Nature of a Lexicon in Compiler Design

Rashmi Arora, Tarika Bhutani, Ronak Juneja
Dronacharya College of Engineering

Abstract: As the name suggests, the research paper includes the description of the lexical analysis of a compiler design. This includes the introduction to the compiler and its various phases with their importance. The lexical analysis includes the understanding of automata and its types. The acceptance of any string is discussed here and how we could convert one type of automata into another. And finally minimization of automata is exemplified.

INDEXED TERMS: *Machine language [2], high level programming language [3], automata [5]*

I. INTRODUCTION

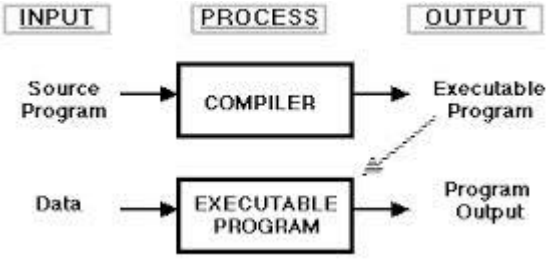
The research paper begins with introduction to the compiler and details about its various phases. The paper is based upon the detailed discussion about lexical analysis, the first phase of the compiler design. Thereafter, it provides details about the automata and its two types. It furthermore includes the inter conversion of two type of automata. Lastly it also provides ways to minimize the automata taking an example.

II. WHAT IS A COMPILER?

In order to reduce the complexity of designing and building computers, nearly all of these are made to execute relatively simple commands (but do so very quickly).[1].Any program for execution is made by combining such multiple commands into a single one which is known as **machine language.**

This job is very time consuming and error prone, thus we need *some high level programming language.* But understanding this high level language is far more difficult than the machine language. Therefore we need some bridging and here comes the role of compilers.

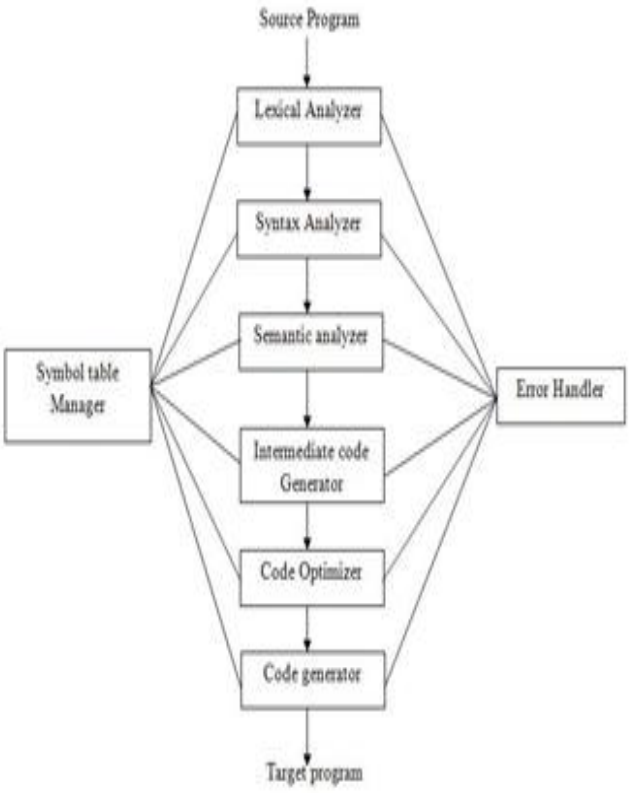
In its most general form, a compiler is a program that accepts as input a program text in a certain language and produces as output a program text in another language, while preserving the meaning of that text.[4]



III. PHASES OF COMPILER

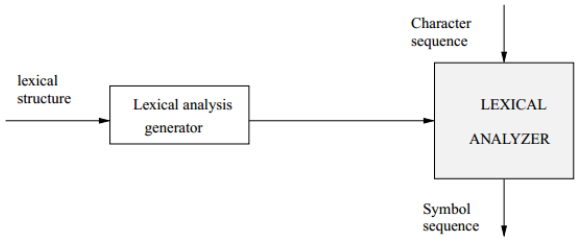
Compilation process is split into several phases with its well defined interfaces. Every phase take input from the output of the previous phase.

These phases operate in a sequence.



a)Lexical Analysis. This is the initial phase. It reads stream of characters and breaks them up into tokens. Each token represents a sequence of characters treated as single entity, i.e., they are the smallest program units that are individually

meaningful.



b)Syntax Analysis. The parser receives the tokens from the lexical analyzer and checks if they arrive in the correct order. It involve grouping the tokens into grammatical phrases that are used by the compiler to synthesize output.

c)Semantic Analysis. Checks the source program for semantic errors and gathers type information for the subsequent code generation phrase. It uses the syntax analysis phase to identify the operators and operands of the expressions and statements.

d)Intermediate code generation. Some compilers generate an explicit intermediate representation of the source program. We can think of this intermediate representation as a program for an abstract machine.

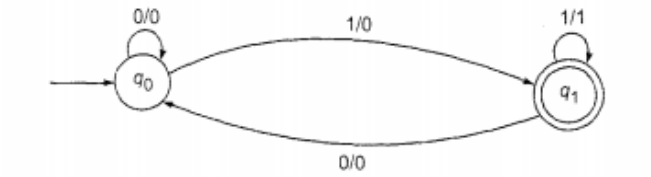
e) Code Optimization. There is a great variation in the amount of code optimization different compilers perform. In those that do the most, called optimizing compilers, a significant fraction of the time of the compiler is spent on this phase.

f) Code Generation. Generation of target code (in general, machine code or assembly code) is done here. Intermediate instructions are each translated into a sequence of machine instructions

IV. LEXICAL ANALYSIS

4.1Finite automata

A finite automata is one that have finite nonempty set of states, inputs and final states. Whereas a *direct transition function* is used to map each input with its respective output.

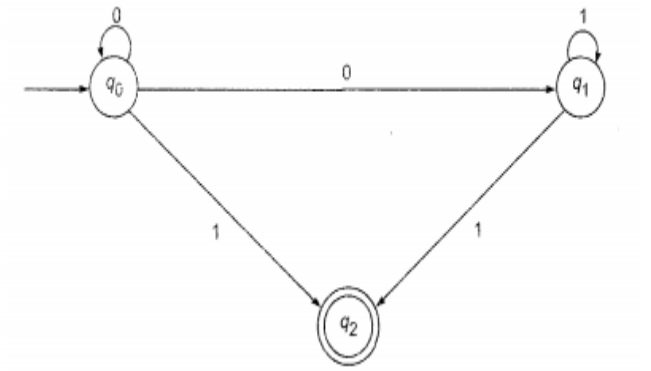


The shown transition system include two states Q={q_0,q_1} , input state is q_0 whereas output state is q_1, the values are {0,1} which brings change in the state using the direct transition function.

4.1.1 Deterministic finite automata (DFA)

A deterministic finite automata is one that accepts a single input on the given state to bring about the transition in the state. i.e., only one input value is defined to obtain output from current state.

4.1.2 Nondeterministic finite automata (NDFA)



Given diagram shows the NDFA. If the automaton is in a state {q_a} and the input symbol is 0, what will be the next state? From the figure it is clear that the next state will be either {q_o} or {q_j}. Thus some moves of the machine cannot be determined uniquely by the input symbol and the present state. Such machines are called nondeterministic automata[6]

4.1.3 Converting a NDFA to DFA

Example: Find deterministic equivalent of M = ({q0,q1,q2} , {a ,b}, δ , {q2}) where δ is given in the table.

State/Σ	a	b
→q_0	q_0, q_1	q_2
q_1	q_0	q_1
(q_2)		q_0, q_1

Solution: Consider that M1 is deterministic equivalent of M. In the given example q0 is the initial state whereas the final state is q2. For the input a q0 gives q0 as well as q1. Similarly q2 gives two states q0 and q1.

The deterministic automaton M_1 equivalent to M is defined as follows:

$$M_1 = (2^Q, \{a, b\}, \delta, [q_0], F')$$

where

$$F = \{[q_2], [q_0, q_2], [q_1, q_2], [q_0, q_1, q_2]\}$$

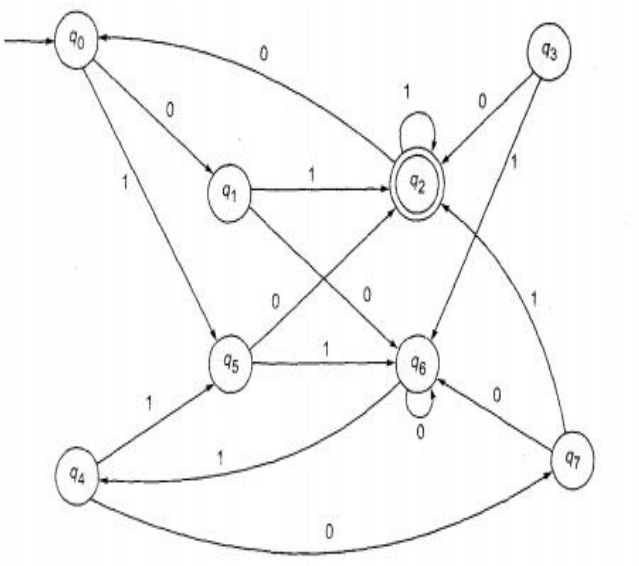
We start the construction by considering $[q_0]$ first. We get $[q_2]$ and $[q_0, q_1]$. Then we construct δ for $[q_2]$ and $[q_0, q_1]$. $[q_1, q_2]$ is a new state appearing under the input columns. After constructing δ for $[q_1, q_2]$, we do not get any new states and so we terminate the construction of δ.

State/Σ	a	b
$[q_0]$	$[q_0, q_1]$	$[q_2]$
$[q_2]$	∅	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_1, q_2]$
$[q_1, q_2]$	$[q_0]$	$[q_0, q_1]$

4.1.4 Minimization of automata

As our interest lies only in strings accepted by automata; what really matters is whether a state is a final state or not. Thus we need to do minimization of the automata

Example:



Solution:

Before we begin we need to construct the transition table of the given transition diagram as follows:

State/Σ	0	1
→q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Here, we will apply step 1 to find out π0 = ({q2}, {q1,q3,q4,q5,q6,q7})

It basically contains two sets of initial and final states separated to construct π0.

Thereafter we will continue making πi until πi = πi+1

Every nest set created will contain splitting of the previous sets such that the states included in the new set must be a part of the previous set constructed whereas previous set will contain all those states that generate the common output states

For example, π1 = ({q2} , {q0,q4,q6} , {q1,q7} , {q3,q5})

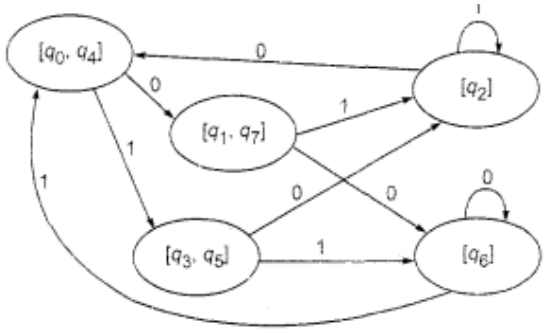
And similarly π2 = ({q2} ,{q0,q4} , {q6} , {q1,q7} ,{q3,q7})

Now further if we try to find out π3, we will find that π3 = π2

Thus we have our minimization steps. And mow we could construct our transition diagram with new states. In this the states that are taken together are considered to be the single state.

State/Σ	0	1
$[q_0, q_4]$	$[q_1, q_7]$	$[q_3, q_5]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$

Thus using this transition table we could construct our transition diagram as follows:



REFERENCES

[1] Basics of compiler design
http://www.diku.dk/~torbenm/Basics/basics_lulu2.pdf

[2] Wikipedia-machine language
http://www.webopedia.com/TERM/M/machine_language.html

[3] Wikipedia-high level programming language
http://en.wikipedia.org/wiki/High-level_programming_language

[4] Modern compiler design by Dick Grune
http://212.1.208.221/fpriolo/fpriolo/Springer.Modern.Compiler.Design.2nd.Edition.Jul.2012.pdf

[5] Wikipedia-automata theory
http://en.wikipedia.org/wiki/Automata_theory.

[6] Theory of computation by K. L. P. Mishra
http://freefundkenotes.files.wordpress.com/2014/02/toc-klp-mishra.pdf