Scanner Design

Reading Assignment: Chapter 3

The purpose of this case is to drive the design of the scanner for Milestone 2.

The scanner reads in the program text and converts "words" into "tokens". Formally, the scanner can be designed as if it were a finite state machine. The work of this case is to design that machine.

The scanner operates as a subroutine to the parser. The communications between the two is a data structure called *tokens*. The purpose of the token structure is to inform the parser of the *type* of the token and to pass along the *value* of the token. The possible tokens are defined by the **lexical rules** of the language. These are defined in this project in the Naive Semantics.

Assignment 1

Read the Naive Semantics document and determine what the various *types* of tokens are required. Now develop a data structure for these tokens. **For example,** many *C* implementations define a *union;* while object-oriented approaches use an inheritance hierarchy.

Assignment 2

Finite State Automata

The finite state automata (sing. *automaton)* paradigm is the basis for the design of the scanner. The finite state automata paradigm as describe in theory texts has two **registers** (or memory locations) that hold the **current character** and **current state.** The program for the machine can be displayed in several standard ways; I want to emphasize a graphical presentation because it is easier to change during design.

To understand the programs, we must first understand the machine execution cycle. This is actually not much different from the cycle used in a standard computer.

1. The machine is in a state as indicated by the **current state** register. If that state is a valid state, then continue; otherwise goto step 5.
2. Read the next character from the input. If there is no next character, halt.
3. The program of the machine is a set of pairs *(state,character)*. Search the list of valid pairs for a match with the contents of the current input register. If there is no match, set the current state to an invalid value.
4. Go to step 1.
5. The machine has a list of special state names called **final states.** If, upon halting, the current state value is in the final state list, then the input is **accepted;** otherwise, it is **rejected.**

The theory of finite state automata is the theory describing what inputs are acceptable to finite state automata. The way the theory is developed is to consider a formalization by asking what the various parts are; there are

five:

1. The set of possible characters in the input. Programming-wise, this is the type of the input and of current input register.
2. The set of possible state values in the current state register.
3. The full set of pairings of *(state,character)*.
4. The list of final states.
5. The start state.

Part of the difficulty of studying such abstractions is that the formulations can be quite different and yet the actual outcomes are the same. For example, this same set of concepts can be captured by **regular expressions, regular production systems,** and **state transition graphs.** The regular expressions used in editors like *vi* and *grep* are basically implemented by an automata given by the above definition.

Scanner Design

Develop a finite state automaton for each class of lexical input specifications.

Assignment 3

The technical difference between *automaton* and *machine* is that an automaton only gives an accept-reject response while a machine does transformations.

Assignment 2 only develop the yes/no portion of the scanner. The scanner must produce a token as output and therefore, we need to think about the saving of characters, possibly converting some (like integers or special characters in strings), etc. We denote this by the weight.

Modify your output of assignment 2 to reflect the processing to save and convert characters to form tokens.