CSE341-HW02

Chapter 3:

Review questions 14: Why can machine language not be used to define statements in operational semantics?

Because the machine language performs one step at a time during the execution the changes in the state of the machine are really small and with a lot of numbers. All the things mentioned above are the reasons why we cannot use machine language to define operational semantics.

Problem set 19: 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.

We have two solutions. We can just change second semantic rule

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

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

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

Semantic rule: <var>[2].env ← <expr>.env

<var>[3].env ← <expr>.env

<expr>.actual_type ← <var>[2].actual_type

Predicate: <var>[2].actual_type = <var>[3].actual_type

3. Syntax rule: <expr> → <var>

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

Predicate: <expr>.actual_type == <expr>.expected_type

4. Syntax rule: <var> → A | B | C

Semantic rule: <var>.actual_typ-e ← look-up(<var>.string)

And second solution is:

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

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

Predicate: <var>[2].actual_type == <var>[3].actual_type

3. Syntax rule: <expr> → <var>

4. Syntax rule: <var> → A | B | C

Semantic rule: <var>.actual_type ← lookup(<var>.string)

Chapter 4:

Review question 1: **What are three reasons why syntax analyzers are based on grammars?**

The syntax analyzers are based on grammars because BNF grammar description is simple and easy to understand. And it can be used asx basis for the syntax analyzer. And if we use BNF as a guide for implementations they can get relatively easier due to their modularity.

Review question 5: **Describe briefly the three approaches to building a lexical analyzer.**

To build a lexical analyzer we need to write a formal description that describes the token patterns. To do so we need to use the descriptive language which is related to the expression.

Than we need to design a state transition diagram. This diagram needs to give all the information needed for the token patterns of the language. Also we need to write a program that would implement the diagram itself.

Finally we need to design another diagram. This diagram should describe the token patterns of the language. And we also need to construct a table-driven implementation of the diagram itself by hand.