Gebze Technical University
Computer Engineering

Programming Languages

CSE 341 - 2018 Autumn

HOMEWORK 2 REPORT

YUNUS ÇEVİK
141044080

Course Teacher: YAKUP GENÇ

Course Assistant: Mahmud Rasih ÇELENLİOĞLU

Question 1) Why can machine languages not be used to define statements in operational semantics? (Chapter 3-14)

The operational semantics is a category of formal programming language semantics in which certain desired properties of a program, such as truth, safety or security, are verified by constructing proofs from logical statements about its execution and procedures, rather than by attaching mathematical meanings to its terms. The machine language can not be used to structures statements in operational semantics because of some problems. First, the individual steps in the execution of machine language and the resulting changes to the state of the machine are too small and many. Because the machine can only understand about “true” or false (‘0’ or ‘1’) but semantics can understand more than machine.

For example, when the machine works a job, the binary numbers as "0 and 1" must be used. But this case rather forces the developer. However, coding a statement with a programming language does not force the developer as semantic (e.g. C programming) and makes easy understanding of developer.

Second, the storage of a real computer is too large and complex. There are usually several levels of memory devices, can connections with other computers and memory devices connection networks. As a result, machine languages and real computers are not used for formal operational semantics.

Question 2) 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. (Chapter 3-19.)

The statement must be assingment to "<expr>.actual_type " as same type of "<expr>.actual_type". However, this rule is not followed in the semantic rule section in figure 3.6. If "(<var> [2] .actual_type = int) and (<var> [3] .actual_type = int)", "<expr> .actual_type ← int" is assign. If not, "<expr> .actual_type ←" is assigned to real. The assignment is considered invalid because of the value on both sides of the assignment is not the same.However, the problem is solved when the expression specified in Table 2 is placed in the area of incompatibility.

Table - 1	**Table - 2**

2. Syntax rule: <expr> → <var>[2] + <var>[3]
Semantic rule: <expr>.actual_type ←
if (<var>[2].actual_type = int) and
(<var>[3].actual_type = int)
then int
else real
end if
Predicate: <expr>.actual_type ==
<expr>.expected_type
	Replace the second semantic rule with:
	<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

Question 3) What are the reasons why using BNF is advantageous over using an informal syntax description?(Chapter 3-1)

Backus-Naur Form (BNF) is a metasyntactic notation procedure used to define the syntax of computer programming languages, command/instruction sets, document formatting and communication protocols. BNF is applied when language descriptions are required. Using BNF rather than an informal system has three advantages. Using BNF, as opposed to using some informal syntax description, has at least three force advantages.

- First, BNF descriptions of the syntax of programs are clear and short, both for humans and for software systems that use them.
- Second, the BNF description can be used as the direct basis for the syntax analyzer.
- Third, implementations based on BNF are relatively easy to maintain because of their modularity.

Question 4) Describe briefly the three approaches to building a lexical analyzer. (Chapter 3-5.)

Lexical analysis is the first level of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code. If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works nearly with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it requests.

1. When a formal description of the token patterns of the language using a descriptive language related to regular expressions write, these descriptions are used as input to a software tool that automatically generates a lexical analyzer. There are many such tools available for this. The oldest of these, is commonly included as part of UNIX systems.
2. Design a state transition diagram that describes the token patterns of the language and write a program that implements the diagram.
3. Design a state transition diagram that describes the token patterns of the language and hand-construct a table-driven implementation of the state diagram.