Name : Bazgha Razi

Roll Number : 04

Subject : Compiler Design (Assignments)

College : Delhi Global Institute of Technology (DGIT)

Compiler Construction Tools

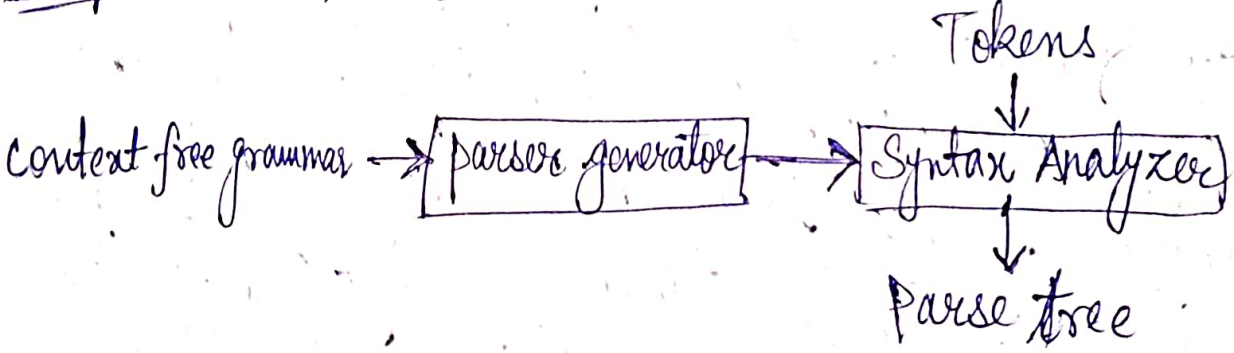
The compiler writer can use some specialized tools that help in implementing various phases of a compiler.

Some commonly used compiler construction tools are:

Parser Generator

It produces syntax analyzers from the input that is based on a grammatical description of programming language or on a context-free grammar. It is useful as the syntax analysis phase is highly complex and consumes more manual and compilation time.

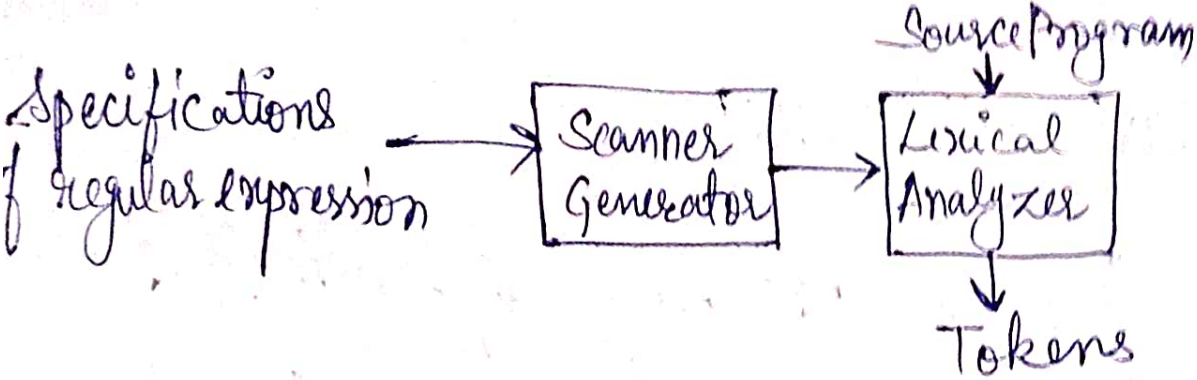
Example: PIC, EQN



Scanner Generator

It generates lexical analyzers from the input that consists of regular expression description based on tokens of a language. It generate a finite automation to recognize the regular expression.

Example: Lex



Syntax directed translation engines

It generates intermediate code with three addre format from the input that consists of a parse tr These engines have routines to traverse the parse and then produces the intermediate code. In this, node of the parse tree is associated with one or mo translations.

* Automatic Code Generators

It generates the machine language for a target machine. Each operation of the in language is translated using a collection rules and then is taken as an input to code generator. A template matching process is used. An intermediate lan statement is replaced by its equivalent machine language statement using temp

Data - Flow analysis engines

It is used in code optimization. Data flow analysis is a key part of the code optimization that gathers the information that is the values that flow from one part of a program to another.

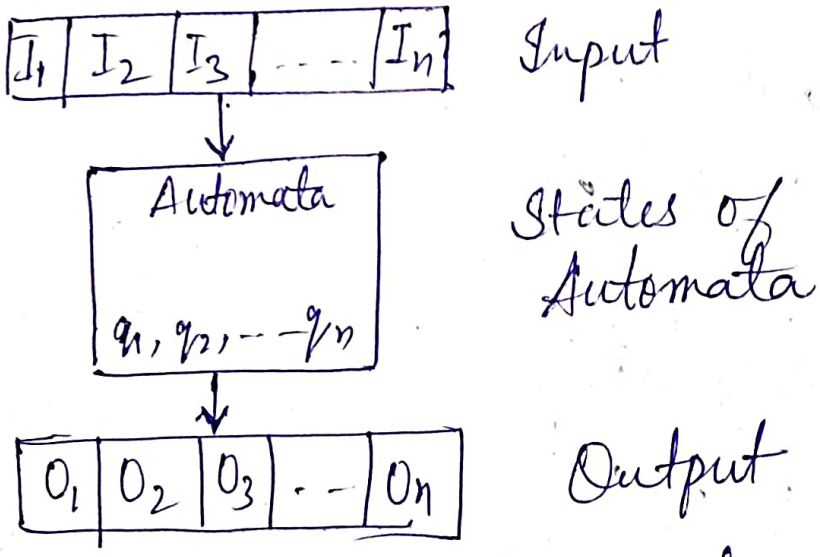
Compiler Construction Toolkits

It provides an integrated set of routines that aids in building compiler components in the construction of various phases of compiler.

Assignment 2

Finite Automata

- is the simplest machine to recognize patterns. finite automata or finite state machine is abstract machine that has five elements tuples. It has a set of states and rules moving from one state to another but it depends upon the applied input symbol. It is abstract model of a digital computer.



The following are the features of automata:-

* Input
* Output
* States of automata
* State relation
* Output relation

A finite automata consists of the

Q : finite set of states

Σ : set of input symbols

q : initial state.

F : set of final states.

δ : transition function

Types of Finite Automata

1) DFA (Deterministic Finite Automata

DFA consists of five tuples

$\{Q, \Sigma, q, F, \delta\}$

Q : set of all states

Σ : set of input symbols

q : initial state

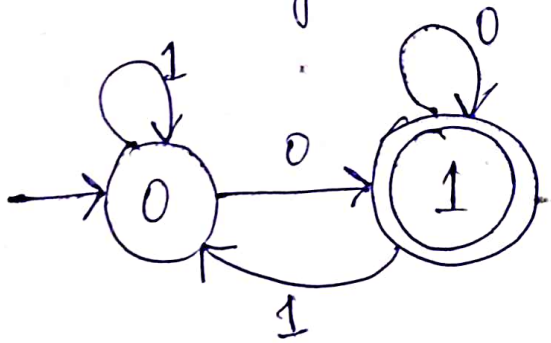
F : final state

δ : transition function

δ is defined as $\delta : Q \times \Sigma \rightarrow Q$

n a DFA, for a particular input character he machine goes to one state only. A transition unction is defined on every state for every nput symbol.
n DFA null move is not allowed. DFA cannot hange state without any input character.

Example: DFA with $\Sigma = \{0, 1\}$ accepts all strings ending with 0.



There can be many possible DFAs for a pattern.
The DFA with minimum number of states is generally preferred.

) NFA (Non Deterministic Finite Automata)

NFA is similar to DFA except following additional features:

- Null moves
forward without
- Ability to transmit to any number
for a particular input.

DFA and NFA both are equivalent.

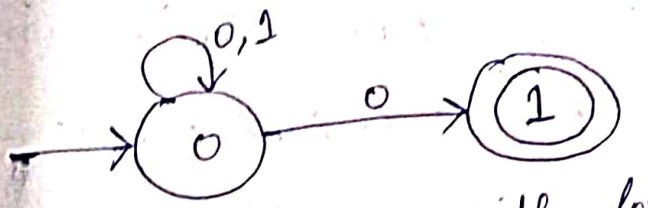
NFA also consist of five tuples
{ Q, Σ, q, F, δ }

Q : set of all states
Σ : set of input symbols
q : initial state
F : final state
δ : transition function

It is defined as $\delta : Q \times (\Sigma \cup \epsilon) \rightarrow$

NFA can go to any state of number

Example: NFA with Σ = {0, 1} accepts all strings ending with 0.



In NFA, if any path for an input string leads to a final state, then the input string is accepted.

For example, in the above NFA, there are multiple paths for the input string "00". Since one of the paths leads to a final state "00" is accepted by the above NFA

Both NFA and DFA have the same power and each NFA can be translated into a DFA.

There can be multiple final states in both DFA and NFA.

NFA is more of theoretical concept.

DFA is used in lexical analysis in compiler.

Conversion of Regular exp... to DFA

Step 1: Construct a transition diagram for given RE by using NFA with ∈ m...

Step 2: Convert NFA with ∈ to NFA without ∈

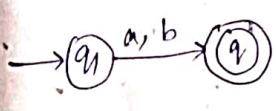
Step 3: Convert the NFA to the equivalent D...

Some basic RE are as follows:

Case 1: For a regular expression 'a', we can construct FA as shown below.

Case 2: For a regular exp^n ab we can construct FA, as follows.

Case 3: For a regular expression (a+b) we can construct FA as follows



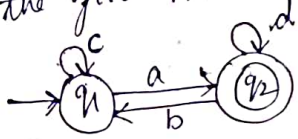
RE = (a+b)

Case 4: For a regular expression (a+b)*, we can construct FA as follows

RE = (a+b)*

Conversion of finite automata to RE

Convert the given FA into regular expression



There are two methods for converting DFA to its regular expression:

Arden's Method

State elimination method

State elimination method to convert FA to RE.

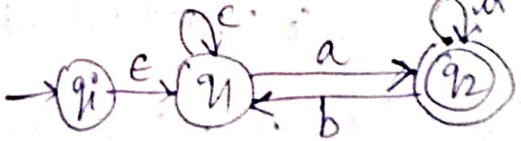
Rule 1: The initial state of DFA must not have any incoming edge. If there is any incoming edge to the initial edge, then create a new initial state having no incoming edge to it.

Rule 2: There must exist only one final state in DFA. If there exist multiple final states, then convert all the final states into non-final states and create a new single final state.

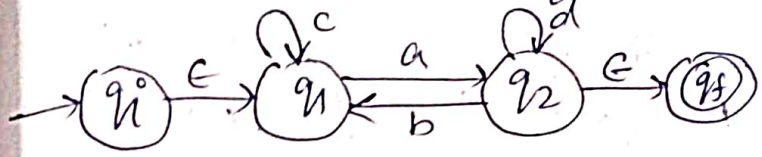
Rule 3: The final state of DFA must not have any outgoing edge. If this exists, then create a new final state having no outgoing edge from it.

Rule 4: Eliminate all intermediate states one by one. Now apply these rules to convert the FA to regular expression.

Step 1: Initial state q_1 has an incoming edge so create a new initial state q_i.

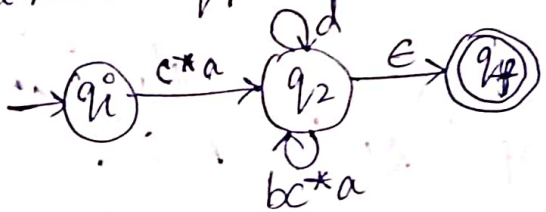


Step 2: Final state q_2 has an outgoing edge. So, create a new final state q_f



Step 3: Start eliminating intermediate states:

Eliminate q_1 state:



Now, eliminate q_2 state:

$c^*a\,(d+bc^*a)^*\,\epsilon$

$c^*a\,(d+bc^*a)^*$

Final regular expression of the finite automata is:

$RE = c^*a(d+bc^*a)^*$

Minimizing no. of states in DFA

Step 1: Eliminate all the dead states and inaccessible states from the given DFA (if any).

Step 2: Draw a state transition table for the given DFA.
Transition table shows the transition of all states on all input symbols in Σ.

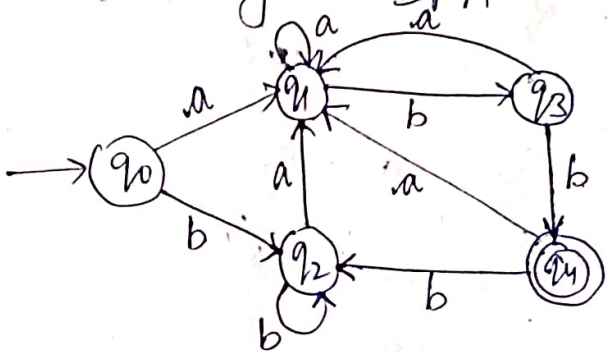
Step 3: Now, start applying equivalence theorem
- Take a counter variable k and initialize it with value 0
- Divide Q into two sets such that one set contains all the non-final states and other set contains all the final states.
- This partition is called P_0

Step 4:
- Increment k by 1
- Find P_k by partitioning the different sets of P_{k-1}
- In each set of P_{k-1}, consider all the possible pair of states within each set and if the two states are distinguishable, partition the set into different sets in P_k

Step 5:
- Repeat step 4 until no change in partition occurs.
- when you find $P_k = P_{k-1}$, then stop.

Step 6: All those states which belongs to the same set are equivalent.
The equivalent states are merged to form single state in the minimal DFA.

Minimize the given DFA



Step 1: The above DFA contains no dead states and inaccessible states.

Step 2: Draw a state transition table

	a	b
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
(q_4)	q_1	q_2

Step 3: Now using equivalence theorem, we ha

$P_0 = \{q_0, q_1, q_2, q_3\}\{q_4\}$

$P_1 = \{q_0, q_1 q_2\}\{q_3\}\{q_4\}$

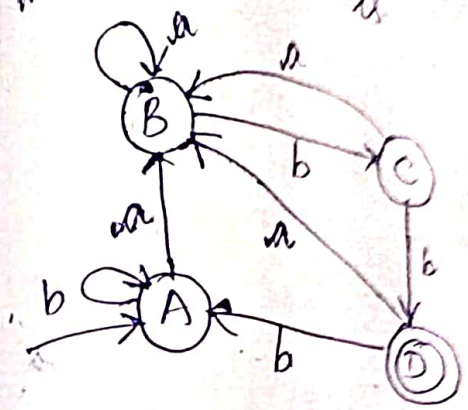
$P_2 = \{q_0, q_2\}\{q_1\}\{q_3\}\{q_4\}$

$P_3 = \{q_0, q_2\}\{q_1\}\{q_3\}\{q_4\}$

Since $P_3 = P_2$, so we stop

From P_3, we infer that states q_0 and q_2 are equivalent and can be merged together.

minimal DF



minimal DFA