APPLICATIONS OF FINITE AUTOMATA.

There are a variety of software design problems that are simplified by automatic conversion of regular expression notation to an efficient computer implementation of the corresponding finite automaton.

LEXICAL ANALYZERS.

The tokens of a programming language are almost without exception expressible as regular sets. For eg: ALGOL identifiers, which are upper- or lower-case letter followed by any sequence of letter & digits, with no limit on length, may be expressed as,

$$(\text{letter})(\text{letter}+\text{digit})^*$$

where 'letter' stands for $A+B+C+\cdots Z+a+b+\cdots+z$, and 'digit' stands for $0+1+\cdots+9$. FORTRAN identifiers, with length limit six & letter restricted to upper case & the symbol $, may be expressed as,

$$(\text{letter})(\varepsilon+\text{letter}+\text{digit})^5$$

where 'letter' now stands for $(\$+A+B+\cdots+Z)$. SNOBOL arithmetic constants (which do not permit the exponential notation present in many other languages) may be expressed as,

$$(\varepsilon+-)(\text{digit}^+(\cdot\,\text{digit}^*+\varepsilon)+\cdot\,\text{digit}^+)$$

A no. of lexical analyzer generators take as input a sequence of regular expressions describing the tokens & produce a single finite automaton recognizing any token. Usually, they convert the regular expression to an NFA with ε-transitions & then construct subsets of states to produce a DFA directly, rather than first eliminating ε-transitions.

Each final state indicates the particular token found, so the automaton is really a Moore machine. The transition function of the FA encoded in one of several ways to take less space than the transition table would take if represented as a 2-dimensional array. The lexical analyzer produced by the generator is a fixed program that interprets coded tables, together with the particular table that represents the FA recognizing the tokens (specified to the generator in regular expression notation). This lexical analyzer may then be used as a module in a compiler.

TEXT EDITORS.

Certain text editors & similar programs permit the substitution of a string for any string matching a given regular expression. For eg: the UNIX text editor allows a command such as

s/bbb*/b/

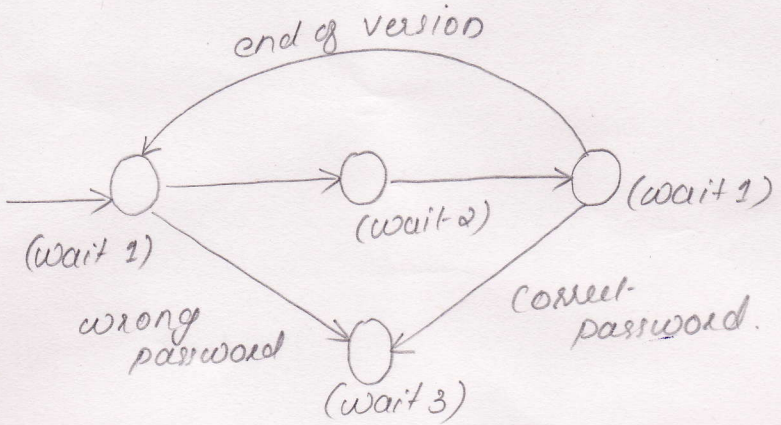
- that substitutes a single blank for the first string of 2 or more blanks found in a given line. Let 'any' denotes the expression $a_1 + a_2 + \cdots + a_n$, where the a_i's are all of a computer characters except the 'newline' character. We could convert a regular expression r to a DFA that accepts any*r. Note that the presence of any* allows us to recognize a member of L(r) beginning anywhere in the line. However a conversion of a regular expression to a DFA takes far more time than it takes to scan a single short line using the DFA, & the DFA could have a no. of states that is an exponential function of the length of the regular expression.

What actually happens in the UNIX text editor is that the regular expression any*r is converted to an

NFA with ε-transitions. However, once a column has been constructed listing all the states the NFA can enter on a particular prefix of the input, the previous column is no longer needed & is thrown away to save space.

LOGIN PROCESS.

Consider the following very simplified version of login process to a computer from the computer point of view. Let us assume for simplicity that this computer accepts a single user at a time. Initially the computer waits for username to be typed in. This is one state of the system. When a name is typed in it checks whether or not the name is valid.



If it is valid, then it asks for & then waits for the password which is another state of the username typed in is not valid. If it goes back to initial state. We could make it to go to a different state & count the no. of login attempts for security purposes. But let us make it simple when a password is typed in & if it is correct then it accepts the user & starts a session. ie, another state though it could further be broken down in to a no. of more states. When the session,

terminates, it gets a signal, goes back to the initial state & waits for another login. If the password typed in it is incorrect, then it informs the user of that & waits for the next try. ie, a 4th state. If the 2nd password fails, it goes to the initial state & starts allows again. Again what we have seen a model for one level of abstraction. Depending on how much detail we are interested in different states would be defined & transitions would have to be selected accordingly.