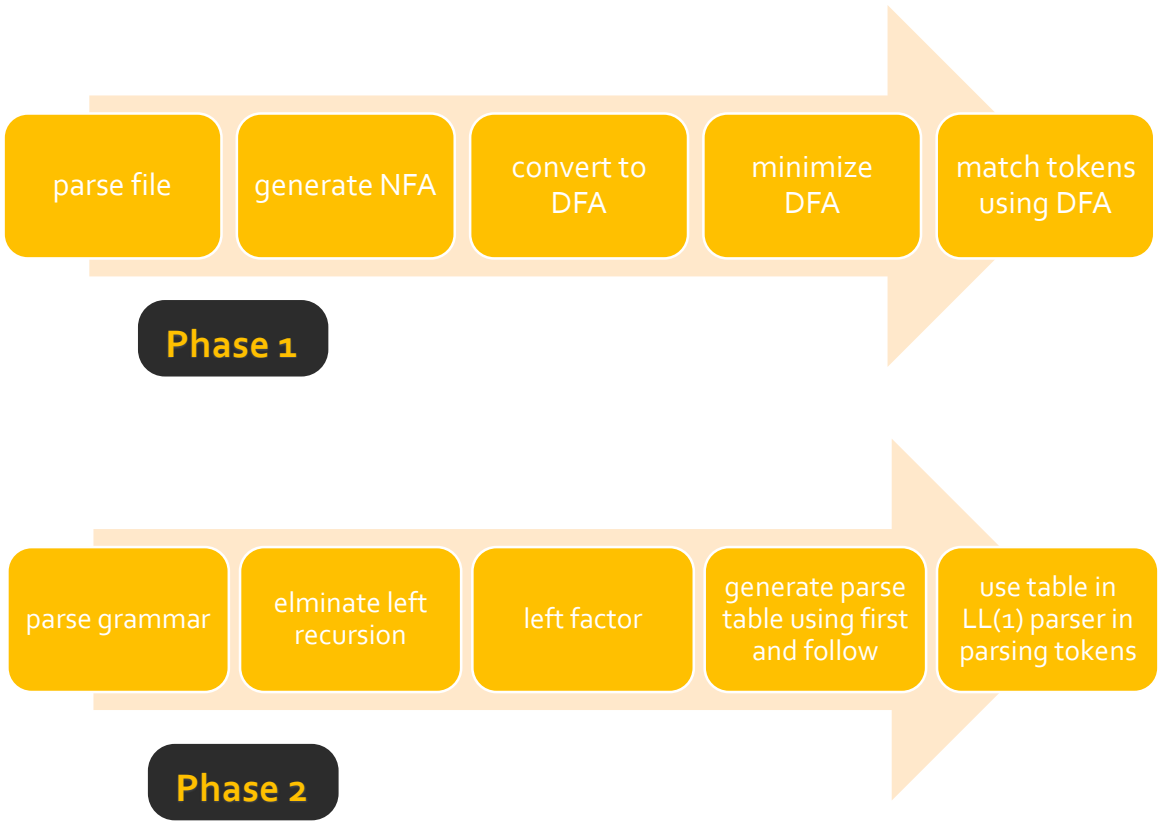


Figure clarifying the function of every phase.


parse file
generate NFA
convert to DFA
minimize DFA
match tokens using DFA
Phase 1
parse grammar
elminate left recursion
left factor
generate parse table using first and follow
use table in LL(1) parser in parsing tokens
Phase 2

ASSUMPTIONS

- assignments of the same types only
- no cond(true && expr) | (true || expr)