CSE 341 – PROGRAMMING LANGUAGES

Homework 2

Name – Surname: Muaz Kurt

Number: 151044062

Q1. Why can machine languages not be used to define statements in operational semantics?

A. The Operational semantics, are definition of a program state’s effects on a machine. If the machine language is using for it, then a human can’t learn language easily or read and understand a parse of a code in that language.

Q2. Write an attribute grammar whose BNF basis is that of Example 3.6 in Section 3.4.5 but whose language rules are as follows: Data types cannot be mixed in expressions, but assignment statements need not have the same types on both sides of the assignment operator.

A.

```
1. Syntax rule: <assign> → <var> = <expr>

 Semantic rule: <var>.actual_type ← <expr>.actual_type

2. Syntax rule: <expr> → <var>[2] + <var>[3]

Semantic rule: <expr>.actual_type ←

        if (look-up(<var>[2] .string) = int) and (look-up(<var>[3].string) = int)  then int

        else

           if(look-up(<var>[2] .string) = real) and (look-up(<var>[3].string) = real)  then real

           end if

        end if

3. Syntax rule:    <expr> → <var>

Semantic rule: <expr>.actual_type ← look-up(<var>.string)

4. Syntax rule:    <var> → A | B | C
```

Q3. What are the reasons why using BNF is advantageous over using an informal syntax description?

A. Because;

it's easier to describe and understand rules from human and a machine with BNF than informal syntax analysis.

BNF based implementations can be easily maintain because of modularity of it. For example: Q2 and A2. If somebody try to maintain a part of the informal syntax description-based implementation, it's too hard because, he/she has to maintain all parts of it.

BNF descriptions can be used as the direct basis of a syntax analyser. Because it's all formal and just cares about what symbol is written and is it legal or not.

Q4. Describe briefly the three approaches to building a lexical analyser.

A. There are tree approaches for building a Lexical Analyzer.

First: Write a formal description of the language using its regular expressions. So that we can easily create a lexical analyser with a common tool such: yacc, lex etc.

Second: Designing a state translation diagram (as a directed graph) that describes the tokens. Each node is a state and all edges are regular expressions that check the next character. This is a final state autamata. If a node has not a matching edge (regular expression) for the next character, then there is an error. This is an automated lexical analyser.

Third: Design a state translation diagram as a table. This table describes the token patterns of the language. Then match the all lexemes with the tokens in the table by hand.