Various Attributes in the Theory of Computation

Kiran Qazi
Department of Computer Science & IT
NUML
Islamabad, Pakistan

ABSTACT: *The computation in modern research is composed of many issues. The theoretical issues include Knowing about the computer and way of computing, Foundation of all modern computers, Pure Science and Philosophical Implications. The practical issues include searching in the web, methods in the matching pattern and the knowledge about the finite state automata. The other significant issues include the devices with finite numbers of state and parsing numbers of different strings, knowledge about context free grammar, recursive rules for generating numbers of words or strings. There are a few more crucial issues such as Cryptography, algebraic complexity and robotics logics. In complex computation the different techniques are used to compress data file, so less numbers of data is to be travel on the network. This paper addresses in a more generic way the issues outlined for modern computation.*

Keywords: IT Attributes, IT Theory, IT Practice

Received: 18 January 2014, Revised 27 February 2014, Accepted 4 March 2014

1. Introduction

Theory of computation relies on processing of a machine or a device.

It not only means how to compute but it means to get a way for a solution of a problem in such a way that there should be use minimum memory and with effective way so that less number of statements can solve a problem.

Its history stated by 1930 when a mathematician found is computer is has any limitation? He also determines the working models and way of computer working in such a way to solve only silicon base computing. A computer is not capable of doing any molecular base computations.

The answer is not simple as we think; we have to concern with all three basic aspect of theory of computation given below.

1. Complexity Theory

2. Computability Theory

3. Automata Theory

Complexity Theory:
Classify the problem according to their complexity or difficulty is called complexity theory.

It means classify or sort the problem on this basis that how much this problem is complex.

Sorting of an array is more complex than searching in that array.

So the making degree of complexity of problem is called complexity theory.

Computability Theory:
Classify problem according to their solvability or insolvability is called computability theory.

History:-

This topic "*Computability theory*" is started in 1930 where three persons named Gödel, Turing and Church discover that some of mathematical fundamental problems are not solvable by computer

Example:

Can arbitrary mathematical statement are solvable or not? Means an individual judgment without any restriction and our computer programming covering all aspect of possible outputs.

For this we have to use many of things

1. Formal definition of the notion of computer.

2. Algorithm and their computation.

Automata:-
A self operating machine which deals as an abstract model for a computer which has limited memory.

This means has concern with properties and definition of different computer model.

The basic computational models are as below.

• Finite Automata.

• Context Free grammar

• Turing Machine

Finite Automata:-
It is very basic component in theory of computation.

In theory of computation, it is used in text processing, compiler and in many electronically devices.

Context free Grammar
This component is used in artificial intelligence and used in defining the programming language.

A context free grammar (CFG) is a simple recursive method of specifying grammar rules by which strings in a language can be generated.

In a simple and descriptive a context free grammar is able to parse the syntax of any high level language and make human to be able to communicate with a machine like computer in his language but with a systematic method.

We can describe any language regular or non regular by using recursive

Example

Palindrome Language.

Alphabet $\Sigma = \{a, b\}$

Rule 1: Null, a, b, ε Palindrome

Rule 2: For any $S \varepsilon$ Palindrome

Then *aSa*, *bSb* are in palindrome.

Note *S* can be considered as a variable.

By getting any element within palindrome series. If one element is parse successfully, so it should parse all Palindrome strings or numbers.

Rule can be considered as a relation,

The resulting rules which are to be applied is shown by

"$\longrightarrow$".

Rule 1: $S \rightarrow$ Null | *a* | *b*

Rule 2: S $\rightarrow$ *aSa* | *bSb*

Null, *a* and *b* are terminals.

S is a non terminal or variable.

Note: The vertical bar "|" means "*or*".

The production of the grammar

S Start Variable

Others represent some auxiliary

Class of strings (class of recursive).

Alphabet $\Sigma = \{a, b\}$

L is defined by the following rules:

$s \rightarrow aSa \mid bSb \mid V$

$V \rightarrow aTb \mid bTa$

$T \rightarrow aT \mid bT \mid$ Null

These are Non Palindrome

Turing Machine

Challenges: Design simplest machine that is "*as powerful*" as conventional computers"

Turing Machine Components

Tape.

• In tape the input is stored to be read in future and output is also be stored for further processing. But an intermediate code is upon the choice of designer or user that he wants to save its intermediate or it should be leave and no future needs will be arise.

• We divide the strip into small logical groups. A long string is to be divided its appropriate numbers of groups.

• Finite alphabet of symbols.

Tape Head.

• Point to one cell of a Tape.

• Reads a symbol from active cell

• Writes a symbol to active cell.

• It will transform each logical group or cell simultaneously.

Importance

• It captures the essence of computational process

• Its computational power is as great any algorithmic system any algorithmic system

• Turing machine is one of the most authentic machines in itself, means if any matter is to be solved and once Turing machine accepts and parse the problem so it will be accepted by other parsing techniques.

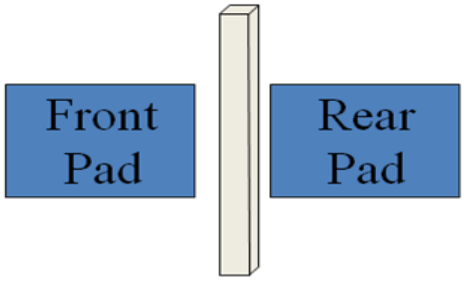
• Represent the theoretical bound on the capability of actual machines.

Making abstract model of a real computer is possible with Turing machine.

For using the Automata component of theory of component in real

Example is 1 way door.

1 Way Door



• Think as a one way automatic door.

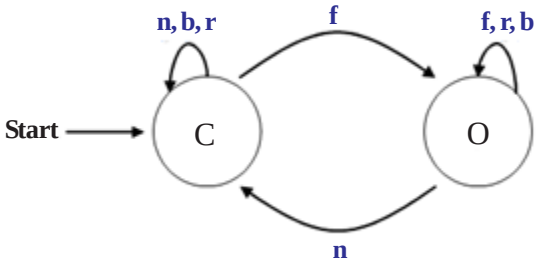
• There are two pads that can determine whether there is on it or not?

• Peoples are allowed to walk through front and toward the rear. They are not allowed to walk the other direction.

For assigning the codes to different possible inputs.

n – Nobody on either pad.
f – Person on front pad.
r – Person on read pad.
B – Person on front and rear pad.

We can design the automaton so door can be open if only direction is from front to rear.



	n	f	r	b
→ **Close**	C	O	C	C
Open	C	O	O	O

In Formal Definition of Finite Automata

• Formally, we can define **Deterministic Finite Automata** as the quintuple

$$M = (Q, \Sigma, \delta, q_0, F), \textbf{ where}$$

• ***Q***: is a finite set of internal states.

• **Σ**: is a finite set of symbols, called input alphabet.

• $\boldsymbol{\delta}$: Q X Σ → Q is a total function called the transition function.

• $q_0 \in Q$ is the initial state.

• $F \subseteq Q$ is a set of final states.

Use Formal Notation

$Q = \{C, O\}$ (usually we'll use q_0 and q_1 instead)
$F = \{\}$ There is no final state
$q_0 = C$ There is the start state
$\Sigma = \{n, f, r, b\}$

• The transition function, δ, can be specified by the table:

	a	b	c	d
→C	C	O	C	O
O	C	O	C	O

• The start state is indicated with the →

If there are final accepting states, that is indicated with *a** in the proper row

As a summary of Turing Machine Some of its aspect are given below.

Goal: Simplest machine that is "*as powerful*" as conventional computers".

Surprising Fact 1. Such machines are very simple: TM is enough!

Surprising Fact 2. Some problems cannot be solved by ANY Computers.

Consequences

• Precursor to general purpose programmable machines.

• Exposes fundamental limitations of all computers.

• Enables us to study the physics and universality of computation.

• No need to seek more powerful machines!

Variation

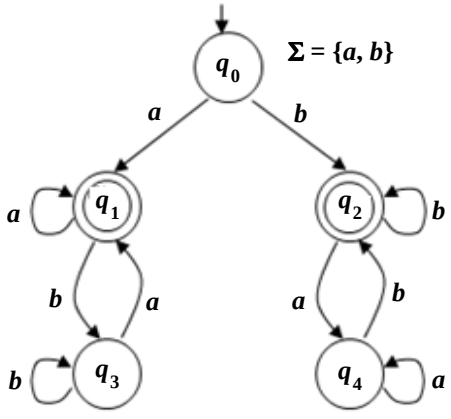
• Instead of just recognizing strings, TM's can produce output: the contents of the tape.

• Instead of *Y* and *N* States TM's can have a plain Halt state?

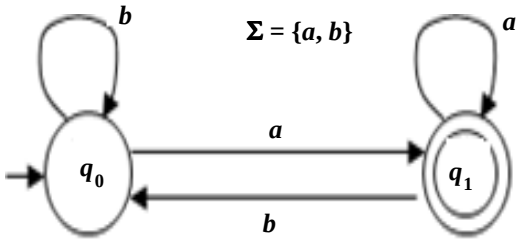
Regular Language

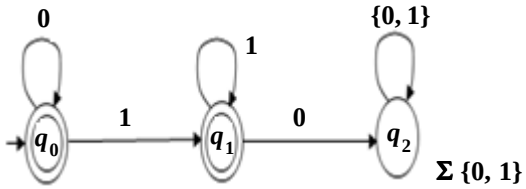
A language is called regular language if and only if there exist a finite automation *M* such that $L = L(M)$.

Assume that $M = (Q, \delta, q, F)$ be a finite automaton. It Means that *M* has all valid finite strings that will accepted by $L(M)$.



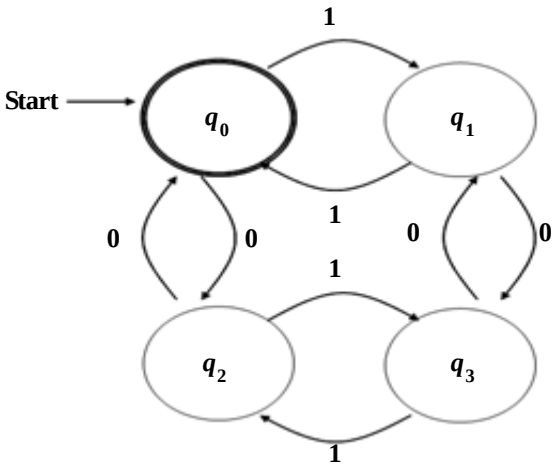
Examples:





DFA example

Below is the DFA for strings of a language that is set 0's and 1's. Which has both 0's and 1's in even state.



In this example, there are four states q_0, q_1, q_2 and q_4. It starts from q_0. A string is to read 1010. When first character is to read 1, we will reach at state q_1 and the next character 0. After reading we will reach at state q_3. For reading next character 1, we will reach at state q_2 and after reading last character we will again reach at state q_0.

The Non-Deterministic Finite Automata (NFA)

A method related or equivalent to transition diagrams are nondeterministic finite automata
NFA is a Combination of

- It consists of a variable that can only take a finite or limited number of many different states.
- It consists of a read head which will read from left to right is also termed with input tape.
- It consists of a transition relation Δ that makes the control of the automation.
- It consists of an initial state.
- It consists of one or several final states.

We will define an example of NFA that will recognize the integer values and also real constants.

An integer values are those values which are numeric values but having no fractional part.

Real constant are also to term as a floating value means it must contain decimal point or an exponent that will distinguish or differentiate it from the integer type.

Examples of valid real constants are as below.

$$.5,\ -100.5, 1000.4,\ 123.45E4$$

NFA that will recognizes both integer and real constants values

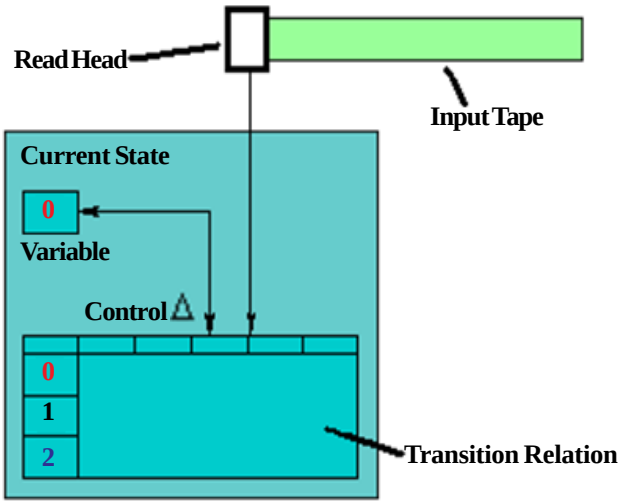
Alphabet = {0, 1... 9, *E*}

Set of States $Q = \{z0, z1 \ldots z7\}$

Initial State $q_0 = z0$

Final State $F = \{z1, z7\}$

	{ 0, 1, ..., 9}	.	E	e
z0	{z1, z2}	ø	ø	ø
z1	{z1}	ø	ø	ø
z2	{z2}	{z3}	ø	ø
z3	{z4}	ø	ø	ø
z4	{z4}	ø	{z5}	{z7}
z5	{z6}	ø	ø	ø
z6	{z7}	ø	ø	ø
z7	ø	ø	ø	ø



Transition Relation

$$\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$$

We will use the Input Value with 123.45E12

(z0, 1, z1)
(z1, 2, z1)
(z1, 3, z1)
no transition

(z0, 1, z2)
(z2, 2, z2)
(z2, 3, z2)
(z2, ., z3)
(z3, 4, z4)
(z4, 5, z4)
(z4, E, z5)
(z5, 1, z6)
(z6, 2, z7)

(z4, e, z7)
no transition

(z4, e, z7)
no transition

The nondeterministic finite automata will start with the state *z*0 and will read the first character '1' from the tape of input. In the next state *z*1 it will only read the subsequent digits; for the point ".", there is no transition in the table. In the state *z*2, it can also read the subsequent digits, and for the character ".", there exists a transition to the state *z*3. So under the character ".", it enters the state *z*3 and so on. In the end, it enters the state *z*7, which is a final state, and accepts the input as a real constant. We than note that the NFA and the transition diagram have the same behavior.

There are three major advantages of Non Deterministic Finite Automata over Deterministic Finite Automata

1. In NFA the drange is the power set 2^0, it means, it defined the set of all possible states that can describes each transition state.

2. Its second advantage is to allow λ as another argument. It means NFA can make transition without using any input symbol.

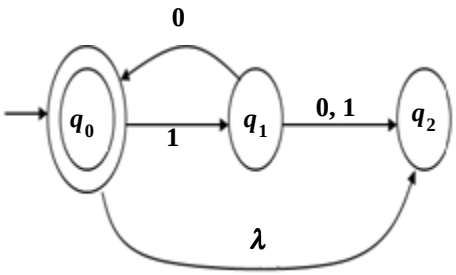
3. δ Can also have empty value it means there can be no transition defined for any specific action.

NFA and DFA

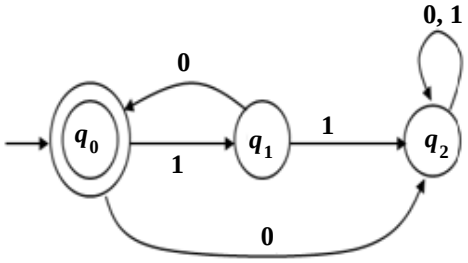
Following are the example of NFA and DFA defines the languages

$$L = \{(10)^{n:}\ ne \geq 0\}$$

NFA



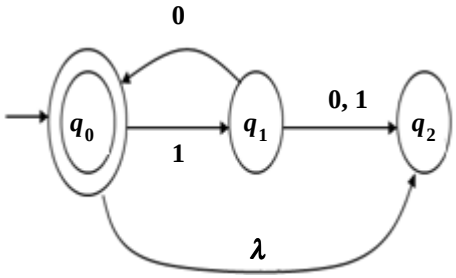
DFA



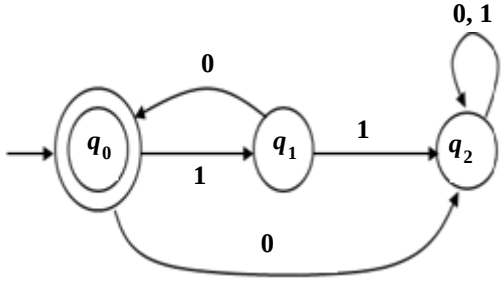
As $L(M_1) = L(M_2) = \{10\}^*$

It means M1 is equal to M2 and set {10}* means all the possible strings in this set.

NFA M1



DFA M2



NFA VS DFA

NFA and DFA can recognize the same set of regular languages.

1. DFA is easier to use or implement while NFA is Difficult to implement.

2. In Simplicity the NFA is much simpler than DFA

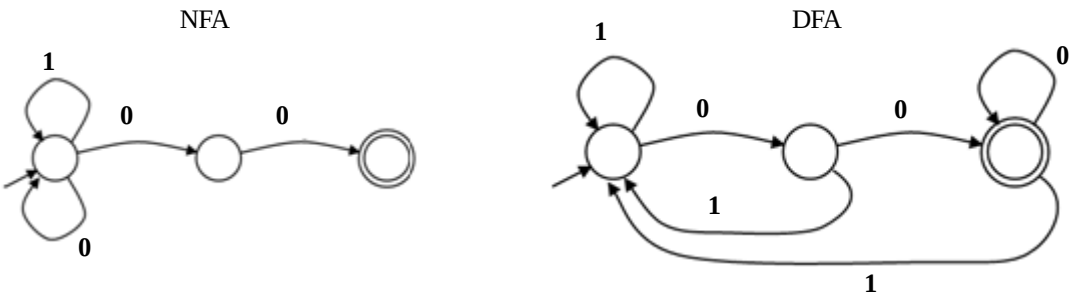
Because NFA do not require edge of each node for every letter of alphabet.

Its best use in game programming. Where the best move is unknown so the backtracking is almost necessary.

For Example

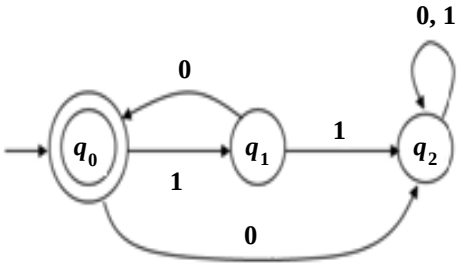
The NFA can be simple than the DFA

In this example the DFA is the exponentially larger than the NFA. The NFA is much simpler in this example where only 4 moves can explain the whole language or each possible string.

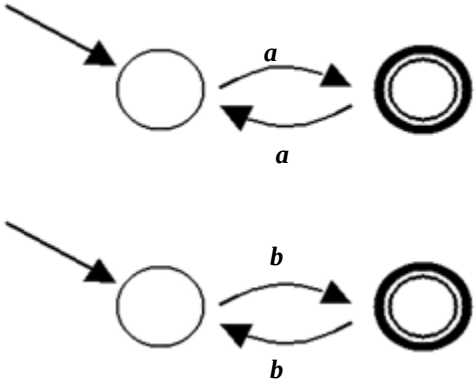


Importance of Regular Expression

It is the algebraic form of the regular language.

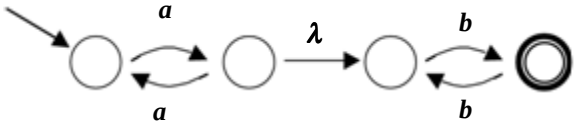


- Each regular language can be described as regular expression.
- It is used to gather or run different finite automaton into single finite automaton.
- It is used to define and express the language of specific automaton.



$A = \{a^n \mid n \text{ is odd}\}$

$B = \{b^m \mid m \text{ is odd}\}$



$\{xy \mid x \in A \text{ and } y \in B\}$

Regular expression: $(a + b) - a^*$

$L((a + b) . a^*) = L((a + b))\, L(a^*)$

$= L\,(a + b)\,L\,(a^*)$
$= L\,(a) \cup L\,(b))\,(L\,(a))^*$
$= (\{a\} \cup \{b\})\,(\{a\})^*$
$= \{a, b\}\{\lambda, a, aa, aaa, ...\}$
$= \{a, aa, aaa, \ldots.., b, ba, baa, \ldots\}$

Here in this example two separate regular expressions will generate the same Language of finite strings

$$L\,((a + b)\,.\,a^*) = L\,((a + b))\,L\,(a^*)$$

Where after solving of this part

$$L\,((a + b))\,L\,(a^*)$$

We will get $L\,(a + b)\,L\,(a^*)$ because the inner parenthesis have no operator for changing the operand.

As $(a + b) = (a) \cup L\,(b))$ So putting this value in equation.

$$L\,(a) \cup L\,(b))\,(L\,(a))^*$$

By getting L common in this equation. Result will be getting

$$(\{a\} \cup \{b\})\,(\{a\})^*$$

As $\{a\} \cup \{b\} = \{a, b\}$

And

$$\{a\}^* = \{\lambda, a, aa, aaa, ...\}$$

So putting this in equation will get the possible finite strings.

$$= \{a, aa, aaa, \ldots.., b, ba, baa, \ldots\}$$

Limitations of DFA

No DFA can recognize the language of all bit strings with an equal number of 0's and 1's

• Suppose on *N*-state DFA can recognize this language.

• Consider following input 0000000011111111

$$N + 1 \quad 0\text{'s} \quad N + 1 \quad 1\text{'s}$$

• DFA must accept this string.

• Same state *x* is revisited during first $N + 1$ 0's since only *N* states.

0000000011111111

X X

• Machine would accept same string without intervening 0's.

000011111111

• This string doesn't have an equal number of 0's and 1's

Fundamental Questions

Which language cannot be described by any RE?

• Bit strings with equal number of 0s and 1s.

• Decimal strings that represent prime numbers.

• Genomic strings that are Watson-Crick complemented palindromes.

• Many more . . . .

How can we extend RE's to describe richer sets of strings?

• Context free grammar ((e.g., Java)).

In a very short summary a regular expression can be described about many aspects. Some of them are given below.

Programmer

• Regular expressions are a powerful pattern matching tool.

• Implement regular expressions with finite state machines.

Theoretician

• Regular expression is a compact description of a set of strings.

• DFA is an abstract machine that solves pattern match problem for regular expressions.

• DFAs and regular expression have limitations.

Variations

• Yes (accept) and No (reject) states sometimes drawn differently

• Terminology: Deterministic Finite state automaton (DFA), Finite State Machine (FSM), Finite state Automaton (FSA) is the same.

• DFA's can have output, specified on the arcs or in the states

These may not have explicit Yes and No states.

Formal definition of computation

These are the some aspect of formal definition of computation.

• Model of computation base on the working performance of a finite automation.

• Finite automata were described informally, using state diagrams, and formally, as a 5-tuple.

• Informal description is easier to understand, but the formal definition is necessary for making notion precise, that will help to resolve ambiguity that may occur in formal description.

2. Conclusion

In a conclusion theory of computation is used to specify a generic way to solve problems with varieties of tools and methodologies. It reveals all aspects of computational problems and their appropriate possible authentic solutions. Some Time its answer may come to you at a point where a little bit confusion will arise, i.e In case of NFA and DFA sometimes its string parsing may can create a little ambiguity but its solution is to authenticate problem solution by two or more different tools like Turing Machine. This study is more helpful to understand working of compiler its inner working that a string is to be parse and make syntax authentication. By using different techniques getting one step further in natural processing and make a new big bang in silicon computational word.

Reference

[1] An Introduction to the theory of computation, by Michael Sipser, PWS Publishing Company, Inc. 1997; ISBN: 053494728X.

[2] Theory of Computation an Introduction, by James L. Hein, Jones & Bartlett Publishers 1996; ISBN: 0-86720-497-4.

[3] http://www.cs.Princeton.Edu/~cos126.

[4] http://en.wikipedia.org/wiki/Theory_of_computation.