SYLLABUS

Coregroup:

Finite Automata, Pushdown Automata. Non-determinism and NFA, DPDA, and PDAs and languges accepted by these structures. Grammars, Languages-types of grammers-type 0, type 1, type 2 and type 3. The relationship between types of grammars, and finite machines. Pushdown automata and context free grammars. Conversion of NFA to DFA. Minimzing the number of states in a DFA.
Context free grammars. Parsing and parse trees. Representation of parse (derivation) trees as rightmot and leftmost derivations.

Elective:

Theory of Computation: Formal language, Need for formal computational models, Non-computational problems, diagonal argument and Russel's paradox.

Deterministic Finite Automaton (DFA), Non-deterministric Finite Automaton (NFA). Regular languages and regular sets, Equivalence of DFA and NFA. Minimizing the number of states of a DFA. Non-regular languages, and Pumping lemma.

Pushdown Automaton (PDA), Deterministic Pushdown Automaton (DPDA), Non-equivalence of PDA and DPDA.

Context-free Grammars: Greibach Normal Form (GNF) and Chomsky Normal form (CNF), Ambiguity, Parse Tree Representation of Derivations. Equivalence of PDA's and CFG's. (Parsing techniques for parsing of general CFG's – Early's, Cook-Kassami-Younger (CKY), and Tomita's parsing.) * Compiler.

Linear Bounded Automata(LBA): Power of LBA, Closure properties.

Turing Machine (TM): One tape, multitape. The notions of time and space complexity in terms of TM. Construction of TM for simple problems. Computational complexity.

Chomsky Hierarchy of languages: Recursive and recursively-enumerable languages.

$$\begin{tikzcd}
 & \text{Theory of Computation} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Automata, Language Grammar} & \text{Computability} & \text{Complexity}
\end{tikzcd}$$

RUSSELL'S PARADOX:

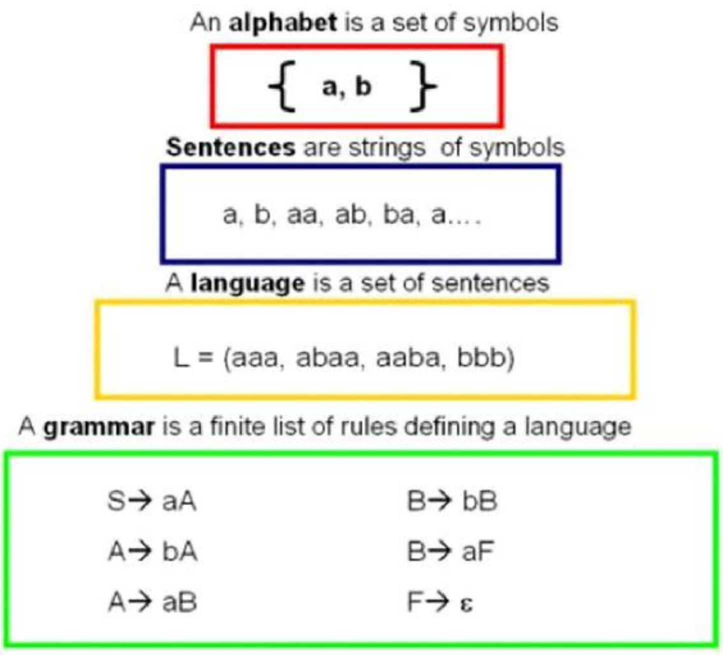
According to naive set theory, any definable collection is a set. Let R be the set of all sets that are not members of themselves. If R qualifies as a member of itself, it would contradict its own definition as a set containing sets that are not members of themselves. On the other hand, if such a set is not a member of itself, it would qualify as a member of itself by the same definition. This contradiction is Russell's paradox. Symbolically:

$$R = \{x \mid x \notin x\}, \text{ then } \mathrm{R} \in \mathrm{R} \Leftrightarrow \mathrm{R} \notin \mathrm{R}$$

Noncomputability is discussed in computability section. Where we also discussed recursive and recursive enumerable classes.

INTRODUCTION:

A **formal language** is a set of words, i.e. finite strings of letters, symbols, or tokens. The set from which these letters are taken is called the **alphabet** over which the language is defined. A formal language is often defined by means of a **formal grammar** (also called its formation rules); accordingly, words that belong to a formal language are sometimes called **well-formed formulas**. Formal languages are studied in computer science and linguistics; the field of formal language theory studies the purely syntactical aspects of such languages (that is, their internal structural patterns).
A hierarchy is defined for any language(or formal language)


An **alphabet** is a set of symbols
{ a, b }
Sentences are strings of symbols
a, b, aa, ab, ba, a....
A **language** is a set of sentences
L = (aaa, abaa, aaba, bbb)
A **grammar** is a finite list of rules defining a language
S→ aA
A→ bA
A→ aB
B→ bB
B→ aF
F→ ε

Computational Problems

In theoretical computer science, a computational problem is a mathematical object representing a collection of questions that computers might want to solve. For example, the problem of factoring "Given a positive integer n, find a nontrivial prime factor of n." is a computational problem. Computational problems are one of the main objects of study in theoretical computer science. The field of algorithms studies methods of solving computational problems efficiently. The complementary field of computational complexity attempts to explain why certain computational problems are intractable for computers

Types of computational problems

A decision problem is a computational problem where the answer for every instance is either yes or no. An example of a decision problem is primality testing: "Given a positive integer n, determine if n is prime."
A **decision problem** is typically represented as the set of all instances for which the answer is yes. For example, primality testing can be represented as the infinite set L = {2, 3, 5, 7, 11, ...}
In a **search problem**, the answers can be arbitrary strings. For example, factoring is a search problem where the instances are (string representations of) positive integers and the solutions are (string representations of) collections of primes.
A search problem is represented as a relation over consisting of all the instance-solution pairs, called a search relation. For example, primality can be represented as the relation

$$R = \{(4, 2), (6, 2), (6, 3), (8, 2), (8, 4), (9, 3), \ldots\}$$

which consist of all pairs of numbers (n, p), where p is a nontrivial prime factor of n.

A **counting problem** asks for the number of solutions to a given search problem. For example, the counting problem associated with primality is "Given a positive integer n, count the number of nontrivial prime factors of n."

A counting problem can be represented by a function f from {0, 1}* to the nonnegative integers. For a search relation R, the counting problem associated to R is the function fR(x) = |{y: (x, y) " R}|.

An **optimization problem** asks for finding the "best possible" solution among the set of all possible solutions to a search problem. One example is the maximum independent set problem: "Given a graph G, find an independent set of G of maximum size."

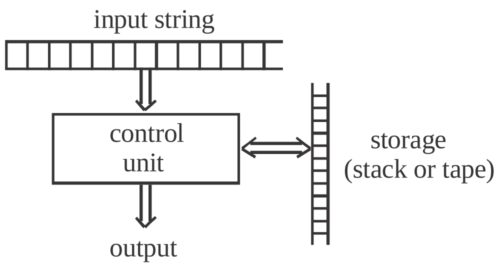
Optimization problems can be represented by their search relations.

Computational models

In computability theory and computational complexity theory, a **model of computation** is the definition of the set of allowable operations used in computation and their respective costs. It is used for measuring the complexity of an algorithm in *execution time* and or *memory space*: by assuming a certain model of computation, it is possible to analyze the computational resources required or to discuss the limitations of algorithms or computers. Some examples of models include **Turing machines**, **recursive functions**, **lambda calculus**, and **production systems**.

Automata:

An automaton is an abstract model of a computer, which accepts some input (a string), produces output (yes/no or a string) and may have internal, storage (usually, a stack or tape).



An automaton operates in a discete time frame (like a real computer). A particular state of te automaton including the state of the conrol unit, input, and storage, is called a configuration. The transaction from a configuration to the next one is a move. If the output is yes/no, the automaton is called an acceptor. If the output is string, it is called a transducer.

Basic concepts:

String: An alphabet is a non-empty finite set of symbols deonted Σ

e.g. $\Sigma = \{a, b, c\}$ is an alphabet

A string is a finite sequence of symbols, strings are usually deonted by u, v and w.

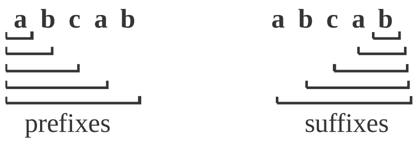
e.g. U = abcab is a string on $\Sigma = \{a, b, c\}$

The empty string (no symbols at all) is denoted λ

A part of a string is a substring.

e.g. Bca is a substring of abcab.

A begininning of a string (up to any symbol) is a prefix, and an ending is a suffix.



Note: A string is a prefix and suffix of itself. λ is a prefix and suffix of any string.

String Operations:
Concatenation: wv – appending v to the end of w.

e.g. w = abc, v = ab; $wv = \underbrace{abc}_{w}\,\underbrace{ab}_{v}$

Note: $\lambda w = w\lambda = w$

Reverse: $w^R \to w$ in reverse order

e.g. $w = abc,$ $w^R = cba$

Length of a string: number of symbols

e.g. $|abc| = 3$

Note: $|\lambda| = 0$; $|w^R| = |w|$; $|wv| = |w| + |v|$

$w^n \to w$ repeated n times, that is, $w^n = \underbrace{ww\ldots\ldots w}_{\text{n times}}$

e.g. $(abc)^2 = abcabcabc$

Note: $w^0 = \lambda$; $|w^n| = n.|w|$

Languages: Σ^* is the set of all strings on Σ.

e.g. $\Sigma = \{a, B, c\}$; $\Sigma^* = \{\lambda, a, b, c, aa, ab, ac, ba, \ldots\ldots\ldots\ldots\}$

Σ^+ is the set of all strings except λ.

Note: Σ^* and Σ^+ are infinite.

A langauage is a subset of Σ^*, usually denoted L.
It may be finite on infinite.

Example: $\{a, ba, abc\}$; $\{\lambda, a, aa, aaa, \ldots\ldots\ldots\ldots\}$, ϕ

If a string w is in L, we say that w is a sentence of L.

Operations on language:
Union, intersection and difference-same as on sets.

e.g. $\{a, ba, abc\} \cap \{\lambda, a, aa, aaa, \ldots\ldots\ldots\} = \{a\}$

Complementation: $\bar{L} = \Sigma^* - L$

e.g. $L = \{\lambda, a, aa, aaa, \ldots\ldots\}$

$\bar{L} = \{w : w \text{ includes b or c}\}$

Reverse: $L^R = \{w^R : w \in L\}$ – the set of reversed strings

e.g. $L = \{a, ba, abc\}$; $\bar{L} = \{a, ab, cba\}$

Determinastic Finite Automata (DFA):

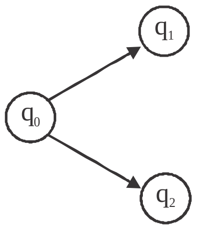
Where, Q = set of internal state

$q_0 \in Q$ = initial/start symbol of FA

Σ = I, P alphabet

δ = transiston function

$F \leq Q$ = Final state.

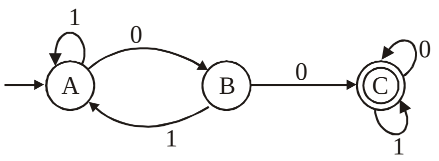


$\boxed{\delta : Q \times \Sigma \to Q} \to$ total function

Why called deterministic at a state for given I/P because we have only one choice either move to another state or itself.

Example 1.1

The following is an FA with 3 states called A, B and C. The start state is A, and C is the only accept state.



Consider its behavior when the input string is 101001:

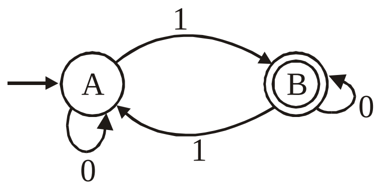
Current State	Symbol Read	New State
A	1	A
A	0	B
B	1	A
A	0	B
B	0	C
C	1	C

Here, the final state is C. Similarly, the final state for 11101 is A, and for 0001 it is C.

What does it take to get the accept state? This machine accepts all strings of 0's and 1's with two consecutive zeroes somewhere.

Example 1.2

Consider the following FA



This FA accepts strings like 100 and 0101001 and 11111, and rejects strings like 000 and 0110.

Can you describe exactly which strings this FA accepts?

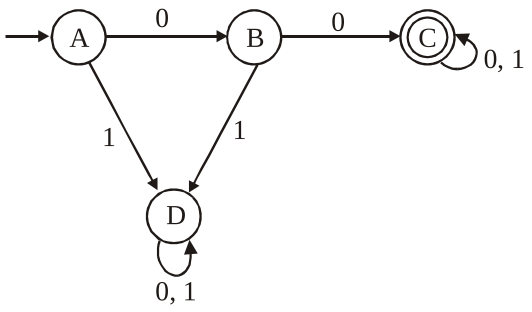
This FA ignores the symbol 0 (it doesn't change state). It only worries about the symbol 1; here it alternates states. The first 1 takes it to state B, the second 1 takes it to state A, the third to state B, and so on. So it is in state B whenever an odd number of 1's have been read.

That is, this machine accepts all strings of 0's and 1's with an odd number of 1's.

Example 1.3

All binary strings starting with 00.

The approach is to read the first two symbols, and that gives one the answer. The rest of the string is ignored (which corresponds to the automaton never changing state).

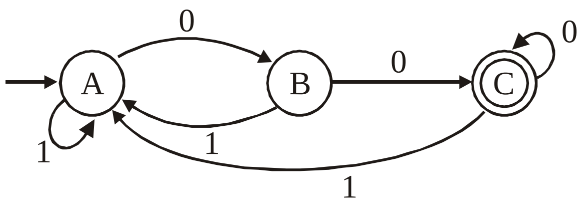


Example 1.4

All the binary strings ending in 00

Well, the first thing you might do is to think about what happens if the first two symbols are 0.That must take you to an accept state. So, there are at least three states: A, B, and C, where A is the start state, C is an accept state, and a 0 takes you from A to B, and from B to C.

But what happens if you read a 1? Well, if you think about it, this should take the FA back to the start state, no matter what state it is in. Another question is, what happens if you get a 0 in state C? Well, staying in C seems reasonable. This yields the following FA:



# A common type of state is a **trap**. This is a state that, once entered, you can never leave. For example, state C of example 1.1 is a trap. How can you recognize a trap from the picture? A trap has no arrow out. Traps can be used in two ways:

(a) As a reject state for partly read strings that will never be accepted

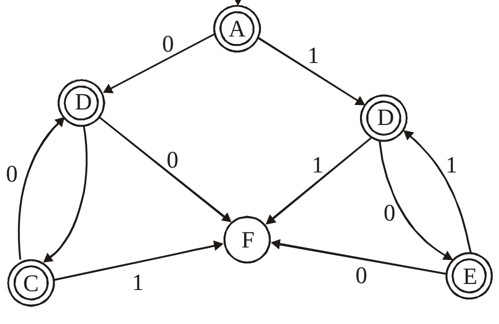
(b) As an accept state for partly read strings that will definitely be accepted.

Example 1.5

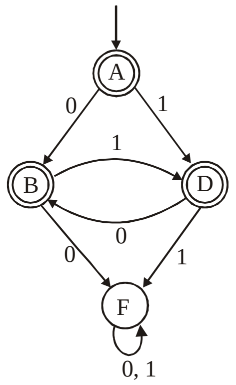
An FA that accepts all binary strings where 0's and 1's alternate

The main idea is a pair of states that oscillate: 0 takes you to one and 1 takes you back again. We also need a trap if ever two consecutive symbols are the same.

One approach is to have a start state that slipts into two sub-machines. This gives the following solution:



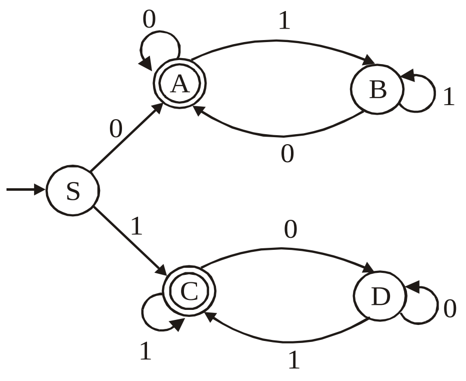
But actually, you can do it with fewer states:



# Although a given FA corresponds to only one language, a given language can have many FAs that accept it. Note that you must always be careful about the empty string: should the FA accept or not. In the preceding example, the empty string is accepted because the start state is also an accept state.
Another useful technique is remembering specific symbols. In the next example you must forever remember the first symbol; so the FA splits into two pieces after the first symbol.

Example 1.6

An FA that accepts all binary strings that begins and end with the same symbol.
It is clear that the automaton must be in two different states based on the first symbol that is read. After that, you need to keep track of the current symbol just read, in case it turns out to be the final symbol. But that is enough.



Note that there must not be any connection between the top and bottom halves, since otherwise the FA might forget the first symbol read.

Formal defintion of FA:

FA is of two types: Deterministic and Nondeterministic. The FA discussed so far are deterministic one. The formal definition of a DFA(Deterministic Finite Automata) is given as a 5-tuple (Q,Σ, q_0, T,δ) where:

- Q is the finite set of states
- Σ is an alphabet of input symbols
- q_0 is the start state
- T is a subset of Q giving the accept states
- δ is the transition function that maps a symbol to a state (δ: Qx Σ '!Q)

Transition table:

It is a matrix that lists the new state given the current state and the symbol read.

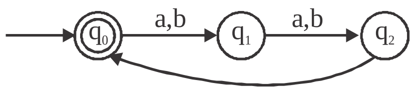
Example 1.7

Here is the transition table for the FA that accepts all binary strings that begin and end with the same symbol.

	Input	
State	0	1
S	A	C
A	A	B
B	A	B
C	D	C
D	D	C

Example 1.8 **GATE-2002**

The smallest DFA accepts the language L = $x \in \{a,b\}^*$, length of n is multiple of 3.

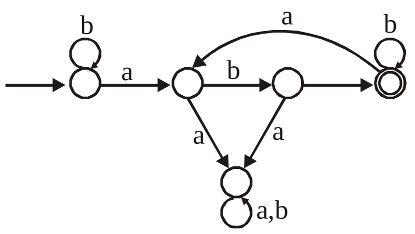


(a) 2 (b) 3 (c) 4 (d) 5

Ans. **(b)**

Example 1.9 **GATE-2005**

Consider the machine M



The language recognized by above machine is:

(a) $\left\{ w \in \{a,b\}^* /\ \text{every a in w, is followed by exactly two b's} \right\}$

(b) $\left\{ w \in \{a,b\}^*\ \text{every a in w, is followed by at least two b's} \right\}$

(c) $\left\{ w \in \{a,b\}^*\ \text{w contains the substring 'abb'} \right\}$

(d) $\left\{ w \in \{a,b\}^*\ \text{w does not contain 'aa' as a substring} \right\}$

Ans. **(b)**

Example 1.10

If the final state and non-final state in DFA below are interchanged then which of
(a) Set of all string that does not end with a and b.
(b) Set of all string that begin with a and b.
(c) Set of all string that does not count.
(d) The set describe by b* a a*(ba)

Ans. **(a)**

REGULAR EXPRESSIONS CHAPTER-2

Regular Expression:: A regular expression (RE) corresponds to a set of strings, that is, a regular expression describes a language. It is built up using the three regular operations called union concatenation, and star, described in the following paragraphs. A regular expression uses normal symbols as well as four special symbols. A regular expressions is interpreted using the following rules:

- The parentheses (and) are used for grouping, just as in normal math.
- The plus sign (+) means union. Thus, writing 0 + 1
  Means either a zero or a one. We also refer to the + as or.
- The concentration of two expressions is obtained by simply writing one after the other without spacing between them. For example, (0 + 1)0
  corresponds to the pair of strings 00 and 10.
- The asterisk (*) is pronounced star and means zero or more copies.
  For example, the regular expression..... a*
  corresponds to any string of a's: $\{\varepsilon, a, aa, aaa, ...\}$. Also , (0 + 1)*
  corresponds to all binary strings. (Its $\varepsilon + (0+1) + (0+1)(0+1+.......)$

Primitive regular expression: If R_1 and R_2 are regular expression then
(1) $R_1 + R_2$ is also a regular expression.
(2) R_1R_2 is also a regular expression
(3) R_1* is also a regular expression
(4) (R_1) is also a regular expression.
(5) Any expression that is obtained by using the above as a rule and any number of times is also a Regular expression.
e.g. Now, $\Sigma = \{a, b\}$
(a) a is regular expression b*, a* is a regular expression
(b) b is regular expression ab, is a regular expression
(c) a + b is regular expression. ba is a regular expression

Problem: Write the regular expression for
(1) All binary string which end with 110
(2) All binary string whose start and end is same value.
(3) All binary string whose start and end is different value.
(4) All binary string containing even number of zero's
(5) All binary string containing 1101 as substring.

Soln. (1) $(0 + 1)^* 110$
(2) $111 (0+1)^*$
(3) $1(0+1)^* 1 + 0(0+1)^* 0 + 1$
(4) $1(0+1)^* 0 + 0(0+1)^* 1$
(5) $(1^*0 + 1^*0)$

Problem: Write Regular Expression for

1. (a) $\{a^nb^m \mid n \geq 0,\ m \geq 0\}$ (b) $\{a^nb^m \mid n \geq 0,\ m \geq 1\}$ (c) $\{a^nb^m \mid n \geq 2,\ m \geq 3\}$

Soln. (a) $\{a^nb^m \mid n \geq 0,\ m \geq 0\} = (a^*b^*)$ (b) $\{a^nb^m \mid n \geq 0,\ m \geq 1\} = (a^*b^+)$ (c) $\{a^nb^m \mid n \geq 2,\ m \geq 3\} = (aaa^*\ \ bbb^*)$