VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Theory of Computation

Module 1

Introduction to Languages and Grammars

Module:1	Introduction to Languages and Grammars	4 hours
Recall on Proof techniques in Mathematics - Overview of a Computational Models - Languages and Grammars - Alphabets - Strings - Operations on Languages, Overview on Automata		

Applications of the Concepts AND New Developments

- Applications include:
  - Text processing (editors, etc.)
  - Compiler design
  - Verification through model checking
- Development's include :

  DNA Computing

  Quantum Turing Machines

- What is TOC?

  In theoretical computer science, the theory of computation is the branch that deals with whether and how efficiently problems can be solved on a model of computation, using an algorithm.

- **Introduction to formal proof:**

- **Basic Symbols used :**
- ∪ – Union
- ∩- Conjunction
- ∈ - Empty String
- Φ – NULL set
- 7- negation
- ‘ – compliment
- = > implies

Basic Laws:

- **Additive inverse:** a+(-a)=0
- **Multiplicative inverse:** a*1/a = 1
- **Universal set** U={1,2,3,4,5}
- **Subset** A={1,3} A' ={2,4,5}
- **Absorption law:** AU(A ∩B) = A, A∩(AUB) = A
- **De Morgan's Law:** (AUB)' =A' ∩ B', (A∩B)' = A' U B'
- **Double compliment:** (A')' =A, A ∩ A' = Φ
- **Logic relations:** a -> b = > 7a U b, 7(a∩b)=7a U 7b

- The sets of even and odd numbers can be expressed as follows,
- Even = {2k : k ∈ Z}
- Odd = {2k + 1 : k ∈ Z}
- Rational = a/b

Relations: Let a and b be two sets a relation R contains aXb.

Relations used in TOC:

Reflexive: a = a

Symmetric: aRb = > bRa

Transition: aRb, bRc = > aRc

If a given relation is reflexive, symmentric and transitive then the relation is called equivalence relation.

- Reasoning is the process of inferring something from a given statement.

2 Reasoning approaches:

- Inductive Proof – Develop a Theory
- Deductive Proof – Test the theory

INDUCTIVE PROOF

It is a special form of proof that deals about the objects in recursion.

Induction Principle:

If we prove S(i) and we prove that for all n>=i, S(n)=>S(n+1), then we may conclude that s(n) for all n>=1.

It has two parts.

1.Basis part.

2.Inductive part.

- **Basis step:** We have to **show S(i) for a particular integer value "i",** here "i " may be zero or one.

- **Inductive step:** We assume n>=i, where "i " is the basis integer and we have **to show that if S(n) then S(n+1)** i.e., S(n) =>S(n+1).

Problems

- Prove by an induction method on “n”

  i=0/1➔ $\sum n^2 = ((n(n+1)(2n+1))/6)$.

Solution:

  i=0/1 ➔$\sum n^2 = ((n(n+1)(2n+1))/6)$. ----> Equ 1

- Induction Principle:

If we prove S(i) and we prove that for all n>=i, S(n)=>S(n+1), then we may conclude that s(n) for all n>=1.

Basis:

Put n=0 in the equ 1

RHS= (0 (0+1) (2*0+1))/6 => 0

LHS = i=0 ∑ n (02) => 0

i.e., LHS = RHS.

Inductive step:

Put n =n in the equ 1

S(n) = (n(n+1)(2n+1))/6.

= $(2n^3+2n^2+n^2+n)/6$. ---> Equ 2

- Put n=n+1 in the equ 1
- S(n+1) => ((n+1)(n+2)(2n+3))/6.

  => $((n^2+3n+2)\ (2n+3))/6$.

  => $(2n^3+6n^2+4n+3n^2+9n+6)/6$.->equ 3

From the induction principle:

= S(n+1) =S(n) + $(n+1)^2$.

= $((2n^3+3n^2+n)/6) + (n+1)^2$

- $= ((2n^3+3n^2+n)/6) + (n^2+2n+1)$
- Cross multiplying the '6' on the numerator.
- $= ((2n^3+3n^2+n + 6n^2+12n+6))/6.$
- $= ((2n^3+9n^2+13n+6))/6.$ --→ equ 4
- Equ 4 satisfies equ 3
- Hence proved.

Deductive proof:

- Consists of sequence of statements whose truth lead us from some initial statement called the hypothesis or the give statement to a conclusion statement.

- Deductive Principle:
- The theorem is proved when we go from a hypothesis H to a conclusion C is the statement " if H then C". i.e C is deduced from H

Additional forms of proof:

- Proof of sets
- Proof by contradiction
- Proof by counter example

Proof of Sets

If E and F are two expressions representing sets, the statement E=F means that the two sets represented are the same

- Theorem R U (S ∩ T) =(R U S) ∩ (R U T)

	Statement	Justification
1.	x is in $R \cup (S \cap T)$	Given
2.	x is in R or x is in $S \cap T$	(1) and definition of union
3.	x is in R or x is in both S and T	(2) and definition of intersection
4.	x is in $R \cup S$	(3) and definition of union
5.	x is in $R \cup T$	(3) and definition of union
6.	x is in $(R \cup S) \cap (R \cup T)$	(4), (5), and definition of intersection

	Statement	Justification
1.	x is in $(R \cup S) \cap (R \cup T)$	Given
2.	x is in $R \cup S$	(1) and definition of intersection
3.	x is in $R \cup T$	(1) and definition of intersection
4.	x is in R or x is in both S and T	(2), (3), and reasoning about unions
5.	x is in R or x is in $S \cap T$	(4) and definition of intersection
6.	x is in $R \cup (S \cap T)$	(5) and definition of union

Proof by contradiction

- Let S be a finite subset of some infinite set U. Let T be the complement of S with respect to U. Then T is infinite.
- **Soln:**
- Assume T was finite.
- Since, S and T are finite, then U must also be finite. But, it is stated in hypothesis that U is infinite. Therefore, it's a contradiction. So T is finite.

Proof by counter Examples

- Finding disproofs for an obvious non theorem or a statement that requires more investigation.
- Eg: All primes are odd.
- Soln:

  Disproof: The integer 2 is a prime, but it is even.

Therefore, the given statement is not a theorem and it requires more investigation.

- **Direct proof (or) Constructive proof**:

If p is true then q is true

- Eg: if a and b are odd numbers then product is also an odd number. Odd number can be represented as 2n+1
- a=2x+1, b=2y+1
- Product of

$$a \times b = (2x+1) \times (2y+1)$$

$$= 2(2xy+x+y)+1$$

$$= 2z+1 \text{ (odd number)}$$

Proof by Contrapositve

- The Contrapositive of the statement " if H then C" is "if not C then not H".A statement and its contrapositive are either both true or both false.
- **Goal:**

  P(x) = Q(x)

  ^Q(x) = ^P(x)

Qn: If n^2 is even, then n is even.

Theorem:

For all integers n, if n^2 is even then n is even.

- We prove the contrapositive.
- Suppose that n is not even, that is, n is odd. Then $n = 2k + 1$ for some integer k.
- So $n^2 = (2k+1)^2 = 4k^2 + 4k + 1$

$= 2(2k^2 + 2k) + 1$

which is odd.

Thus we have proved: if n is not even, then n^2 is not even.

So by the contrapositive, we can conclude that if n^2 is even, then n is even.

Proof by Contradiction

- H and not C implies falsehood.

That is, start by assuming both the hypothesis H and the negation of the conclusion C. Complete the proof by showing that something known to be false follows logically from H and not C. This form of proof is called *proof by contradiction.*

It often is easier to prove that a statement is not a theorem than to prove it *is* a theorem. As we mentioned, if S is any statement, then the statement "S is not a theorem" is itself a statement without parameters, and thus can

Be regarded as an observation than a theorem.

- **Qn:** $\sqrt{2}$ is irrational

  Proof. Suppose $\sqrt{2}$ is rational.

Then integers a and b exist so that $\sqrt{2} = a/b$.

Without loss of generality we can assume that a and b have no factors in common (i.e., the fraction is in simplest form).

Multiplying both sides by b and squaring, we have

$2b^2 = a^2$ so we see that a^2 is even.

This means that a is even (how would you prove this?) so $a = 2m$ for some $m \in Z$.

Then $2b^2 = a^2 = (2m)^2 = 4m^2$

which, after dividing by 2, gives $b^2 = 2m^2$ so b^2 is even. This means $b = 2n$ for some $n \in Z$.

We've seen that if $\sqrt{2} = a/b$ then both a and b must be even and so are both multiples of 2. This contradicts the fact that we know a and b can be chosen to have no common factors. Thus, $\sqrt{2}$ must not be rational, so $\sqrt{2}$ is irrational.

Overview of computational models

There are six basic computational models such as Turing, von Neumann, dataflow, applicative, object-based, predicate logic-based, etc. These models are known as basic models because they can be declared using a basic set of abstractions.

Computational models, languages, and architecture classes

Computational Model	Language Class	Architecture Class
Turing	'Type 0' languages	
von Neumann	Imperative	Von Neumann
Dataflow	Single Assignment (dataflow)	Dataflow
Applicative	Functional	Reduction
Object-based	Object-Oriented	Object-Oriented
Predicate logic based	Logic Programming	So far unnamed

Key Features of Computational models:

- **Basic items of Computation** – The first abstraction recognizes the basic items of computation. This is a requirement of the items the computation defines any type of computations that can be implemented on them.

- **Problem description model** – It refers to both the style and method of the problem description. The **problem description style** specifies how problems in a specific computational model are represented. The style is either procedural or declarative.

- **Execution Model** – The last element of the computational model is the execution model. The first component declares the interpretation of the computation, which is powerfully associated with the problem description method. The possibility of the problem description method and the interpretation of the computation commonly regulate and infer each other.

Languages and Grammars

Introduction of Formal Proof - Basic Definitions

1. **Alphabet** - a finite set of symbols.
   - Notation: Σ .

   Examples: Binary alphabet {0,1},
   English alphabet {a,...,z,!,?,...}
2. **String over an alphabet** Σ - a finite sequence of symbols from Σ.
   - Notation: (a) Letters u, v, w, x, y, and z denote strings.
     (b) Convention: concatenate the symbols. No parentheses or commas used.
   - Examples: 0000 is a string over the binary alphabet.
     a!? is a string over the English alphabet.

3. **Empty string**: e or ε denotes the empty sequence of symbols.
4. **Language over alphabet** Σ - a set of strings over Σ.
   - Notation: L.
   - Examples:
     - {0, 00, 000, ...} is an "infinite" language over the binary alphabet.
     - {a, b, c} is a "finite" language over the English alphabet.
5. **Empty language** - empty set of strings. Notation: Φ.
6. **Binary operation on strings**: Concatenation of two strings u.v - concatenate the symbols of u and v.
   - Notation: uv
   - Examples:
     - 00.11 = 0011.
     - ε.u = u.ε = u for every u. (identity for concatenation)

Binary relations on strings

1. **Prefix** - u is a prefix of v if there is a w such that v = uw.
   - Examples:
     - ε is a prefix of 0 since 0 = ε0
     - apple is a prefix of appleton since appleton = apple.ton
2. **Suffix** - u is a suffix of v if there is a w such that v = wu.
   - Examples:
     - 0 is a suffix of 0 since 0 = ?
     - ton is a suffix of appleton since ?

3. **Substring** - u is a substring of v if there are x and y such that v = xuy.
   - Examples:
     - let is a substring of appleton since appleton = app.let.on
     - 0 is a substring of 0 since 0 = epsilon.0.epsilon

Observe that prefix and suffix are special cases of substring.

Regular Expressions

Operations on Language

OPERATION	DEFINITION
union of L and M written $L \cup M$	$L \cup M = \{ s \mid s \text{ is in } L \text{ or } s \text{ is in } M \}$
concatenation of L and M written LM	$LM = \{ st \mid s \text{ is in } L \text{ and } t \text{ is in } M \}$
Kleene closure of L written L^*	$L^* = \bigcup_{i=0}^{\infty} L^i$ L^* denotes "zero or more concatenations of" L.
positive closure of L written L^+	$L^+ = \bigcup_{i=1}^{\infty} L^i$ L^+ denotes "one or more concatenations of" L.

Grammar

It is a set of Rule to define a language.
It consist of 4 tuples

$G_1 = (V, \Sigma, P, S)$

V - (Vocabulary) Variable.
T - Set of terminal
S - Start Symbol
P - Set of production

The set $(V-T) = N$ (non terminal)

* Every P must contain atleast one nonterminal on its left side.

Example

Let $G_1 = \{V, T, P, S\}$ where

$V = \{a, b, A, B, S\}$

$T = \{a, b\}$

$S = \{S\}$

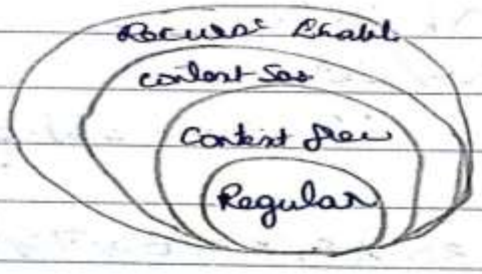
$P = \{S \to ABa,\ A \to BB,\ B \to ab,\ AB \to b\}$

Then check abababa is derived from S

$S \to ABa \Rightarrow BBBa \Rightarrow abBBa \Rightarrow ababBa \Rightarrow ababa$

Types of Grammar

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	unrestricted gram	Recursive enumer.	Turing Mach
Type 1	context Sensitive g.	C S lan	Linea bounded au
Type 2	C f g	C f lang	PDA
Type 3	Reg grammar.	R.E lang	FSA



Type 3

→ It have single NT on L.H.S

→ a single T or Single T followed by NonTerm

$X \rightarrow \varepsilon$

$X \rightarrow b$

$X \rightarrow a / bY$

Type 2
→ General context free language

Production $A \rightarrow \gamma$.

$A \Rightarrow NT$

$\gamma \in (T \cup NT)^*$ (String of terminal / NT)

$S \rightarrow Xa$
$S \rightarrow aX$
$S \rightarrow \varepsilon$
$S \rightarrow abc$ . . .

Type 1
→ General context sensitive language

$\alpha A \beta \rightarrow \alpha \gamma \beta$

where $A = NT$

$\alpha, \beta, \alpha, \beta \in (T \cup NT)^*$

∴ $AB \rightarrow AbBC$

$B \rightarrow a$

$Ba \rightarrow A$

Type 0

→ General recursively enumerable language.

→ Production can be of

$$\alpha \longrightarrow \beta$$

where α = NT, T but not null

β = T, NT

Overview on Automata

Introduction of Automaton

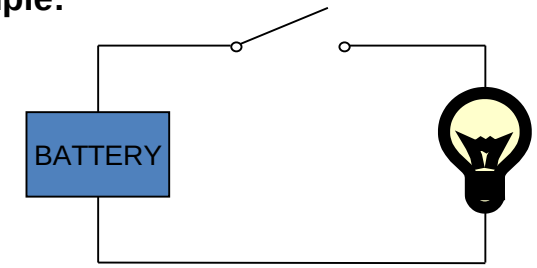
An automata is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

Automata-based programming is a programming paradigm in which the program or its part, is thought as a model of a finite state machine or any other formal automation.

What is Automata Theory?

- Automata theory is the study of abstract computational devices
- Abstract devices are (simplified) models of real computations
- Computations happen everywhere: On your laptop, on your cell phone, in nature, …

Example:



input: switch

output: light bulb

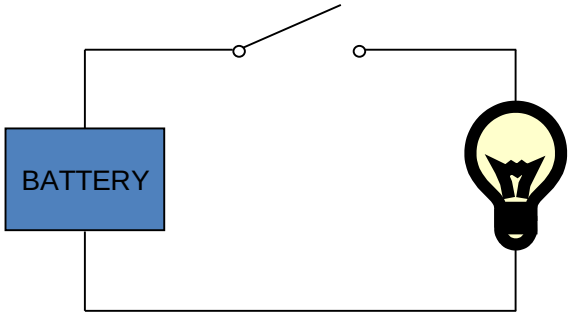
actions: flip switch

states: on, off

1. Electric Switch
2. Fan Regulator
3. Automatic door controller
4. Binary string with divisible by 4

Simple Computer

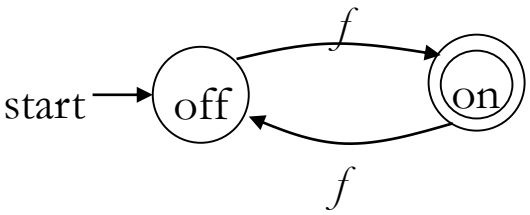
Example:



input : switch
Output : light bulb
Actions : flip switch
States : on, off

OTHER EXAMPLES

1. Electric Switch
2. Fan Regulator
3. Automatic door controller
4. Binary string with divisible by 4



bulb is on if and only if there was an odd number of flips

Types of Automata

finite automata	Devices with a finite amount of memory. Used to model “small” computers.
push-down automata	Devices with infinite memory that can be accessed in a restricted way. Used to model parsers, etc.
Turing Machines	Devices with infinite memory. Used to model any computer.